Category Theory for ZFC in HOL II
Elementary Theory of 1-Categories

Mihails Milehins

March 19, 2025

Abstract

This article provides a formalization of the foundations of the theory of 1-categories in the object logic *ZFC in HOL* ([11], also see [9]) of the formal proof assistant *Isabelle* [10]. The methodology chosen for the formalization rests on the ideas that were originally expressed in [5]. Thus, in the context of this work, each category is represented as a term of the type V embedded into a stage of the von Neumann hierarchy [14].

Acknowledgements

The author would like to acknowledge the assistance that he received from the users of the mailing list of Isabelle in the form of answers given to his general queries. Special thanks go to Andreas Lochbihler for suggesting the use of the combination of unrestricted overloading and locales for structuring mathematical knowledge on the mailing list of Isabelle: the design pattern that is used in this study builds upon this idea. Special thanks also go to Thomas Sewell for suggesting a trick for rewriting expressions modulo associativity on the mailing list of Isabelle, which was used on numerous occasions throughout the development of this work. Furthermore, the author would like to mention that the tool "Sketch-and-Explore" [6] from the standard distribution of Isabelle was used extensively in the development of this work. Moreover, the author would like to acknowledge the positive role that numerous Q&A posted on the Stack Exchange network (especially Mathematics Stack Exchange, Stack Overflow and TeX Stack Exchange) played in the development of this work. Lastly, the author would like to express gratitude to all members of his family and friends for their continuous support.

Contents

1 Introduction

1.1 Background

This article provides a formalization of the elementary theory of 1-categories without an additional structure. For further information see chapter Introduction in [8].

1.2 Preliminaries

named-theorems *cat-op-simps*
named-theorems *cat-op-intros*

named-theorems *cat-cs-simps*
named-theorems *cat-cs-intros*

named-theorems *cat-arrow-cs-intros*

1.3 CS setup for foundations

lemmas (in $\mathcal{Z}$**)** [*cat-cs-intros*] = $\mathcal{Z}$-β

2 Category

2.1 Background

lemmas [*cat-cs-simps*] = *dg-shared-cs-simps*
lemmas [*cat-cs-intros*] = *dg-shared-cs-intros*

definition *CId* :: *V*
  where [*dg-field-simps*]: *CId* = $5_{\mathbb{N}}$

2.1.1 Slicing

definition *cat-smc* :: *V* ⇒ *V*
  where *cat-smc* $\mathfrak{C}$ = [$\mathfrak{C}(\!|Obj|\!)$, $\mathfrak{C}(\!|Arr|\!)$, $\mathfrak{C}(\!|Dom|\!)$, $\mathfrak{C}(\!|Cod|\!)$, $\mathfrak{C}(\!|Comp|\!)$]$_\circ$

Components.

lemma *cat-smc-components*[*slicing-simps*]:
  shows *cat-smc* $\mathfrak{C}(\!|Obj|\!)$ = $\mathfrak{C}(\!|Obj|\!)$
    and *cat-smc* $\mathfrak{C}(\!|Arr|\!)$ = $\mathfrak{C}(\!|Arr|\!)$
    and *cat-smc* $\mathfrak{C}(\!|Dom|\!)$ = $\mathfrak{C}(\!|Dom|\!)$
    and *cat-smc* $\mathfrak{C}(\!|Cod|\!)$ = $\mathfrak{C}(\!|Cod|\!)$
    and *cat-smc* $\mathfrak{C}(\!|Comp|\!)$ = $\mathfrak{C}(\!|Comp|\!)$
  unfolding *cat-smc-def dg-field-simps* **by** (*auto simp*: *nat-omega-simps*)

Regular definitions.

lemma *cat-smc-is-arr*[*slicing-simps*]:
  $f : a \mapsto_{cat\text{-}smc\ \mathfrak{C}}\ b \longleftrightarrow f : a \mapsto_{\mathfrak{C}}\ b$
  unfolding *is-arr-def slicing-simps* **..**

lemmas [*slicing-intros*] = *cat-smc-is-arr*[*THEN iffD2*]

lemma *cat-smc-composable-arrs*[*slicing-simps*]:
  composable-arrs (*cat-smc* $\mathfrak{C}$) = *composable-arrs* $\mathfrak{C}$
  unfolding *composable-arrs-def slicing-simps* **..**

lemma *cat-smc-is-monic-arr*[*slicing-simps*]:
  $f : a \mapsto_{mon\,cat\text{-}smc\ \mathfrak{C}}\ b \longleftrightarrow f : a \mapsto_{mon\,\mathfrak{C}}\ b$
  unfolding *is-monic-arr-def slicing-simps* **..**

lemmas [*slicing-intros*] = *cat-smc-is-monic-arr*[*THEN iffD2*]

lemma *cat-smc-is-epic-arr*[*slicing-simps*]:
  $f : a \mapsto_{epi\,cat\text{-}smc\ \mathfrak{C}}\ b \longleftrightarrow f : a \mapsto_{epi\,\mathfrak{C}}\ b$
  unfolding *is-epic-arr-def slicing-simps op-smc-def*
  by (*simp add*: *nat-omega-simps*)

lemmas [*slicing-intros*] = *cat-smc-is-epic-arr*[*THEN iffD2*]

lemma *cat-smc-is-idem-arr*[*slicing-simps*]:
  $f : \mapsto_{ide\,cat\text{-}smc\ \mathfrak{C}}\ b \longleftrightarrow f : \mapsto_{ide\,\mathfrak{C}}\ b$
  unfolding *is-idem-arr-def slicing-simps* **..**

lemmas [*slicing-intros*] = *cat-smc-is-idem-arr*[*THEN iffD2*]

lemma *cat-smc-obj-terminal*[*slicing-simps*]:
  obj-terminal (*cat-smc* $\mathfrak{C}$) *a* ⟷ *obj-terminal* $\mathfrak{C}$ *a*
  unfolding *obj-terminal-def slicing-simps* **..**

lemmas [*slicing-intros*] = *cat-smc-obj-terminal*[*THEN iffD2*]

lemma *cat-smc-obj-intial*[*slicing-simps*]:
obj-initial (*cat-smc* $\mathfrak{C}$) $a \longleftrightarrow$ *obj-initial* $\mathfrak{C}$ a
unfolding *obj-initial-def obj-terminal-def*
unfolding *smc-op-simps slicing-simps*
..

lemmas [*slicing-intros*] = *cat-smc-obj-intial*[*THEN iffD2*]

lemma *cat-smc-obj-null*[*slicing-simps*]:
obj-null (*cat-smc* $\mathfrak{C}$) $a \longleftrightarrow$ *obj-null* $\mathfrak{C}$ a
unfolding *obj-null-def slicing-simps smc-op-simps* **..**

lemmas [*slicing-intros*] = *cat-smc-obj-null*[*THEN iffD2*]

lemma *cat-smc-is-zero-arr*[*slicing-simps*]:
$f : a \mapsto_{0 \, cat\text{-}smc \, \mathfrak{C}} b \longleftrightarrow f : a \mapsto_{0 \, \mathfrak{C}} b$
unfolding *is-zero-arr-def slicing-simps* **..**

lemmas [*slicing-intros*] = *cat-smc-is-zero-arr*[*THEN iffD2*]

2.2 Definition and elementary properties

The definition of a category that is used in this work is is similar to the definition that can be found in Chapter I-2 in [7]. The amendments to the definitions that are associated with size have already been explained in [8].

locale *category* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ + *CId*: *vsv* ‹$\mathfrak{C}(\!|CId|\!)$› **for** α $\mathfrak{C}$ +
assumes *cat-length*[*cat-cs-simps*]: *vcard* $\mathfrak{C} = 6_{\mathbb{N}}$
and *cat-semicategory*[*slicing-intros*]: *semicategory* α (*cat-smc* $\mathfrak{C}$)
and *cat-CId-vdomain*[*cat-cs-simps*]: $\mathcal{D}_{\circ}$ ($\mathfrak{C}(\!|CId|\!)$) = $\mathfrak{C}(\!|Obj|\!)$
and *cat-CId-is-arr*[*cat-cs-intros*]: $a \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{C}} a$
and *cat-CId-left-left*[*cat-cs-simps*]:
$f : a \mapsto_{\mathfrak{C}} b \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|b|\!) \circ_{A \mathfrak{C}} f = f$
and *cat-CId-right-left*[*cat-cs-simps*]:
$f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A \mathfrak{C}} \mathfrak{C}(\!|CId|\!)(\!|b|\!) = f$

lemmas [*cat-cs-simps*] =
category.cat-length
category.cat-CId-vdomain
category.cat-CId-left-left
category.cat-CId-right-left

lemma (**in** *category*) *cat-CId-is-arr'*[*cat-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $b = a$ **and** $c = a$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) : b \mapsto_{\mathfrak{C}'} c$
using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cat-CId-is-arr*)

lemmas [*cat-cs-intros*] = *category.cat-CId-is-arr'*

lemma (**in** *category*) *cat-CId-is-arr''*[*cat-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
shows $f : a \mapsto_{\mathfrak{C}} a$
using *assms*(*1*)
unfolding *assms*(*2*)
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

lemmas [*cat-cs-intros*] = *category.cat-CId-is-arr''*

lemmas [*slicing-intros*] = *category.cat-semicategory*

lemma (**in** *category*) *cat-CId-vrange*: $\mathcal{R}_{\circ}\ (\mathfrak{C}(\!|CId|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!)$
proof
  fix f **assume** $f \in_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{C}(\!|CId|\!))$
  with *cat-CId-vdomain* **obtain** a **where** $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
    by (*auto elim*!: *CId.vrange-atE*)
  with *cat-CId-is-arr* **show** $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)$ **by** *auto*
qed

Rules.

lemma (**in** *category*) *category-axioms'*[*cat-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *category* $\alpha'\ \mathfrak{C}$
  unfolding *assms* **by** (*rule category-axioms*)

mk-ide rf *category-def*[*unfolded category-axioms-def*]
  |*intro categoryI*|
  |*dest categoryD*[*dest*]|
  |*elim categoryE*[*elim*]|

lemma *categoryI'*:
  assumes $\mathcal{Z}\ \alpha$
    and *vfsequence* $\mathfrak{C}$
    and *vcard* $\mathfrak{C} = 6_{\mathbb{N}}$
    and *vsv* $(\mathfrak{C}(\!|Dom|\!))$
    and *vsv* $(\mathfrak{C}(\!|Cod|\!))$
    and *vsv* $(\mathfrak{C}(\!|Comp|\!))$
    and *vsv* $(\mathfrak{C}(\!|CId|\!))$
    and $\mathcal{D}_{\circ}\ (\mathfrak{C}(\!|Dom|\!)) = \mathfrak{C}(\!|Arr|\!)$
    and $\mathcal{R}_{\circ}\ (\mathfrak{C}(\!|Dom|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
    and $\mathcal{D}_{\circ}\ (\mathfrak{C}(\!|Cod|\!)) = \mathfrak{C}(\!|Arr|\!)$
    and $\mathcal{R}_{\circ}\ (\mathfrak{C}(\!|Cod|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
    and $\bigwedge gf.\ gf \in_{\circ} \mathcal{D}_{\circ}\ (\mathfrak{C}(\!|Comp|\!)) \longleftrightarrow$
      $(\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b)$
    and $\mathcal{D}_{\circ}\ (\mathfrak{C}(\!|CId|\!)) = \mathfrak{C}(\!|Obj|\!)$
    and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
    and $\bigwedge c\ d\ h\ b\ g\ a\ f.\ [\![\ h : c \mapsto_{\mathfrak{C}} d;\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
      $(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$
    and $\bigwedge a.\ a \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{C}} a$
    and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{C}} b \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|b|\!) \circ_{A\mathfrak{C}} f = f$
    and $\bigwedge b\ c\ f.\ f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A\mathfrak{C}} \mathfrak{C}(\!|CId|\!)(\!|b|\!) = f$
    and $\mathfrak{C}(\!|Obj|\!) \subseteq_{\circ} Vset\ \alpha$
    and $\bigwedge A\ B.\ [\![\ A \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!);\ B \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!);\ A \in_{\circ} Vset\ \alpha;\ B \in_{\circ} Vset\ \alpha\ ]\!] \Longrightarrow$
      $(\bigcup_{\circ} a \in_{\circ} A.\ \bigcup_{\circ} b \in_{\circ} B.\ Hom\ \mathfrak{C}\ a\ b) \in_{\circ} Vset\ \alpha$
  shows *category* $\alpha\ \mathfrak{C}$
  by (*intro categoryI semicategoryI'*, *unfold cat-smc-components slicing-simps*)
    (*simp-all add*: *assms smc-dg-def nat-omega-simps cat-smc-def*)

lemma *categoryD'*:
  assumes *category* $\alpha\ \mathfrak{C}$
  shows $\mathcal{Z}\ \alpha$
    and *vfsequence* $\mathfrak{C}$
    and *vcard* $\mathfrak{C} = 6_{\mathbb{N}}$
    and *vsv* $(\mathfrak{C}(\!|Dom|\!))$
    and *vsv* $(\mathfrak{C}(\!|Cod|\!))$

and *vsv* (ℭ(|*Comp*|))
and *vsv* (ℭ(|*CId*|))
and $\mathcal{D}_\circ$ (ℭ(|*Dom*|)) = ℭ(|*Arr*|)
and $\mathcal{R}_\circ$ (ℭ(|*Dom*|)) $\subseteq_\circ$ ℭ(|*Obj*|)
and $\mathcal{D}_\circ$ (ℭ(|*Cod*|)) = ℭ(|*Arr*|)
and $\mathcal{R}_\circ$ (ℭ(|*Cod*|)) $\subseteq_\circ$ ℭ(|*Obj*|)
and $\bigwedge gf.\ gf \in_\circ \mathcal{D}_\circ$ (ℭ(|*Comp*|)) $\longleftrightarrow$
($\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_\circ \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b$)
and $\mathcal{D}_\circ$ (ℭ(|*CId*|)) = ℭ(|*Obj*|)
and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{C}} c; f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
and $\bigwedge c\ d\ h\ b\ g\ a\ f.\ [\![\ h : c \mapsto_{\mathfrak{C}} d; g : b \mapsto_{\mathfrak{C}} c; f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
$(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$
and $\bigwedge a.\ a \in_\circ$ ℭ(|*Obj*|) $\Longrightarrow$ ℭ(|*CId*|)(|*a*|) : $a \mapsto_{\mathfrak{C}} a$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{C}} b \Longrightarrow$ ℭ(|*CId*|)(|*b*|) $\circ_{A\mathfrak{C}} f = f$
and $\bigwedge b\ c\ f.\ f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A\mathfrak{C}}$ ℭ(|*CId*|)(|*b*|) $= f$
and ℭ(|*Obj*|) $\subseteq_\circ$ *Vset* α
and $\bigwedge A\ B.\ [\![\ A \subseteq_\circ$ ℭ(|*Obj*|); $B \subseteq_\circ$ ℭ(|*Obj*|); $A \in_\circ$ *Vset* α; $B \in_\circ$ *Vset* $\alpha\ ]\!] \Longrightarrow$
$(\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.$ *Hom* ℭ $a\ b) \in_\circ$ *Vset* α
by
(
simp-all add:
categoryD(2–9)[*OF assms*]
semicategoryD′[*OF categoryD*(5)[*OF assms*], *unfolded slicing-simps*]
)

lemma *categoryE′*:
assumes *category* α ℭ
obtains $\mathcal{Z}$ α
and *vfsequence* ℭ
and *vcard* ℭ = $6_{\mathbb{N}}$
and *vsv* (ℭ(|*Dom*|))
and *vsv* (ℭ(|*Cod*|))
and *vsv* (ℭ(|*Comp*|))
and *vsv* (ℭ(|*CId*|))
and $\mathcal{D}_\circ$ (ℭ(|*Dom*|)) = ℭ(|*Arr*|)
and $\mathcal{R}_\circ$ (ℭ(|*Dom*|)) $\subseteq_\circ$ ℭ(|*Obj*|)
and $\mathcal{D}_\circ$ (ℭ(|*Cod*|)) = ℭ(|*Arr*|)
and $\mathcal{R}_\circ$ (ℭ(|*Cod*|)) $\subseteq_\circ$ ℭ(|*Obj*|)
and $\bigwedge gf.\ gf \in_\circ \mathcal{D}_\circ$ (ℭ(|*Comp*|)) $\longleftrightarrow$
($\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_\circ \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b$)
and $\mathcal{D}_\circ$ (ℭ(|*CId*|)) = ℭ(|*Obj*|)
and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{C}} c; f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
and $\bigwedge c\ d\ h\ b\ g\ a\ f.\ [\![\ h : c \mapsto_{\mathfrak{C}} d; g : b \mapsto_{\mathfrak{C}} c; f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
$(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$
and $\bigwedge a.\ a \in_\circ$ ℭ(|*Obj*|) $\Longrightarrow$ ℭ(|*CId*|)(|*a*|) : $a \mapsto_{\mathfrak{C}} a$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{C}} b \Longrightarrow$ ℭ(|*CId*|)(|*b*|) $\circ_{A\mathfrak{C}} f = f$
and $\bigwedge b\ c\ f.\ f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A\mathfrak{C}}$ ℭ(|*CId*|)(|*b*|) $= f$
and ℭ(|*Obj*|) $\subseteq_\circ$ *Vset* α
and $\bigwedge A\ B.\ [\![\ A \subseteq_\circ$ ℭ(|*Obj*|); $B \subseteq_\circ$ ℭ(|*Obj*|); $A \in_\circ$ *Vset* α; $B \in_\circ$ *Vset* $\alpha\ ]\!] \Longrightarrow$
$(\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.$ *Hom* ℭ $a\ b) \in_\circ$ *Vset* α
using *assms* **by** (*simp add*: *categoryD′*)

Slicing.

context *category*
begin

interpretation *smc*: *semicategory* α ‹*cat-smc* ℭ› **by** (*rule cat-semicategory*)

sublocale *Dom*: *vsv* ‹𝔈(|*Dom*|)›
  by (*rule smc.Dom.vsv-axioms*[*unfolded slicing-simps*])
sublocale *Cod*: *vsv* ‹𝔈(|*Cod*|)›
  by (*rule smc.Cod.vsv-axioms*[*unfolded slicing-simps*])
sublocale *Comp*: *pbinop* ‹𝔈(|*Arr*|)› ‹𝔈(|*Comp*|)›
  by (*rule smc.Comp.pbinop-axioms*[*unfolded slicing-simps*])

lemmas-with [*unfolded slicing-simps*]:
  cat-Dom-vdomain[*cat-cs-simps*] = *smc.smc-Dom-vdomain*
  and *cat-Dom-vrange* = *smc.smc-Dom-vrange*
  and *cat-Cod-vdomain*[*cat-cs-simps*] = *smc.smc-Cod-vdomain*
  and *cat-Cod-vrange* = *smc.smc-Cod-vrange*
  and *cat-Obj-vsubset-Vset* = *smc.smc-Obj-vsubset-Vset*
  and *cat-Hom-vifunion-in-Vset*[*cat-cs-intros*] = *smc.smc-Hom-vifunion-in-Vset*
  and *cat-Obj-if-Dom-vrange* = *smc.smc-Obj-if-Dom-vrange*
  and *cat-Obj-if-Cod-vrange* = *smc.smc-Obj-if-Cod-vrange*
  and *cat-is-arrD* = *smc.smc-is-arrD*
  and *cat-is-arrE*[*elim*] = *smc.smc-is-arrE*
  and *cat-in-ArrE*[*elim*] = *smc.smc-in-ArrE*
  and *cat-Hom-in-Vset*[*cat-cs-intros*] = *smc.smc-Hom-in-Vset*
  and *cat-Arr-vsubset-Vset* = *smc.smc-Arr-vsubset-Vset*
  and *cat-Dom-vsubset-Vset* = *smc.smc-Dom-vsubset-Vset*
  and *cat-Cod-vsubset-Vset* = *smc.smc-Cod-vsubset-Vset*
  and *cat-Obj-in-Vset* = *smc.smc-Obj-in-Vset*
  and *cat-in-Obj-in-Vset*[*cat-cs-intros*] = *smc.smc-in-Obj-in-Vset*
  and *cat-Arr-in-Vset* = *smc.smc-Arr-in-Vset*
  and *cat-in-Arr-in-Vset*[*cat-cs-intros*] = *smc.smc-in-Arr-in-Vset*
  and *cat-Dom-in-Vset* = *smc.smc-Dom-in-Vset*
  and *cat-Cod-in-Vset* = *smc.smc-Cod-in-Vset*
  and *cat-semicategory-if-ge-Limit* = *smc.smc-semicategory-if-ge-Limit*
  and *cat-Dom-app-in-Obj* = *smc.smc-Dom-app-in-Obj*
  and *cat-Cod-app-in-Obj* = *smc.smc-Cod-app-in-Obj*
  and *cat-Arr-vempty-if-Obj-vempty* = *smc.smc-Arr-vempty-if-Obj-vempty*
  and *cat-Dom-vempty-if-Arr-vempty* = *smc.smc-Dom-vempty-if-Arr-vempty*
  and *cat-Cod-vempty-if-Arr-vempty* = *smc.smc-Cod-vempty-if-Arr-vempty*

lemmas [*cat-cs-intros*] = *cat-is-arrD*(*2*,*3*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
  cat-Comp-vdomain = *smc.smc-Comp-vdomain*
  and *cat-Comp-is-arr*[*cat-cs-intros*] = *smc.smc-Comp-is-arr*
  and *cat-Comp-assoc*[*cat-cs-intros*] = *smc.smc-Comp-assoc*
  and *cat-Comp-vdomainI*[*cat-cs-intros*] = *smc.smc-Comp-vdomainI*
  and *cat-Comp-vdomainE*[*elim*!] = *smc.smc-Comp-vdomainE*
  and *cat-Comp-vdomain-is-composable-arrs* =
    smc.smc-Comp-vdomain-is-composable-arrs
  and *cat-Comp-vrange* = *smc.smc-Comp-vrange*
  and *cat-Comp-vsubset-Vset* = *smc.smc-Comp-vsubset-Vset*
  and *cat-Comp-in-Vset* = *smc.smc-Comp-in-Vset*
  and *cat-Comp-vempty-if-Arr-vempty* = *smc.smc-Comp-vempty-if-Arr-vempty*
  and *cat-assoc-helper* = *smc.smc-assoc-helper*
  and *cat-pattern-rectangle-right* = *smc.smc-pattern-rectangle-right*
  and *cat-pattern-rectangle-left* = *smc.smc-pattern-rectangle-left*
  and *is-epic-arrI* = *smc.is-epic-arrI*
  and *is-epic-arrD*[*dest*] = *smc.is-epic-arrD*
  and *is-epic-arrE*[*elim*!] = *smc.is-epic-arrE*
  and *cat-comp-is-monic-arr*[*cat-arrow-cs-intros*] = *smc.smc-Comp-is-monic-arr*
  and *cat-comp-is-epic-arr*[*cat-arrow-cs-intros*] = *smc.smc-Comp-is-epic-arr*

and *cat-comp-is-monic-arr-is-monic-arr* =
smc.smc-Comp-is-monic-arr-is-monic-arr
and *cat-is-zero-arr-comp-right*[*cat-arrow-cs-intros*] =
smc.smc-is-zero-arr-Comp-right
and *cat-is-zero-arr-comp-left*[*cat-arrow-cs-intros*] =
smc.smc-is-zero-arr-Comp-left

lemma *cat-Comp-is-arr'*[*cat-cs-intros*]:
assumes $g : b \mapsto_{\mathfrak{C}} c$
and $f : a \mapsto_{\mathfrak{C}} b$
and $\mathfrak{C}' = \mathfrak{C}$
shows $g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}'} c$
using *assms*(*1*,*2*) **unfolding** *assms*(*3*) **by** (*rule cat-Comp-is-arr*)

end

lemmas [*cat-cs-simps*] = *is-idem-arrD*(*2*)

lemmas [*cat-cs-simps*] = *category.cat-Comp-assoc*

lemmas [*cat-cs-intros*] =
category.cat-Comp-vdomainI
category.cat-Hom-in-Vset
category.cat-is-arrD(*1–3*)
category.cat-Comp-is-arr'
category.cat-Comp-is-arr

lemmas [*cat-arrow-cs-intros*] =
is-monic-arrD(*1*)
is-epic-arr-is-arr
category.cat-comp-is-monic-arr
category.cat-comp-is-epic-arr
category.cat-is-zero-arr-comp-right
category.cat-is-zero-arr-comp-left

lemmas [*cat-cs-intros*] = *HomI*
lemmas [*cat-cs-simps*] = *in-Hom-iff*

Elementary properties.

lemma *cat-eqI*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{A}(\lfloor Obj \rfloor) = \mathfrak{B}(\lfloor Obj \rfloor)$
and $\mathfrak{A}(\lfloor Arr \rfloor) = \mathfrak{B}(\lfloor Arr \rfloor)$
and $\mathfrak{A}(\lfloor Dom \rfloor) = \mathfrak{B}(\lfloor Dom \rfloor)$
and $\mathfrak{A}(\lfloor Cod \rfloor) = \mathfrak{B}(\lfloor Cod \rfloor)$
and $\mathfrak{A}(\lfloor Comp \rfloor) = \mathfrak{B}(\lfloor Comp \rfloor)$
and $\mathfrak{A}(\lfloor CId \rfloor) = \mathfrak{B}(\lfloor CId \rfloor)$
shows $\mathfrak{A} = \mathfrak{B}$
proof–
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom*: $\mathcal{D}_\circ\ \mathfrak{A} = 6_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps V-cs-simps*)
show $\mathcal{D}_\circ\ \mathfrak{A} = \mathcal{D}_\circ\ \mathfrak{B}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps V-cs-simps*)

```
    show a ∈∘ 𝒟∘ 𝔄 ⟹ 𝔄(|a|) = 𝔅(|a|) for a
      by (unfold dom, elim-in-numeral, insert assms) (auto simp: dg-field-simps)
  qed auto
qed

lemma cat-smc-eqI:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔄(|CId|) = 𝔅(|CId|)
    and cat-smc 𝔄 = cat-smc 𝔅
  shows 𝔄 = 𝔅
proof(rule cat-eqI[of α])
  from assms(4) have
    cat-smc 𝔄(|Obj|) = cat-smc 𝔅(|Obj|)
    cat-smc 𝔄(|Arr|) = cat-smc 𝔅(|Arr|)
    cat-smc 𝔄(|Dom|) = cat-smc 𝔅(|Dom|)
    cat-smc 𝔄(|Cod|) = cat-smc 𝔅(|Cod|)
    cat-smc 𝔄(|Comp|) = cat-smc 𝔅(|Comp|)
    by auto
  then show
    𝔄(|Obj|) = 𝔅(|Obj|)
    𝔄(|Arr|) = 𝔅(|Arr|)
    𝔄(|Dom|) = 𝔅(|Dom|)
    𝔄(|Cod|) = 𝔅(|Cod|)
    𝔄(|Comp|) = 𝔅(|Comp|)
    unfolding slicing-simps by simp-all
qed (auto simp: assms)

lemma (in category) cat-def:
  𝔈 = [𝔈(|Obj|), 𝔈(|Arr|), 𝔈(|Dom|), 𝔈(|Cod|), 𝔈(|Comp|), 𝔈(|CId|)]∘
proof(rule vsv-eqI)
  have dom-lhs: 𝒟∘ 𝔈 = 6ℕ
    by (cs-concl cs-shallow cs-simp: cat-cs-simps V-cs-simps)
  have dom-rhs: 𝒟∘ [𝔈(|Obj|), 𝔈(|Arr|), 𝔈(|Dom|), 𝔈(|Cod|), 𝔈(|Comp|), 𝔈(|CId|)]∘ = 6ℕ
    by (simp add: nat-omega-simps)
  then show 𝒟∘ 𝔈 = 𝒟∘ [𝔈(|Obj|), 𝔈(|Arr|), 𝔈(|Dom|), 𝔈(|Cod|), 𝔈(|Comp|), 𝔈(|CId|)]∘
    unfolding dom-lhs dom-rhs by simp
  show a ∈∘ 𝒟∘ 𝔈 ⟹
    𝔈(|a|) = [𝔈(|Obj|), 𝔈(|Arr|), 𝔈(|Dom|), 𝔈(|Cod|), 𝔈(|Comp|), 𝔈(|CId|)]∘(|a|)
    for a
    unfolding dom-lhs
    by elim-in-numeral (simp-all add: dg-field-simps nat-omega-simps)
qed auto
```

Size.

```
lemma (in category) cat-CId-vsubset-Vset: 𝔈(|CId|) ⊆∘ Vset α
proof(intro vsubsetI)
  fix af assume af ∈∘ 𝔈(|CId|)
  then obtain a f
    where af-def: af = ⟨a, f⟩
      and a: a ∈∘ 𝒟∘ (𝔈(|CId|))
      and f: f ∈∘ ℛ∘ (𝔈(|CId|))
    by (auto elim: CId.vbrelation-vinE)
  from a have a ∈∘ Vset α by (auto simp: cat-cs-simps intro: cat-cs-intros)
  from f cat-CId-vrange have f ∈∘ 𝔈(|Arr|) by auto
  then have f ∈∘ Vset α by (auto simp: cat-cs-simps intro: cat-cs-intros)
  then show af ∈∘ Vset α
    by (simp add: af-def Limit-vpair-in-VsetI ‹a ∈∘ Vset α›)
```

```
qed

lemma (in category) cat-category-in-Vset-4: 𝔈 ∈∘ Vset (α + 4ℕ)
proof–
  note [folded VPow-iff, folded Vset-succ[OF Ord-α], cat-cs-intros] =
    cat-Obj-vsubset-Vset
    cat-Arr-vsubset-Vset
    cat-Dom-vsubset-Vset
    cat-Cod-vsubset-Vset
    cat-Comp-vsubset-Vset
    cat-CId-vsubset-Vset
  show ?thesis
    by (subst cat-def, succ-of-numeral)
      (
        cs-concl
          cs-simp: plus-V-succ-right V-cs-simps
          cs-intro: cat-cs-intros V-cs-intros
      )
qed

lemma (in category) cat-CId-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔈(|CId|) ∈∘ Vset β
proof–
  interpret 𝒵 β by (rule assms(1))
  from assms have 𝒟∘ (𝔈(|CId|)) ∈∘ Vset β
    by (auto simp: cat-cs-simps cat-Obj-in-Vset)
  moreover from assms cat-CId-vrange have ℛ∘ (𝔈(|CId|)) ∈∘ Vset β
    by (auto intro: cat-Arr-in-Vset)
  ultimately show ?thesis by (blast intro: 𝒵-Limit-αω)
qed

lemma (in category) cat-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔈 ∈∘ Vset β
proof–
  interpret β: 𝒵 β by (rule assms(1))
  show ?thesis
  proof(rule vsv.vsv-Limit-vsv-in-VsetI)
    have dom: 𝒟∘ 𝔈 = 6ℕ
      by (cs-concl cs-shallow cs-simp: cat-cs-simps V-cs-simps)
    from assms show 𝒟∘ 𝔈 ∈∘ Vset β
      unfolding dom by (simp add: 𝒵.ord-of-nat-in-Vset)
    have n ∈∘ 𝒟∘ 𝔈 ⟹ 𝔈(|n|) ∈∘ Vset β for n
      unfolding dom
      by
        (
          elim-in-numeral,
          all‹rewrite in ⌑ ∈∘ - dg-field-simps[symmetric]›,
          insert assms
        )
        (
          auto simp:
            cat-Obj-in-Vset
            cat-Arr-in-Vset
            cat-Dom-in-Vset
            cat-Cod-in-Vset
            cat-Comp-in-Vset
```

```
        cat-CId-in-Vset
    )
  then show ℛ∘ 𝔈 ⊆∘ Vset β by (metis vsubsetI vrange-atD)
  show vfinite (𝒟∘ 𝔈) unfolding dom by auto
 qed (simp-all add: 𝒵-Limit-αω vsv-axioms)
qed

lemma (in category) cat-category-if-ge-Limit:
 assumes 𝒵 β and α ∈∘ β
 shows category β 𝔈
 by (rule categoryI)
  (
   auto
    intro: cat-cs-intros
    simp: cat-cs-simps assms vfsequence-axioms cat-semicategory-if-ge-Limit
  )

lemma tiny-category[simp]: small {𝔈. category α 𝔈}
proof(cases ‹𝒵 α›)
 case True
 from category.cat-in-Vset[of α] show ?thesis
  by (intro down[of - ‹Vset (α + ω)›])
   (auto simp: True 𝒵.𝒵-Limit-αω 𝒵.𝒵-ω-αω 𝒵.intro 𝒵.𝒵-α-αω)
next
 case False
 then have {𝔈. category α 𝔈} = {} by auto
 then show ?thesis by simp
qed

lemma (in 𝒵) categories-in-Vset:
 assumes 𝒵 β and α ∈∘ β
 shows set {𝔈. category α 𝔈} ∈∘ Vset β
proof(rule vsubset-in-VsetI)
 interpret β: 𝒵 β by (rule assms(1))
 show set {𝔈. category α 𝔈} ⊆∘ Vset (α + 4ℕ)
 proof(intro vsubsetI)
  fix 𝔈 assume prems: 𝔈 ∈∘ set {𝔈. category α 𝔈}
  interpret category α 𝔈 using prems by simp
  show 𝔈 ∈∘ Vset (α + 4ℕ)
   unfolding VPow-iff by (rule cat-category-in-Vset-4)
 qed
 from assms(2) show Vset (α + 4ℕ) ∈∘ Vset β
  by (cs-concl cs-shallow cs-intro: V-cs-intros Ord-cs-intros)
qed

lemma category-if-category:
 assumes category β 𝔈
  and 𝒵 α
  and 𝔈(|Obj|) ⊆∘ Vset α
  and ⋀A B. ⟦ A ⊆∘ 𝔈(|Obj|); B ⊆∘ 𝔈(|Obj|); A ∈∘ Vset α; B ∈∘ Vset α ⟧ ⟹
   (⋃∘a∈∘A. ⋃∘b∈∘B. Hom 𝔈 a b) ∈∘ Vset α
 shows category α 𝔈
proof–
 interpret category β 𝔈 by (rule assms(1))
 interpret α: 𝒵 α by (rule assms(2))
 show ?thesis
 proof(intro categoryI)
  show vfsequence 𝔈 by (simp add: vfsequence-axioms)
```

```
    show semicategory α (cat-smc 𝔈)
      by (rule semicategory-if-semicategory, unfold slicing-simps)
        (auto intro!: assms(1,3,4) slicing-intros)
  qed (auto intro: cat-cs-intros simp: cat-cs-simps)
qed
```

Further elementary properties.

```
sublocale category ⊆ CId: v11 ‹𝔈(|CId|)›
proof(rule vsv.vsv-valeq-v11I, unfold cat-cs-simps)
  fix a b assume prems:
    a ∈₀ 𝔈(|Obj|) b ∈₀ 𝔈(|Obj|) 𝔈(|CId|)(|a|) = 𝔈(|CId|)(|b|)
  have 𝔈(|CId|)(|a|) : b ↦_𝔈 b 𝔈(|CId|)(|a|) : a ↦_𝔈 a
    by (subst prems(3))
      (cs-concl cs-simp: cat-cs-simps cs-intro: prems(1,2) cat-cs-intros)+
  with prems show a = b by auto
qed auto

lemma (in category) cat-CId-vempty-if-Arr-vempty:
  assumes 𝔈(|Arr|) = 0
  shows 𝔈(|CId|) = 0
  using assms cat-CId-vrange by (auto intro: CId.vsv-vrange-vempty)
```

2.3 Opposite category

2.3.1 Definition and elementary properties

See Chapter II-2 in [7].

```
definition op-cat :: V ⇒ V
  where op-cat 𝔈 = [𝔈(|Obj|), 𝔈(|Arr|), 𝔈(|Cod|), 𝔈(|Dom|), fflip (𝔈(|Comp|)), 𝔈(|CId|)]₀
```

Components.

```
lemma op-cat-components:
  shows [cat-op-simps]: op-cat 𝔈(|Obj|) = 𝔈(|Obj|)
    and [cat-op-simps]: op-cat 𝔈(|Arr|) = 𝔈(|Arr|)
    and [cat-op-simps]: op-cat 𝔈(|Dom|) = 𝔈(|Cod|)
    and [cat-op-simps]: op-cat 𝔈(|Cod|) = 𝔈(|Dom|)
    and op-cat 𝔈(|Comp|) = fflip (𝔈(|Comp|))
    and [cat-op-simps]: op-cat 𝔈(|CId|) = 𝔈(|CId|)
  unfolding op-cat-def dg-field-simps by (auto simp: nat-omega-simps)

lemma op-cat-component-intros[cat-op-intros]:
  shows a ∈₀ 𝔈(|Obj|) ⟹ a ∈₀ op-cat 𝔈(|Obj|)
    and f ∈₀ 𝔈(|Arr|) ⟹ f ∈₀ op-cat 𝔈(|Arr|)
  unfolding cat-op-simps by simp-all
```

Slicing.

```
lemma cat-smc-op-cat[slicing-commute]: op-smc (cat-smc 𝔈) = cat-smc (op-cat 𝔈)
  unfolding cat-smc-def op-cat-def op-smc-def dg-field-simps
  by (simp add: nat-omega-simps)

lemma (in category) op-smc-op-cat[cat-op-simps]: op-smc (op-cat 𝔈) = cat-smc 𝔈
  using Comp.pbinop-fflip-fflip
  unfolding op-smc-def op-cat-def cat-smc-def dg-field-simps
  by (simp add: nat-omega-simps)

lemma op-cat-is-arr[cat-op-simps]: f : b ↦_op-cat 𝔈 a ⟷ f : a ↦_𝔈 b
  unfolding cat-op-simps is-arr-def by auto
```

lemmas [*cat-op-intros*] = *op-cat-is-arr*[*THEN iffD2*]

lemma *op-cat-Hom*[*cat-op-simps*]: *Hom* (*op-cat* $\mathfrak{C}$) *a b* = *Hom* $\mathfrak{C}$ *b a*
  unfolding *cat-op-simps* **by** *simp*

lemma *op-cat-obj-initial*[*cat-op-simps*]:
  obj-initial (*op-cat* $\mathfrak{C}$) *a* ⟷ *obj-terminal* $\mathfrak{C}$ *a*
  unfolding *obj-initial-def obj-terminal-def*
  unfolding *smc-op-simps cat-op-simps*
  ..

lemmas [*cat-op-intros*] = *op-cat-obj-initial*[*THEN iffD2*]

lemma *op-cat-obj-terminal*[*cat-op-simps*]:
  obj-terminal (*op-cat* $\mathfrak{C}$) *a* ⟷ *obj-initial* $\mathfrak{C}$ *a*
  unfolding *obj-initial-def obj-terminal-def*
  unfolding *smc-op-simps cat-op-simps*
  ..

lemmas [*cat-op-intros*] = *op-cat-obj-terminal*[*THEN iffD2*]

lemma *op-cat-obj-null*[*cat-op-simps*]: *obj-null* (*op-cat* $\mathfrak{C}$) *a* ⟷ *obj-null* $\mathfrak{C}$ *a*
  unfolding *obj-null-def cat-op-simps* **by** *auto*

lemmas [*cat-op-intros*] = *op-cat-obj-null*[*THEN iffD2*]

context *category*
begin

interpretation *smc*: *semicategory* α ‹*cat-smc* $\mathfrak{C}$› **by** (*rule cat-semicategory*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
  op-cat-Comp-vrange[*cat-op-simps*] = *smc.op-smc-Comp-vrange*
  and *op-cat-Comp*[*cat-op-simps*] = *smc.op-smc-Comp*
  and *op-cat-is-epic-arr*[*cat-op-simps*] = *smc.op-smc-is-epic-arr*
  and *op-cat-is-monic-arr*[*cat-op-simps*] = *smc.op-smc-is-monic-arr*
  and *op-cat-is-zero-arr*[*cat-op-simps*] = *smc.op-smc-is-zero-arr*

end

lemmas [*cat-op-simps*] =
  category.op-cat-Comp-vrange
  category.op-cat-Comp
  category.op-cat-is-epic-arr
  category.op-cat-is-monic-arr
  category.op-cat-is-zero-arr

context
  fixes $\mathfrak{C}$ :: *V*
begin

lemmas-with [
  where $\mathfrak{C}$=‹*cat-smc* $\mathfrak{C}$›, *unfolded slicing-simps slicing-commute*[*symmetric*]
  ]:
  op-cat-Comp-vdomain[*cat-op-simps*] = *op-smc-Comp-vdomain*

end

Elementary properties.

lemma *op-cat-vsv*[*cat-op-intros*]: *vsv* (*op-cat* $\mathfrak{C}$) **unfolding** *op-cat-def* **by** *auto*

2.3.2 Further properties

lemma (**in** *category*) *category-op*[*cat-cs-intros*]: *category* α (*op-cat* $\mathfrak{C}$)
proof(*intro categoryI*, *unfold cat-op-simps*)
  show *vfsequence* (*op-cat* $\mathfrak{C}$) **unfolding** *op-cat-def* **by** *simp*
  show *vcard* (*op-cat* $\mathfrak{C}$) = $6_{\mathbb{N}}$
    unfolding *op-cat-def* **by** (*simp add*: *nat-omega-simps*)
next
  fix *f a b* **assume** *f* : *b* $\mapsto_{\mathfrak{C}}$ *a*
  with *category-axioms* **show** $\mathfrak{C}(\lfloor CId \rfloor)(\lfloor b \rfloor) \circ_{A\, op\text{-}cat}\, \mathfrak{C}$ *f* = *f*
    by (*cs-concl* **cs-simp**: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*)
next
  fix *f b c* **assume** *f* : *c* $\mapsto_{\mathfrak{C}}$ *b*
  with *category-axioms* **show** *f* $\circ_{A\, op\text{-}cat}\, \mathfrak{C}$ $\mathfrak{C}(\lfloor CId \rfloor)(\lfloor b \rfloor)$ = *f*
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)
qed
  (
    auto simp:
      cat-cs-simps
      cat-op-simps
      slicing-commute[*symmetric*]
      smc-op-intros
      cat-cs-intros
      cat-semicategory
)

lemmas *category-op*[*cat-op-intros*] = *category.category-op*

lemma (**in** *category*) *cat-op-cat-op-cat*[*cat-op-simps*]: *op-cat* (*op-cat* $\mathfrak{C}$) = $\mathfrak{C}$
proof(*rule cat-eqI*, *unfold cat-op-simps op-cat-components*)
  show *category* α (*op-cat* (*op-cat* $\mathfrak{C}$))
    by (*simp add*: *category.category-op category-op*)
  show *fflip* (*fflip* ($\mathfrak{C}(\lfloor Comp \rfloor)$)) = $\mathfrak{C}(\lfloor Comp \rfloor)$ **by** (*rule Comp.pbinop-fflip-fflip*)
qed (*auto simp*: *cat-cs-intros*)

lemmas *cat-op-cat-op-cat*[*cat-op-simps*] = *category.cat-op-cat-op-cat*

lemma *eq-op-cat-iff*[*cat-op-simps*]:
  assumes *category* α $\mathfrak{A}$ **and** *category* α $\mathfrak{B}$
  shows *op-cat* $\mathfrak{A}$ = *op-cat* $\mathfrak{B}$ $\longleftrightarrow$ $\mathfrak{A}$ = $\mathfrak{B}$
proof
  interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
  assume *prems*: *op-cat* $\mathfrak{A}$ = *op-cat* $\mathfrak{B}$
  show $\mathfrak{A}$ = $\mathfrak{B}$
  proof(*rule cat-eqI*)
    show
      $\mathfrak{A}(\lfloor Obj \rfloor)$ = $\mathfrak{B}(\lfloor Obj \rfloor)$
      $\mathfrak{A}(\lfloor Arr \rfloor)$ = $\mathfrak{B}(\lfloor Arr \rfloor)$
      $\mathfrak{A}(\lfloor Dom \rfloor)$ = $\mathfrak{B}(\lfloor Dom \rfloor)$
      $\mathfrak{A}(\lfloor Cod \rfloor)$ = $\mathfrak{B}(\lfloor Cod \rfloor)$

$\mathfrak{A}(\!|Comp|\!) = \mathfrak{B}(\!|Comp|\!)$
$\mathfrak{A}(\!|CId|\!) = \mathfrak{B}(\!|CId|\!)$
by (*metis* $\mathfrak{A}$.*cat-op-cat-op-cat* $\mathfrak{B}$.*cat-op-cat-op-cat prems*)+
qed (*auto intro*: *cat-cs-intros*)
qed *auto*

2.4 Monic arrow and epic arrow

lemma (**in** *category*) *cat-CId-is-monic-arr*[*cat-arrow-cs-intros*]:
assumes $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{mon\mathfrak{C}} a$
using *assms cat-CId-is-arr*′ *cat-CId-left-left* **by** (*force intro*!: *is-monic-arrI*)

lemmas [*cat-arrow-cs-intros*] = *category.cat-CId-is-monic-arr*

lemma (**in** *category*) *cat-CId-is-epic-arr*[*cat-arrow-cs-intros*]:
assumes $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{epi\mathfrak{C}} a$
proof–
from *assms* **have** $a \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$ **unfolding** *cat-op-simps* **.**
from *category.cat-CId-is-monic-arr*[*OF category-op this*, *unfolded cat-op-simps*]
show *?thesis***.**
qed

lemmas [*cat-arrow-cs-intros*] = *category.cat-CId-is-epic-arr*

2.5 Right inverse and left inverse of an arrow

See Chapter I-5 in [7].

definition *is-right-inverse* :: $V \Rightarrow V \Rightarrow V \Rightarrow bool$
where *is-right-inverse* $\mathfrak{C}$ g f =
$(\exists a\ b.\ g : b \mapsto_{\mathfrak{C}} a \wedge f : a \mapsto_{\mathfrak{C}} b \wedge f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!))$

definition *is-left-inverse* :: $V \Rightarrow V \Rightarrow V \Rightarrow bool$
where *is-left-inverse* $\mathfrak{C} \equiv$ *is-right-inverse* (*op-cat* $\mathfrak{C}$)

Rules.

lemma *is-right-inverseI*:
assumes $g : b \mapsto_{\mathfrak{C}} a$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **and** $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
shows *is-right-inverse* $\mathfrak{C}$ g f
using *assms* **unfolding** *is-right-inverse-def* **by** *auto*

lemma *is-right-inverseD*[*dest*]:
assumes *is-right-inverse* $\mathfrak{C}$ g f
shows $\exists a\ b.\ g : b \mapsto_{\mathfrak{C}} a \wedge f : a \mapsto_{\mathfrak{C}} b \wedge f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
using *assms* **unfolding** *is-right-inverse-def* **by** *clarsimp*

lemma *is-right-inverseE*[*elim*]:
assumes *is-right-inverse* $\mathfrak{C}$ g f
obtains a b **where** $g : b \mapsto_{\mathfrak{C}} a$
and $f : a \mapsto_{\mathfrak{C}} b$
and $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
using *assms* **by** *auto*

lemma (**in** *category*) *is-left-inverseI*:
assumes $g : b \mapsto_{\mathfrak{C}} a$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **and** $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
shows *is-left-inverse* $\mathfrak{C}$ g f

proof–
  from *assms*(*3*) **have** $f \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ g = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
    unfolding *op-cat-Comp*[*OF assms*(*1,2*)].
  from
    is-right-inverseI[*of* ‹*op-cat* $\mathfrak{C}$›, *unfolded cat-op-simps*, *OF assms*(*1,2*) *this*]
  show *?thesis*
    unfolding *is-left-inverse-def* .
qed

lemma (**in** *category*) *is-left-inverseD*[*dest*]:
  assumes *is-left-inverse* $\mathfrak{C}\ g\ f$
  shows $\exists\, a\ b.\ g : b \mapsto_{\mathfrak{C}} a \wedge f : a \mapsto_{\mathfrak{C}} b \wedge g \circ_{A\,\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
proof–
  from *is-right-inverseD*[*OF assms*[*unfolded is-left-inverse-def*]] **obtain** *a b*
    where $g : b \mapsto_{op\text{-}cat\ \mathfrak{C}} a$
      and $f : a \mapsto_{op\text{-}cat\ \mathfrak{C}} b$
      and *fg*: $f \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ g = op\text{-}cat\ \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
    by *clarsimp*
  then have *g*: $g : a \mapsto_{\mathfrak{C}} b$ **and** *f*: $f : b \mapsto_{\mathfrak{C}} a$
    unfolding *cat-op-simps* **by** *simp-all*
  moreover from *fg* **have** $g \circ_{A\,\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
    unfolding *op-cat-Comp*[*OF g f*] *cat-op-simps* **by** *simp*
  ultimately show *?thesis* **by** *blast*
qed

lemma (**in** *category*) *is-left-inverseE*[*elim*]:
  assumes *is-left-inverse* $\mathfrak{C}\ g\ f$
  obtains *a b* **where** $g : b \mapsto_{\mathfrak{C}} a$
    and $f : a \mapsto_{\mathfrak{C}} b$
    and $g \circ_{A\,\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
  using *assms* **by** *auto*

Elementary properties.

lemma (**in** *category*) *op-cat-is-left-inverse*[*cat-op-simps*]:
  is-left-inverse (*op-cat* $\mathfrak{C}$) $g\ f \longleftrightarrow$ *is-right-inverse* $\mathfrak{C}\ g\ f$
  unfolding *is-left-inverse-def is-right-inverse-def cat-op-simps* **by** *simp*

lemmas [*cat-op-simps*] = *category.op-cat-is-left-inverse*

lemmas [*cat-op-intros*] = *category.op-cat-is-left-inverse*[*THEN iffD2*]

lemma (**in** *category*) *op-cat-is-right-inverse*[*cat-op-simps*]:
  is-right-inverse (*op-cat* $\mathfrak{C}$) $g\ f \longleftrightarrow$ *is-left-inverse* $\mathfrak{C}\ g\ f$
  unfolding *is-left-inverse-def is-right-inverse-def cat-op-simps* **by** *simp*

lemmas [*cat-op-simps*] = *category.op-cat-is-right-inverse*

lemmas [*cat-op-intros*] = *category.op-cat-is-right-inverse*[*THEN iffD2*]

2.6 Inverse of an arrow

See Chapter I-5 in [7].

definition *is-inverse* :: $V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  where *is-inverse* $\mathfrak{C}\ g\ f$ =
    (
      $\exists\, a\ b.$
        $g : b \mapsto_{\mathfrak{C}} a \wedge$

$f : a \mapsto_{\mathfrak{C}} b \wedge$
$g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!) \wedge$
$f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
)

Rules.

lemma *is-inverseI*:
assumes $g : b \mapsto_{\mathfrak{C}} a$
and $f : a \mapsto_{\mathfrak{C}} b$
and $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
and $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
shows *is-inverse* $\mathfrak{C}$ *g f*
using *assms* **unfolding** *is-inverse-def* **by** *auto*

lemma *is-inverseD*[*dest*]:
assumes *is-inverse* $\mathfrak{C}$ *g f*
shows
(
$\exists a\ b.$
$g : b \mapsto_{\mathfrak{C}} a \wedge$
$f : a \mapsto_{\mathfrak{C}} b \wedge$
$g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!) \wedge$
$f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
)
using *assms* **unfolding** *is-inverse-def* **by** *auto*

lemma *is-inverseE*[*elim*]:
assumes *is-inverse* $\mathfrak{C}$ *g f*
obtains *a b* **where** $g : b \mapsto_{\mathfrak{C}} a$
and $f : a \mapsto_{\mathfrak{C}} b$
and $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
and $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
using *assms* **by** *auto*

Elementary properties.

lemma (**in** *category*) *op-cat-is-inverse*[*cat-op-simps*]:
is-inverse (*op-cat* $\mathfrak{C}$) *g f* $\longleftrightarrow$ *is-inverse* $\mathfrak{C}$ *g f*
by (*rule iffI*; *unfold is-inverse-def cat-op-simps*) (*metis op-cat-Comp*)+

lemmas [*cat-op-simps*] = *category.op-cat-is-inverse*

lemmas [*cat-op-intros*] = *category.op-cat-is-inverse*[*THEN iffD2*]

lemma *is-inverse-sym*: *is-inverse* $\mathfrak{C}$ *g f* $\longleftrightarrow$ *is-inverse* $\mathfrak{C}$ *f g*
unfolding *is-inverse-def* **by** *auto*

lemma (**in** *category*) *cat-is-inverse-eq*:
— See Chapter I-5 in [7].
assumes *is-inverse* $\mathfrak{C}$ *h f* **and** *is-inverse* $\mathfrak{C}$ *g f*
shows *h* = *g*
using *assms*
proof(*elim is-inverseE*)
fix *a b a′ b′*
assume *prems*:
$h : b \mapsto_{\mathfrak{C}} a$
$f : a \mapsto_{\mathfrak{C}} b$
$h \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
$f \circ_{A\mathfrak{C}} h = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$

```
    g : b' ↦_ℭ a'
    f : a' ↦_ℭ b'
    g ∘_Aℭ f = ℭ(|CId|)(|a'|)
  then have ab: a' = a b' = b by auto
  from prems have gf: g ∘_Aℭ f = ℭ(|CId|)(|a|) and g: g : b ↦_ℭ a
    unfolding ab by simp-all
  from prems(1) have h = (g ∘_Aℭ f) ∘_Aℭ h
    unfolding gf by (simp add: cat-cs-simps)
  also with category-axioms prems(1,2) g have ... = g
    by
      (
        cs-concl cs-shallow
          cs-simp: prems(4) cat-cs-simps cs-intro: cat-cs-intros
      )
  finally show h = g by simp
qed

lemma is-inverse-Comp-CId-left:
  — See Chapter I-5 in [7].
  assumes is-inverse ℭ g' g and g : a ↦_ℭ b
  shows g' ∘_Aℭ g = ℭ(|CId|)(|a|)
  using assms by auto

lemma is-inverse-Comp-CId-right:
  assumes is-inverse ℭ g' g and g : a ↦_ℭ b
  shows g ∘_Aℭ g' = ℭ(|CId|)(|b|)
  by (metis assms is-arrD(3) is-inverseE)

lemma (in category) cat-is-inverse-Comp:
  — See Chapter I-5 in [7].
  assumes gbc[intro]: g : b ↦_ℭ c
    and fab[intro]: f : a ↦_ℭ b
    and g'g[intro]: is-inverse ℭ g' g
    and f'f[intro]: is-inverse ℭ f' f
  shows is-inverse ℭ (f' ∘_Aℭ g') (g ∘_Aℭ f)
proof–
  from g'g gbc f'f fab have g'cb: g' : c ↦_ℭ b and f'ba: f' : b ↦_ℭ a
    by (metis is-arrD(2,3) is-inverseD)+
  with assms have f'g': f' ∘_Aℭ g' : c ↦_ℭ a and gf: g ∘_Aℭ f : a ↦_ℭ c
    by (auto intro: cat-Comp-is-arr)
  have ff': is-inverse ℭ f f' using assms by (simp add: is-inverse-sym)
  note [simp] =
    cat-Comp-assoc[symmetric, OF f'g' gbc fab]
    cat-Comp-assoc[OF f'ba g'cb gbc]
    is-inverse-Comp-CId-left[OF g'g gbc]
    cat-Comp-assoc[symmetric, OF gf f'ba g'cb]
    cat-Comp-assoc[OF gbc fab f'ba]
    is-inverse-Comp-CId-left[OF ff' f'ba]
    cat-CId-right-left[OF f'ba]
    cat-CId-right-left[OF gbc]
  show ?thesis
    by (intro is-inverseI, rule f'g', rule gf)
      (auto intro: is-inverse-Comp-CId-left is-inverse-Comp-CId-right)
qed

lemma (in category) cat-is-inverse-Comp':
  assumes g : b ↦_ℭ c
    and f : a ↦_ℭ b
```

and *is-inverse* $\mathfrak{C}$ g' g
and *is-inverse* $\mathfrak{C}$ f' f
and $f'g' = f' \circ_{A\mathfrak{C}} g'$
and $gf = g \circ_{A\mathfrak{C}} f$
shows *is-inverse* $\mathfrak{C}$ $f'g'$ gf
using *assms*(*1–4*) **unfolding** *assms*(*5,6*) **by** (*intro cat-is-inverse-Comp*)

lemmas [*cat-cs-intros*] = *category.cat-is-inverse-Comp'*

lemma *is-inverse-is-right-inverse*[*dest*]:
assumes *is-inverse* $\mathfrak{C}$ g f
shows *is-right-inverse* $\mathfrak{C}$ g f
using *assms* **by** (*auto intro*: *is-right-inverseI*)

lemma (**in** *category*) *cat-is-inverse-is-left-inverse*[*dest*]:
assumes *is-inverse* $\mathfrak{C}$ g f
shows *is-left-inverse* $\mathfrak{C}$ g f
proof–
interpret *op*: *category* α ‹*op-cat* $\mathfrak{C}$› **by** (*auto intro*!: *cat-cs-intros*)
from *assms* **have** *is-inverse* (*op-cat* $\mathfrak{C}$) g f **by** (*simp add*: *cat-op-simps*)
from *is-inverse-is-right-inverse*[*OF this*] **show** *?thesis*
unfolding *is-left-inverse-def* **.**
qed

lemma (**in** *category*) *cat-is-right-left-inverse-is-inverse*:
assumes *is-right-inverse* $\mathfrak{C}$ g f *is-left-inverse* $\mathfrak{C}$ g f
shows *is-inverse* $\mathfrak{C}$ g f
using *assms*
proof(*elim is-right-inverseE is-left-inverseE*)
fix a b c d **assume** *prems*:
$g : b \mapsto_{\mathfrak{C}} a$
$f : a \mapsto_{\mathfrak{C}} b$
$f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor b \rfloor)$
$g : d \mapsto_{\mathfrak{C}} c$
$f : c \mapsto_{\mathfrak{C}} d$
$g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor)$
then have *dbca*: $d = b$ $c = a$ **by** *auto*
note [*cat-cs-simps*] = *prems*(*3,6*)[*unfolded dbca*]
from *prems*(*1,2*) **show** *is-inverse* $\mathfrak{C}$ g f
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros is-inverseI*
)
qed

2.7 Isomorphism

See Chapter I-5 in [7].

definition *is-iso-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
where *is-iso-arr* $\mathfrak{C}$ a b $f \longleftrightarrow$
$(f : a \mapsto_{\mathfrak{C}} b \wedge (\exists g.\ \textit{is-inverse}\ \mathfrak{C}\ g\ f))$

syntax *-is-iso-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹- : - ↦$_{iso_1}$ -› [*51*, *51*, *51*] *51*)
syntax-consts *-is-iso-arr* ⇌ *is-iso-arr*
translations $f : a \mapsto_{iso\mathfrak{C}} b \rightleftharpoons$ *CONST is-iso-arr* $\mathfrak{C}$ a b f

Rules.

lemma *is-iso-arrI*:
assumes $f : a \mapsto_{\mathfrak{C}} b$ **and** *is-inverse* $\mathfrak{C}$ g f
shows $f : a \mapsto_{iso\mathfrak{C}} b$
using *assms* **unfolding** *is-iso-arr-def* **by** *auto*

lemma *is-iso-arrD*[*dest*]:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$
shows $f : a \mapsto_{\mathfrak{C}} b$ **and** $\exists g.$ *is-inverse* $\mathfrak{C}$ g f
using *assms* **unfolding** *is-iso-arr-def* **by** *auto*

lemma *is-iso-arrE*[*elim*]:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$
obtains g **where** $f : a \mapsto_{\mathfrak{C}} b$ **and** *is-inverse* $\mathfrak{C}$ g f
using *assms* **by** *force*

lemma *is-iso-arrE′*:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$
obtains g **where** $g : b \mapsto_{iso\mathfrak{C}} a$
and $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor a \rfloor)$
and $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor b \rfloor)$
proof–
from *assms* **obtain** g **where** *f*: $f : a \mapsto_{\mathfrak{C}} b$ *is-inverse* $\mathfrak{C}$ g f **by** *auto*
then have $g : b \mapsto_{\mathfrak{C}} a$
and $f : a \mapsto_{\mathfrak{C}} b$
and *gf*: $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor a \rfloor)$
and *fg*: $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor b \rfloor)$
by *auto*
then have *g*: $g : b \mapsto_{iso\mathfrak{C}} a$
by (*cs-concl* **cs-shallow cs-intro**: *is-inverseI is-iso-arrI*)
from *that f g gf fg* **show** *?thesis* **by** *simp*
qed

Elementary properties.

lemma (**in** *category*) *op-cat-is-iso-arr*[*cat-op-simps*]:
$f : b \mapsto_{iso\,op\text{-}cat\,\mathfrak{C}} a \longleftrightarrow f : a \mapsto_{iso\mathfrak{C}} b$
unfolding *is-iso-arr-def cat-op-simps* **by** *simp*

lemmas [*cat-op-simps*] = *category.op-cat-is-iso-arr*

lemmas [*cat-op-intros*] = *category.op-cat-is-iso-arr*[*THEN iffD2*]

lemma (**in** *category*) *is-iso-arrI′*:
assumes $f : a \mapsto_{\mathfrak{C}} b$
and $g : b \mapsto_{\mathfrak{C}} a$
and $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor a \rfloor)$
and $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor b \rfloor)$
shows $f : a \mapsto_{iso\mathfrak{C}} b$ **and** $g : b \mapsto_{iso\mathfrak{C}} a$
proof–
from *assms* **have** *gf*: *is-inverse* $\mathfrak{C}$ g f **by** (*auto intro*: *is-inverseI*)
from *assms* **have** *fg*: *is-inverse* $\mathfrak{C}$ f g **by** (*auto intro*: *is-inverseI*)
show $f : a \mapsto_{iso\mathfrak{C}} b$ **and** $g : b \mapsto_{iso\mathfrak{C}} a$
by
(
intro
is-iso-arrI[*OF assms*(*1*) *gf*]
is-iso-arrI[*OF assms*(*2*) *fg*]
)+

qed

lemma (**in** *category*) *cat-is-inverse-is-iso-arr*:
assumes $f : a \mapsto_{\mathfrak{C}} b$ **and** *is-inverse* $\mathfrak{C}\ g\ f$
shows $g : b \mapsto_{iso\mathfrak{C}} a$
proof(*intro is-iso-arrI is-inverseI*)
from *assms*(*2*) **obtain** $a'\ b'$
where *g*: $g : b' \mapsto_{\mathfrak{C}} a'$
and *f*: $f : a' \mapsto_{\mathfrak{C}} b'$
and *gf*: $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a'|\!)$
and *fg*: $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b'|\!)$
by *auto*
with *assms*(*1*) **have** *a'b'*: $a' = a\ b' = b$ **by** *auto*
from *g f gf fg* **show**
$g : b \mapsto_{\mathfrak{C}} a$
$f : a \mapsto_{\mathfrak{C}} b$
$g : b \mapsto_{\mathfrak{C}} a$
$f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
$g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
unfolding *a'b'* **by** *auto*
qed

lemma (**in** *category*) *cat-Comp-is-iso-arr*[*cat-arrow-cs-intros*]:
assumes $g : b \mapsto_{iso\mathfrak{C}} c$ **and** $f : a \mapsto_{iso\mathfrak{C}} b$
shows $g \circ_{A\mathfrak{C}} f : a \mapsto_{iso\mathfrak{C}} c$
proof–
from *assms* **have** [*intro*]: $g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
by (*auto intro*: *cat-cs-intros*)
from *assms*(*1*) **obtain** g' **where** *g'g*: *is-inverse* $\mathfrak{C}\ g'\ g$ **by** *force*
with *assms*(*1*) **have** [*intro*]: $g' : c \mapsto_{\mathfrak{C}} b$
by (*elim is-iso-arrE*)
(*auto simp*: *is-iso-arrD cat-is-inverse-is-iso-arr*)
from *assms*(*2*) **obtain** f' **where** *f'f*: *is-inverse* $\mathfrak{C}\ f'\ f$ **by** *auto*
with *assms*(*2*) **have** [*intro*]: $f' : b \mapsto_{\mathfrak{C}} a$
by (*elim is-iso-arrE*)
(*auto simp*: *is-iso-arrD cat-is-inverse-is-iso-arr*)
have $f' \circ_{A\mathfrak{C}} g' : c \mapsto_{\mathfrak{C}} a$ **by** (*auto intro*: *cat-cs-intros*)
from *cat-is-inverse-Comp*[*OF - - g'g f'f*] *assms*
have *is-inverse* $\mathfrak{C}\ (f' \circ_{A\mathfrak{C}} g')\ (g \circ_{A\mathfrak{C}} f)$
by (*elim is-iso-arrE*) *simp*
then show *?thesis* **by** (*auto intro*: *is-iso-arrI*)
qed

lemmas [*cat-arrow-cs-intros*] = *category.cat-Comp-is-iso-arr*

lemma (**in** *category*) *cat-CId-is-iso-arr*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{iso\mathfrak{C}} a$
using *assms*
by
(
cs-concl **cs-shallow**
cs-intro: *cat-cs-intros is-inverseI cat-is-inverse-is-iso-arr*
cs-simp: *cat-cs-simps*
)

lemma (**in** *category*) *cat-CId-is-iso-arr'*[*cat-arrow-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$

```
    and 𝔈′ = 𝔈
    and b = a
    and c = a
  shows 𝔈(⌊CId⌋)(⌊a⌋) : b ↦_iso𝔈′ c
  using assms(1)
  unfolding assms(2–4)
  by (rule cat-CId-is-iso-arr)

lemmas [cat-arrow-cs-intros] = category.cat-CId-is-iso-arr′

lemma (in category) cat-is-iso-arr-is-monic-arr[cat-arrow-cs-intros]:
  assumes f : a ↦_iso𝔈 b
  shows f : a ↦_mon𝔈 b
proof(intro is-monic-arrI)
  note [cat-cs-intros] = is-iso-arrD(1)
  show f : a ↦_𝔈 b by (intro is-iso-arrD(1)[OF assms])
  fix h g c assume prems:
    h : c ↦_𝔈 a  g : c ↦_𝔈 a  f ∘_A𝔈 h = f ∘_A𝔈 g
  from assms obtain f′
    where f′: f′ : b ↦_iso𝔈 a
      and [cat-cs-simps]: f′ ∘_A𝔈 f = 𝔈(⌊CId⌋)(⌊a⌋)
    by (auto elim: is-iso-arrE′)
  from category-axioms assms prems(1,2) have h = (f′ ∘_A𝔈 f) ∘_A𝔈 h
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  also from category-axioms assms prems(1,2) f′ have ... = (f′ ∘_A𝔈 f) ∘_A𝔈 g
    by (cs-concl cs-simp: prems(3) cat-cs-simps cs-intro: cat-cs-intros)
  also from category-axioms assms prems(1,2) f′ have ... = g
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  finally show h = g by simp
qed

lemmas [cat-arrow-cs-intros] = category.cat-is-iso-arr-is-monic-arr

lemma (in category) cat-is-iso-arr-is-epic-arr:
  assumes f : a ↦_iso𝔈 b
  shows f : a ↦_epi𝔈 b
  using assms
  by
    (
      rule
        category.cat-is-iso-arr-is-monic-arr[
          OF category-op, unfolded cat-op-simps
          ]
    )

lemmas [cat-arrow-cs-intros] = category.cat-is-iso-arr-is-epic-arr

lemma (in category) cat-is-iso-arr-if-is-monic-arr-is-right-inverse:
  assumes f : a ↦_mon𝔈 b and is-right-inverse 𝔈 g f
  shows f : a ↦_iso𝔈 b
proof–
  note f-is-monic-arrD = is-monic-arrD[OF assms(1)]
  from is-right-inverseD[OF assms(2)] f-is-monic-arrD(1)
  have g: g : b ↦_𝔈 a and fg: f ∘_A𝔈 g = 𝔈(⌊CId⌋)(⌊b⌋)
    by auto
  from f-is-monic-arrD(1) g have gf: g ∘_A𝔈 f : a ↦_𝔈 a
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from g have CId-a: 𝔈(⌊CId⌋)(⌊a⌋) : a ↦_𝔈 a
```

```
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ?thesis
  proof
    (
      intro
        is-iso-arrI
        cat-is-right-left-inverse-is-inverse
        is-left-inverseI,
      rule f-is-monic-arrD(1),
      rule assms(2),
      rule g,
      rule f-is-monic-arrD(1)
    )
    from f-is-monic-arrD(1) g have f ∘_A\<^bsub>𝔈\<^esub> (g ∘_A\<^bsub>𝔈\<^esub> f) = f ∘_A\<^bsub>𝔈\<^esub> g ∘_A\<^bsub>𝔈\<^esub> f
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    also from f-is-monic-arrD(1) g have ... = f ∘_A\<^bsub>𝔈\<^esub> 𝔈(|CId|)(|a|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps fg cs-intro: cat-cs-intros)
    finally have f ∘_A\<^bsub>𝔈\<^esub> (g ∘_A\<^bsub>𝔈\<^esub> f) = f ∘_A\<^bsub>𝔈\<^esub> 𝔈(|CId|)(|a|) by simp
    from f-is-monic-arrD(2)[OF gf CId-a this] show g ∘_A\<^bsub>𝔈\<^esub> f = 𝔈(|CId|)(|a|).
  qed
qed

lemma (in category) cat-is-iso-arr-if-is-epic-arr-is-left-inverse:
  assumes f : a ↦_epi\<^bsub>𝔈\<^esub> b and is-left-inverse 𝔈 g f
  shows f : a ↦_iso\<^bsub>𝔈\<^esub> b
  using assms
  by
    (
      rule category.cat-is-iso-arr-if-is-monic-arr-is-right-inverse[
        OF category-op, unfolded cat-op-simps
        ]
    )
```

2.8 The inverse arrow

See Chapter I-5 in [7].

```
definition the-inverse :: V ⇒ V ⇒ V (‹(-^-1\<^bsub>C1\<^esub>)› [1000] 999)
  where f^-1\<^bsub>C𝔈\<^esub> = (THE g. is-inverse 𝔈 g f)
```

Elementary properties.

```
lemma (in category) cat-is-inverse-is-inverse-the-inverse:
  assumes is-inverse 𝔈 g f
  shows is-inverse 𝔈 (f^-1\<^bsub>C𝔈\<^esub>) f
  unfolding the-inverse-def
proof(rule theI)
  fix g' assume is-inverse 𝔈 g' f
  then show g' = g by (meson cat-is-inverse-eq assms)
qed (rule assms)

lemma (in category) cat-is-inverse-eq-the-inverse:
  assumes is-inverse 𝔈 g f
  shows g = f^-1\<^bsub>C𝔈\<^esub>
  by (meson assms cat-is-inverse-is-inverse-the-inverse cat-is-inverse-eq)
```

The inverse arrow is an inverse of an isomorphism.

```
lemma (in category) cat-the-inverse-is-inverse:
  assumes f : a ↦_iso\<^bsub>𝔈\<^esub> b
```

shows *is-inverse* $\mathfrak{C}$ $(f^{-1}{}_{C\mathfrak{C}})$ f
proof–
from *assms* **obtain** g **where** *is-inverse* $\mathfrak{C}$ g f **by** *auto*
then show *is-inverse* $\mathfrak{C}$ $(f^{-1}{}_{C\mathfrak{C}})$ f
by (*rule cat-is-inverse-is-inverse-the-inverse*)
qed

lemma (**in** *category*) *cat-the-inverse-is-iso-arr*:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$
shows $f^{-1}{}_{C\mathfrak{C}} : b \mapsto_{iso\mathfrak{C}} a$
proof–
from *assms* **have** *f*: $f : a \mapsto_{\mathfrak{C}} b$ **by** *auto*
have *is-inverse* $\mathfrak{C}$ $(f^{-1}{}_{C\mathfrak{C}})$ f **by** (*rule cat-the-inverse-is-inverse*[*OF assms*])
from *cat-is-inverse-is-iso-arr*[*OF f this*] **show** *?thesis* **.**
qed

lemma (**in** *category*) *cat-the-inverse-is-iso-arr'*:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $f^{-1}{}_{C\mathfrak{C}} : b \mapsto_{iso\mathfrak{C}'} a$
using *assms*(*1*)
unfolding *assms*(*2*)
by (*rule cat-the-inverse-is-iso-arr*)

lemmas [*cat-cs-intros*] = *category.cat-the-inverse-is-iso-arr'*

lemma (**in** *category*) *op-cat-the-inverse*:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$
shows $f^{-1}{}_{C\,op\text{-}cat\ \mathfrak{C}} = f^{-1}{}_{C\mathfrak{C}}$
proof–
from *assms* **have** $f : b \mapsto_{iso\,op\text{-}cat\ \mathfrak{C}} a$ **unfolding** *cat-op-simps* **by** *simp*
from *assms* **show** *?thesis*
by
(
intro
category.cat-is-inverse-eq-the-inverse[
symmetric, *OF category-op*, *unfolded cat-op-simps*
]
cat-the-inverse-is-inverse
)
qed

lemmas [*cat-op-simps*] = *category.op-cat-the-inverse*

lemma (**in** *category*) *cat-Comp-the-inverse*:
assumes $g : b \mapsto_{iso\mathfrak{C}} c$ **and** $f : a \mapsto_{iso\mathfrak{C}} b$
shows $(g \circ_{A\mathfrak{C}} f)^{-1}{}_{C\mathfrak{C}} = f^{-1}{}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} g^{-1}{}_{C\mathfrak{C}}$
proof–
from *assms* **have** $g \circ_{A\mathfrak{C}} f : a \mapsto_{iso\mathfrak{C}} c$
by (*cs-concl* **cs-shallow cs-intro**: *cat-arrow-cs-intros*)
then have *inv-gf*: *is-inverse* $\mathfrak{C}$ $((g \circ_{A\mathfrak{C}} f)^{-1}{}_{C\mathfrak{C}})$ $(g \circ_{A\mathfrak{C}} f)$
by (*intro cat-the-inverse-is-inverse*)
from *assms* **have** *is-inverse* $\mathfrak{C}$ $(g^{-1}{}_{C\mathfrak{C}})$ g *is-inverse* $\mathfrak{C}$ $(f^{-1}{}_{C\mathfrak{C}})$ f
by (*auto intro*: *cat-the-inverse-is-inverse*)
with *category-axioms assms* **have**
is-inverse $\mathfrak{C}$ $(f^{-1}{}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} g^{-1}{}_{C\mathfrak{C}})$ $(g \circ_{A\mathfrak{C}} f)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-arrow-cs-intros*)
from *inv-gf this* **show** $(g \circ_{A\mathfrak{C}} f)^{-1}{}_{C\mathfrak{C}} = f^{-1}{}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} g^{-1}{}_{C\mathfrak{C}}$
by (*meson cat-is-inverse-eq*)

qed

lemmas [*cat-cs-simps*] = *category.cat-Comp-the-inverse*

lemma (**in** *category*) *cat-the-inverse-Comp-CId*:
  assumes $f : a \mapsto_{iso\mathfrak{C}} b$
  shows *cat-the-inverse-Comp-CId-left*: ${f^{-1}}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
    and *cat-the-inverse-Comp-CId-right*: $f \circ_{A\mathfrak{C}} {f^{-1}}_{C\mathfrak{C}} = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
proof–
  from *assms* **show** ${f^{-1}}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
    by
      (
        cs-concl
          cs-simp: *is-inverse-Comp-CId-left*
          cs-intro: *cat-the-inverse-is-inverse cat-arrow-cs-intros*
)
  from *assms* **show** $f \circ_{A\mathfrak{C}} {f^{-1}}_{C\mathfrak{C}} = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
    by
      (
        cs-concl
          cs-simp: *is-inverse-Comp-CId-right*
          cs-intro: *cat-the-inverse-is-inverse cat-arrow-cs-intros*
)
qed

lemmas [*cat-cs-simps*] = *category.cat-the-inverse-Comp-CId*

lemma (**in** *category*) *cat-the-inverse-the-inverse*:
  assumes $f : a \mapsto_{iso\mathfrak{C}} b$
  shows $({f^{-1}}_{C\mathfrak{C}})^{-1}{}_{C\mathfrak{C}} = f$
proof–
  from *assms* **have** $({f^{-1}}_{C\mathfrak{C}})^{-1}{}_{C\mathfrak{C}} = ({f^{-1}}_{C\mathfrak{C}})^{-1}{}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} {f^{-1}}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} f$
    by
      (
        cs-concl
          cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-arrow-cs-intros*
)
  also from *assms* **have** $\ldots = f$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-arrow-cs-intros*
)
  finally show *?thesis* **.**
qed

lemmas [*cat-cs-simps*] = *category.cat-the-inverse-the-inverse*

2.9 Isomorphic objects

See Chapter I-5 in [7].

definition *obj-iso* :: $V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *obj-iso* $\mathfrak{C}\ a\ b \longleftrightarrow (\exists f.\ f : a \mapsto_{iso\mathfrak{C}} b)$

syntax *-obj-iso* :: $V \Rightarrow V \Rightarrow V \Rightarrow bool$ $(\langle(\text{-}/ \approx_{obj1} \text{-})\rangle\ [55, 56]\ 55)$
syntax-consts *-obj-iso* $\rightleftharpoons$ *obj-iso*
translations $a \approx_{obj\mathfrak{C}} b \rightleftharpoons CONST\ obj\text{-}iso\ \mathfrak{C}\ a\ b$

Rules.

lemma *obj-isoI*:
assumes $f : a \mapsto_{iso\mathfrak{C}} b$
shows $a \approx_{obj\mathfrak{C}} b$
using *assms* **unfolding** *obj-iso-def* **by** *auto*

lemma *obj-isoD*[*dest*]:
assumes $a \approx_{obj\mathfrak{C}} b$
shows $\exists f.\ f : a \mapsto_{iso\mathfrak{C}} b$
using *assms* **unfolding** *obj-iso-def* **by** *auto*

lemma *obj-isoE*[*elim*!]:
assumes $a \approx_{obj\mathfrak{C}} b$
obtains f **where** $f : a \mapsto_{iso\mathfrak{C}} b$
using *assms* **by** *auto*

Elementary properties.

lemma (**in** *category*) *op-cat-obj-iso*[*cat-op-simps*]:
$a \approx_{obj op\text{-}cat\ \mathfrak{C}} b = b \approx_{obj\mathfrak{C}} a$
unfolding *obj-iso-def cat-op-simps* **..**

lemmas [*cat-op-simps*] = *category.op-cat-obj-iso*

lemmas [*cat-op-intros*] = *category.op-cat-obj-iso*[*THEN iffD2*]

Equivalence relation.

lemma (**in** *category*) *cat-obj-iso-refl*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $a \approx_{obj\mathfrak{C}} a$
using *assms* **by** (*auto intro*: *obj-isoI cat-arrow-cs-intros*)

lemma (**in** *category*) *cat-obj-iso-sym*[*sym*]:
assumes $a \approx_{obj\mathfrak{C}} b$
shows $b \approx_{obj\mathfrak{C}} a$
using *assms*
by (*elim obj-isoE is-iso-arrE*)
(*metis obj-iso-def cat-is-inverse-is-iso-arr*)

lemma (**in** *category*) *cat-obj-iso-trans*[*trans*]:
assumes $a \approx_{obj\mathfrak{C}} b$ **and** $b \approx_{obj\mathfrak{C}} c$
shows $a \approx_{obj\mathfrak{C}} c$
using *assms* **by** (*auto intro*: *cat-Comp-is-iso-arr obj-isoI*)

2.10 Terminal object and initial object

lemma (**in** *category*) *cat-obj-terminal-CId*:
— See Chapter I-5 in [7].
assumes *obj-terminal* $\mathfrak{C}$ a **and** $f : a \mapsto_{\mathfrak{C}} a$
shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) = f$
using *assms* **by** (*elim obj-terminalE*) (*metis cat-CId-is-arr*)

lemma (**in** *category*) *cat-obj-initial-CId*:
— See Chapter I-5 in [7].
assumes *obj-initial* $\mathfrak{C}$ a **and** $f : a \mapsto_{\mathfrak{C}} a$
shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) = f$
using *assms*
by (*rule category.cat-obj-terminal-CId*[*OF category-op*, *unfolded cat-op-simps*])

lemma (in *category*) *cat-obj-terminal-obj-iso*:
— See Chapter I-5 in [7].
assumes *obj-terminal* $\mathfrak{C}$ *a* **and** *obj-terminal* $\mathfrak{C}$ a'
shows $a \approx_{obj\mathfrak{C}} a'$
proof–
from *assms* **obtain** *f* **where** *f*: $f : a \mapsto_{\mathfrak{C}} a'$ **by** *auto*
from *assms* **obtain** f' **where** f': $f' : a' \mapsto_{\mathfrak{C}} a$ **by** *auto*
from *f f′ cat-obj-terminal-CId cat-Comp-is-arr*
have *f′f*: *is-inverse* $\mathfrak{C}$ f' *f*
by (*intro is-inverseI*[*OF f′ f*]) (*metis assms*(*1*), *metis assms*(*2*))
with *f* **show** *?thesis*
by (*cs-concl* **cs-shallow cs-intro**: *obj-isoI is-iso-arrI*)
qed

lemma (in *category*) *cat-obj-initial-obj-iso*:
— See Chapter I-5 in [7].
assumes *obj-initial* $\mathfrak{C}$ *a* **and** *obj-initial* $\mathfrak{C}$ a'
shows $a' \approx_{obj\mathfrak{C}} a$
proof–
interpret *op*: *category* α ‹*op-cat* $\mathfrak{C}$› **by** (*auto intro*: *cat-cs-intros*)
from *assms* **show** *?thesis*
by (*rule op.cat-obj-terminal-obj-iso*[*unfolded cat-op-simps*])
qed

2.11 Null object

lemma (in *category*) *cat-obj-null-obj-iso*:
— See Chapter I-5 in [7].
assumes *obj-null* $\mathfrak{C}$ *z* **and** *obj-null* $\mathfrak{C}$ z'
shows $z \approx_{obj\mathfrak{C}} z'$
using *assms* **by** (*simp add*: *cat-obj-terminal-obj-iso obj-nullD*(*2*))

2.12 Groupoid

See Chapter I-5 in [7].

locale *groupoid* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
assumes *grpd-is-iso-arr*: $f : a \mapsto_{\mathfrak{C}} b \Longrightarrow f : a \mapsto_{iso\mathfrak{C}} b$

Rules.

mk-ide rf *groupoid-def*[*unfolded groupoid-axioms-def*]
|*intro groupoidI*|
|*dest groupoidD*[*dest*]|
|*elim groupoidE*[*elim*]|

3 Smallness for categories

3.1 Background

An explanation of the methodology chosen for the exposition of all matters related to the size of the categories and associated entities is given in [8].

named-theorems *cat-small-cs-simps*
named-theorems *cat-small-cs-intros*

3.2 Tiny category

3.2.1 Definition and elementary properties

locale *tiny-category* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ + *CId*: *vsv* ‹$\mathfrak{C}(\!|CId|\!)$› **for** α $\mathfrak{C}$ +
  assumes *tiny-cat-length*[*cat-cs-simps*]: *vcard* $\mathfrak{C}$ = $6_{\mathbb{N}}$
    and *tiny-cat-tiny-semicategory*[*slicing-intros*]:
      tiny-semicategory α (*cat-smc* $\mathfrak{C}$)
    and *tiny-cat-CId-vdomain*[*cat-cs-simps*]: $\mathcal{D}_{\circ}$ ($\mathfrak{C}(\!|CId|\!)$) = $\mathfrak{C}(\!|Obj|\!)$
    and *tiny-cat-CId-is-arr*[*cat-cs-intros*]:
      $a \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{C}} a$
    and *tiny-cat-CId-left-left*[*cat-cs-simps*]:
      $f : a \mapsto_{\mathfrak{C}} b \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|b|\!) \circ_{A\mathfrak{C}} f = f$
    and *tiny-cat-CId-right-left*[*cat-cs-simps*]:
      $f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A\mathfrak{C}} \mathfrak{C}(\!|CId|\!)(\!|b|\!) = f$

lemmas [*slicing-intros*] = *tiny-category.tiny-cat-tiny-semicategory*

Rules.

lemma (**in** *tiny-category*) *tiny-category-axioms'*[*cat-small-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *tiny-category* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule tiny-category-axioms*)

mk-ide rf *tiny-category-def*[*unfolded tiny-category-axioms-def*]
  |*intro tiny-categoryI*|
  |*dest tiny-categoryD*[*dest*]|
  |*elim tiny-categoryE*[*elim*]|

lemma *tiny-categoryI'*:
  assumes *category* α $\mathfrak{C}$ **and** $\mathfrak{C}(\!|Obj|\!) \in_{\circ}$ *Vset* α **and** $\mathfrak{C}(\!|Arr|\!) \in_{\circ}$ *Vset* α
  shows *tiny-category* α $\mathfrak{C}$
proof–
  interpret *category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))
  show *?thesis*
  proof(*intro tiny-categoryI*)
    from *assms* **show** *tiny-semicategory* α (*cat-smc* $\mathfrak{C}$)
      by (*intro tiny-semicategoryI'*) (*auto simp*: *slicing-simps*)
  qed (*auto simp*: *vfsequence-axioms cat-cs-simps cat-cs-intros*)
qed

lemma *tiny-categoryI''*:
  assumes $\mathcal{Z}$ α
    and *vfsequence* $\mathfrak{C}$
    and *vcard* $\mathfrak{C}$ = $6_{\mathbb{N}}$
    and *vsv* ($\mathfrak{C}(\!|Dom|\!)$)
    and *vsv* ($\mathfrak{C}(\!|Cod|\!)$)
    and *vsv* ($\mathfrak{C}(\!|Comp|\!)$)
    and *vsv* ($\mathfrak{C}(\!|CId|\!)$)

and $\mathcal{D}_\circ\ (\mathfrak{C}(\!|Dom|\!)) = \mathfrak{C}(\!|Arr|\!)$
and $\mathcal{R}_\circ\ (\mathfrak{C}(\!|Dom|\!)) \subseteq_\circ \mathfrak{C}(\!|Obj|\!)$
and $\mathcal{D}_\circ\ (\mathfrak{C}(\!|Cod|\!)) = \mathfrak{C}(\!|Arr|\!)$
and $\mathcal{R}_\circ\ (\mathfrak{C}(\!|Cod|\!)) \subseteq_\circ \mathfrak{C}(\!|Obj|\!)$
and $\bigwedge gf.\ gf \in_\circ \mathcal{D}_\circ\ (\mathfrak{C}(\!|Comp|\!)) \longleftrightarrow$
$(\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_\circ \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b)$
and $\mathcal{D}_\circ\ (\mathfrak{C}(\!|CId|\!)) = \mathfrak{C}(\!|Obj|\!)$
and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
and $\bigwedge c\ d\ h\ b\ g\ a\ f.\ [\![\ h : c \mapsto_{\mathfrak{C}} d;\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
$(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$
and $\bigwedge a.\ a \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{C}} a$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{C}} b \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|b|\!) \circ_{A\mathfrak{C}} f = f$
and $\bigwedge b\ c\ f.\ f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A\mathfrak{C}} \mathfrak{C}(\!|CId|\!)(\!|b|\!) = f$
and $\mathfrak{C}(\!|Obj|\!) \in_\circ Vset\ \alpha$
and $\mathfrak{C}(\!|Arr|\!) \in_\circ Vset\ \alpha$
shows *tiny-category* $\alpha\ \mathfrak{C}$
by (*intro tiny-categoryI tiny-semicategoryI''*, *unfold slicing-simps*)
(*simp-all add*: *cat-smc-def nat-omega-simps assms*)

Slicing.

context *tiny-category*
begin

interpretation *smc*: *tiny-semicategory* α ‹*cat-smc* $\mathfrak{C}$›
by (*rule tiny-cat-tiny-semicategory*)

lemmas-with [*unfolded slicing-simps*]:
tiny-cat-semicategory = *smc.semicategory-axioms*
and *tiny-cat-Obj-in-Vset*[*cat-small-cs-intros*] = *smc.tiny-smc-Obj-in-Vset*
and *tiny-cat-Arr-in-Vset*[*cat-small-cs-intros*] = *smc.tiny-smc-Arr-in-Vset*
and *tiny-cat-Dom-in-Vset*[*cat-small-cs-intros*] = *smc.tiny-smc-Dom-in-Vset*
and *tiny-cat-Cod-in-Vset*[*cat-small-cs-intros*] = *smc.tiny-smc-Cod-in-Vset*
and *tiny-cat-Comp-in-Vset*[*cat-small-cs-intros*] = *smc.tiny-smc-Comp-in-Vset*

end

Elementary properties.

sublocale *tiny-category* ⊆ *category*
by (*rule categoryI*)
(
auto simp:
vfsequence-axioms tiny-cat-semicategory cat-cs-intros cat-cs-simps
)

lemmas (**in** *tiny-category*) *tiny-cat-category* = *category-axioms*

lemmas [*cat-small-cs-intros*] = *tiny-category.tiny-cat-category*

Size.

lemma (**in** *tiny-category*) *tiny-cat-CId-in-Vset*: $\mathfrak{C}(\!|CId|\!) \in_\circ Vset\ \alpha$
proof–
from *tiny-cat-Obj-in-Vset* **have** $\mathcal{D}_\circ\ (\mathfrak{C}(\!|CId|\!)) \in_\circ Vset\ \alpha$
by (*simp add*: *tiny-cat-Obj-in-Vset cat-cs-simps*)
moreover from *tiny-cat-Arr-in-Vset cat-CId-vrange tiny-cat-Arr-in-Vset* **have**
$\mathcal{R}_\circ\ (\mathfrak{C}(\!|CId|\!)) \in_\circ Vset\ \alpha$
by *auto*
ultimately show *?thesis* **by** (*blast intro*: *Z-Limit-αω*)

qed

lemma (**in** *tiny-category*) *tiny-cat-in-Vset*: $\mathfrak{C} \in_{\circ} \mathit{Vset}\ \alpha$
proof–
note [*cat-cs-intros*] =
tiny-cat-Obj-in-Vset
tiny-cat-Arr-in-Vset
tiny-cat-Dom-in-Vset
tiny-cat-Cod-in-Vset
tiny-cat-Comp-in-Vset
tiny-cat-CId-in-Vset
show *?thesis*
by (*subst cat-def*) (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros V-cs-intros*)
qed

lemma *tiny-category*[*simp*]: *small* $\{\mathfrak{C}.\ \mathit{tiny\text{-}category}\ \alpha\ \mathfrak{C}\}$
proof(*rule down*)
show $\{\mathfrak{C}.\ \mathit{tiny\text{-}category}\ \alpha\ \mathfrak{C}\} \subseteq \mathit{elts}\ (\mathit{set}\ \{\mathfrak{C}.\ \mathit{category}\ \alpha\ \mathfrak{C}\})$
by (*auto intro*: *cat-small-cs-intros*)
qed

lemma *small-categories-vsubset-Vset*: *set* $\{\mathfrak{C}.\ \mathit{tiny\text{-}category}\ \alpha\ \mathfrak{C}\} \subseteq_{\circ} \mathit{Vset}\ \alpha$
by (*rule vsubsetI*) (*simp-all add*: *tiny-category.tiny-cat-in-Vset*)

lemma (**in** *category*) *cat-tiny-category-if-ge-Limit*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows *tiny-category* $\beta\ \mathfrak{C}$
proof(*intro tiny-categoryI*)
show *tiny-semicategory* β (*cat-smc* $\mathfrak{C}$)
by
(
rule semicategory.smc-tiny-semicategory-if-ge-Limit,
rule cat-semicategory;
intro assms
)
qed (*auto simp*: *assms*(*1*) *cat-cs-simps cat-cs-intros vfsequence-axioms*)

3.2.2 Opposite tiny category

lemma (**in** *tiny-category*) *tiny-category-op*: *tiny-category* α (*op-cat* $\mathfrak{C}$)
by (*intro tiny-categoryI'*)
(*auto simp*: *cat-op-simps cat-cs-intros cat-small-cs-intros*)

lemmas *tiny-category-op*[*cat-op-intros*] = *tiny-category.tiny-category-op*

3.3 Finite category

3.3.1 Definition and elementary properties

A definition of a finite category can be found in nLab [1][1].

locale *finite-category* = $\mathcal{Z}\ \alpha$ + *vfsequence* $\mathfrak{C}$ + *CId*: *vsv* $\langle \mathfrak{C}(\!|\mathit{CId}|\!)\rangle$ **for** $\alpha\ \mathfrak{C}$ +
assumes *fin-cat-length*[*cat-cs-simps*]: *vcard* $\mathfrak{C} = 6_{\mathbb{N}}$
and *fin-cat-finite-semicategory*[*slicing-intros*]:
finite-semicategory α (*cat-smc* $\mathfrak{C}$)
and *fin-cat-CId-vdomain*[*cat-cs-simps*]: $\mathcal{D}_{\circ}\ (\mathfrak{C}(\!|\mathit{CId}|\!)) = \mathfrak{C}(\!|\mathit{Obj}|\!)$
and *fin-cat-CId-is-arr*[*cat-cs-intros*]:

[1] https://ncatlab.org/nlab/show/finite+category

$a \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{C}} a$
and *fin-cat-CId-left-left*[*cat-cs-simps*]:
$f : a \mapsto_{\mathfrak{C}} b \Longrightarrow \mathfrak{C}(\!|CId|\!)(\!|b|\!) \circ_{A\mathfrak{C}} f = f$
and *fin-cat-CId-right-left*[*cat-cs-simps*]:
$f : b \mapsto_{\mathfrak{C}} c \Longrightarrow f \circ_{A\mathfrak{C}} \mathfrak{C}(\!|CId|\!)(\!|b|\!) = f$

lemmas [*slicing-intros*] = *finite-category.fin-cat-finite-semicategory*

Rules.

lemma (in *finite-category*) *fin-category-axioms'*[*cat-small-cs-intros*]:
assumes $\alpha' = \alpha$
shows *finite-category* α' $\mathfrak{C}$
unfolding *assms* **by** (*rule finite-category-axioms*)

mk-ide rf *finite-category-def*[*unfolded finite-category-axioms-def*]
|*intro finite-categoryI*|
|*dest finite-categoryD*[*dest*]|
|*elim finite-categoryE*[*elim*]|

lemma *finite-categoryI'*:
assumes *category* α $\mathfrak{C}$ **and** *vfinite* ($\mathfrak{C}(\!|Obj|\!)$) **and** *vfinite* ($\mathfrak{C}(\!|Arr|\!)$)
shows *finite-category* α $\mathfrak{C}$
proof–
interpret *category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro finite-categoryI*)
from *assms* **show** *finite-semicategory* α (*cat-smc* $\mathfrak{C}$)
by (*intro finite-semicategoryI'*) (*auto simp*: *slicing-simps*)
qed (*auto simp*: *vfsequence-axioms cat-cs-simps cat-cs-intros*)
qed

lemma *finite-categoryI''*:
assumes *tiny-category* α $\mathfrak{C}$ **and** *vfinite* ($\mathfrak{C}(\!|Obj|\!)$) **and** *vfinite* ($\mathfrak{C}(\!|Arr|\!)$)
shows *finite-category* α $\mathfrak{C}$
using *assms* **by** (*intro finite-categoryI'*) (*auto intro*: *cat-small-cs-intros*)

Slicing.

context *finite-category*
begin

interpretation *smc*: *finite-semicategory* α ‹*cat-smc* $\mathfrak{C}$›
by (*rule fin-cat-finite-semicategory*)

lemmas-with [*unfolded slicing-simps*]:
fin-cat-tiny-semicategory = *smc.tiny-semicategory-axioms*
and *fin-smc-Obj-vfinite*[*cat-small-cs-intros*] = *smc.fin-smc-Obj-vfinite*
and *fin-smc-Arr-vfinite*[*cat-small-cs-intros*] = *smc.fin-smc-Arr-vfinite*

end

Elementary properties.

sublocale *finite-category* ⊆ *tiny-category*
by (*rule tiny-categoryI*)
(
auto
simp: *vfsequence-axioms*
intro:

cat-cs-intros cat-cs-simps cat-small-cs-intros
finite-category.fin-cat-tiny-semicategory
)

lemmas (**in** *finite-category*) *fin-cat-tiny-category* = *tiny-category-axioms*

lemmas [*cat-small-cs-intros*] = *finite-category.fin-cat-tiny-category*

lemma (**in** *finite-category*) *fin-cat-in-Vset*: $\mathfrak{C} \in_{\circ} \mathit{Vset}\ \alpha$
by (*rule tiny-cat-in-Vset*)

Size.

lemma *small-finite-categories*[*simp*]: *small* $\{\mathfrak{C}.\ \mathit{finite\text{-}category}\ \alpha\ \mathfrak{C}\}$
proof(*rule down*)
show $\{\mathfrak{C}.\ \mathit{finite\text{-}category}\ \alpha\ \mathfrak{C}\} \subseteq \mathit{elts}\ (\mathit{set}\ \{\mathfrak{C}.\ \mathit{tiny\text{-}category}\ \ \alpha\ \mathfrak{C}\})$
by (*auto intro*: *cat-small-cs-intros*)
qed

lemma *small-finite-categories-vsubset-Vset*:
set $\{\mathfrak{C}.\ \mathit{finite\text{-}category}\ \alpha\ \mathfrak{C}\} \subseteq_{\circ} \mathit{Vset}\ \alpha$
by (*rule vsubsetI*) (*simp-all add*: *finite-category.fin-cat-in-Vset*)

3.3.2 Opposite finite category

lemma (**in** *finite-category*) *finite-category-op*: *finite-category* α (*op-cat* $\mathfrak{C}$)
by (*intro finite-categoryI'*, *unfold cat-op-simps*)
(*auto simp*: *cat-cs-intros cat-small-cs-intros*)

lemmas *finite-category-op*[*cat-op-intros*] = *finite-category.finite-category-op*

4 Functor

4.1 Background

named-theorems *cf-cs-simps*
named-theorems *cf-cs-intros*

named-theorems *cat-cn-cs-simps*
named-theorems *cat-cn-cs-intros*

lemmas [*cat-cs-simps*] = *dg-shared-cs-simps*
lemmas [*cat-cs-intros*] = *dg-shared-cs-intros*

4.1.1 Slicing

definition *cf-smcf* :: $V \Rightarrow V$
  where *cf-smcf* $\mathfrak{C}$ =
    $[\mathfrak{C}(\!|ObjMap|\!),\ \mathfrak{C}(\!|ArrMap|\!),\ cat\text{-}smc\ (\mathfrak{C}(\!|HomDom|\!)),\ cat\text{-}smc\ (\mathfrak{C}(\!|HomCod|\!))]_\circ$

Components.

lemma *cf-smcf-components*:
  shows [*slicing-simps*]: *cf-smcf* $\mathfrak{F}(\!|ObjMap|\!)$ = $\mathfrak{F}(\!|ObjMap|\!)$
    and [*slicing-simps*]: *cf-smcf* $\mathfrak{F}(\!|ArrMap|\!)$ = $\mathfrak{F}(\!|ArrMap|\!)$
    and [*slicing-commute*]: *cf-smcf* $\mathfrak{F}(\!|HomDom|\!)$ = *cat-smc* $(\mathfrak{F}(\!|HomDom|\!))$
    and [*slicing-commute*]: *cf-smcf* $\mathfrak{F}(\!|HomCod|\!)$ = *cat-smc* $(\mathfrak{F}(\!|HomCod|\!))$
  unfolding *cf-smcf-def dghm-field-simps* **by** (*auto simp*: *nat-omega-simps*)

4.2 Definition and elementary properties

See Chapter I-3 in [7].

locale *is-functor* =
  $\mathcal{Z}\ \alpha$ + *vfsequence* $\mathfrak{F}$ + *HomDom*: *category* $\alpha\ \mathfrak{A}$ + *HomCod*: *category* $\alpha\ \mathfrak{B}$
  for $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ +
  assumes *cf-length*[*cat-cs-simps*]: *vcard* $\mathfrak{F}$ = $\mathfrak{4}_{\mathbb{N}}$
    and *cf-is-semifunctor*[*slicing-intros*]:
      cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\!\!\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{B}$
    and *cf-HomDom*[*cat-cs-simps*]: $\mathfrak{F}(\!|HomDom|\!) = \mathfrak{A}$
    and *cf-HomCod*[*cat-cs-simps*]: $\mathfrak{F}(\!|HomCod|\!) = \mathfrak{B}$
    and *cf-ObjMap-CId*[*cat-cs-intros*]:
      $c \in_\circ \mathfrak{A}(\!|Obj|\!) \Longrightarrow \mathfrak{F}(\!|ArrMap|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!) = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)|\!)$

syntax *-is-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} \mapsto\!\!\mapsto_{C1} \text{-})\rangle\ [51,\ 51,\ 51]\ 51)$
syntax-consts *-is-functor* ⇌ *is-functor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{B}$ ⇌ *CONST is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$

abbreviation (*input*) *is-cn-cf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *is-cn-cf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F} \equiv \mathfrak{F} : op\text{-}cat\ \mathfrak{A} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{B}$

syntax *-is-cn-cf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} \ {}_{C}\!\mapsto\!\!\mapsto_{1} \text{-})\rangle\ [51,\ 51,\ 51]\ 51)$
syntax-consts *-is-cn-cf* ⇌ *is-cn-cf*
translations $\mathfrak{F} : \mathfrak{A}\ {}_{C}\!\mapsto\!\!\mapsto_{\alpha} \mathfrak{B} \rightharpoonup CONST\ is\text{-}cn\text{-}cf\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$

abbreviation *all-cfs* :: $V \Rightarrow V$
  where *all-cfs* $\alpha \equiv set\ \{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{B}\}$

abbreviation *cfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$

where *cfs* α $\mathfrak{A}$ $\mathfrak{B}$ ≡ *set* $\{\mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$

lemmas [*cat-cs-simps*] =
is-functor.cf-length
is-functor.cf-HomDom
is-functor.cf-HomCod
is-functor.cf-ObjMap-CId

lemma *cn-cf-ObjMap-CId*[*cat-cn-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{A}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$ **and** $c \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
shows $\mathfrak{F}(\!|ArrMap|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!) = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)|\!)$
proof–
interpret *is-functor* α ‹*op-cat* $\mathfrak{A}$› $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
from *assms*(*2*) **have** *c*: $c \in_{\circ}$ *op-cat* $\mathfrak{A}(\!|Obj|\!)$ **unfolding** *cat-op-simps* **by** *simp*
show *?thesis* **by** (*rule cf-ObjMap-CId*[*OF c*, *unfolded cat-op-simps*])
qed

lemma (**in** *is-functor*) *cf-is-semifunctor′*:
assumes $\mathfrak{A}'$ = *cat-smc* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
shows *cf-smcf* $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{SMC\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule cf-is-semifunctor*)

lemmas [*slicing-intros*] = *is-functor.cf-is-semifunctor′*

lemma *cn-smcf-comp-is-semifunctor*:
assumes $\mathfrak{F} : \mathfrak{A}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
shows *cf-smcf* $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ ${}_{SMC}\mapsto\mapsto_{\alpha}$*cat-smc* $\mathfrak{B}$
using *assms*
unfolding *slicing-simps slicing-commute*
by (*rule is-functor.cf-is-semifunctor*)

lemma *cn-smcf-comp-is-semifunctor′*[*slicing-intros*]:
assumes $\mathfrak{F} : \mathfrak{A}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
and $\mathfrak{A}'$ = *op-smc* (*cat-smc* $\mathfrak{A}$)
and $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
shows *cf-smcf* $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{SMC\alpha} \mathfrak{B}'$
using *assms*(*1*) **unfolding** *assms*(*2*,*3*) **by** (*rule cn-smcf-comp-is-semifunctor*)

Rules.

lemma (**in** *is-functor*) *is-functor-axioms′*[*cat-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C\alpha'} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-functor-axioms*)

mk-ide rf *is-functor-def*[*unfolded is-functor-axioms-def*]
|*intro is-functorI*|
|*dest is-functorD*[*dest*]|
|*elim is-functorE*[*elim*]|

lemmas [*cat-cs-intros*] = *is-functorD*(*3*,*4*)

lemma *is-functorI′*:
assumes $\mathcal{Z}$ α
and *vfsequence* $\mathfrak{F}$
and *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and *vcard* $\mathfrak{F} = 4_{\mathbb{N}}$
and $\mathfrak{F}(\!|HomDom|\!) = \mathfrak{A}$

and $\mathfrak{F}(\lfloor HomCod \rfloor) = \mathfrak{B}$
and $vsv\ (\mathfrak{F}(\lfloor ObjMap \rfloor))$
and $vsv\ (\mathfrak{F}(\lfloor ArrMap \rfloor))$
and $\mathcal{D}_{\circ}\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
and $\mathcal{R}_{\circ}\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) \subseteq_{\circ} \mathfrak{B}(\lfloor Obj \rfloor)$
and $\mathcal{D}_{\circ}\ (\mathfrak{F}(\lfloor ArrMap \rfloor)) = \mathfrak{A}(\lfloor Arr \rfloor)$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{A}} c;\ f : a \mapsto_{\mathfrak{A}} b\ ]\!] \Longrightarrow$
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \circ_{A\mathfrak{A}} f \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$
and $(\bigwedge c.\ c \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor) \Longrightarrow \mathfrak{F}(\lfloor ArrMap \rfloor)(\mathfrak{A}(\lfloor CId \rfloor)(\lfloor c \rfloor)) = \mathfrak{B}(\lfloor CId \rfloor)(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)))$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by
(
intro is-functorI is-semifunctorI′,
unfold cf-smcf-components slicing-simps
)
(*simp-all add*: *assms cf-smcf-def nat-omega-simps category.cat-semicategory*)

lemma *is-functorD′*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{F}$
and *category* $\alpha\ \mathfrak{A}$
and *category* $\alpha\ \mathfrak{B}$
and $vcard\ \mathfrak{F} = \mathfrak{4}_{\mathbb{N}}$
and $\mathfrak{F}(\lfloor HomDom \rfloor) = \mathfrak{A}$
and $\mathfrak{F}(\lfloor HomCod \rfloor) = \mathfrak{B}$
and $vsv\ (\mathfrak{F}(\lfloor ObjMap \rfloor))$
and $vsv\ (\mathfrak{F}(\lfloor ArrMap \rfloor))$
and $\mathcal{D}_{\circ}\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
and $\mathcal{R}_{\circ}\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) \subseteq_{\circ} \mathfrak{B}(\lfloor Obj \rfloor)$
and $\mathcal{D}_{\circ}\ (\mathfrak{F}(\lfloor ArrMap \rfloor)) = \mathfrak{A}(\lfloor Arr \rfloor)$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{A}} c;\ f : a \mapsto_{\mathfrak{A}} b\ ]\!] \Longrightarrow$
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \circ_{A\mathfrak{A}} f \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$
and $(\bigwedge c.\ c \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor) \Longrightarrow \mathfrak{F}(\lfloor ArrMap \rfloor)(\mathfrak{A}(\lfloor CId \rfloor)(\lfloor c \rfloor)) = \mathfrak{B}(\lfloor CId \rfloor)(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)))$
by
(
simp-all add:
is-functorD(*2–9*)[*OF assms*]
is-semifunctorD′[*OF is-functorD*(*6*)[*OF assms*], *unfolded slicing-simps*]
)

lemma *is-functorE′*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
obtains $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{F}$
and *category* $\alpha\ \mathfrak{A}$
and *category* $\alpha\ \mathfrak{B}$
and $vcard\ \mathfrak{F} = \mathfrak{4}_{\mathbb{N}}$
and $\mathfrak{F}(\lfloor HomDom \rfloor) = \mathfrak{A}$
and $\mathfrak{F}(\lfloor HomCod \rfloor) = \mathfrak{B}$
and $vsv\ (\mathfrak{F}(\lfloor ObjMap \rfloor))$
and $vsv\ (\mathfrak{F}(\lfloor ArrMap \rfloor))$
and $\mathcal{D}_{\circ}\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
and $\mathcal{R}_{\circ}\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) \subseteq_{\circ} \mathfrak{B}(\lfloor Obj \rfloor)$

and $\mathcal{D}_\circ\ (\mathfrak{F}(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{A}} c;\ f : a \mapsto_{\mathfrak{A}} b\ ]\!] \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)(\!|g \circ_{A\mathfrak{A}} f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
and $(\bigwedge c.\ c \in_\circ \mathfrak{A}(\!|Obj|\!) \Longrightarrow \mathfrak{F}(\!|ArrMap|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!) = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)|\!))$
using *assms* **by** (*simp add*: *is-functorD'*)

A functor is a semifunctor.

context *is-functor*
begin

interpretation *smcf*: *is-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule cf-is-semifunctor*)

sublocale *ObjMap*: *vsv* ‹$\mathfrak{F}(\!|ObjMap|\!)$›
by (*rule smcf.ObjMap.vsv-axioms*[*unfolded slicing-simps*])
sublocale *ArrMap*: *vsv* ‹$\mathfrak{F}(\!|ArrMap|\!)$›
by (*rule smcf.ArrMap.vsv-axioms*[*unfolded slicing-simps*])

lemmas-with [*unfolded slicing-simps*]:
cf-ObjMap-vsv = *smcf.smcf-ObjMap-vsv*
and *cf-ArrMap-vsv* = *smcf.smcf-ArrMap-vsv*
and *cf-ObjMap-vdomain*[*cat-cs-simps*] = *smcf.smcf-ObjMap-vdomain*
and *cf-ObjMap-vrange* = *smcf.smcf-ObjMap-vrange*
and *cf-ArrMap-vdomain*[*cat-cs-simps*] = *smcf.smcf-ArrMap-vdomain*
and *cf-ArrMap-is-arr* = *smcf.smcf-ArrMap-is-arr*
and *cf-ArrMap-is-arr''*[*cat-cs-intros*] = *smcf.smcf-ArrMap-is-arr''*
and *cf-ArrMap-is-arr'*[*cat-cs-intros*] = *smcf.smcf-ArrMap-is-arr'*
and *cf-ObjMap-app-in-HomCod-Obj*[*cat-cs-intros*] =
smcf.smcf-ObjMap-app-in-HomCod-Obj
and *cf-ArrMap-vrange* = *smcf.smcf-ArrMap-vrange*
and *cf-ArrMap-app-in-HomCod-Arr*[*cat-cs-intros*] =
smcf.smcf-ArrMap-app-in-HomCod-Arr
and *cf-ObjMap-vsubset-Vset* = *smcf.smcf-ObjMap-vsubset-Vset*
and *cf-ArrMap-vsubset-Vset* = *smcf.smcf-ArrMap-vsubset-Vset*
and *cf-ObjMap-in-Vset* = *smcf.smcf-ObjMap-in-Vset*
and *cf-ArrMap-in-Vset* = *smcf.smcf-ArrMap-in-Vset*
and *cf-is-semifunctor-if-ge-Limit* = *smcf.smcf-is-semifunctor-if-ge-Limit*
and *cf-is-arr-HomCod* = *smcf.smcf-is-arr-HomCod*
and *cf-vimage-dghm-ArrMap-vsubset-Hom* =
smcf.smcf-vimage-dghm-ArrMap-vsubset-Hom

lemmas-with [*unfolded slicing-simps*]:
cf-ArrMap-Comp = *smcf.smcf-ArrMap-Comp*

end

lemmas [*cat-cs-simps*] =
is-functor.cf-ObjMap-vdomain
is-functor.cf-ArrMap-vdomain
is-functor.cf-ArrMap-Comp

lemmas [*cat-cs-intros*] =
is-functor.cf-ObjMap-app-in-HomCod-Obj
is-functor.cf-ArrMap-app-in-HomCod-Arr
is-functor.cf-ArrMap-is-arr'

Elementary properties.

lemma *cn-cf-ArrMap-Comp*[*cat-cn-cs-simps*]:
  assumes *category* α $\mathfrak{A}$
    and $\mathfrak{F} : \mathfrak{A}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
    and $g : c \mapsto_{\mathfrak{A}} b$
    and $f : b \mapsto_{\mathfrak{A}} a$
  shows $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \circ_{A\mathfrak{A}} g \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$
proof–
  interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{F}$: *is-functor* α ‹*op-cat* $\mathfrak{A}$› $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
  show *?thesis*
    by
      (
        rule cn-smcf-ArrMap-Comp
          [
            OF
              $\mathfrak{A}$.*cat-semicategory*
              $\mathfrak{F}$.*cf-is-semifunctor*[*unfolded slicing-commute*[*symmetric*]],
            unfolded slicing-simps,
            OF assms(*3*,*4*)
          ]
)
qed

lemma *cf-eqI*:
  assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
    and $\mathfrak{G}(\lfloor ObjMap \rfloor) = \mathfrak{F}(\lfloor ObjMap \rfloor)$
    and $\mathfrak{G}(\lfloor ArrMap \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)$
    and $\mathfrak{A} = \mathfrak{C}$
    and $\mathfrak{B} = \mathfrak{D}$
  shows $\mathfrak{G} = \mathfrak{F}$
proof(*rule vsv-eqI*)
  interpret *L*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
  interpret *R*: *is-functor* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
  from *assms*(*1*) **show** *vsv* $\mathfrak{G}$ **by** *auto*
  from *assms*(*2*) **show** *vsv* $\mathfrak{F}$ **by** *auto*
  have *dom*: $\mathcal{D}_{\circ}\ \mathfrak{G} = \mathcal{4}_{\mathbb{N}}$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps V-cs-simps*)
  show $\mathcal{D}_{\circ}\ \mathfrak{G} = \mathcal{D}_{\circ}\ \mathfrak{F}$ **by** (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps V-cs-simps*)
  from *assms*(*5*,*6*) **have** *sup*: $\mathfrak{G}(\lfloor HomDom \rfloor) = \mathfrak{F}(\lfloor HomDom \rfloor)$ $\mathfrak{G}(\lfloor HomCod \rfloor) = \mathfrak{F}(\lfloor HomCod \rfloor)$
    by (*simp-all add*: *cat-cs-simps*)
  show $a \in_{\circ} \mathcal{D}_{\circ}\ \mathfrak{G} \Longrightarrow \mathfrak{G}(\lfloor a \rfloor) = \mathfrak{F}(\lfloor a \rfloor)$ **for** *a*
    by (*unfold dom*, *elim-in-numeral*, *insert assms*(*3*,*4*) *sup*)
      (*auto simp*: *dghm-field-simps*)
qed

lemma *cf-smcf-eqI*:
  assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
    and $\mathfrak{A} = \mathfrak{C}$
    and $\mathfrak{B} = \mathfrak{D}$
    and *cf-smcf* $\mathfrak{G}$ = *cf-smcf* $\mathfrak{F}$
  shows $\mathfrak{G} = \mathfrak{F}$
proof(*rule cf-eqI*)
  from *assms*(*5*) **have**
    cf-smcf $\mathfrak{G}(\lfloor ObjMap \rfloor)$ = *cf-smcf* $\mathfrak{F}(\lfloor ObjMap \rfloor)$
    cf-smcf $\mathfrak{G}(\lfloor ArrMap \rfloor)$ = *cf-smcf* $\mathfrak{F}(\lfloor ArrMap \rfloor)$

```
    by simp-all
  then show 𝔊(⁅ObjMap⁆) = 𝔉(⁅ObjMap⁆) 𝔊(⁅ArrMap⁆) = 𝔉(⁅ArrMap⁆)
    unfolding slicing-simps by simp-all
qed (auto intro: assms(1,2) simp: assms(3–5))

lemma (in is-functor) cf-def: 𝔉 = [𝔉(⁅ObjMap⁆), 𝔉(⁅ArrMap⁆), 𝔉(⁅HomDom⁆), 𝔉(⁅HomCod⁆)]∘
proof(rule vsv-eqI)
  have dom-lhs: 𝒟∘ 𝔉 = 4ℕ
    by (cs-concl cs-shallow cs-simp: cat-cs-simps V-cs-simps)
  have dom-rhs: 𝒟∘ [𝔉(⁅Obj⁆), 𝔉(⁅Arr⁆), 𝔉(⁅Dom⁆), 𝔉(⁅Cod⁆)]∘ = 4ℕ
    by (simp add: nat-omega-simps)
  then show 𝒟∘ 𝔉 = 𝒟∘ [𝔉(⁅ObjMap⁆), 𝔉(⁅ArrMap⁆), 𝔉(⁅HomDom⁆), 𝔉(⁅HomCod⁆)]∘
    unfolding dom-lhs dom-rhs by (simp add: nat-omega-simps)
  show a ∈∘ 𝒟∘ 𝔉 ⟹ 𝔉(⁅a⁆) = [𝔉(⁅ObjMap⁆), 𝔉(⁅ArrMap⁆), 𝔉(⁅HomDom⁆), 𝔉(⁅HomCod⁆)]∘(⁅a⁆)
    for a
    by (unfold dom-lhs, elim-in-numeral, unfold dghm-field-simps)
      (simp-all add: nat-omega-simps)
qed (auto simp: vsv-axioms)
```

Size.

```
lemma (in is-functor) cf-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔉 ∈∘ Vset β
proof–
  interpret β: 𝒵 β by (rule assms(1))
  note [cat-cs-intros] =
    cf-ObjMap-in-Vset
    cf-ArrMap-in-Vset
    HomDom.cat-in-Vset
    HomCod.cat-in-Vset
  from assms(2) show ?thesis
    by (subst cf-def)
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros
      )
qed

lemma (in is-functor) cf-is-functor-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔉 : 𝔄 ↦↦_{C β} 𝔅
  by (rule is-functorI)
    (
      auto simp:
        cat-cs-simps
        assms
        vfsequence-axioms
        cf-is-semifunctor-if-ge-Limit
        HomDom.cat-category-if-ge-Limit
        HomCod.cat-category-if-ge-Limit
        intro: cat-cs-intros
    )

lemma small-all-cfs[simp]: small {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C α} 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-functor.cf-in-Vset show ?thesis
    by (intro down[of - ‹Vset (α + ω)›])
```

```
    (auto simp: True 𝒵.𝒵-Limit-αω 𝒵.𝒵-ω-αω 𝒵.intro 𝒵.𝒵-α-αω)
next
  case False
  then have {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{Cα} 𝔅} = {} by auto
  then show ?thesis by simp
qed

lemma (in is-functor) cf-in-Vset-𝒵: 𝔉 ∈∘ Vset (α + 𝒵ₙ)
proof–
  note [folded VPow-iff, folded Vset-succ[OF Ord-α], cat-cs-intros] =
    cf-ObjMap-vsubset-Vset
    cf-ArrMap-vsubset-Vset
  from HomDom.cat-category-in-Vset-4 have [cat-cs-intros]:
    𝔄 ∈∘ Vset (succ (succ (succ (succ α))))
    by (succ-of-numeral)
      (cs-prems cs-shallow cs-simp: plus-V-succ-right V-cs-simps)
  from HomCod.cat-category-in-Vset-4 have [cat-cs-intros]:
    𝔅 ∈∘ Vset (succ (succ (succ (succ α))))
    by (succ-of-numeral)
      (cs-prems cs-shallow cs-simp: plus-V-succ-right V-cs-simps)
  show ?thesis
    by (subst cf-def, succ-of-numeral)
      (
        cs-concl
          cs-simp: plus-V-succ-right V-cs-simps cat-cs-simps
          cs-intro: cat-cs-intros V-cs-intros
      )
qed

lemma (in 𝒵) all-cfs-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows all-cfs α ∈∘ Vset β
proof(rule vsubset-in-VsetI)
  interpret β: 𝒵 β by (rule assms(1))
  show all-cfs α ⊆∘ Vset (α + 𝒵ₙ)
  proof(intro vsubsetI)
    fix 𝔉 assume 𝔉 ∈∘ all-cfs α
    then obtain 𝔄 𝔅 where 𝔉: 𝔉 : 𝔄 ↦↦_{Cα} 𝔅 by clarsimp
    interpret is-functor α 𝔄 𝔅 𝔉 using 𝔉 by simp
    show 𝔉 ∈∘ Vset (α + 𝒵ₙ) by (rule cf-in-Vset-𝒵)
  qed
  from assms(2) show Vset (α + 𝒵ₙ) ∈∘ Vset β
    by (cs-concl cs-shallow cs-intro: V-cs-intros Ord-cs-intros)
qed

lemma small-cfs[simp]: small {𝔉. 𝔉 : 𝔄 ↦↦_{Cα} 𝔅}
  by (rule down[of - ‹set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{Cα} 𝔅}›]) auto
```

4.2.1 Further properties

```
lemma (in is-functor) cf-ArrMap-is-iso-arr:
  assumes f : a ↦_{iso 𝔄} b
  shows 𝔉(|ArrMap|)(|f|) : 𝔉(|ObjMap|)(|a|) ↦_{iso 𝔅} 𝔉(|ObjMap|)(|b|)
proof–

  note f = is-iso-arrD(1)[OF assms(1)]
  note HomDom.cat-the-inverse-is-iso-arr[OF assms]
  note inv-f = this is-iso-arrD(1)[OF this]
```

show *?thesis*
proof(*intro is-iso-arrI is-inverseI*)
from *inv-f*(*2*) **show** $\mathfrak{F}$-*inv-f*:
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f^{-1}{}_{C\mathfrak{A}} \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
note *cf-ArrMap-Comp is-functor.cf-ArrMap-Comp*[*cat-cs-simps del*]
from *assms f*(*1*) *inv-f* **show**
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f^{-1}{}_{C\mathfrak{A}} \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{B}(\lfloor CId \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)$
$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f^{-1}{}_{C\mathfrak{A}} \rfloor) = \mathfrak{B}(\lfloor CId \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor) \rfloor)$
by
(
cs-concl
cs-simp: *cat-cs-simps cf-ArrMap-Comp*[*symmetric*]
cs-intro: *cat-cs-intros*
)+
qed (*intro cf-ArrMap-is-arr*[*OF f*(*1*)])+

qed

lemma (**in** *is-functor*) *cf-ArrMap-is-iso-arr'*[*cat-arrow-cs-intros*]:
assumes $f : a \mapsto_{iso\mathfrak{A}} b$ **and** $\mathfrak{F}a = \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$ **and** $\mathfrak{F}b = \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
shows $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : \mathfrak{F}a \mapsto_{iso\mathfrak{B}} \mathfrak{F}b$
using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule cf-ArrMap-is-iso-arr*)

lemmas [*cat-arrow-cs-intros*] = *is-functor.cf-ArrMap-is-iso-arr'*

4.3 Opposite functor

4.3.1 Definition and elementary properties

See Chapter II-2 in [7].

definition *op-cf* :: $V \Rightarrow V$
where *op-cf* $\mathfrak{F}$ =
$[\mathfrak{F}(\lfloor ObjMap \rfloor), \mathfrak{F}(\lfloor ArrMap \rfloor), op\text{-}cat\ (\mathfrak{F}(\lfloor HomDom \rfloor)), op\text{-}cat\ (\mathfrak{F}(\lfloor HomCod \rfloor))]_{\circ}$

Components.

lemma *op-cf-components*[*cat-op-simps*]:
shows *op-cf* $\mathfrak{F}(\lfloor ObjMap \rfloor) = \mathfrak{F}(\lfloor ObjMap \rfloor)$
and *op-cf* $\mathfrak{F}(\lfloor ArrMap \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)$
and *op-cf* $\mathfrak{F}(\lfloor HomDom \rfloor) = op\text{-}cat\ (\mathfrak{F}(\lfloor HomDom \rfloor))$
and *op-cf* $\mathfrak{F}(\lfloor HomCod \rfloor) = op\text{-}cat\ (\mathfrak{F}(\lfloor HomCod \rfloor))$
unfolding *op-cf-def dghm-field-simps* **by** (*auto simp*: *nat-omega-simps*)

Slicing.

lemma *cf-smcf-op-cf*[*slicing-commute*]: *op-smcf* (*cf-smcf* $\mathfrak{F}$) = *cf-smcf* (*op-cf* $\mathfrak{F}$)
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_{\circ}$ (*op-smcf* (*cf-smcf* $\mathfrak{F}$)) = $4_{\mathbb{N}}$
unfolding *op-smcf-def* **by** (*auto simp*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_{\circ}$ (*cf-smcf* (*op-cf* $\mathfrak{F}$)) = $4_{\mathbb{N}}$
unfolding *cf-smcf-def* **by** (*auto simp*: *nat-omega-simps*)
show $\mathcal{D}_{\circ}$ (*op-smcf* (*cf-smcf* $\mathfrak{F}$)) = $\mathcal{D}_{\circ}$ (*cf-smcf* (*op-cf* $\mathfrak{F}$))
unfolding *dom-lhs dom-rhs* **by** *simp*
show $a \in_{\circ} \mathcal{D}_{\circ}$ (*op-smcf* (*cf-smcf* $\mathfrak{F}$)) $\Longrightarrow$
op-smcf (*cf-smcf* $\mathfrak{F}$)($\lfloor a \rfloor$) = *cf-smcf* (*op-cf* $\mathfrak{F}$)($\lfloor a \rfloor$)
for *a*
by

(
unfold dom-lhs,
elim-in-numeral,
unfold cf-smcf-def op-cf-def op-smcf-def dghm-field-simps
)
(*auto simp*: *nat-omega-simps slicing-commute*)
qed (*auto simp*: *cf-smcf-def op-smcf-def*)

Elementary properties.

lemma *op-cf-vsv*[*cat-op-intros*]: *vsv* (*op-cf* $\mathfrak{F}$) **unfolding** *op-cf-def* **by** *auto*

4.3.2 Further properties

lemma (**in** *is-functor*) *is-functor-op*: *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{B}$
proof(*intro is-functorI*, *unfold cat-op-simps*)
show *vfsequence* (*op-cf* $\mathfrak{F}$) **unfolding** *op-cf-def* **by** *simp*
show *vcard* (*op-cf* $\mathfrak{F}$) = $4_{\mathbb{N}}$
unfolding *op-cf-def* **by** (*auto simp*: *nat-omega-simps*)
fix *c* **assume** *c* $\in_{\circ}$ $\mathfrak{A}$(| *Obj* |)
then show $\mathfrak{F}$(| *ArrMap* |)(|$\mathfrak{A}$(| *CId* |)(| *c* |)|) = $\mathfrak{B}$(| *CId* |)(|$\mathfrak{F}$(| *ObjMap* |)(| *c* |)|)
unfolding *cat-op-simps* **by** (*auto intro*: *cat-cs-intros*)
qed
(
auto simp:
cat-cs-simps
slicing-commute[*symmetric*]
is-semifunctor.is-semifunctor-op
cf-is-semifunctor
HomCod.category-op
HomDom.category-op
)

lemma (**in** *is-functor*) *is-functor-op'*[*cat-op-intros*]:
assumes $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
shows *op-cf* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}'$
unfolding *assms*(*1*,*2*) **by** (*rule is-functor-op*)

lemmas *is-functor-op*[*cat-op-intros*] = *is-functor.is-functor-op'*

lemma (**in** *is-functor*) *cf-op-cf-op-cf*[*cat-op-simps*]: *op-cf* (*op-cf* $\mathfrak{F}$) = $\mathfrak{F}$
proof(*rule cf-eqI*[*of* α $\mathfrak{A}$ $\mathfrak{B}$ - $\mathfrak{A}$ $\mathfrak{B}$], *unfold cat-op-simps*)
show *op-cf* (*op-cf* $\mathfrak{F}$) : $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by
(
metis
HomCod.cat-op-cat-op-cat
HomDom.cat-op-cat-op-cat
is-functor.is-functor-op
is-functor-op
)
qed (*auto simp*: *cat-cs-intros*)

lemmas *cf-op-cf-op-cf*[*cat-op-simps*] = *is-functor.cf-op-cf-op-cf*

lemma *eq-op-cf-iff*[*cat-op-simps*]:
assumes $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$ **and** $\mathfrak{F}$: $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{D}$
shows *op-cf* $\mathfrak{G}$ = *op-cf* $\mathfrak{F}$ $\longleftrightarrow$ $\mathfrak{G}$ = $\mathfrak{F}$
proof

```
  interpret L: is-functor α 𝔄 𝔅 𝔊 by (rule assms(1))
  interpret R: is-functor α ℭ 𝔇 𝔉 by (rule assms(2))
  assume prems: op-cf 𝔊 = op-cf 𝔉
  show 𝔊 = 𝔉
  proof(rule cf-eqI[OF assms])
    from prems R.cf-op-cf-op-cf L.cf-op-cf-op-cf show
      𝔊(|ObjMap|) = 𝔉(|ObjMap|) 𝔊(|ArrMap|) = 𝔉(|ArrMap|)
      by metis+
    from prems R.cf-op-cf-op-cf L.cf-op-cf-op-cf have
      𝔊(|HomDom|) = 𝔉(|HomDom|) 𝔊(|HomCod|) = 𝔉(|HomCod|)
      by auto
    then show 𝔄 = ℭ 𝔅 = 𝔇 by (simp-all add: cat-cs-simps)
  qed
qed auto
```

4.4 Composition of covariant functors

4.4.1 Definition and elementary properties

```
abbreviation (input) cf-comp :: V ⇒ V ⇒ V (infixl ‹∘CF› 55)
  where cf-comp ≡ dghm-comp
```

Slicing.

```
lemma cf-smcf-smcf-comp[slicing-commute]:
  cf-smcf 𝔊 ∘SMCF cf-smcf 𝔉 = cf-smcf (𝔊 ∘CF 𝔉)
  unfolding dghm-comp-def cf-smcf-def dghm-field-simps
  by (simp add: nat-omega-simps)
```

4.4.2 Object map

```
lemma cf-comp-ObjMap-vsv[cat-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦Cα ℭ and 𝔉 : 𝔄 ↦↦Cα 𝔅
  shows vsv ((𝔊 ∘CF 𝔉)(|ObjMap|))
proof–
  interpret L: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-comp-ObjMap-vsv
          [
            OF L.cf-is-semifunctor R.cf-is-semifunctor,
            unfolded slicing-simps slicing-commute
          ]
      )
qed

lemma cf-comp-ObjMap-vdomain[cat-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦Cα ℭ and 𝔉 : 𝔄 ↦↦Cα 𝔅
  shows 𝒟∘ ((𝔊 ∘CF 𝔉)(|ObjMap|)) = 𝔄(|Obj|)
proof–
  interpret L: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-comp-ObjMap-vdomain
          [
            OF L.cf-is-semifunctor R.cf-is-semifunctor,
```

```
      unfolded slicing-simps slicing-commute
    ]
  )
qed

lemma cf-comp-ObjMap-vrange:
  assumes 𝔊 : 𝔅 ↦↦_{Cα} ℭ and 𝔉 : 𝔄 ↦↦_{Cα} 𝔅
  shows ℛ∘ ((𝔊 ∘_{CF} 𝔉)(⁅ObjMap⁆)) ⊆∘ ℭ(⁅Obj⁆)
proof–
  interpret L: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
      rule smcf-comp-ObjMap-vrange
        [
          OF L.cf-is-semifunctor R.cf-is-semifunctor,
          unfolded slicing-simps slicing-commute
        ]
      )
qed

lemma cf-comp-ObjMap-app[cat-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦_{Cα} ℭ and 𝔉 : 𝔄 ↦↦_{Cα} 𝔅 and [simp]: a ∈∘ 𝔄(⁅Obj⁆)
  shows (𝔊 ∘_{CF} 𝔉)(⁅ObjMap⁆)(⁅a⁆) = 𝔊(⁅ObjMap⁆)(⁅𝔉(⁅ObjMap⁆)(⁅a⁆)⁆)
proof–
  interpret L: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
      rule smcf-comp-ObjMap-app
        [
          OF L.cf-is-semifunctor R.cf-is-semifunctor,
          unfolded slicing-simps slicing-commute,
          OF assms(3)
        ]
      )
qed
```

4.4.3 Arrow map

```
lemma cf-comp-ArrMap-vsv[cat-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦_{Cα} ℭ and 𝔉 : 𝔄 ↦↦_{Cα} 𝔅
  shows vsv ((𝔊 ∘_{CF} 𝔉)(⁅ArrMap⁆))
proof–
  interpret L: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
      rule smcf-comp-ArrMap-vsv
        [
          OF L.cf-is-semifunctor R.cf-is-semifunctor,
          unfolded slicing-simps slicing-commute
        ]
      )
qed
```

lemma *cf-comp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ\ ((\mathfrak{G} \circ_{CF} \mathfrak{F})(\lfloor ArrMap \rfloor)) = \mathfrak{A}(\lfloor Arr \rfloor)$
proof–
interpret *L*: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret *R*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
rule smcf-comp-ArrMap-vdomain
[
OF L.cf-is-semifunctor R.cf-is-semifunctor,
unfolded slicing-simps slicing-commute
]
)
qed

lemma *cf-comp-ArrMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ\ ((\mathfrak{G} \circ_{CF} \mathfrak{F})(\lfloor ArrMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof–
interpret *L*: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret *R*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
rule smcf-comp-ArrMap-vrange
[
OF L.cf-is-semifunctor R.cf-is-semifunctor,
unfolded slicing-simps slicing-commute
]
)
qed

lemma *cf-comp-ArrMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** [*simp*]: $f \in_\circ \mathfrak{A}(\lfloor Arr \rfloor)$
shows $(\mathfrak{G} \circ_{CF} \mathfrak{F})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor)$
proof–
interpret *L*: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret *R*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
rule smcf-comp-ArrMap-app
[
OF L.cf-is-semifunctor R.cf-is-semifunctor,
unfolded slicing-simps slicing-commute,
OF assms(*3*)
]
)
qed

4.4.4 Further properties

lemma *cf-comp-is-functorI*[*cat-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

```
proof–
  interpret L: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
  proof(rule is-functorI, unfold dghm-comp-components(3,4))
    show vfsequence (𝔊 ∘_CF 𝔉) by (simp add: dghm-comp-def)
    show vcard (𝔊 ∘_CF 𝔉) = 4_ℕ
      unfolding dghm-comp-def by (simp add: nat-omega-simps)
    show cf-smcf (𝔊 ∘_CF 𝔉) : cat-smc 𝔄 ↦↦_SMCα cat-smc ℭ
      unfolding cf-smcf-smcf-comp[symmetric]
      by
        (
          cs-concl
            cs-intro: smc-cs-intros slicing-intros cat-cs-intros
        )
    fix c assume c ∈₀ 𝔄(|Obj|)
    with assms show (𝔊 ∘_CF 𝔉)(|ArrMap|)(|𝔄(|CId|)(|c|)|) = ℭ(|CId|)(|(𝔊 ∘_CF 𝔉)(|ObjMap|)(|c|)|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed (auto simp: cat-cs-simps intro: cat-cs-intros)
qed

lemma cf-comp-assoc[cat-cs-simps]:
  assumes ℌ : ℭ ↦↦_Cα 𝔇 and 𝔊 : 𝔅 ↦↦_Cα ℭ and 𝔉 : 𝔄 ↦↦_Cα 𝔅
  shows (ℌ ∘_CF 𝔊) ∘_CF 𝔉 = ℌ ∘_CF (𝔊 ∘_CF 𝔉)
proof(rule cf-eqI[of α 𝔄 𝔇 - 𝔄 𝔇])
  interpret ℌ: is-functor α ℭ 𝔇 ℌ by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  interpret 𝔉: is-functor α 𝔄 𝔅 𝔉 by (rule assms(3))
  from 𝔉.is-functor-axioms 𝔊.is-functor-axioms ℌ.is-functor-axioms
  show ℌ ∘_CF (𝔊 ∘_CF 𝔉) : 𝔄 ↦↦_Cα 𝔇 and ℌ ∘_CF 𝔊 ∘_CF 𝔉 : 𝔄 ↦↦_Cα 𝔇
    by (auto simp: cat-cs-simps intro: cat-cs-intros)
qed (simp-all add: dghm-comp-components vcomp-assoc)
```

The opposite of the covariant composition of functors.

```
lemma op-cf-cf-comp[cat-op-simps]: op-cf (𝔊 ∘_CF 𝔉) = op-cf 𝔊 ∘_CF op-cf 𝔉
  unfolding dghm-comp-def op-cf-def dghm-field-simps
  by (simp add: nat-omega-simps)
```

Composition helper.

```
lemma cf-comp-assoc-helper:
  assumes 𝔉 : 𝔄 ↦↦_Cα 𝔅
    and 𝔊 : 𝔅 ↦↦_Cα ℭ
    and ℌ : ℭ ↦↦_Cα 𝔇
    and ℌ ∘_CF 𝔊 = 𝔔
  shows ℌ ∘_CF (𝔊 ∘_CF 𝔉) = 𝔔 ∘_CF 𝔉
proof–
  interpret 𝔉: is-functor α 𝔄 𝔅 𝔉 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  interpret ℌ: is-functor α ℭ 𝔇 ℌ by (rule assms(3))
  show ?thesis
    using assms(1–3) unfolding assms(4)[symmetric]
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
qed
```

4.5 Composition of contravariant functors

4.5.1 Definition and elementary properties

See section 1.2 in [3].

definition *cf-cn-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** $\langle_{CF\circ}\rangle$ *55*)
  where $\mathfrak{G}\ _{CF\circ}\ \mathfrak{F}$ =
    [
      $\mathfrak{G}(\lfloor ObjMap \rfloor) \circ_\circ \mathfrak{F}(\lfloor ObjMap \rfloor)$,
      $\mathfrak{G}(\lfloor ArrMap \rfloor) \circ_\circ \mathfrak{F}(\lfloor ArrMap \rfloor)$,
      op-cat $(\mathfrak{F}(\lfloor HomDom \rfloor))$,
      $\mathfrak{G}(\lfloor HomCod \rfloor)$
    ]$_\circ$

Components.

lemma *cf-cn-comp-components*:
  shows $(\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})(\lfloor ObjMap \rfloor) = \mathfrak{G}(\lfloor ObjMap \rfloor) \circ_\circ \mathfrak{F}(\lfloor ObjMap \rfloor)$
    and $(\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})(\lfloor ArrMap \rfloor) = \mathfrak{G}(\lfloor ArrMap \rfloor) \circ_\circ \mathfrak{F}(\lfloor ArrMap \rfloor)$
    and [*cat-cn-cs-simps*]: $(\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})(\lfloor HomDom \rfloor) =$ *op-cat* $(\mathfrak{F}(\lfloor HomDom \rfloor))$
    and [*cat-cn-cs-simps*]: $(\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})(\lfloor HomCod \rfloor) = \mathfrak{G}(\lfloor HomCod \rfloor)$
  unfolding *cf-cn-comp-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cf-smcf-cf-cn-comp*[*slicing-commute*]:
  cf-smcf $\mathfrak{G}\ _{SMCF\circ}$ *cf-smcf* $\mathfrak{F}$ = *cf-smcf* $(\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})$
  unfolding *smcf-cn-comp-def cf-cn-comp-def cf-smcf-def*
  by (*simp add*: *nat-omega-simps slicing-commute dghm-field-simps*)

4.5.2 Object map: two contravariant functors

lemma *cf-cn-comp-ObjMap-vsv*[*cat-cn-cs-intros*]:
  assumes $\mathfrak{G} : \mathfrak{B}\ _{C\mapsto\mapsto\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{C\mapsto\mapsto\alpha}\ \mathfrak{B}$
  shows *vsv* $((\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})(\lfloor ObjMap \rfloor))$
proof–
  interpret *L*: *is-functor* α $\langle$*op-cat* $\mathfrak{B}\rangle$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
  interpret *R*: *is-functor* α $\langle$*op-cat* $\mathfrak{A}\rangle$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
  show *?thesis*
    by
      (
        rule smcf-cn-cov-comp-ObjMap-vsv
          [
            OF
              L.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]]
              R.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]],
            unfolded slicing-commute slicing-simps
          ]
)
qed

lemma *cf-cn-comp-ObjMap-vdomain*[*cat-cn-cs-simps*]:
  assumes $\mathfrak{G} : \mathfrak{B}\ _{C\mapsto\mapsto\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{C\mapsto\mapsto\alpha}\ \mathfrak{B}$
  shows $\mathcal{D}_\circ\ ((\mathfrak{G}\ _{CF\circ}\ \mathfrak{F})(\lfloor ObjMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
proof–
  interpret *L*: *is-functor* α $\langle$*op-cat* $\mathfrak{B}\rangle$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
  interpret *R*: *is-functor* α $\langle$*op-cat* $\mathfrak{A}\rangle$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
  show *?thesis*
    by
      (

```
    rule smcf-cn-comp-ObjMap-vdomain
      [
        OF
          L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
          R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
        unfolded slicing-commute slicing-simps
      ]
  )
qed

lemma cf-cn-comp-ObjMap-vrange:
  assumes 𝔊 : 𝔅 CF↦↦α ℭ and 𝔉 : 𝔄 CF↦↦α 𝔅
  shows ℛ∘ ((𝔊 CF∘ 𝔉)(|ObjMap|)) ⊆∘ ℭ(|Obj|)
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
    rule smcf-cn-comp-ObjMap-vrange
      [
        OF
          L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
          R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
        unfolded slicing-commute slicing-simps
      ]
  )
qed

lemma cf-cn-comp-ObjMap-app[cat-cn-cs-simps]:
  assumes 𝔊 : 𝔅 CF↦↦α ℭ and 𝔉 : 𝔄 CF↦↦α 𝔅 and a ∈∘ 𝔄(|Obj|)
  shows (𝔊 CF∘ 𝔉)(|ObjMap|)(|a|) = 𝔊(|ObjMap|)(|𝔉(|ObjMap|)(|a|)|)
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
    rule smcf-cn-comp-ObjMap-app
      [
        OF
          L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
          R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
        unfolded slicing-commute slicing-simps,
        OF assms(3)
      ]
  )
qed
```

4.5.3 Arrow map: two contravariant functors

```
lemma cf-cn-comp-ArrMap-vsv[cat-cn-cs-intros]:
  assumes 𝔊 : 𝔅 CF↦↦α ℭ and 𝔉 : 𝔄 CF↦↦α 𝔅
  shows vsv ((𝔊 CF∘ 𝔉)(|ArrMap|))
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔅 𝔉 by (rule assms(2))
  show ?thesis
```

```
    by
      (
        rule smcf-cn-cov-comp-ArrMap-vsv
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-comp-ArrMap-vdomain[cat-cn-cs-simps]:
  assumes 𝔊 : 𝔙 CF↦↦→α ℭ and 𝔉 : 𝔄 CF↦↦→α 𝔙
  shows 𝒟∘ ((𝔊 CF∘ 𝔉)(|ArrMap|)) = 𝔄(|Arr|)
proof–
  interpret L: is-functor α ‹op-cat 𝔙› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔙 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-comp-ArrMap-vdomain
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-comp-ArrMap-vrange:
  assumes 𝔊 : 𝔙 CF↦↦→α ℭ and 𝔉 : 𝔄 CF↦↦→α 𝔙
  shows ℛ∘ ((𝔊 CF∘ 𝔉)(|ArrMap|)) ⊆∘ ℭ(|Arr|)
proof–
  interpret L: is-functor α ‹op-cat 𝔙› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔙 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-comp-ArrMap-vrange
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-comp-ArrMap-app[cat-cn-cs-simps]:
  assumes 𝔊 : 𝔙 CF↦↦→α ℭ and 𝔉 : 𝔄 CF↦↦→α 𝔙 and a ∈∘ 𝔄(|Arr|)
  shows (𝔊 CF∘ 𝔉)(|ArrMap|)(|a|) = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|a|)|)
proof–
  interpret L: is-functor α ‹op-cat 𝔙› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔙 𝔉 by (rule assms(2))
  show ?thesis
    by
```

(
rule smcf-cn-comp-ArrMap-app
[
OF
L.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]]
R.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]],
unfolded slicing-commute slicing-simps,
OF assms(*3*)
]
)
qed

4.5.4 Object map: contravariant and covariant functor

lemma *cf-cn-cov-comp-ObjMap-vsv*[*cat-cn-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *vsv* $((\mathfrak{G}\ {}_{CF}\circ\ \mathfrak{F})(\!|ObjMap|\!))$
proof–
interpret *L*: *is-functor* α ‹*op-cat* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret *R*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
rule smcf-cn-cov-comp-ObjMap-vsv
[
OF
L.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]]
R.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]],
unfolded slicing-commute slicing-simps
]
)
qed

lemma *cf-cn-cov-comp-ObjMap-vdomain*[*cat-cn-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_{\circ}\ ((\mathfrak{G}\ {}_{CF}\circ\ \mathfrak{F})(\!|ObjMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
proof–
interpret *L*: *is-functor* α ‹*op-cat* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret *R*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
rule smcf-cn-cov-comp-ObjMap-vdomain
[
OF
L.cf-is-semifunctor[*unfolded slicing-commute*[*symmetric*]]
R.cf-is-semifunctor,
unfolded slicing-commute slicing-simps
]
)
qed

lemma *cf-cn-cov-comp-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B}\ {}_{C}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{G}\ {}_{CF}\circ\ \mathfrak{F})(\!|ObjMap|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
proof–
interpret *L*: *is-functor* α ‹*op-cat* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret *R*: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))

```
  show ?thesis
    by
      (
        rule smcf-cn-cov-comp-ObjMap-vrange
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor,
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-cov-comp-ObjMap-app[cat-cn-cs-simps]:
  assumes 𝔊 : 𝔅 CF↦↦→α ℭ and 𝔉 : 𝔄 ↦↦→Cα 𝔅 and a ∈∘ 𝔄(|Obj|)
  shows (𝔊 CF∘ 𝔉)(|ObjMap|)(|a|) = 𝔊(|ObjMap|)(|𝔉(|ObjMap|)(|a|)|)
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-cov-comp-ObjMap-app
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor,
            unfolded slicing-commute slicing-simps,
            OF assms(3)
          ]
      )
qed
```

4.5.5 Arrow map: contravariant and covariant functors

```
lemma cf-cn-cov-comp-ArrMap-vsv[cat-cn-cs-intros]:
  assumes 𝔊 : 𝔅 CF↦↦→α ℭ and 𝔉 : 𝔄 ↦↦→Cα 𝔅
  shows vsv ((𝔊 CF∘ 𝔉)(|ArrMap|))
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-cov-comp-ArrMap-vsv
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor[unfolded slicing-commute[symmetric]],
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-cov-comp-ArrMap-vdomain[cat-cn-cs-simps]:
  assumes 𝔊 : 𝔅 CF↦↦→α ℭ and 𝔉 : 𝔄 ↦↦→Cα 𝔅
  shows 𝒟∘ ((𝔊 CF∘ 𝔉)(|ArrMap|)) = 𝔄(|Arr|)
proof–
```

```
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-cov-comp-ArrMap-vdomain
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor,
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-cov-comp-ArrMap-vrange:
  assumes 𝔊 : 𝔅 C↦↦α ℭ and 𝔉 : 𝔄 ↦↦C α 𝔅
  shows ℛ∘ ((𝔊 C F∘ 𝔉)(|ArrMap|)) ⊆∘ ℭ(|Arr|)
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-cov-comp-ArrMap-vrange
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor,
            unfolded slicing-commute slicing-simps
          ]
      )
qed

lemma cf-cn-cov-comp-ArrMap-app[cat-cn-cs-simps]:
  assumes 𝔊 : 𝔅 C↦↦α ℭ and 𝔉 : 𝔄 ↦↦C α 𝔅 and a ∈∘ 𝔄(|Arr|)
  shows (𝔊 C F∘ 𝔉)(|ArrMap|)(|a|) = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|a|)|)
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule smcf-cn-cov-comp-ArrMap-app
          [
            OF
              L.cf-is-semifunctor[unfolded slicing-commute[symmetric]]
              R.cf-is-semifunctor,
            unfolded slicing-commute slicing-simps,
            OF assms(3)
          ]
      )
qed
```

4.5.6 Further properties

```
lemma cf-cn-comp-is-functorI[cat-cn-cs-intros]:
  assumes category α 𝔄 and 𝔊 : 𝔅 C↦↦α ℭ and 𝔉 : 𝔄 C↦↦α 𝔅
```

```
  shows 𝔊 _CF∘ 𝔉 : 𝔄 ↦↦_Cα ℭ
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(2))
  interpret R: is-functor α ‹op-cat 𝔄› 𝔅 𝔉 by (rule assms(3))
  interpret 𝔄: category α 𝔄 by (rule assms(1))
  show ?thesis
  proof(rule is-functorI, unfold cf-cn-comp-components(3,4) cat-op-simps)
    show vfsequence (𝔊 _CF∘ 𝔉)
      unfolding cf-cn-comp-def by (simp add: nat-omega-simps)
    show vcard (𝔊 _CF∘ 𝔉) = ℶ₄
      unfolding cf-cn-comp-def by (simp add: nat-omega-simps)
    from assms(1) L.cf-is-semifunctor R.cf-is-semifunctor show
      cf-smcf (𝔊 _CF∘ 𝔉) : cat-smc 𝔄 ↦↦_SMCα cat-smc ℭ
      unfolding cf-smcf-cf-cn-comp[symmetric]
      by
        (
          cs-concl cs-shallow
            cs-intro: cat-cs-intros slicing-intros smc-cn-cs-intros
        )
    fix c assume c ∈₀ 𝔄(|Obj|)
    with assms show
      (𝔊 _CF∘ 𝔉)(|ArrMap|)(|𝔄(|CId|)(|c|)|) = ℭ(|CId|)(|(𝔊 _CF∘ 𝔉)(|ObjMap|)(|c|)|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-op-simps cat-cn-cs-simps cs-intro: cat-cs-intros
        )
  qed (auto simp: cat-cs-simps cat-cs-intros cat-op-simps)
qed
```

See section 1.2 in [3]).

```
lemma cf-cn-cov-comp-is-functor[cat-cn-cs-intros]:
  assumes 𝔊 : 𝔅 _C↦↦_α ℭ and 𝔉 : 𝔄 ↦↦_Cα 𝔅
  shows 𝔊 _CF∘ 𝔉 : 𝔄 _C↦↦_α ℭ
proof–
  interpret L: is-functor α ‹op-cat 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-functor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
  proof
    (
      rule is-functorI,
      unfold cf-cn-comp-components(3,4) cat-op-simps slicing-commute[symmetric]
    )
    show vfsequence (𝔊 _CF∘ 𝔉) unfolding cf-cn-comp-def by simp
    show vcard (𝔊 _CF∘ 𝔉) = ℶ₄
      unfolding cf-cn-comp-def by (auto simp: nat-omega-simps)
    from L.cf-is-semifunctor show
      cf-smcf 𝔊 _SMCF∘ cf-smcf 𝔉 : op-smc (cat-smc 𝔄) ↦↦_SMCα cat-smc ℭ
      by
        (
          cs-concl cs-shallow
            cs-intro: cat-cs-intros slicing-intros smc-cn-cs-intros
        )
    fix c assume c ∈₀ 𝔄(|Obj|)
    with assms show (𝔊 _CF∘ 𝔉)(|ArrMap|)(|𝔄(|CId|)(|c|)|) = ℭ(|CId|)(|(𝔊 _CF∘ 𝔉)(|ObjMap|)(|c|)|)
      by
        (
          cs-concl
```

cs-simp: *cat-cs-simps cat-cn-cs-simps*
cs-intro: *cat-cs-intros*
)
qed (*auto simp*: *cat-cs-simps cat-cs-intros*)
qed

See section 1.2 in [3].

lemma *cf-cov-cn-comp-is-functor*[*cat-cn-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ {}_{C}\mapsto\mapsto_{\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A}\ {}_{C}\mapsto\mapsto_{\alpha} \mathfrak{C}$
using *assms* **by** (*rule cf-comp-is-functorI*)

The opposite of the contravariant composition of functors.

lemma *op-cf-cf-cn-comp*[*cat-op-simps*]: *op-cf* $(\mathfrak{G}\ {}_{CF}\circ \mathfrak{F})$ = *op-cf* $\mathfrak{G}\ {}_{CF}\circ$ *op-cf* $\mathfrak{F}$
unfolding *op-cf-def cf-cn-comp-def dghm-field-simps*
by (*auto simp*: *nat-omega-simps*)

4.6 Identity functor

4.6.1 Definition and elementary properties

See Chapter I-3 in [7].

abbreviation (*input*) *cf-id* :: $V \Rightarrow V$ **where** *cf-id* ≡ *dghm-id*

Slicing.

lemma *cf-smcf-cf-id*[*slicing-commute*]: *smcf-id* (*cat-smc* $\mathfrak{C}$) = *cf-smcf* (*cf-id* $\mathfrak{C}$)
unfolding *dghm-id-def cat-smc-def cf-smcf-def dghm-field-simps dg-field-simps*
by (*simp add*: *nat-omega-simps*)

context *category*
begin

interpretation *smc*: *semicategory* α ‹*cat-smc* $\mathfrak{C}$› **by** (*rule cat-semicategory*)

lemmas-with [*unfolded slicing-simps*]:
cat-smcf-id-is-semifunctor = *smc.smc-smcf-id-is-semifunctor*

end

4.6.2 Object map

lemmas [*cat-cs-simps*] = *dghm-id-ObjMap-app*

4.6.3 Arrow map

lemmas [*cat-cs-simps*] = *dghm-id-ArrMap-app*

4.6.4 Opposite of an identity functor.

lemma *op-cf-cf-id*[*cat-op-simps*]: *op-cf* (*cf-id* $\mathfrak{C}$) = *cf-id* (*op-cat* $\mathfrak{C}$)
unfolding *dghm-id-def op-cat-def op-cf-def dghm-field-simps dg-field-simps*
by (*auto simp*: *nat-omega-simps*)

4.6.5 An identity functor is a functor

lemma (**in** *category*) *cat-cf-id-is-functor*: *cf-id* $\mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*rule is-functorI*, *unfold dghm-id-components*)
from *cat-smcf-id-is-semifunctor* **show**

cf-smcf (*cf-id* $\mathfrak{C}$) : *cat-smc* $\mathfrak{C}$ $\mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{C}$
by (*simp add*: *slicing-commute*)
from *cat-CId-is-arr* **show**
$c \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow$ *vid-on* $(\mathfrak{C}(\!|Arr|\!))(\!|\mathfrak{C}(\!|CId|\!)(\!|c|\!)|\!) = \mathfrak{C}(\!|CId|\!)(\!|$*vid-on* $(\mathfrak{C}(\!|Obj|\!))(\!|c|\!)|\!)$
for c
by *auto*
qed (*auto simp*: *dghm-id-def nat-omega-simps cat-cs-intros*)

lemma (**in** *category*) *cat-cf-id-is-functor'*:
assumes $\mathfrak{A} = \mathfrak{C}$ **and** $\mathfrak{B} = \mathfrak{C}$
shows *cf-id* $\mathfrak{C} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule cat-cf-id-is-functor*)

lemmas [*cat-cs-intros*] = *category.cat-cf-id-is-functor'*

4.6.6 Further properties

lemma (**in** *is-functor*) *cf-cf-comp-cf-id-left*[*cat-cs-simps*]: *cf-id* $\mathfrak{B} \circ_{CF} \mathfrak{F} = \mathfrak{F}$
— See Chapter I-3 in [7]).
by
(
rule cf-eqI,
unfold dghm-id-components dghm-comp-components dghm-id-components
)
(*auto intro*: *cat-cs-intros simp*: *cf-ArrMap-vrange cf-ObjMap-vrange*)

lemmas [*cat-cs-simps*] = *is-functor.cf-cf-comp-cf-id-left*

lemma (**in** *is-functor*) *cf-cf-comp-cf-id-right*[*cat-cs-simps*]: $\mathfrak{F} \circ_{CF}$ *cf-id* $\mathfrak{A} = \mathfrak{F}$
— See Chapter I-3 in [7]).
by
(
rule cf-eqI,
unfold dghm-id-components dghm-comp-components dghm-id-components
)
(
auto
intro: *cat-cs-intros*
simp: *cat-cs-simps cf-ArrMap-vrange cf-ObjMap-vrange*
)

lemmas [*cat-cs-simps*] = *is-functor.cf-cf-comp-cf-id-right*

4.7 Constant functor

4.7.1 Definition and elementary properties

See Chapter III-3 in [7].

abbreviation *cf-const* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-const* $\mathfrak{C}$ $\mathfrak{D}$ $a \equiv$ *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $(\mathfrak{D}(\!|CId|\!)(\!|a|\!))$

Slicing.

lemma *cf-smcf-cf-const*[*slicing-commute*]:
smcf-const (*cat-smc* $\mathfrak{C}$) (*cat-smc* $\mathfrak{D}$) a $(\mathfrak{D}(\!|CId|\!)(\!|a|\!)) =$ *cf-smcf* (*cf-const* $\mathfrak{C}$ $\mathfrak{D}$ a)
unfolding
dghm-const-def cat-smc-def cf-smcf-def dghm-field-simps dg-field-simps
by (*simp add*: *nat-omega-simps*)

4.7.2 Object map and arrow map

context
  fixes $\mathfrak{D}$ a :: V
begin

lemmas-with [**where** $\mathfrak{D}$=$\mathfrak{D}$ **and** a=a **and** f=‹$\mathfrak{D}$(|CId|)(|a|)›, $cat\text{-}cs\text{-}simps$]:
  $dghm\text{-}const\text{-}ObjMap\text{-}app$
  $dghm\text{-}const\text{-}ArrMap\text{-}app$

end

4.7.3 Opposite constant functor

lemma $op\text{-}cf\text{-}cf\text{-}const$[$cat\text{-}op\text{-}simps$]:
  $op\text{-}cf$ ($cf\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a) = $cf\text{-}const$ ($op\text{-}cat$ $\mathfrak{C}$) ($op\text{-}cat$ $\mathfrak{D}$) a
  unfolding $dghm\text{-}const\text{-}def$ $op\text{-}cat\text{-}def$ $op\text{-}cf\text{-}def$ $dghm\text{-}field\text{-}simps$ $dg\text{-}field\text{-}simps$
  by ($auto$ $simp$: $nat\text{-}omega\text{-}simps$)

4.7.4 A constant functor is a functor

lemma $cf\text{-}const\text{-}is\text{-}functor$:
  assumes $category$ α $\mathfrak{C}$ **and** $category$ α $\mathfrak{D}$ **and** a $\in_{\circ}$ $\mathfrak{D}$(|Obj|)
  shows $cf\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a : $\mathfrak{C}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{D}$
proof–
  interpret $\mathfrak{C}$: $category$ α $\mathfrak{C}$ **by** ($rule$ $assms(1)$)
  interpret $\mathfrak{D}$: $category$ α $\mathfrak{D}$ **by** ($rule$ $assms(2)$)
  show $?thesis$
  proof($intro$ $is\text{-}functorI$, $tactic$‹$distinct\text{-}subgoals\text{-}tac$›)
    show $vfsequence$ ($dghm\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a ($\mathfrak{D}$(|CId|)(|a|)))
      unfolding $dghm\text{-}const\text{-}def$ **by** $simp$
    show $vcard$ ($cf\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a) = $4_{\mathbb{N}}$
      unfolding $dghm\text{-}const\text{-}def$ **by** ($simp$ add: $nat\text{-}omega\text{-}simps$)
    from $assms$ **show** $cf\text{-}smcf$ ($cf\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a) : $cat\text{-}smc$ $\mathfrak{C}$ $\mapsto\!\!\mapsto_{SMC\alpha}$ $cat\text{-}smc$ $\mathfrak{D}$
      by
        (
          $cs\text{-}concl$ **cs-shallow**
            cs-simp: $cat\text{-}cs\text{-}simps$ $slicing\text{-}simps$ $slicing\text{-}commute$[$symmetric$]
            cs-intro: $smc\text{-}cs\text{-}intros$ $cat\text{-}cs\text{-}intros$ $slicing\text{-}intros$
)
    fix c **assume** c $\in_{\circ}$ $\mathfrak{C}$(|Obj|)
    with $assms$ **show**
      $cf\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a(|$ArrMap$|)(|$\mathfrak{C}$(|CId|)(|c|)|) = $\mathfrak{D}$(|CId|)(|$cf\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a(|$ObjMap$|)(|c|)|)
      by ($cs\text{-}concl$ **cs-shallow** **cs-simp**: $cat\text{-}cs\text{-}simps$ **cs-intro**: $cat\text{-}cs\text{-}intros$)
  qed ($auto$ $simp$: $dghm\text{-}const\text{-}components$ $assms$)
qed

lemma $cf\text{-}const\text{-}is\text{-}functor'$[$cat\text{-}cs\text{-}intros$]:
  assumes $category$ α $\mathfrak{C}$
    and $category$ α $\mathfrak{D}$
    and a $\in_{\circ}$ $\mathfrak{D}$(|Obj|)
    and $\mathfrak{A}$ = $\mathfrak{C}$
    and $\mathfrak{B}$ = $\mathfrak{D}$
    and f = ($\mathfrak{D}$(|CId|)(|a|))
  shows $dghm\text{-}const$ $\mathfrak{C}$ $\mathfrak{D}$ a f : $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
  using $assms(1\text{–}3)$ **unfolding** $assms(4\text{–}6)$ **by** ($rule$ $cf\text{-}const\text{-}is\text{-}functor$)

4.7.5 Further properties

lemma *cf-comp-cf-const-right*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ b = *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))$
proof(*rule cf-eqI*)

interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))

from *assms*(*3*) **show** $\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
from *assms*(*3*) **show** *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)) : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms*(*3*) **have** *ObjMap-dom-lhs*:
$\mathcal{D}_\circ$ $((\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b)(\!|ObjMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms*(*3*) **have** *ObjMap-dom-rhs*:
$\mathcal{D}_\circ$ (*cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|ObjMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
show $(\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b)(\!|ObjMap|\!)$ = *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|ObjMap|\!)$
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
fix a **assume** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
with *assms*(*3*) **show** $(\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b)(\!|ObjMap|\!)(\!|a|\!)$ =
cf-const $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|ObjMap|\!)(\!|a|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto intro*: *assms*(*3*) *cat-cs-intros*)
from *assms*(*3*) **have** *ArrMap-dom-lhs*:
$\mathcal{D}_\circ$ $((\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b)(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms*(*3*) **have** *ArrMap-dom-rhs*:
$\mathcal{D}_\circ$ (*cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $(\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b)(\!|ArrMap|\!)$ = *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|ArrMap|\!)$
proof(*rule vsv-eqI*, *unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
fix a **assume** $a \in_\circ \mathfrak{A}(\!|Arr|\!)$
with *assms*(*3*) **show** $(\mathfrak{G} \circ_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{B}$ $b)(\!|ArrMap|\!)(\!|a|\!)$ =
cf-const $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|ArrMap|\!)(\!|a|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto intro*: *assms*(*3*) *cat-cs-intros*)

qed *simp-all*

lemma *cf-comp-cf-const-right'*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $f = \mathfrak{B}(\!|CId|\!)(\!|b|\!)$
shows $\mathfrak{G} \circ_{CF}$ *dghm-const* $\mathfrak{A}$ $\mathfrak{B}$ b f = *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-comp-cf-const-right*)

lemma (**in** *is-functor*) *cf-comp-cf-const-left*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$ **and** $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *cf-const* $\mathfrak{B}$ $\mathfrak{C}$ $a \circ_{CF} \mathfrak{F}$ = *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ a
proof(*rule cf-smcf-eqI*)
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))

from *assms*(*2*) **show** *cf-const* $\mathfrak{B}$ $\mathfrak{C}$ *a* $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms*(*2*) **show** *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ *a* : $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms*(*2*) **have** *CId-a*: $\mathfrak{C}(\!|CId|\!)(\!|a|\!)$: *a* $\mapsto_{\mathfrak{C}}$ *a*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms*(*2*) **have** *CId-CId-a*: $\mathfrak{C}(\!|CId|\!)(\!|a|\!)$ $\circ_{A\mathfrak{C}}$ $\mathfrak{C}(\!|CId|\!)(\!|a|\!)$ = $\mathfrak{C}(\!|CId|\!)(\!|a|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from
is-semifunctor.smcf-smcf-comp-smcf-const
[
OF cf-is-semifunctor $\mathfrak{C}$.*cat-semicategory*,
unfolded slicing-simps,
OF CId-a CId-CId-a
]
show *cf-smcf* (*cf-const* $\mathfrak{B}$ $\mathfrak{C}$ *a* $\circ_{DGHM}$ $\mathfrak{F}$) = *cf-smcf* (*cf-const* $\mathfrak{A}$ $\mathfrak{C}$ *a*)
by (*cs-prems* **cs-shallow cs-simp**: *slicing-simps slicing-commute*)
qed *simp-all*

lemma (**in** *is-functor*) *cf-comp-cf-const-left'*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$
and *a* $\in_{\circ}$ $\mathfrak{C}(\!|Obj|\!)$
and *f* = $\mathfrak{C}(\!|CId|\!)(\!|a|\!)$
shows *dghm-const* $\mathfrak{B}$ $\mathfrak{C}$ *a f* $\circ_{CF}$ $\mathfrak{F}$ = *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ *a*
using *assms*(*1*,*2*) **unfolding** *assms*(*3*) **by** (*rule cf-comp-cf-const-left*)

lemmas [*cat-cs-simps*] = *is-functor.cf-comp-cf-const-left'*

4.8 Faithful functor

4.8.1 Definition and elementary properties

See Chapter I-3 in [7]).

locale *is-ft-functor* = *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *ft-cf-is-ft-semifunctor*[*slicing-intros*]:
cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.faithful\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-ft-functor* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
(‹(- :/ - $\mapsto\mapsto_{C.faithful^{!}}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-ft-functor* ⇌ *is-ft-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.faithful\alpha}$ $\mathfrak{B}$ ⇌ *CONST is-ft-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

lemma (**in** *is-ft-functor*) *ft-cf-is-ft-functor'*:
assumes $\mathfrak{A}'$ = *cat-smc* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
shows *cf-smcf* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{SMC.faithful\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule ft-cf-is-ft-semifunctor*)

lemmas [*slicing-intros*] = *is-ft-functor.ft-cf-is-ft-functor'*

Rules.

lemma (**in** *is-ft-functor*) *is-ft-functor-axioms'*[*cf-cs-intros*]:
assumes α' = α **and** $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$ = $\mathfrak{B}$
shows $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{C.faithful_{\alpha'}}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule is-ft-functor-axioms*)

mk-ide rf *is-ft-functor-def*[*unfolded is-ft-functor-axioms-def*]
|*intro is-ft-functorI*|
|*dest is-ft-functorD*[*dest*]|

|*elim is-ft-functorE*[*elim*]|

lemmas [*cf-cs-intros*] = *is-ft-functorD*(*1*)

lemma *is-ft-functorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow v11\ (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.faithful\alpha} \mathfrak{B}$
using *assms*
by (*intro is-ft-functorI*)
(
simp-all add:
assms(*1*)
is-ft-semifunctorI'[*OF is-functorD*(*6*)[
OF assms(*1*)], *unfolded slicing-simps*
]
)

lemma *is-ft-functorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.faithful\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow v11\ (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
by
(
simp-all add:
is-ft-functorD[*OF assms*(*1*)]
is-ft-semifunctorD'(*2*)[
OF is-ft-functorD(*2*)[*OF assms*(*1*)], *unfolded slicing-simps*
]
)

lemma *is-ft-functorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.faithful\alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow v11\ (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
using *assms* **by** (*simp-all add*: *is-ft-functorD'*)

lemma *is-ft-functorI''*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b\ g\ f.$
$[\![\ g : a \mapsto_{\mathfrak{A}} b;\ f : a \mapsto_{\mathfrak{A}} b;\ \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)\ ]\!] \Longrightarrow g = f$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.faithful\alpha} \mathfrak{B}$
by
(
intro is-ft-functorI assms,
rule is-ft-semifunctorI'',
unfold slicing-simps,
rule is-functor.cf-is-semifunctor[*OF assms*(*1*)],
rule assms(*2*)
)

Elementary properties.

context *is-ft-functor*
begin

interpretation *smcf*: *is-ft-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule ft-cf-is-ft-semifunctor*)

lemmas-with [*unfolded slicing-simps*]:
ft-cf-v11-on-Hom = *smcf.ft-smcf-v11-on-Hom*
and *ft-cf-ArrMap-eqD* = *smcf.ft-smcf-ArrMap-eqD*

end

4.8.2 Opposite faithful functor.

lemma (**in** *is-ft-functor*) *is-ft-functor-op'*:
op-cf $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C.faithful\alpha}$ *op-cat* $\mathfrak{B}$
by (*rule is-ft-functorI*, *unfold slicing-commute*[*symmetric*])
(
simp-all add:
is-functor-op is-ft-semifunctor.is-ft-semifunctor-op
ft-cf-is-ft-semifunctor
)

lemma (**in** *is-ft-functor*) *is-ft-functor-op*:
assumes $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
shows *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C.faithful\alpha}$ *op-cat* $\mathfrak{B}$
unfolding *assms* **by** (*rule is-ft-functor-op'*)

lemmas *is-ft-functor-op*[*cat-op-intros*] = *is-ft-functor.is-ft-functor-op'*

4.8.3 The composition of faithful functors is a faithful functor

lemma *cf-comp-is-ft-functor*[*cf-cs-intros*]:
assumes $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\mapsto_{C.faithful\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.faithful\alpha}$ $\mathfrak{B}$
shows $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.faithful\alpha}$ $\mathfrak{C}$
proof(*intro is-ft-functorI*)
interpret $\mathfrak{G}$: *is-ft-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*simp add*: *assms*(*1*))
interpret $\mathfrak{F}$: *is-ft-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*simp add*: *assms*(*2*))
from $\mathfrak{F}$.*is-functor-axioms* $\mathfrak{G}$.*is-functor-axioms* **show** $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
then interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ ‹$\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$› .
show *cf-smcf* ($\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$) : *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.faithful\alpha}$ *cat-smc* $\mathfrak{C}$
by
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *cf-cs-intros smcf-cs-intros slicing-intros*
)
qed

4.9 Full functor

4.9.1 Definition and elementary properties

See Chapter I-3 in [7]).

locale *is-fl-functor* = *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *fl-cf-is-fl-semifunctor*:
cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.full\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-fl-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\mapsto\mapsto_{C.full1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-fl-functor* $\rightleftharpoons$ *is-fl-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.full\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$ *CONST is-fl-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

lemma (in *is-fl-functor*) *fl-cf-is-fl-functor'*[*slicing-intros*]:
assumes $\mathfrak{A}' = \textit{cat-smc}\ \mathfrak{A}$ **and** $\mathfrak{B}' = \textit{cat-smc}\ \mathfrak{B}$
shows $\textit{cf-smcf}\ \mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{SMC.full\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule fl-cf-is-fl-semifunctor*)

lemmas [*slicing-intros*] = *is-fl-functor.fl-cf-is-fl-semifunctor*

Rules.

lemma (in *is-fl-functor*) *is-fl-functor-axioms'*[*cf-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C.full_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-fl-functor-axioms*)

mk-ide rf *is-fl-functor-def*[*unfolded is-fl-functor-axioms-def*]
|*intro is-fl-functorI*|
|*dest is-fl-functorD*[*dest*]|
|*elim is-fl-functorE*[*elim*]|

lemmas [*cf-cs-intros*] = *is-fl-functorD*(*1*)

lemma *is-fl-functorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)\ \text{`}_\circ\ (Hom\ \mathfrak{A}\ a\ b) = Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.full\alpha} \mathfrak{B}$
using *assms*
by (*intro is-fl-functorI*)
(
simp-all add:
assms(*1*)
is-fl-semifunctorI'[
OF is-functorD(*6*)[*OF assms*(*1*)], *unfolded slicing-simps*
]
)

lemma *is-fl-functorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.full\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)\ \text{`}_\circ\ (Hom\ \mathfrak{A}\ a\ b) = Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
by
(
simp-all add:
is-fl-functorD[*OF assms*(*1*)]
is-fl-semifunctorD'(*2*)[
OF is-fl-functorD(*2*)[*OF assms*(*1*)], *unfolded slicing-simps*
]
)

lemma *is-fl-functorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.full\alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)\ \text{`}_\circ\ (Hom\ \mathfrak{A}\ a\ b) = Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
using *assms* **by** (*simp-all add*: *is-fl-functorD'*)

Elementary properties.

context *is-fl-functor*

begin

interpretation *smcf*: *is-fl-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule fl-cf-is-fl-semifunctor*)

lemmas-with [*unfolded slicing-simps*]:
fl-cf-surj-on-Hom = *smcf.fl-smcf-surj-on-Hom*

end

4.9.2 Opposite full functor

lemma (**in** *is-fl-functor*) *is-fl-functor-op*[*cat-op-intros*]:
op-cf $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C.full\alpha}$ *op-cat* $\mathfrak{B}$
by (*rule is-fl-functorI*, *unfold slicing-commute*[*symmetric*])
(*simp-all add*: *cat-op-intros smc-op-intros slicing-intros*)

lemmas *is-fl-functor-op*[*cat-op-intros*] = *is-fl-functor.is-fl-functor-op*

4.9.3 The composition of full functor is a full functor

lemma *cf-comp-is-fl-functor*[*cf-cs-intros*]:
assumes $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\mapsto_{C.full\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.full\alpha}$ $\mathfrak{B}$
shows $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.full\alpha}$ $\mathfrak{C}$
proof(*intro is-fl-functorI*)
interpret $\mathfrak{F}$: *is-fl-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **using** *assms*(*2*) **by** *simp*
interpret $\mathfrak{G}$: *is-fl-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **using** *assms*(*1*) **by** *simp*
from $\mathfrak{F}$.*is-functor-axioms* $\mathfrak{G}$.*is-functor-axioms* **show** $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show *cf-smcf* ($\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$) : *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.full\alpha}$ *cat-smc* $\mathfrak{C}$
by
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *cf-cs-intros smcf-cs-intros slicing-intros*
)
qed

4.10 Fully faithful functor

4.10.1 Definition and elementary properties

See Chapter I-3 in [7]).

locale *is-ff-functor* = *is-ft-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ + *is-fl-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$
for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

syntax *-is-ff-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\mapsto\mapsto_{C.ff1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-ff-functor* $\rightleftharpoons$ *is-ff-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.ff\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$ *CONST is-ff-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

Rules.

mk-ide rf *is-ff-functor-def*
|*intro is-ff-functorI*|
|*dest is-ff-functorD*[*dest*]|
|*elim is-ff-functorE*[*elim*]|

lemmas [*cf-cs-intros*] = *is-ff-functorD*

Elementary properties.

lemma (**in** *is-ff-functor*) *ff-cf-is-ff-semifunctor*:
cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.ff\alpha}$ *cat-smc* $\mathfrak{B}$
by (*rule is-ff-semifunctorI*) (*auto intro*: *slicing-intros*)

lemma (**in** *is-ff-functor*) *ff-cf-is-ff-semifunctor'*[*slicing-intros*]:
assumes $\mathfrak{A}'$ = *cat-smc* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
shows *cf-smcf* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{SMC.ff\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule ff-cf-is-ff-semifunctor*)

lemmas [*slicing-intros*] = *is-ff-functor.ff-cf-is-ff-semifunctor'*

4.10.2 Opposite fully faithful functor

lemma (**in** *is-ff-functor*) *is-ff-functor-op*:
op-cf $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.ff\alpha}$ *op-cat* $\mathfrak{B}$
by (*rule is-ff-functorI*) (*auto simp*: *is-fl-functor-op is-ft-functor-op*)

lemma (**in** *is-ff-functor*) *is-ff-functor-op'*[*cat-op-intros*]:
assumes $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
shows *op-cf* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{C.ff\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule is-ff-functor-op*)

lemmas *is-ff-functor-op*[*cat-op-intros*] = *is-ff-functor.is-ff-functor-op*

4.10.3 The composition of fully faithful functors is a fully faithful functor

lemma *cf-comp-is-ff-functor*[*cf-cs-intros*]:
assumes $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C.ff\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.ff\alpha}$ $\mathfrak{B}$
shows $\mathfrak{G}$ $\circ_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.ff\alpha}$ $\mathfrak{C}$
using *assms*
by (*intro is-ff-functorI*, *elim is-ff-functorE*) (*auto simp*: *cf-cs-intros*)

4.11 Isomorphism of categories

4.11.1 Definition and elementary properties

See Chapter I-3 in [7]).

locale *is-iso-functor* = *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *iso-cf-is-iso-semifunctor*:
cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.iso\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-iso-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\mapsto\!\!\mapsto_{C.iso^1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-iso-functor* ⇌ *is-iso-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.iso\alpha}$ $\mathfrak{B}$ ⇌ *CONST is-iso-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

lemma (**in** *is-iso-functor*) *iso-cf-is-iso-semifunctor'*[*slicing-intros*]:
assumes $\mathfrak{A}'$ = *cat-smc* $\mathfrak{A}$ $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
shows *cf-smcf* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{SMC.iso\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule iso-cf-is-iso-semifunctor*)

lemmas [*slicing-intros*] = *is-iso-semifunctor.iso-smcf-is-iso-dghm'*

Rules.

lemma (**in** *is-iso-functor*) *is-iso-functor-axioms'*[*cf-cs-intros*]:
assumes α' = α **and** $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$ = $\mathfrak{B}$

shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C.iso_\alpha'} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-iso-functor-axioms*)

mk-ide rf *is-iso-functor-def*[*unfolded is-iso-functor-axioms-def*]
|*intro is-iso-functorI*|
|*dest is-iso-functorD*[*dest*]|
|*elim is-iso-functorE*[*elim*]|

lemma *is-iso-functorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *v11* ($\mathfrak{F}(\lfloor ObjMap \rfloor)$)
and *v11* ($\mathfrak{F}(\lfloor ArrMap \rfloor)$)
and $\mathcal{R}_\circ$ ($\mathfrak{F}(\lfloor ObjMap \rfloor)$) = $\mathfrak{B}(\lfloor Obj \rfloor)$
and $\mathcal{R}_\circ$ ($\mathfrak{F}(\lfloor ArrMap \rfloor)$) = $\mathfrak{B}(\lfloor Arr \rfloor)$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
using *assms*
by (*intro is-iso-functorI*)
(
simp-all add:
assms(*1*)
is-iso-semifunctorI'[
OF is-functorD(*6*)[*OF assms*(*1*)], *unfolded slicing-simps*
]
)

lemma *is-iso-functorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *v11* ($\mathfrak{F}(\lfloor ObjMap \rfloor)$)
and *v11* ($\mathfrak{F}(\lfloor ArrMap \rfloor)$)
and $\mathcal{R}_\circ$ ($\mathfrak{F}(\lfloor ObjMap \rfloor)$) = $\mathfrak{B}(\lfloor Obj \rfloor)$
and $\mathcal{R}_\circ$ ($\mathfrak{F}(\lfloor ArrMap \rfloor)$) = $\mathfrak{B}(\lfloor Arr \rfloor)$
by
(
simp-all add:
is-iso-functorD[*OF assms*(*1*)]
is-iso-semifunctorD'(*2–5*)[
OF is-iso-functorD(*2*)[*OF assms*(*1*)], *unfolded slicing-simps*
]
)

lemma *is-iso-functorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *v11* ($\mathfrak{F}(\lfloor ObjMap \rfloor)$)
and *v11* ($\mathfrak{F}(\lfloor ArrMap \rfloor)$)
and $\mathcal{R}_\circ$ ($\mathfrak{F}(\lfloor ObjMap \rfloor)$) = $\mathfrak{B}(\lfloor Obj \rfloor)$
and $\mathcal{R}_\circ$ ($\mathfrak{F}(\lfloor ArrMap \rfloor)$) = $\mathfrak{B}(\lfloor Arr \rfloor)$
using *assms* **by** (*simp-all add*: *is-iso-functorD'*)

Elementary properties.

context *is-iso-functor*
begin

interpretation *smcf*: *is-iso-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule iso-cf-is-iso-semifunctor*)

lemmas-with [*unfolded slicing-simps*]:

iso-cf-ObjMap-vrange[*simp*] = *smcf.iso-smcf-ObjMap-vrange*
and *iso-cf-ArrMap-vrange*[*simp*] = *smcf.iso-smcf-ArrMap-vrange*

sublocale *ObjMap*: *v11* ‹$\mathfrak{F}(\!|ObjMap|\!)$›
rewrites $\mathcal{D}_{\circ}$ ($\mathfrak{F}(\!|ObjMap|\!)$) = $\mathfrak{A}(\!|Obj|\!)$ **and** $\mathcal{R}_{\circ}$ ($\mathfrak{F}(\!|ObjMap|\!)$) = $\mathfrak{B}(\!|Obj|\!)$
by (*rule smcf.ObjMap.v11-axioms*[*unfolded slicing-simps*])
(*simp-all add*: *cat-cs-simps cf-cs-simps*)

sublocale *ArrMap*: *v11* ‹$\mathfrak{F}(\!|ArrMap|\!)$›
rewrites $\mathcal{D}_{\circ}$ ($\mathfrak{F}(\!|ArrMap|\!)$) = $\mathfrak{A}(\!|Arr|\!)$ **and** $\mathcal{R}_{\circ}$ ($\mathfrak{F}(\!|ArrMap|\!)$) = $\mathfrak{B}(\!|Arr|\!)$
by (*rule smcf.ArrMap.v11-axioms*[*unfolded slicing-simps*])
(*simp-all add*: *cat-cs-simps smcf-cs-simps*)

lemmas-with [*unfolded slicing-simps*]:
iso-cf-Obj-HomDom-if-Obj-HomCod[*elim*] =
smcf.iso-smcf-Obj-HomDom-if-Obj-HomCod
and *iso-cf-Arr-HomDom-if-Arr-HomCod*[*elim*] =
smcf.iso-smcf-Arr-HomDom-if-Arr-HomCod
and *iso-cf-ObjMap-eqE*[*elim*] = *smcf.iso-smcf-ObjMap-eqE*
and *iso-cf-ArrMap-eqE*[*elim*] = *smcf.iso-smcf-ArrMap-eqE*

end

sublocale *is-iso-functor* ⊆ *is-ff-functor*
proof(*intro is-ff-functorI*)
interpret *is-iso-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule iso-cf-is-iso-semifunctor*)
show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.faithful\alpha} \mathfrak{B}$ **by** *unfold-locales*
show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.full\alpha} \mathfrak{B}$ **by** *unfold-locales*
qed

lemmas (**in** *is-iso-functor*) *iso-cf-is-ff-functor* = *is-ff-functor-axioms*
lemmas [*cf-cs-intros*] = *is-iso-functor.iso-cf-is-ff-functor*

4.11.2 Opposite isomorphism of categories

lemma (**in** *is-iso-functor*) *is-iso-functor-op*:
op-cf $\mathfrak{F}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C.iso\alpha}$ *op-cat* $\mathfrak{B}$
by (*rule is-iso-functorI*, *unfold slicing-simps slicing-commute*[*symmetric*])
(*simp-all add*: *cat-op-intros smc-op-intros slicing-intros*)

lemma (**in** *is-iso-functor*) *is-iso-functor-op'*:
assumes $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
shows *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C.iso\alpha}$ *op-cat* $\mathfrak{B}$
unfolding *assms* **by** (*rule is-iso-functor-op*)

lemmas *is-iso-functor-op*[*cat-op-intros*] =
is-iso-functor.is-iso-functor-op'

4.11.3 The composition of isomorphisms of categories is an isomorphism of categories

lemma *cf-comp-is-iso-functor*[*cf-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C.iso\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{C}$
proof(*intro is-iso-functorI*)
interpret $\mathfrak{F}$: *is-iso-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **using** *assms* **by** *auto*
interpret $\mathfrak{G}$: *is-iso-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **using** *assms* **by** *auto*

from $\mathfrak{F}$.*is-functor-axioms* $\mathfrak{G}$.*is-functor-axioms* **show** $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show *cf-smcf* $(\mathfrak{G} \circ_{CF} \mathfrak{F})$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha}$ *cat-smc* $\mathfrak{C}$
unfolding *slicing-commute*[*symmetric*]
by (*cs-concl* **cs-shallow cs-intro**: *smcf-cs-intros slicing-intros*)
qed

4.12 Inverse functor

abbreviation (*input*) *inv-cf* :: $V \Rightarrow V$
where *inv-cf* ≡ *inv-dghm*

Slicing.

lemma *dghm-inv-semifunctor*[*slicing-commute*]:
inv-smcf (*cf-smcf* $\mathfrak{F}$) = *cf-smcf* (*inv-cf* $\mathfrak{F}$)
unfolding *cf-smcf-def inv-dghm-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

context *is-iso-functor*
begin

interpretation *smcf*: *is-iso-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule iso-cf-is-iso-semifunctor*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
inv-cf-ObjMap-v11 = *smcf.inv-smcf-ObjMap-v11*
and *inv-cf-ObjMap-vdomain* = *smcf.inv-smcf-ObjMap-vdomain*
and *inv-cf-ObjMap-app* = *smcf.inv-smcf-ObjMap-app*
and *inv-cf-ObjMap-vrange* = *smcf.inv-smcf-ObjMap-vrange*
and *inv-cf-ArrMap-v11* = *smcf.inv-smcf-ArrMap-v11*
and *inv-cf-ArrMap-vdomain* = *smcf.inv-smcf-ArrMap-vdomain*
and *inv-cf-ArrMap-app* = *smcf.inv-smcf-ArrMap-app*
and *inv-cf-ArrMap-vrange* = *smcf.inv-smcf-ArrMap-vrange*
and *iso-cf-ObjMap-inv-cf-ObjMap-app*[*cf-cs-simps*] =
smcf.iso-smcf-ObjMap-inv-smcf-ObjMap-app
and *iso-cf-ArrMap-inv-cf-ArrMap-app*[*cf-cs-simps*] =
smcf.iso-smcf-ArrMap-inv-smcf-ArrMap-app
and *iso-cf-HomDom-is-arr-conv* = *smcf.iso-smcf-HomDom-is-arr-conv*
and *iso-cf-HomCod-is-arr-conv* = *smcf.iso-smcf-HomCod-is-arr-conv*
and *iso-inv-cf-ObjMap-cf-ObjMap-app*[*cf-cs-simps*] =
smcf.iso-inv-smcf-ObjMap-smcf-ObjMap-app
and *iso-inv-cf-ArrMap-cf-ArrMap-app*[*cf-cs-simps*] =
smcf.iso-inv-smcf-ArrMap-smcf-ArrMap-app

end

lemmas [*cf-cs-intros*] =
is-iso-functor.inv-cf-ObjMap-v11
is-iso-functor.inv-cf-ArrMap-v11

lemmas [*cf-cs-simps*] =
is-iso-functor.inv-cf-ObjMap-vdomain
is-iso-functor.inv-cf-ObjMap-app
is-iso-functor.inv-cf-ObjMap-vrange
is-iso-functor.inv-cf-ArrMap-vdomain
is-iso-functor.inv-cf-ArrMap-app
is-iso-functor.inv-cf-ArrMap-vrange
is-iso-functor.iso-cf-ObjMap-inv-cf-ObjMap-app

is-iso-functor.iso-cf-ArrMap-inv-cf-ArrMap-app
is-iso-functor.iso-inv-cf-ObjMap-cf-ObjMap-app
is-iso-functor.iso-inv-cf-ArrMap-cf-ArrMap-app

4.13 An isomorphism of categories is an isomorphism in the category *CAT*

lemma *is-iso-arr-is-iso-functor*:
— See Chapter I-3 in [7].
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
and $\mathfrak{G} \circ_{CF} \mathfrak{F} = \textit{cf-id}\ \mathfrak{A}$
and $\mathfrak{F} \circ_{CF} \mathfrak{G} = \textit{cf-id}\ \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{A}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule is-iso-functorI*)
have $\mathfrak{GFA}$: *cf-smcf* $\mathfrak{G} \circ_{SMCF}$ *cf-smcf* $\mathfrak{F}$ = *smcf-id* (*cat-smc* $\mathfrak{A}$)
by (*simp add*: *assms*(*3*) *cf-smcf-cf-id cf-smcf-smcf-comp*)
have $\mathfrak{FGB}$: *cf-smcf* $\mathfrak{F} \circ_{SMCF}$ *cf-smcf* $\mathfrak{G}$ = *smcf-id* (*cat-smc* $\mathfrak{B}$)
by (*simp add*: *assms*(*4*) *cf-smcf-cf-id cf-smcf-smcf-comp*)
from $\mathfrak{F}$.*cf-is-semifunctor* $\mathfrak{G}$.*cf-is-semifunctor* $\mathfrak{GFA}$ $\mathfrak{FGB}$ **show**
cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha}$ *cat-smc* $\mathfrak{B}$
by (*rule is-iso-arr-is-iso-semifunctor*)
qed (*auto simp*: *cat-cs-intros*)
qed

lemma *is-iso-functor-is-iso-arr*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
shows [*cf-cs-intros*]: *inv-cf* $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C.iso\alpha} \mathfrak{A}$
and [*cf-cs-simps*]: *inv-cf* $\mathfrak{F} \circ_{CF} \mathfrak{F}$ = *cf-id* $\mathfrak{A}$
and [*cf-cs-simps*]: $\mathfrak{F} \circ_{CF}$ *inv-cf* $\mathfrak{F}$ = *cf-id* $\mathfrak{B}$
proof–

let *?*$\mathfrak{G}$ = *inv-cf* $\mathfrak{F}$

interpret *is-iso-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))

show $\mathfrak{G}$: *?*$\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C.iso\alpha} \mathfrak{A}$
proof(*intro is-iso-functorI is-functorI*, *unfold inv-dghm-components*)
show *vfsequence ?*$\mathfrak{G}$ **by** (*simp add*: *inv-dghm-def*)
show *vcard ?*$\mathfrak{G}$ = $4_{\mathbb{N}}$
unfolding *inv-dghm-def* **by** (*simp add*: *nat-omega-simps*)
show *cf-smcf ?*$\mathfrak{G}$: *cat-smc* $\mathfrak{B} \mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{A}$
by
(
metis
dghm-inv-semifunctor
iso-cf-is-iso-semifunctor
is-iso-semifunctor-def
is-iso-semifunctor-is-iso-arr(*1*)
)
show *cf-smcf ?*$\mathfrak{G}$: *cat-smc* $\mathfrak{B} \mapsto\mapsto_{SMC.iso\alpha}$ *cat-smc* $\mathfrak{A}$
by
(
metis
dghm-inv-semifunctor

```
        iso-cf-is-iso-semifunctor
        is-iso-semifunctor-is-iso-arr(1)
    )
  fix c assume prems: c ∈₀ 𝔅(|Obj|)
  from prems show (𝔉(|ArrMap|))⁻¹∘(|𝔅(|CId|)(|c|)|) = 𝔄(|CId|)(|(𝔉(|ObjMap|))⁻¹∘(|c|)|)
    by (intro v11.v11-vconverse-app)
      (
        cs-concl cs-shallow
          cs-intro: cat-cs-intros V-cs-intros
          cs-simp: V-cs-simps cat-cs-simps
       )+
qed (simp-all add: cat-cs-simps cat-cs-intros)

show ?𝔊 ∘_CF 𝔉 = cf-id 𝔄
proof(rule cf-eqI, unfold dghm-comp-components inv-dghm-components)
  from 𝔊 is-functor-axioms show ?𝔊 ∘_CF 𝔉 : 𝔄 ↦↦_Cα 𝔄
    by (blast intro: cat-cs-intros)
qed
  (
    simp-all add:
      HomDom.cat-cf-id-is-functor
      ObjMap.v11-vcomp-vconverse
      ArrMap.v11-vcomp-vconverse
      dghm-id-components
  )

show 𝔉 ∘_CF ?𝔊 = cf-id 𝔅
proof(rule cf-eqI, unfold dghm-comp-components inv-dghm-components)
  from 𝔊 is-functor-axioms show 𝔉 ∘_CF ?𝔊 : 𝔅 ↦↦_Cα 𝔅
    by (blast intro: cat-cs-intros)
  show cf-id 𝔅 : 𝔅 ↦↦_Cα 𝔅 by (simp add: HomCod.cat-cf-id-is-functor)
qed
  (
    simp-all add:
      ObjMap.v11-vcomp-vconverse′
      ArrMap.v11-vcomp-vconverse′
      dghm-id-components
  )

qed
```

4.13.1 An identity functor is an isomorphism of categories

```
lemma (in category) cat-cf-id-is-iso-functor: cf-id ℭ : ℭ ↦↦_C.isoα ℭ
  by (rule is-iso-functorI, unfold slicing-commute[symmetric])
    (
      simp-all add:
        cat-cf-id-is-functor
        semicategory.smc-smcf-id-is-iso-semifunctor
        cat-semicategory
    )
```

4.14 Isomorphic categories

4.14.1 Definition and elementary properties

See Chapter I-3 in [7]).

```
locale iso-category = L: category α 𝔄 + R: category α 𝔅 for α 𝔄 𝔅 +
```

assumes *iso-cat-is-iso-functor*: $\exists \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$

notation *iso-category* (**infixl** $\langle\approx_{C^1}\rangle$ *50*)

Rules.

lemma *iso-categoryI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
using *assms* **unfolding** *iso-category-def iso-category-axioms-def* **by** *auto*

lemma *iso-categoryD*[*dest*]:
assumes $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
shows $\exists \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
using *assms* **unfolding** *iso-category-def iso-category-axioms-def* **by** *simp-all*

lemma *iso-categoryE*[*elim*]:
assumes $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
obtains $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
using *assms* **by** *auto*

Isomorphic categories are isomorphic semicategories.

lemma (**in** *iso-category*) *iso-cat-iso-semicategory*:
cat-smc $\mathfrak{A} \approx_{SMC\alpha}$ *cat-smc* $\mathfrak{B}$
using *iso-cat-is-iso-functor*
by (*auto intro*: *slicing-intros iso-semicategoryI*)

4.14.2 A category isomorphism is an equivalence relation

lemma *iso-category-refl*:
assumes *category* α $\mathfrak{A}$
shows $\mathfrak{A} \approx_{C\alpha} \mathfrak{A}$
proof(*rule iso-categoryI*[*of* - - - $\langle$*cf-id* $\mathfrak{A}\rangle$])
interpret *category* α $\mathfrak{A}$ **by** (*rule assms*)
show *cf-id* $\mathfrak{A} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{A}$ **by** (*simp add*: *cat-cf-id-is-iso-functor*)
qed

lemma *iso-category-sym*[*sym*]:
assumes $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
shows $\mathfrak{B} \approx_{C\alpha} \mathfrak{A}$
proof–
interpret *iso-category* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*)
from *iso-cat-is-iso-functor* **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$ **by** *clarsimp*
from *is-iso-functor-is-iso-arr*(*1*)[*OF this*] **show** *?thesis*
by (*auto intro*: *iso-categoryI*)
qed

lemma *iso-category-trans*[*trans*]:
assumes $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \approx_{C\alpha} \mathfrak{C}$
shows $\mathfrak{A} \approx_{C\alpha} \mathfrak{C}$
proof–
interpret *L*: *iso-category* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(*1*))
interpret *R*: *iso-category* α $\mathfrak{B}$ $\mathfrak{C}$ **by** (*rule assms*(*2*))
from *L.iso-cat-is-iso-functor R.iso-cat-is-iso-functor* **show** *?thesis*
by (*auto intro*: *iso-categoryI is-iso-functorI cf-comp-is-iso-functor*)
qed

5 Smallness for functors

5.1 Functor with tiny maps

5.1.1 Definition and elementary properties

locale *is-tm-functor* = *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *tm-cf-is-semifunctor*[*slicing-intros*]:
    cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-tm-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\mapsto\mapsto_{C.tm1}$ -)$\rangle$ [*51*, *51*, *51*] *51*)
syntax-consts *-is-tm-functor* $\rightleftharpoons$ *is-tm-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$ *CONST is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation (*input*) *is-cn-tm-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  where *is-cn-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\equiv$ $\mathfrak{F}$: *op-dg* $\mathfrak{A}$ $\mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{B}$

syntax *-is-cn-tm-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $_{C.tm}\mapsto\mapsto_1$ -)$\rangle$ [*51*, *51*, *51*] *51*)
syntax-consts *-is-cn-tm-functor* $\rightleftharpoons$ *is-cn-tm-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $_{C.tm}\mapsto\mapsto_{\alpha}$ $\mathfrak{B}$ $\rightharpoonup$ *CONST is-cn-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation *all-tm-cfs* :: $V \Rightarrow V$
  where *all-tm-cfs* α $\equiv$ *set* $\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

abbreviation *small-tm-cfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *small-tm-cfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\equiv$ *set* $\{\mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

lemma (**in** *is-tm-functor*) *tm-cf-is-semifunctor'*:
  assumes α' = α
    and $\mathfrak{A}'$ = *cat-smc* $\mathfrak{A}$
    and $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
  shows *cf-smcf* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{SMC.tm_{\alpha'}}$ $\mathfrak{B}'$
  unfolding *assms* **by** (*rule tm-cf-is-semifunctor*)

lemmas [*slicing-intros*] = *is-tm-functor.tm-cf-is-semifunctor'*

Rules.

lemma (**in** *is-tm-functor*) *is-tm-functor-axioms'*[*cat-small-cs-intros*]:
  assumes α' = α **and** $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$ = $\mathfrak{B}$
  shows $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{C.tm_{\alpha'}}$ $\mathfrak{B}'$
  unfolding *assms* **by** (*rule is-tm-functor-axioms*)

mk-ide rf *is-tm-functor-def*[*unfolded is-tm-functor-axioms-def*]
  |*intro is-tm-functorI*|
  |*dest is-tm-functorD*[*dest*]|
  |*elim is-tm-functorE*[*elim*]|

lemmas [*cat-small-cs-intros*] = *is-tm-functorD*(*1*)

Slicing.

context *is-tm-functor*
begin

interpretation *smcf*: *is-tm-semifunctor* α $\langle$*cat-smc* $\mathfrak{A}\rangle$ $\langle$*cat-smc* $\mathfrak{B}\rangle$ $\langle$*cf-smcf* $\mathfrak{F}\rangle$
  by (*rule tm-cf-is-semifunctor*)

lemmas-with [*unfolded slicing-simps*]:

tm-cf-ObjMap-in-Vset[*cat-cs-intros*] = *smcf.tm-smcf-ObjMap-in-Vset*
and *tm-cf-ArrMap-in-Vset*[*cat-cs-intros*] = *smcf.tm-smcf-ArrMap-in-Vset*

end

sublocale *is-tm-functor* ⊆ *HomDom*: *tiny-category* α $\mathfrak{A}$
proof(*rule tiny-categoryI'*)
show $\mathfrak{A}(\!|Obj|\!) \in_{\circ} Vset\ \alpha$
by (*rule vdomain-in-VsetI*[*OF tm-cf-ObjMap-in-Vset*, *unfolded cat-cs-simps*])
show $\mathfrak{A}(\!|Arr|\!) \in_{\circ} Vset\ \alpha$
by (*rule vdomain-in-VsetI*[*OF tm-cf-ArrMap-in-Vset*, *unfolded cat-cs-simps*])
qed (*simp add*: *cat-cs-intros*)

Further rules.

lemma *is-tm-functorI'*:
assumes [*simp*]: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and [*simp*]: $\mathfrak{F}(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha$
and [*simp*]: $\mathfrak{F}(\!|ArrMap|\!) \in_{\circ} Vset\ \alpha$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof(*intro is-tm-functorI*)
interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
show *cf-smcf* $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{B}$
by (*intro is-tm-semifunctorI'*, *unfold slicing-simps*)
(*auto simp*: *slicing-intros*)
qed *simp-all*

lemma *is-tm-functorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F}(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha$
and $\mathfrak{F}(\!|ArrMap|\!) \in_{\circ} Vset\ \alpha$
proof–
interpret *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F}(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha$
and $\mathfrak{F}(\!|ArrMap|\!) \in_{\circ} Vset\ \alpha$
by (*auto intro*: *cat-cs-intros*)
qed

lemmas [*cat-small-cs-intros*] = *is-tm-functorD'*(*1*)

lemma *is-tm-functorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F}(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha$
and $\mathfrak{F}(\!|ArrMap|\!) \in_{\circ} Vset\ \alpha$
using *is-tm-functorD'*[*OF assms*] **by** *simp*

Size.

lemma *small-all-tm-cfs*[*simp*]: *small* $\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$
proof(*rule down*)
show
$\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\} \subseteq$
elts (*set* $\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$)
proof
(
simp only: *elts-of-set small-all-cfs if-True*,
rule subsetI,

unfold mem-Collect-eq
)
fix $\mathfrak{F}$ **assume** $\exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
then obtain $\mathfrak{A}\ \mathfrak{B}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$ **by** *clarsimp*
then interpret *is-tm-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **.**
show $\exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** (*blast intro*: *is-functor-axioms'*)
qed
qed

5.1.2 Opposite functor with tiny maps

lemma (**in** *is-tm-functor*) *is-tm-functor-op*:
op-cf $\mathfrak{F}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C.tm\alpha}$ *op-cat* $\mathfrak{B}$
by (*intro is-tm-functorI'*, *unfold cat-op-simps*)
(*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros*)

lemma (**in** *is-tm-functor*) *is-tm-functor-op'*[*cat-op-intros*]:
assumes $\mathfrak{A}' =$ *op-cat* $\mathfrak{A}$ **and** $\mathfrak{B}' =$ *op-cat* $\mathfrak{B}$ **and** $\alpha' = \alpha$
shows *op-cf* $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C.tm\alpha'} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-tm-functor-op*)

lemmas *is-tm-functor-op*[*cat-op-intros*] = *is-tm-functor.is-tm-functor-op'*

5.1.3 Composition of functors with tiny maps

lemma *cf-comp-is-tm-functor*[*cat-small-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*rule is-tm-functorI*)
interpret $\mathfrak{F}$: *is-tm-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}$: *is-tm-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*1*))
from $\mathfrak{F}$.*tm-cf-is-semifunctor* $\mathfrak{G}$.*tm-cf-is-semifunctor* **show**
cf-smcf $(\mathfrak{G} \circ_{CF} \mathfrak{F})$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{C}$
by (*auto simp*: *smc-small-cs-intros slicing-commute*[*symmetric*])
show $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **by** (*auto intro*: *cat-cs-intros*)
qed

5.1.4 Finite categories and functors with tiny maps

lemma (**in** *is-functor*) *cf-is-tm-functor-if-HomDom-finite-category*:
assumes *finite-category* $\alpha\ \mathfrak{A}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof(*intro is-tm-functorI*)
interpret $\mathfrak{A}$: *finite-category* $\alpha\ \mathfrak{A}$ **by** (*rule assms*(*1*))
show *cf-smcf* $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{B}$
by
(
rule
is-semifunctor.smcf-is-tm-semifunctor-if-HomDom-finite-semicategory[
OF cf-is-semifunctor $\mathfrak{A}$.*fin-cat-finite-semicategory*
]
)
qed (*auto intro*: *cat-cs-intros*)

5.1.5 Constant functor with tiny maps

lemma *cf-const-is-tm-functor*:
assumes *tiny-category* $\alpha\ \mathfrak{C}$ **and** *category* $\alpha\ \mathfrak{D}$ **and** $a \in_{\circ} \mathfrak{D}(\lfloor Obj \rfloor)$
shows *cf-const* $\mathfrak{C}\ \mathfrak{D}\ a : \mathfrak{C} \mapsto\mapsto_{C.tm\alpha} \mathfrak{D}$

proof(*intro is-tm-functorI*)
from *assms* **show** *cf-smcf* (*cf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a*) : *cat-smc* $\mathfrak{C}$ $\mapsto\!\!\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{D}$
by
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*] *slicing-simps cat-cs-simps*
cs-intro: *slicing-intros cat-cs-intros smc-small-cs-intros*
)+
from *assms* **show** *cf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a* : $\mathfrak{C}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{D}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-small-cs-intros*)
qed

lemma *cf-const-is-tm-functor'*[*cat-small-cs-intros*]:
assumes *tiny-category* α $\mathfrak{C}$
and *category* α $\mathfrak{D}$
and *a* $\in_\circ$ $\mathfrak{D}$(|*Obj*|)
and $\mathfrak{C}'$ = $\mathfrak{C}$
and $\mathfrak{D}'$ = $\mathfrak{D}$
shows *cf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a* : $\mathfrak{C}'$ $\mapsto\!\!\mapsto_{C.tm\alpha}$ $\mathfrak{D}'$
using *assms*(*1–3*) **unfolding** *assms*(*4,5*) **by** (*rule cf-const-is-tm-functor*)

5.2 Tiny functor

5.2.1 Definition and elementary properties

locale *is-tiny-functor* = *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *tiny-cf-is-tiny-semifunctor*[*slicing-intros*]:
cf-smcf $\mathfrak{F}$: *cat-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.tiny\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-tiny-functor* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
(‹(- :/ - $\mapsto\!\!\mapsto_{C.tiny_1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-tiny-functor* ⇌ *is-tiny-functor*
translations $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.tiny\alpha}$ $\mathfrak{B}$ ⇌ *CONST is-tiny-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation (*input*) *is-cn-tiny-cf* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
where *is-cn-tiny-cf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ ≡ $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.tiny\alpha}$ $\mathfrak{B}$

syntax *-is-cn-tiny-cf* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
(‹(- :/ - $_{C.tiny}\!\mapsto\!\!\mapsto_1$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-cn-tiny-cf* ⇌ *is-cn-cf*
translations $\mathfrak{F}$: $\mathfrak{A}$ $_{C.tiny}\!\mapsto\!\!\mapsto_\alpha$ $\mathfrak{B}$ ↽ *CONST is-cn-cf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation *all-tiny-cfs* :: *V* ⇒ *V*
where *all-tiny-cfs* α ≡ *set* {$\mathfrak{F}$. ∃$\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.tiny\alpha}$ $\mathfrak{B}$}

abbreviation *tiny-cfs* :: *V* ⇒ *V* ⇒ *V* ⇒ *V*
where *tiny-cfs* α $\mathfrak{A}$ $\mathfrak{B}$ ≡ *set* {$\mathfrak{F}$. $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.tiny\alpha}$ $\mathfrak{B}$}

lemmas [*slicing-intros*] = *is-tiny-functor.tiny-cf-is-tiny-semifunctor*

Rules.

lemma (**in** *is-tiny-functor*) *is-tiny-functor-axioms'*[*cat-small-cs-intros*]:
assumes α' = α **and** $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$ = $\mathfrak{B}$
shows $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{C.tiny_{\alpha'}}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule is-tiny-functor-axioms*)

mk-ide rf *is-tiny-functor-def*[*unfolded is-tiny-functor-axioms-def*]
|*intro is-tiny-functorI*|

|*dest is-tiny-functorD*[*dest*]|
|*elim is-tiny-functorE*[*elim*]|

lemmas [*cat-small-cs-intros*] = *is-tiny-functorD*(*1*)

Elementary properties.

sublocale *is-tiny-functor* ⊆ *HomDom*: *tiny-category* α $\mathfrak{A}$
proof(*intro tiny-categoryI'*)
interpret *smcf*: *is-tiny-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule tiny-cf-is-tiny-semifunctor*)
show $\mathfrak{A}(\!|Obj|\!) \in_{\circ} Vset\ \alpha$
by (*rule smcf.HomDom.tiny-smc-Obj-in-Vset*[*unfolded slicing-simps*])
show $\mathfrak{A}(\!|Arr|\!) \in_{\circ} Vset\ \alpha$
by (*rule smcf.HomDom.tiny-smc-Arr-in-Vset*[*unfolded slicing-simps*])
qed (*auto simp*: *cat-cs-intros*)

sublocale *is-tiny-functor* ⊆ *HomCod*: *tiny-category* α $\mathfrak{B}$
proof(*intro tiny-categoryI'*)
interpret *smcf*: *is-tiny-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule tiny-cf-is-tiny-semifunctor*)
show $\mathfrak{B}(\!|Obj|\!) \in_{\circ} Vset\ \alpha$
by (*rule smcf.HomCod.tiny-smc-Obj-in-Vset*[*unfolded slicing-simps*])
show $\mathfrak{B}(\!|Arr|\!) \in_{\circ} Vset\ \alpha$
by (*rule smcf.HomCod.tiny-smc-Arr-in-Vset*[*unfolded slicing-simps*])
qed (*auto simp*: *cat-cs-intros*)

sublocale *is-tiny-functor* ⊆ *is-tm-functor*
proof(*intro is-tm-functorI'*)
interpret *smcf*: *is-tiny-semifunctor* α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$›
by (*rule tiny-cf-is-tiny-semifunctor*)
note *Vset*[*unfolded slicing-simps*] =
smcf.tm-smcf-ObjMap-in-Vset
smcf.tm-smcf-ArrMap-in-Vset
show $\mathfrak{F}(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha\ \mathfrak{F}(\!|ArrMap|\!) \in_{\circ} Vset\ \alpha$ **by** (*intro Vset*)+
qed (*auto simp*: *cat-cs-intros*)

Further rules.

lemma *is-tiny-functorI'*:
assumes [*simp*]: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *tiny-category* α $\mathfrak{A}$
and *tiny-category* α $\mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof(*intro is-tiny-functorI*)
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{A}$: *tiny-category* α $\mathfrak{A}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{B}$: *tiny-category* α $\mathfrak{B}$ **by** (*rule assms*(*3*))
show *cf-smcf* $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha}$ *cat-smc* $\mathfrak{B}$
by (*intro is-tiny-semifunctorI'*) (*auto intro*: *slicing-intros*)
qed (*rule assms*(*1*))

lemma *is-tiny-functorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and *tiny-category* α $\mathfrak{A}$
and *tiny-category* α $\mathfrak{B}$
proof–
interpret *is-tiny-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** *tiny-category* α $\mathfrak{A}$ **and** *tiny-category* α $\mathfrak{B}$

```
    by (auto intro: cat-small-cs-intros)
qed

lemmas [cat-small-cs-intros] = is-tiny-functorD'(2,3)

lemma is-tiny-functorE':
  assumes 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅
  obtains 𝔉 : 𝔄 ↦↦_{C α} 𝔅
    and tiny-category α 𝔄
    and tiny-category α 𝔅
  using is-tiny-functorD'[OF assms] by auto

lemma is-tiny-functor-iff:
  𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅 ⟷
    (𝔉 : 𝔄 ↦↦_{C α} 𝔅 ∧ tiny-category α 𝔄 ∧ tiny-category α 𝔅)
  by (auto intro: is-tiny-functorI' dest: is-tiny-functorD'(2,3))
```

Size.

```
lemma (in is-tiny-functor) tiny-cf-in-Vset: 𝔉 ∈₀ Vset α
proof–
  note [cat-cs-intros] =
    tm-cf-ObjMap-in-Vset
    tm-cf-ArrMap-in-Vset
    HomDom.tiny-cat-in-Vset
    HomCod.tiny-cat-in-Vset
  show ?thesis
    by (subst cf-def)
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros
      )
qed

lemma small-all-tiny-cfs[simp]: small {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅}
proof(rule down)
  show
    {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅} ⊆
      elts (set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C α} 𝔅})
  proof
    (
      simp only: elts-of-set small-all-cfs if-True,
      rule subsetI,
      unfold mem-Collect-eq
    )
    fix 𝔉 assume ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅
    then obtain 𝔄 𝔅 where 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅 by clarsimp
    then interpret is-tiny-functor α 𝔄 𝔅 𝔉 by simp
    show ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C α} 𝔅 by (meson is-functor-axioms)
  qed
qed

lemma small-tiny-cfs[simp]: small {𝔉. 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅}
  by (rule down[of - ‹set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅}›]) auto

lemma all-tiny-cfs-vsubset-Vset[simp]:
  set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅} ⊆₀ Vset α
proof(rule vsubsetI)
  fix 𝔉 assume 𝔉 ∈₀ all-tiny-cfs α
```

then obtain $\mathfrak{A}$ $\mathfrak{B}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$ **by** *clarsimp*
then show $\mathfrak{F} \in_\circ$ *Vset* α **by** (*auto simp*: *is-tiny-functor.tiny-cf-in-Vset*)
qed

lemma (**in** *is-functor*) *cf-is-tiny-functor-if-ge-Limit*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\beta} \mathfrak{B}$
proof(*intro is-tiny-functorI*)
show *cf-smcf* $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tiny\beta}$ *cat-smc* $\mathfrak{B}$
by
(
rule is-semifunctor.smcf-is-tiny-semifunctor-if-ge-Limit,
rule cf-is-semifunctor;
intro assms
)
qed (*simp add*: *cf-is-functor-if-ge-Limit assms*)

5.2.2 Opposite tiny semifunctor

lemma (**in** *is-tiny-functor*) *is-tiny-functor-op*:
op-cf $\mathfrak{F}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C.tiny\alpha}$ *op-cat* $\mathfrak{B}$
by (*intro is-tiny-functorI'*)
(*cs-concl* **cs-intro**: *cat-op-intros cat-small-cs-intros*)+

lemma (**in** *is-tiny-functor*) *is-tiny-functor-op'*[*cat-op-intros*]:
assumes $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$ **and** $\alpha' = \alpha$
shows *op-cf* $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C.tiny_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-tiny-functor-op*)

lemmas *is-tiny-functor-op*[*cat-op-intros*] =
is-tiny-functor.is-tiny-functor-op'

5.2.3 Composition of tiny functors

lemma *cf-comp-is-tiny-functor*[*cat-small-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{F}$: *is-tiny-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}$: *is-tiny-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
show *?thesis* **by** (*rule is-tiny-functorI'*) (*auto intro*: *cat-small-cs-intros*)
qed

5.2.4 Tiny constant functor

lemma *cf-const-is-tiny-functor*:
assumes *tiny-category* α $\mathfrak{C}$ **and** *tiny-category* α $\mathfrak{D}$ **and** $a \in_\circ \mathfrak{D}(\!|Obj|\!)$
shows *cf-const* $\mathfrak{C}$ $\mathfrak{D}$ $a : \mathfrak{C} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{D}$
proof(*intro is-tiny-functorI'*)
from *assms* **show** *cf-const* $\mathfrak{C}$ $\mathfrak{D}$ $a : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
by (*cs-concl* **cs-intro**: *cat-small-cs-intros*)
qed (*auto simp*: *assms*(*1,2*))

lemma *cf-const-is-tiny-functor'*:
assumes *tiny-category* α $\mathfrak{C}$
and *tiny-category* α $\mathfrak{D}$
and $a \in_\circ \mathfrak{D}(\!|Obj|\!)$
and $\mathfrak{C}' = \mathfrak{C}$

and $\mathfrak{D}' = \mathfrak{D}$
shows *cf-const* $\mathfrak{C}$ $\mathfrak{D}$ $a : \mathfrak{C}' \mapsto\mapsto_{C.tiny\alpha} \mathfrak{D}'$
using *assms*(*1–3*) **unfolding** *assms*(*4*,*5*) **by** (*rule cf-const-is-tiny-functor*)

lemmas [*cat-small-cs-intros*] = *cf-const-is-tiny-functor'*

6 Natural transformation

6.1 Background

named-theorems *ntcf-cs-simps*
named-theorems *ntcf-cs-intros*

lemmas [*cat-cs-simps*] = *dg-shared-cs-simps*
lemmas [*cat-cs-intros*] = *dg-shared-cs-intros*

6.1.1 Slicing

definition *ntcf-ntsmcf* :: $V \Rightarrow V$
  where *ntcf-ntsmcf* $\mathfrak{N}$ =
    [
      $\mathfrak{N}(\!|NTMap|\!)$,
      cf-smcf ($\mathfrak{N}(\!|NTDom|\!)$),
      cf-smcf ($\mathfrak{N}(\!|NTCod|\!)$),
      cat-smc ($\mathfrak{N}(\!|NTDGDom|\!)$),
      cat-smc ($\mathfrak{N}(\!|NTDGCod|\!)$)
    ]$_\circ$

Components.

lemma *ntcf-ntsmcf-components*:
  shows [*slicing-simps*]: *ntcf-ntsmcf* $\mathfrak{N}(\!|NTMap|\!)$ = $\mathfrak{N}(\!|NTMap|\!)$
    and [*slicing-commute*]: *ntcf-ntsmcf* $\mathfrak{N}(\!|NTDom|\!)$ = *cf-smcf* ($\mathfrak{N}(\!|NTDom|\!)$)
    and [*slicing-commute*]: *ntcf-ntsmcf* $\mathfrak{N}(\!|NTCod|\!)$ = *cf-smcf* ($\mathfrak{N}(\!|NTCod|\!)$)
    and [*slicing-commute*]: *ntcf-ntsmcf* $\mathfrak{N}(\!|NTDGDom|\!)$ = *cat-smc* ($\mathfrak{N}(\!|NTDGDom|\!)$)
    and [*slicing-commute*]: *ntcf-ntsmcf* $\mathfrak{N}(\!|NTDGCod|\!)$ = *cat-smc* ($\mathfrak{N}(\!|NTDGCod|\!)$)
  unfolding *ntcf-ntsmcf-def nt-field-simps* **by** (*auto simp*: *nat-omega-simps*)

6.2 Definition and elementary properties

The definition of a natural transformation that is used in this work is is similar to the definition that can be found in Chapter I-4 in [7].

locale *is-ntcf* =
  $\mathcal{Z}$ α +
  vfsequence $\mathfrak{N}$ +
  NTDom: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  NTCod: *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$
  for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ +
  assumes *ntcf-length*[*cat-cs-simps*]: *vcard* $\mathfrak{N}$ = $5_{\mathbf{N}}$
    and *ntcf-is-ntsmcf*[*slicing-intros*]: *ntcf-ntsmcf* $\mathfrak{N}$:
      cf-smcf $\mathfrak{F}$ $\mapsto_{SMC}$ *cf-smcf* $\mathfrak{G}$: *cat-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{B}$
    and *ntcf-NTDom*[*cat-cs-simps*]: $\mathfrak{N}(\!|NTDom|\!)$ = $\mathfrak{F}$
    and *ntcf-NTCod*[*cat-cs-simps*]: $\mathfrak{N}(\!|NTCod|\!)$ = $\mathfrak{G}$
    and *ntcf-NTDGDom*[*cat-cs-simps*]: $\mathfrak{N}(\!|NTDGDom|\!)$ = $\mathfrak{A}$
    and *ntcf-NTDGCod*[*cat-cs-simps*]: $\mathfrak{N}(\!|NTDGCod|\!)$ = $\mathfrak{B}$

syntax *-is-ntcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\mapsto_{CF}$ - :/ - $\mapsto\!\!\mapsto_{C1}$ -)$\rangle$ [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-ntcf* $\rightleftharpoons$ *is-ntcf*
translations $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$ *CONST is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

abbreviation *all-ntcfs* :: $V \Rightarrow V$
  where *all-ntcfs* $\alpha \equiv$ *set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\!\!\mapsto_{C\alpha} \mathfrak{B}\}$

abbreviation *ntcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$

where *ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\equiv$ *set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$

abbreviation *these-ntcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *these-ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\equiv$ *set* $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$

lemmas [*cat-cs-simps*] =
is-ntcf.ntcf-length
is-ntcf.ntcf-NTDom
is-ntcf.ntcf-NTCod
is-ntcf.ntcf-NTDGDom
is-ntcf.ntcf-NTDGCod

lemma (**in** *is-ntcf*) *ntcf-is-ntsmcf'*:
assumes $\mathfrak{F}'$ = *cf-smcf* $\mathfrak{F}$
and $\mathfrak{G}'$ = *cf-smcf* $\mathfrak{G}$
and $\mathfrak{A}'$ = *cat-smc* $\mathfrak{A}$
and $\mathfrak{B}'$ = *cat-smc* $\mathfrak{B}$
shows *ntcf-ntsmcf* $\mathfrak{N} : \mathfrak{F}' \mapsto_{SMCF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{SMC\alpha} \mathfrak{B}'$
unfolding *assms*(*1–4*) **by** (*rule ntcf-is-ntsmcf*)

lemmas [*slicing-intros*] = *is-ntcf.ntcf-is-ntsmcf'*

Rules.

lemma (**in** *is-ntcf*) *is-ntcf-axioms'*[*cat-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$ **and** $\mathfrak{F}' = \mathfrak{F}$ **and** $\mathfrak{G}' = \mathfrak{G}$
shows $\mathfrak{N} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C\alpha'} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-ntcf-axioms*)

mk-ide rf *is-ntcf-def*[*unfolded is-ntcf-axioms-def*]
|*intro is-ntcfI*|
|*dest is-ntcfD*[*dest*]|
|*elim is-ntcfE*[*elim*]|

lemmas [*cat-cs-intros*] =
is-ntcfD(*3*,*4*)

lemma *is-ntcfI'*:
assumes $\mathcal{Z}$ α
and *vfsequence* $\mathfrak{N}$
and *vcard* $\mathfrak{N}$ = $5_{\mathbb{N}}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N}(\lfloor NTDom \rfloor) = \mathfrak{F}$
and $\mathfrak{N}(\lfloor NTCod \rfloor) = \mathfrak{G}$
and $\mathfrak{N}(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
and $\mathfrak{N}(\lfloor NTDGCod \rfloor) = \mathfrak{B}$
and *vsv* $(\mathfrak{N}(\lfloor NTMap \rfloor))$
and $\mathcal{D}_\circ\ (\mathfrak{N}(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
and $\bigwedge a.\ a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor) \Longrightarrow \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
$\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{B}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*intro is-ntcfI is-ntsmcfI'*, *unfold ntcf-ntsmcf-components slicing-simps*)
(
simp-all add:
assms nat-omega-simps
ntcf-ntsmcf-def
is-functorD(*6*)[*OF assms*(*4*)]

is-functorD(*6*)[*OF assms*(*5*)]
)

lemma *is-ntcfD'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{N}$
and *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N}(\lfloor NTDom \rfloor) = \mathfrak{F}$
and $\mathfrak{N}(\lfloor NTCod \rfloor) = \mathfrak{G}$
and $\mathfrak{N}(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
and $\mathfrak{N}(\lfloor NTDGCod \rfloor) = \mathfrak{B}$
and *vsv* $(\mathfrak{N}(\lfloor NTMap \rfloor))$
and $\mathcal{D}_{\circ}\ (\mathfrak{N}(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
and $\bigwedge a.\ a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor) \Longrightarrow \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
$\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{B}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
by
(
simp-all add:
is-ntcfD(*2–10*)[*OF assms*]
is-ntsmcfD'[*OF is-ntcfD*(*6*)[*OF assms*], *unfolded slicing-simps*]
)

lemma *is-ntcfE'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
obtains $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{N}$
and *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N}(\lfloor NTDom \rfloor) = \mathfrak{F}$
and $\mathfrak{N}(\lfloor NTCod \rfloor) = \mathfrak{G}$
and $\mathfrak{N}(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
and $\mathfrak{N}(\lfloor NTDGCod \rfloor) = \mathfrak{B}$
and *vsv* $(\mathfrak{N}(\lfloor NTMap \rfloor))$
and $\mathcal{D}_{\circ}\ (\mathfrak{N}(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
and $\bigwedge a.\ a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor) \Longrightarrow \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
$\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{B}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
using *assms* **by** (*simp add*: *is-ntcfD'*)

Slicing.

context *is-ntcf*
begin

interpretation *ntsmcf*:
is-ntsmcf α ‹*cat-smc* $\mathfrak{A}$› ‹*cat-smc* $\mathfrak{B}$› ‹*cf-smcf* $\mathfrak{F}$› ‹*cf-smcf* $\mathfrak{G}$› ‹*ntcf-ntsmcf* $\mathfrak{N}$›
by (*rule ntcf-is-ntsmcf*)

lemmas-with [*unfolded slicing-simps*]:
ntcf-NTMap-vsv = *ntsmcf.ntsmcf-NTMap-vsv*
and *ntcf-NTMap-vdomain*[*cat-cs-simps*] = *ntsmcf.ntsmcf-NTMap-vdomain*
and *ntcf-NTMap-is-arr* = *ntsmcf.ntsmcf-NTMap-is-arr*
and *ntcf-NTMap-is-arr'*[*cat-cs-intros*] = *ntsmcf.ntsmcf-NTMap-is-arr'*

sublocale *NTMap*: *vsv* ‹$\mathfrak{N}(\!|NTMap|\!)$›
  rewrites $\mathcal{D}_{\circ}\ (\mathfrak{N}(\!|NTMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
  by (*rule ntcf-NTMap-vsv*) (*simp add*: *cat-cs-simps*)

lemmas-with [*unfolded slicing-simps*]:
  ntcf-NTMap-app-in-Arr[*cat-cs-intros*] = *ntsmcf.ntsmcf-NTMap-app-in-Arr*
  and *ntcf-NTMap-vrange-vifunion* = *ntsmcf.ntsmcf-NTMap-vrange-vifunion*
  and *ntcf-NTMap-vrange* = *ntsmcf.ntsmcf-NTMap-vrange*
  and *ntcf-NTMap-vsubset-Vset* = *ntsmcf.ntsmcf-NTMap-vsubset-Vset*
  and *ntcf-NTMap-in-Vset* = *ntsmcf.ntsmcf-NTMap-in-Vset*
  and *ntcf-is-ntsmcf-if-ge-Limit* = *ntsmcf.ntsmcf-is-ntsmcf-if-ge-Limit*

lemmas-with [*unfolded slicing-simps*]:
  ntcf-Comp-commute[*cat-cs-intros*] = *ntsmcf.ntsmcf-Comp-commute*
  and *ntcf-Comp-commute'* = *ntsmcf.ntsmcf-Comp-commute'*
  and *ntcf-Comp-commute''* = *ntsmcf.ntsmcf-Comp-commute''*

end

lemmas [*cat-cs-simps*] = *is-ntcf.ntcf-NTMap-vdomain*

lemmas [*cat-cs-intros*] =
  is-ntcf.ntcf-NTMap-vsv
  is-ntcf.ntcf-NTMap-is-arr'
  ntsmcf-hcomp-NTMap-vsv

Elementary properties.

lemma *ntcf-eqI*:
  assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
    and $\mathfrak{N}(\!|NTMap|\!) = \mathfrak{N}'(\!|NTMap|\!)$
    and $\mathfrak{F} = \mathfrak{F}'$
    and $\mathfrak{G} = \mathfrak{G}'$
    and $\mathfrak{A} = \mathfrak{A}'$
    and $\mathfrak{B} = \mathfrak{B}'$
  shows $\mathfrak{N} = \mathfrak{N}'$
proof–
  interpret *L*: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
  interpret *R*: *is-ntcf* $\alpha\ \mathfrak{A}'\ \mathfrak{B}'\ \mathfrak{F}'\ \mathfrak{G}'\ \mathfrak{N}'$ **by** (*rule assms*(*2*))
  show *?thesis*
  proof(*rule vsv-eqI*)
    have *dom*: $\mathcal{D}_{\circ}\ \mathfrak{N} = 5_{\mathbb{N}}$
      by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps V-cs-simps*)
    show $\mathcal{D}_{\circ}\ \mathfrak{N} = \mathcal{D}_{\circ}\ \mathfrak{N}'$
      by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps V-cs-simps*)
    from *assms*(*4–7*) **have** *sup*:
      $\mathfrak{N}(\!|NTDom|\!) = \mathfrak{N}'(\!|NTDom|\!)$ $\mathfrak{N}(\!|NTCod|\!) = \mathfrak{N}'(\!|NTCod|\!)$
      $\mathfrak{N}(\!|NTDGDom|\!) = \mathfrak{N}'(\!|NTDGDom|\!)$ $\mathfrak{N}(\!|NTDGCod|\!) = \mathfrak{N}'(\!|NTDGCod|\!)$
      by (*simp-all add*: *cat-cs-simps*)
    show $a \in_{\circ} \mathcal{D}_{\circ}\ \mathfrak{N} \Longrightarrow \mathfrak{N}(\!|a|\!) = \mathfrak{N}'(\!|a|\!)$ **for** *a*
      by (*unfold dom*, *elim-in-numeral*, *insert assms*(*3*) *sup*)
        (*auto simp*: *nt-field-simps*)
  qed *auto*
qed

lemma *ntcf-ntsmcf-eqI*:
  assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$

```
    and 𝔉 = 𝔉′
    and 𝔊 = 𝔊′
    and 𝔄 = 𝔄′
    and 𝔅 = 𝔅′
    and ntcf-ntsmcf 𝔑 = ntcf-ntsmcf 𝔑′
  shows 𝔑 = 𝔑′
proof(rule ntcf-eqI[of α])
  from assms(7) have ntcf-ntsmcf 𝔑(|NTMap|) = ntcf-ntsmcf 𝔑′(|NTMap|) by simp
  then show 𝔑(|NTMap|) = 𝔑′(|NTMap|) unfolding slicing-simps by simp-all
  from assms(3–6) show 𝔉 = 𝔉′ 𝔊 = 𝔊′ 𝔄 = 𝔄′ 𝔅 = 𝔅′ by simp-all
qed (auto simp: assms(1,2))

lemma (in is-ntcf) ntcf-def:
  𝔑 = [𝔑(|NTMap|), 𝔑(|NTDom|), 𝔑(|NTCod|), 𝔑(|NTDGDom|), 𝔑(|NTDGCod|)]∘
proof(rule vsv-eqI)
  have dom-lhs: 𝒟∘ 𝔑 = 5ℕ
    by (cs-concl cs-shallow cs-simp: cat-cs-simps V-cs-simps)
  have dom-rhs:
    𝒟∘ [𝔑(|NTMap|), 𝔑(|NTDGDom|), 𝔑(|NTDGCod|), 𝔑(|NTDom|), 𝔑(|NTCod|)]∘ = 5ℕ
    by (simp add: nat-omega-simps)
  then show
    𝒟∘ 𝔑 = 𝒟∘ [𝔑(|NTMap|), 𝔑(|NTDom|), 𝔑(|NTCod|), 𝔑(|NTDGDom|), 𝔑(|NTDGCod|)]∘
    unfolding dom-lhs dom-rhs by (simp add: nat-omega-simps)
  show a ∈∘ 𝒟∘ 𝔑 ⟹
    𝔑(|a|) = [𝔑(|NTMap|), 𝔑(|NTDom|), 𝔑(|NTCod|), 𝔑(|NTDGDom|), 𝔑(|NTDGCod|)]∘(|a|)
    for a
    by (unfold dom-lhs, elim-in-numeral, unfold nt-field-simps)
      (simp-all add: nat-omega-simps)
qed (auto simp: vsv-axioms)

lemma (in is-ntcf) ntcf-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔑 ∈∘ Vset β
proof–
  interpret β: 𝒵 β by (rule assms(1))
  note [cat-cs-intros] =
    ntcf-NTMap-in-Vset
    NTDom.cf-in-Vset
    NTCod.cf-in-Vset
    NTDom.HomDom.cat-in-Vset
    NTDom.HomCod.cat-in-Vset
  from assms(2) show ?thesis
    by (subst ntcf-def)
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros
      )
qed

lemma (in is-ntcf) ntcf-is-ntcf-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔑 : 𝔉 ↦↦CF 𝔊 : 𝔄 ↦↦↦Cβ 𝔅
proof(intro is-ntcfI)
  show ntcf-ntsmcf 𝔑 :
    cf-smcf 𝔉 ↦↦SMCF cf-smcf 𝔊 : cat-smc 𝔄 ↦↦↦SMCβ cat-smc 𝔅
    by (rule is-ntsmcf.ntsmcf-is-ntsmcf-if-ge-Limit[OF ntcf-is-ntsmcf assms])
qed
  (
```

```
    cs-concl cs-shallow
      cs-simp: cat-cs-simps
      cs-intro:
        V-cs-intros
        assms
        NTDom.cf-is-functor-if-ge-Limit
        NTCod.cf-is-functor-if-ge-Limit
  )+

lemma small-all-ntcfs[simp]:
  small {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-ntcf.ntcf-in-Vset show ?thesis
    by (intro down[of - ‹Vset (α + ω)›])
      (auto simp: True 𝒵.𝒵-Limit-αω 𝒵.𝒵-ω-αω 𝒵.intro 𝒵.𝒵-α-αω)
next
  case False
  then have {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅} = {} by auto
  then show ?thesis by simp
qed

lemma small-ntcfs[simp]: small {𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅}
  by (rule down[of - ‹set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅}›]) auto

lemma small-these-ntcfs[simp]: small {𝔑. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅}
  by (rule down[of - ‹set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅}›]) auto
```

Further elementary results.

```
lemma these-ntcfs-iff:
  𝔑 ∈₀ these-ntcfs α 𝔄 𝔅 𝔉 𝔊 ⟷ 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
  by auto
```

6.3 Opposite natural transformation

See section 1.5 in [3].

```
definition op-ntcf :: V ⇒ V
  where op-ntcf 𝔑 =
    [
      𝔑(|NTMap|),
      op-cf (𝔑(|NTCod|)),
      op-cf (𝔑(|NTDom|)),
      op-cat (𝔑(|NTDGDom|)),
      op-cat (𝔑(|NTDGCod|))
    ]₀
```

Components.

```
lemma op-ntcf-components[cat-op-simps]:
  shows op-ntcf 𝔑(|NTMap|) = 𝔑(|NTMap|)
    and op-ntcf 𝔑(|NTDom|) = op-cf (𝔑(|NTCod|))
    and op-ntcf 𝔑(|NTCod|) = op-cf (𝔑(|NTDom|))
    and op-ntcf 𝔑(|NTDGDom|) = op-cat (𝔑(|NTDGDom|))
    and op-ntcf 𝔑(|NTDGCod|) = op-cat (𝔑(|NTDGCod|))
  unfolding op-ntcf-def nt-field-simps by (auto simp: nat-omega-simps)
```

Slicing.

```
lemma ntcf-ntsmcf-op-ntcf[slicing-commute]:
```

```
  op-ntsmcf (ntcf-ntsmcf 𝔑) = ntcf-ntsmcf (op-ntcf 𝔑)
proof(rule vsv-eqI)
  have dom-lhs: 𝒟₀ (op-ntsmcf (ntcf-ntsmcf 𝔑)) = 5ℕ
    unfolding op-ntsmcf-def by (auto simp: nat-omega-simps)
  have dom-rhs: 𝒟₀ (ntcf-ntsmcf (op-ntcf 𝔑)) = 5ℕ
    unfolding ntcf-ntsmcf-def by (auto simp: nat-omega-simps)
  show 𝒟₀ (op-ntsmcf (ntcf-ntsmcf 𝔑)) = 𝒟₀ (ntcf-ntsmcf (op-ntcf 𝔑))
    unfolding dom-lhs dom-rhs by simp
  show a ∈₀ 𝒟₀ (op-ntsmcf (ntcf-ntsmcf 𝔑)) ⟹
    op-ntsmcf (ntcf-ntsmcf 𝔑)(‖a‖) = ntcf-ntsmcf (op-ntcf 𝔑)(‖a‖)
    for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold nt-field-simps ntcf-ntsmcf-def op-ntcf-def op-ntsmcf-def
      )
      (auto simp: nat-omega-simps slicing-commute[symmetric])
qed (auto simp: ntcf-ntsmcf-def op-ntsmcf-def)
```

Elementary properties.

```
lemma op-ntcf-vsv[cat-op-intros]: vsv (op-ntcf 𝔉)
  unfolding op-ntcf-def by auto
```

6.3.1 Further properties

```
lemma (in is-ntcf) is-ntcf-op:
  op-ntcf 𝔑 : op-cf 𝔊 ↦CF op-cf 𝔉 : op-cat 𝔄 ↦↦Cα op-cat 𝔅
proof(rule is-ntcfI, unfold cat-op-simps)
  show vfsequence (op-ntcf 𝔑) by (simp add: op-ntcf-def)
  show vcard (op-ntcf 𝔑) = 5ℕ by (simp add: op-ntcf-def nat-omega-simps)
qed
  (
    use is-ntcf-axioms in
    ‹
      cs-concl cs-shallow
        cs-simp: cat-cs-simps slicing-commute[symmetric]
        cs-intro: cat-cs-intros cat-op-intros smc-op-intros slicing-intros
    ›
  )+

lemma (in is-ntcf) is-ntcf-op'[cat-op-intros]:
  assumes 𝔊' = op-cf 𝔊
    and 𝔉' = op-cf 𝔉
    and 𝔄' = op-cat 𝔄
    and 𝔅' = op-cat 𝔅
  shows op-ntcf 𝔑 : 𝔊' ↦CF 𝔉' : 𝔄' ↦↦Cα 𝔅'
  unfolding assms by (rule is-ntcf-op)

lemmas [cat-op-intros] = is-ntcf.is-ntcf-op'

lemma (in is-ntcf) ntcf-op-ntcf-op-ntcf[cat-op-simps]:
  op-ntcf (op-ntcf 𝔑) = 𝔑
proof(rule ntcf-eqI[of α 𝔄 𝔅 𝔉 𝔊 - 𝔄 𝔅 𝔉 𝔊], unfold cat-op-simps)
  interpret op:
    is-ntcf α ‹op-cat 𝔄› ‹op-cat 𝔅› ‹op-cf 𝔊› ‹op-cf 𝔉› ‹op-ntcf 𝔑›
    by (rule is-ntcf-op)
  from op.is-ntcf-op show
```

op-ntcf (*op-ntcf* $\mathfrak{N}$) : $\mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*simp add*: *cat-op-simps*)
qed (*auto simp*: *cat-cs-intros*)

lemmas *ntcf-op-ntcf-op-ntcf*[*cat-op-simps*] =
is-ntcf.ntcf-op-ntcf-op-ntcf

lemma *eq-op-ntcf-iff*[*cat-op-simps*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
shows *op-ntcf* $\mathfrak{N}$ = *op-ntcf* $\mathfrak{N}' \longleftrightarrow \mathfrak{N} = \mathfrak{N}'$
proof
interpret *L*: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
interpret *R*: *is-ntcf* α $\mathfrak{A}'$ $\mathfrak{B}'$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{N}'$ **by** (*rule assms*(*2*))
assume *prems*: *op-ntcf* $\mathfrak{N}$ = *op-ntcf* $\mathfrak{N}'$
show $\mathfrak{N} = \mathfrak{N}'$
proof(*rule ntcf-eqI*[*OF assms*])
from *prems L.ntcf-op-ntcf-op-ntcf R.ntcf-op-ntcf-op-ntcf* **show**
$\mathfrak{N}(\!|NTMap|\!) = \mathfrak{N}'(\!|NTMap|\!)$
by *metis+*
from *prems L.ntcf-op-ntcf-op-ntcf R.ntcf-op-ntcf-op-ntcf*
have $\mathfrak{N}(\!|NTDom|\!) = \mathfrak{N}'(\!|NTDom|\!)$
and $\mathfrak{N}(\!|NTCod|\!) = \mathfrak{N}'(\!|NTCod|\!)$
and $\mathfrak{N}(\!|NTDGDom|\!) = \mathfrak{N}'(\!|NTDGDom|\!)$
and $\mathfrak{N}(\!|NTDGCod|\!) = \mathfrak{N}'(\!|NTDGCod|\!)$
by *metis+*
then show $\mathfrak{F} = \mathfrak{F}'$ $\mathfrak{G} = \mathfrak{G}'$ $\mathfrak{A} = \mathfrak{A}'$ $\mathfrak{B} = \mathfrak{B}'$
by (*auto simp*: *cat-cs-simps*)
qed
qed *auto*

6.4 Vertical composition of natural transformations

6.4.1 Definition and elementary properties

See Chapter II-4 in [7].

abbreviation (*input*) *ntcf-vcomp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** $\langle\bullet_{NTCF}\rangle$ *55*)
where *ntcf-vcomp* ≡ *ntsmcf-vcomp*

lemmas [*cat-cs-simps*] = *ntsmcf-vcomp-components*(*2–5*)

Slicing.

lemma *ntcf-ntsmcf-ntcf-vcomp*[*slicing-commute*]:
ntcf-ntsmcf $\mathfrak{M}$ $\bullet_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$ = *ntcf-ntsmcf* ($\mathfrak{M}$ $\bullet_{NTCF}$ $\mathfrak{N}$)
unfolding
ntsmcf-vcomp-def ntcf-ntsmcf-def cat-smc-def nt-field-simps dg-field-simps
by (*simp add*: *nat-omega-simps*)

6.4.2 Natural transformation map

lemma *ntcf-vcomp-NTMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ$ (($\mathfrak{M}$ $\bullet_{NTCF}$ $\mathfrak{N}$)($\!|NTMap|\!$)) = $\mathfrak{A}(\!|Obj|\!)$
proof–
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **using** *assms* **by** *auto*
show *?thesis*
by
(
rule ntsmcf-vcomp-NTMap-vdomain

[
OF $\mathfrak{N}$.*ntcf-is-ntsmcf*,
of ‹*ntcf-ntsmcf* $\mathfrak{M}$›,
unfolded slicing-commute slicing-simps
]
)
qed

lemma *ntcf-vcomp-NTMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
shows $(\mathfrak{M} \cdot_{NTCF} \mathfrak{N})(\!|NTMap|\!)(\!|a|\!) = \mathfrak{M}(\!|NTMap|\!)(\!|a|\!) \circ_{A\mathfrak{B}} \mathfrak{N}(\!|NTMap|\!)(\!|a|\!)$
proof–
interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **using** *assms* **by** *clarsimp*
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **using** *assms* **by** *clarsimp*
show *?thesis*
by
(
rule ntsmcf-vcomp-NTMap-app
[
OF $\mathfrak{M}$.*ntcf-is-ntsmcf* $\mathfrak{N}$.*ntcf-is-ntsmcf*,
unfolded slicing-commute slicing-simps,
OF assms(*3*)
]
)
qed

lemma *ntcf-vcomp-NTMap-vrange*:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{M} \cdot_{NTCF} \mathfrak{N})(\!|NTMap|\!)) \subseteq_{\circ} \mathfrak{B}(\!|Arr|\!)$
proof–
interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **using** *assms* **by** *auto*
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **using** *assms* **by** *auto*
show *?thesis*
by
(
rule
ntsmcf-vcomp-NTMap-vrange[
OF $\mathfrak{M}$.*ntcf-is-ntsmcf* $\mathfrak{N}$.*ntcf-is-ntsmcf*,
unfolded slicing-simps slicing-commute
]
)
qed

6.4.3 Further properties

lemma *ntcf-vcomp-composable-commute*[*cat-cs-simps*]:
— See Chapter II-4 in [7].
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and [*intro*]: $f : a \mapsto_{\mathfrak{A}} b$
shows
$(\mathfrak{M}(\!|NTMap|\!)(\!|b|\!) \circ_{A\mathfrak{B}} \mathfrak{N}(\!|NTMap|\!)(\!|b|\!)) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) =$
$\mathfrak{H}(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{B}} (\mathfrak{M}(\!|NTMap|\!)(\!|a|\!) \circ_{A\mathfrak{B}} \mathfrak{N}(\!|NTMap|\!)(\!|a|\!))$
proof–
interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))

```
  show ?thesis
    by
      (
        rule ntsmcf-vcomp-composable-commute[
            OF 𝔐.ntcf-is-ntsmcf 𝔑.ntcf-is-ntsmcf,
            unfolded slicing-simps,
            OF assms(3)
          ]
      )
qed

lemma ntcf-vcomp-is-ntcf[cat-cs-intros]:
  — see Chapter II-4 in [7].
  assumes 𝔐 : 𝔊 ↦CF ℌ : 𝔄 ↦↦Cα 𝔅 and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
  shows 𝔐 ∘NTCF 𝔑 : 𝔉 ↦CF ℌ : 𝔄 ↦↦Cα 𝔅
proof–
  interpret 𝔐: is-ntcf α 𝔄 𝔅 𝔊 ℌ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  show ?thesis
  proof(intro is-ntcfI)
    show vfsequence (𝔐 ∘NTCF 𝔑) by (simp add: ntsmcf-vcomp-def)
    show vcard (𝔐 ∘NTCF 𝔑) = 5ℕ
      unfolding ntsmcf-vcomp-def by (simp add: nat-omega-simps)
    show ntcf-ntsmcf (𝔐 ∘NTCF 𝔑) :
      cf-smcf 𝔉 ↦SMCF cf-smcf ℌ : cat-smc 𝔄 ↦↦SMCα cat-smc 𝔅
      by
        (
          rule ntsmcf-vcomp-is-ntsmcf[
            OF 𝔐.ntcf-is-ntsmcf 𝔑.ntcf-is-ntsmcf,
            unfolded slicing-simps slicing-commute
            ]
        )
  qed (auto simp: ntsmcf-vcomp-components(1) cat-cs-simps cat-cs-intros)
qed

lemma ntcf-vcomp-assoc[cat-cs-simps]:
  — See Chapter II-4 in [7].
  assumes 𝔏 : ℌ ↦CF 𝔎 : 𝔄 ↦↦Cα 𝔅
    and 𝔐 : 𝔊 ↦CF ℌ : 𝔄 ↦↦Cα 𝔅
    and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
  shows (𝔏 ∘NTCF 𝔐) ∘NTCF 𝔑 = 𝔏 ∘NTCF (𝔐 ∘NTCF 𝔑)
proof–
  interpret 𝔏: is-ntcf α 𝔄 𝔅 ℌ 𝔎 𝔏 by (rule assms(1))
  interpret 𝔐: is-ntcf α 𝔄 𝔅 𝔊 ℌ 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(3))
  show ?thesis
  proof(rule ntcf-eqI[of α])
    from ntsmcf-vcomp-assoc[
        OF 𝔏.ntcf-is-ntsmcf 𝔐.ntcf-is-ntsmcf 𝔑.ntcf-is-ntsmcf,
        unfolded slicing-simps slicing-commute
      ]
    have
      ntcf-ntsmcf (𝔏 ∘NTCF 𝔐 ∘NTCF 𝔑)⦇NTMap⦈ =
        ntcf-ntsmcf (𝔏 ∘NTCF (𝔐 ∘NTCF 𝔑))⦇NTMap⦈
      by simp
    then show (𝔏 ∘NTCF 𝔐 ∘NTCF 𝔑)⦇NTMap⦈ = (𝔏 ∘NTCF (𝔐 ∘NTCF 𝔑))⦇NTMap⦈
      unfolding slicing-simps .
  qed (auto intro: cat-cs-intros)
```

qed

6.4.4 The opposite of the vertical composition of natural transformations

lemma *op-ntcf-ntcf-vcomp*[*cat-op-simps*]:
  assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
  shows *op-ntcf* $(\mathfrak{M} \bullet_{NTCF} \mathfrak{N})$ = *op-ntcf* $\mathfrak{N} \bullet_{NTCF}$ *op-ntcf* $\mathfrak{M}$
proof–
  interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **using** *assms*(*1*) **by** *auto*
  interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **using** *assms*(*2*) **by** *auto*
  show *?thesis*
  proof(*rule sym*, *rule ntcf-eqI*[*of* α])
    from
      op-ntsmcf-ntsmcf-vcomp
        [
          OF $\mathfrak{M}$.*ntcf-is-ntsmcf* $\mathfrak{N}$.*ntcf-is-ntsmcf*,
          unfolded slicing-simps slicing-commute
        ]
    have *ntcf-ntsmcf* (*op-ntcf* $\mathfrak{N} \bullet_{NTCF}$ *op-ntcf* $\mathfrak{M}$)(‖*NTMap*‖) =
      ntcf-ntsmcf (*op-ntcf* $(\mathfrak{M} \bullet_{NTCF} \mathfrak{N})$)(‖*NTMap*‖)
      by *simp*
    then show (*op-ntcf* $\mathfrak{N} \bullet_{NTCF}$ *op-ntcf* $\mathfrak{M}$)(‖*NTMap*‖) = *op-ntcf* $(\mathfrak{M} \bullet_{NTCF} \mathfrak{N})$(‖*NTMap*‖)
      unfolding *slicing-simps* **.**
  qed (*auto intro*: *cat-cs-intros cat-op-intros*)
qed

6.5 Horizontal composition of natural transformations

6.5.1 Definition and elementary properties

See Chapter II-5 in [7].

abbreviation (*input*) *ntcf-hcomp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\circ_{NTCF}$› *55*)
  where *ntcf-hcomp* ≡ *ntsmcf-hcomp*

lemmas [*cat-cs-simps*] = *ntsmcf-hcomp-components*(*2–5*)

Slicing.

lemma *ntcf-ntsmcf-ntcf-hcomp*[*slicing-commute*]:
  ntcf-ntsmcf $\mathfrak{M} \circ_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$ = *ntcf-ntsmcf* $(\mathfrak{M} \circ_{NTCF} \mathfrak{N})$
proof(*rule vsv-eqI*)
  show *vsv* (*ntcf-ntsmcf* $\mathfrak{M} \circ_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$)
    unfolding *ntsmcf-hcomp-def* **by** *auto*
  show *vsv* (*ntcf-ntsmcf* $(\mathfrak{M} \circ_{NTCF} \mathfrak{N})$) **unfolding** *ntcf-ntsmcf-def* **by** *auto*
  have *dom-lhs*:
    $\mathcal{D}_\circ$ (*ntcf-ntsmcf* $\mathfrak{M} \circ_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$) = $5_{\mathbf{N}}$
    unfolding *ntsmcf-hcomp-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*ntcf-ntsmcf* $(\mathfrak{M} \circ_{NTCF} \mathfrak{N})$) = $5_{\mathbf{N}}$
    unfolding *ntcf-ntsmcf-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*ntcf-ntsmcf* $\mathfrak{M} \circ_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$) =
    $\mathcal{D}_\circ$ (*ntcf-ntsmcf* $(\mathfrak{M} \circ_{NTCF} \mathfrak{N})$)
    unfolding *dom-lhs dom-rhs* **..**
  fix *a* **assume** *a* $\in_\circ \mathcal{D}_\circ$ (*ntcf-ntsmcf* $\mathfrak{M} \circ_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$)
  then show
    (*ntcf-ntsmcf* $\mathfrak{M} \circ_{NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$)(‖*a*‖) = *ntcf-ntsmcf* $(\mathfrak{M} \circ_{NTCF} \mathfrak{N})$(‖*a*‖)
    unfolding *dom-lhs*
    by (*elim-in-numeral*; *fold nt-field-simps*)
      (*simp-all add*: *ntsmcf-hcomp-components slicing-simps slicing-commute*)

qed

6.5.2 Natural transformation map

lemma *ntcf-hcomp-NTMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ\ ((\mathfrak{M} \circ_{NTCF} \mathfrak{N})(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
proof–
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
show *?thesis* **unfolding** *ntsmcf-hcomp-components* **by** (*simp add*: *cat-cs-simps*)
qed

lemma *ntcf-hcomp-NTMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
shows $(\mathfrak{M} \circ_{NTCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor a \rfloor) =$
$\mathfrak{G}'(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) \rfloor) \circ_{A\mathfrak{C}} \mathfrak{M}(\lfloor NTMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)$
proof–
interpret $\mathfrak{M}$: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}'\ \mathfrak{G}'\ \mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*2*))
from *assms*(*3*) **show** *?thesis*
unfolding *ntsmcf-hcomp-components* **by** (*simp add*: *cat-cs-simps*)
qed

lemma *ntcf-hcomp-NTMap-vrange*:
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ\ ((\mathfrak{M} \circ_{NTCF} \mathfrak{N})(\lfloor NTMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof–
interpret $\mathfrak{M}$: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}'\ \mathfrak{G}'\ \mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
rule ntsmcf-hcomp-NTMap-vrange[
OF $\mathfrak{M}$.*ntcf-is-ntsmcf* $\mathfrak{N}$.*ntcf-is-ntsmcf*,
unfolded slicing-simps slicing-commute
]
)
qed

6.5.3 Further properties

lemma *ntcf-hcomp-composable-commute*:
— See Chapter II-5 in [7].
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $f : a \mapsto_{\mathfrak{A}} b$
shows
$(\mathfrak{M} \circ_{NTCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{F}' \circ_{CF} \mathfrak{F})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) =$
$(\mathfrak{G}' \circ_{CF} \mathfrak{G})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{M} \circ_{NTCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
(**is** $\langle ?\mathfrak{M}\mathfrak{N}b \circ_{A\mathfrak{C}} ?\mathfrak{F}'\mathfrak{F}f = ?\mathfrak{G}'\mathfrak{G}f \circ_{A\mathfrak{C}} ?\mathfrak{M}\mathfrak{N}a \rangle$)
proof–
interpret $\mathfrak{M}$: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}'\ \mathfrak{G}'\ \mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(

```
      rule ntsmcf-hcomp-composable-commute[
        OF 𝔐.ntcf-is-ntsmcf 𝔑.ntcf-is-ntsmcf,
        unfolded slicing-simps slicing-commute,
        OF assms(3)
        ]
    )
qed

lemma ntcf-hcomp-is-ntcf:
  — See Chapter II-5 in [7].
  assumes 𝔐 : 𝔉' ↦CF 𝔊' : 𝔅 ↦↦Cα ℭ and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
  shows 𝔐 ∘NTCF 𝔑 : 𝔉' ∘CF 𝔉 ↦CF 𝔊' ∘CF 𝔊 : 𝔄 ↦↦Cα ℭ
proof–
  interpret 𝔐: is-ntcf α 𝔅 ℭ 𝔉' 𝔊' 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  show ?thesis
  proof(intro is-ntcfI)
    show vfsequence (𝔐 ∘NTCF 𝔑)
      unfolding ntsmcf-hcomp-def by (simp add: nat-omega-simps)
    show vcard (𝔐 ∘NTCF 𝔑) = 5ℕ
      unfolding ntsmcf-hcomp-def by (simp add: nat-omega-simps)
    show ntcf-ntsmcf (𝔐 ∘NTCF 𝔑) :
      cf-smcf (𝔉' ∘SMCF 𝔉) ↦SMCF cf-smcf (𝔊' ∘CF 𝔊) :
      cat-smc 𝔄 ↦↦SMCα cat-smc ℭ
      by
        (
        rule ntsmcf-hcomp-is-ntsmcf[
          OF 𝔐.ntcf-is-ntsmcf 𝔑.ntcf-is-ntsmcf,
          unfolded slicing-simps slicing-commute
          ]
        )
  qed (auto simp: ntsmcf-hcomp-components(1) cat-cs-simps intro: cat-cs-intros)
qed

lemma ntcf-hcomp-is-ntcf'[cat-cs-intros]:
  assumes 𝔐 : 𝔉' ↦CF 𝔊' : 𝔅 ↦↦Cα ℭ
    and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
    and 𝔖 = 𝔉' ∘CF 𝔉
    and 𝔖' = 𝔊' ∘CF 𝔊
  shows 𝔐 ∘NTCF 𝔑 : 𝔖 ↦CF 𝔖' : 𝔄 ↦↦Cα ℭ
  using assms(1,2) unfolding assms(3,4) by (rule ntcf-hcomp-is-ntcf)

lemma ntcf-hcomp-associativ[cat-cs-simps]:
  assumes 𝔏 : 𝔉'' ↦CF 𝔊'' : ℭ ↦↦Cα 𝔇
    and 𝔐 : 𝔉' ↦CF 𝔊' : 𝔅 ↦↦Cα ℭ
    and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
  shows (𝔏 ∘NTCF 𝔐) ∘NTCF 𝔑 = 𝔏 ∘NTCF (𝔐 ∘NTCF 𝔑)
proof–
  interpret 𝔏: is-ntcf α ℭ 𝔇 𝔉'' 𝔊'' 𝔏 by (rule assms(1))
  interpret 𝔐: is-ntcf α 𝔅 ℭ 𝔉' 𝔊' 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(3))
  show ?thesis
  proof(rule ntcf-eqI[of α])
    show 𝔏 ∘NTCF (𝔐 ∘NTCF 𝔑) :
      𝔉'' ∘CF 𝔉' ∘CF 𝔉 ↦CF 𝔊'' ∘CF 𝔊' ∘CF 𝔊 : 𝔄 ↦↦Cα 𝔇
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from ntsmcf-hcomp-assoc[
      OF 𝔏.ntcf-is-ntsmcf 𝔐.ntcf-is-ntsmcf 𝔑.ntcf-is-ntsmcf,
```

unfolded slicing-commute
]
have
ntcf-ntsmcf ($\mathfrak{L} \circ_{NTCF} \mathfrak{M} \circ_{NTCF} \mathfrak{N}$)(|*NTMap*|) =
ntcf-ntsmcf ($\mathfrak{L} \circ_{NTCF}$ ($\mathfrak{M} \circ_{NTCF} \mathfrak{N}$))(|*NTMap*|)
by *simp*
then show ($\mathfrak{L} \circ_{NTCF} \mathfrak{M} \circ_{NTCF} \mathfrak{N}$)(|*NTMap*|) = ($\mathfrak{L} \circ_{NTCF}$ ($\mathfrak{M} \circ_{NTCF} \mathfrak{N}$))(|*NTMap*|)
unfolding *slicing-simps* **.**
qed (*auto intro*: *cat-cs-intros*)
qed

6.5.4 The opposite of the horizontal composition of natural transformations

lemma *op-ntcf-ntcf-hcomp*[*cat-op-simps*]:
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *op-ntcf* ($\mathfrak{M} \circ_{NTCF} \mathfrak{N}$) = *op-ntcf* $\mathfrak{M} \circ_{NTCF}$ *op-ntcf* $\mathfrak{N}$
proof–
interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule sym*, *rule ntcf-eqI*[*of* α])
from *op-ntsmcf-ntsmcf-hcomp*[
OF $\mathfrak{M}$.*ntcf-is-ntsmcf* $\mathfrak{N}$.*ntcf-is-ntsmcf*,
unfolded slicing-simps slicing-commute
]
have *ntcf-ntsmcf* (*op-ntcf* $\mathfrak{M} \circ_{NTCF}$ *op-ntcf* $\mathfrak{N}$)(|*NTMap*|) =
ntcf-ntsmcf (*op-ntcf* ($\mathfrak{M} \circ_{NTCF} \mathfrak{N}$))(|*NTMap*|)
by *simp*
then show (*op-ntcf* $\mathfrak{M} \circ_{NTCF}$ *op-ntcf* $\mathfrak{N}$)(|*NTMap*|) = *op-ntcf* ($\mathfrak{M} \circ_{NTCF} \mathfrak{N}$)(|*NTMap*|)
unfolding *slicing-simps* **.**
have $\mathfrak{M} \circ_{NTCF} \mathfrak{N} : \mathfrak{F}' \circ_{CF} \mathfrak{F} \mapsto_{CF} \mathfrak{G}' \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*rule ntcf-hcomp-is-ntcf*[*OF assms*])
from *is-ntcf.is-ntcf-op*[*OF this*] **show**
op-ntcf ($\mathfrak{M} \circ_{NTCF} \mathfrak{N}$) :
op-cf $\mathfrak{G}' \circ_{CF}$ *op-cf* $\mathfrak{G} \mapsto_{CF}$ *op-cf* $\mathfrak{F}' \circ_{CF}$ *op-cf* $\mathfrak{F}$:
op-cat $\mathfrak{A} \mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
unfolding *cat-op-simps* **.**
qed (*auto intro*: *cat-op-intros cat-cs-intros*)
qed

6.6 Interchange law

lemma *ntcf-comp-interchange-law*:
— See Chapter II-5 in [7].
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{M}' : \mathfrak{G}' \mapsto_{CF} \mathfrak{H}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows (($\mathfrak{M}' \cdot_{NTCF} \mathfrak{N}'$) $\circ_{NTCF}$ ($\mathfrak{M} \cdot_{NTCF} \mathfrak{N}$)) = ($\mathfrak{M}' \circ_{NTCF} \mathfrak{M}$) $\cdot_{NTCF}$ ($\mathfrak{N}' \circ_{NTCF} \mathfrak{N}$)
proof–
interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{M}'$: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}'$ $\mathfrak{H}'$ $\mathfrak{M}'$ **by** (*rule assms*(*3*))
interpret $\mathfrak{N}'$: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{N}'$ **by** (*rule assms*(*4*))
show *?thesis*
proof(*rule ntcf-eqI*)
from *ntsmcf-comp-interchange-law*
[

```
      OF
        𝔐.ntcf-is-ntsmcf
        𝔑.ntcf-is-ntsmcf
        𝔐'.ntcf-is-ntsmcf
        𝔑'.ntcf-is-ntsmcf
    ]
  have
    (
      (ntcf-ntsmcf 𝔐' ⊙⇩N⇩T⇩S⇩M⇩C⇩F ntcf-ntsmcf 𝔑') ∘⇩N⇩T⇩S⇩M⇩C⇩F
      (ntcf-ntsmcf 𝔐 ⊙⇩N⇩T⇩S⇩M⇩C⇩F ntcf-ntsmcf 𝔑)
    )(⌊NTMap⌋) =
    (
      (ntcf-ntsmcf 𝔐' ∘⇩N⇩T⇩S⇩M⇩C⇩F ntcf-ntsmcf 𝔐) ⊙⇩N⇩T⇩C⇩F
      (ntcf-ntsmcf 𝔑' ∘⇩N⇩T⇩S⇩M⇩C⇩F ntcf-ntsmcf 𝔑)
    )(⌊NTMap⌋)
    by simp
  then show
    (𝔐' ⊙⇩N⇩T⇩C⇩F 𝔑' ∘⇩N⇩T⇩C⇩F (𝔐 ⊙⇩N⇩T⇩C⇩F 𝔑))(⌊NTMap⌋) =
      (𝔐' ∘⇩N⇩T⇩C⇩F 𝔐 ⊙⇩N⇩T⇩C⇩F (𝔑' ∘⇩N⇩T⇩C⇩F 𝔑))(⌊NTMap⌋)
    unfolding slicing-simps slicing-commute .
qed (auto intro: cat-cs-intros)
qed
```

6.7 Identity natural transformation

6.7.1 Definition and elementary properties

See Chapter II-4 in [7].

```
definition ntcf-id :: V ⇒ V
  where ntcf-id 𝔉 = [𝔉(⌊HomCod⌋)(⌊CId⌋) ∘⇩∘ 𝔉(⌊ObjMap⌋), 𝔉, 𝔉, 𝔉(⌊HomDom⌋), 𝔉(⌊HomCod⌋)]⇩∘
```

Components.

```
lemma ntcf-id-components:
  shows ntcf-id 𝔉(⌊NTMap⌋) = 𝔉(⌊HomCod⌋)(⌊CId⌋) ∘⇩∘ 𝔉(⌊ObjMap⌋)
    and [dg-shared-cs-simps, cat-cs-simps]: ntcf-id 𝔉(⌊NTDom⌋) = 𝔉
    and [dg-shared-cs-simps, cat-cs-simps]: ntcf-id 𝔉(⌊NTCod⌋) = 𝔉
    and [dg-shared-cs-simps, cat-cs-simps]: ntcf-id 𝔉(⌊NTDGDom⌋) = 𝔉(⌊HomDom⌋)
    and [dg-shared-cs-simps, cat-cs-simps]: ntcf-id 𝔉(⌊NTDGCod⌋) = 𝔉(⌊HomCod⌋)
  unfolding ntcf-id-def nt-field-simps by (simp-all add: nat-omega-simps)

lemma (in is-functor) is-functor-ntcf-id-components:
  shows ntcf-id 𝔉(⌊NTMap⌋) = 𝔅(⌊CId⌋) ∘⇩∘ 𝔉(⌊ObjMap⌋)
    and ntcf-id 𝔉(⌊NTDom⌋) = 𝔉
    and ntcf-id 𝔉(⌊NTCod⌋) = 𝔉
    and ntcf-id 𝔉(⌊NTDGDom⌋) = 𝔄
    and ntcf-id 𝔉(⌊NTDGCod⌋) = 𝔅
  unfolding ntcf-id-components by (simp-all add: cat-cs-simps)
```

6.7.2 Natural transformation map

```
lemma (in is-functor) ntcf-id-NTMap-vdomain[cat-cs-simps]:
  𝒟∘ (ntcf-id 𝔉(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
  using cf-ObjMap-vrange unfolding is-functor-ntcf-id-components
  by (auto simp: cat-cs-simps)

lemmas [cat-cs-simps] = is-functor.ntcf-id-NTMap-vdomain

lemma (in is-functor) ntcf-id-NTMap-app-vdomain[cat-cs-simps]:
```

assumes [*simp*]: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
shows *ntcf-id* $\mathfrak{F}(\!|NTMap|\!)(\!|a|\!) = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)|\!)$
unfolding *is-functor-ntcf-id-components*
by (*rule vsv-vcomp-at*) (*auto simp*: *cf-ObjMap-vrange cat-cs-simps cat-cs-intros*)

lemmas [*cat-cs-simps*] = *is-functor.ntcf-id-NTMap-app-vdomain*

lemma (**in** *is-functor*) *ntcf-id-NTMap-vsv*[*cat-cs-intros*]:
vsv (*ntcf-id* $\mathfrak{F}(\!|NTMap|\!)$)
unfolding *is-functor-ntcf-id-components* **by** (*auto intro*: *vsv-vcomp*)

lemmas [*cat-cs-intros*] = *is-functor.ntcf-id-NTMap-vsv*

lemma (**in** *is-functor*) *ntcf-id-NTMap-vrange*:
$\mathcal{R}_{\circ}$ (*ntcf-id* $\mathfrak{F}(\!|NTMap|\!)$) $\subseteq_{\circ} \mathfrak{B}(\!|Arr|\!)$
proof(*rule vsubsetI*)
interpret *vsv* ‹*ntcf-id* $\mathfrak{F}(\!|NTMap|\!)$› **by** (*rule ntcf-id-NTMap-vsv*)
fix *f* **assume** $f \in_{\circ} \mathcal{R}_{\circ}$ (*ntcf-id* $\mathfrak{F}(\!|NTMap|\!)$)
then obtain *a*
where *f-def*: $f =$ *ntcf-id* $\mathfrak{F}(\!|NTMap|\!)(\!|a|\!)$ **and** *a*: $a \in_{\circ} \mathcal{D}_{\circ}$ (*ntcf-id* $\mathfrak{F}(\!|NTMap|\!)$)
using *vrange-atD* **by** *metis*
then have $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ **and** $f = \mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)|\!)$
by (*auto simp*: *cat-cs-simps*)
then show $f \in_{\circ} \mathfrak{B}(\!|Arr|\!)$
by (*auto dest*: *cf-ObjMap-app-in-HomCod-Obj HomCod.cat-CId-is-arr*)
qed

6.7.3 Further properties

lemma (**in** *is-functor*) *cf-ntcf-id-is-ntcf*[*cat-cs-intros*]:
ntcf-id $\mathfrak{F} : \mathfrak{F} \mapsto_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof(*rule is-ntcfI*, *unfold is-functor-ntcf-id-components*(*2*,*3*,*4*,*5*))
show *ntcf-ntsmcf* (*ntcf-id* $\mathfrak{F}$) :
cf-smcf $\mathfrak{F} \mapsto_{SMCF}$ *cf-smcf* $\mathfrak{F}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{B}$
proof
(
rule is-ntsmcfI,
unfold slicing-simps slicing-commute is-functor-ntcf-id-components(*2*,*3*,*4*,*5*)
)
show *ntsmcf-tdghm* (*ntcf-ntsmcf* (*ntcf-id* $\mathfrak{F}$)) :
smcf-dghm (*cf-smcf* $\mathfrak{F}$) $\mapsto_{DGHM}$ *smcf-dghm* (*cf-smcf* $\mathfrak{F}$) :
smc-dg (*cat-smc* $\mathfrak{A}$) $\mapsto\mapsto_{DG\alpha}$ *smc-dg* (*cat-smc* $\mathfrak{B}$)
by
(
rule is-tdghmI,
unfold
slicing-simps
slicing-commute
is-functor-ntcf-id-components(*2*,*3*,*4*,*5*)
)
(
auto
simp:
cat-cs-simps
cat-cs-intros
nat-omega-simps
ntsmcf-tdghm-def
cf-is-semifunctor

```
        intro: slicing-intros
    )
  fix f a b assume f : a ↦_𝔄 b
  with is-functor-axioms show
    ntcf-id 𝔉(|NTMap|)(|b|) ∘_A𝔅 𝔉(|ArrMap|)(|f|) =
      𝔉(|ArrMap|)(|f|) ∘_A𝔅 ntcf-id 𝔉(|NTMap|)(|a|)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 qed (auto simp: ntcf-ntsmcf-def nat-omega-simps intro: slicing-intros)
qed (auto simp: ntcf-id-def nat-omega-simps intro: cat-cs-intros)

lemma (in is-functor) cf-ntcf-id-is-ntcf':
 assumes 𝔊' = 𝔉 and ℌ' = 𝔉
 shows ntcf-id 𝔉 : 𝔊' ↦_CF ℌ' : 𝔄 ↦↦_Cα 𝔅
 unfolding assms by (rule cf-ntcf-id-is-ntcf)

lemmas [cat-cs-intros] = is-functor.cf-ntcf-id-is-ntcf'

lemma (in is-ntcf) ntcf-ntcf-vcomp-ntcf-id-left-left[cat-cs-simps]:
 — See Chapter II-4 in [7].
 ntcf-id 𝔊 ⊙_NTCF 𝔑 = 𝔑
proof(rule ntcf-eqI[of α])
 interpret id: is-ntcf α 𝔄 𝔅 𝔊 𝔊 ‹ntcf-id 𝔊›
   by (rule NTCod.cf-ntcf-id-is-ntcf)
 show (ntcf-id 𝔊 ⊙_NTCF 𝔑)(|NTMap|) = 𝔑(|NTMap|)
 proof(rule vsv-eqI)
   show [simp]: 𝒟_∘ ((ntcf-id 𝔊 ⊙_NTCF 𝔑)(|NTMap|)) = 𝒟_∘ (𝔑(|NTMap|))
     unfolding ntsmcf-vcomp-components
     by (simp add: cat-cs-simps)
   fix a assume a ∈_∘ 𝒟_∘ ((ntcf-id 𝔊 ⊙_NTCF 𝔑)(|NTMap|))
   then have a ∈_∘ 𝔄(|Obj|) by (simp add: cat-cs-simps)
   then show (ntcf-id 𝔊 ⊙_NTCF 𝔑)(|NTMap|)(|a|) = 𝔑(|NTMap|)(|a|)
     by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 qed (auto simp: ntsmcf-vcomp-components)
qed (auto intro: cat-cs-intros)

lemmas [cat-cs-simps] = is-ntcf.ntcf-ntcf-vcomp-ntcf-id-left-left

lemma (in is-ntcf) ntcf-ntcf-vcomp-ntcf-id-right-left[cat-cs-simps]:
 — See Chapter II-4 in [7].
 𝔑 ⊙_NTCF ntcf-id 𝔉 = 𝔑
proof(rule ntcf-eqI[of α])
 interpret id: is-ntcf α 𝔄 𝔅 𝔉 𝔉 ‹ntcf-id 𝔉›
   by (rule NTDom.cf-ntcf-id-is-ntcf)
 show (𝔑 ⊙_NTCF ntcf-id 𝔉)(|NTMap|) = 𝔑(|NTMap|)
 proof(rule vsv-eqI)
   show [simp]: 𝒟_∘ ((𝔑 ⊙_NTCF ntcf-id 𝔉)(|NTMap|)) = 𝒟_∘ (𝔑(|NTMap|))
     unfolding ntsmcf-vcomp-components by (simp add: cat-cs-simps)
   fix a assume a ∈_∘ 𝒟_∘ ((𝔑 ⊙_NTCF ntcf-id 𝔉)(|NTMap|))
   then have a ∈_∘ 𝔄(|Obj|) by (simp add: cat-cs-simps)
   then show (𝔑 ⊙_NTCF ntcf-id 𝔉)(|NTMap|)(|a|) = 𝔑(|NTMap|)(|a|)
     by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 qed (auto simp: ntsmcf-vcomp-components)
qed (auto intro: cat-cs-intros)

lemmas [cat-cs-simps] = is-ntcf.ntcf-ntcf-vcomp-ntcf-id-right-left

lemma (in is-ntcf) ntcf-ntcf-hcomp-ntcf-id-left-left[cat-cs-simps]:
 — See Chapter II-5 in [7].
```

ntcf-id (*cf-id* $\mathfrak{B}$) $\circ_{NTCF}$ $\mathfrak{N}$ = $\mathfrak{N}$
proof(*rule ntcf-eqI*)
interpret *id*: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{B}$ ‹*cf-id* $\mathfrak{B}$› ‹*cf-id* $\mathfrak{B}$› ‹*ntcf-id* (*cf-id* $\mathfrak{B}$)›
by
(
simp add:
NTDom.HomCod.cat-cf-id-is-functor is-functor.cf-ntcf-id-is-ntcf
)
show *ntcf-id* (*cf-id* $\mathfrak{B}$) $\circ_{NTCF}$ $\mathfrak{N}$:
cf-id $\mathfrak{B}$ $\circ_{CF}$ $\mathfrak{F}$ $\mapsto_{CF}$ *cf-id* $\mathfrak{B}$ $\circ_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show (*ntcf-id* (*cf-id* $\mathfrak{B}$) $\circ_{NTCF}$ $\mathfrak{N}$)(|*NTMap*|) = $\mathfrak{N}$(|*NTMap*|)
proof(*rule vsv-eqI*)
fix *a* **assume** *a* $\in_{\circ}$ $\mathcal{D}_{\circ}$ ((*ntcf-id* (*cf-id* $\mathfrak{B}$) $\circ_{NTCF}$ $\mathfrak{N}$)(|*NTMap*|))
then have *a*: *a* $\in_{\circ}$ $\mathfrak{A}$(|*Obj*|)
unfolding *ntcf-hcomp-NTMap-vdomain*[*OF is-ntcf-axioms*] **by** *simp*
with *is-ntcf-axioms* **show**
(*ntcf-id* (*cf-id* $\mathfrak{B}$) $\circ_{NTCF}$ $\mathfrak{N}$)(|*NTMap*|)(|*a*|) = $\mathfrak{N}$(|*NTMap*|)(|*a*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto simp*: *ntsmcf-hcomp-components*(*1*) *cat-cs-simps*)
qed (*auto simp*: *cat-cs-simps intro*: *cat-cs-intros*)

lemmas [*cat-cs-simps*] = *is-ntcf.ntcf-ntcf-hcomp-ntcf-id-left-left*

lemma (**in** *is-ntcf*) *ntcf-ntcf-hcomp-ntcf-id-right-left*[*cat-cs-simps*]:
— See Chapter II-5 in [7].
$\mathfrak{N}$ $\circ_{NTCF}$ *ntcf-id* (*cf-id* $\mathfrak{A}$) = $\mathfrak{N}$
proof(*rule ntcf-eqI*[*of* α])
interpret *id*: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{A}$ ‹*cf-id* $\mathfrak{A}$› ‹*cf-id* $\mathfrak{A}$› ‹*ntcf-id* (*cf-id* $\mathfrak{A}$)›
by
(
simp add:
NTDom.HomDom.cat-cf-id-is-functor is-functor.cf-ntcf-id-is-ntcf
)
show $\mathfrak{N}$ $\circ_{NTCF}$ *ntcf-id* (*cf-id* $\mathfrak{A}$) :
$\mathfrak{F}$ $\circ_{CF}$ *cf-id* $\mathfrak{A}$ $\mapsto_{CF}$ $\mathfrak{G}$ $\circ_{CF}$ *cf-id* $\mathfrak{A}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show ($\mathfrak{N}$ $\circ_{NTCF}$ *ntcf-id* (*cf-id* $\mathfrak{A}$))(|*NTMap*|) = $\mathfrak{N}$(|*NTMap*|)
proof(*rule vsv-eqI*)
fix *a* **assume** *a* $\in_{\circ}$ $\mathcal{D}_{\circ}$ (($\mathfrak{N}$ $\circ_{NTCF}$ *ntcf-id* (*cf-id* $\mathfrak{A}$))(|*NTMap*|))
then have *a*: *a* $\in_{\circ}$ $\mathfrak{A}$(|*Obj*|)
unfolding *ntcf-hcomp-NTMap-vdomain*[*OF id.is-ntcf-axioms*] **by** *simp*
with *is-ntcf-axioms* **show**
($\mathfrak{N}$ $\circ_{NTCF}$ *ntcf-id* (*cf-id* $\mathfrak{A}$))(|*NTMap*|)(|*a*|) = $\mathfrak{N}$(|*NTMap*|)(|*a*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto simp*: *ntsmcf-hcomp-components*(*1*) *cat-cs-simps*)
qed (*auto simp*: *cat-cs-simps cat-cs-intros*)

lemmas [*cat-cs-simps*] = *is-ntcf.ntcf-ntcf-hcomp-ntcf-id-right-left*

6.7.4 The opposite identity natural transformation

lemma (**in** *is-functor*) *cf-ntcf-id-op-cf*: *ntcf-id* (*op-cf* $\mathfrak{F}$) = *op-ntcf* (*ntcf-id* $\mathfrak{F}$)
proof(*rule ntcf-eqI*)
show *ntcfid-op*:
ntcf-id (*op-cf* $\mathfrak{F}$) : *op-cf* $\mathfrak{F}$ $\mapsto_{CF}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{B}$
by (*simp add*: *is-functor.cf-ntcf-id-is-ntcf local.is-functor-op*)
show *ntcf-id* (*op-cf* $\mathfrak{F}$)(|*NTMap*|) = *op-ntcf* (*ntcf-id* $\mathfrak{F}$)(|*NTMap*|)

by (*rule vsv-eqI*, *unfold cat-op-simps*)
(
auto
simp: *cat-op-simps cat-cs-simps ntcf-id-components*(*1*)
intro: *vsv-vcomp*
)
qed (*auto intro*: *cat-op-intros cat-cs-intros*)

6.7.5 Identity natural transformation of a composition of functors

lemma *ntcf-id-cf-comp*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *ntcf-id* $(\mathfrak{G} \circ_{CF} \mathfrak{F})$ = *ntcf-id* $\mathfrak{G} \circ_{NTCF}$ *ntcf-id* $\mathfrak{F}$
proof(*rule ntcf-eqI*)
from *assms* **show** $\mathfrak{GF}$: *ntcf-id* $(\mathfrak{G} \circ_{CF} \mathfrak{F}) : \mathfrak{G} \circ_{CF} \mathfrak{F} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
interpret $\mathfrak{GF}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› ‹*ntcf-id* $(\mathfrak{G} \circ_{CF} \mathfrak{F})$›
by (*rule* $\mathfrak{GF}$)
from *assms* **show** $\mathfrak{G}$-$\mathfrak{F}$:
ntcf-id $\mathfrak{G} \circ_{NTCF}$ *ntcf-id* $\mathfrak{F} : \mathfrak{G} \circ_{CF} \mathfrak{F} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
interpret $\mathfrak{G}$-$\mathfrak{F}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› ‹*ntcf-id* $\mathfrak{G} \circ_{NTCF}$ *ntcf-id* $\mathfrak{F}$›
by (*rule* $\mathfrak{G}$-$\mathfrak{F}$)
show *ntcf-id* $(\mathfrak{G} \circ_{CF} \mathfrak{F})(\!|NTMap|\!)$ = $(ntcf\text{-}id\ \mathfrak{G} \circ_{NTCF} ntcf\text{-}id\ \mathfrak{F})(\!|NTMap|\!)$
proof(*rule vsv-eqI*, *unfold* $\mathfrak{GF}$.*ntcf-NTMap-vdomain* $\mathfrak{G}$-$\mathfrak{F}$.*ntcf-NTMap-vdomain*)
fix *a* **assume** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
with *assms* **show**
ntcf-id $(\mathfrak{G} \circ_{CF} \mathfrak{F})(\!|NTMap|\!)(\!|a|\!)$ = $(ntcf\text{-}id\ \mathfrak{G} \circ_{NTCF} ntcf\text{-}id\ \mathfrak{F})(\!|NTMap|\!)(\!|a|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed *auto*
qed *auto*

lemmas [*cat-cs-simps*] = *ntcf-id-cf-comp*[*symmetric*]

6.8 Composition of a natural transformation and a functor

6.8.1 Definition and elementary properties

abbreviation (*input*) *ntcf-cf-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\circ_{NTCF\text{-}CF}$› *55*)
where *ntcf-cf-comp* ≡ *tdghm-dghm-comp*

Slicing.

lemma *ntsmcf-tdghm-ntsmcf-smcf-comp*[*slicing-commute*]:
ntcf-ntsmcf $\mathfrak{N} \circ_{NTSMCF\text{-}SMCF}$ *cf-smcf* $\mathfrak{H}$ = *ntcf-ntsmcf* $(\mathfrak{N} \circ_{NTCF\text{-}CF} \mathfrak{H})$
unfolding
ntcf-ntsmcf-def
cf-smcf-def
cat-smc-def
tdghm-dghm-comp-def
dghm-comp-def
ntsmcf-tdghm-def
smcf-dghm-def
smc-dg-def
dg-field-simps
dghm-field-simps
nt-field-simps
by (*simp add*: *nat-omega-simps*)

6.8.2 Natural transformation map

mk-VLambda (**in** *is-functor*)
tdghm-dghm-comp-components(*1*)[**where** $\mathfrak{H}$=$\mathfrak{F}$, *unfolded cf-HomDom*]
|*vdomain ntcf-cf-comp-NTMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-cf-comp-NTMap-app*[*cat-cs-simps*]|

lemmas [*cat-cs-simps*] =
is-functor.ntcf-cf-comp-NTMap-vdomain
is-functor.ntcf-cf-comp-NTMap-app

lemma *ntcf-cf-comp-NTMap-vrange*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H})(\!|NTMap|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!)$
proof–
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{H}$ **by** (*rule assms*(*2*))
show *?thesis* **unfolding** *tdghm-dghm-comp-components*
by (*auto simp*: *cat-cs-simps intro*: *cat-cs-intros*)
qed

6.8.3 Opposite of the composition of a natural transformation and a functor

lemma *op-ntcf-ntcf-cf-comp*[*cat-op-simps*]:
op-ntcf $(\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H})$ = *op-ntcf* $\mathfrak{N} \circ_{NTCF-CF}$ *op-cf* $\mathfrak{H}$
unfolding
tdghm-dghm-comp-def
dghm-comp-def
op-ntcf-def
op-cf-def
op-cat-def
dg-field-simps
dghm-field-simps
nt-field-simps
by (*simp add*: *nat-omega-simps*)

6.8.4 Composition of a natural transformation and a functor is a natural transformation

lemma *ntcf-cf-comp-is-ntcf*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H} : \mathfrak{F} \circ_{CF} \mathfrak{H} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{H}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule is-ntcfI*)
show *vfsequence* $(\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H})$
unfolding *tdghm-dghm-comp-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** $\mathfrak{F} \circ_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
from *assms* **show** $\mathfrak{G} \circ_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
show *vcard* $(\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H}) = 5_{\mathbf{N}}$
unfolding *tdghm-dghm-comp-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show**
ntcf-ntsmcf $(\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H})$:
cf-smcf $(\mathfrak{F} \circ_{CF} \mathfrak{H}) \mapsto_{SMCF}$ *cf-smcf* $(\mathfrak{G} \circ_{CF} \mathfrak{H})$:
cat-smc $\mathfrak{A} \mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{C}$

```
    by
      (
        cs-concl
          cs-simp: slicing-commute[symmetric]
          cs-intro: slicing-intros smc-cs-intros cat-cs-intros
      )
  qed (auto simp: tdghm-dghm-comp-components(1) cat-cs-simps)
qed

lemma ntcf-cf-comp-is-ntcf'[cat-cs-intros]:
  assumes 𝔑 : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα ℭ
    and ℌ : 𝔄 ↦↦Cα 𝔅
    and 𝔉' = 𝔉 ∘CF ℌ
    and 𝔊' = 𝔊 ∘CF ℌ
  shows 𝔑 ∘NTCF-CF ℌ : 𝔉' ↦CF 𝔊' : 𝔄 ↦↦Cα ℭ
  using assms(1,2) unfolding assms(3,4) by (simp add: ntcf-cf-comp-is-ntcf)
```

6.8.5 Further properties

```
lemma ntcf-cf-comp-ntcf-cf-comp-assoc:
  assumes 𝔑 : ℌ ↦CF ℌ' : ℭ ↦↦Cα 𝔇
    and 𝔊 : 𝔅 ↦↦Cα ℭ
    and 𝔉 : 𝔄 ↦↦Cα 𝔅
  shows (𝔑 ∘NTCF-CF 𝔊) ∘NTCF-CF 𝔉 = 𝔑 ∘NTCF-CF (𝔊 ∘CF 𝔉)
proof–
  interpret 𝔑: is-ntcf α ℭ 𝔇 ℌ ℌ' 𝔑 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  interpret 𝔉: is-functor α 𝔄 𝔅 𝔉 by (rule assms(3))
  show ?thesis
  proof(rule ntcf-ntsmcf-eqI)
    from assms show
      (𝔑 ∘NTCF-CF 𝔊) ∘NTCF-CF 𝔉 :
        ℌ ∘CF 𝔊 ∘CF 𝔉 ↦CF ℌ' ∘CF 𝔊 ∘CF 𝔉 : 𝔄 ↦↦Cα 𝔇
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show 𝔑 ∘NTCF-CF (𝔊 ∘CF 𝔉) :
      ℌ ∘CF 𝔊 ∘CF 𝔉 ↦CF ℌ' ∘CF 𝔊 ∘CF 𝔉 : 𝔄 ↦↦Cα 𝔇
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms show
      ntcf-ntsmcf ((𝔑 ∘NTCF-CF 𝔊) ∘NTCF-CF 𝔉) =
        ntcf-ntsmcf (𝔑 ∘NTCF-CF (𝔊 ∘CF 𝔉))
      by
        (
          cs-concl
            cs-simp: slicing-commute[symmetric]
            cs-intro: slicing-intros ntsmcf-smcf-comp-ntsmcf-smcf-comp-assoc
        )
  qed simp-all
qed

lemma (in is-ntcf) ntcf-ntcf-cf-comp-cf-id[cat-cs-simps]:
  𝔑 ∘NTCF-CF cf-id 𝔄 = 𝔑
proof(rule ntcf-ntsmcf-eqI)
  show 𝔑 ∘NTCF-CF cf-id 𝔄 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ntcf-ntsmcf (𝔑 ∘NTCF-CF cf-id 𝔄) = ntcf-ntsmcf 𝔑
    by
```

(
cs-concl **cs-shallow**
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *cat-cs-intros slicing-intros smc-cs-simps*
)
qed *simp-all*

lemmas [*cat-cs-simps*] = *is-ntcf.ntcf-ntcf-cf-comp-cf-id*

lemma *ntcf-vcomp-ntcf-cf-comp*[*cat-cs-simps*]:
assumes $\mathfrak{K} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $(\mathfrak{M} \circ_{NTCF-CF} \mathfrak{K}) \cdot_{NTCF} (\mathfrak{N} \circ_{NTCF-CF} \mathfrak{K}) = (\mathfrak{M} \cdot_{NTCF} \mathfrak{N}) \circ_{NTCF-CF} \mathfrak{K}$
proof(*rule ntcf-ntsmcf-eqI*)
from *assms* **show**
$\mathfrak{M} \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} (\mathfrak{N} \circ_{NTCF-CF} \mathfrak{K}) :$
$\mathfrak{F} \circ_{CF} \mathfrak{K} \mapsto_{CF} \mathfrak{H} \circ_{CF} \mathfrak{K} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **show**
ntcf-ntsmcf $(\mathfrak{M} \circ_{NTCF-CF} \mathfrak{K} \cdot_{NTCF} (\mathfrak{N} \circ_{NTCF-CF} \mathfrak{K})) =$
ntcf-ntsmcf $(\mathfrak{M} \cdot_{NTCF} \mathfrak{N} \circ_{NTCF-CF} \mathfrak{K})$
unfolding *slicing-commute*[*symmetric*]
by (*intro ntsmcf-vcomp-ntsmcf-smcf-comp*)
(*cs-concl* **cs-intro**: *slicing-intros*)
qed (*use assms* **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+

6.9 Composition of a functor and a natural transformation

6.9.1 Definition and elementary properties

abbreviation (*input*) *cf-ntcf-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\circ_{CF-NTCF}$› *55*)
where *cf-ntcf-comp* ≡ *dghm-tdghm-comp*

Slicing.

lemma *ntcf-ntsmcf-cf-ntcf-comp*[*slicing-commute*]:
cf-smcf $\mathfrak{H} \circ_{SMCF-NTSMCF}$ *ntcf-ntsmcf* $\mathfrak{N}$ = *ntcf-ntsmcf* $(\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N})$
unfolding
ntcf-ntsmcf-def
cf-smcf-def
cat-smc-def
dghm-tdghm-comp-def
dghm-comp-def
ntsmcf-tdghm-def
smcf-dghm-def
smc-dg-def
dg-field-simps
dghm-field-simps
nt-field-simps
by (*simp add*: *nat-omega-simps*)

6.9.2 Natural transformation map

mk-VLambda (**in** *is-ntcf*)
dghm-tdghm-comp-components(*1*)[**where** $\mathfrak{N}$=$\mathfrak{N}$, *unfolded ntcf-NTDGDom*]
|*vdomain cf-ntcf-comp-NTMap-vdomain*|
|*app cf-ntcf-comp-NTMap-app*|

lemmas [*cat-cs-simps*] =

is-ntcf.cf-ntcf-comp-NTMap-vdomain
is-ntcf.cf-ntcf-comp-NTMap-app

lemma *cf-ntcf-comp-NTMap-vrange*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ\ ((\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N})(\lfloor NTMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof–
interpret $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))
show *?thesis*
unfolding *dghm-tdghm-comp-components*
by (*auto simp*: *cat-cs-simps intro*: *cat-cs-intros*)
qed

6.9.3 Opposite of the composition of a functor and a natural transformation

lemma *op-ntcf-cf-ntcf-comp*[*cat-op-simps*]:
op-ntcf $(\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N})$ = *op-cf* $\mathfrak{H} \circ_{CF-NTCF}$ *op-ntcf* $\mathfrak{N}$
unfolding
dghm-tdghm-comp-def
dghm-comp-def
op-ntcf-def
op-cf-def
op-cat-def
dg-field-simps
dghm-field-simps
nt-field-simps
by (*simp add*: *nat-omega-simps*)

6.9.4 Composition of a functor and a natural transformation is a natural transformation

lemma *cf-ntcf-comp-is-ntcf*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{H} \circ_{CF} \mathfrak{F} \mapsto_{CF} \mathfrak{H} \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule is-ntcfI*)
show *vfsequence* $(\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N})$ **unfolding** *dghm-tdghm-comp-def* **by** *simp*
from *assms* **show** $\mathfrak{H} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
from *assms* **show** $\mathfrak{H} \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
show *vcard* $(\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N}) = 5_{\mathbf{N}}$
unfolding *dghm-tdghm-comp-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** *ntcf-ntsmcf* $(\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N})$:
cf-smcf $(\mathfrak{H} \circ_{CF} \mathfrak{F})$ $\mapsto_{SMCF}$ *cf-smcf* $(\mathfrak{H} \circ_{CF} \mathfrak{G})$:
cat-smc $\mathfrak{A}$ $\mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{C}$
by
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *slicing-intros smc-cs-intros*
)
qed (*auto simp*: *dghm-tdghm-comp-components*(*1*) *cat-cs-simps*)
qed

lemma *cf-ntcf-comp-is-functor'*[*cat-cs-intros*]:
  assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{F}' = \mathfrak{H} \circ_{CF} \mathfrak{F}$
    and $\mathfrak{G}' = \mathfrak{H} \circ_{CF} \mathfrak{G}$
  shows $\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*simp add*: *cf-ntcf-comp-is-ntcf*)

6.9.5 Further properties

lemma *cf-comp-cf-ntcf-comp-assoc*:
  assumes $\mathfrak{N} : \mathfrak{H} \mapsto_{CF} \mathfrak{H}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
  shows $(\mathfrak{G} \circ_{CF} \mathfrak{F}) \circ_{CF-NTCF} \mathfrak{N} = \mathfrak{G} \circ_{CF-NTCF} (\mathfrak{F} \circ_{CF-NTCF} \mathfrak{N})$
proof(*rule ntcf-ntsmcf-eqI*)
  interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{H}\ \mathfrak{H}'\ \mathfrak{N}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}$ **by** (*rule assms*(*2*))
  interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{G}$ **by** (*rule assms*(*3*))
  from *assms* **show** $(\mathfrak{G} \circ_{CF} \mathfrak{F}) \circ_{CF-NTCF} \mathfrak{N} :$
    $\mathfrak{G} \circ_{CF} \mathfrak{F} \circ_{CF} \mathfrak{H} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} \circ_{CF} \mathfrak{H}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros*)
  from *assms* **show** $\mathfrak{G} \circ_{CF-NTCF} (\mathfrak{F} \circ_{CF-NTCF} \mathfrak{N}) :$
    $\mathfrak{G} \circ_{CF} \mathfrak{F} \circ_{CF} \mathfrak{H} \mapsto_{CF} \mathfrak{G} \circ_{CF} \mathfrak{F} \circ_{CF} \mathfrak{H}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *assms* **show**
    ntcf-ntsmcf $(\mathfrak{G} \circ_{CF} \mathfrak{F} \circ_{CF-NTCF} \mathfrak{N}) =$
      ntcf-ntsmcf $(\mathfrak{G} \circ_{CF-NTCF} (\mathfrak{F} \circ_{CF-NTCF} \mathfrak{N}))$
    by
      (
        cs-concl
          cs-simp: *slicing-commute*[*symmetric*]
          cs-intro: *slicing-intros smcf-comp-smcf-ntsmcf-comp-assoc*
)
qed *simp-all*

lemma (**in** *is-ntcf*) *ntcf-cf-ntcf-comp-cf-id*[*cat-cs-simps*]:
  cf-id $\mathfrak{B} \circ_{CF-NTCF} \mathfrak{N} = \mathfrak{N}$
proof(*rule ntcf-ntsmcf-eqI*)
  show *cf-id* $\mathfrak{B} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
  show *ntcf-ntsmcf* (*smcf-id* $\mathfrak{B} \circ_{SMCF-NTSMCF} \mathfrak{N}$) = *ntcf-ntsmcf* $\mathfrak{N}$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *slicing-commute*[*symmetric*]
          cs-intro: *cat-cs-intros slicing-intros smc-cs-simps*
)
qed *simp-all*

lemmas [*cat-cs-simps*] = *is-ntcf.ntcf-cf-ntcf-comp-cf-id*

lemma *cf-ntcf-comp-ntcf-cf-comp-assoc*:
  assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{H} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$

```
    and 𝔎 : 𝔄 ↦↦_Cα 𝔅
  shows (𝔊 ∘_CF-NTCF 𝔑) ∘_NTCF-CF 𝔎 = 𝔊 ∘_CF-NTCF (𝔑 ∘_NTCF-CF 𝔎)
proof–
  interpret 𝔑: is-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret 𝔊: is-functor α ℭ 𝔇 𝔊 by (rule assms(2))
  interpret 𝔎: is-functor α 𝔄 𝔅 𝔎 by (rule assms(3))
  show ?thesis
    by (rule ntcf-ntsmcf-eqI)
      (
        use assms in
          ‹
            cs-concl
              cs-simp: cat-cs-simps slicing-commute[symmetric]
              cs-intro:
                cat-cs-intros
                slicing-intros
                smcf-ntsmcf-comp-ntsmcf-smcf-comp-assoc
          ›
      )+
qed

lemma ntcf-cf-comp-ntcf-id[cat-cs-simps]:
  assumes 𝔉 : 𝔅 ↦↦_Cα ℭ and 𝔎 : 𝔄 ↦↦_Cα 𝔅
  shows ntcf-id 𝔉 ∘_NTCF-CF 𝔎 = ntcf-id 𝔉 ∘_NTCF ntcf-id 𝔎
proof(rule ntcf-eqI)
  from assms have dom-lhs: 𝒟_∘ ((ntcf-id 𝔉 ∘_NTCF-CF 𝔎)(‖NTMap‖)) = 𝔄(‖Obj‖)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms have dom-rhs: 𝒟_∘ ((ntcf-id 𝔉 ∘_NTCF ntcf-id 𝔎)(‖NTMap‖)) = 𝔄(‖Obj‖)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show (ntcf-id 𝔉 ∘_NTCF-CF 𝔎)(‖NTMap‖) = (ntcf-id 𝔉 ∘_NTCF ntcf-id 𝔎)(‖NTMap‖)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume a ∈_∘ 𝔄(‖Obj‖)
    with assms show
      (ntcf-id 𝔉 ∘_NTCF-CF 𝔎)(‖NTMap‖)(‖a‖) = (ntcf-id 𝔉 ∘_NTCF ntcf-id 𝔎)(‖NTMap‖)(‖a‖)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed (auto intro: cat-cs-intros)
qed (use assms in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+

lemma cf-ntcf-comp-ntcf-vcomp:
  assumes 𝔎 : 𝔅 ↦↦_Cα ℭ
    and 𝔐 : 𝔊 ↦_CF 𝔎 : 𝔄 ↦↦_Cα 𝔅
    and 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
  shows 𝔎 ∘_CF-NTCF (𝔐 ⊙_NTCF 𝔑) = (𝔎 ∘_CF-NTCF 𝔐) ⊙_NTCF (𝔎 ∘_CF-NTCF 𝔑)
proof–
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔐: is-ntcf α 𝔄 𝔅 𝔊 𝔎 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(3))
  show 𝔎 ∘_CF-NTCF (𝔐 ⊙_NTCF 𝔑) = 𝔎 ∘_CF-NTCF 𝔐 ⊙_NTCF (𝔎 ∘_CF-NTCF 𝔑)
    by (rule ntcf-ntsmcf-eqI)
      (
        use assms in
          ‹
            cs-concl
              cs-simp: smc-cs-simps slicing-commute[symmetric]
              cs-intro:
                cat-cs-intros
                slicing-intros
                smcf-ntsmcf-comp-ntsmcf-vcomp
```

›
)+
qed

6.10 Constant natural transformation

6.10.1 Definition and elementary properties

See Chapter III in [7].

definition *ntcf-const* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f =
[
vconst-on ($\mathfrak{J}$(|*Obj*|)) f,
cf-const $\mathfrak{J}$ $\mathfrak{C}$ ($\mathfrak{C}$(|*Dom*|)(|f|)),
cf-const $\mathfrak{J}$ $\mathfrak{C}$ ($\mathfrak{C}$(|*Cod*|)(|f|)),
$\mathfrak{J}$,
$\mathfrak{C}$
]$_\circ$

Components.

lemma *ntcf-const-components*:
shows *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTMap*|) = *vconst-on* ($\mathfrak{J}$(|*Obj*|)) f
and *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTDom*|) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ ($\mathfrak{C}$(|*Dom*|)(|f|))
and *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTCod*|) = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ ($\mathfrak{C}$(|*Cod*|)(|f|))
and *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTDGDom*|) = $\mathfrak{J}$
and *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTDGCod*|) = $\mathfrak{C}$
unfolding *ntcf-const-def nt-field-simps* **by** (*auto simp*: *nat-omega-simps*)

6.10.2 Natural transformation map

mk-VLambda *ntcf-const-components*(*1*)[*folded VLambda-vconst-on*]
|*vsv ntcf-const-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-const-ObjMap-vdomain*[*cat-cs-simps*]|
|*app ntcf-const-ObjMap-app*[*cat-cs-simps*]|

lemma *ntcf-const-NTMap-ne-vrange*:
assumes $\mathfrak{J}$(|*Obj*|) $\neq$ *0*
shows $\mathcal{R}_\circ$ (*ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTMap*|)) = *set* {f}
using *assms* **unfolding** *ntcf-const-components* **by** *simp*

lemma *ntcf-const-NTMap-vempty-vrange*:
assumes $\mathfrak{J}$(|*Obj*|) = *0*
shows $\mathcal{R}_\circ$ (*ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f(|*NTMap*|)) = *0*
using *assms* **unfolding** *ntcf-const-components* **by** *simp*

6.10.3 Constant natural transformation is a natural transformation

lemma *ntcf-const-is-ntcf*:
assumes *category* α $\mathfrak{J}$ **and** *category* α $\mathfrak{C}$ **and** $f : a \mapsto_{\mathfrak{C}} b$
shows *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f : *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ $a \mapsto_{CF}$ *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ b : $\mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{J}$: *category* α $\mathfrak{J}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*intro is-ntcfI'*)
show *vfsequence* (*ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f) **unfolding** *ntcf-const-def* **by** *auto*
show *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ a : $\mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*rule cf-const-is-functor*)

```
    from assms(3) show a ∈∘ ℭ(|Obj|) by (simp add: cat-cs-intros)
  qed (auto simp: cat-cs-intros)
  from assms(3) show const-b-is-functor:
    cf-const 𝔍 ℭ b : 𝔍 ↦↦_{Cα} ℭ
    by (auto intro: cf-const-is-functor cat-cs-intros)
  show vcard (ntcf-const 𝔍 ℭ f) = 5_ℕ
    unfolding ntcf-const-def by (simp add: nat-omega-simps)
  show
    ntcf-const 𝔍 ℭ f(|NTMap|)(|a'|) :
      cf-const 𝔍 ℭ a(|ObjMap|)(|a'|) ↦_ℭ cf-const 𝔍 ℭ b(|ObjMap|)(|a'|)
    if a' ∈∘ 𝔍(|Obj|) for a'
    by (simp add: that assms(3) ntcf-const-components(1) dghm-const-ObjMap-app)
  from assms(3) show
    ntcf-const 𝔍 ℭ f(|NTMap|)(|b'|) ∘_{Aℭ} cf-const 𝔍 ℭ a(|ArrMap|)(|f'|) =
      cf-const 𝔍 ℭ b (|ArrMap|)(|f'|) ∘_{Aℭ} ntcf-const 𝔍 ℭ f(|NTMap|)(|a'|)
    if f' : a' ↦_𝔍 b' for f' a' b'
    using that dghm-const-ArrMap-app
    by (auto simp: ntcf-const-components cat-cs-intros cat-cs-simps)
qed (use assms(3) in ‹auto simp: ntcf-const-components›)
qed

lemma ntcf-const-is-ntcf'[cat-cs-intros]:
  assumes category α 𝔍
    and category α ℭ
    and f : a ↦_ℭ b
    and 𝔄 = cf-const 𝔍 ℭ a
    and 𝔅 = cf-const 𝔍 ℭ b
    and 𝔍' = 𝔍
    and ℭ' = ℭ
  shows ntcf-const 𝔍 ℭ f : 𝔄 ↦_{CF} 𝔅 : 𝔍' ↦↦_{Cα} ℭ'
  using assms(1–3) unfolding assms(4–7) by (rule ntcf-const-is-ntcf)
```

6.10.4 Opposite constant natural transformation

```
lemma op-ntcf-ntcf-const[cat-op-simps]:
  op-ntcf (ntcf-const 𝔍 ℭ f) = ntcf-const (op-cat 𝔍) (op-cat ℭ) f
  unfolding
    nt-field-simps dghm-field-simps dg-field-simps
    dghm-const-def ntcf-const-def op-cat-def op-cf-def op-ntcf-def
  by (simp-all add: nat-omega-simps)
```

6.10.5 Further properties

```
lemma ntcf-const-ntcf-vcomp[cat-cs-simps]:
  assumes category α 𝔍
    and category α ℭ
    and g : b ↦_ℭ c
    and f : a ↦_ℭ b
  shows ntcf-const 𝔍 ℭ g ⋅_{NTCF} ntcf-const 𝔍 ℭ f = ntcf-const 𝔍 ℭ (g ∘_{Aℭ} f)
proof–
  interpret 𝔍: category α 𝔍 by (rule assms(1))
  interpret ℭ: category α ℭ by (rule assms(2))
  from assms(3,4) have gf: g ∘_{Aℭ} f : a ↦_ℭ c by (simp add: cat-cs-intros)
  note 𝔍ℭg = ntcf-const-is-ntcf[OF assms(1,2,3)]
    and 𝔍ℭf = ntcf-const-is-ntcf[OF assms(1,2,4)]
  show ?thesis
  proof(rule ntcf-eqI)
    from ntcf-const-is-ntcf[OF assms(1,2,3)] ntcf-const-is-ntcf[OF assms(1,2,4)]
```

```
    show
      ntcf-const 𝔍 𝔈 g ⊙NTCF ntcf-const 𝔍 𝔈 f :
        cf-const 𝔍 𝔈 a ↦CF cf-const 𝔍 𝔈 c : 𝔍 ↦↦Cα 𝔈
      by (rule ntcf-vcomp-is-ntcf)
    show
      ntcf-const 𝔍 𝔈 (g ∘A𝔈 f) :
        cf-const 𝔍 𝔈 a ↦CF cf-const 𝔍 𝔈 c : 𝔍 ↦↦Cα 𝔈
      by (rule ntcf-const-is-ntcf[OF assms(1,2) gf])
    show (ntcf-const 𝔍 𝔈 g ⊙NTCF ntcf-const 𝔍 𝔈 f)(|NTMap|) =
      ntcf-const 𝔍 𝔈 (g ∘A𝔈 f)(|NTMap|)
      unfolding ntcf-const-components
    proof(rule vsv-eqI, unfold ntcf-vcomp-NTMap-vdomain[OF 𝔍𝔈f])
      fix a assume prems: a ∈∘ 𝔍(|Obj|)
      then show
        (ntcf-const 𝔍 𝔈 g ⊙NTCF ntcf-const 𝔍 𝔈 f)(|NTMap|)(|a|) =
          vconst-on (𝔍(|Obj|)) (g ∘A𝔈 f)(|a|)
        unfolding ntcf-vcomp-NTMap-app[OF 𝔍𝔈g 𝔍𝔈f prems]
        by (simp add: ntcf-const-components)
    qed (simp-all add: ntsmcf-vcomp-components)
  qed auto
qed

lemma ntcf-id-cf-const[cat-cs-simps]:
  assumes category α 𝔍 and category α 𝔈 and c ∈∘ 𝔈(|Obj|)
  shows ntcf-id (cf-const 𝔍 𝔈 c) = ntcf-const 𝔍 𝔈 (𝔈(|CId|)(|c|))
proof(rule ntcf-eqI)
  interpret 𝔍: category α 𝔍 by (rule assms(1))
  interpret 𝔈: category α 𝔈 by (rule assms(2))
  from assms show ntcf-const 𝔍 𝔈 (𝔈(|CId|)(|c|)) :
    cf-const 𝔍 𝔈 c ↦CF cf-const 𝔍 𝔈 c : 𝔍 ↦↦Cα 𝔈
    by (auto intro: ntcf-const-is-ntcf)
  interpret const-c: is-functor α 𝔍 𝔈 ‹cf-const 𝔍 𝔈 c›
    by (rule cf-const-is-functor) (auto simp: assms(3) cat-cs-intros)
  show ntcf-id (cf-const 𝔍 𝔈 c) :
    cf-const 𝔍 𝔈 c ↦CF cf-const 𝔍 𝔈 c : 𝔍 ↦↦Cα 𝔈
    by (rule const-c.cf-ntcf-id-is-ntcf)
  show ntcf-id (cf-const 𝔍 𝔈 c)(|NTMap|) = ntcf-const 𝔍 𝔈 (𝔈(|CId|)(|c|))(|NTMap|)
  proof(rule vsv-eqI, unfold ntcf-const-components)
    show vsv (ntcf-id (cf-const 𝔍 𝔈 c)(|NTMap|))
      unfolding ntcf-id-components by (auto simp: cat-cs-simps intro: vsv-vcomp)
  qed (auto simp: cat-cs-simps)
qed simp-all

lemma ntcf-cf-comp-cf-const-right[cat-cs-simps]:
  assumes 𝔑 : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα 𝔈
    and category α 𝔄
    and b ∈∘ 𝔅(|Obj|)
  shows 𝔑 ∘NTCF−CF cf-const 𝔄 𝔅 b = ntcf-const 𝔄 𝔈 (𝔑(|NTMap|)(|b|))
proof–
  interpret 𝔑: is-ntcf α 𝔅 𝔈 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret 𝔄: category α 𝔄 by (rule assms(2))
  show ?thesis
  proof(rule ntcf-eqI)
    from assms(3) show 𝔑 ∘NTCF−CF cf-const 𝔄 𝔅 b :
      cf-const 𝔄 𝔈 (𝔉(|ObjMap|)(|b|)) ↦CF cf-const 𝔄 𝔈 (𝔊(|ObjMap|)(|b|)) :
      𝔄 ↦↦Cα 𝔈
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3) show ntcf-const 𝔄 𝔈 (𝔑(|NTMap|)(|b|)) :
```

```
      cf-const 𝔄 ℭ (ℱ(|ObjMap|)(|b|)) ↦_CF cf-const 𝔄 ℭ (𝔊(|ObjMap|)(|b|)) :
      𝔄 ↦↦_Cα ℭ
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3) have dom-lhs:
      𝒟∘ ((𝔑 ∘_NTCF-CF cf-const 𝔄 𝔅 b)(|NTMap|)) = 𝔄(|Obj|)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(3) have dom-rhs:
      𝒟∘ (ntcf-const 𝔄 ℭ (𝔑(|NTMap|)(|b|))(|NTMap|)) = 𝔄(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show
      (𝔑 ∘_NTCF-CF cf-const 𝔄 𝔅 b)(|NTMap|) = ntcf-const 𝔄 ℭ (𝔑(|NTMap|)(|b|))(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈∘ 𝔄(|Obj|)
      with assms(3) show
        (𝔑 ∘_NTCF-CF cf-const 𝔄 𝔅 b)(|NTMap|)(|a|) =
          ntcf-const 𝔄 ℭ (𝔑(|NTMap|)(|b|))(|NTMap|)(|a|)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed (auto intro: cat-cs-intros)
  qed simp-all
qed

lemma cf-ntcf-comp-ntcf-id[cat-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦_Cα ℭ and ℱ : 𝔄 ↦↦_Cα 𝔅
  shows 𝔊 ∘_CF-NTCF ntcf-id ℱ = ntcf-id 𝔊 ∘_NTCF ntcf-id ℱ
proof–
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret ℱ: is-functor α 𝔄 𝔅 ℱ by (rule assms(2))
  show ?thesis
  proof(rule ntcf-eqI)
    show 𝔊 ∘_CF-NTCF ntcf-id ℱ : 𝔊 ∘_CF ℱ ↦_CF 𝔊 ∘_CF ℱ : 𝔄 ↦↦_Cα ℭ
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show ntcf-id 𝔊 ∘_NTCF ntcf-id ℱ : 𝔊 ∘_CF ℱ ↦_CF 𝔊 ∘_CF ℱ : 𝔄 ↦↦_Cα ℭ
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    have dom-lhs: 𝒟∘ ((𝔊 ∘_CF-NTCF ntcf-id ℱ)(|NTMap|)) = 𝔄(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    have dom-rhs: 𝒟∘ ((ntcf-id 𝔊 ∘_NTCF ntcf-id ℱ)(|NTMap|)) = 𝔄(|Obj|)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show (𝔊 ∘_CF-NTCF ntcf-id ℱ)(|NTMap|) = (ntcf-id 𝔊 ∘_NTCF ntcf-id ℱ)(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈∘ 𝔄(|Obj|)
      then show
        (𝔊 ∘_CF-NTCF ntcf-id ℱ)(|NTMap|)(|a|) =
          (ntcf-id 𝔊 ∘_NTCF ntcf-id ℱ)(|NTMap|)(|a|)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed (cs-concl cs-intro: cat-cs-intros)
  qed simp-all
qed

lemma (in is-functor) cf-ntcf-cf-comp-ntcf-const[cat-cs-simps]:
  assumes category α ℭ and f : a ↦_ℭ b
  shows ntcf-const 𝔅 ℭ f ∘_NTCF-CF ℱ = ntcf-const 𝔄 ℭ f
proof(rule ntcf-eqI)
  interpret ℭ: category α ℭ by (rule assms(1))
  from assms(2) show ntcf-const 𝔅 ℭ f ∘_NTCF-CF ℱ :
    cf-const 𝔄 ℭ a ↦_CF cf-const 𝔄 ℭ b : 𝔄 ↦↦_Cα ℭ
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  then have dom-lhs: 𝒟∘ ((ntcf-const 𝔅 ℭ f ∘_NTCF-CF ℱ)(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
```

from *assms*(*2*) **show**
ntcf-const $\mathfrak{A}$ $\mathfrak{C}$ f : *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ a $\mapsto_{CF}$ *cf-const* $\mathfrak{A}$ $\mathfrak{C}$ b : $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_\circ$ (*ntcf-const* $\mathfrak{A}$ $\mathfrak{C}$ f(⌊*NTMap*⌋)) = $\mathfrak{A}$(⌊*Obj*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
show (*ntcf-const* $\mathfrak{B}$ $\mathfrak{C}$ f $\circ_{NTCF-CF}$ $\mathfrak{F}$)(⌊*NTMap*⌋) = *ntcf-const* $\mathfrak{A}$ $\mathfrak{C}$ f(⌊*NTMap*⌋)
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix a **assume** a $\in_\circ$ $\mathfrak{A}$(⌊*Obj*⌋)
then show
(*ntcf-const* $\mathfrak{B}$ $\mathfrak{C}$ f $\circ_{NTCF-CF}$ $\mathfrak{F}$)(⌊*NTMap*⌋)(⌊a⌋) = *ntcf-const* $\mathfrak{A}$ $\mathfrak{C}$ f(⌊*NTMap*⌋)(⌊a⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed *simp-all*

lemmas [*cat-cs-simps*] = *is-functor.cf-ntcf-cf-comp-ntcf-const*

lemma (**in** *is-functor*) *cf-ntcf-comp-cf-ntcf-const*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{J}$
and f : r' $\mapsto_{\mathfrak{A}}$ r
shows $\mathfrak{F}$ $\circ_{CF-NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{A}$ f = *ntcf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ArrMap*⌋)(⌊f⌋))
proof(*rule ntcf-eqI*)
interpret $\mathfrak{J}$: *category* α $\mathfrak{J}$ **by** (*rule assms*(*1*))
from *assms*(*2*) **have** r': r' $\in_\circ$ $\mathfrak{A}$(⌊*Obj*⌋) **and** r: r $\in_\circ$ $\mathfrak{A}$(⌊*Obj*⌋) **by** *auto*
from *assms*(*2*) **show** $\mathfrak{F}$ $\circ_{CF-NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{A}$ f :
cf-const $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ObjMap*⌋)(⌊r'⌋)) $\mapsto_{CF}$ *cf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ObjMap*⌋)(⌊r⌋)) :
$\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
with *assms*(*2*) **have** *dom-lhs*:
$\mathcal{D}_\circ$ (($\mathfrak{F}$ $\circ_{CF-NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{A}$ f)(⌊*NTMap*⌋)) = $\mathfrak{J}$(⌊*Obj*⌋)
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
from *assms*(*2*) **show** *ntcf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ArrMap*⌋)(⌊f⌋)) :
cf-const $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ObjMap*⌋)(⌊r'⌋)) $\mapsto_{CF}$ *cf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ObjMap*⌋)(⌊r⌋)) :
$\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
with *assms*(*2*) **have** *dom-rhs*:
$\mathcal{D}_\circ$ (*ntcf-const* $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ArrMap*⌋)(⌊f⌋))(⌊*NTMap*⌋)) = $\mathfrak{J}$(⌊*Obj*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
show
($\mathfrak{F}$ $\circ_{CF-NTCF}$ *ntcf-const* $\mathfrak{J}$ $\mathfrak{A}$ f)(⌊*NTMap*⌋) =
ntcf-const $\mathfrak{J}$ $\mathfrak{B}$ ($\mathfrak{F}$(⌊*ArrMap*⌋)(⌊f⌋))(⌊*NTMap*⌋)
by (*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
(
use assms(*2*) **in**
‹*cs-concl cs-shallow cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*›
)+
qed *simp-all*

lemmas [*cat-cs-simps*] = *is-functor.cf-ntcf-comp-cf-ntcf-const*

6.11 Natural isomorphism

See Chapter I-4 in [7].

locale *is-iso-ntcf* = *is-ntcf* +
assumes *iso-ntcf-is-iso-arr*[*cat-arrow-cs-intros*]:
a $\in_\circ$ $\mathfrak{A}$(⌊*Obj*⌋) $\Longrightarrow$ $\mathfrak{N}$(⌊*NTMap*⌋)(⌊a⌋) : $\mathfrak{F}$(⌊*ObjMap*⌋)(⌊a⌋) $\mapsto_{iso_{\mathfrak{B}}}$ $\mathfrak{G}$(⌊*ObjMap*⌋)(⌊a⌋)

syntax *-is-iso-ntcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*

(‹(- : - ↦$_{CF.iso}$ - : - ↦↦$_{C1}$ -)› [51, 51, 51, 51, 51] 51)
syntax-consts *-is-iso-ntcf* ⇌ *is-iso-ntcf*
translations $\mathfrak{N}$: $\mathfrak{F}$ ↦$_{CF.iso}$ $\mathfrak{G}$: $\mathfrak{A}$ ↦↦$_{C\alpha}$ $\mathfrak{B}$ ⇌
CONST is-iso-ntcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

lemma (in *is-iso-ntcf*) *iso-ntcf-is-iso-arr'*:
assumes a ∈$_{\circ}$ $\mathfrak{A}$(|*Obj*|)
and A = $\mathfrak{F}$(|*ObjMap*|)(|a|)
and B = $\mathfrak{G}$(|*ObjMap*|)(|a|)
shows $\mathfrak{N}$(|*NTMap*|)(|a|) : A ↦$_{iso\mathfrak{B}}$ B
using *assms* **by** (*auto intro*: *cat-arrow-cs-intros*)

lemmas [*cat-arrow-cs-intros*] =
is-iso-ntcf.iso-ntcf-is-iso-arr'

lemma (in *is-iso-ntcf*) *iso-ntcf-is-iso-arr''*:
assumes a ∈$_{\circ}$ $\mathfrak{A}$(|*Obj*|)
and A = $\mathfrak{F}$(|*ObjMap*|)(|a|)
and B = $\mathfrak{G}$(|*ObjMap*|)(|a|)
and F = $\mathfrak{N}$(|*NTMap*|)(|a|)
and $\mathfrak{B}'$ = $\mathfrak{B}$
shows F : A ↦$_{iso\mathfrak{B}'}$ B
using *assms* **by** (*auto intro*: *cat-arrow-cs-intros*)

Rules.

lemma (in *is-iso-ntcf*) *is-iso-ntcf-axioms'*[*cat-cs-intros*]:
assumes α' = α **and** $\mathfrak{F}'$ = $\mathfrak{F}$ **and** $\mathfrak{G}'$ = $\mathfrak{G}$ **and** $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$ = $\mathfrak{B}$
shows $\mathfrak{N}$: $\mathfrak{F}'$ ↦$_{CF.iso}$ $\mathfrak{G}'$: $\mathfrak{A}'$ ↦↦$_{C_{\alpha}'}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule is-iso-ntcf-axioms*)

mk-ide rf *is-iso-ntcf-def*[*unfolded is-iso-ntcf-axioms-def*]
|*intro is-iso-ntcfI*|
|*dest is-iso-ntcfD*[*dest*]|
|*elim is-iso-ntcfE*[*elim*]|

lemmas [*ntcf-cs-intros*] = *is-iso-ntcfD*(*1*)

6.12 Inverse natural transformation

6.12.1 Definition and elementary properties

definition *inv-ntcf* :: *V* ⇒ *V*
where *inv-ntcf* $\mathfrak{N}$ =
[
(λa∈$_{\circ}$$\mathfrak{N}$(|*NTDGDom*|)(|*Obj*|). *SOME g. is-inverse* ($\mathfrak{N}$(|*NTDGCod*|)) g ($\mathfrak{N}$(|*NTMap*|)(|a|))),
$\mathfrak{N}$(|*NTCod*|),
$\mathfrak{N}$(|*NTDom*|),
$\mathfrak{N}$(|*NTDGDom*|),
$\mathfrak{N}$(|*NTDGCod*|)
]$_{\circ}$

Slicing.

lemma *inv-ntcf-components*:
shows *inv-ntcf* $\mathfrak{N}$(|*NTMap*|) =
(λa∈$_{\circ}$$\mathfrak{N}$(|*NTDGDom*|)(|*Obj*|). *SOME g. is-inverse* ($\mathfrak{N}$(|*NTDGCod*|)) g ($\mathfrak{N}$(|*NTMap*|)(|a|)))
and [*cat-cs-simps*]: *inv-ntcf* $\mathfrak{N}$(|*NTDom*|) = $\mathfrak{N}$(|*NTCod*|)
and [*cat-cs-simps*]: *inv-ntcf* $\mathfrak{N}$(|*NTCod*|) = $\mathfrak{N}$(|*NTDom*|)
and [*cat-cs-simps*]: *inv-ntcf* $\mathfrak{N}$(|*NTDGDom*|) = $\mathfrak{N}$(|*NTDGDom*|)

and [*cat-cs-simps*]: *inv-ntcf* $\mathfrak{N}(\lfloor NTDGCod \rfloor) = \mathfrak{N}(\lfloor NTDGCod \rfloor)$
unfolding *inv-ntcf-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Components.

lemma (**in** *is-iso-ntcf*) *is-iso-ntcf-inv-ntcf-components*[*cat-cs-simps*]:
inv-ntcf $\mathfrak{N}(\lfloor NTDom \rfloor) = \mathfrak{G}$
inv-ntcf $\mathfrak{N}(\lfloor NTCod \rfloor) = \mathfrak{F}$
inv-ntcf $\mathfrak{N}(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
inv-ntcf $\mathfrak{N}(\lfloor NTDGCod \rfloor) = \mathfrak{B}$
unfolding *inv-ntcf-components* **by** (*simp-all add*: *cat-cs-simps*)

6.12.2 Natural transformation map

lemma *inv-ntcf-NTMap-vsv*[*cat-cs-intros*]: *vsv* (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)$)
unfolding *inv-ntcf-components* **by** *auto*

lemma (**in** *is-iso-ntcf*) *iso-ntcf-inv-ntcf-NTMap-app-is-inverse*[*cat-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor)$
shows *is-inverse* $\mathfrak{B}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$) ($\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$)
proof–
from *assms is-iso-ntcf-axioms* **have** $\exists g.$ *is-inverse* $\mathfrak{B}$ g ($\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$) **by** *auto*
from *assms someI2-ex*[*OF this*] **show**
is-inverse $\mathfrak{B}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$) ($\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$)
unfolding *inv-ntcf-components* **by** (*simp add*: *cat-cs-simps*)
qed

lemma (**in** *is-iso-ntcf*) *iso-ntcf-inv-ntcf-NTMap-app-is-the-inverse*[*cat-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor)$
shows *inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) = (\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor))^{-1}{}_{C\mathfrak{B}}$
proof–
have *is-inverse* $\mathfrak{B}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$) ($\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$)
by (*rule iso-ntcf-inv-ntcf-NTMap-app-is-inverse*[*OF assms*])
from *NTDom.HomCod.cat-is-inverse-eq-the-inverse*[*OF this*] **show** *?thesis* **.**
qed

lemmas [*cat-cs-simps*] = *is-iso-ntcf.iso-ntcf-inv-ntcf-NTMap-app-is-the-inverse*

lemma (**in** *is-ntcf*) *inv-ntcf-NTMap-vdomain*[*cat-cs-simps*]:
$\mathcal{D}_{\circ}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)$) = $\mathfrak{A}(\lfloor Obj \rfloor)$
unfolding *inv-ntcf-components* **by** (*simp add*: *cat-cs-simps*)

lemmas [*cat-cs-simps*] = *is-ntcf.inv-ntcf-NTMap-vdomain*

lemma (**in** *is-iso-ntcf*) *inv-ntcf-NTMap-vrange*:
$\mathcal{R}_{\circ}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)$) $\subseteq_{\circ}$ $\mathfrak{B}(\lfloor Arr \rfloor)$
proof(*rule vsubsetI*)
interpret *inv-*$\mathfrak{N}$: *vsv* ‹*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)$› **by** (*rule inv-ntcf-NTMap-vsv*)
fix f **assume** $f \in_{\circ} \mathcal{R}_{\circ}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)$)
then obtain a
where *f-def*: f = *inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$ **and** $a \in_{\circ} \mathcal{D}_{\circ}$ (*inv-ntcf* $\mathfrak{N}(\lfloor NTMap \rfloor)$)
by (*blast elim*: *inv-*$\mathfrak{N}$*.vrange-atE*)
then have $a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor)$ **by** (*simp add*: *cat-cs-simps*)
then have *is-inverse* $\mathfrak{B}$ f ($\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$)
unfolding *f-def* **by** (*intro iso-ntcf-inv-ntcf-NTMap-app-is-inverse*)
then show $f \in_{\circ} \mathfrak{B}(\lfloor Arr \rfloor)$ **by** *auto*
qed

6.12.3 Opposite natural isomorphism

lemma (**in** *is-iso-ntcf*) *is-iso-ntcf-op*:
  op-ntcf $\mathfrak{N}$: *op-cf* $\mathfrak{G}$ $\mapsto_{CF.iso}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{B}$
proof–
  from *is-iso-ntcf-axioms* **have**
    op-ntcf $\mathfrak{N}$: *op-cf* $\mathfrak{G}$ $\mapsto_{CF}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ *op-cat* $\mathfrak{B}$
    by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)
  then show *?thesis*
    by (*intro is-iso-ntcfI*) (*auto simp*: *cat-op-simps cat-arrow-cs-intros*)
qed

lemma (**in** *is-iso-ntcf*) *is-iso-ntcf-op'*[*cat-op-intros*]:
  assumes $\mathfrak{G}'$ = *op-cf* $\mathfrak{G}$
    and $\mathfrak{F}'$ = *op-cf* $\mathfrak{F}$
    and $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$
    and $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
  shows *op-ntcf* $\mathfrak{N}$: $\mathfrak{G}'$ $\mapsto_{CF.iso}$ $\mathfrak{F}'$: $\mathfrak{A}'$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}'$
  unfolding *assms* **by** (*rule is-iso-ntcf-op*)

lemmas *is-iso-ntcf-op*[*cat-op-intros*] = *is-iso-ntcf.is-iso-ntcf-op*

6.13 A natural isomorphism is an isomorphism in the category *Funct*

The results that are presented in this subsection can be found in nLab (see [1][2]).

lemma *is-iso-arr-is-iso-ntcf*:
  assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    and $\mathfrak{M}$: $\mathfrak{G}$ $\mapsto_{CF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    and $\mathfrak{N}$ $\cdot_{NTCF}$ $\mathfrak{M}$ = *ntcf-id* $\mathfrak{G}$
    and $\mathfrak{M}$ $\cdot_{NTCF}$ $\mathfrak{N}$ = *ntcf-id* $\mathfrak{F}$
  shows $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF.iso}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
proof–
  interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{M}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{F}$ $\mathfrak{M}$ **by** (*rule assms*(*2*))
  show *?thesis*
  proof(*rule is-iso-ntcfI*)
    fix *a* **assume** *prems*: *a* $\in_{\circ}$ $\mathfrak{A}$(|*Obj*|)
    show $\mathfrak{N}$(|*NTMap*|)(|*a*|) : $\mathfrak{F}$(|*ObjMap*|)(|*a*|) $\mapsto_{iso\mathfrak{B}}$ $\mathfrak{G}$(|*ObjMap*|)(|*a*|)
    proof(*rule is-iso-arrI*)
      show *is-inverse* $\mathfrak{B}$ ($\mathfrak{M}$(|*NTMap*|)(|*a*|)) ($\mathfrak{N}$(|*NTMap*|)(|*a*|))
      proof(*rule is-inverseI*)
        from *prems* **have**
          $\mathfrak{M}$(|*NTMap*|)(|*a*|) $\circ_{A\mathfrak{B}}$ $\mathfrak{N}$(|*NTMap*|)(|*a*|) = ($\mathfrak{M}$ $\cdot_{NTCF}$ $\mathfrak{N}$)(|*NTMap*|)(|*a*|)
          by (*simp add*: *ntcf-vcomp-NTMap-app*[*OF assms*(*2,1*) *prems*])
        also have ... = *ntcf-id* $\mathfrak{F}$(|*NTMap*|)(|*a*|) **unfolding** *assms*(*4*) **by** *simp*
        also from *prems* $\mathfrak{N}$.*NTDom.ntcf-id-NTMap-app-vdomain* **have**
          ... = $\mathfrak{B}$(|*CId*|)(|$\mathfrak{F}$(|*ObjMap*|)(|*a*|)|)
          unfolding *ntcf-id-components* **by** *auto*
        finally show $\mathfrak{M}$(|*NTMap*|)(|*a*|) $\circ_{A\mathfrak{B}}$ $\mathfrak{N}$(|*NTMap*|)(|*a*|) = $\mathfrak{B}$(|*CId*|)(|$\mathfrak{F}$(|*ObjMap*|)(|*a*|)|).
        from *prems* **have**
          $\mathfrak{N}$(|*NTMap*|)(|*a*|) $\circ_{A\mathfrak{B}}$ $\mathfrak{M}$(|*NTMap*|)(|*a*|) = ($\mathfrak{N}$ $\cdot_{NTCF}$ $\mathfrak{M}$)(|*NTMap*|)(|*a*|)
          by (*simp add*: *ntcf-vcomp-NTMap-app*[*OF assms*(*1,2*) *prems*])
        also have ... = *ntcf-id* $\mathfrak{G}$(|*NTMap*|)(|*a*|) **unfolding** *assms*(*3*) **by** *simp*
        also from *prems* $\mathfrak{N}$.*NTCod.ntcf-id-NTMap-app-vdomain* **have**
          ... = $\mathfrak{B}$(|*CId*|)(|$\mathfrak{G}$(|*ObjMap*|)(|*a*|)|)
          unfolding *ntcf-id-components* **by** *auto*

[2] https://ncatlab.org/nlab/show/natural+isomorphism

```
      finally show 𝔑(⁅NTMap⁆)(⁅a⁆) ∘_{A𝔅} 𝔐(⁅NTMap⁆)(⁅a⁆) = 𝔅(⁅CId⁆)(⁅𝔊(⁅ObjMap⁆)(⁅a⁆)⁆).
    qed (auto simp: prems cat-cs-intros)
  qed (auto simp: prems cat-cs-intros)
 qed (auto simp: cat-cs-intros)
qed

lemma iso-ntcf-is-iso-arr:
 assumes 𝔑 : 𝔉 ↦_{CF.iso} 𝔊 : 𝔄 ↦↦_{Cα} 𝔅
 shows [ntcf-cs-intros]: inv-ntcf 𝔑 : 𝔊 ↦_{CF.iso} 𝔉 : 𝔄 ↦↦_{Cα} 𝔅
  and 𝔑 ⋆_{NTCF} inv-ntcf 𝔑 = ntcf-id 𝔊
  and inv-ntcf 𝔑 ⋆_{NTCF} 𝔑 = ntcf-id 𝔉
proof–

 interpret is-iso-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))

 define m where m a = inv-ntcf 𝔑(⁅NTMap⁆)(⁅a⁆) for a
 have is-inverse[intro]: a ∈₀ 𝔄(⁅Obj⁆) ⟹ is-inverse 𝔅 (m a) (𝔑(⁅NTMap⁆)(⁅a⁆))
  for a
  unfolding m-def by (cs-concl cs-shallow cs-intro: cat-cs-intros)
 have [dest, intro, simp]:
  a ∈₀ 𝔄(⁅Obj⁆) ⟹ m a : 𝔊(⁅ObjMap⁆)(⁅a⁆) ↦_{iso𝔅} 𝔉(⁅ObjMap⁆)(⁅a⁆) for a
 proof–
  assume prems: a ∈₀ 𝔄(⁅Obj⁆)
  from prems have 𝔑(⁅NTMap⁆)(⁅a⁆) : 𝔉(⁅ObjMap⁆)(⁅a⁆) ↦_{iso𝔅} 𝔊(⁅ObjMap⁆)(⁅a⁆)
   by (auto intro: cat-cs-intros cat-arrow-cs-intros)
  with is-inverse[OF prems] show m a : 𝔊(⁅ObjMap⁆)(⁅a⁆) ↦_{iso𝔅} 𝔉(⁅ObjMap⁆)(⁅a⁆)
   by
    (
     meson
      NTDom.HomCod.cat-is-inverse-is-iso-arr is-iso-arrD
    )
 qed
 have [intro]:
  f : a ↦_𝔄 b ⟹ m b ∘_{A𝔅} 𝔊(⁅ArrMap⁆)(⁅f⁆) = 𝔉(⁅ArrMap⁆)(⁅f⁆) ∘_{A𝔅} m a
  for f a b
 proof–
  assume prems: f : a ↦_𝔄 b
  then have ma: m a : 𝔊(⁅ObjMap⁆)(⁅a⁆) ↦_{iso𝔅} 𝔉(⁅ObjMap⁆)(⁅a⁆)
   and mb: m b : 𝔊(⁅ObjMap⁆)(⁅b⁆) ↦_{iso𝔅} 𝔉(⁅ObjMap⁆)(⁅b⁆)
   and 𝔊f: 𝔊(⁅ArrMap⁆)(⁅f⁆) : 𝔊(⁅ObjMap⁆)(⁅a⁆) ↦_𝔅 𝔊(⁅ObjMap⁆)(⁅b⁆)
   and 𝔑a: 𝔑(⁅NTMap⁆)(⁅a⁆) : 𝔉(⁅ObjMap⁆)(⁅a⁆) ↦_𝔅 𝔊(⁅ObjMap⁆)(⁅a⁆)
   and 𝔉f: 𝔉(⁅ArrMap⁆)(⁅f⁆) : 𝔉(⁅ObjMap⁆)(⁅a⁆) ↦_𝔅 𝔉(⁅ObjMap⁆)(⁅b⁆)
   and 𝔑b: 𝔑(⁅NTMap⁆)(⁅b⁆) : 𝔉(⁅ObjMap⁆)(⁅b⁆) ↦_𝔅 𝔊(⁅ObjMap⁆)(⁅b⁆)
   by (auto intro: cat-cs-intros)
  then have 𝔑b𝔉f:
   𝔑(⁅NTMap⁆)(⁅b⁆) ∘_{A𝔅} 𝔉(⁅ArrMap⁆)(⁅f⁆) : 𝔉(⁅ObjMap⁆)(⁅a⁆) ↦_𝔅 𝔊(⁅ObjMap⁆)(⁅b⁆)
   by (auto intro: cat-cs-intros)
  from prems have inv-ma: is-inverse 𝔅 (m a) (𝔑(⁅NTMap⁆)(⁅a⁆))
   and inv-mb: is-inverse 𝔅 (𝔑(⁅NTMap⁆)(⁅b⁆)) (m b)
   by (auto simp: is-inverse-sym)
  from mb have mb-𝔑b: m b ∘_{A𝔅} 𝔑(⁅NTMap⁆)(⁅b⁆) = 𝔅(⁅CId⁆)(⁅𝔉(⁅ObjMap⁆)(⁅b⁆)⁆)
   by (auto intro: is-inverse-Comp-CId-right[OF inv-mb])
  from prems have 𝔑a-ma: 𝔑(⁅NTMap⁆)(⁅a⁆) ∘_{A𝔅} m a = 𝔅(⁅CId⁆)(⁅𝔊(⁅ObjMap⁆)(⁅a⁆)⁆)
   using 𝔑a inv-ma ma by (meson is-inverse-Comp-CId-right)
  from 𝔊f have m b ∘_{A𝔅} 𝔊(⁅ArrMap⁆)(⁅f⁆) =
   m b ∘_{A𝔅} (𝔊(⁅ArrMap⁆)(⁅f⁆) ∘_{A𝔅} (𝔑(⁅NTMap⁆)(⁅a⁆) ∘_{A𝔅} m a))
   unfolding 𝔑a-ma by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  also have ... = m b ∘_{A𝔅} ((𝔊(⁅ArrMap⁆)(⁅f⁆) ∘_{A𝔅} 𝔑(⁅NTMap⁆)(⁅a⁆)) ∘_{A𝔅} m a)
```

```
    by
      (
       metis
         ma 𝔊f 𝔑a NTDom.HomCod.cat-Comp-assoc is-iso-arrD(1)
      )
  also from prems have
    ... = m b ∘_A𝔅 ((𝔑(|NTMap|)(|b|) ∘_A𝔅 𝔉(|ArrMap|)(|f|)) ∘_A𝔅 m a)
    by (metis ntcf-Comp-commute)
  also have ... = (m b ∘_A𝔅 (𝔑(|NTMap|)(|b|) ∘_A𝔅 𝔉(|ArrMap|)(|f|))) ∘_A𝔅 m a
    by
      (
       metis
         𝔑b𝔉f ma mb NTDom.HomCod.cat-Comp-assoc is-iso-arrD(1)
      )
  also from 𝔉f 𝔑b mb NTDom.HomCod.cat-Comp-assoc have
    ... =  ((m b ∘_A𝔅 𝔑(|NTMap|)(|b|)) ∘_A𝔅 𝔉(|ArrMap|)(|f|)) ∘_A𝔅 m a
    by (metis is-iso-arrD(1))
  also from 𝔉f have ... = 𝔉(|ArrMap|)(|f|) ∘_A𝔅 m a
    unfolding mb-𝔑b by (simp add: cat-cs-simps)
  finally show m b ∘_A𝔅 𝔊(|ArrMap|)(|f|) = 𝔉(|ArrMap|)(|f|) ∘_A𝔅 m a by simp
qed

show 𝔐: inv-ntcf 𝔑 : 𝔊 ↦_CF.iso 𝔉 : 𝔄 ↦↦_Cα 𝔅
proof(intro is-iso-ntcfI is-ntcfI', unfold m-def[symmetric])
  show vfsequence (inv-ntcf 𝔑) unfolding inv-ntcf-def by simp
  show vcard (inv-ntcf 𝔑) = 5_ℕ
    unfolding inv-ntcf-def by (simp add: nat-omega-simps)
qed (auto simp: cat-cs-simps intro: cat-cs-intros)

interpret 𝔐: is-iso-ntcf α 𝔄 𝔅 𝔊 𝔉 ‹inv-ntcf 𝔑› by (rule 𝔐)

show 𝔑𝔐: 𝔑 ∙_NTCF inv-ntcf 𝔑 = ntcf-id 𝔊
proof(rule ntcf-eqI)
  from NTCod.cf-ntcf-id-is-ntcf show ntcf-id 𝔊 : 𝔊 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
    by auto
  show (𝔑 ∙_NTCF inv-ntcf 𝔑)(|NTMap|) = ntcf-id 𝔊(|NTMap|)
  proof(rule vsv-eqI)
    fix a assume a ∈_∘ 𝒟_∘ ((𝔑 ∙_NTCF inv-ntcf 𝔑)(|NTMap|))
    then have a ∈_∘ 𝔄(|Obj|)
      unfolding ntcf-vcomp-NTMap-vdomain[OF 𝔐.is-ntcf-axioms] by simp
    then show (𝔑 ∙_NTCF inv-ntcf 𝔑)(|NTMap|)(|a|) = ntcf-id 𝔊(|NTMap|)(|a|)
      by
        (
         cs-concl cs-shallow
           cs-simp: cat-cs-simps
           cs-intro: cat-cs-intros cat-arrow-cs-intros
        )
  qed
    (
     auto
       simp: ntsmcf-vcomp-components(1) cat-cs-simps
       intro: cat-cs-intros
    )
qed (auto intro: cat-cs-intros)

show 𝔐𝔑: inv-ntcf 𝔑 ∙_NTCF 𝔑 = ntcf-id 𝔉
proof(rule ntcf-eqI)
  show (inv-ntcf 𝔑 ∙_NTCF 𝔑)(|NTMap|) = ntcf-id 𝔉(|NTMap|)
```

```
    proof(rule vsv-eqI)
      show 𝒟ₒ ((inv-ntcf 𝔑 ⊙ₙₜ꜀ 𝔑)(|NTMap|)) = 𝒟ₒ (ntcf-id 𝔉(|NTMap|))
        by (simp add: ntsmcf-vcomp-components(1) cat-cs-simps)
      fix a assume a ∈ₒ 𝒟ₒ ((inv-ntcf 𝔑 ⊙ₙₜ꜀ 𝔑)(|NTMap|))
      then have a ∈ₒ 𝔄(|Obj|)
        unfolding ntsmcf-vcomp-components by (simp add: cat-cs-simps)
      then show (inv-ntcf 𝔑 ⊙ₙₜ꜀ 𝔑)(|NTMap|)(|a|) = ntcf-id 𝔉(|NTMap|)(|a|)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps
              cs-intro: cat-cs-intros cat-arrow-cs-intros
          )
    qed
      (
        auto simp:
          ntsmcf-vcomp-components(1)
          ntcf-id-components(1)
          cat-cs-simps
          intro: vsv-vcomp
      )
  qed (auto intro: cat-cs-intros)
qed
```

6.13.1 The operation of taking the inverse natural transformation is an involution

```
lemma (in is-iso-ntcf) iso-ntcf-inv-ntcf-inv-ntcf[ntcf-cs-simps]:
  inv-ntcf (inv-ntcf 𝔑) = 𝔑
proof(rule ntcf-eqI)
  show 𝔑 : 𝔉 ↦꜀𝔉 𝔊 : 𝔄 ↦↦꜀α 𝔅 by (cs-concl cs-intro: cat-cs-intros)
  show inv-ntcf (inv-ntcf 𝔑) : 𝔉 ↦꜀𝔉 𝔊 : 𝔄 ↦↦꜀α 𝔅
    by (cs-concl cs-shallow cs-intro: ntcf-cs-intros cat-cs-intros)
  then have dom-lhs: 𝒟ₒ (inv-ntcf (inv-ntcf 𝔑)(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps)
  show inv-ntcf (inv-ntcf 𝔑)(|NTMap|) = 𝔑(|NTMap|)
  proof(rule vsv-eqI, unfold cat-cs-simps dom-lhs)
    fix a assume prems: a ∈ₒ 𝔄(|Obj|)
    then show inv-ntcf (inv-ntcf 𝔑)(|NTMap|)(|a|) = 𝔑(|NTMap|)(|a|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps
            cs-intro: cat-arrow-cs-intros ntcf-cs-intros cat-cs-intros
        )
  qed (auto intro: cat-cs-intros)
qed simp-all

lemmas [ntcf-cs-simps] = is-iso-ntcf.iso-ntcf-inv-ntcf-inv-ntcf
```

6.13.2 Natural isomorphisms from natural transformations

```
lemma iso-ntcf-if-is-inverse:
  assumes 𝔑 : 𝔉 ↦꜀𝔉 𝔊 : 𝔄 ↦↦꜀α 𝔅
    and 𝔐 : 𝔊 ↦꜀𝔉 𝔉 : 𝔄 ↦↦꜀α 𝔅
    and ⋀a. a ∈ₒ 𝔄(|Obj|) ⟹ is-inverse 𝔅 (𝔐(|NTMap|)(|a|)) (𝔑(|NTMap|)(|a|))
  shows 𝔑 : 𝔉 ↦꜀𝔉.iso 𝔊 : 𝔄 ↦↦꜀α 𝔅
    and 𝔐 : 𝔊 ↦꜀𝔉.iso 𝔉 : 𝔄 ↦↦꜀α 𝔅
    and 𝔐 = inv-ntcf 𝔑
```

```
    and 𝔑 = inv-ntcf 𝔐
proof–
  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret 𝔐: is-ntcf α 𝔄 𝔅 𝔊 𝔉 𝔐 by (rule assms(2))
  show 𝔑: 𝔑 : 𝔉 ↦_CF.iso 𝔊 : 𝔄 ↦↦_Cα 𝔅
  proof(intro is-iso-ntcfI assms(1))
    fix a assume prems: a ∈∘ 𝔄(|Obj|)
    show 𝔑(|NTMap|)(|a|) : 𝔉(|ObjMap|)(|a|) ↦_iso𝔅 𝔊(|ObjMap|)(|a|)
      by
        (
          rule is-iso-arrI[
            OF 𝔑.ntcf-NTMap-is-arr[OF prems] assms(3)[OF prems]
            ]
        )
  qed
  show 𝔐: 𝔐 : 𝔊 ↦_CF.iso 𝔉 : 𝔄 ↦↦_Cα 𝔅
  proof(intro is-iso-ntcfI assms(2))
    fix a assume prems: a ∈∘ 𝔄(|Obj|)
    show 𝔐(|NTMap|)(|a|) : 𝔊(|ObjMap|)(|a|) ↦_iso𝔅 𝔉(|ObjMap|)(|a|)
      by
        (
          rule is-iso-arrI
            [
              OF
                𝔐.ntcf-NTMap-is-arr[OF prems]
                is-inverse-sym[THEN iffD1, OF assms(3)[OF prems]]
            ]
        )
  qed
  have 𝔐-NTMap-unique: g = 𝔐(|NTMap|)(|a|)
    if a ∈∘ 𝔄(|Obj|) and is-inverse 𝔅 g (𝔑(|NTMap|)(|a|)) for g a
    by (rule 𝔑.NTDom.HomCod.cat-is-inverse-eq[OF that(2) assms(3)[OF that(1)]])
  show 𝔐 = inv-ntcf 𝔑
  proof(rule ntcf-eqI, rule assms(2))
    from 𝔑 show inv-ntcf 𝔑 : 𝔊 ↦_CF 𝔉 : 𝔄 ↦↦_Cα 𝔅
      by (cs-concl cs-shallow cs-intro: ntcf-cs-intros)
    show 𝔐(|NTMap|) = inv-ntcf 𝔑(|NTMap|)
    proof(rule vsv-eqI, unfold cat-cs-simps)
      fix a assume prems: a ∈∘ 𝔄(|Obj|)
      show 𝔐(|NTMap|)(|a|) = inv-ntcf 𝔑(|NTMap|)(|a|)
      proof(intro 𝔐-NTMap-unique[symmetric] prems)
        from prems assms(3)[OF prems] show
          is-inverse 𝔅 (inv-ntcf 𝔑(|NTMap|)(|a|)) (𝔑(|NTMap|)(|a|))
          unfolding inv-ntcf-components cat-cs-simps
          by
            (
              cs-concl cs-shallow
                cs-intro: V-cs-intros cs-simp: some-eq-ex V-cs-simps
            )
      qed
    qed (auto simp: inv-ntcf-components)
  qed simp-all
  then have inv-ntcf (inv-ntcf 𝔑) = inv-ntcf 𝔐 by simp
  from this 𝔐 𝔑 show 𝔑 = inv-ntcf 𝔐
    by (cs-prems cs-shallow cs-simp: ntcf-cs-simps)
qed
```

6.13.3 Vertical composition of natural isomorphisms

lemma *ntcf-vcomp-is-iso-ntcf*[*cat-cs-intros*]:
  assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF.iso} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
  shows $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof(*intro is-iso-arr-is-iso-ntcf*)
  note *inv-ntcf-*$\mathfrak{M}$ = *iso-ntcf-is-iso-arr*[*OF assms*(*1*)]
    and *inv-ntcf-*$\mathfrak{N}$ = *iso-ntcf-is-iso-arr*[*OF assms*(*2*)]
  note [*cat-cs-simps*] = *inv-ntcf-*$\mathfrak{M}$(*2,3*) *inv-ntcf-*$\mathfrak{N}$(*2,3*)
  from *assms* **show** $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
  from *inv-ntcf-*$\mathfrak{M}$(*1*) *inv-ntcf-*$\mathfrak{N}$(*1*) **show**
    inv-ntcf $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{M} : \mathfrak{H} \mapsto_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
  from *assms inv-ntcf-*$\mathfrak{M}$(*1*) *inv-ntcf-*$\mathfrak{N}$(*1*) **have**
    $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} \bullet_{NTCF}$ (*inv-ntcf* $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{M}$) =
      $\mathfrak{M} \bullet_{NTCF} (\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{N}) \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{M}$
    by
      (
        cs-concl
          cs-simp: *ntcf-vcomp-assoc* **cs-intro**: *cat-cs-intros ntcf-cs-intros*
)
  also from *assms* **have** ... = *ntcf-id* $\mathfrak{H}$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *ntcf-cs-intros*)
  finally show $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} \bullet_{NTCF}$ (*inv-ntcf* $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{M}$) = *ntcf-id* $\mathfrak{H}$
    by *simp*
  from *assms inv-ntcf-*$\mathfrak{M}$(*1*) *inv-ntcf-*$\mathfrak{N}$(*1*) **have**
    inv-ntcf $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{M} \bullet_{NTCF}$ ($\mathfrak{M} \bullet_{NTCF} \mathfrak{N}$) =
      inv-ntcf $\mathfrak{N} \bullet_{NTCF}$ (*inv-ntcf* $\mathfrak{M} \bullet_{NTCF} \mathfrak{M}$) $\bullet_{NTCF} \mathfrak{N}$
    by
      (
        cs-concl
          cs-simp: *ntcf-vcomp-assoc* **cs-intro**: *cat-cs-intros ntcf-cs-intros*
)
  also from *assms* **have** ... = *ntcf-id* $\mathfrak{F}$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *ntcf-cs-intros*)
  finally show *inv-ntcf* $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{M} \bullet_{NTCF}$ ($\mathfrak{M} \bullet_{NTCF} \mathfrak{N}$) = *ntcf-id* $\mathfrak{F}$
    by *simp*
qed

6.13.4 Horizontal composition of natural isomorphisms

lemma *ntcf-hcomp-is-iso-ntcf*:
  assumes $\mathfrak{M} : \mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
  shows $\mathfrak{M} \circ_{NTCF} \mathfrak{N} : \mathfrak{F}' \circ_{CF} \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G}' \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-iso-arr-is-iso-ntcf*)
  note *inv-ntcf-*$\mathfrak{M}$ = *iso-ntcf-is-iso-arr*[*OF assms*(*1*)]
    and *inv-ntcf-*$\mathfrak{N}$ = *iso-ntcf-is-iso-arr*[*OF assms*(*2*)]
  note [*cat-cs-simps*] = *inv-ntcf-*$\mathfrak{M}$(*2,3*) *inv-ntcf-*$\mathfrak{N}$(*2,3*)
  from *assms* **show** $\mathfrak{M} \circ_{NTCF} \mathfrak{N} : \mathfrak{F}' \circ_{CF} \mathfrak{F} \mapsto_{CF} \mathfrak{G}' \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
  from *inv-ntcf-*$\mathfrak{M}$(*1*) *inv-ntcf-*$\mathfrak{N}$(*1*) **show**
    inv-ntcf $\mathfrak{M} \circ_{NTCF}$ *inv-ntcf* $\mathfrak{N} : \mathfrak{G}' \circ_{CF} \mathfrak{G} \mapsto_{CF} \mathfrak{F}' \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
  from *assms inv-ntcf-*$\mathfrak{M}$(*1*) *inv-ntcf-*$\mathfrak{N}$(*1*) **have**
    $\mathfrak{M} \circ_{NTCF} \mathfrak{N} \bullet_{NTCF}$ (*inv-ntcf* $\mathfrak{M} \circ_{NTCF}$ *inv-ntcf* $\mathfrak{N}$) =
      ntcf-id $\mathfrak{G}' \circ_{NTCF}$ *ntcf-id* $\mathfrak{G}$

```
    by
      (
        cs-concl
          cs-simp: ntcf-comp-interchange-law[symmetric] cat-cs-simps
          cs-intro: ntcf-cs-intros
      )
  also from assms have ... = ntcf-id (𝔊' ∘_CF 𝔊)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros ntcf-cs-intros
      )
  finally show
    𝔐 ∘_NTCF 𝔑 ⋆_NTCF (inv-ntcf 𝔐 ∘_NTCF inv-ntcf 𝔑) = ntcf-id (𝔊' ∘_CF 𝔊)
    by simp
  from assms inv-ntcf-𝔐(1) inv-ntcf-𝔑(1) have
    inv-ntcf 𝔐 ∘_NTCF inv-ntcf 𝔑 ⋆_NTCF (𝔐 ∘_NTCF 𝔑) =
      ntcf-id 𝔉' ∘_NTCF ntcf-id 𝔉
    by
      (
        cs-concl
          cs-simp: ntcf-comp-interchange-law[symmetric] cat-cs-simps
          cs-intro: ntcf-cs-intros
      )
  also from assms have ... = ntcf-id (𝔉' ∘_CF 𝔉)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros ntcf-cs-intros)
  finally show
    inv-ntcf 𝔐 ∘_NTCF inv-ntcf 𝔑 ⋆_NTCF (𝔐 ∘_NTCF 𝔑) = ntcf-id (𝔉' ∘_CF 𝔉)
    by simp
qed

lemma ntcf-hcomp-is-iso-ntcf'[ntcf-cs-intros]:
  assumes 𝔐 : 𝔉' ↦_CF.iso 𝔊' : 𝔅 ↦↦_Cα ℭ
    and 𝔑 : 𝔉 ↦_CF.iso 𝔊 : 𝔄 ↦↦_Cα 𝔅
    and ℌ' = 𝔉' ∘_CF 𝔉
    and ℌ'' = 𝔊' ∘_CF 𝔊
  shows 𝔐 ∘_NTCF 𝔑 : ℌ' ↦_CF.iso ℌ'' : 𝔄 ↦↦_Cα ℭ
  using assms(1,2) unfolding assms(3,4) by (rule ntcf-hcomp-is-iso-ntcf)
```

6.13.5 Composition of a natural isomorphism and a functor

```
lemma ntcf-cf-comp-is-iso-ntcf:
  assumes 𝔑 : 𝔉 ↦_CF.iso 𝔊 : 𝔅 ↦↦_Cα ℭ and ℌ : 𝔄 ↦↦_Cα 𝔅
  shows 𝔑 ∘_NTCF-CF ℌ : 𝔉 ∘_CF ℌ ↦_CF.iso 𝔊 ∘_CF ℌ : 𝔄 ↦↦_Cα ℭ
proof(intro is-iso-ntcfI ntcf-cf-comp-is-ntcf)
  interpret 𝔑: is-iso-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(1))
  show 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔅 ↦↦_Cα ℭ by (rule 𝔑.is-ntcf-axioms)
  fix a assume a ∈₀ 𝔄⦇Obj⦈
  with assms(2) show
    (𝔑 ∘_NTCF-CF ℌ)⦇NTMap⦈⦇a⦈ : (𝔉 ∘_CF ℌ)⦇ObjMap⦈⦇a⦈ ↦_iso ℭ (𝔊 ∘_CF ℌ)⦇ObjMap⦈⦇a⦈
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-arrow-cs-intros
      )
qed (rule assms(2))

lemma ntcf-cf-comp-is-iso-ntcf'[cat-cs-intros]:
```

assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F}' = \mathfrak{F} \circ_{CF} \mathfrak{H}$
and $\mathfrak{G}' = \mathfrak{G} \circ_{CF} \mathfrak{H}$
shows $\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H} : \mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*rule ntcf-cf-comp-is-iso-ntcf*)

6.13.6 An identity natural transformation is a natural isomorphism

lemma (**in** *is-functor*) *cf-ntcf-id-is-iso-ntcf*:
ntcf-id $\mathfrak{F} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof–
have *ntcf-id* $\mathfrak{F} : \mathfrak{F} \mapsto_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** (*auto intro*: *cat-cs-intros*)
moreover then have *ntcf-id* $\mathfrak{F} \cdot_{NTCF}$ *ntcf-id* $\mathfrak{F}$ = *ntcf-id* $\mathfrak{F}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
ultimately show *?thesis* **by** (*auto intro*: *is-iso-arr-is-iso-ntcf*)
qed

lemma (**in** *is-functor*) *cf-ntcf-id-is-iso-ntcf'*[*ntcf-cs-intros*]:
assumes $\mathfrak{G}' = \mathfrak{F}$ **and** $\mathfrak{H}' = \mathfrak{F}$
shows *ntcf-id* $\mathfrak{F} : \mathfrak{G}' \mapsto_{CF.iso} \mathfrak{H}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule cf-ntcf-id-is-iso-ntcf*)

lemmas [*ntcf-cs-intros*] = *is-functor.cf-ntcf-id-is-iso-ntcf'*

6.14 Functor isomorphism

6.14.1 Definition and elementary properties

See subsection 1.5 in [3].

locale *iso-functor* =
fixes α $\mathfrak{F}$ $\mathfrak{G}$
assumes *iso-cf-is-iso-ntcf*: $\exists \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

notation *iso-functor* (**infixl** $\langle \approx_{CF^1} \rangle$ *50*)

Rules.

lemma *iso-functorI*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{F} \approx_{CF\alpha} \mathfrak{G}$
using *assms* **unfolding** *iso-functor-def* **by** *auto*

lemma *iso-functorD*[*dest*]:
assumes $\mathfrak{F} \approx_{CF\alpha} \mathfrak{G}$
shows $\exists \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
using *assms* **unfolding** *iso-functor-def* **by** *auto*

lemma *iso-functorE*[*elim*]:
assumes $\mathfrak{F} \approx_{CF\alpha} \mathfrak{G}$
obtains $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}$ **where** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
using *assms* **unfolding** *iso-functor-def* **by** *auto*

6.14.2 A functor isomorphism is an equivalence relation

lemma *iso-functor-refl*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{F} \approx_{CF\alpha} \mathfrak{F}$
proof(*rule iso-functorI*)

from *assms* **show** *ntcf-id* $\mathfrak{F} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*cs-concl* **cs-shallow cs-intro**: *ntcf-cs-intros*)
qed

lemma *iso-functor-sym*[*sym*]:
assumes $\mathfrak{F} \approx_{CF\alpha} \mathfrak{G}$
shows $\mathfrak{G} \approx_{CF\alpha} \mathfrak{F}$
proof–
from *assms* **obtain** $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$ **where** $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** *auto*
from *iso-ntcf-is-iso-arr*(*1*)[*OF* $\mathfrak{N}$] **show** $\mathfrak{G} \approx_{CF\alpha} \mathfrak{F}$
by (*auto simp*: *iso-functorI*)
qed

lemma *iso-functor-trans*[*trans*, *intro*]:
assumes $\mathfrak{F} \approx_{CF\alpha} \mathfrak{G}$ **and** $\mathfrak{G} \approx_{CF\alpha} \mathfrak{H}$
shows $\mathfrak{F} \approx_{CF\alpha} \mathfrak{H}$
proof–
from *assms*(*1*) **obtain** $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$ **where** $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by *auto*
moreover from *assms*(*2*) **obtain** $\mathfrak{A}'$ $\mathfrak{B}'$ $\mathfrak{M}$
where $\mathfrak{M}$: $\mathfrak{M} : \mathfrak{G} \mapsto_{CF.iso} \mathfrak{H} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
by *auto*
ultimately have $\mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** *blast+*
then have *eq*: $\mathfrak{A}' = \mathfrak{A}$ $\mathfrak{B}' = \mathfrak{B}$ **by** *auto*
from $\mathfrak{M}$ **have** $\mathfrak{M}$: $\mathfrak{M} : \mathfrak{G} \mapsto_{CF.iso} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **unfolding** *eq* **.**
from *ntcf-vcomp-is-iso-ntcf*[*OF* $\mathfrak{M}$ $\mathfrak{N}$] **show** *?thesis* **by** (*rule iso-functorI*)
qed

6.14.3 Opposite functor isomorphism

lemma (**in** *iso-functor*) *iso-functor-op*: *op-cf* $\mathfrak{F} \approx_{CF\alpha}$ *op-cf* $\mathfrak{G}$
proof–
from *iso-functor-axioms* **obtain** $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$ **where** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by *auto*
from *is-iso-ntcf-op*[*OF this*] **have** *op-cf* $\mathfrak{G} \approx_{CF\alpha}$ *op-cf* $\mathfrak{F}$
by (*auto simp*: *iso-functorI*)
then show *op-cf* $\mathfrak{F} \approx_{CF\alpha}$ *op-cf* $\mathfrak{G}$ **by** (*rule iso-functor-sym*)
qed

lemmas *iso-functor-op*[*cat-op-intros*] = *iso-functor.iso-functor-op*

7 Smallness for natural transformations

7.1 Natural transformation of functors with tiny maps

7.1.1 Definition and elementary properties

locale *is-tm-ntcf* = *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ +
  assumes *tm-ntcf-is-tm-ntsmcf*: *ntcf-ntsmcf* $\mathfrak{N}$:
    cf-smcf $\mathfrak{F}$ $\mapsto_{SMCF.tm}$ *cf-smcf* $\mathfrak{G}$: *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-tm-ntcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\mapsto_{CF.tm}$ - :/ - $\mapsto\mapsto_{C.tm1}$ -)$\rangle$ [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-tm-ntcf* $\rightleftharpoons$ *is-tm-ntcf*
translations $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$
  CONST is-tm-ntcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

abbreviation *all-tm-ntcfs* :: $V \Rightarrow V$
  where *all-tm-ntcfs* $\alpha \equiv$
    set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

abbreviation *tm-ntcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *tm-ntcfs* α $\mathfrak{A}$ $\mathfrak{B} \equiv$
    set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

abbreviation *these-tm-ntcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *these-tm-ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G} \equiv$
    set $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

lemma (**in** *is-tm-ntcf*) *tm-ntcf-is-tm-ntsmcf'*:
  assumes $\mathfrak{F}' =$ *cf-smcf* $\mathfrak{F}$
    and $\mathfrak{G}' =$ *cf-smcf* $\mathfrak{G}$
    and $\mathfrak{A}' =$ *cat-smc* $\mathfrak{A}$
    and $\mathfrak{B}' =$ *cat-smc* $\mathfrak{B}$
  shows *ntcf-ntsmcf* $\mathfrak{N}$: $\mathfrak{F}' \mapsto_{SMCF.tm}$ $\mathfrak{G}'$: $\mathfrak{A}' \mapsto\mapsto_{SMC.tm\alpha}$ $\mathfrak{B}'$
  unfolding *assms* **by** (*rule tm-ntcf-is-tm-ntsmcf*)

lemmas [*slicing-intros*] = *is-tm-ntcf.tm-ntcf-is-tm-ntsmcf'*

Rules.

lemma (**in** *is-tm-ntcf*) *is-tm-ntcf-axioms'*[*cat-small-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$ **and** $\mathfrak{F}' = \mathfrak{F}$ **and** $\mathfrak{G}' = \mathfrak{G}$
  shows $\mathfrak{N}$: $\mathfrak{F}' \mapsto_{CF.tm}$ $\mathfrak{G}'$: $\mathfrak{A}' \mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{B}'$
  unfolding *assms* **by** (*rule is-tm-ntcf-axioms*)

mk-ide rf *is-tm-ntcf-def*[*unfolded is-tm-ntcf-axioms-def*]
  |*intro is-tm-ntcfI*|
  |*dest is-tm-ntcfD*[*dest*]|
  |*elim is-tm-ntcfE*[*elim*]|

lemmas [*cat-small-cs-intros*] = *is-tm-ntcfD*(*1*)

context *is-tm-ntcf*
begin

interpretation *ntsmcf*: *is-tm-ntsmcf*
  α $\langle$*cat-smc* $\mathfrak{A}\rangle$ $\langle$*cat-smc* $\mathfrak{B}\rangle$ $\langle$*cf-smcf* $\mathfrak{F}\rangle$ $\langle$*cf-smcf* $\mathfrak{G}\rangle$ $\langle$*ntcf-ntsmcf* $\mathfrak{N}\rangle$
  by (*rule tm-ntcf-is-tm-ntsmcf*)

lemmas-with [*unfolded slicing-simps*]:

tm-ntcf-NTMap-in-Vset = *ntsmcf.tm-ntsmcf-NTMap-in-Vset*

end

sublocale *is-tm-ntcf* ⊆ *NTDom*: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$
using *tm-ntcf-is-tm-ntsmcf*
by (*intro is-tm-functorI*) (*auto intro*: *cat-cs-intros is-tm-ntsmcfD'*)

sublocale *is-tm-ntcf* ⊆ *NTCod*: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$
using *tm-ntcf-is-tm-ntsmcf*
by (*intro is-tm-functorI*) (*auto intro*: *cat-cs-intros is-tm-ntsmcfD'*)

Further rules.

lemma *is-tm-ntcfI'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof–
interpret *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(*3*))
show *?thesis*
proof(*intro is-tm-ntcfI*)
show *ntcf-ntsmcf* $\mathfrak{N}$:
cf-smcf $\mathfrak{F}$ $\mapsto_{SMCF.tm}$ *cf-smcf* $\mathfrak{G}$: *cat-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.tm\alpha}$ *cat-smc* $\mathfrak{B}$
by (*intro is-tm-ntsmcfI'*) (*auto intro*: *slicing-intros*)
qed (*auto intro*: *cat-cs-intros*)
qed

lemma *is-tm-ntcfD'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof–
interpret *is-tm-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
by (*auto simp*: *cat-small-cs-intros*)
qed

lemmas [*cat-small-cs-intros*] = *is-tm-ntcfD'*(*2*,*3*)

lemma *is-tm-ntcfE'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
obtains $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
using *is-tm-ntcfD'*[*OF assms*] **by** *auto*

The set of all natural transformations with tiny maps is small.

lemma *small-all-tm-ntcfs*[*simp*]:
small $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$
proof(*rule down*)
show
$\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\} \subseteq$

```
    elts (set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅})
  proof
    (
      simp only: elts-of-set small-all-ntcfs if-True,
      rule subsetI,
      unfold mem-Collect-eq
    )
    fix 𝔑 assume ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅
    then obtain 𝔉 𝔊 𝔄 𝔅 where 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅
      by clarsimp
    then interpret is-tm-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by simp
    have 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅 by (auto simp: cat-cs-intros)
    then show ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅 by auto
  qed
qed

lemma small-tm-ntcfs[simp]:
  small {𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅}
  by (rule down[of - ‹set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅}›])
    auto

lemma small-these-tm-ntcfs[simp]:
  small {𝔑. 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅}
  by (rule down[of - ‹set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅}›])
    auto
```

Further elementary results.

```
lemma these-tm-ntcfs-iff:
  𝔑 ∈₀ these-tm-ntcfs α 𝔄 𝔅 𝔉 𝔊 ⟷ 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅
  by auto
```

7.1.2 Opposite natural transformation of functors with tiny maps

```
lemma (in is-tm-ntcf) is-tm-ntcf-op: op-ntcf 𝔑 :
  op-cf 𝔊 ↦_CF.tm op-cf 𝔉 : op-cat 𝔄 ↦↦_C.tmα op-cat 𝔅
  by (intro is-tm-ntcfI′)
    (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+

lemma (in is-tm-ntcf) is-tm-ntcf-op′[cat-op-intros]:
  assumes 𝔊′ = op-cf 𝔊
    and 𝔉′ = op-cf 𝔉
    and 𝔄′ = op-cat 𝔄
    and 𝔅′ = op-cat 𝔅
  shows op-ntcf 𝔑 : 𝔊′ ↦_CF.tm 𝔉′ : 𝔄′ ↦↦_C.tmα 𝔅′
  unfolding assms by (rule is-tm-ntcf-op)

lemmas is-tm-ntcf-op[cat-op-intros] = is-tm-ntcf.is-tm-ntcf-op′
```

7.1.3 Vertical composition of natural transformations of functors with tiny maps

```
lemma ntcf-vcomp-is-tm-ntcf[cat-small-cs-intros]:
  assumes 𝔐 : 𝔊 ↦_CF.tm ℌ : 𝔄 ↦↦_C.tmα 𝔅
    and 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅
  shows 𝔐 ∙_NTCF 𝔑 : 𝔉 ↦_CF.tm ℌ : 𝔄 ↦↦_C.tmα 𝔅
proof–
  interpret 𝔐: is-tm-ntcf α 𝔄 𝔅 𝔊 ℌ 𝔐 by (rule assms(1))
  interpret 𝔑: is-tm-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  show ?thesis
```

by (*rule is-tm-ntcfI′*) (*auto intro*: *cat-cs-intros cat-small-cs-intros*)
qed

7.1.4 Identity natural transformation of a functor with tiny maps

lemma (**in** *is-tm-functor*) *tm-cf-ntcf-id-is-tm-ntcf*:
ntcf-id $\mathfrak{F} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
by (*intro is-tm-ntcfI′*) (*auto intro*: *cat-cs-intros cat-small-cs-intros*)

lemma (**in** *is-tm-functor*) *tm-cf-ntcf-id-is-tm-ntcf′*:
assumes $\mathfrak{F}' = \mathfrak{F}$ **and** $\mathfrak{G}' = \mathfrak{F}$
shows *ntcf-id* $\mathfrak{F} : \mathfrak{F}' \mapsto_{CF.tm} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
unfolding *assms*(*1,2*) **by** (*rule tm-cf-ntcf-id-is-tm-ntcf*)

lemmas [*cat-small-cs-intros*] = *is-tm-functor.tm-cf-ntcf-id-is-tm-ntcf′*

7.1.5 Constant natural transformation of functors with tiny maps

lemma *ntcf-const-is-tm-ntcf*:
assumes *tiny-category* α $\mathfrak{J}$ **and** *category* α $\mathfrak{C}$ **and** $f : a \mapsto_{\mathfrak{C}} b$
shows *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f :
cf-const $\mathfrak{J}$ $\mathfrak{C}$ $a \mapsto_{CF.tm}$ *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ $b : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
(**is** ‹*?Cf* : *?Ca* $\mapsto_{CF.tm}$ *?Cb* : $\mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$›)
proof(*intro is-tm-ntcfI′*)
interpret $\mathfrak{J}$: *tiny-category* α $\mathfrak{J}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
from *assms* **show**
?Cf : *?Ca* $\mapsto_{CF}$ *?Cb* : $\mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
cf-const $\mathfrak{J}$ $\mathfrak{C}$ $a : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
cf-const $\mathfrak{J}$ $\mathfrak{C}$ $b : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros*)+
qed

lemma *ntcf-const-is-tm-ntcf′*[*cat-small-cs-intros*]:
assumes *tiny-category* α $\mathfrak{J}$
and *category* α $\mathfrak{C}$
and $f : a \mapsto_{\mathfrak{C}} b$
and $\mathfrak{A}$ = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ a
and $\mathfrak{B}$ = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ b
and $\mathfrak{J}' = \mathfrak{J}$
and $\mathfrak{C}' = \mathfrak{C}$
shows *ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ $f : \mathfrak{A} \mapsto_{CF.tm} \mathfrak{B} : \mathfrak{J}' \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}'$
using *assms*(*1–3*) **unfolding** *assms*(*4–7*) **by** (*rule ntcf-const-is-tm-ntcf*)

7.1.6 Natural isomorphisms of functors with tiny maps

locale *is-tm-iso-ntcf* = *is-iso-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ + *is-tm-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$
for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

syntax *-is-tm-iso-ntcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- : - $\mapsto_{CF.tm.iso}$ - : - $\mapsto\mapsto_{C.tm1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-tm-iso-ntcf* ⇌ *is-tm-iso-ntcf*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$ ⇌
CONST is-tm-iso-ntcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

Rules.

mk-ide rf *is-tm-iso-ntcf-def*
|*intro is-tm-iso-ntcfI*|

```
  |dest is-tm-iso-ntcfD[dest]|
  |elim is-tm-iso-ntcfE[elim]|

lemmas [ntcf-cs-intros] = is-tm-iso-ntcfD

lemma iso-tm-ntcf-is-iso-arr:
  assumes category α 𝔅 and 𝔑 : 𝔉 ↦_{CF.tm.iso} 𝔊 : 𝔄 ↦↦_{C.tmα} 𝔅
  shows [ntcf-cs-intros]: inv-ntcf 𝔑 : 𝔊 ↦_{CF.tm.iso} 𝔉 : 𝔄 ↦↦_{C.tmα} 𝔅
    and 𝔑 ⋆_{NTCF} inv-ntcf 𝔑 = ntcf-id 𝔊
    and inv-ntcf 𝔑 ⋆_{NTCF} 𝔑 = ntcf-id 𝔉
proof–
  interpret 𝔅: category α 𝔅 by (rule assms(1))
  interpret 𝔑: is-tm-iso-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms)
  note inv-𝔑 = iso-ntcf-is-iso-arr[OF 𝔑.is-iso-ntcf-axioms]
  show inv-ntcf 𝔑 : 𝔊 ↦_{CF.tm.iso} 𝔉 : 𝔄 ↦↦_{C.tmα} 𝔅
  proof(intro is-tm-iso-ntcfI)
    show inv-ntcf 𝔑 : 𝔊 ↦_{CF.iso} 𝔉 : 𝔄 ↦↦_{Cα} 𝔅 by (intro inv-𝔑(1))
    interpret inv-𝔑: is-iso-ntcf α 𝔄 𝔅 𝔊 𝔉 ‹inv-ntcf 𝔑› by (rule inv-𝔑(1))
    show inv-ntcf 𝔑 : 𝔊 ↦_{CF.tm} 𝔉 : 𝔄 ↦↦_{C.tmα} 𝔅
      by (intro is-tm-ntcfI′) (auto intro: cat-cs-intros cat-small-cs-intros)
  qed
  show 𝔑 ⋆_{NTCF} inv-ntcf 𝔑 = ntcf-id 𝔊 inv-ntcf 𝔑 ⋆_{NTCF} 𝔑 = ntcf-id 𝔉
    by (intro inv-𝔑(2,3))+
qed

lemma is-iso-arr-is-tm-iso-ntcf:
  assumes 𝔑 : 𝔉 ↦_{CF.tm} 𝔊 : 𝔄 ↦↦_{C.tmα} 𝔅
    and 𝔐 : 𝔊 ↦_{CF.tm} 𝔉 : 𝔄 ↦↦_{C.tmα} 𝔅
    and [simp]: 𝔑 ⋆_{NTCF} 𝔐 = ntcf-id 𝔊
    and [simp]: 𝔐 ⋆_{NTCF} 𝔑 = ntcf-id 𝔉
  shows 𝔑 : 𝔉 ↦_{CF.tm.iso} 𝔊 : 𝔄 ↦↦_{C.tmα} 𝔅
proof–
  interpret 𝔑: is-tm-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret 𝔐: is-tm-ntcf α 𝔄 𝔅 𝔊 𝔉 𝔐 by (rule assms(2))
  show ?thesis
  proof(rule is-tm-iso-ntcfI)
    show 𝔑 : 𝔉 ↦_{CF.iso} 𝔊 : 𝔄 ↦↦_{Cα} 𝔅
      by (rule is-iso-arr-is-iso-ntcf) (auto intro: cat-small-cs-intros)
    show 𝔑 : 𝔉 ↦_{CF.tm} 𝔊 : 𝔄 ↦↦_{C.tmα} 𝔅
      by (rule is-tm-ntcfI′)
        (auto simp: 𝔑.tm-ntcf-NTMap-in-Vset intro: cat-small-cs-intros)
  qed
qed
```

7.1.7 Composition of a natural transformation of functors with tiny maps and a functor with tiny maps

```
lemma ntcf-cf-comp-is-tm-ntcf:
  assumes 𝔑 : 𝔉 ↦_{CF.tm} 𝔊 : 𝔅 ↦↦_{C.tmα} ℭ and ℌ : 𝔄 ↦↦_{C.tmα} 𝔅
  shows 𝔑 ∘_{NTCF-CF} ℌ : 𝔉 ∘_{CF} ℌ ↦_{CF.tm} 𝔊 ∘_{CF} ℌ : 𝔄 ↦↦_{C.tmα} ℭ
proof–
  interpret 𝔑: is-tm-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret ℌ: is-tm-functor α 𝔄 𝔅 ℌ by (rule assms(2))
  from assms show ?thesis
    by (intro is-tm-ntcfI)
      (
        cs-concl cs-shallow
          cs-simp: slicing-commute[symmetric]
```

cs-intro: *cat-cs-intros smc-small-cs-intros slicing-intros*
)+
qed

lemma *ntcf-cf-comp-is-tm-ntcf'*[*cat-small-cs-intros*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{F}' = \mathfrak{F} \circ_{CF} \mathfrak{H}$
and $\mathfrak{G}' = \mathfrak{G} \circ_{CF} \mathfrak{H}$
shows $\mathfrak{N} \circ_{NTCF-CF} \mathfrak{H} : \mathfrak{F}' \mapsto_{CF.tm} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
using *assms*(*1,2*) **unfolding** *assms*(*3,4*) **by** (*rule ntcf-cf-comp-is-tm-ntcf*)

7.1.8 Composition of a functor with tiny maps and a natural transformation of functors with tiny maps

lemma *cf-ntcf-comp-is-tm-ntcf*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
shows $\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{H} \circ_{CF} \mathfrak{F} \mapsto_{CF.tm} \mathfrak{H} \circ_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{H}$: *is-tm-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-tm-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))
from *assms* **show** *?thesis*
by (*intro is-tm-ntcfI*)
(
cs-concl **cs-shallow**
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *cat-cs-intros smc-small-cs-intros slicing-intros*
)+
qed

lemma *cf-ntcf-comp-is-tm-ntcf'*[*cat-small-cs-intros*]:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{F}' = \mathfrak{H} \circ_{CF} \mathfrak{F}$
and $\mathfrak{G}' = \mathfrak{H} \circ_{CF} \mathfrak{G}$
shows $\mathfrak{H} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{F}' \mapsto_{CF.tm} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
using *assms*(*1,2*) **unfolding** *assms*(*3,4*) **by** (*rule cf-ntcf-comp-is-tm-ntcf*)

7.2 Tiny natural transformation of functors

7.2.1 Definition and elementary properties

locale *is-tiny-ntcf* = *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ +
assumes *tiny-ntcf-is-tiny-ntsmcf*:
ntcf-ntsmcf $\mathfrak{N}$:
cf-smcf $\mathfrak{F} \mapsto_{SMCF.tiny}$ *cf-smcf* $\mathfrak{G}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha}$ *cat-smc* $\mathfrak{B}$

syntax *-is-tiny-ntcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\mapsto_{CF.tiny}$ - :/ - $\mapsto\mapsto_{C.tiny1}$ -)› [*51, 51, 51, 51, 51*] *51*)
syntax-consts *-is-tiny-ntcf* ⇌ *is-tiny-ntcf*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$ ⇌
CONST is-tiny-ntcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

abbreviation *all-tiny-ntcfs* :: $V \Rightarrow V$
where *all-tiny-ntcfs* $\alpha \equiv$
set {$\mathfrak{N}$. $\exists \mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$. $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$}

abbreviation *tiny-ntcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *tiny-ntcfs* α $\mathfrak{A}$ $\mathfrak{B} \equiv$

set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}\}$

abbreviation *these-tiny-ntcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *these-tiny-ntcfs* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G} \equiv$
set $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}\}$

lemma (**in** *is-tiny-ntcf*) *tiny-ntcf-is-tiny-ntsmcf'*:
assumes $\alpha' = \alpha$
and $\mathfrak{F}' =$ *cf-smcf* $\mathfrak{F}$
and $\mathfrak{G}' =$ *cf-smcf* $\mathfrak{G}$
and $\mathfrak{A}' =$ *cat-smc* $\mathfrak{A}$
and $\mathfrak{B}' =$ *cat-smc* $\mathfrak{B}$
shows *ntcf-ntsmcf* $\mathfrak{N} : \mathfrak{F}' \mapsto_{SMCF.tiny} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{SMC.tiny_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule tiny-ntcf-is-tiny-ntsmcf*)

lemmas [*slicing-intros*] = *is-tiny-ntcf.tiny-ntcf-is-tiny-ntsmcf'*

Rules.

lemma (**in** *is-tiny-ntcf*) *is-tiny-ntcf-axioms'*[*cat-small-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$ **and** $\mathfrak{F}' = \mathfrak{F}$ **and** $\mathfrak{G}' = \mathfrak{G}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule is-tiny-ntcf-axioms*)

mk-ide rf *is-tiny-ntcf-def*[*unfolded is-tiny-ntcf-axioms-def*]
|*intro is-tiny-ntcfI*|
|*dest is-tiny-ntcfD*[*dest*]|
|*elim is-tiny-ntcfE*[*elim*]|

Elementary properties.

sublocale *is-tiny-ntcf* ⊆ *NTDom*: *is-tiny-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$
using *tiny-ntcf-is-tiny-ntsmcf*
by (*intro is-tiny-functorI*)
(*auto intro*: *cat-cs-intros simp*: *is-tiny-ntsmcf-iff*)

sublocale *is-tiny-ntcf* ⊆ *NTCod*: *is-tiny-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$
using *tiny-ntcf-is-tiny-ntsmcf*
by (*intro is-tiny-functorI*)
(*auto intro*: *cat-cs-intros simp*: *is-tiny-ntsmcf-iff*)

sublocale *is-tiny-ntcf* ⊆ *is-tm-ntcf*
by (*rule is-tm-ntcfI'*) (*auto intro*: *cat-cs-intros cat-small-cs-intros*)

lemmas (**in** *is-tiny-ntcf*) *tiny-ntcf-is-tm-ntcf*[*cat-small-cs-intros*] =
is-tm-ntcf-axioms

lemmas [*cat-small-cs-intros*] = *is-tiny-ntcf.tiny-ntcf-is-tm-ntcf*

Further rules.

lemma *is-tiny-ntcfI'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-tiny-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}$: *is-tiny-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$ **by** (*rule assms*(*3*))

```
  show 𝔑 : 𝔉 ↦_{CF.tiny} 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
    by (intro is-tiny-ntcfI is-tiny-ntsmcfI')
      (auto intro: cat-cs-intros slicing-intros)
qed

lemma is-tiny-ntcfD':
  assumes 𝔑 : 𝔉 ↦_{CF.tiny} 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
  shows 𝔑 : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{C α} 𝔅
    and 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅
    and 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
proof–
  interpret 𝔑: is-tiny-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
  show 𝔑 : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{C α} 𝔅
    and 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅
    and 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
    by (auto intro: cat-small-cs-intros)
qed

lemmas [cat-small-cs-intros] = is-tiny-ntcfD'(2,3)

lemma is-tiny-ntcfE':
  assumes 𝔑 : 𝔉 ↦_{CF.tiny} 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
  obtains 𝔑 : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{C α} 𝔅
    and 𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅
    and 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
  using assms by (auto dest: is-tiny-ntcfD'(2,3))

lemma is-tiny-ntcf-iff:
  𝔑 : 𝔉 ↦_{CF.tiny} 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅 ⟷
    (
      𝔑 : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{C α} 𝔅 ∧
      𝔉 : 𝔄 ↦↦_{C.tiny α} 𝔅 ∧
      𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅
    )
   by (auto intro: is-tiny-ntcfI' dest: is-tiny-ntcfD'(2,3))

lemma (in is-tiny-ntcf) tiny-ntcf-in-Vset: 𝔑 ∈ₒ Vset α
proof–
  note [cat-cs-intros] =
    tm-ntcf-NTMap-in-Vset
    NTDom.tiny-cf-in-Vset
    NTCod.tiny-cf-in-Vset
    NTDom.HomDom.tiny-cat-in-Vset
    NTDom.HomCod.tiny-cat-in-Vset
  show ?thesis
    by (subst ntcf-def)
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros V-cs-intros
      )
qed

lemma small-all-tiny-ntcfs[simp]:
  small {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_{CF.tiny} 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅}
proof(rule down)
  show {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_{CF.tiny} 𝔊 : 𝔄 ↦↦_{C.tiny α} 𝔅} ⊆
    elts (set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{C α} 𝔅})
  proof
```

```
    (
      simp only: elts-of-set small-all-ntcfs if-True,
      rule subsetI,
      unfold mem-Collect-eq
    )
    fix 𝔑 assume ∃ℱ 𝒢 𝒜 ℬ. 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ
    then obtain ℱ 𝒢 𝒜 ℬ where 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ
      by clarsimp
    then interpret is-tiny-ntcf α 𝒜 ℬ ℱ 𝒢 𝔑 .
    have 𝔑 : ℱ ↦_CF 𝒢 : 𝒜 ↦↦_Cα ℬ by (auto intro: cat-cs-intros)
    then show ∃ℱ 𝒢 𝒜 ℬ. 𝔑 : ℱ ↦_CF 𝒢 : 𝒜 ↦↦_Cα ℬ by auto
  qed
qed

lemma small-tiny-ntcfs[simp]:
  small {𝔑. ∃ℱ 𝒢. 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ}
  by
    (
      rule
        down[
          of - ‹set {𝔑. ∃ℱ 𝒢 𝒜 ℬ. 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ}›
          ]
    )
    auto

lemma small-these-tiny-ntcfs[simp]:
  small {𝔑. 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ}
  by
    (
      rule
        down[
          of - ‹set {𝔑. ∃ℱ 𝒢 𝒜 ℬ. 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ}›
          ]
    )
    auto

lemma tiny-ntcfs-vsubset-Vset[simp]:
  set {𝔑. ∃ℱ 𝒢. 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ} ⊆∘ Vset α
  (is ‹set ?ntcfs ⊆∘ -›)
proof(cases ‹tiny-category α 𝒜 ∧ tiny-category α ℬ›)
  case True
  then have tiny-category α 𝒜 and tiny-category α ℬ by auto
  show ?thesis
  proof(rule vsubsetI)
    fix 𝔑 assume 𝔑 ∈∘ set ?ntcfs
    then obtain ℱ 𝒢 where 𝔑 : ℱ ↦_CF.tiny 𝒢 : 𝒜 ↦↦_C.tinyα ℬ by auto
    then interpret is-tiny-ntcf α 𝒜 ℬ ℱ 𝒢 𝔑 by simp
    from tiny-ntcf-in-Vset show 𝔑 ∈∘ Vset α by simp
  qed
next
  case False
  then have set ?ntcfs = 0
    unfolding is-tiny-ntcf-iff is-tiny-functor-iff by auto
  then show ?thesis by simp
qed
```

Further elementary results.

```
lemma these-tiny-ntcfs-iff:
```

$\mathfrak{N} \in_{\circ}$ *these-tiny-ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G} \longleftrightarrow \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
by *auto*

Size.

lemma (**in** *is-ntcf*) *ntcf-is-tiny-ntcf-if-ge-Limit*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_{\circ} \beta$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\beta} \mathfrak{B}$
proof(*intro is-tiny-ntcfI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\beta} \mathfrak{B}$
by (*intro ntcf-is-ntcf-if-ge-Limit*)
(*use assms*(*2*) **in** ‹*cs-concl cs-shallow cs-intro*: *dg-cs-intros*›)+
show *ntcf-ntsmcf* $\mathfrak{N}$:
cf-smcf $\mathfrak{F} \mapsto_{SMCF.tiny}$ *cf-smcf* $\mathfrak{G}$: *cat-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.tiny\beta}$ *cat-smc* $\mathfrak{B}$
by
(
rule is-ntsmcf.ntsmcf-is-tiny-ntsmcf-if-ge-Limit,
rule ntcf-is-ntsmcf;
intro assms
)
qed

7.2.2 Opposite natural transformation of tiny functors

lemma (**in** *is-tiny-ntcf*) *is-tm-ntcf-op*: *op-ntcf* $\mathfrak{N}$:
op-cf $\mathfrak{G} \mapsto_{CF.tiny}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{A} \mapsto\mapsto_{C.tiny\alpha}$ *op-cat* $\mathfrak{B}$
by (*intro is-tiny-ntcfI'*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)+

lemma (**in** *is-tiny-ntcf*) *is-tiny-ntcf-op'*[*cat-op-intros*]:
assumes $\mathfrak{G}'$ = *op-cf* $\mathfrak{G}$
and $\mathfrak{F}'$ = *op-cf* $\mathfrak{F}$
and $\mathfrak{A}'$ = *op-cat* $\mathfrak{A}$
and $\mathfrak{B}'$ = *op-cat* $\mathfrak{B}$
shows *op-ntcf* $\mathfrak{N} : \mathfrak{G}' \mapsto_{CF.tiny} \mathfrak{F}' : \mathfrak{A}' \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-tm-ntcf-op*)

lemmas *is-tiny-ntcf-op*[*cat-op-intros*] = *is-tiny-ntcf.is-tiny-ntcf-op'*

7.2.3 Vertical composition of tiny natural transformations

lemma *ntsmcf-vcomp-is-tiny-ntsmcf*[*cat-small-cs-intros*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF.tiny} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{M}$: *is-tiny-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-tiny-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*2*))
show *?thesis* **by** (*rule is-tiny-ntcfI'*) (*auto intro*: *cat-small-cs-intros*)
qed

7.2.4 Tiny identity natural transformation

lemma (**in** *is-tiny-functor*) *tiny-cf-ntcf-id-is-tiny-ntcf*:
ntcf-id $\mathfrak{F} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
by (*intro is-tiny-ntcfI'*) (*auto intro*: *cat-small-cs-intros*)

lemma (**in** *is-tiny-functor*) *tiny-cf-ntcf-id-is-tiny-ntcf'*[*cat-small-cs-intros*]:

assumes $\mathfrak{F}' = \mathfrak{F}$ **and** $\mathfrak{G}' = \mathfrak{F}$
shows *ntcf-id* $\mathfrak{F} : \mathfrak{F}' \mapsto_{CF.tiny} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule tiny-cf-ntcf-id-is-tiny-ntcf*)

lemmas [*cat-small-cs-intros*] = *is-tiny-functor.tiny-cf-ntcf-id-is-tiny-ntcf'*

7.3 Tiny natural isomorphisms

7.3.1 Definition and elementary properties

locale *is-tiny-iso-ntcf* = *is-iso-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ + *is-tiny-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$
for $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$

syntax *-is-tiny-iso-ntcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- : - $\mapsto_{CF.tiny.iso}$ - : - $\mapsto\mapsto_{C.tiny^1}$ -)$\rangle$ [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-tiny-iso-ntcf* $\rightleftharpoons$ *is-tiny-iso-ntcf*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B} \rightleftharpoons$
CONST is-tiny-iso-ntcf $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$

Rules.

mk-ide rf *is-tiny-iso-ntcf-def*
|*intro is-tiny-iso-ntcfI*|
|*dest is-tiny-iso-ntcfD*[*dest*]|
|*elim is-tiny-iso-ntcfE*[*elim*]|

lemmas [*ntcf-cs-intros*] = *is-tiny-iso-ntcfD*(*2*)

Elementary properties.

sublocale *is-tiny-iso-ntcf* $\subseteq$ *is-tm-iso-ntcf*
by (*rule is-tm-iso-ntcfI*) (*auto intro*: *cat-cs-intros cat-small-cs-intros*)

lemmas (**in** *is-tiny-iso-ntcf*) *is-tm-iso-ntcf-axioms'* = *is-tm-iso-ntcf-axioms*

lemmas [*ntcf-cs-intros*] = *is-tiny-iso-ntcf.is-tm-iso-ntcf-axioms'*

Further rules.

lemma *is-tiny-iso-ntcfI'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{N}$: *is-iso-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-tiny-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}$: *is-tiny-functor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$ **by** (*rule assms*(*3*))
show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
by (*intro is-tiny-iso-ntcfI is-tiny-ntcfI'*)
(*auto intro*: *cat-cs-intros cat-small-cs-intros*)
qed

lemma *is-tiny-iso-ntcfD'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{N}$: *is-tiny-iso-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(*1*))
show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
by (*auto intro*: *cat-cs-intros cat-small-cs-intros*)
qed

lemma *is-tiny-iso-ntcfE'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
obtains $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
using *assms* **by** (*auto dest*: *is-tiny-ntcfD'*(*2*,*3*))

lemma *is-tiny-iso-ntcf-iff*:
$\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B} \longleftrightarrow$
(
$\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B} \wedge$
$\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B} \wedge$
$\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
)
by (*auto intro*: *is-tiny-iso-ntcfI'* *dest*: *is-tiny-ntcfD'*(*2*,*3*))

7.3.2 Further properties

lemma *iso-tiny-ntcf-is-iso-arr*:
assumes *category* α $\mathfrak{B}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
shows [*ntcf-cs-intros*]: *inv-ntcf* $\mathfrak{N} : \mathfrak{G} \mapsto_{CF.tiny.iso} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{N}$ = *ntcf-id* $\mathfrak{G}$
and *inv-ntcf* $\mathfrak{N} \bullet_{NTCF} \mathfrak{N}$ = *ntcf-id* $\mathfrak{F}$
proof–
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-tiny-iso-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*)
note *inv-*$\mathfrak{N}$ = *iso-ntcf-is-iso-arr*[*OF* $\mathfrak{N}$.*is-iso-ntcf-axioms*]
show *inv-ntcf* $\mathfrak{N} : \mathfrak{G} \mapsto_{CF.tiny.iso} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof(*intro is-tiny-iso-ntcfI*)
show *inv-ntcf* $\mathfrak{N} : \mathfrak{G} \mapsto_{CF.iso} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** (*intro inv-*$\mathfrak{N}$(*1*))
interpret *inv-*$\mathfrak{N}$: *is-iso-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{F}$ ‹*inv-ntcf* $\mathfrak{N}$› **by** (*rule inv-*$\mathfrak{N}$(*1*))
show *inv-ntcf* $\mathfrak{N} : \mathfrak{G} \mapsto_{CF.tiny} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
by (*intro is-tiny-ntcfI'*) (*auto intro*: *cat-small-cs-intros cat-cs-intros*)
qed
show $\mathfrak{N} \bullet_{NTCF}$ *inv-ntcf* $\mathfrak{N}$ = *ntcf-id* $\mathfrak{G}$ *inv-ntcf* $\mathfrak{N} \bullet_{NTCF} \mathfrak{N}$ = *ntcf-id* $\mathfrak{F}$
by (*intro inv-*$\mathfrak{N}$(*2*,*3*))+
qed

lemma *is-iso-arr-is-tiny-iso-ntcf*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and $\mathfrak{M} : \mathfrak{G} \mapsto_{CF.tiny} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
and [*simp*]: $\mathfrak{N} \bullet_{NTCF} \mathfrak{M}$ = *ntcf-id* $\mathfrak{G}$
and [*simp*]: $\mathfrak{M} \bullet_{NTCF} \mathfrak{N}$ = *ntcf-id* $\mathfrak{F}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{N}$: *is-tiny-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{M}$: *is-tiny-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{F}$ $\mathfrak{M}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule is-tiny-iso-ntcfI*)
show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*rule is-iso-arr-is-iso-ntcf*) (*auto intro*: *cat-small-cs-intros*)
show $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tiny\alpha} \mathfrak{B}$
by (*rule is-tiny-ntcfI'*) (*auto intro*: *cat-small-cs-intros*)

qed
qed

8 Product category

8.1 Background

See Chapter II-3 in [7].

named-theorems *cat-prod-cs-simps*
named-theorems *cat-prod-cs-intros*

8.2 Product category: definition and elementary properties

definition *cat-prod* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  where *cat-prod* I $\mathfrak{A}$ =
    [
      $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!))$,
      $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!))$,
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Dom|\!))(\!|f(\!|i|\!)|\!)))$,
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Cod|\!))(\!|f(\!|i|\!)|\!)))$,
      (
        $\lambda gf \in_{\circ} composable\text{-}arrs\ (dg\text{-}prod\ I\ \mathfrak{A})$.
          $(\lambda i \in_{\circ} I.\ vpfst\ gf(\!|i|\!)\ \circ_{A\mathfrak{A}\ i}\ vpsnd\ gf(\!|i|\!))$
      ),
      $(\lambda a \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|CId|\!))(\!|a(\!|i|\!)|\!)))$
    ]$_{\circ}$

syntax *-PCATEGORY* :: *pttrn* $\Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  ($\langle$($3\prod_{C}$-$\in_{\circ}$-./ -)$\rangle$ [*0*, *0*, *10*] *10*)
syntax-consts *-PCATEGORY* $\rightleftharpoons$ *cat-prod*
translations $\prod_{C} i \in_{\circ} I.\ \mathfrak{A} \rightleftharpoons CONST\ cat\text{-}prod\ I\ (\lambda i.\ \mathfrak{A})$

Components.

lemma *cat-prod-components*:
  shows $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!) = (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!))$
    and $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!) = (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!))$
    and $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Dom|\!) =$
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Dom|\!))(\!|f(\!|i|\!)|\!)))$
    and $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Cod|\!) =$
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Cod|\!))(\!|f(\!|i|\!)|\!)))$
    and $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Comp|\!) =$
      (
        $\lambda gf \in_{\circ} composable\text{-}arrs\ (dg\text{-}prod\ I\ \mathfrak{A})$.
          $(\lambda i \in_{\circ} I.\ vpfst\ gf(\!|i|\!)\ \circ_{A\mathfrak{A}\ i}\ vpsnd\ gf(\!|i|\!))$
)
    and $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!) =$
      $(\lambda a \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|CId|\!))(\!|a(\!|i|\!)|\!)))$
  unfolding *cat-prod-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-cat-prod*[*slicing-commute*]:
  smc-prod I $(\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i)) = cat\text{-}smc\ (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)$
  unfolding *dg-prod-def cat-smc-def cat-prod-def smc-prod-def dg-field-simps*
  by (*simp-all add*: *nat-omega-simps*)

context
  fixes $\mathfrak{A}$ $\varphi :: V \Rightarrow V$
    and $\mathfrak{C} :: V$
begin

lemmas-with [
where $\mathfrak{A}$=‹λi. *cat-smc* ($\mathfrak{A}$ *i*)›, *unfolded slicing-simps slicing-commute*
]:
cat-prod-ObjI = *smc-prod-ObjI*
and *cat-prod-ObjD* = *smc-prod-ObjD*
and *cat-prod-ObjE* = *smc-prod-ObjE*
and *cat-prod-Obj-cong* = *smc-prod-Obj-cong*
and *cat-prod-ArrI* = *smc-prod-ArrI*
and *cat-prod-ArrD* = *smc-prod-ArrD*
and *cat-prod-ArrE* = *smc-prod-ArrE*
and *cat-prod-Arr-cong* = *smc-prod-Arr-cong*
and *cat-prod-Dom-vsv*[*cat-cs-intros*] = *smc-prod-Dom-vsv*
and *cat-prod-Dom-vdomain*[*cat-cs-simps*] = *smc-prod-Dom-vdomain*
and *cat-prod-Dom-app* = *smc-prod-Dom-app*
and *cat-prod-Dom-app-component-app*[*cat-cs-simps*] =
smc-prod-Dom-app-component-app
and *cat-prod-Cod-vsv*[*cat-cs-intros*] = *smc-prod-Cod-vsv*
and *cat-prod-Cod-app* = *smc-prod-Cod-app*
and *cat-prod-Cod-vdomain*[*cat-cs-simps*] = *smc-prod-Cod-vdomain*
and *cat-prod-Cod-app-component-app*[*cat-cs-simps*] =
smc-prod-Cod-app-component-app
and *cat-prod-Comp* = *smc-prod-Comp*
and *cat-prod-Comp-vdomain*[*cat-cs-simps*] = *smc-prod-Comp-vdomain*
and *cat-prod-Comp-app* = *smc-prod-Comp-app*
and *cat-prod-Comp-app-component*[*cat-cs-simps*] =
smc-prod-Comp-app-component
and *cat-prod-Comp-app-vdomain* = *smc-prod-Comp-app-vdomain*
and *cat-prod-vunion-Obj-in-Obj* = *smc-prod-vunion-Obj-in-Obj*
and *cat-prod-vdiff-vunion-Obj-in-Obj* = *smc-prod-vdiff-vunion-Obj-in-Obj*
and *cat-prod-vunion-Arr-in-Arr* = *smc-prod-vunion-Arr-in-Arr*
and *cat-prod-vdiff-vunion-Arr-in-Arr* = *smc-prod-vdiff-vunion-Arr-in-Arr*

end

8.3 Local assumptions for a product category

locale *pcategory-base* = $\mathcal{Z}$ α **for** α *I* $\mathfrak{A}$ +
assumes *pcat-categories*: *i* $\in_\circ$ *I* $\Longrightarrow$ *category* α ($\mathfrak{A}$ *i*)
and *pcat-index-in-Vset*[*cat-cs-intros*]: *I* $\in_\circ$ *Vset* α

lemma (**in** *pcategory-base*) *pcat-categories'*[*cat-prod-cs-intros*]:
assumes *i* $\in_\circ$ *I* **and** α' = α
shows *category* α' ($\mathfrak{A}$ *i*)
using *assms*(*1*) **unfolding** *assms*(*2*) **by** (*rule pcat-categories*)

Rules.

lemma (**in** *pcategory-base*) *pcategory-base-axioms'*[*cat-prod-cs-intros*]:
assumes α' = α **and** *I'* = *I*
shows *pcategory-base* α' *I'* $\mathfrak{A}$
unfolding *assms* **by** (*rule pcategory-base-axioms*)

mk-ide rf *pcategory-base-def*[*unfolded pcategory-base-axioms-def*]
|*intro pcategory-baseI*|
|*dest pcategory-baseD*[*dest*]|
|*elim pcategory-baseE*[*elim*]|

lemma *pcategory-base-psemicategory-baseI*:
assumes *psemicategory-base* α *I* (λi. *cat-smc* ($\mathfrak{A}$ *i*))

and $\bigwedge i.\ i \in_\circ I \Longrightarrow$ *category* α $(\mathfrak{A}\ i)$
shows *pcategory-base* α I $\mathfrak{A}$
proof–
interpret *psemicategory-base* α I $\langle\lambda i.$ *cat-smc* $(\mathfrak{A}\ i)\rangle$ **by** (*rule assms*(*1*))
show *?thesis*
by (*intro pcategory-baseI*)
(*auto simp*: *assms*(*2*) *psmc-index-in-Vset psmc-Obj-in-Vset psmc-Arr-in-Vset*)
qed

Product category is a product semicategory.

context *pcategory-base*
begin

lemma *pcat-psemicategory-base*: *psemicategory-base* α I $(\lambda i.$ *cat-smc* $(\mathfrak{A}\ i))$
proof(*intro psemicategory-baseI*)
from *pcat-index-in-Vset* **show** $I \in_\circ$ *Vset* α **by** *auto*
qed (*auto simp*: *category.cat-semicategory cat-prod-cs-intros*)

interpretation *psmc*: *psemicategory-base* α I $\langle\lambda i.$ *cat-smc* $(\mathfrak{A}\ i)\rangle$
by (*rule pcat-psemicategory-base*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
pcat-Obj-in-Vset = *psmc.psmc-Obj-in-Vset*
and *pcat-Arr-in-Vset* = *psmc.psmc-Arr-in-Vset*
and *pcat-smc-prod-Obj-in-Vset* = *psmc.psmc-smc-prod-Obj-in-Vset*
and *pcat-smc-prod-Arr-in-Vset* = *psmc.psmc-smc-prod-Arr-in-Vset*
and *cat-prod-Dom-app-in-Obj*[*cat-cs-intros*] = *psmc.smc-prod-Dom-app-in-Obj*
and *cat-prod-Cod-app-in-Obj*[*cat-cs-intros*] = *psmc.smc-prod-Cod-app-in-Obj*
and *cat-prod-is-arrI* = *psmc.smc-prod-is-arrI*
and *cat-prod-is-arrD*[*dest*] = *psmc.smc-prod-is-arrD*
and *cat-prod-is-arrE*[*elim*] = *psmc.smc-prod-is-arrE*

end

lemma *cat-prod-dg-prod-is-arr*:
$g : b \mapsto_{dg\text{-}prod\ I\ \mathfrak{A}}\ c \longleftrightarrow g : b \mapsto_{(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)}\ c$
unfolding *is-arr-def cat-prod-def smc-prod-def dg-prod-def dg-field-simps*
by (*simp add*: *nat-omega-simps*)

lemma *smc-prod-composable-arrs-dg-prod*:
composable-arrs (*dg-prod* I $\mathfrak{A}$) = *composable-arrs* $(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$
unfolding *composable-arrs-def cat-prod-dg-prod-is-arr* **by** *simp*

Elementary properties.

lemma (**in** *pcategory-base*) *pcat-vsubset-index-pcategory-base*:
assumes $J \subseteq_\circ I$
shows *pcategory-base* α J $\mathfrak{A}$
proof(*intro pcategory-baseI*)
show *category* α $(\mathfrak{A}\ i)$ **if** $i \in_\circ J$ **for** i
using *that assms* **by** (*auto intro*: *cat-prod-cs-intros*)
from *assms* **show** $J \in_\circ$ *Vset* α **by** (*simp add*: *vsubset-in-VsetI cat-cs-intros*)
qed *auto*

8.3.1 Identity

lemma *cat-prod-CId-vsv*[*cat-cs-intros*]: *vsv* $((\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{CId}|\!))$
unfolding *cat-prod-components* **by** *auto*

lemma *cat-prod-CId-vdomain*[*cat-cs-simps*]:
  $\mathcal{D}_\circ\ ((\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)) = (\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
  unfolding *cat-prod-components* **by** *simp*

lemma *cat-prod-CId-app*:
  assumes $a \in_\circ (\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
  shows $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|a|\!) = (\lambda i \in_\circ I.\ \mathfrak{A}\ i(\!|CId|\!)(\!|a(\!|i|\!)|\!))$
  using *assms* **unfolding** *cat-prod-components* **by** *simp*

lemma *cat-prod-CId-app-component*[*cat-cs-simps*]:
  assumes $a \in_\circ (\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$ **and** $i \in_\circ I$
  shows $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|a|\!)(\!|i|\!) = \mathfrak{A}\ i(\!|CId|\!)(\!|a(\!|i|\!)|\!)$
  using *assms* **unfolding** *cat-prod-components* **by** *simp*

lemma (**in** *pcategory-base*) *cat-prod-CId-vrange*:
  $\mathcal{R}_\circ\ ((\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)) \subseteq_\circ (\prod_{\circ} i \in_\circ I.\ \mathfrak{A}\ i(\!|Arr|\!))$
proof(*intro vsubsetI*)
  interpret *CId*: *vsv* ‹$((\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!))$› **by** (*rule cat-prod-CId-vsv*)
  fix *f* **assume** $f \in_\circ \mathcal{R}_\circ\ ((\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!))$
  then obtain *a* **where** *f-def*: $f = ((\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!))(\!|a|\!)$
    and $a \in_\circ \mathcal{D}_\circ\ ((\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!))$
    by (*blast dest*: *CId.vrange-atD*)
  then have *a*: $a \in_\circ (\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
    unfolding *cat-prod-components* **by** *simp*
  show $f \in_\circ (\prod_{\circ} i \in_\circ I.\ \mathfrak{A}\ i(\!|Arr|\!))$
    unfolding *f-def cat-prod-CId-app*[*OF a*]
  proof(*rule VLambda-in-vproduct*)
    fix *i* **assume** *prems*: $i \in_\circ I$
    interpret $\mathfrak{A}$: *category* α ‹$\mathfrak{A}\ i$›
      by (*simp add*: ‹$i \in_\circ I$› *cat-cs-intros cat-prod-cs-intros*)
    from *prems a* **have** $a(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Obj|\!)$ **unfolding** *cat-prod-components* **by** *auto*
    with *is-arrD*(*1*) **show** $\mathfrak{A}\ i(\!|CId|\!)(\!|a(\!|i|\!)|\!) \in_\circ \mathfrak{A}\ i(\!|Arr|\!)$
      by (*auto intro*: *cat-cs-intros*)
  qed
qed

8.3.2 A product α-category is a tiny β-category

lemma (**in** *pcategory-base*) *pcat-tiny-category-cat-prod*:
  assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
  shows *tiny-category* β $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)$
proof–

  interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(*1*))

  show *?thesis*
  proof(*intro tiny-categoryI*, (*unfold slicing-simps*)*?*)

    show Π: *tiny-semicategory* β (*cat-smc* $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)$)
      unfolding *slicing-commute*[*symmetric*]
      by
        (
          intro psemicategory-base.psmc-tiny-semicategory-smc-prod;
          (*rule assms pcat-psemicategory-base*)*?*
)
    interpret Π: *tiny-semicategory* β ‹*cat-smc* $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)$› **by** (*rule* Π)

    show *vfsequence* $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)$ **unfolding** *cat-prod-def* **by** *auto*

show *vcard* $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i) = 6_{\mathbb{N}}$
unfolding *cat-prod-def* **by** (*simp add*: *nat-omega-simps*)

show *CId*: $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|a|\!) : a \mapsto_{(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)} a$
if *a*: $a \in_{\circ} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$ **for** *a*
proof(*rule cat-prod-is-arrI*)
have [*cat-cs-intros*]: $a(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Obj|\!)$ **if** *i*: $i \in_{\circ} I$ **for** *i*
by (*rule cat-prod-ObjD*(*3*)[*OF a i*])
from *that* **show** $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|a|\!)(\!|i|\!) : a(\!|i|\!) \mapsto_{\mathfrak{A}\ i} a(\!|i|\!)$
if $i \in_{\circ} I$ **for** *i*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros that*
)
qed (*use that* **in** ‹*auto simp*: *cat-prod-components cat-prod-CId-app that*›)

show $(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) \circ_{A} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)\ f = f$
if $f : a \mapsto_{(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)} b$ **for** *f a b*
proof(*rule cat-prod-Arr-cong*)
note $f = \Pi.smc\text{-}is\text{-}arrD[unfolded\ slicing\text{-}simps,\ OF\ that]$
note $a = f(2)$ **and** $b = f(3)$ **and** $f = f(1)$
from *CId*[*OF b*] **have** *CId-b*:
$(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) : b \mapsto_{(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)} b$
by *simp*
from $\Pi.smc\text{-}Comp\text{-}is\text{-}arr[unfolded\ slicing\text{-}simps,\ OF\ this\ that]$ **show**
$(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) \circ_{A} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)\ f \in_{\circ} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
by (*simp add*: *cat-cs-intros*)
from *that* **show** $f \in_{\circ} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$ **by** *auto*
fix *i* **assume** *prems*: $i \in_{\circ} I$
interpret $\mathfrak{A}i$: *category* α ‹$\mathfrak{A}\ i$› **by** (*simp add*: *prems cat-prod-cs-intros*)
from *prems cat-prod-is-arrD*(*7*)[*OF that*] **have** *fi*:
$f(\!|i|\!) : a(\!|i|\!) \mapsto_{\mathfrak{A}\ i} b(\!|i|\!)$
by *auto*
from *prems* **show** $((\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) \circ_{A} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)\ f)(\!|i|\!) = f(\!|i|\!)$
unfolding *cat-prod-Comp-app-component*[*OF CId-b that prems*]
unfolding *cat-prod-CId-app*[*OF b*]
by (*auto intro*: $\mathfrak{A}i.cat\text{-}CId\text{-}left\text{-}left$[*OF fi*])
qed

show $f \circ_{A} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)\ (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) = f$
if $f : b \mapsto_{(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)} c$ **for** *f b c*
proof(*rule cat-prod-Arr-cong*)
note $f = \Pi.smc\text{-}is\text{-}arrD[unfolded\ slicing\text{-}simps,\ OF\ that]$
note $b = f(2)$ **and** $c = f(3)$ **and** $f = f(1)$
from *CId*[*OF b*] **have** *CId-b*:
$(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) : b \mapsto_{(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)} b$
by *simp*
from $\Pi.smc\text{-}Comp\text{-}is\text{-}arr[unfolded\ slicing\text{-}simps,\ OF\ that\ this]$ **show**
$f \circ_{A} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)\ (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|b|\!) \in_{\circ} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
by (*simp add*: *cat-cs-intros*)
from *that* **show** $f \in_{\circ} (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$ **by** *auto*
fix *i* **assume** *prems*: $i \in_{\circ} I$
interpret $\mathfrak{A}i$: *category* α ‹$\mathfrak{A}\ i$› **by** (*simp add*: *prems cat-prod-cs-intros*)
from *prems cat-prod-is-arrD*[*OF that*] **have** *fi*: $f(\!|i|\!) : b(\!|i|\!) \mapsto_{\mathfrak{A}\ i} c(\!|i|\!)$

by *simp*
from *prems* **show** $(f \circ_{A(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)} (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\lfloor CId \rfloor)(\lfloor b \rfloor))(\lfloor i \rfloor) = f(\lfloor i \rfloor)$
unfolding *cat-prod-Comp-app-component*[*OF that CId-b prems*]
unfolding *cat-prod-CId-app*[*OF b*]
by (*auto intro*: $\mathfrak{A}$*i.cat-CId-right-left*[*OF fi*])
qed

qed (*auto simp*: *cat-cs-intros cat-cs-simps intro*: *cat-cs-intros*)

qed

8.4 Further local assumptions for product categories

8.4.1 Definition and elementary properties

locale *pcategory* = *pcategory-base* α I $\mathfrak{A}$ **for** α I $\mathfrak{A}$ +
assumes *pcat-Obj-vsubset-Vset*: $J \subseteq_\circ I \Longrightarrow (\prod_C i \in_\circ J.\ \mathfrak{A}\ i)(\lfloor Obj \rfloor) \subseteq_\circ Vset\ \alpha$
and *pcat-Hom-vifunion-in-Vset*:
$[\![$
$J \subseteq_\circ I$;
$A \subseteq_\circ (\prod_C i \in_\circ J.\ \mathfrak{A}\ i)(\lfloor Obj \rfloor)$;
$B \subseteq_\circ (\prod_C i \in_\circ J.\ \mathfrak{A}\ i)(\lfloor Obj \rfloor)$;
$A \in_\circ Vset\ \alpha$;
$B \in_\circ Vset\ \alpha$
$]\!] \Longrightarrow (\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ Hom\ (\prod_C i \in_\circ J.\ \mathfrak{A}\ i)\ a\ b) \in_\circ Vset\ \alpha$

Rules.

lemma (**in** *pcategory*) *pcategory-axioms'*[*cat-prod-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $I' = I$
shows *pcategory* α' I' $\mathfrak{A}$
unfolding *assms* **by** (*rule pcategory-axioms*)

mk-ide rf *pcategory-def*[*unfolded pcategory-axioms-def*]
|*intro pcategoryI*|
|*dest pcategoryD*[*dest*]|
|*elim pcategoryE*[*elim*]|

lemmas [*cat-prod-cs-intros*] = *pcategoryD*(*1*)

lemma *pcategory-psemicategoryI*:
assumes *psemicategory* α I $(\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i))$
and $\bigwedge i.\ i \in_\circ I \Longrightarrow category\ \alpha\ (\mathfrak{A}\ i)$
shows *pcategory* α I $\mathfrak{A}$
proof–
interpret *psemicategory* α I ‹$\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i)$› **by** (*rule assms*(*1*))
note [*unfolded slicing-simps slicing-commute*, *cat-cs-intros*] =
psmc-Obj-vsubset-Vset
psmc-Hom-vifunion-in-Vset
show *?thesis*
by (*intro pcategoryI pcategory-base-psemicategory-baseI*)
(*auto simp*: *assms*(*2*) *smc-prod-cs-intros intro*!: *cat-cs-intros*)
qed

Product category is a product semicategory.

context *pcategory*
begin

lemma *pcat-psemicategory*: *psemicategory* α I $(\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i))$

proof(*intro psemicategoryI*, *unfold slicing-simps slicing-commute*)
show *psemicategory-base* α I $(\lambda i.$ *cat-smc* $(\mathfrak{A}\ i))$
by (*rule pcat-psemicategory-base*)
qed (*auto intro*!: *pcat-Obj-vsubset-Vset pcat-Hom-vifunion-in-Vset*)

interpretation *psmc*: *psemicategory* α I ‹$\lambda i.$ *cat-smc* $(\mathfrak{A}\ i)$›
by (*rule pcat-psemicategory*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
pcat-Obj-vsubset-Vset' = *psmc.psmc-Obj-vsubset-Vset'*
and *pcat-Hom-vifunion-in-Vset'* = *psmc.psmc-Hom-vifunion-in-Vset'*
and *pcat-cat-prod-vunion-is-arr* = *psmc.psmc-smc-prod-vunion-is-arr*
and *pcat-cat-prod-vdiff-vunion-is-arr* = *psmc.psmc-smc-prod-vdiff-vunion-is-arr*

lemmas-with [*unfolded slicing-simps slicing-commute*]:
pcat-cat-prod-vunion-Comp = *psmc.psmc-smc-prod-vunion-Comp*
and *pcat-cat-prod-vdiff-vunion-Comp* = *psmc.psmc-smc-prod-vdiff-vunion-Comp*

end

Elementary properties.

lemma (**in** *pcategory*) *pcat-vsubset-index-pcategory*:
assumes $J \subseteq_\circ I$
shows *pcategory* α J $\mathfrak{A}$
proof(*intro pcategoryI pcategory-psemicategoryI*)
show *cat-prod* J' $\mathfrak{A}(\!|Obj|\!) \subseteq_\circ$ *Vset* α **if** ‹$J' \subseteq_\circ J$› **for** J'
proof–
from *that assms* **have** $J' \subseteq_\circ I$ **by** *simp*
then show *cat-prod* J' $\mathfrak{A}(\!|Obj|\!) \subseteq_\circ$ *Vset* α **by** (*rule pcat-Obj-vsubset-Vset*)
qed
fix A B J' **assume** *prems*:
$J' \subseteq_\circ J$
$A \subseteq_\circ (\prod_C i \in_\circ J'.\ \mathfrak{A}\ i)(\!|Obj|\!)$
$B \subseteq_\circ (\prod_C i \in_\circ J'.\ \mathfrak{A}\ i)(\!|Obj|\!)$
$A \in_\circ$ *Vset* α
$B \in_\circ$ *Vset* α
show $(\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.$ *Hom* $(\prod_C i \in_\circ J'.\ \mathfrak{A}\ i)\ a\ b) \in_\circ$ *Vset* α
proof–
from *prems*(*1*) *assms* **have** $J' \subseteq_\circ I$ **by** *simp*
from *pcat-Hom-vifunion-in-Vset*[*OF this prems*(*2–5*)] **show** *?thesis*.
qed

qed (*rule pcat-vsubset-index-pcategory-base*[*OF assms*])

8.4.2 A product α-category is an α-category

lemma (**in** *pcategory*) *pcat-category-cat-prod*: *category* α $(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$
proof–
interpret *tiny-category* ‹$\alpha + \omega$› ‹$\prod_C i \in_\circ I.\ \mathfrak{A}\ i$›
by (*intro pcat-tiny-category-cat-prod*)
(*auto simp*: $\mathcal{Z}$-α-$\alpha\omega$ $\mathcal{Z}$.*intro* $\mathcal{Z}$-*Limit-*$\alpha\omega$ $\mathcal{Z}$-ω-$\alpha\omega$)
show *?thesis*
by (*rule category-if-category*)
(
auto
intro!: *pcat-Hom-vifunion-in-Vset pcat-Obj-vsubset-Vset*
intro: *cat-cs-intros*
)

qed

8.5 Local assumptions for a finite product category

8.5.1 Definition and elementary properties

locale *finite-pcategory* = *pcategory-base* α I $\mathfrak{A}$ **for** α I $\mathfrak{A}$ +
assumes *fin-pcat-index-vfinite*: *vfinite* I

Rules.

lemma (**in** *finite-pcategory*) *finite-pcategory-axioms*[*cat-prod-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $I' = I$
shows *finite-pcategory* α' I' $\mathfrak{A}$
unfolding *assms* **by** (*rule finite-pcategory-axioms*)

mk-ide rf *finite-pcategory-def*[*unfolded finite-pcategory-axioms-def*]
|*intro finite-pcategoryI*|
|*dest finite-pcategoryD*[*dest*]|
|*elim finite-pcategoryE*[*elim*]|

lemmas [*cat-prod-cs-intros*] = *finite-pcategoryD*(*1*)

lemma *finite-pcategory-finite-psemicategoryI*:
assumes *finite-psemicategory* α I (λi. *cat-smc* ($\mathfrak{A}$ i))
and $\bigwedge i$. $i \in_\circ I \Longrightarrow$ *category* α ($\mathfrak{A}$ i)
shows *finite-pcategory* α I $\mathfrak{A}$
proof–
interpret *finite-psemicategory* α I ‹λi. *cat-smc* ($\mathfrak{A}$ i)› **by** (*rule assms*(*1*))
show *?thesis*
by
(
intro
assms
finite-pcategoryI
pcategory-base-psemicategory-baseI
finite-psemicategoryD(*1*)[*OF assms*(*1*)]
fin-psmc-index-vfinite
)
qed

8.5.2 Local assumptions for a finite product semicategory and local assumptions for an arbitrary product semicategory

sublocale *finite-pcategory* ⊆ *pcategory* α I $\mathfrak{A}$
proof–
interpret *finite-psemicategory* α I ‹λi. *cat-smc* ($\mathfrak{A}$ i)›
proof(*intro finite-psemicategoryI psemicategory-baseI*)
fix i **assume** $i \in_\circ I$
then interpret $\mathfrak{A}i$: *category* α ‹$\mathfrak{A}$ i› **by** (*simp add*: *pcat-categories*)
show *semicategory* α (*cat-smc* ($\mathfrak{A}$ i)) **by** (*simp add*: $\mathfrak{A}i$.*cat-semicategory*)
qed (*auto intro*!: *cat-cs-intros fin-pcat-index-vfinite*)
show *pcategory* α I $\mathfrak{A}$
by (*intro pcategory-psemicategoryI*)
(*simp-all add*: *pcat-categories psemicategory-axioms*)
qed

8.6 Binary union and complement

lemma (**in** *pcategory*) *pcat-cat-prod-vunion-CId*:

assumes *vdisjnt J K*
and $J \subseteq_\circ I$
and $K \subseteq_\circ I$
and $a \in_\circ (\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\lfloor Obj \rfloor)$
and $b \in_\circ (\prod_C j \in_\circ K.\ \mathfrak{A}\ j)(\lfloor Obj \rfloor)$
shows
$(\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\lfloor CId \rfloor)(\lfloor a \rfloor) \cup_\circ (\prod_C j \in_\circ K.\ \mathfrak{A}\ j)(\lfloor CId \rfloor)(\lfloor b \rfloor) =$
$(\prod_C i \in_\circ J \cup_\circ K.\ \mathfrak{A}\ i)(\lfloor CId \rfloor)(\lfloor a \cup_\circ b \rfloor)$
proof–

interpret $J\mathfrak{A}$: *pcategory* α J $\mathfrak{A}$
using *assms*(*2*) **by** (*simp add*: *pcat-vsubset-index-pcategory*)
interpret $K\mathfrak{A}$: *pcategory* α K $\mathfrak{A}$
using *assms*(*3*) **by** (*simp add*: *pcat-vsubset-index-pcategory*)
interpret $JK\mathfrak{A}$: *pcategory* α ‹$J \cup_\circ K$› $\mathfrak{A}$
using *assms*(*2*,*3*) **by** (*simp add*: *pcat-vsubset-index-pcategory*)

interpret $J\mathfrak{A}'$: *category* α ‹*cat-prod J* $\mathfrak{A}$›
by (*rule J*$\mathfrak{A}$*.pcat-category-cat-prod*)
interpret $K\mathfrak{A}'$: *category* α ‹*cat-prod K* $\mathfrak{A}$›
by (*rule K*$\mathfrak{A}$*.pcat-category-cat-prod*)
interpret $JK\mathfrak{A}'$: *category* α ‹*cat-prod* ($J \cup_\circ K$) $\mathfrak{A}$›
by (*rule JK*$\mathfrak{A}$*.pcat-category-cat-prod*)

from *assms*(*4*) **have** *CId-a*: *cat-prod J* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \rfloor) : a \mapsto_{(\prod_C j \in_\circ J.\ \mathfrak{A}\ j)} a$
by (*auto intro*: *cat-cs-intros*)
from *assms*(*5*) **have** *CId-b*: *cat-prod K* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor b \rfloor) : b \mapsto_{(\prod_C k \in_\circ K.\ \mathfrak{A}\ k)} b$
by (*auto intro*: *cat-cs-intros*)
have *CId-a-CId-b*: *cat-prod J* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \rfloor) \cup_\circ$ *cat-prod K* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor b \rfloor) :$
$a \cup_\circ b \mapsto_{\textit{cat-prod}\ (J \cup_\circ K)\ \mathfrak{A}} a \cup_\circ b$
by (*rule pcat-cat-prod-vunion-is-arr*[*OF assms*(*1–3*) *CId-a CId-b*])
from *CId-a* **have** *a*: $a \in_\circ$ *cat-prod J* $\mathfrak{A}(\lfloor Obj \rfloor)$ **by** (*auto intro*: *cat-cs-intros*)
from *CId-b* **have** *b*: $b \in_\circ$ *cat-prod K* $\mathfrak{A}(\lfloor Obj \rfloor)$ **by** (*auto intro*: *cat-cs-intros*)
from *CId-a-CId-b* **have** *ab*: $a \cup_\circ b \in_\circ$ *cat-prod* ($J \cup_\circ K$) $\mathfrak{A}(\lfloor Obj \rfloor)$
by (*auto intro*: *cat-cs-intros*)

note *CId-aD* = *J*$\mathfrak{A}$*.cat-prod-is-arrD*[*OF CId-a*]
and *CId-bD* = *K*$\mathfrak{A}$*.cat-prod-is-arrD*[*OF CId-b*]

show *?thesis*
proof(*rule cat-prod-Arr-cong*[*of* - ‹$J \cup_\circ K$› $\mathfrak{A}$])
from *CId-a-CId-b* **show**
cat-prod J $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \rfloor) \cup_\circ$ *cat-prod K* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor b \rfloor) \in_\circ$ *cat-prod* ($J \cup_\circ K$) $\mathfrak{A}(\lfloor Arr \rfloor)$
by *auto*
from *ab* **show** *cat-prod* ($J \cup_\circ K$) $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \cup_\circ b \rfloor) \in_\circ$ *cat-prod* ($J \cup_\circ K$) $\mathfrak{A}(\lfloor Arr \rfloor)$
by (*auto intro*: *JK*$\mathfrak{A}'$*.cat-is-arrD*(*1*) *cat-cs-intros*)
fix *i* **assume** $i \in_\circ J \cup_\circ K$
then consider (*iJ*) ‹$i \in_\circ J$› | (*iK*) ‹$i \in_\circ K$› **by** *auto*
then show (*cat-prod J* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \rfloor) \cup_\circ$ *cat-prod K* $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor b \rfloor))(\lfloor i \rfloor) =$
cat-prod ($J \cup_\circ K$) $\mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \cup_\circ b \rfloor)(\lfloor i \rfloor)$
by *cases*
(
auto simp:
assms(*1*)
CId-aD(*1–4*)
CId-bD(*1–4*)
cat-prod-CId-app[*OF ab*]

cat-prod-CId-app[*OF a*]
cat-prod-CId-app[*OF b*]
)
qed

qed

lemma (**in** *pcategory*) *pcat-cat-prod-vdiff-vunion-CId*:
assumes $J \subseteq_\circ I$
and $a \in_\circ (\prod_C j \in_\circ I -_\circ J.\ \mathfrak{A}\ j)(\lfloor Obj \rfloor)$
and $b \in_\circ (\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\lfloor Obj \rfloor)$
shows
$(\prod_C j \in_\circ I -_\circ J.\ \mathfrak{A}\ j)(\lfloor CId \rfloor)(\lfloor a \rfloor) \cup_\circ (\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\lfloor CId \rfloor)(\lfloor b \rfloor) =$
$(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\lfloor CId \rfloor)(\lfloor a \cup_\circ b \rfloor)$
by
(
vdiff-of-vunion'
rule: *pcat-cat-prod-vunion-CId assms*: *assms*(*2–3*) *subset*: *assms*(*1*)
)

8.7 Projection

8.7.1 Definition and elementary properties

See Chapter II-3 in [7].

definition *cf-proj* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow V$ ($\langle \pi_C \rangle$)
where $\pi_C\ I\ \mathfrak{A}\ i =$
[
$(\lambda a \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\lfloor Obj \rfloor)).\ a(\lfloor i \rfloor))$,
$(\lambda f \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\lfloor Arr \rfloor)).\ f(\lfloor i \rfloor))$,
$(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$,
$\mathfrak{A}\ i$
]$_\circ$

Components.

lemma *cf-proj-components*:
shows $\pi_C\ I\ \mathfrak{A}\ i(\lfloor ObjMap \rfloor) = (\lambda a \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\lfloor Obj \rfloor)).\ a(\lfloor i \rfloor))$
and $\pi_C\ I\ \mathfrak{A}\ i(\lfloor ArrMap \rfloor) = (\lambda f \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\lfloor Arr \rfloor)).\ f(\lfloor i \rfloor))$
and $\pi_C\ I\ \mathfrak{A}\ i(\lfloor HomDom \rfloor) = (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$
and $\pi_C\ I\ \mathfrak{A}\ i(\lfloor HomCod \rfloor) = \mathfrak{A}\ i$
unfolding *cf-proj-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing

lemma *cf-smcf-cf-proj*[*slicing-commute*]:
$\pi_{SMC}\ I\ (\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i))\ i = cf\text{-}smcf\ (\pi_C\ I\ \mathfrak{A}\ i)$
unfolding
cat-smc-def
cf-smcf-def
smcf-proj-def
cf-proj-def
cat-prod-def
smc-prod-def
dg-prod-def
dg-field-simps
dghm-field-simps
by (*simp add*: *nat-omega-simps*)

context *pcategory*
begin

interpretation *psmc*: *psemicategory* α I ‹$\lambda i.$ *cat-smc* ($\mathfrak{A}$ i)›
  by (*rule pcat-psemicategory*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
  pcat-cf-proj-is-semifunctor = *psmc.psmc-smcf-proj-is-semifunctor*

end

8.7.2 Projection functor is a functor

lemma (**in** *pcategory*) *pcat-cf-proj-is-functor*:
  assumes $i \in_\circ I$
  shows π_C I $\mathfrak{A}$ i : $(\prod_C i \in_\circ I.\ \mathfrak{A}\ i) \mapsto\mapsto_{C\alpha} \mathfrak{A}\ i$
proof(*intro is-functorI*)
  interpret $\mathfrak{A}$: *category* α ‹$(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$›
    by (*simp add*: *pcat-category-cat-prod*)
  show *vfsequence* (π_C I $\mathfrak{A}$ i) **unfolding** *cf-proj-def* **by** *simp*
  show *category* α $(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$ **by** (*simp add*: $\mathfrak{A}$.*category-axioms*)
  show *vcard* (π_C I $\mathfrak{A}$ i) = $\mathfrak{4}_{\mathbb{N}}$
    unfolding *cf-proj-def* **by** (*simp add*: *nat-omega-simps*)
  show π_C I $\mathfrak{A}$ $i(\!|ArrMap|\!)(\!|(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|c|\!)|\!) = \mathfrak{A}\ i(\!|CId|\!)(\!|\pi_C\ I\ \mathfrak{A}\ i(\!|ObjMap|\!)(\!|c|\!)|\!)$
    if $c \in_\circ (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$ **for** c
  proof–
    interpret $\mathfrak{A}i$: *category* α ‹$\mathfrak{A}\ i$›
      by (*auto intro*: *assms cat-prod-cs-intros*)
    from *that* **have** $(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|c|\!) : c \mapsto_{(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)} c$
      by (*simp add*: $\mathfrak{A}$.*cat-CId-is-arr*)
    then have $(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|c|\!) \in_\circ (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
      by (*auto intro*: *cat-cs-intros*)
    with *assms* **have**
      π_C I $\mathfrak{A}$ $i(\!|ArrMap|\!)(\!|(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|c|\!)|\!) = (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|c|\!)(\!|i|\!)$
      unfolding *cf-proj-components cat-prod-components* **by** *simp*
    also from *assms* **have** ... = $\mathfrak{A}\ i(\!|CId|\!)(\!|c(\!|i|\!)|\!)$
      unfolding *cat-prod-CId-app*[*OF that*] **by** *simp*
    also from *that* **have** ... = $\mathfrak{A}\ i(\!|CId|\!)(\!|\pi_C\ I\ \mathfrak{A}\ i(\!|ObjMap|\!)(\!|c|\!)|\!)$
      unfolding *cf-proj-components cat-prod-components* **by** *simp*
    finally show
      π_C I $\mathfrak{A}$ $i(\!|ArrMap|\!)(\!|(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|CId|\!)(\!|c|\!)|\!) = \mathfrak{A}\ i(\!|CId|\!)(\!|\pi_C\ I\ \mathfrak{A}\ i(\!|ObjMap|\!)(\!|c|\!)|\!)$
      by *simp*
  qed
qed
  (
    auto simp:
      assms cf-proj-components pcat-cf-proj-is-semifunctor cat-prod-cs-intros
)

lemma (**in** *pcategory*) *pcat-cf-proj-is-functor'*:
  assumes $i \in_\circ I$ **and** $\mathfrak{C} = (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$ **and** $\mathfrak{D} = \mathfrak{A}\ i$
  shows π_C I $\mathfrak{A}$ i : $\mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
  using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule pcat-cf-proj-is-functor*)

lemmas [*cat-cs-intros*] = *pcategory.pcat-cf-proj-is-functor'*

8.8 Category product universal property functor

8.8.1 Definition and elementary properties

The functor that is presented in this section is used in the proof of the universal property of the product category later in this work.

definition *cf-up* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  where *cf-up* I $\mathfrak{A}$ $\mathfrak{C}$ φ =
    [
      $(\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ObjMap|\!)(\!|a|\!)))$,
      $(\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ArrMap|\!)(\!|f|\!)))$,
      $\mathfrak{C}$,
      $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)$
    ]$_\circ$

Components.

lemma *cf-up-components*:
  shows *cf-up* I $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|ObjMap|\!)$ = $(\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ObjMap|\!)(\!|a|\!)))$
    and *cf-up* I $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|ArrMap|\!)$ = $(\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ArrMap|\!)(\!|f|\!)))$
    and *cf-up* I $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|HomDom|\!)$ = $\mathfrak{C}$
    and *cf-up* I $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|HomCod|\!)$ = $(\prod_{C} i \in_\circ I.\ \mathfrak{A}\ i)$
  unfolding *cf-up-def dghm-field-simps* **by** (*simp add*: *nat-omega-simps*)

Slicing.

lemma *smcf-dghm-cf-up*[*slicing-commute*]:
  smcf-up I $(\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i))$ $(cat\text{-}smc\ \mathfrak{C})$ $(\lambda i.\ cf\text{-}smcf\ (\varphi\ i))$ =
    cf-smcf (*cf-up* I $\mathfrak{A}$ $\mathfrak{C}$ φ)
  unfolding
    cat-smc-def
    cf-smcf-def
    cf-up-def
    smcf-up-def
    cat-prod-def
    smc-prod-def
    dg-prod-def
    dg-field-simps
    dghm-field-simps
  by (*simp add*: *nat-omega-simps*)

context
  fixes $\mathfrak{A}$ φ :: $V \Rightarrow V$
    and $\mathfrak{C}$ :: V
begin

lemmas-with
  [
    where $\mathfrak{A}$=‹$\lambda i.\ cat\text{-}smc\ (\mathfrak{A}\ i)$› **and** φ=‹$\lambda i.\ cf\text{-}smcf\ (\varphi\ i)$› **and** $\mathfrak{C}$ = ‹*cat-smc* $\mathfrak{C}$›,
    unfolded slicing-simps slicing-commute
  ]:
  cf-up-ObjMap-vdomain[*simp*] = *smcf-up-ObjMap-vdomain*
  and *cf-up-ObjMap-app* = *smcf-up-ObjMap-app*
  and *cf-up-ObjMap-app-vdomain*[*simp*] = *smcf-up-ObjMap-app-vdomain*
  and *cf-up-ObjMap-app-component* = *smcf-up-ObjMap-app-component*
  and *cf-up-ArrMap-vdomain*[*simp*] = *smcf-up-ArrMap-vdomain*
  and *cf-up-ArrMap-app* = *smcf-up-ArrMap-app*
  and *cf-up-ArrMap-app-vdomain*[*simp*] = *smcf-up-ArrMap-app-vdomain*
  and *cf-up-ArrMap-app-component* = *smcf-up-ArrMap-app-component*

lemma *cf-up-ObjMap-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{cf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ObjMap}|\!)) \subseteq_\circ (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{Obj}|\!)$
proof
(
rule smcf-up-ObjMap-vrange[
where $\mathfrak{A}$=$\langle\lambda i.\ \mathit{cat\text{-}smc}\ (\mathfrak{A}\ i)\rangle$
and φ=$\langle\lambda i.\ \mathit{cf\text{-}smcf}\ (\varphi\ i)\rangle$
and $\mathfrak{C}$=$\langle\mathit{cat\text{-}smc}\ \mathfrak{C}\rangle$,
unfolded slicing-simps slicing-commute
]
)
fix i **assume** $i \in_\circ I$
then interpret *is-functor* $\alpha\ \mathfrak{C}\ \langle\mathfrak{A}\ i\rangle\ \langle\varphi\ i\rangle$ **by** (*rule assms*)
show *cf-smcf* $(\varphi\ i) : \mathit{cat\text{-}smc}\ \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathit{cat\text{-}smc}\ (\mathfrak{A}\ i)$
by (*rule cf-is-semifunctor*)
qed

lemma *cf-up-ObjMap-app-vrange*:
assumes $a \in_\circ \mathfrak{C}(\!|\mathit{Obj}|\!)$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{cf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ObjMap}|\!)(\!|a|\!)) \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|\mathit{Obj}|\!))$
proof
(
rule smcf-up-ObjMap-app-vrange[
where $\mathfrak{A}$=$\langle\lambda i.\ \mathit{cat\text{-}smc}\ (\mathfrak{A}\ i)\rangle$
and φ=$\langle\lambda i.\ \mathit{cf\text{-}smcf}\ (\varphi\ i)\rangle$
and $\mathfrak{C}$=$\langle\mathit{cat\text{-}smc}\ \mathfrak{C}\rangle$,
unfolded slicing-simps slicing-commute
]
)
show $a \in_\circ \mathfrak{C}(\!|\mathit{Obj}|\!)$ **by** (*rule assms*)
fix i **assume** $i \in_\circ I$
then interpret *is-functor* $\alpha\ \mathfrak{C}\ \langle\mathfrak{A}\ i\rangle\ \langle\varphi\ i\rangle$ **by** (*rule assms*(*2*))
show *cf-smcf* $(\varphi\ i) : \mathit{cat\text{-}smc}\ \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathit{cat\text{-}smc}\ (\mathfrak{A}\ i)$
by (*rule cf-is-semifunctor*)
qed

lemma *cf-up-ArrMap-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{cf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)) \subseteq_\circ (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{Arr}|\!)$
proof
(
rule smcf-up-ArrMap-vrange[
where $\mathfrak{A}$=$\langle\lambda i.\ \mathit{cat\text{-}smc}\ (\mathfrak{A}\ i)\rangle$
and φ=$\langle\lambda i.\ \mathit{cf\text{-}smcf}\ (\varphi\ i)\rangle$
and $\mathfrak{C}$=$\langle\mathit{cat\text{-}smc}\ \mathfrak{C}\rangle$,
unfolded slicing-simps slicing-commute
]
)
fix i **assume** $i \in_\circ I$
then interpret *is-functor* $\alpha\ \mathfrak{C}\ \langle\mathfrak{A}\ i\rangle\ \langle\varphi\ i\rangle$ **by** (*rule assms*)
show *cf-smcf* $(\varphi\ i) : \mathit{cat\text{-}smc}\ \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathit{cat\text{-}smc}\ (\mathfrak{A}\ i)$
by (*rule cf-is-semifunctor*)
qed

lemma *cf-up-ArrMap-app-vrange*:
assumes $a \in_\circ \mathfrak{C}(\!|\mathit{Arr}|\!)$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{cf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!)) \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|\mathit{Arr}|\!))$

```
proof
  (
    rule smcf-up-ArrMap-app-vrange
      [
        where 𝔄=‹λi. cat-smc (𝔄 i)›
          and φ=‹λi. cf-smcf (φ i)›
          and 𝔈=‹cat-smc 𝔈›,
        unfolded slicing-simps slicing-commute
      ]
  )
  fix i assume i ∈₀ I
  then interpret is-functor α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms(2))
  show cf-smcf (φ i) : cat-smc 𝔈 ↦↦_{SMCα} cat-smc (𝔄 i)
    by (rule cf-is-semifunctor)
qed (rule assms)

end

context pcategory
begin

interpretation psmc: psemicategory α I ‹λi. cat-smc (𝔄 i)›
  by (rule pcat-psemicategory)

lemmas-with [unfolded slicing-simps slicing-commute]:
  pcat-smcf-comp-smcf-proj-smcf-up = psmc.psmc-Comp-smcf-proj-smcf-up
  and pcat-smcf-up-eq-smcf-proj = psmc.psmc-smcf-up-eq-smcf-proj

end
```

8.8.2 Category product universal property functor is a functor

```
lemma (in pcategory) pcat-cf-up-is-functor:
  assumes category α 𝔈 and ⋀i. i ∈₀ I ⟹ φ i : 𝔈 ↦↦_{Cα} 𝔄 i
  shows cf-up I 𝔄 𝔈 φ : 𝔈 ↦↦_{Cα} (∏_C i∈₀I. 𝔄 i)
proof–
  interpret 𝔈: category α 𝔈 by (simp add: assms(1))
  interpret 𝔄: category α ‹(∏_C i∈₀I. 𝔄 i)› by (rule pcat-category-cat-prod)
  show ?thesis
  proof(intro is-functorI)
    show vfsequence (cf-up I 𝔄 𝔈 φ) unfolding cf-up-def by simp
    show vcard (cf-up I 𝔄 𝔈 φ) = ℶ_ℕ
      unfolding cf-up-def by (simp add: nat-omega-simps)
    show cf-smcf (cf-up I 𝔄 𝔈 φ) : cat-smc 𝔈 ↦↦_{SMCα} cat-smc (∏_C i∈₀I. 𝔄 i)
      unfolding slicing-commute[symmetric]
      by (rule psemicategory.psmc-smcf-up-is-semifunctor)
        (
          auto simp:
            assms(2)
            pcat-psemicategory
            is-functor.cf-is-semifunctor
            slicing-intros
        )
    show cf-up I 𝔄 𝔈 φ(|ArrMap|)(|𝔈(|CId|)(|c|)|) =
      (∏_C i∈₀I. 𝔄 i)(|CId|)(|cf-up I 𝔄 𝔈 φ(|ObjMap|)(|c|)|)
      if c ∈₀ 𝔈(|Obj|) for c
    proof(rule cat-prod-Arr-cong)
      from that is-arrD(1) have CId-c: 𝔈(|CId|)(|c|) ∈₀ 𝔈(|Arr|)
```

```
    by (auto intro: cat-cs-intros)
  from CId-c cf-up-ArrMap-vrange[OF assms(2), simplified]
  show cf-up I 𝔄 ℭ φ(|ArrMap|)(|ℭ(|CId|)(|c|)|) ∈∘ (∏C i∈∘I. 𝔄 i)(|Arr|)
    unfolding cf-up-components by force
  have cf-up-φ-c: cf-up I 𝔄 ℭ φ(|ObjMap|)(|c|) ∈∘ (∏C i∈∘I. 𝔄 i)(|Obj|)
    unfolding cat-prod-components
  proof(intro vproductI ballI)
    fix i assume prems: i ∈∘ I
    interpret φ: is-functor α ℭ ‹𝔄 i› ‹φ i› by (simp add: prems assms(2))
    from that show  cf-up I 𝔄 ℭ φ(|ObjMap|)(|c|)(|i|) ∈∘ 𝔄 i(|Obj|)
      unfolding cf-up-ObjMap-app-component[OF that prems]
      by (auto intro: cat-cs-intros)
  qed (simp-all add: cf-up-ObjMap-app that cf-up-ObjMap-app[OF that])
  from 𝔄.cat-CId-is-arr[OF this] show
    (∏C i∈∘I. 𝔄 i)(|CId|)(|cf-up I 𝔄 ℭ φ(|ObjMap|)(|c|)|) ∈∘ (∏C i∈∘I. 𝔄 i)(|Arr|)
    by auto
  fix i assume prems: i ∈∘ I
  interpret φ: is-functor α ℭ ‹𝔄 i› ‹φ i› by (simp add: prems assms(2))
  from cf-up-φ-c prems show
    cf-up I 𝔄 ℭ φ(|ArrMap|)(|ℭ(|CId|)(|c|)|)(|i|) =
      (∏C i∈∘I. 𝔄 i)(|CId|)(|cf-up I 𝔄 ℭ φ(|ObjMap|)(|c|)|)(|i|)
    unfolding cf-up-ArrMap-app-component[OF CId-c prems] cat-prod-components
    by
      (
        simp add:
          that cf-up-ObjMap-app-component[OF that prems] φ.cf-ObjMap-CId
      )
  qed
qed (auto simp: cf-up-components cat-cs-intros)
qed
```

8.8.3 Further properties

```
lemma (in pcategory) pcat-Comp-cf-proj-cf-up:
  assumes category α ℭ
    and ⋀i. i ∈∘ I ⟹ φ i : ℭ ↦↦Cα 𝔄 i
    and i ∈∘ I
  shows φ i = πC I 𝔄 i ∘CF (cf-up I 𝔄 ℭ φ)
proof–
  interpret φ: is-functor α ℭ ‹𝔄 i› ‹φ i› by (rule assms(2)[OF assms(3)])
  interpret π: is-functor α ‹(∏C i∈∘I. 𝔄 i)› ‹𝔄 i› ‹πC I 𝔄 i›
    by (simp add: assms(3) pcat-cf-proj-is-functor)
  interpret up: is-functor α ℭ ‹(∏C i∈∘I. 𝔄 i)› ‹cf-up I 𝔄 ℭ φ›
    by (simp add: assms(2) φ.HomDom.category-axioms pcat-cf-up-is-functor)
  show ?thesis
  proof(rule cf-smcf-eqI)
    show πC I 𝔄 i ∘CF cf-up I 𝔄 ℭ φ : ℭ ↦↦Cα 𝔄 i
      by (auto intro: cat-cs-intros)
    from assms show cf-smcf (φ i) = cf-smcf (πC I 𝔄 i ∘CF cf-up I 𝔄 ℭ φ)
      unfolding slicing-simps slicing-commute[symmetric]
      by
        (
          intro pcat-smcf-comp-smcf-proj-smcf-up[
            where φ=‹λi. cf-smcf (φ i)›, unfolded slicing-commute[symmetric]
            ]
        )
        (auto simp: is-functor.cf-is-semifunctor)
  qed (auto intro: cat-cs-intros)
```

qed

lemma (**in** *pcategory*) *pcat-cf-up-eq-cf-proj*:
assumes $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$
and $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i = \pi_C\ I\ \mathfrak{A}\ i \circ_{CF} \mathfrak{F}$
shows *cf-up* $I\ \mathfrak{A}\ \mathfrak{C}\ \varphi = \mathfrak{F}$
proof(*rule cf-smcf-eqI*)
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}\ \langle(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)\rangle\ \mathfrak{F}$ **by** (*rule assms*(*1*))
show *cf-up* $I\ \mathfrak{A}\ \mathfrak{C}\ \varphi : \mathfrak{C} \mapsto\mapsto_{C\alpha} (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$
proof(*rule pcat-cf-up-is-functor*)
fix i **assume** *prems*: $i \in_\circ I$
then interpret π: *is-functor* $\alpha\ \langle(\prod_C i \in_\circ I.\ \mathfrak{A}\ i)\rangle\ \langle\mathfrak{A}\ i\rangle\ \langle\pi_C\ I\ \mathfrak{A}\ i\rangle$
by (*rule pcat-cf-proj-is-functor*)
show $\varphi\ i : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}\ i$
unfolding *assms*(*2*)[*OF prems*] **by** (*auto intro*: *cat-cs-intros*)
qed (*auto intro*: *cat-cs-intros*)
show $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} (\prod_C i \in_\circ I.\ \mathfrak{A}\ i)$ **by** (*rule assms*(*1*))
from *assms* **show** *cf-smcf* (*cf-up* $I\ \mathfrak{A}\ \mathfrak{C}\ \varphi$) = *cf-smcf* $\mathfrak{F}$
unfolding *slicing-commute*[*symmetric*]
by (*intro pcat-smcf-up-eq-smcf-proj*) (*auto simp*: *slicing-commute*)
qed *simp-all*

8.9 Prodfunctor with respect to a fixed argument

A prodfunctor is a functor whose domain is a product category. It is a generalization of the concept of the bifunctor, as presented in Chapter II-3 in [7].

definition *prodfunctor-proj* :: $V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *prodfunctor-proj* $\mathfrak{S}\ I\ \mathfrak{A}\ \mathfrak{D}\ J\ c =$
[
$(\lambda b \in_\circ (\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_\circ c|\!))$,
$(\lambda f \in_\circ (\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ (\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\!|CId|\!)(\!|c|\!)|\!))$,
$(\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)$,
$\mathfrak{D}$
]$_\circ$

syntax -*PPRODFUNCTOR-PROJ* :: $V \Rightarrow$ *pttrn* $\Rightarrow V \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow V \Rightarrow V$
$(\langle(\text{-}(3\prod_C \text{-} \in_\circ \text{-} -_\circ \text{-}./\text{-}),\text{-}/'(/-,\text{-}/')\rangle\ [51, 51, 51, 51, 51, 51, 51]\ 51)$
syntax-consts -*PPRODFUNCTOR-PROJ* $\rightleftharpoons$ *prodfunctor-proj*
translations $\mathfrak{S}_{\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A},\mathfrak{D}}(-,c) \rightleftharpoons$
CONST prodfunctor-proj $\mathfrak{S}\ I\ (\lambda i.\ \mathfrak{A})\ \mathfrak{D}\ J\ c$

Components.

lemma *prodfunctor-proj-components*:
shows $(\mathfrak{S}_{\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i,\mathfrak{D}}(-,c))(\!|ObjMap|\!) =$
$(\lambda b \in_\circ (\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_\circ c|\!))$
and $(\mathfrak{S}_{\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i,\mathfrak{D}}(-,c))(\!|ArrMap|\!) =$
$(\lambda f \in_\circ (\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ (\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\!|CId|\!)(\!|c|\!)|\!))$
and $(\mathfrak{S}_{\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i,\mathfrak{D}}(-,c))(\!|HomDom|\!) = (\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)$
and $(\mathfrak{S}_{\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i,\mathfrak{D}}(-,c))(\!|HomCod|\!) = \mathfrak{D}$
unfolding *prodfunctor-proj-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

8.9.1 Object map

mk-VLambda *prodfunctor-proj-components*(*1*)
|*vsv prodfunctor-proj-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain prodfunctor-proj-ObjMap-vdomain*[*cat-cs-simps*]|

|*app prodfunctor-proj-ObjMap-app*[*cat-cs-simps*]|

8.9.2 Arrow map

mk-VLambda *prodfunctor-proj-components*(*2*)
|*vsv prodfunctor-proj-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain prodfunctor-proj-ArrMap-vdomain*[*cat-cs-simps*]|
|*app prodfunctor-proj-ArrMap-app*[*cat-cs-simps*]|

8.9.3 Prodfunctor with respect to a fixed argument is a functor

lemma (**in** *pcategory*) *pcat-prodfunctor-proj-is-functor*:
assumes $\mathfrak{S} : (\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i) \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $c \in_{\circ} (\prod_{C} j \in_{\circ} J.\ \mathfrak{A}\ j)(\!|Obj|\!)$
and $J \subseteq_{\circ} I$
shows $(\mathfrak{S}_{\prod_{C} i \in_{\circ} I -_{\circ} J.\ \mathfrak{A}\ i, \mathfrak{D}}(-,c)) : (\prod_{C} i \in_{\circ} I -_{\circ} J.\ \mathfrak{A}\ i) \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof–

interpret *is-functor* α $\langle(\prod_{C} i \in_{\circ} I.\ \mathfrak{A}\ i)\rangle$ $\mathfrak{D}$ $\mathfrak{S}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{A}$: *pcategory* α J $\mathfrak{A}$
using *assms*(*3*) **by** (*intro pcat-vsubset-index-pcategory*) *auto*
interpret *J*-$\mathfrak{A}$: *category* α $\langle\prod_{C} i \in_{\circ} J.\ \mathfrak{A}\ i\rangle$ **by** (*rule* $\mathfrak{A}$.*pcat-category-cat-prod*)
interpret *IJ*: *pcategory* α $\langle I -_{\circ} J\rangle$ $\mathfrak{A}$
using *assms*(*3*) **by** (*intro pcat-vsubset-index-pcategory*) *auto*
interpret *IJ*-$\mathfrak{A}$: *category* α $\langle\prod_{C} i \in_{\circ} I -_{\circ} J.\ \mathfrak{A}\ i\rangle$
by (*rule IJ.pcat-category-cat-prod*)

let *?IJ*$\mathfrak{A}$ = $\langle(\prod_{C} i \in_{\circ} I -_{\circ} J.\ \mathfrak{A}\ i)\rangle$

from *assms*(*2*) **have** $c \in_{\circ} (\prod_{\circ} j \in_{\circ} J.\ \mathfrak{A}\ j(\!|Obj|\!))$
unfolding *cat-prod-components* **by** *simp*
then have $(\prod_{\circ} j \in_{\circ} J.\ \mathfrak{A}\ j(\!|Obj|\!)) \neq 0$ **by** (*auto intro*!: *cat-cs-intros*)

show *?thesis*
proof(*intro is-functorI'*, *unfold prodfunctor-proj-components*)

show *vfsequence* (*prodfunctor-proj* $\mathfrak{S}$ I $\mathfrak{A}$ $\mathfrak{D}$ J c)
unfolding *prodfunctor-proj-def* **by** *simp*
show *vcard* (*prodfunctor-proj* $\mathfrak{S}$ I $\mathfrak{A}$ $\mathfrak{D}$ J c) = $\mathfrak{4}_{\mathbf{N}}$
unfolding *prodfunctor-proj-def* **by** (*simp add*: *nat-omega-simps*)

show $\mathcal{R}_{\circ}$ $(\lambda b \in_{\circ} ?IJ\mathfrak{A}(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_{\circ} c|\!)) \subseteq_{\circ} \mathfrak{D}(\!|Obj|\!)$
proof(*intro vsubsetI*)
fix x **assume** $x \in_{\circ} \mathcal{R}_{\circ}$ $(\lambda b \in_{\circ} ?IJ\mathfrak{A}(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_{\circ} c|\!))$
then obtain b **where** *x-def*: $x = \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_{\circ} c|\!)$ **and** b: $b \in_{\circ} ?IJ\mathfrak{A}(\!|Obj|\!)$
by *auto*
have $b \cup_{\circ} c \in_{\circ}$ *cat-prod* I $\mathfrak{A}(\!|Obj|\!)$
proof(*rule cat-prod-vdiff-vunion-Obj-in-Obj*)
show $b \in_{\circ} ?IJ\mathfrak{A}(\!|Obj|\!)$ **by** (*rule b*)
qed (*intro assms*(*2*,*3*))+
then show $x \in_{\circ} \mathfrak{D}(\!|Obj|\!)$ **unfolding** *x-def* **by** (*auto intro*: *cat-cs-intros*)
qed

show *is-arr*:
$(\lambda f \in_{\circ} ?IJ\mathfrak{A}(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_{\circ} cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!))(\!|f|\!)$:
$(\lambda b \in_{\circ} ?IJ\mathfrak{A}(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_{\circ} c|\!))(\!|a|\!) \mapsto_{\mathfrak{D}}$
$(\lambda b \in_{\circ} ?IJ\mathfrak{A}(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_{\circ} c|\!))(\!|b|\!)$
(**is** $\langle ?V\text{-}f$: $?V\text{-}a \mapsto_{\mathfrak{D}} ?V\text{-}b\rangle$)

if $f : a \mapsto_{?IJ\mathfrak{A}} b$ **for** $f\ a\ b$
proof–
let $?fc = \langle f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)\rangle$
have $?fc : a \cup_\circ c \mapsto_{cat\text{-}prod\ I\ \mathfrak{A}} b \cup_\circ c$
proof(*rule pcat-cat-prod-vdiff-vunion-is-arr*)
show $f : a \mapsto_{?IJ\mathfrak{A}} b$ **by** (*rule that*)
qed (*auto simp*: *assms cat-cs-intros*)
then have $\mathfrak{S}(\!|ArrMap|\!)(\!|?fc|\!) : \mathfrak{S}(\!|ObjMap|\!)(\!|a \cup_\circ c|\!) \mapsto_{\mathfrak{D}} \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_\circ c|\!)$
by (*auto intro*: *cat-cs-intros*)
moreover from *that* **have** $f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!)\ a \in_\circ ?IJ\mathfrak{A}(\!|Obj|\!)\ b \in_\circ ?IJ\mathfrak{A}(\!|Obj|\!)$
by (*auto intro*: *cat-cs-intros*)
ultimately show *?thesis* **by** *simp*
qed

show
$(\lambda f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!))(\!|g \circ_{A\ ?IJ\mathfrak{A}} f|\!) =$
$(\lambda f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!))(\!|g|\!) \circ_{A\mathfrak{D}}$
$(\lambda f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!))(\!|f|\!)$
if $g : b' \mapsto_{?IJ\mathfrak{A}} c'$ **and** $f : a' \mapsto_{?IJ\mathfrak{A}} b'$ **for** $g\ b'\ c'\ f\ a'$
proof–
from *that* **have** *gf*: $g \circ_{A\ ?IJ\mathfrak{A}} f : a' \mapsto_{?IJ\mathfrak{A}} c'$
by (*auto intro*: *cat-cs-intros*)
from *assms*(*2*) **have** *CId-c*: $cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!) : c \mapsto_{cat\text{-}prod\ J\ \mathfrak{A}} c$
by (*auto intro*: *cat-cs-intros*)
then have [*simp*]:
$cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!) \circ_{A\ cat\text{-}prod\ J\ \mathfrak{A}} cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!) =$
$cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)$
by (*auto simp*: *cat-cs-simps*)
from *assms*(*3*) *that*(*1*) *CId-c* **have** *g-CId-c*:
$g \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!) : b' \cup_\circ c \mapsto_{cat\text{-}prod\ I\ \mathfrak{A}} c' \cup_\circ c$
by (*rule pcat-cat-prod-vdiff-vunion-is-arr*)
from *assms*(*3*) *that*(*2*) *CId-c* **have** *f-CId-c*:
$f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!) : a' \cup_\circ c \mapsto_{cat\text{-}prod\ I\ \mathfrak{A}} b' \cup_\circ c$
by (*rule pcat-cat-prod-vdiff-vunion-is-arr*)
have
$\mathfrak{S}(\!|ArrMap|\!)(\!|(g \circ_{A\ ?IJ\mathfrak{A}} f) \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!) =$
$\mathfrak{S}(\!|ArrMap|\!)(\!|g \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!) \circ_{A\mathfrak{D}}$
$\mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!)$
unfolding
pcat-cat-prod-vdiff-vunion-Comp[
OF assms(*3*) *that*(*1*) *CId-c that*(*2*) *CId-c*, *simplified*
]
by (*intro cf-ArrMap-Comp*[*OF g-CId-c f-CId-c*])
moreover from *gf* **have** $g \circ_{A\ ?IJ\mathfrak{A}} f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!)$ **by** *auto*
moreover from *that* **have** $g \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!)\ f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!)$ **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

show
$(\lambda f \in_\circ ?IJ\mathfrak{A}(\!|Arr|\!).\ \mathfrak{S}(\!|ArrMap|\!)(\!|f \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!)|\!))(\!|?IJ\mathfrak{A}(\!|CId|\!)(\!|c'|\!)|\!) =$
$\mathfrak{D}(\!|CId|\!)(\!|(\lambda b \in_\circ ?IJ\mathfrak{A}(\!|Obj|\!).\ \mathfrak{S}(\!|ObjMap|\!)(\!|b \cup_\circ c|\!))(\!|c'|\!)|\!)$
if $c' \in_\circ ?IJ\mathfrak{A}(\!|Obj|\!)$ **for** c'
proof–
have $?IJ\mathfrak{A}(\!|CId|\!)(\!|c'|\!) \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\!|CId|\!)(\!|c|\!) = cat\text{-}prod\ I\ \mathfrak{A}(\!|CId|\!)(\!|c' \cup_\circ c|\!)$
unfolding *pcat-cat-prod-vdiff-vunion-CId*[*OF assms*(*3*) *that assms*(*2*)] **..**
moreover from *assms*(*3*) *that assms*(*2*) **have** $c' \cup_\circ c \in_\circ cat\text{-}prod\ I\ \mathfrak{A}(\!|Obj|\!)$
by (*rule cat-prod-vdiff-vunion-Obj-in-Obj*)

ultimately have $\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor ?IJ\mathfrak{A}(\lfloor CId \rfloor)(\lfloor c' \rfloor) \cup_\circ cat\text{-}prod\ J\ \mathfrak{A}(\lfloor CId \rfloor)(\lfloor c \rfloor)) =$
$\mathfrak{D}(\lfloor CId \rfloor)(\lfloor \mathfrak{S}(\lfloor ObjMap \rfloor)(\lfloor c' \cup_\circ c \rfloor) \rfloor)$
by (*auto intro*: *cat-cs-intros*)
moreover from *that* **have** *CId-c'*: $?IJ\mathfrak{A}(\lfloor CId \rfloor)(\lfloor c' \rfloor) \in_\circ ?IJ\mathfrak{A}(\lfloor Arr \rfloor)$
by (*auto dest*!: *IJ-𝔄.cat-CId-is-arr*)
ultimately show *?thesis* **by** (*simp add*: *that*)
qed

qed (*auto intro*: *cat-cs-intros*)

qed

lemma (**in** *pcategory*) *pcat-prodfunctor-proj-is-functor'*:
assumes $\mathfrak{S} : (\prod_C i \in_\circ I.\ \mathfrak{A}\ i) \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $c \in_\circ (\prod_C j \in_\circ J.\ \mathfrak{A}\ j)(\lfloor Obj \rfloor)$
and $J \subseteq_\circ I$
and $\mathfrak{A}' = (\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i)$
and $\mathfrak{B}' = \mathfrak{D}$
shows $(\mathfrak{S}_{\prod_C i \in_\circ I -_\circ J.\ \mathfrak{A}\ i, \mathfrak{D}}(-, c)) : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
using *assms*(*1–3*)
unfolding *assms*(*4,5*)
by (*rule pcat-prodfunctor-proj-is-functor*)

lemmas [*cat-cs-intros*] = *pcategory.pcat-prodfunctor-proj-is-functor'*

8.10 Singleton category

8.10.1 Slicing

context
fixes $\mathfrak{C} :: V$
begin

lemmas-with [**where** $\mathfrak{C}$=‹*cat-smc* $\mathfrak{C}$›, *unfolded slicing-simps slicing-commute*]:
cat-singleton-ObjI = *smc-singleton-ObjI*
and *cat-singleton-ObjE* = *smc-singleton-ObjE*
and *cat-singleton-ArrI* = *smc-singleton-ArrI*
and *cat-singleton-ArrE* = *smc-singleton-ArrE*

end

context *category*
begin

interpretation *smc*: *semicategory* α ‹*cat-smc* $\mathfrak{C}$› **by** (*rule cat-semicategory*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
cat-finite-psemicategory-cat-singleton =
smc.smc-finite-psemicategory-smc-singleton
and *cat-singleton-is-arrI* = *smc.smc-singleton-is-arrI*
and *cat-singleton-is-arrD* = *smc.smc-singleton-is-arrD*
and *cat-singleton-is-arrE* = *smc.smc-singleton-is-arrE*

end

8.10.2 Identity

lemma *cat-singleton-CId-app*:
assumes *set* $\{\langle j,\ a\rangle\} \in_\circ (\prod_C i \in_\circ set\ \{j\}.\ \mathfrak{C})(\lfloor Obj \rfloor)$

shows $(\prod_C i \in_\circ set\ \{j\}.\ \mathfrak{C})(\!|CId|\!)(\!|set\ \{\langle j,\ a\rangle\}|\!) = set\ \{\langle j,\ \mathfrak{C}(\!|CId|\!)(\!|a|\!)\rangle\}$
using *assms* **unfolding** *cat-prod-components VLambda-vsingleton* **by** *simp*

8.10.3 Singleton category is a category

lemma (**in** *category*) *cat-finite-pcategory-cat-singleton*:
assumes $j \in_\circ Vset\ \alpha$
shows *finite-pcategory* α $(set\ \{j\})$ $(\lambda i.\ \mathfrak{C})$
by
(
auto intro:
assms
category-axioms
finite-pcategory-finite-psemicategoryI
cat-finite-psemicategory-cat-singleton
)

lemma (**in** *category*) *cat-category-cat-singleton*:
assumes $j \in_\circ Vset\ \alpha$
shows *category* α $(\prod_C i \in_\circ set\ \{j\}.\ \mathfrak{C})$
proof–
interpret *finite-pcategory* α ‹*set* $\{j\}$› ‹$\lambda i.\ \mathfrak{C}$›
using *assms* **by** (*rule cat-finite-pcategory-cat-singleton*)
show *?thesis* **by** (*rule pcat-category-cat-prod*)
qed

8.11 Singleton functor

8.11.1 Definition and elementary properties

definition *cf-singleton* :: $V \Rightarrow V \Rightarrow V$
where *cf-singleton* j $\mathfrak{C}$ =
[
$(\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).\ set\ \{\langle j,\ a\rangle\})$,
$(\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).\ set\ \{\langle j,\ f\rangle\})$,
$\mathfrak{C}$,
$(\prod_C i \in_\circ set\ \{j\}.\ \mathfrak{C})$
]$_\circ$

Components.

lemma *cf-singleton-components*:
shows *cf-singleton* j $\mathfrak{C}(\!|ObjMap|\!) = (\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).\ set\ \{\langle j,\ a\rangle\})$
and *cf-singleton* j $\mathfrak{C}(\!|ArrMap|\!) = (\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).\ set\ \{\langle j,\ f\rangle\})$
and *cf-singleton* j $\mathfrak{C}(\!|HomDom|\!) = \mathfrak{C}$
and *cf-singleton* j $\mathfrak{C}(\!|HomCod|\!) = (\prod_C i \in_\circ set\ \{j\}.\ \mathfrak{C})$
unfolding *cf-singleton-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cf-smcf-cf-singleton*[*slicing-commute*]:
smcf-singleton j (*cat-smc* $\mathfrak{C}$)= *cf-smcf* (*cf-singleton* j $\mathfrak{C}$)
unfolding *smcf-singleton-def cf-singleton-def slicing-simps slicing-commute*
by
(
simp add:
nat-omega-simps dghm-field-simps dg-field-simps cat-smc-def cf-smcf-def
)

context

fixes $\mathfrak{C} :: V$
begin

lemmas-with [**where** $\mathfrak{C}$=‹*cat-smc* $\mathfrak{C}$›, *unfolded slicing-simps slicing-commute*]:
cf-singleton-ObjMap-vsv[*cat-cs-intros*] = *smcf-singleton-ObjMap-vsv*
and *cf-singleton-ObjMap-vdomain*[*cat-cs-simps*] = *smcf-singleton-ObjMap-vdomain*
and *cf-singleton-ObjMap-vrange* = *smcf-singleton-ObjMap-vrange*
and *cf-singleton-ObjMap-app*[*cat-prod-cs-simps*] = *smcf-singleton-ObjMap-app*
and *cf-singleton-ArrMap-vsv*[*cat-cs-intros*] = *smcf-singleton-ArrMap-vsv*
and *cf-singleton-ArrMap-vdomain*[*cat-cs-simps*] = *smcf-singleton-ArrMap-vdomain*
and *cf-singleton-ArrMap-vrange* = *smcf-singleton-ArrMap-vrange*
and *cf-singleton-ArrMap-app*[*cat-prod-cs-simps*] = *smcf-singleton-ArrMap-app*

end

8.11.2 Singleton functor is an isomorphism of categories

lemma (**in** *category*) *cat-cf-singleton-is-functor*:
assumes $j \in_\circ$ *Vset* α
shows *cf-singleton* j $\mathfrak{C}$: $\mathfrak{C} \mapsto\mapsto_{C.iso\alpha}$ ($\prod_C i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
proof(*intro is-iso-functorI is-functorI*)
from *assms* **show** *smcf-singleton*: *cf-smcf* (*cf-singleton* j $\mathfrak{C}$) :
cat-smc $\mathfrak{C} \mapsto\mapsto_{SMC.iso\alpha}$ *cat-smc* ($\prod_C i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
unfolding *slicing-commute*[*symmetric*]
by (*intro semicategory.smc-smcf-singleton-is-iso-semifunctor*)
(*auto intro*: *smc-cs-intros slicing-intros*)
show *vfsequence* (*cf-singleton* j $\mathfrak{C}$) **unfolding** *cf-singleton-def* **by** *simp*
show *vcard* (*cf-singleton* j $\mathfrak{C}$) = $\mathfrak{4}_\mathbb{N}$
unfolding *cf-singleton-def* **by** (*simp add*: *nat-omega-simps*)
show *cf-smcf* (*cf-singleton* j $\mathfrak{C}$) :
cat-smc $\mathfrak{C} \mapsto\mapsto_{SMC\alpha}$ *cat-smc* ($\prod_C i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
by (*intro is-iso-semifunctor.axioms*(*1*) *smcf-singleton*)
show *cf-singleton* j $\mathfrak{C}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor) \rfloor)$ =
($\prod_C i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)$(\lfloor CId \rfloor)(\lfloor$ *cf-singleton* j $\mathfrak{C}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor) \rfloor)$
if $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$ **for** c
proof–
from *that* **have** *CId-c*: $\mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor) : c \mapsto_{\mathfrak{C}} c$ **by** (*auto simp*: *cat-cs-intros*)
have *set* $\{\langle j, c \rangle\} \in_\circ$ ($\prod_C i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)$(\lfloor Obj \rfloor)$
by (*simp add*: *cat-singleton-ObjI that*)
with *that* **have** ($\prod_C i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)$(\lfloor CId \rfloor)(\lfloor$ *cf-singleton* j $\mathfrak{C}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor) \rfloor)$ =
set $\{\langle j, \mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor)\rangle\}$
by (*simp add*: *cf-singleton-ObjMap-app cat-singleton-CId-app*)
moreover from *CId-c* **have**
cf-singleton j $\mathfrak{C}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor) \rfloor)$ = *set* $\{\langle j, \mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor)\rangle\}$
by (*auto simp*: *cf-singleton-ArrMap-app cat-cs-intros*)
ultimately show *?thesis* **by** *simp*
qed
qed
(
auto simp:
cat-cs-intros assms cat-category-cat-singleton cf-singleton-components
)

8.12 Product of two categories

8.12.1 Definition and elementary properties.

See Chapter II-3 in [7].

definition *cat-prod-2* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\times_C\rangle$ *80*)
where $\mathfrak{A} \times_C \mathfrak{B} \equiv$ *cat-prod* $(2_{\mathbb{N}})$ $(\lambda i.$ *if* $i = 0$ *then* $\mathfrak{A}$ *else* $\mathfrak{B})$

Slicing.

lemma *cat-smc-cat-prod-2*[*slicing-commute*]:
cat-smc $\mathfrak{A} \times_{SMC}$ *cat-smc* $\mathfrak{B}$ = *cat-smc* $(\mathfrak{A} \times_C \mathfrak{B})$
unfolding *cat-prod-2-def smc-prod-2-def slicing-commute*[*symmetric*] *if-distrib*
by *simp*

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

lemmas-with
[
where $\mathfrak{A}$=$\langle$*cat-smc* $\mathfrak{A}\rangle$ **and** $\mathfrak{B}$=$\langle$*cat-smc* $\mathfrak{B}\rangle$,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*cat-semicategory* $\mathfrak{B}$.*cat-semicategory*
]:
cat-prod-2-ObjI = *smc-prod-2-ObjI*
and *cat-prod-2-ObjI'*[*cat-prod-cs-intros*] = *smc-prod-2-ObjI'*
and *cat-prod-2-ObjE* = *smc-prod-2-ObjE*
and *cat-prod-2-ArrI* = *smc-prod-2-ArrI*
and *cat-prod-2-ArrI'*[*cat-prod-cs-intros*] = *smc-prod-2-ArrI'*
and *cat-prod-2-ArrE* = *smc-prod-2-ArrE*
and *cat-prod-2-is-arrI* = *smc-prod-2-is-arrI*
and *cat-prod-2-is-arrI'*[*cat-prod-cs-intros*] = *smc-prod-2-is-arrI'*
and *cat-prod-2-is-arrE* = *smc-prod-2-is-arrE*
and *cat-prod-2-Dom-vsv* = *smc-prod-2-Dom-vsv*
and *cat-prod-2-Dom-vdomain*[*cat-cs-simps*] = *smc-prod-2-Dom-vdomain*
and *cat-prod-2-Dom-app*[*cat-prod-cs-simps*] = *smc-prod-2-Dom-app*
and *cat-prod-2-Dom-vrange* = *smc-prod-2-Dom-vrange*
and *cat-prod-2-Cod-vsv* = *smc-prod-2-Cod-vsv*
and *cat-prod-2-Cod-vdomain*[*cat-cs-simps*] = *smc-prod-2-Cod-vdomain*
and *cat-prod-2-Cod-app*[*cat-prod-cs-simps*] = *smc-prod-2-Cod-app*
and *cat-prod-2-Cod-vrange* = *smc-prod-2-Cod-vrange*
and *cat-prod-2-op-cat-cat-Obj*[*cat-op-simps*] = *smc-prod-2-op-smc-smc-Obj*
and *cat-prod-2-cat-op-cat-Obj*[*cat-op-simps*] = *smc-prod-2-smc-op-smc-Obj*
and *cat-prod-2-op-cat-cat-Arr*[*cat-op-simps*] = *smc-prod-2-op-smc-smc-Arr*
and *cat-prod-2-cat-op-cat-Arr*[*cat-op-simps*] = *smc-prod-2-smc-op-smc-Arr*

lemmas-with
[
where $\mathfrak{A}$=$\langle$*cat-smc* $\mathfrak{A}\rangle$ **and** $\mathfrak{B}$=$\langle$*cat-smc* $\mathfrak{B}\rangle$,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*cat-semicategory* $\mathfrak{B}$.*cat-semicategory*
]:
cat-prod-2-Comp-app[*cat-prod-cs-simps*] = *smc-prod-2-Comp-app*

end

8.12.2 Product of two categories is a category

context

fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule categoryD*[*OF* $\mathfrak{A}$])
interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

lemma *finite-pcategory-cat-prod-2*: *finite-pcategory* α ($2_{\mathbb{N}}$) (*if2* $\mathfrak{A}$ $\mathfrak{B}$)
proof(*intro finite-pcategoryI pcategory-baseI*)
from *Axiom-of-Infinity* **show** *z1-in-Vset*: $2_{\mathbb{N}} \in_{\circ}$ *Vset* α **by** *blast*
show *category* α (*i* = *0* ? $\mathfrak{A}$: $\mathfrak{B}$) **if** $i \in_{\circ} 2_{\mathbb{N}}$ **for** *i*
by (*auto simp*: *cat-cs-intros*)
qed *auto*

interpretation *finite-pcategory* α ‹$2_{\mathbb{N}}$› ‹*if2* $\mathfrak{A}$ $\mathfrak{B}$›
by (*intro finite-pcategory-cat-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *category-cat-prod-2*[*cat-cs-intros*]: *category* α ($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)
unfolding *cat-prod-2-def* **by** (*rule pcat-category-cat-prod*)

end

8.12.3 Identity

lemma *cat-prod-2-CId-vsv*[*cat-cs-intros*]: *vsv* (($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)(|*CId*|))
unfolding *cat-prod-2-def cat-prod-components* **by** *simp*

lemma *cat-prod-2-CId-vdomain*[*cat-cs-simps*]:
$\mathcal{D}_{\circ}$ (($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)(|*CId*|)) = ($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)(|*Obj*|)
unfolding *cat-prod-2-def cat-prod-components* **by** *simp*

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

interpretation *finite-pcategory* α ‹$2_{\mathbb{N}}$› ‹(λ*i*. *if i = 0 then* $\mathfrak{A}$ *else* $\mathfrak{B}$)›
by (*intro finite-pcategory-cat-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *cat-prod-2-CId-app*[*cat-prod-cs-simps*]:
assumes $[a, b]_{\circ} \in_{\circ}$ ($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)(|*Obj*|)
shows ($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)(|*CId*|)(|*a*, *b*|)$_{\bullet}$ = [$\mathfrak{A}$(|*CId*|)(|*a*|), $\mathfrak{B}$(|*CId*|)(|*b*|)]$_{\circ}$
proof–
have ($\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)(|*CId*|)(|*a*, *b*|)$_{\bullet}$ =
(λ*i*∈$_{\circ}$ $2_{\mathbb{N}}$. (*if i = 0 then* $\mathfrak{A}$ *else* $\mathfrak{B}$)(|*CId*|)(|[*a*, *b*]$_{\circ}$(|*i*|)|))
by
(
rule
cat-prod-CId-app[
OF assms[*unfolded cat-prod-2-def*], *folded cat-prod-2-def*
]
)
also have
(λ*i*∈$_{\circ}$ $2_{\mathbb{N}}$. (*if i = 0 then* $\mathfrak{A}$ *else* $\mathfrak{B}$)(|*CId*|)(|[*a*, *b*]$_{\circ}$(|*i*|)|)) =

```
    [𝔄(⦇CId⦈)(⦇a⦈), 𝔅(⦇CId⦈)(⦇b⦈)]∘
  proof(rule vsv-eqI, unfold vdomain-VLambda)
    fix i assume i ∈∘ 2_ℕ
    then consider ‹i = 0› | ‹i = 1_ℕ› unfolding two by auto
    then show
      (λi∈∘2_ℕ. (if i = 0 then 𝔄 else 𝔅)(⦇CId⦈)(⦇[a, b]∘(⦇i⦈)⦈))(⦇i⦈) =
        [𝔄(⦇CId⦈)(⦇a⦈), 𝔅(⦇CId⦈)(⦇b⦈)]∘(⦇i⦈)
      by cases (simp-all add: two nat-omega-simps)
  qed (auto simp: two nat-omega-simps)
  finally show ?thesis by simp
qed

lemma cat-prod-2-CId-vrange: ℛ∘ ((𝔄 ×_C 𝔅)(⦇CId⦈)) ⊆∘ (𝔄 ×_C 𝔅)(⦇Arr⦈)
proof(rule vsv.vsv-vrange-vsubset, unfold cat-cs-simps)
  show vsv ((𝔄 ×_C 𝔅)(⦇CId⦈)) by (rule cat-prod-2-CId-vsv)
  fix ab assume ab ∈∘ (𝔄 ×_C 𝔅)(⦇Obj⦈)
  then obtain a b where ab-def: ab = [a, b]∘
    and a: a ∈∘ 𝔄(⦇Obj⦈)
    and b: b ∈∘ 𝔅(⦇Obj⦈)
    by (elim cat-prod-2-ObjE[OF 𝔄 𝔅])
  from 𝔄 𝔅 a b show (𝔄 ×_C 𝔅)(⦇CId⦈)(⦇ab⦈) ∈∘ (𝔄 ×_C 𝔅)(⦇Arr⦈)
    unfolding ab-def by (cs-concl cs-intro: cat-cs-intros cat-prod-cs-intros)
qed

end
```

8.12.4 Opposite product category

```
context
  fixes α 𝔄 𝔅
  assumes 𝔄: category α 𝔄 and 𝔅: category α 𝔅
begin

interpretation 𝔄: category α 𝔄 by (rule 𝔄)
interpretation 𝔅: category α 𝔅 by (rule 𝔅)

lemma op-smc-smc-prod-2[smc-op-simps]:
  op-cat (𝔄 ×_C 𝔅) = op-cat 𝔄 ×_C op-cat 𝔅
proof(rule cat-smc-eqI [of α])
  from 𝔄 𝔅 show cat-lhs: category α (op-cat (𝔄 ×_C 𝔅))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-op-intros
      )
  interpret cat-lhs: category α ‹op-cat (𝔄 ×_C 𝔅)› by (rule cat-lhs)
  from 𝔄 𝔅 show cat-rhs: category α (op-cat 𝔄 ×_C op-cat 𝔅)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
      )
  interpret cat-rhs: category α ‹op-cat 𝔄 ×_C op-cat 𝔅› by (rule cat-rhs)
  show op-cat (𝔄 ×_C 𝔅)(⦇CId⦈) = (op-cat 𝔄 ×_C op-cat 𝔅)(⦇CId⦈)
    unfolding cat-op-simps
  proof(rule vsv-eqI, unfold cat-cs-simps)
    show vsv ((𝔄 ×_C 𝔅)(⦇CId⦈)) by (rule cat-prod-2-CId-vsv)
    show vsv ((op-cat 𝔄 ×_C op-cat 𝔅)(⦇CId⦈)) by (rule cat-prod-2-CId-vsv)
```

```
    from 𝔄 𝔅 show (𝔄 ×_C 𝔅)(|Obj|) = (op-cat 𝔄 ×_C op-cat 𝔅)(|Obj|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-op-intros
        )
    show (𝔄 ×_C 𝔅)(|CId|)(|ab|) = (op-cat 𝔄 ×_C op-cat 𝔅)(|CId|)(|ab|)
      if ab ∈_∘ (𝔄 ×_C 𝔅)(|Obj|) for ab
      using that unfolding cat-cs-simps
    proof–
      from that obtain a b
        where ab-def: ab = [a, b]_∘
          and a: a ∈_∘ 𝔄(|Obj|)
          and b: b ∈_∘ 𝔅(|Obj|)
        by (elim cat-prod-2-ObjE[OF 𝔄 𝔅])
      from 𝔄 𝔅 a b show (𝔄 ×_C 𝔅)(|CId|)(|ab|) = (op-cat 𝔄 ×_C op-cat 𝔅)(|CId|)(|ab|)
        unfolding ab-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-op-simps cat-prod-cs-simps
              cs-intro: cat-op-intros cat-prod-cs-intros
          )
    qed
  qed

  from 𝔄 𝔅 show cat-smc (op-cat (𝔄 ×_C 𝔅)) = cat-smc (op-cat 𝔄 ×_C op-cat 𝔅)
    unfolding slicing-commute[symmetric]
    by (cs-concl cs-shallow cs-simp: smc-op-simps cs-intro: slicing-intros)

qed

end
```

8.12.5 Flip

```
context
  fixes α 𝔄 𝔅
  assumes 𝔄: category α 𝔄 and 𝔅: category α 𝔅
begin

interpretation 𝔄: category α 𝔄 by (rule 𝔄)
interpretation 𝔅: category α 𝔅 by (rule 𝔅)

lemma cat-prod-2-Obj-fconverse[cat-cs-simps]:
  ((𝔄 ×_C 𝔅)(|Obj|))⁻¹_• = (𝔅 ×_C 𝔄)(|Obj|)
proof–
  interpret fbrelation ‹((𝔄 ×_C 𝔅)(|Obj|))›
    by (auto elim: cat-prod-2-ObjE[OF 𝔄 𝔅])
  show ?thesis
  proof(intro vsubset-antisym vsubsetI)
    fix ba assume prems: ba ∈_∘ ((𝔄 ×_C 𝔅)(|Obj|))⁻¹_•
    then obtain a b where ba-def: ba = [b, a]_∘ by clarsimp
    from prems[unfolded ba-def] have [a, b]_∘ ∈_∘ (𝔄 ×_C 𝔅)(|Obj|) by auto
    then have a ∈_∘ 𝔄(|Obj|) and b ∈_∘ 𝔅(|Obj|)
      by (auto elim: cat-prod-2-ObjE[OF 𝔄 𝔅])
    with 𝔄 𝔅 show ba ∈_∘ (𝔅 ×_C 𝔄)(|Obj|)
      unfolding ba-def by (cs-concl cs-shallow cs-intro: cat-prod-cs-intros)
```

```
  next
    fix ba assume ba ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
    then obtain a b
      where ba-def: ba = [b, a]∘
        and b: b ∈∘ 𝔅(|Obj|)
        and a: a ∈∘ 𝔄(|Obj|)
      by (elim cat-prod-2-ObjE[OF 𝔅 𝔄])
    from b a show ba ∈∘ ((𝔄 ×_C 𝔅)(|Obj|))⁻¹•
      unfolding ba-def by (auto simp: cat-prod-2-ObjI[OF 𝔄 𝔅 a b])
  qed
qed

lemma cat-prod-2-Arr-fconverse[cat-cs-simps]:
  ((𝔄 ×_C 𝔅)(|Arr|))⁻¹• = (𝔅 ×_C 𝔄)(|Arr|)
proof–
  interpret fbrelation ‹((𝔄 ×_C 𝔅)(|Arr|))›
    by (auto elim: cat-prod-2-ArrE[OF 𝔄 𝔅])
  show ?thesis
  proof(intro vsubset-antisym vsubsetI)
    fix ba assume prems: ba ∈∘ ((𝔄 ×_C 𝔅)(|Arr|))⁻¹•
    then obtain a b where ba-def: ba = [b, a]∘ by clarsimp
    from prems[unfolded ba-def] have [a, b]∘ ∈∘ (𝔄 ×_C 𝔅)(|Arr|) by auto
    then have a ∈∘ 𝔄(|Arr|) and b ∈∘ 𝔅(|Arr|)
      by (auto elim: cat-prod-2-ArrE[OF 𝔄 𝔅])
    with 𝔄 𝔅 show ba ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
      unfolding ba-def
      by
        (
          cs-concl
            cs-simp: cat-prod-cs-simps
            cs-intro: cat-prod-cs-intros cat-cs-intros
        )
  next
    fix ba assume ba ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
    then obtain a b
      where ba-def: ba = [b, a]∘
        and b: b ∈∘ 𝔅(|Arr|)
        and a: a ∈∘ 𝔄(|Arr|)
      by (elim cat-prod-2-ArrE[OF 𝔅 𝔄])
    from b a show ba ∈∘ ((𝔄 ×_C 𝔅)(|Arr|))⁻¹•
      unfolding ba-def by (auto simp: cat-prod-2-ArrI[OF 𝔄 𝔅 a b])
  qed
qed

end
```

8.13 Projections for the product of two categories

8.13.1 Definition and elementary properties

See Chapter II-3 in [7].

```
definition cf-proj-fst :: V ⇒ V ⇒ V (‹π_C.1›)
  where π_C.1 𝔄 𝔅 = cf-proj (2_ℕ) (λi. if i = 0 then 𝔄 else 𝔅) 0
definition cf-proj-snd :: V ⇒ V ⇒ V (‹π_C.2›)
  where π_C.2 𝔄 𝔅 = cf-proj (2_ℕ) (λi. if i = 0 then 𝔄 else 𝔅) (1_ℕ)
```

Slicing

```
lemma cf-smcf-cf-proj-fst[slicing-commute]:
```

$\pi_{SMC.1}$ (*cat-smc* $\mathfrak{A}$) (*cat-smc* $\mathfrak{B}$) = *cf-smcf* ($\pi_{C.1}$ $\mathfrak{A}$ $\mathfrak{B}$)
unfolding
cf-proj-fst-def smcf-proj-fst-def slicing-commute[*symmetric*] *if-distrib* **..**

lemma *cf-smcf-cf-proj-snd*[*slicing-commute*]:
$\pi_{SMC.2}$ (*cat-smc* $\mathfrak{A}$) (*cat-smc* $\mathfrak{B}$) = *cf-smcf* ($\pi_{C.2}$ $\mathfrak{A}$ $\mathfrak{B}$)
unfolding
cf-proj-snd-def smcf-proj-snd-def slicing-commute[*symmetric*] *if-distrib* **..**

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

lemmas-with
[
where $\mathfrak{A}$=‹*cat-smc* $\mathfrak{A}$› **and** $\mathfrak{B}$=‹*cat-smc* $\mathfrak{B}$›,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*cat-semicategory* $\mathfrak{B}$.*cat-semicategory*
]:
cf-proj-fst-ObjMap-app = *smcf-proj-fst-ObjMap-app*
and *cf-proj-snd-ObjMap-app* = *smcf-proj-snd-ObjMap-app*
and *cf-proj-fst-ArrMap-app* = *smcf-proj-fst-ArrMap-app*
and *cf-proj-snd-ArrMap-app* = *smcf-proj-snd-ArrMap-app*

end

8.13.2 Domain and codomain of a projection of a product of two categories

lemma *cf-proj-fst-HomDom*: $\pi_{C.1}$ $\mathfrak{A}$ $\mathfrak{B}$(|*HomDom*|) = $\mathfrak{A}$ $\times_C$ $\mathfrak{B}$
unfolding *cf-proj-fst-def cf-proj-components cat-prod-2-def* **..**

lemma *cf-proj-fst-HomCod*: $\pi_{C.1}$ $\mathfrak{A}$ $\mathfrak{B}$(|*HomCod*|) = $\mathfrak{A}$
unfolding *cf-proj-fst-def cf-proj-components cat-prod-2-def* **by** *simp*

lemma *cf-proj-snd-HomDom*: $\pi_{C.2}$ $\mathfrak{A}$ $\mathfrak{B}$(|*HomDom*|) = $\mathfrak{A}$ $\times_C$ $\mathfrak{B}$
unfolding *cf-proj-snd-def cf-proj-components cat-prod-2-def* **..**

lemma *cf-proj-snd-HomCod*: $\pi_{C.2}$ $\mathfrak{A}$ $\mathfrak{B}$(|*HomCod*|) = $\mathfrak{B}$
unfolding *cf-proj-snd-def cf-proj-components cat-prod-2-def* **by** *simp*

8.13.3 Projection of a product of two categories is a functor

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule categoryD*[*OF* $\mathfrak{A}$])
interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation *finite-pcategory* α ‹$2_{\mathbb{N}}$› ‹*if2* $\mathfrak{A}$ $\mathfrak{B}$›
by (*intro finite-pcategory-cat-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *cf-proj-fst-is-functor*:

assumes $i \in_{\circ} I$
shows $\pi_{C.1}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by
(
rule
pcat-cf-proj-is-functor[
where *i=0*, *simplified*, *folded cf-proj-fst-def cat-prod-2-def*
]
)

lemma *cf-proj-fst-is-functor'*[*cat-cs-intros*]:
assumes $i \in_{\circ} I$ **and** $\mathfrak{C} = \mathfrak{A} \times_C \mathfrak{B}$ **and** $\mathfrak{D} = \mathfrak{A}$
shows $\pi_{C.1}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule cf-proj-fst-is-functor*)

lemma *cf-proj-snd-is-functor*:
assumes $i \in_{\circ} I$
shows $\pi_{C.2}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by
(
rule
pcat-cf-proj-is-functor[
where *i=*$\langle 1_{\mathbb{N}} \rangle$, *simplified*, *folded cf-proj-snd-def cat-prod-2-def*
]
)

lemma *cf-proj-snd-is-functor'*[*cat-cs-intros*]:
assumes $i \in_{\circ} I$ **and** $\mathfrak{C} = \mathfrak{A} \times_C \mathfrak{B}$ **and** $\mathfrak{D} = \mathfrak{B}$
shows $\pi_{C.2}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule cf-proj-snd-is-functor*)

end

8.14 Product of three categories

8.14.1 Definition and elementary properties.

definition *cat-prod-3* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle$(- $\times_{C3}$ - $\times_{C3}$ -)$\rangle$ [*81*, *81*, *81*] *80*)
where $\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C} = (\prod_C i \in_{\circ} 3_{\mathbb{N}}.\ if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i)$

abbreviation *cat-pow-3* :: $V \Rightarrow V$ ($\langle$-$\widehat{\ }_{C3}\rangle$ [*81*] *80*)
where $\mathfrak{C}\widehat{\ }_{C3} \equiv \mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$

Slicing.

lemma *cat-smc-cat-prod-3*[*slicing-commute*]:
cat-smc $\mathfrak{A} \times_{SMC3}$ *cat-smc* $\mathfrak{B} \times_{SMC3}$ *cat-smc* $\mathfrak{C}$ = *cat-smc* $(\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})$
unfolding *cat-prod-3-def smc-prod-3-def slicing-commute*[*symmetric*] *if-distrib*
by (*simp add*: *if-distrib*[*symmetric*])

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$
assumes $\mathfrak{A}$: *category* $\alpha\ \mathfrak{A}$ **and** $\mathfrak{B}$: *category* $\alpha\ \mathfrak{B}$ **and** $\mathfrak{C}$: *category* $\alpha\ \mathfrak{C}$
begin

interpretation $\mathfrak{A}$: *category* $\alpha\ \mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* $\alpha\ \mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation $\mathfrak{C}$: *category* $\alpha\ \mathfrak{C}$ **by** (*rule* $\mathfrak{C}$)

lemmas-with
[
where $\mathfrak{A}$=‹*cat-smc* $\mathfrak{A}$› **and** $\mathfrak{B}$=‹*cat-smc* $\mathfrak{B}$› **and** $\mathfrak{C}$=‹*cat-smc* $\mathfrak{C}$›,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*cat-semicategory* $\mathfrak{B}$.*cat-semicategory* $\mathfrak{C}$.*cat-semicategory*
]:
cat-prod-3-ObjI = *smc-prod-3-ObjI*
and *cat-prod-3-ObjI'*[*cat-prod-cs-intros*] = *smc-prod-3-ObjI'*
and *cat-prod-3-ObjE* = *smc-prod-3-ObjE*
and *cat-prod-3-ArrI* = *smc-prod-3-ArrI*
and *cat-prod-3-ArrI'*[*cat-prod-cs-intros*] = *smc-prod-3-ArrI'*
and *cat-prod-3-ArrE* = *smc-prod-3-ArrE*
and *cat-prod-3-is-arrI* = *smc-prod-3-is-arrI*
and *cat-prod-3-is-arrI'*[*cat-prod-cs-intros*] = *smc-prod-3-is-arrI'*
and *cat-prod-3-is-arrE* = *smc-prod-3-is-arrE*
and *cat-prod-3-Dom-vsv* = *smc-prod-3-Dom-vsv*
and *cat-prod-3-Dom-vdomain*[*cat-cs-simps*] = *smc-prod-3-Dom-vdomain*
and *cat-prod-3-Dom-app*[*cat-prod-cs-simps*] = *smc-prod-3-Dom-app*
and *cat-prod-3-Dom-vrange* = *smc-prod-3-Dom-vrange*
and *cat-prod-3-Cod-vsv* = *smc-prod-3-Cod-vsv*
and *cat-prod-3-Cod-vdomain*[*cat-cs-simps*] = *smc-prod-3-Cod-vdomain*
and *cat-prod-3-Cod-app*[*cat-prod-cs-simps*] = *smc-prod-3-Cod-app*
and *cat-prod-3-Cod-vrange* = *smc-prod-3-Cod-vrange*

lemmas-with
[
where $\mathfrak{A}$=‹*cat-smc* $\mathfrak{A}$› **and** $\mathfrak{B}$=‹*cat-smc* $\mathfrak{B}$› **and** $\mathfrak{C}$=‹*cat-smc* $\mathfrak{C}$›,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*cat-semicategory* $\mathfrak{B}$.*cat-semicategory* $\mathfrak{C}$.*cat-semicategory*
]:
cat-prod-3-Comp-app[*cat-prod-cs-simps*] = *smc-prod-3-Comp-app*

end

8.14.2 Product of three categories is a category

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$ **and** $\mathfrak{C}$: *category* α $\mathfrak{C}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule categoryD*[*OF* $\mathfrak{A}$])
interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule* $\mathfrak{C}$)

lemma *finite-pcategory-cat-prod-3*: *finite-pcategory* α ($3_{\mathbb{N}}$) (*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$)
proof(*intro finite-pcategoryI pcategory-baseI*)
from *Axiom-of-Infinity* **show** *z1-in-Vset*: $3_{\mathbb{N}}$ $\in_{\circ}$ *Vset* α **by** *blast*
show *category* α (*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ *i*) **if** *i* $\in_{\circ}$ $3_{\mathbb{N}}$ **for** *i*
by (*auto simp*: *cat-cs-intros*)
qed *auto*

interpretation *finite-pcategory* α ‹$3_{\mathbb{N}}$› ‹*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$›
by (*intro finite-pcategory-cat-prod-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$)

lemma *category-cat-prod-3*[*cat-cs-intros*]: *category* α ($\mathfrak{A}$ $\times_{C3}$ $\mathfrak{B}$ $\times_{C3}$ $\mathfrak{C}$)
unfolding *cat-prod-3-def* **by** (*rule pcat-category-cat-prod*)

end

8.14.3 Identity

lemma *cat-prod-3-CId-vsv*[*cat-cs-intros*]: *vsv* ((𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*CId*|))
unfolding *cat-prod-3-def cat-prod-components* **by** *simp*

lemma *cat-prod-3-CId-vdomain*[*cat-cs-simps*]:
$\mathcal{D}_\circ$ ((𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*CId*|)) = (𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*Obj*|)
unfolding *cat-prod-3-def cat-prod-components* **by** *simp*

context
fixes α 𝔄 𝔅 ℭ
assumes 𝔄: *category* α 𝔄 **and** 𝔅: *category* α 𝔅 **and** ℭ: *category* α ℭ
begin

interpretation 𝔄: *category* α 𝔄 **by** (*rule* 𝔄)
interpretation 𝔅: *category* α 𝔅 **by** (*rule* 𝔅)
interpretation ℭ: *category* α ℭ **by** (*rule* ℭ)

interpretation *finite-pcategory* α ‹$3_{\mathbb{N}}$› ‹*if3* 𝔄 𝔅 ℭ›
by (*intro finite-pcategory-cat-prod-3* 𝔄 𝔅 ℭ)

lemma *cat-prod-3-CId-app*[*cat-prod-cs-simps*]:
assumes [*a*, *b*, *c*]$_\circ$ ∈$_\circ$ (𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*Obj*|)
shows (𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*CId*|)(|*a*, *b*, *c*|)$_\bullet$ = [𝔄(|*CId*|)(|*a*|), 𝔅(|*CId*|)(|*b*|), ℭ(|*CId*|)(|*c*|)]$_\circ$
proof–
have (𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*CId*|)(|*a*, *b*, *c*|)$_\bullet$ =
(λ*i*∈$_\circ$$3_{\mathbb{N}}$. *if3* 𝔄 𝔅 ℭ *i*(|*CId*|)(|[*a*, *b*, *c*]$_\circ$(|*i*|)|))
by
(
rule
cat-prod-CId-app[
OF assms[*unfolded cat-prod-3-def*], *folded cat-prod-3-def*
]
)
also have
(λ*i*∈$_\circ$$3_{\mathbb{N}}$. *if3* 𝔄 𝔅 ℭ *i*(|*CId*|)(|[*a*, *b*, *c*]$_\circ$(|*i*|)|)) = [𝔄(|*CId*|)(|*a*|), 𝔅(|*CId*|)(|*b*|), ℭ(|*CId*|)(|*c*|)]$_\circ$
proof(*rule vsv-eqI*, *unfold vdomain-VLambda*)
fix *i* **assume** *i* ∈$_\circ$ $3_{\mathbb{N}}$
then consider ‹*i* = *0*› | ‹*i* = $1_{\mathbb{N}}$› | ‹*i* = $2_{\mathbb{N}}$› **unfolding** *three* **by** *auto*
then show
(λ*i*∈$_\circ$$3_{\mathbb{N}}$. (*if3* 𝔄 𝔅 ℭ *i*)(|*CId*|)(|[*a*, *b*, *c*]$_\circ$(|*i*|)|))(|*i*|) =
[𝔄(|*CId*|)(|*a*|), 𝔅(|*CId*|)(|*b*|), ℭ(|*CId*|)(|*c*|)]$_\circ$(|*i*|)
by *cases* (*simp-all add*: *three nat-omega-simps*)
qed (*auto simp*: *three nat-omega-simps*)
finally show *?thesis* **by** *simp*
qed

lemma *cat-prod-3-CId-vrange*:
$\mathcal{R}_\circ$ ((𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*CId*|)) ⊆$_\circ$ (𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*Arr*|)
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-cs-simps*)
show *vsv* ((𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*CId*|)) **by** (*rule cat-prod-3-CId-vsv*)
fix *abc* **assume** *abc* ∈$_\circ$ (𝔄 ×$_{C3}$ 𝔅 ×$_{C3}$ ℭ)(|*Obj*|)
then obtain *a b c* **where** *abc-def*: *abc* = [*a*, *b*, *c*]$_\circ$
and *a*: *a* ∈$_\circ$ 𝔄(|*Obj*|)
and *b*: *b* ∈$_\circ$ 𝔅(|*Obj*|)

and c: $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
by (*elim cat-prod-3-ObjE*[*OF* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$])
from $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ a b c **show** $(\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|CId|\!)(\!|abc|\!) \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!)$
unfolding *abc-def*
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*)
qed

end

8.15 Conversion of a product of three categories to products of two categories

definition *cf-cat-prod-21-of-3* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ =
[
$(\lambda A \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!).\ [[A(\!|0|\!),\ A(\!|1_{\mathbb{N}}|\!)]_\circ,\ A(\!|2_{\mathbb{N}}|\!)]_\circ)$,
$(\lambda F \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!).\ [[F(\!|0|\!),\ F(\!|1_{\mathbb{N}}|\!)]_\circ,\ F(\!|2_{\mathbb{N}}|\!)]_\circ)$,
$\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C}$,
$(\mathfrak{A} \times_C \mathfrak{B}) \times_C \mathfrak{C}$
]$_\circ$

definition *cf-cat-prod-12-of-3* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ =
[
$(\lambda A \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!).\ [A(\!|0|\!),\ [A(\!|1_{\mathbb{N}}|\!),\ A(\!|2_{\mathbb{N}}|\!)]_\circ]_\circ)$,
$(\lambda F \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!).\ [F(\!|0|\!),\ [F(\!|1_{\mathbb{N}}|\!),\ F(\!|2_{\mathbb{N}}|\!)]_\circ]_\circ)$,
$\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C}$,
$\mathfrak{A} \times_C (\mathfrak{B} \times_C \mathfrak{C})$
]$_\circ$

Components.

lemma *cf-cat-prod-21-of-3-components*:
shows *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!) =$
$(\lambda A \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!).\ [[A(\!|0|\!),\ A(\!|1_{\mathbb{N}}|\!)]_\circ,\ A(\!|2_{\mathbb{N}}|\!)]_\circ)$
and *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ArrMap|\!) =$
$(\lambda F \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!).\ [[F(\!|0|\!),\ F(\!|1_{\mathbb{N}}|\!)]_\circ,\ F(\!|2_{\mathbb{N}}|\!)]_\circ)$
and [*cat-cs-simps*]: *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|HomDom|\!) = \mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C}$
and [*cat-cs-simps*]: *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|HomCod|\!) = (\mathfrak{A} \times_C \mathfrak{B}) \times_C \mathfrak{C}$
unfolding *cf-cat-prod-21-of-3-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

lemma *cf-cat-prod-12-of-3-components*:
shows *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!) =$
$(\lambda A \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!).\ [A(\!|0|\!),\ [A(\!|1_{\mathbb{N}}|\!),\ A(\!|2_{\mathbb{N}}|\!)]_\circ]_\circ)$
and *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ArrMap|\!) =$
$(\lambda F \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!).\ [F(\!|0|\!),\ [F(\!|1_{\mathbb{N}}|\!),\ F(\!|2_{\mathbb{N}}|\!)]_\circ]_\circ)$
and [*cat-cs-simps*]: *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|HomDom|\!) = \mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C}$
and [*cat-cs-simps*]: *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|HomCod|\!) = \mathfrak{A} \times_C (\mathfrak{B} \times_C \mathfrak{C})$
unfolding *cf-cat-prod-12-of-3-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

8.15.1 Object

mk-VLambda *cf-cat-prod-21-of-3-components*(*1*)
|*vsv cf-cat-prod-21-of-3-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-cat-prod-21-of-3-ObjMap-vdomain*[*cat-cs-simps*]|
|*app cf-cat-prod-21-of-3-ObjMap-app'*|

mk-VLambda *cf-cat-prod-12-of-3-components*(*1*)

|*vsv cf-cat-prod-12-of-3-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-cat-prod-12-of-3-ObjMap-vdomain*[*cat-cs-simps*]|
|*app cf-cat-prod-12-of-3-ObjMap-app'*|

lemma *cf-cat-prod-21-of-3-ObjMap-app*[*cat-cs-simps*]:
assumes $A = [a, b, c]_\circ$ **and** $[a, b, c]_\circ \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!)$
shows *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|A|\!) = [[a, b]_\circ, c]_\circ$
using *assms*(*2*)
unfolding *assms*(*1*)
by (*simp add*: *cf-cat-prod-21-of-3-ObjMap-app'* *nat-omega-simps*)

lemma *cf-cat-prod-12-of-3-ObjMap-app*[*cat-cs-simps*]:
assumes $A = [a, b, c]_\circ$ **and** $[a, b, c]_\circ \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!)$
shows *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|A|\!) = [a, [b, c]_\circ]_\circ$
using *assms*(*2*)
unfolding *assms*(*1*)
by (*simp add*: *cf-cat-prod-12-of-3-ObjMap-app'* *nat-omega-simps*)

lemma *cf-cat-prod-21-of-3-ObjMap-vrange*:
assumes *category* α $\mathfrak{A}$ **and** *category* α $\mathfrak{B}$ **and** *category* α $\mathfrak{C}$
shows $\mathcal{R}_\circ$ (*cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_\circ$ $((\mathfrak{A} \times_C \mathfrak{B}) \times_C \mathfrak{C})(\!|Obj|\!)$
proof–
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*3*))
show *?thesis*
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-cat-prod-21-of-3-ObjMap-vdomain*)
fix *A* **assume** *prems*: $A \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!)$
then show *cf-cat-prod-21-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|A|\!) \in_\circ ((\mathfrak{A} \times_C \mathfrak{B}) \times_C \mathfrak{C})(\!|Obj|\!)$
by (*elim cat-prod-3-ObjE*[*OF assms*], *insert prems*, *simp only*:)
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-prod-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
qed

lemma *cf-cat-prod-12-of-3-ObjMap-vrange*:
assumes *category* α $\mathfrak{A}$ **and** *category* α $\mathfrak{B}$ **and** *category* α $\mathfrak{C}$
shows $\mathcal{R}_\circ$ (*cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_\circ$ $(\mathfrak{A} \times_C (\mathfrak{B} \times_C \mathfrak{C}))(\!|Obj|\!)$
proof–
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*3*))
show *?thesis*
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-cat-prod-12-of-3-ObjMap-vdomain*)
fix *A* **assume** *prems*: $A \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!)$
then show *cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|A|\!) \in_\circ (\mathfrak{A} \times_C (\mathfrak{B} \times_C \mathfrak{C}))(\!|Obj|\!)$
by (*elim cat-prod-3-ObjE*[*OF assms*], *insert prems*, *simp only*:)
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-prod-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
qed

8.15.2 Arrow

mk-VLambda *cf-cat-prod-21-of-3-components*(*2*)
|*vsv cf-cat-prod-21-of-3-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-cat-prod-21-of-3-ArrMap-vdomain*[*cat-cs-simps*]|
|*app cf-cat-prod-21-of-3-ArrMap-app′*|

mk-VLambda *cf-cat-prod-12-of-3-components*(*2*)
|*vsv cf-cat-prod-12-of-3-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-cat-prod-12-of-3-ArrMap-vdomain*[*cat-cs-simps*]|
|*app cf-cat-prod-12-of-3-ArrMap-app′*|

lemma *cf-cat-prod-21-of-3-ArrMap-app*[*cat-cs-simps*]:
assumes $F = [h,\ g,\ f]_{\circ}$ **and** $[h,\ g,\ f]_{\circ} \in_{\circ} (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!)$
shows *cf-cat-prod-21-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ArrMap|\!)(\!|F|\!) = [[h,\ g]_{\circ},\ f]_{\circ}$
using *assms*(*2*) **unfolding** *assms*(*1*)
by (*simp add*: *cf-cat-prod-21-of-3-ArrMap-app′ nat-omega-simps*)

lemma *cf-cat-prod-12-of-3-ArrMap-app*[*cat-cs-simps*]:
assumes $F = [h,\ g,\ f]_{\circ}$ **and** $[h,\ g,\ f]_{\circ} \in_{\circ} (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Arr|\!)$
shows *cf-cat-prod-12-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ArrMap|\!)(\!|F|\!) = [h,\ [g,\ f]_{\circ}]_{\circ}$
using *assms*(*2*)
unfolding *assms*(*1*)
by (*simp add*: *cf-cat-prod-12-of-3-ArrMap-app′ nat-omega-simps*)

8.15.3 Conversion of a product of three categories to products of two categories is a functor

lemma *cf-cat-prod-21-of-3-is-functor*:
assumes *category* $\alpha\ \mathfrak{A}$ **and** *category* $\alpha\ \mathfrak{B}$ **and** *category* $\alpha\ \mathfrak{C}$
shows *cf-cat-prod-21-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C} : \mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C} \mapsto\mapsto_{C\alpha} (\mathfrak{A} \times_{C} \mathfrak{B}) \times_{C} \mathfrak{C}$
proof–

interpret $\mathfrak{A}$: *category* $\alpha\ \mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* $\alpha\ \mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{C}$: *category* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*3*))

show *?thesis*
proof(*rule is-functorI′*)
show *vfsequence* (*cf-cat-prod-21-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$)
unfolding *cf-cat-prod-21-of-3-def* **by** *auto*
show *vcard* (*cf-cat-prod-21-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$) = $\mathcal{4}_{\mathbb{N}}$
unfolding *cf-cat-prod-21-of-3-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*cf-cat-prod-21-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_{\circ}$ $((\mathfrak{A} \times_{C} \mathfrak{B}) \times_{C} \mathfrak{C})(\!|Obj|\!)$
by (*rule cf-cat-prod-21-of-3-ObjMap-vrange*[*OF assms*])
show
cf-cat-prod-21-of-3 $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ArrMap|\!)(\!|F|\!)$:
cf-cat-prod-21-of-3 $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ObjMap|\!)(\!|A|\!) \mapsto_{(\mathfrak{A} \times_{C} \mathfrak{B}) \times_{C} \mathfrak{C}}$
cf-cat-prod-21-of-3 $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ObjMap|\!)(\!|B|\!)$
if $F : A \mapsto_{\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C}} B$
for *A B F*
using *that*
by (*elim cat-prod-3-is-arrE*[*OF assms*], *insert that*, *simp only*:)
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
show

```
      cf-cat-prod-21-of-3 𝔄 𝔅 ℭ(|ArrMap|)(|G ∘_A𝔄 ×_C3 𝔅 ×_C3 ℭ F|) =
        cf-cat-prod-21-of-3 𝔄 𝔅 ℭ(|ArrMap|)(|G|) ∘_A(𝔄 ×_C 𝔅) ×_C ℭ
        cf-cat-prod-21-of-3 𝔄 𝔅 ℭ(|ArrMap|)(|F|)
      if G : B ↦_𝔄 ×_C3 𝔅 ×_C3 ℭ C and F : A ↦_𝔄 ×_C3 𝔅 ×_C3 ℭ B
      for B C G A F
    proof–
      from that(2) obtain f f' f'' a a' a'' b b' b''
        where F-def: F = [f, f', f'']∘
          and A-def: A = [a, a', a'']∘
          and B-def: B = [b, b', b'']∘
          and f: f : a ↦_𝔄 b
          and f': f' : a' ↦_𝔅 b'
          and f'': f'' : a'' ↦_ℭ b''
        by (elim cat-prod-3-is-arrE[OF assms])
      with that(1) obtain g g' g'' c c' c''
        where G-def: G = [g, g', g'']∘
          and C-def: C = [c, c', c'']∘
          and g: g : b ↦_𝔄 c
          and g': g' : b' ↦_𝔅 c'
          and g'': g'' : b'' ↦_ℭ c''
        by (auto elim: cat-prod-3-is-arrE[OF assms])
      from that f f' f'' g g' g'' show ?thesis
        unfolding F-def A-def B-def G-def C-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cat-prod-cs-simps
              cs-intro: cat-cs-intros cat-prod-cs-intros
          )
    qed
    show
      cf-cat-prod-21-of-3 𝔄 𝔅 ℭ(|ArrMap|)(|(𝔄 ×_C3 𝔅 ×_C3 ℭ)(|CId|)(|C|)|) =
        ((𝔄 ×_C 𝔅) ×_C ℭ)(|CId|)(|cf-cat-prod-21-of-3 𝔄 𝔅 ℭ(|ObjMap|)(|C|)|)
      if C ∈_∘ (𝔄 ×_C3 𝔅 ×_C3 ℭ)(|Obj|) for C
      using that
      by (elim cat-prod-3-ObjE[OF assms], insert that, simp only: )
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-prod-cs-simps
            cs-intro: cat-cs-intros cat-prod-cs-intros
        )
  qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

qed

lemma cf-cat-prod-21-of-3-is-functor'[cat-cs-intros]:
  assumes category α 𝔄
    and category α 𝔅
    and category α ℭ
    and 𝔄' = 𝔄 ×_C3 𝔅 ×_C3 ℭ
    and 𝔅' = (𝔄 ×_C 𝔅) ×_C ℭ
  shows cf-cat-prod-21-of-3 𝔄 𝔅 ℭ : 𝔄' ↦↦_Cα 𝔅'
  using assms(1–3) unfolding assms(4,5) by (rule cf-cat-prod-21-of-3-is-functor)

lemma cf-cat-prod-12-of-3-is-functor:
  assumes category α 𝔄 and category α 𝔅 and category α ℭ
  shows cf-cat-prod-12-of-3 𝔄 𝔅 ℭ : 𝔄 ×_C3 𝔅 ×_C3 ℭ ↦↦_Cα 𝔄 ×_C (𝔅 ×_C ℭ)
proof–
```

interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*3*))

show *?thesis*
proof(*rule is-functorI'*)
show *vfsequence* (*cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$)
unfolding *cf-cat-prod-12-of-3-def* **by** *auto*
show *vcard* (*cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$) = $\mathfrak{4}_{\mathbb{N}}$
unfolding *cf-cat-prod-12-of-3-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*cf-cat-prod-12-of-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ObjMap*|)) $\subseteq_{\circ}$ ($\mathfrak{A}$ $\times_C$ ($\mathfrak{B}$ $\times_C$ $\mathfrak{C}$))(|*Obj*|)
by (*rule cf-cat-prod-12-of-3-ObjMap-vrange*[*OF assms*])
show
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|*F*|) :
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ObjMap*|)(|*A*|) $\mapsto_{\mathfrak{A}\times_C(\mathfrak{B}\times_C\mathfrak{C})}$
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ObjMap*|)(|*B*|)
if $F : A \mapsto_{\mathfrak{A}\times_{C3}\mathfrak{B}\times_{C3}\mathfrak{C}} B$
for *A B F*
using *that*
by (*elim cat-prod-3-is-arrE*[*OF assms*], *insert that*, *simp only*:)
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
show
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|$G \circ_{A\mathfrak{A}\times_{C3}\mathfrak{B}\times_{C3}\mathfrak{C}} F$|) =
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|*G*|) $\circ_{A\mathfrak{A}\times_C(\mathfrak{B}\times_C\mathfrak{C})}$
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|*F*|)
if $G : B \mapsto_{\mathfrak{A}\times_{C3}\mathfrak{B}\times_{C3}\mathfrak{C}} C$ **and** $F : A \mapsto_{\mathfrak{A}\times_{C3}\mathfrak{B}\times_{C3}\mathfrak{C}} B$
for *B C G A F*
proof–
from *that*(*2*) **obtain** *f f' f'' a a' a'' b b' b''*
where *F-def*: $F = [f, f', f'']_{\circ}$
and *A-def*: $A = [a, a', a'']_{\circ}$
and *B-def*: $B = [b, b', b'']_{\circ}$
and *f*: $f : a \mapsto_{\mathfrak{A}} b$
and *f'*: $f' : a' \mapsto_{\mathfrak{B}} b'$
and *f''*: $f'' : a'' \mapsto_{\mathfrak{C}} b''$
by (*elim cat-prod-3-is-arrE*[*OF assms*])
with *that*(*1*) **obtain** *g g' g'' c c' c''*
where *G-def*: $G = [g, g', g'']_{\circ}$
and *C-def*: $C = [c, c', c'']_{\circ}$
and *g*: $g : b \mapsto_{\mathfrak{A}} c$
and *g'*: $g' : b' \mapsto_{\mathfrak{B}} c'$
and *g''*: $g'' : b'' \mapsto_{\mathfrak{C}} c''$
by (*auto elim*: *cat-prod-3-is-arrE*[*OF assms*])
from *that f f' f'' g g' g''* **show** *?thesis*
unfolding *F-def A-def B-def G-def C-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-prod-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed
show
cf-cat-prod-12-of-3 $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|($\mathfrak{A}$ $\times_{C3}$ $\mathfrak{B}$ $\times_{C3}$ $\mathfrak{C}$)(|*CId*|)(|*C*|)|) =

$(\mathfrak{A} \times_C (\mathfrak{B} \times_C \mathfrak{C}))(\!|CId|\!)(\!|cf\text{-}cat\text{-}prod\text{-}12\text{-}of\text{-}3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}(\!|ObjMap|\!)(\!|C|\!)|\!)$
if $C \in_\circ (\mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C})(\!|Obj|\!)$ **for** C
using *that*
by (*elim cat-prod-3-ObjE*[*OF assms*], *insert that*, *simp only*:)
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-prod-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)+

qed

lemma *cf-cat-prod-12-of-3-is-functor'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and *category* α $\mathfrak{C}$
and $\mathfrak{A}' = \mathfrak{A} \times_{C3} \mathfrak{B} \times_{C3} \mathfrak{C}$
and $\mathfrak{B}' = \mathfrak{A} \times_C (\mathfrak{B} \times_C \mathfrak{C})$
shows *cf-cat-prod-12-of-3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
using *assms*(*1–3*) **unfolding** *assms*(*4*,*5*) **by** (*rule cf-cat-prod-12-of-3-is-functor*)

8.16 Bifunctors

A bifunctor is defined as a functor from a product of two categories to a category (see Chapter II-3 in [7]). This subsection exposes the elementary properties of the projections of the bifunctors established by fixing an argument in a functor (see Chapter II-3 in [7] for further information).

8.16.1 Definitions and elementary properties

definition *bifunctor-proj-fst* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
$(\langle\text{-},\text{-}/'(/\text{-},\text{-}/')/_{CF}\rangle\ [51, 51, 51, 51]\ 51)$
where $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} =$
$(\mathfrak{S}_{\prod_C i \in_\circ 2_{\mathbb{N}}\ -_\circ\ set\ \{1_{\mathbb{N}}\}.\ (i = 0\ ?\ \mathfrak{A} : \mathfrak{B}),\mathfrak{S}(\!|HomCod|\!)}(-,set\ \{\langle 1_{\mathbb{N}},\ b\rangle\}))\ \circ_{CF}$
cf-singleton 0 $\mathfrak{A}$

definition *bifunctor-proj-snd* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
$(\langle\text{-},\text{-}/'(/\text{-},\text{-}/')/_{CF}\rangle\ [51, 51, 51, 51]\ 51)$
where $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} =$
$(\mathfrak{S}_{\prod_C i \in_\circ 2_{\mathbb{N}}\ -_\circ\ set\ \{0\}.\ (i = 0\ ?\ \mathfrak{A} : \mathfrak{B}),\mathfrak{S}(\!|HomCod|\!)}(-,set\ \{\langle 0,\ a\rangle\}))\ \circ_{CF}$
cf-singleton $(1_{\mathbb{N}})$ $\mathfrak{B}$

abbreviation *bcf-ObjMap-app* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (**infixl** $\langle\otimes_{HM.O^1}\rangle$ *55*)
where $a \otimes_{HM.O\mathfrak{S}} b \equiv \mathfrak{S}(\!|ObjMap|\!)(\!|a,\ b|\!)_\bullet$
abbreviation *bcf-ArrMap-app* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (**infixl** $\langle\otimes_{HM.A^1}\rangle$ *55*)
where $g \otimes_{HM.A\mathfrak{S}} f \equiv \mathfrak{S}(\!|ArrMap|\!)(\!|g,\ f|\!)_\bullet$

Elementary properties.

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation *finite-pcategory* α $\langle 2_{\mathbb{N}}\rangle$ $\langle$*if2* $\mathfrak{A}$ $\mathfrak{B}\rangle$
by (*intro finite-pcategory-cat-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

```
lemma cat-singleton-qm-fst-def[simp]:
  (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅)) = (∏_C i∈_∘set {0}. 𝔄)
proof(rule cat-eqI[of α])
  show (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Obj|) = (∏_C i∈_∘set {0}. 𝔄)(|Obj|)
    unfolding cat-prod-components by (subst vproduct-vsingleton-def) simp
  show [simp]: (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Arr|) = (∏_C i∈_∘set {0}. 𝔄)(|Arr|)
    unfolding cat-prod-components by (subst vproduct-vsingleton-def) simp
  show [simp]: (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Dom|) = (∏_C i∈_∘set {0}. 𝔄)(|Dom|)
    unfolding cat-prod-components
    by (subst vproduct-vsingleton-def, subst (1 2) VLambda-vsingleton-def) simp
  show [simp]:
    (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Cod|) = (∏_C i∈_∘set {0}. 𝔄)(|Cod|)
    unfolding cat-prod-components
    by (subst vproduct-vsingleton-def, subst (1 2) VLambda-vsingleton-def) simp
  have [simp]:
    f : a ↦_(∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅)) b ⟷
      f : a ↦_(∏_C i∈_∘set {0}. 𝔄) b
    for f a b
    unfolding is-arr-def by simp
  show (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Comp|) = (∏_C i∈_∘set {0}. 𝔄)(|Comp|)
  proof(rule vsv-eqI)
    show vsv ((∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Comp|))
      unfolding cat-prod-components by simp
    show vsv ((∏_C i∈_∘set {0}. 𝔄)(|Comp|))
      unfolding cat-prod-components by simp
    show 𝒟_∘ ((∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Comp|)) =
      𝒟_∘ ((∏_C i∈_∘set {0}. 𝔄)(|Comp|))
      by (simp add: composable-arrs-def cat-cs-simps)
    show (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Comp|)(|gf|) =
      (∏_C i∈_∘set {0}. 𝔄)(|Comp|)(|gf|)
      if gf ∈_∘ 𝒟_∘ ((∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|Comp|)) for gf
    proof–
      from that have gf ∈_∘ composable-arrs (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))
        by (simp add: cat-cs-simps)
      then obtain g f a b c where gf-def: gf = [g, f]_∘
        and g: g : b ↦_(∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅)) c
        and f: f : a ↦_(∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅)) b
        by clarsimp
      then have g′: g : b ↦_(∏_C i∈_∘set {0}. 𝔄) c
        and f′: f : a ↦_(∏_C i∈_∘set {0}. 𝔄) b
        by simp-all
      show ?thesis
        unfolding gf-def
        unfolding cat-prod-Comp-app[OF g f] cat-prod-Comp-app[OF g′ f′]
        by (subst (1 2) VLambda-vsingleton-def) simp
    qed
  qed
  show (∏_C i∈_∘set {0}. (i = 0 ? 𝔄 : 𝔅))(|CId|) = (∏_C i∈_∘set {0}. 𝔄)(|CId|)
    unfolding cat-prod-components
    by (subst vproduct-vsingleton-def, subst (1 2) VLambda-vsingleton-def) simp
qed
  (
    simp-all add:
      𝔄.cat-category-cat-singleton
      pcategory.pcat-category-cat-prod
      pcat-vsubset-index-pcategory
```

```
    vsubset-vsingleton-leftI
  )

lemma cat-singleton-qm-snd-def[simp]:
  (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅)) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)
proof(rule cat-eqI[of α])
  show (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Obj|) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Obj|)
    unfolding cat-prod-components by (subst vproduct-vsingleton-def) simp
  show [simp]:
    (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Arr|) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Arr|)
    unfolding cat-prod-components by (subst vproduct-vsingleton-def) simp
  show [simp]:
    (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Dom|) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Dom|)
    unfolding cat-prod-components
    by (subst vproduct-vsingleton-def, subst (1 2) VLambda-vsingleton-def) simp
  show [simp]:
    (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Cod|) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Cod|)
    unfolding cat-prod-components
    by (subst vproduct-vsingleton-def, subst (1 2) VLambda-vsingleton-def) simp
  have [simp]: f : a ↦_(∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅)) b ⟷
    f : a ↦_(∏_C i∈_∘ set {1_ℕ}. 𝔅) b
    for f a b
    unfolding is-arr-def by simp
  show (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Comp|) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Comp|)
  proof(rule vsv-eqI)
    show vsv ((∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Comp|))
      unfolding cat-prod-components by simp
    show vsv ((∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Comp|))
      unfolding cat-prod-components by simp
    show 𝒟_∘ ((∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Comp|)) =
      𝒟_∘ ((∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Comp|))
      by (simp add: composable-arrs-def cat-cs-simps)
    show (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Comp|)(|gf|) =
      (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|Comp|)(|gf|)
      if gf ∈_∘ 𝒟_∘ ((∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|Comp|)) for gf
    proof–
      from that have gf ∈_∘ composable-arrs (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))
        by (simp add: cat-cs-simps)
      then obtain g f a b c where gf-def: gf = [g, f]_∘
        and g: g : b ↦_(∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅)) c
        and f: f : a ↦_(∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅)) b
        by clarsimp
      then have g′: g : b ↦_(∏_C i∈_∘ set {1_ℕ}. 𝔅) c
        and f′: f : a ↦_(∏_C i∈_∘ set {1_ℕ}. 𝔅) b
        by simp-all
      show ?thesis
        unfolding gf-def
        unfolding cat-prod-Comp-app[OF g f] cat-prod-Comp-app[OF g′ f′]
        by (subst (1 2) VLambda-vsingleton-def) simp
    qed
  qed
  show (∏_C i∈_∘ set {1_ℕ}. (i = 0 ? 𝔄 : 𝔅))(|CId|) = (∏_C i∈_∘ set {1_ℕ}. 𝔅)(|CId|)
    unfolding cat-prod-components
    by (subst vproduct-vsingleton-def, subst (1 2) VLambda-vsingleton-def) simp
qed
  (
    simp-all add:
```

```
    𝔅.cat-category-cat-singleton
    pcategory.pcat-category-cat-prod
    pcat-vsubset-index-pcategory
    vsubset-vsingleton-leftI
  )

end
```

8.16.2 Object map

```
context
  fixes α 𝔄 𝔅
  assumes 𝔄: category α 𝔄 and 𝔅: category α 𝔅
begin

interpretation 𝔄: category α 𝔄 by (rule 𝔄)
interpretation 𝔅: category α 𝔅 by (rule 𝔅)

interpretation finite-pcategory α ‹2_ℕ› ‹if2 𝔄 𝔅›
  by (intro finite-pcategory-cat-prod-2 𝔄 𝔅)

lemmas-with [OF 𝔄.category-axioms 𝔅.category-axioms, simp]:
  cat-singleton-qm-fst-def and cat-singleton-qm-snd-def

lemma bifunctor-proj-fst-ObjMap-app[cat-cs-simps]:
  assumes [a, b]ₒ ∈ₒ (𝔄 ×_C 𝔅)⦇Obj⦈
  shows (𝔖_{𝔄,𝔅}(−,b)_{C F})⦇ObjMap⦈⦇a⦈ = 𝔖⦇ObjMap⦈⦇a, b⦈.
proof–

  let ?𝔇 = ‹𝔖⦇HomCod⦈›
  let ?𝔖 = ‹𝔖_{∏_C i∈ₒ 2_ℕ−ₒset {1_ℕ}.(i = 0 ? 𝔄 : 𝔅),?𝔇}(−,set {⟨1_ℕ, b⟩})›
  let ?cfs = ‹cf-singleton 0 𝔄›

  from assms have a: a ∈ₒ 𝔄⦇Obj⦈ and b: b ∈ₒ 𝔅⦇Obj⦈
    by (all‹elim cat-prod-2-ObjE[OF 𝔄 𝔅]›) auto

  from a have za: set {⟨0, a⟩} ∈ₒ (∏_C i∈ₒset {0}. 𝔄)⦇Obj⦈
    by (intro cat-singleton-ObjI[where a=a]) simp
  have [simp]: vinsert ⟨0, a⟩ (set {⟨1_ℕ, b⟩}) = [a, b]ₒ
    using ord-of-nat-succ-vempty unfolding vcons-def
    by (simp add: vinsert-vempty insert-commute vinsert-vsingleton)

  have (𝔖_{𝔄,𝔅}(−,b)_{C F})⦇ObjMap⦈⦇a⦈ = (?𝔖⦇ObjMap⦈ ∘ₒ ?cfs⦇ObjMap⦈)⦇a⦈
    unfolding bifunctor-proj-fst-def dghm-comp-components by simp
  also have ... = ?𝔖⦇ObjMap⦈⦇?cfs⦇ObjMap⦈⦇a⦈⦈
    by (rule vsv-vcomp-at)
      (
        simp-all add:
          two a za
          cf-singleton-components
          prodfunctor-proj-components
          cf-singleton-ObjMap-app


      )
  also from za have ... = 𝔖⦇ObjMap⦈⦇a, b⦈.
    unfolding two cf-singleton-ObjMap-app[OF a] prodfunctor-proj-components
    by simp
  finally show ?thesis by simp
```

qed

lemma *bifunctor-proj-snd-ObjMap-app*[*cat-cs-simps*]:
assumes $[a,\ b]_\circ \in_\circ (\mathfrak{A} \times_C \mathfrak{B})(\!|Obj|\!)$
shows $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)(\!|b|\!) = \mathfrak{S}(\!|ObjMap|\!)(\!|a,\ b|\!)_\bullet$
proof–

let *?𝔇* = ‹$\mathfrak{S}(\!|HomCod|\!)$›
let *?𝔖* = ‹$\mathfrak{S}_{\prod_C i \in_\circ 2_{\mathbb{N}} -_\circ set\ \{0\}.(i = 0\ ?\ \mathfrak{A} : \mathfrak{B}),?\mathfrak{D}(-,set\ \{\langle 0,\ a\rangle\})}$›
let *?cfs* = ‹*cf-singleton* $(1_{\mathbb{N}})$ $\mathfrak{B}$›

from *assms* **have** *a*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** *b*: $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
by (*all*‹*elim cat-prod-2-ObjE*[*OF* $\mathfrak{A}$ $\mathfrak{B}$]›) *auto*
from *a* **have** *za*: $set\ \{\langle 0,\ a\rangle\} \in_\circ (\prod_C i \in_\circ set\ \{0\}.\ \mathfrak{A})(\!|Obj|\!)$
by (*intro cat-singleton-ObjI*[**where** *a*=*a*]) *simp*
from *b* **have** *ob*: $set\ \{\langle 1_{\mathbb{N}},\ b\rangle\} \in_\circ (\prod_C i \in_\circ set\ \{1_{\mathbb{N}}\}.\ \mathfrak{B})(\!|Obj|\!)$
by (*intro cat-singleton-ObjI*[**where** *a*=*b*]) *simp*
have[*simp*]: $vinsert\ \langle 1_{\mathbb{N}},\ b\rangle\ (set\ \{\langle 0,\ a\rangle\}) = [a,\ b]_\circ$
using *ord-of-nat-succ-vempty* **unfolding** *vcons-def*
by (*simp add*: *vinsert-vempty*)

have $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)(\!|b|\!) = (?\mathfrak{S}(\!|ObjMap|\!) \circ_\circ ?cfs(\!|ObjMap|\!))(\!|b|\!)$
unfolding *bifunctor-proj-snd-def dghm-comp-components* **by** *simp*
also have $\ldots = ?\mathfrak{S}(\!|ObjMap|\!)(\!|?cfs(\!|ObjMap|\!)(\!|b|\!)|\!)$
by (*rule vsv-vcomp-at*)
(
simp-all add:
two
cf-singleton-components
prodfunctor-proj-components
cf-singleton-ObjMap-app
ob b
)
also from *ob* **have** $\ldots = \mathfrak{S}(\!|ObjMap|\!)(\!|a,\ b|\!)_\bullet$
unfolding *two cf-singleton-ObjMap-app*[*OF b*] *prodfunctor-proj-components*
by *simp*
finally show *?thesis* **by** *simp*

qed

end

8.16.3 Arrow map

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

interpretation *finite-pcategory* α ‹$2_{\mathbb{N}}$› ‹*if2* $\mathfrak{A}$ $\mathfrak{B}$›
by (*intro finite-pcategory-cat-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemmas-with [*OF* $\mathfrak{A}$.*category-axioms* $\mathfrak{B}$.*category-axioms*, *simp*]:
cat-singleton-qm-fst-def **and** *cat-singleton-qm-snd-def*

lemma *bifunctor-proj-fst-ArrMap-app*[*cat-cs-simps*]:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$ **and** $f \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
shows $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{S}(\!|ArrMap|\!)(\!|f,\ \mathfrak{B}(\!|CId|\!)(\!|b|\!)|\!)$.
proof–

let *?*$\mathfrak{D}$ = ‹$\mathfrak{S}(\!|HomCod|\!)$›
let *?*$\mathfrak{S}$ = ‹$\mathfrak{S}_{\prod_C i \in_{\circ} 2_{\mathbb{N}} -_{\circ} set\ \{1_{\mathbb{N}}\}.(i = 0\ ?\ \mathfrak{A} : \mathfrak{B}),?\mathfrak{D}(-,set\ \{\langle 1_{\mathbb{N}},\ b\rangle\})}$›
let *?cfs* = ‹*cf-singleton 0* $\mathfrak{A}$›

from *assms*(*1*) **have** $\mathfrak{B}(\!|CId|\!)(\!|b|\!) : b \mapsto_{\mathfrak{B}} b$ **by** (*auto intro*: *cat-cs-intros*)
then have *CId-b*: $\mathfrak{B}(\!|CId|\!)(\!|b|\!) \in_{\circ} \mathfrak{B}(\!|Arr|\!)$ **by** *auto*

from *assms*(*2*) **have** *zf*: *set* $\{\langle 0,\ f\rangle\} \in_{\circ} (\prod_C i \in_{\circ} set\ \{0\}.\ \mathfrak{A})(\!|Arr|\!)$
by (*intro cat-singleton-ArrI*[**where** *a*=*f*]) *simp*
from *assms*(*1*) **have** *ob*: *set* $\{\langle 1_{\mathbb{N}},\ b\rangle\} \in_{\circ} (\prod_C i \in_{\circ} set\ \{1_{\mathbb{N}}\}.\ \mathfrak{B})(\!|Obj|\!)$
by (*intro cat-singleton-ObjI*[**where** *a*=*b*]) *simp*
have [*simp*]: *vinsert* $\langle 0,\ f\rangle$ (*set* $\{\langle 1_{\mathbb{N}},\ \mathfrak{B}(\!|CId|\!)(\!|b|\!)\rangle\}$) = $[f,\ \mathfrak{B}(\!|CId|\!)(\!|b|\!)]_{\circ}$
using *ord-of-nat-succ-vempty* **unfolding** *vcons-def*
by (*simp add*: *insert-commute ord-of-nat-vone vinsert-vempty vinsert-vsingleton*)

have $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ArrMap|\!)(\!|f|\!) = (?\mathfrak{S}(\!|ArrMap|\!) \circ_{\circ} ?cfs(\!|ArrMap|\!))(\!|f|\!)$
unfolding *bifunctor-proj-fst-def dghm-comp-components* **by** *simp*
also have ... = $?\mathfrak{S}(\!|ArrMap|\!)(\!|?cfs(\!|ArrMap|\!)(\!|f|\!)|\!)$
by (*rule vsv-vcomp-at*)
(
simp-all add:
two
assms(*2*)
cf-singleton-components
prodfunctor-proj-components
cf-singleton-ArrMap-app
zf
)
also from *assms*(*1*) *zf* **have** ... = $\mathfrak{S}(\!|ArrMap|\!)(\!|f,\ \mathfrak{B}(\!|CId|\!)(\!|b|\!)|\!)$.
unfolding *cf-singleton-ArrMap-app*[*OF assms*(*2*)] *prodfunctor-proj-components*
by (*simp add*: *two cat-singleton-CId-app*[*OF ob*])
finally show *?thesis* **by** *simp*

qed

lemma *bifunctor-proj-snd-ArrMap-app*[*cat-cs-simps*]:
assumes $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ **and** $g \in_{\circ} \mathfrak{B}(\!|Arr|\!)$
shows $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{S}(\!|ArrMap|\!)(\!|\mathfrak{A}(\!|CId|\!)(\!|a|\!),\ g|\!)$.
proof–

let *?*$\mathfrak{D}$ = ‹$\mathfrak{S}(\!|HomCod|\!)$›
let *?*$\mathfrak{S}$ = ‹$\mathfrak{S}_{\prod_C i \in_{\circ} 2_{\mathbb{N}} -_{\circ} set\ \{0\}.(i = 0\ ?\ \mathfrak{A} : \mathfrak{B}),?\mathfrak{D}(-,set\ \{\langle 0,\ a\rangle\})}$›
let *?cfs* = ‹*cf-singleton* $(1_{\mathbb{N}})$ $\mathfrak{B}$›

from *assms*(*1*) **have** $\mathfrak{A}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{A}} a$ **by** (*auto intro*: *cat-cs-intros*)
then have *CId-a*: $\mathfrak{A}(\!|CId|\!)(\!|a|\!) \in_{\circ} \mathfrak{A}(\!|Arr|\!)$ **by** *auto*

from *assms*(*2*) **have** *og*: *set* $\{\langle 1_{\mathbb{N}},\ g\rangle\} \in_{\circ} (\prod_C i \in_{\circ} set\ \{1_{\mathbb{N}}\}.\ \mathfrak{B})(\!|Arr|\!)$
by (*intro cat-singleton-ArrI*[**where** *a*=*g*]) *simp*
from *assms*(*1*) **have** *ob*: *set* $\{\langle 0,\ a\rangle\} \in_{\circ} (\prod_C i \in_{\circ} set\ \{0\}.\ \mathfrak{A})(\!|Obj|\!)$
by (*intro cat-singleton-ObjI*[**where** *a*=*a*]) *simp*
have [*simp*]: *vinsert* $\langle 1_{\mathbb{N}},\ g\rangle$ (*set* $\{\langle 0,\ \mathfrak{A}(\!|CId|\!)(\!|a|\!)\rangle\}$) = $[\mathfrak{A}(\!|CId|\!)(\!|a|\!),\ g]_{\circ}$

using *ord-of-nat-succ-vempty* **unfolding** *vcons-def*
by (*simp add*: *vinsert-vempty*)

have $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) = (?\mathfrak{S}(\lfloor ArrMap \rfloor) \circ_\circ ?cfs(\lfloor ArrMap \rfloor))(\lfloor g \rfloor)$
unfolding *two bifunctor-proj-snd-def dghm-comp-components* **by** *simp*
also have $\ldots = ?\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor ?cfs(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \rfloor)$
by (*rule vsv-vcomp-at*)
(
simp-all add:
two
assms(*2*)
cf-singleton-components
prodfunctor-proj-components
cf-singleton-ArrMap-app
og
)
also from *assms*(*1*) *og* **have** $\ldots = \mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{A}(\lfloor CId \rfloor)(\lfloor a \rfloor),\ g \rfloor)_\bullet$
unfolding *cf-singleton-ArrMap-app*[*OF assms*(*2*)] *prodfunctor-proj-components*
by (*simp add*: *two cat-singleton-CId-app*[*OF ob*])
finally show *?thesis* **by** *simp*

qed

end

8.16.4 Bifunctor projections are functors

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

interpretation *finite-pcategory* α ‹$2_{\mathbb{N}}$› ‹*if2* $\mathfrak{A}$ $\mathfrak{B}$›
by (*intro finite-pcategory-cat-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemmas-with [*OF* $\mathfrak{A}$.*category-axioms* $\mathfrak{B}$.*category-axioms*, *simp*]:
cat-singleton-qm-fst-def **and** *cat-singleton-qm-snd-def*

lemma *bifunctor-proj-fst-is-functor*:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
shows $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof–

interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{A} \times_C \mathfrak{B}$› $\mathfrak{D}$ $\mathfrak{S}$ **by** (*rule assms*(*1*))

show *?thesis*
unfolding *bifunctor-proj-fst-def*
proof
(
intro cf-comp-is-functorI[**where** $\mathfrak{B}$=‹$(\prod_C i \in_\circ set\ \{0\}.\ \mathfrak{A})$›],
unfold $\mathfrak{S}$.*cf-HomCod*
)
from *assms*(*2*) **have** *zb*:
$set\ \{\langle 1_{\mathbb{N}},\ b\rangle\} \in_\circ (\prod_C j \in_\circ set\ \{1_{\mathbb{N}}\}.\ if\ j = 0\ then\ \mathfrak{A}\ else\ \mathfrak{B})(\lfloor Obj \rfloor)$
unfolding *cat-prod-components* **by** (*intro vproduct-vsingletonI*) *simp-all*

have *o-zo*: *set* $\{1_{\mathbb{N}}\} \subseteq_{\circ} 2_{\mathbb{N}}$ **by** *clarsimp*
from *pcat-prodfunctor-proj-is-functor*[
folded cat-prod-2-def, **where** *J*=‹*set* $\{1_{\mathbb{N}}\}$›, *OF assms*(*1*) *zb o-zo*
]
show $\mathfrak{S}_{\prod_C i \in_{\circ} 2_{\mathbb{N}} -_{\circ} set\ \{1_{\mathbb{N}}\}.(i\ =\ 0\ ?\ \mathfrak{A}\ :\ \mathfrak{B}),\mathfrak{D}}(-,set\ \{\langle 1_{\mathbb{N}},\ b\rangle\})$:
$(\prod_C i \in_{\circ} set\ \{0\}.\ \mathfrak{A}) \mapsto\mapsto_{C\alpha} \mathfrak{D}$
unfolding *two* **by** *simp*
from *category.cat-cf-singleton-is-functor*[*OF* $\mathfrak{A}$.*category-axioms*, *of 0*] **show**
cf-singleton 0 $\mathfrak{A}$: $\mathfrak{A} \mapsto\mapsto_{C\alpha} (\prod_C i \in_{\circ} set\ \{0\}.\ \mathfrak{A})$
by *force*
qed

qed

lemma *bifunctor-proj-fst-is-functor'*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$ **and** $\mathfrak{A}' = \mathfrak{A}$
shows $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1,2*) **unfolding** *assms*(*3*) **by** (*rule bifunctor-proj-fst-is-functor*)

lemma *bifunctor-proj-fst-ObjMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows *vsv* $((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ObjMap|\!))$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$›
by (*rule bifunctor-proj-fst-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ObjMap-vsv*)
qed

lemma *bifunctor-proj-fst-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ObjMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$›
by (*rule bifunctor-proj-fst-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ObjMap-vdomain*)
qed

lemma *bifunctor-proj-fst-ArrMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows *vsv* $((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ArrMap|\!))$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$›
by (*rule bifunctor-proj-fst-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ArrMap-vsv*)
qed

lemma *bifunctor-proj-fst-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$›
by (*rule bifunctor-proj-fst-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ArrMap-vdomain*)
qed

lemma *bifunctor-proj-snd-is-functor*:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$

shows $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof–

interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{A} \times_C \mathfrak{B}$› $\mathfrak{D}$ $\mathfrak{S}$ **by** (*rule assms*(*1*))

show *?thesis*
unfolding *bifunctor-proj-snd-def*
proof
(
intro cf-comp-is-functorI[**where** $\mathfrak{B}$=‹$(\prod_C i \in_\circ set\ \{1_\mathbb{N}\}.\ \mathfrak{B})$›],
unfold $\mathfrak{S}$.*cf-HomCod*
)
from *assms*(*2*) **have** *zb*:
set $\{\langle 0, a\rangle\} \in_\circ (\prod_C j \in_\circ set\ \{0\}.\ if\ j = 0\ then\ \mathfrak{A}\ else\ \mathfrak{B})(\!|Obj|\!)$
unfolding *cat-prod-components* **by** (*intro vproduct-vsingletonI*) *simp-all*
have *o-zo*: *set* $\{0\} \subseteq_\circ 2_\mathbb{N}$ **by** *clarsimp*
from
pcat-prodfunctor-proj-is-functor[
folded cat-prod-2-def, **where** *J*=‹*set* $\{0\}$›, *OF assms*(*1*) *zb o-zo*
]
show $\mathfrak{S}_{\prod_C i \in_\circ 2_\mathbb{N} -_\circ set\ \{0\}.(i\ =\ 0\ ?\ \mathfrak{A}\ :\ \mathfrak{B}),\mathfrak{D}}(-, set\ \{\langle 0, a\rangle\}) :$
$(\prod_C i \in_\circ set\ \{1_\mathbb{N}\}.\ \mathfrak{B}) \mapsto\mapsto_{C\alpha} \mathfrak{D}$
unfolding *two* **by** *simp*
from *category.cat-cf-singleton-is-functor*[*OF* $\mathfrak{B}$.*category-axioms*, *of* ‹$1_\mathbb{N}$›]
show *cf-singleton* $(1_\mathbb{N})$ $\mathfrak{B} : \mathfrak{B} \mapsto\mapsto_{C\alpha} (\prod_C i \in_\circ set\ \{1_\mathbb{N}\}.\ \mathfrak{B})$
by *force*
qed

qed

lemma *bifunctor-proj-snd-is-functor'*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1*,*2*) **unfolding** *assms*(*3*) **by** (*rule bifunctor-proj-snd-is-functor*)

lemma *bifunctor-proj-snd-ObjMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows *vsv* $((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!))$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$›
by (*rule bifunctor-proj-snd-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ObjMap-vsv*)
qed

lemma *bifunctor-proj-snd-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\mathcal{D}_\circ ((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)) = \mathfrak{B}(\!|Obj|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$›
by (*rule bifunctor-proj-snd-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ObjMap-vdomain*)
qed

lemma *bifunctor-proj-snd-ArrMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows *vsv* $((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ArrMap|\!))$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$›

by (*rule bifunctor-proj-snd-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ArrMap-vsv*)
qed

lemma *bifunctor-proj-snd-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\mathcal{D}_\circ\ ((\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ArrMap|\!)) = \mathfrak{B}(\!|Arr|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{D}$ ‹$\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$›
by (*rule bifunctor-proj-snd-is-functor*[*OF assms*])
show *?thesis* **by** (*rule* $\mathfrak{S}$.*cf-ArrMap-vdomain*)
qed

end

8.17 Bifunctor flip

8.17.1 Definition and elementary properties

definition *bifunctor-flip* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ =
[*fflip* ($\mathfrak{F}(\!|ObjMap|\!)$), *fflip* ($\mathfrak{F}(\!|ArrMap|\!)$), $\mathfrak{B} \times_C \mathfrak{A}$, $\mathfrak{F}(\!|HomCod|\!)$]$_\circ$

Components

lemma *bifunctor-flip-components*:
shows *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ObjMap|\!)$ = *fflip* ($\mathfrak{F}(\!|ObjMap|\!)$)
and *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ArrMap|\!)$ = *fflip* ($\mathfrak{F}(\!|ArrMap|\!)$)
and *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|HomDom|\!) = \mathfrak{B} \times_C \mathfrak{A}$
and *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|HomCod|\!) = \mathfrak{F}(\!|HomCod|\!)$
unfolding *bifunctor-flip-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

8.17.2 Bifunctor flip object map

lemma *bifunctor-flip-ObjMap-vsv*[*cat-cs-intros*]:
vsv (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ObjMap|\!)$)
unfolding *bifunctor-flip-components* **by** (*rule fflip-vsv*)

lemma *bifunctor-flip-ObjMap-app*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|b, a|\!)_\bullet = \mathfrak{F}(\!|ObjMap|\!)(\!|a, b|\!)_\bullet$
using *assms*
unfolding *bifunctor-flip-components assms*(*4*,*5*)
by
(
cs-concl **cs-shallow**
cs-simp: *V-cs-simps cat-cs-simps* **cs-intro**: *cat-prod-cs-intros*
)

lemma *bifunctor-flip-ObjMap-app'*[*cat-cs-simps*]:
assumes *ba* = [*b*, *a*]$_\circ$
and *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

```
    and a ∈∘ 𝔄(|Obj|)
    and b ∈∘ 𝔅(|Obj|)
  shows bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|ba|) = 𝔉(|ObjMap|)(|a, b|)•
  using assms(2–6) unfolding assms(1) by (rule bifunctor-flip-ObjMap-app)

lemma bifunctor-flip-ObjMap-vdomain[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔉 : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
  shows 𝒟∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) = (𝔅 ×_C 𝔄)(|Obj|)
  using assms
  unfolding bifunctor-flip-components
  by (cs-concl cs-shallow cs-simp: V-cs-simps cat-cs-simps)

lemma bifunctor-flip-ObjMap-vrange[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔉 : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
  shows ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) = ℛ∘ (𝔉(|ObjMap|))
proof–

  interpret 𝔉: is-functor α ‹𝔄 ×_C 𝔅› ℭ 𝔉 by (rule assms(3))

  show ?thesis
  proof(intro vsubset-antisym)

    show ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) ⊆∘ ℛ∘ (𝔉(|ObjMap|))
    proof
      (
        intro vsv.vsv-vrange-vsubset,
        unfold bifunctor-flip-ObjMap-vdomain[OF assms]
      )
      fix ba assume ba ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
      then obtain a b
        where ba-def: ba = [b, a]∘
          and b: b ∈∘ 𝔅(|Obj|)
          and a: a ∈∘ 𝔄(|Obj|)
        by (elim cat-prod-2-ObjE[OF assms(2,1)])
      from assms a b show
        bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|ba|) ∈∘ ℛ∘ (𝔉(|ObjMap|))
        unfolding ba-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-prod-cs-intros
          )
    qed (auto intro: cat-cs-intros)

    show ℛ∘ (𝔉(|ObjMap|)) ⊆∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|))
    proof(intro vsv.vsv-vrange-vsubset, unfold 𝔉.cf-ObjMap-vdomain)
      fix ab assume prems: ab ∈∘ (𝔄 ×_C 𝔅)(|Obj|)
      then obtain a b
        where ab-def: ab = [a, b]∘
          and a: a ∈∘ 𝔄(|Obj|)
          and b: b ∈∘ 𝔅(|Obj|)
        by (elim cat-prod-2-ObjE[OF assms(1,2)])
      from assms a b have ba: [b, a]∘ ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
        by (cs-concl cs-shallow cs-intro: cat-prod-cs-intros)
```

from *assms bifunctor-flip-ObjMap-vsv prems a b ba* **show**
$\mathfrak{F}(\!|ObjMap|\!)(\!|ab|\!) \in_{\circ} \mathcal{R}_{\circ}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ObjMap|\!)$)
by
(
cs-concl **cs-shallow**
cs-simp: *ab-def cat-cs-simps* **cs-intro**: *V-cs-intros*
)
qed *auto*

qed

qed

8.17.3 Bifunctor flip arrow map

lemma *bifunctor-flip-ArrMap-vsv*[*cat-cs-intros*]:
vsv (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ArrMap|\!)$)
unfolding *bifunctor-flip-components* **by** (*rule fflip-vsv*)

lemma *bifunctor-flip-ArrMap-app*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $g \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
and $f \in_{\circ} \mathfrak{B}(\!|Arr|\!)$
shows *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ArrMap|\!)(\!|f, g|\!)_{\bullet} = \mathfrak{F}(\!|ArrMap|\!)(\!|g, f|\!)_{\bullet}$
using *assms*
unfolding *bifunctor-flip-components*
by
(
cs-concl **cs-shallow**
cs-simp: *V-cs-simps cat-cs-simps* **cs-intro**: *cat-prod-cs-intros*
)

lemma *bifunctor-flip-ArrMap-app'*[*cat-cs-simps*]:
assumes $fg = [f, g]_{\circ}$
and *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $g \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
and $f \in_{\circ} \mathfrak{B}(\!|Arr|\!)$
shows *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ArrMap|\!)(\!|fg|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g, f|\!)_{\bullet}$
using *assms*(*2–6*) **unfolding** *assms*(*1*) **by** (*rule bifunctor-flip-ArrMap-app*)

lemma *bifunctor-flip-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_{\circ}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ArrMap|\!)$) = $(\mathfrak{B} \times_C \mathfrak{A})(\!|Arr|\!)$
using *assms*
unfolding *bifunctor-flip-components*
by (*cs-concl* **cs-shallow cs-simp**: *V-cs-simps cat-cs-simps*)

lemma *bifunctor-flip-ArrMap-vrange*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\!|ArrMap|\!)$) = $\mathcal{R}_{\circ}$ ($\mathfrak{F}(\!|ArrMap|\!)$)

```
proof–

  interpret 𝔉: is-functor α ‹𝔄 ×_C 𝔅› ℭ 𝔉 by (rule assms(3))

  show ?thesis
  proof(intro vsubset-antisym)

    show ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) ⊆∘ ℛ∘ (𝔉(|ArrMap|))
    proof
      (
        intro vsv.vsv-vrange-vsubset,
        unfold bifunctor-flip-ArrMap-vdomain[OF assms]
      )
      fix fg assume fg ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
      then obtain f g
        where fg-def: fg = [f, g]∘
          and f: f ∈∘ 𝔅(|Arr|)
          and g: g ∈∘ 𝔄(|Arr|)
        by (elim cat-prod-2-ArrE[OF assms(2,1)])
      from f obtain a b where f: f : a ↦_𝔅 b by (auto intro: is-arrI)
      from g obtain a' b' where g: g : a' ↦_𝔄 b' by (auto intro: is-arrI)
      from 𝔉.cf-ArrMap-vsv assms f g show
        bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|fg|) ∈∘ ℛ∘ (𝔉(|ArrMap|))
        unfolding fg-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps
              cs-intro: V-cs-intros cat-cs-intros cat-prod-cs-intros
          )
    qed (auto intro: cat-cs-intros)

    show ℛ∘ (𝔉(|ArrMap|)) ⊆∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|))
    proof(intro vsv.vsv-vrange-vsubset, unfold 𝔉.cf-ArrMap-vdomain)
      fix gf assume prems: gf ∈∘ (𝔄 ×_C 𝔅)(|Arr|)
      then obtain g f
        where gf-def: gf = [g, f]∘
          and g: g ∈∘ 𝔄(|Arr|)
          and f: f ∈∘ 𝔅(|Arr|)
        by (elim cat-prod-2-ArrE[OF assms(1,2)])
      from assms g f have fg: [f, g]∘ ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
        by (cs-concl cs-shallow cs-intro: cat-prod-cs-intros)
      from assms bifunctor-flip-ArrMap-vsv prems g f fg show
        𝔉(|ArrMap|)(|gf|) ∈∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|))
        unfolding gf-def
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: V-cs-intros)
    qed auto

  qed

qed
```

8.17.4 Bifunctor flip is a bifunctor

```
lemma bifunctor-flip-is-functor:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔉 : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
```

```
  shows bifunctor-flip 𝔄 𝔅 𝔉 : 𝔅 ×_C 𝔄 ↦↦_Cα ℭ
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔅: category α 𝔅 by (rule assms(2))
  interpret 𝔉: is-functor α ‹𝔄 ×_C 𝔅› ℭ 𝔉 by (rule assms)

  show ?thesis
  proof(intro is-functorI')
    show vfsequence (bifunctor-flip 𝔄 𝔅 𝔉)
      unfolding bifunctor-flip-def by simp
    from assms(1,2) show category α (𝔅 ×_C 𝔄)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show vcard (bifunctor-flip 𝔄 𝔅 𝔉) = ℶ_ℕ
      unfolding bifunctor-flip-def by (simp add: nat-omega-simps)
    show vsv (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) by (auto intro: cat-cs-intros)
    show vsv (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) by (auto intro: cat-cs-intros)
    from assms show 𝒟_∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) = (𝔅 ×_C 𝔄)(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from assms 𝔉.cf-ObjMap-vrange show
      ℛ_∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) ⊆_∘ ℭ(|Obj|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from assms show 𝒟_∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) = (𝔅 ×_C 𝔄)(|Arr|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    show bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|gf|) :
      bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|ba|) ↦_ℭ
      bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|b'a'|)
      if gf : ba ↦_𝔅 ×_C 𝔄 b'a' for ba b'a' gf
    proof–
      from that obtain g f a b a' b'
        where gf-def: gf = [g, f]_∘
          and ba-def: ba = [b, a]_∘
          and b'a'-def: b'a' = [b', a']_∘
          and g: g : b ↦_𝔅 b'
          and f: f : a ↦_𝔄 a'
        by (elim cat-prod-2-is-arrE[OF assms(2,1)])
      from assms g f show ?thesis
        unfolding gf-def ba-def b'a'-def
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
          )
    qed
    show
      bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|gg' ∘_A𝔅 ×_C 𝔄 ff'|) =
        bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|gg'|) ∘_Aℭ
        bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|ff'|)
      if gg': gg' : bb' ↦_𝔅 ×_C 𝔄 cc' and ff': ff' : aa' ↦_𝔅 ×_C 𝔄 bb'
      for bb' cc' gg' aa' ff'
    proof–
      obtain g g' b b' c c'
        where gg'-def: gg' = [g, g']_∘
          and bb'-def: bb' = [b, b']_∘
          and cc'-def: cc' = [c, c']_∘
          and g: g : b ↦_𝔅 c
          and g': g' : b' ↦_𝔄 c'
        by (elim cat-prod-2-is-arrE[OF assms(2,1) gg'])
```

```
    moreover obtain f f' a a' b'' b'''
      where ff'-def: ff' = [f, f']∘
        and aa'-def: aa' = [a, a']∘
        and bb' = [b'', b''']∘
        and f : a ↦𝔅 b''
        and f' : a' ↦𝔄 b'''
      by (elim cat-prod-2-is-arrE[OF assms(2,1) ff'])
    ultimately have f: f : a ↦𝔅 b and f': f' : a' ↦𝔄 b'
      by (auto simp: cat-op-simps)
    from assms g g' f f' have [cat-cs-simps]:
      𝔉(|ArrMap|)(|g' ∘A𝔄 f', g ∘A𝔅 f|)• =
        𝔉(|ArrMap|)(|[g', g]∘ ∘A𝔄 ×C 𝔅 [f', f]∘|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-prod-2-Comp-app cs-intro: cat-prod-cs-intros
        )
    from assms g g' f f' show
      bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|gg' ∘A𝔅 ×C 𝔄 ff'|) =
        bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|gg'|) ∘A𝔈
        bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|ff'|)
      unfolding gg'-def ff'-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-prod-cs-simps cat-cs-simps
            cs-intro: cat-prod-cs-intros cat-cs-intros
        )
  qed
  show
    bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|(𝔅 ×C 𝔄)(|CId|)(|ba|)|) =
      𝔈(|CId|)(|bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|ba|)|)
    if ba ∈∘ (𝔅 ×C 𝔄)(|Obj|) for ba
  proof–
    from that obtain b a
      where ba-def: ba = [b, a]∘
        and b: b ∈∘ 𝔅(|Obj|)
        and a: a ∈∘ 𝔄(|Obj|)
      by (elim cat-prod-2-ObjE[rotated 2]) (auto intro: cat-cs-intros)
    from assms b a have [cat-cs-simps]:
      𝔉(|ArrMap|)(|𝔄(|CId|)(|a|), 𝔅(|CId|)(|b|)|)• =
        𝔉(|ArrMap|)(|(𝔄 ×C 𝔅)(|CId|)(|a, b|)•|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-prod-2-CId-app cs-intro: cat-prod-cs-intros
        )
    from assms b a show ?thesis
      unfolding ba-def
      by
        (
          cs-concl cs-shallow
            cs-intro: cat-cs-intros cat-prod-cs-intros
            cs-simp: cat-prod-cs-simps cat-cs-simps
        )
  qed
qed (auto simp: bifunctor-flip-components cat-cs-simps cat-cs-intros)
```

qed

lemma *bifunctor-flip-is-functor'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{D} = \mathfrak{B} \times_C \mathfrak{A}$
shows *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*intro bifunctor-flip-is-functor*)

8.17.5 Double-flip of a bifunctor

lemma *bifunctor-flip-flip*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *bifunctor-flip* $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$) = $\mathfrak{F}$
proof(*rule cf-eqI*)

interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-functor* α ‹$\mathfrak{A} \times_C \mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*3*))

from *assms* **show**
bifunctor-flip $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$) : $\mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

from *assms* **have** *ObjMap-dom-lhs*:
$\mathcal{D}_\circ$ (*bifunctor-flip* $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)(⦇*ObjMap*⦈)) =
($\mathfrak{A} \times_C \mathfrak{B}$)(⦇*Obj*⦈)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
have *ObjMap-dom-rhs*: $\mathcal{D}_\circ$ ($\mathfrak{F}$(⦇*ObjMap*⦈)) = ($\mathfrak{A} \times_C \mathfrak{B}$)(⦇*Obj*⦈)
by (*simp add*: *cat-cs-simps*)
from *assms* **have** *ArrMap-dom-lhs*:
$\mathcal{D}_\circ$ (*bifunctor-flip* $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)(⦇*ArrMap*⦈)) =
($\mathfrak{A} \times_C \mathfrak{B}$)(⦇*Arr*⦈)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
have *ArrMap-dom-rhs*: $\mathcal{D}_\circ$ ($\mathfrak{F}$(⦇*ArrMap*⦈)) = ($\mathfrak{A} \times_C \mathfrak{B}$)(⦇*Arr*⦈)
by (*simp add*: *cat-cs-simps*)

show *bifunctor-flip* $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)(⦇*ObjMap*⦈) = $\mathfrak{F}$(⦇*ObjMap*⦈)
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
fix *ab* **assume** *ab* $\in_\circ$ ($\mathfrak{A} \times_C \mathfrak{B}$)(⦇*Obj*⦈)
then obtain *a b*
where *ab-def*: *ab* = [*a*, *b*]$_\circ$ **and** *a*: *a* $\in_\circ$ $\mathfrak{A}$(⦇*Obj*⦈) **and** *b*: *b* $\in_\circ$ $\mathfrak{B}$(⦇*Obj*⦈)
by (*rule cat-prod-2-ObjE*[*OF assms*(*1,2*)])
from *assms a b* **show**
bifunctor-flip $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)(⦇*ObjMap*⦈)(⦇*ab*⦈) = $\mathfrak{F}$(⦇*ObjMap*⦈)(⦇*ab*⦈)
unfolding *ab-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto simp*: *cat-cs-intros*)

show *bifunctor-flip* $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)(⦇*ArrMap*⦈) = $\mathfrak{F}$(⦇*ArrMap*⦈)
proof(*rule vsv-eqI*, *unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
fix *ab* **assume** *ab* $\in_\circ$ ($\mathfrak{A} \times_C \mathfrak{B}$)(⦇*Arr*⦈)
then obtain *a b*
where *ab-def*: *ab* = [*a*, *b*]$_\circ$ **and** *a*: *a* $\in_\circ$ $\mathfrak{A}$(⦇*Arr*⦈) **and** *b*: *b* $\in_\circ$ $\mathfrak{B}$(⦇*Arr*⦈)
by (*rule cat-prod-2-ArrE*[*OF assms*(*1,2*)])

from *assms a b* **show**
bifunctor-flip $\mathfrak{B}$ $\mathfrak{A}$ (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)(⌊*ArrMap*⌋)(⌊*ab*⌋) = $\mathfrak{F}$(⌊*ArrMap*⌋)(⌊*ab*⌋)
unfolding *ab-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto simp*: *cat-cs-intros*)

qed (*simp-all add*: *assms*(*3*))

8.17.6 A projection of a bifunctor flip

lemma *bifunctor-flip-proj-snd*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}$(⌊*Obj*⌋)
shows *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF} = \mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
proof(*rule cf-eqI*)

from *assms* **show** *f-Fb*: *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **show** $\mathfrak{F}b$: $\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

from *assms* **have** *ObjMap-dom-lhs*:
$\mathcal{D}_\circ$ ((*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ObjMap*⌋)) = $\mathfrak{A}$(⌊*Obj*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms* **have** *ObjMap-dom-rhs*: $\mathcal{D}_\circ$ (($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ObjMap*⌋)) = $\mathfrak{A}$(⌊*Obj*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *assms* **have** *ArrMap-dom-lhs*:
$\mathcal{D}_\circ$ ((*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ArrMap*⌋)) = $\mathfrak{A}$(⌊*Arr*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms* **have** *ArrMap-dom-rhs*: $\mathcal{D}_\circ$ (($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ArrMap*⌋)) = $\mathfrak{A}$(⌊*Arr*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)

show (*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ObjMap*⌋) = ($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ObjMap*⌋)
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
from *assms* **show** *vsv* ((*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ObjMap*⌋))
by (*intro bifunctor-proj-snd-ObjMap-vsv*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **show** *vsv* (($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ObjMap*⌋))
by (*intro bifunctor-proj-fst-ObjMap-vsv*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
fix *a* **assume** $a \in_\circ \mathfrak{A}$(⌊*Obj*⌋)
with *assms* **show**
(*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ObjMap*⌋)(⌊*a*⌋) =
($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ObjMap*⌋)(⌊*a*⌋)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-prod-cs-intros*)
qed *simp*

show
(*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ArrMap*⌋) = ($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ArrMap*⌋)
proof(*rule vsv-eqI*, *unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
from *assms* **show** *vsv* ((*bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}_{\mathfrak{B},\mathfrak{A}}(b,-)_{CF}$)(⌊*ArrMap*⌋))
by (*intro bifunctor-proj-snd-ArrMap-vsv*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **show** *vsv* (($\mathfrak{F}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$)(⌊*ArrMap*⌋))
by (*intro bifunctor-proj-fst-ArrMap-vsv*)

```
      (cs-concl cs-shallow cs-intro: cat-cs-intros)
    fix f assume f ∈₀ 𝔄(|Arr|)
    with assms show
      (bifunctor-flip 𝔄 𝔅 𝔉_{𝔅,𝔄}(b,−)_{CF})(|ArrMap|)(|f|) =
        (𝔉_{𝔄,𝔅}(−,b)_{CF})(|ArrMap|)(|f|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  qed simp

qed simp-all

lemma bifunctor-flip-proj-fst[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔉 : 𝔄 ×_C 𝔅 ↦↦_{Cα} ℭ
    and a ∈₀ 𝔄(|Obj|)
  shows bifunctor-flip 𝔄 𝔅 𝔉_{𝔅,𝔄}(−,a)_{CF} = 𝔉_{𝔄,𝔅}(a,−)_{CF}
proof–
  from assms have f-𝔉: bifunctor-flip 𝔄 𝔅 𝔉 : 𝔅 ×_C 𝔄 ↦↦_{Cα} ℭ
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ?thesis
    by
      (
        rule
          bifunctor-flip-proj-snd
            [
              OF assms(2,1) f-𝔉 assms(4),
              unfolded bifunctor-flip-flip[OF assms(1,2,3)],
              symmetric
            ]
      )
qed
```

8.17.7 A flip of a bifunctor isomorphism

```
lemma bifunctor-flip-is-iso-functor:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔉 : 𝔄 ×_C 𝔅 ↦↦_{C.iso α} ℭ
  shows bifunctor-flip 𝔄 𝔅 𝔉 : 𝔅 ×_C 𝔄 ↦↦_{C.iso α} ℭ
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔅: category α 𝔅 by (rule assms(2))
  interpret 𝔉: is-iso-functor α ‹𝔄 ×_C 𝔅› ℭ 𝔉 by (rule assms(3))

  from assms have f-𝔉: bifunctor-flip 𝔄 𝔅 𝔉 : 𝔅 ×_C 𝔄 ↦↦_{Cα} ℭ
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)

  from f-𝔉 have ObjMap-dom:
    𝒟₀ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) = (𝔅 ×_C 𝔄)(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps)
  from f-𝔉 have ArrMap-dom:
    𝒟₀ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) = (𝔅 ×_C 𝔄)(|Arr|)
    by (cs-concl cs-simp: cat-cs-simps)

  show ?thesis
  proof(intro is-iso-functorI' vsv.vsv-valeq-v11I, unfold ObjMap-dom ArrMap-dom)
    from assms show bifunctor-flip 𝔄 𝔅 𝔉 : 𝔅 ×_C 𝔄 ↦↦_{Cα} ℭ
```

```
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  fix ba b'a'
  assume prems:
    ba ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
    b'a' ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
    bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|ba|) = bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|b'a'|)
  from prems(1) obtain b a
    where ba-def: ba = [b, a]∘
      and b: b ∈∘ 𝔅(|Obj|)
      and a: a ∈∘ 𝔄(|Obj|)
    by (elim cat-prod-2-ObjE[OF assms(2,1)])
  from prems(2) obtain a' b'
    where b'a'-def: b'a' = [b', a']∘
      and b': b' ∈∘ 𝔅(|Obj|)
      and a': a' ∈∘ 𝔄(|Obj|)
    by (rule cat-prod-2-ObjE[OF assms(2,1)])
  from prems(3) assms a b b' a' have 𝔉ab-𝔉a'b':
    𝔉(|ObjMap|)(|a, b|)• = 𝔉(|ObjMap|)(|a', b'|)•
    unfolding ba-def b'a'-def
    by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cf-cs-intros)
  from assms a b a' b' have [a, b]∘ = [a', b']∘
    by
      (
        cs-concl cs-shallow
          cs-intro:
            𝔉.ObjMap.v11-eq-iff[THEN iffD1, OF - - 𝔉ab-𝔉a'b']
            cat-prod-cs-intros
      )
  then show ba = b'a' unfolding ba-def b'a'-def by simp
next
  fix fg f'g' assume prems:
    fg ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
    f'g' ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
    bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|fg|) = bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|f'g'|)
  from prems(1) obtain f g
    where fg-def: fg = [f, g]∘
      and f: f ∈∘ 𝔅(|Arr|)
      and g: g ∈∘ 𝔄(|Arr|)
    by (elim cat-prod-2-ArrE[OF assms(2,1)])
  from prems(2) obtain f' g'
    where f'g'-def: f'g' = [f', g']∘
      and f': f' ∈∘ 𝔅(|Arr|)
      and g': g' ∈∘ 𝔄(|Arr|)
    by (rule cat-prod-2-ArrE[OF assms(2,1)])
  from prems(3) assms f g f' g' have 𝔉gf-𝔉g'f':
    𝔉(|ArrMap|)(|g, f|)• = 𝔉(|ArrMap|)(|g', f'|)•
    unfolding fg-def f'g'-def
    by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cf-cs-intros)
  from assms g f g' f' have [g, f]∘ = [g', f']∘
    by
      (
        cs-concl cs-shallow
          cs-intro:
            𝔉.ArrMap.v11-eq-iff[THEN iffD1, OF - - 𝔉gf-𝔉g'f']
            cat-prod-cs-intros
      )
  then show fg = f'g' unfolding fg-def f'g'-def by simp
next
```

```
show ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) = ℭ(|Obj|)
proof(rule vsubset-antisym)
  show ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)) ⊆∘ ℭ(|Obj|)
  proof(rule vsv.vsv-vrange-vsubset, unfold ObjMap-dom)
    fix ba assume ba ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
    then obtain b a
      where ba-def: ba = [b, a]∘
        and b: b ∈∘ 𝔅(|Obj|)
        and a: a ∈∘ 𝔄(|Obj|)
      by (elim cat-prod-2-ObjE[OF assms(2,1)])
    from assms b a show bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|ba|) ∈∘ ℭ(|Obj|)
      unfolding ba-def
      by (cs-concl cs-intro: cat-cs-intros cf-cs-intros cat-prod-cs-intros)
  qed (auto simp: cat-cs-intros)
  show ℭ(|Obj|) ⊆∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|))
  proof(intro vsubsetI)
    fix c assume prems: c ∈∘ ℭ(|Obj|)
    from prems obtain ab
      where ab: ab ∈∘ (𝔄 ×_C 𝔅)(|Obj|) and 𝔉ab: 𝔉(|ObjMap|)(|ab|) = c
      by blast
    from ab obtain b a
      where ab-def: ab = [a, b]∘
        and a: a ∈∘ 𝔄(|Obj|)
        and b: b ∈∘ 𝔅(|Obj|)
      by (elim cat-prod-2-ObjE[OF assms(1,2)])
    show c ∈∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|))
    proof(intro vsv.vsv-vimageI2', unfold ObjMap-dom)
      from assms a b show [b, a]∘ ∈∘ (𝔅 ×_C 𝔄)(|Obj|)
        by (cs-concl cs-shallow cs-intro: cat-prod-cs-intros)
      from assms b a prems show c = bifunctor-flip 𝔄 𝔅 𝔉(|ObjMap|)(|b, a|)∙
        by
          (
            cs-concl cs-shallow
              cs-simp: 𝔉ab[unfolded ab-def] cat-cs-simps
              cs-intro: cf-cs-intros
          )
    qed (auto intro: cat-cs-intros)
  qed
qed

show ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) = ℭ(|Arr|)
proof(rule vsubset-antisym)
  show ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) ⊆∘ ℭ(|Arr|)
  proof(rule vsv.vsv-vrange-vsubset, unfold ArrMap-dom)
    show vsv (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)) by (auto intro: cat-cs-intros)
    fix fg assume fg ∈∘ (𝔅 ×_C 𝔄)(|Arr|)
    then obtain f g
      where fg-def: fg = [f, g]∘
        and f: f ∈∘ 𝔅(|Arr|)
        and g: g ∈∘ 𝔄(|Arr|)
      by (elim cat-prod-2-ArrE[OF assms(2,1)])
    from g f obtain a b a' b'
      where f: f : a ↦_𝔅 b and g: g : a' ↦_𝔄 b'
      by (auto intro!: is-arrI)
    from assms f g show bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|fg|) ∈∘ ℭ(|Arr|)
      by
        (
```

```
          cs-concl cs-shallow
            cs-simp: fg-def cs-intro: cat-cs-intros cat-prod-cs-intros
        )
    qed
    show 𝔈(|Arr|) ⊆∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|))
    proof(intro vsubsetI)
      fix c assume prems: c ∈∘ 𝔈(|Arr|)
      from prems obtain ab
        where ab: ab ∈∘ (𝔄 ×C 𝔅)(|Arr|) and 𝔉ab: 𝔉(|ArrMap|)(|ab|) = c
        by blast
      from ab obtain b a
        where ab-def: ab = [a, b]∘
          and a: a ∈∘ 𝔄(|Arr|)
          and b: b ∈∘ 𝔅(|Arr|)
        by (elim cat-prod-2-ArrE[OF assms(1,2)])
      show c ∈∘ ℛ∘ (bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|))
      proof(intro vsv.vsv-vimageI2', unfold ArrMap-dom)
        from assms a b show [b, a]∘ ∈∘ (𝔅 ×C 𝔄)(|Arr|)
          by (cs-concl cs-shallow cs-intro: cat-prod-cs-intros)
        from assms b a prems show c = bifunctor-flip 𝔄 𝔅 𝔉(|ArrMap|)(|b, a|)∘
          by
            (
              cs-concl cs-shallow
                cs-simp: 𝔉ab[unfolded ab-def] cat-cs-simps
                cs-intro: cat-cs-intros
            )
      qed (auto intro: cat-cs-intros)
    qed
  qed

  qed (auto intro: cat-cs-intros)

qed
```

8.18 Array bifunctor

8.18.1 Definition and elementary properties

See Chapter II-3 in [7].

```
definition cf-array :: V ⇒ V ⇒ V ⇒ (V ⇒ V) ⇒ (V ⇒ V) ⇒ V
  where cf-array 𝔅 𝔈 𝔇 𝔉 𝔊 =
    [
      (λa∈∘(𝔅 ×C 𝔈)(|Obj|). 𝔊 (vpfst a)(|ObjMap|)(|vpsnd a|)),
      (
        λf∈∘(𝔅 ×C 𝔈)(|Arr|).
          𝔊 (𝔅(|Cod|)(|vpfst f|))(|ArrMap|)(|vpsnd f|) ∘A𝔇
          𝔉 (𝔈(|Dom|)(|vpsnd f|))(|ArrMap|)(|vpfst f|)
      ),
      𝔅 ×C 𝔈,
      𝔇
    ]∘
```

Components.

```
lemma cf-array-components:
  shows cf-array 𝔅 𝔈 𝔇 𝔉 𝔊(|ObjMap|) =
    (λa∈∘(𝔅 ×C 𝔈)(|Obj|). 𝔊 (vpfst a)(|ObjMap|)(|vpsnd a|))
    and cf-array 𝔅 𝔈 𝔇 𝔉 𝔊(|ArrMap|) =
      (
```

$\lambda f \in_\circ (\mathfrak{B} \times_C \mathfrak{C})(\lfloor Arr \rfloor).$
$\mathfrak{G}\ (\mathfrak{B}(\lfloor Cod \rfloor)(\lfloor vpfst\ f \rfloor))(\lfloor ArrMap \rfloor)(\lfloor vpsnd\ f \rfloor) \circ_{A\mathfrak{D}}$
$\mathfrak{F}\ (\mathfrak{C}(\lfloor Dom \rfloor)(\lfloor vpsnd\ f \rfloor))(\lfloor ArrMap \rfloor)(\lfloor vpfst\ f \rfloor)$
)
and *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor HomDom \rfloor) = \mathfrak{B} \times_C \mathfrak{C}$
and *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor HomCod \rfloor) = \mathfrak{D}$
unfolding *cf-array-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

8.18.2 Object map

lemma *cf-array-ObjMap-vsv*: *vsv* (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ObjMap \rfloor)$)
unfolding *cf-array-components* **by** *simp*

lemma *cf-array-ObjMap-vdomain*[*cat-cs-simps*]:
$\mathcal{D}_\circ$ (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ObjMap \rfloor)$) = $(\mathfrak{B} \times_C \mathfrak{C})(\lfloor Obj \rfloor)$
unfolding *cf-array-components* **by** *simp*

lemma *cf-array-ObjMap-app*[*cat-cs-simps*]:
assumes $[b, c]_\circ \in_\circ (\mathfrak{B} \times_C \mathfrak{C})(\lfloor Obj \rfloor)$
shows *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor b, c \rfloor)_\bullet = \mathfrak{G}\ b(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)$
using *assms* **unfolding** *cf-array-components* **by** (*simp add*: *nat-omega-simps*)

lemma *cf-array-ObjMap-vrange*:
assumes *category* $\alpha\ \mathfrak{B}$
and *category* $\alpha\ \mathfrak{C}$
and $\bigwedge b.\ b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor) \Longrightarrow \mathfrak{G}\ b : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ObjMap \rfloor)$) $\subseteq_\circ \mathfrak{D}(\lfloor Obj \rfloor)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-array-ObjMap-vdomain*)
show *vsv* (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ObjMap \rfloor)$) **by** (*rule cf-array-ObjMap-vsv*)
fix x **assume** *prems*: $x \in_\circ (\mathfrak{B} \times_C \mathfrak{C})(\lfloor Obj \rfloor)$
then obtain $b\ c$ **where** *x-def*: $x = [b, c]_\circ$
and *b*: $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
and *c*: $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
by (*elim cat-prod-2-ObjE*[*OF assms*(*1,2*)])
interpret $\mathfrak{G}b$: *is-functor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \langle\mathfrak{G}\ b\rangle$ **by** (*rule assms*(*3*)[*OF b*])
from *prems c* **show** *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) \in_\circ \mathfrak{D}(\lfloor Obj \rfloor)$
unfolding *x-def cf-array-components*
by (*auto simp*: *nat-omega-simps cat-cs-intros*)
qed

8.18.3 Arrow map

lemma *cf-array-ArrMap-vsv*: *vsv* (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ArrMap \rfloor)$)
unfolding *cf-array-components* **by** *simp*

lemma *cf-array-ArrMap-vdomain*[*cat-cs-simps*]:
$\mathcal{D}_\circ$ (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ArrMap \rfloor)$) = $(\mathfrak{B} \times_C \mathfrak{C})(\lfloor Arr \rfloor)$
unfolding *cf-array-components* **by** *simp*

lemma *cf-array-ArrMap-app*[*cat-cs-simps*]:
assumes *category* $\alpha\ \mathfrak{B}$
and *category* $\alpha\ \mathfrak{C}$
and $g : a \mapsto_{\mathfrak{B}} b$
and $f : a' \mapsto_{\mathfrak{C}} b'$
shows *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_\bullet =$
$\mathfrak{G}\ b(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{D}} \mathfrak{F}\ a'(\lfloor ArrMap \rfloor)(\lfloor g \rfloor)$
proof–
interpret $\mathfrak{B}$: *category* $\alpha\ \mathfrak{B}$ **by** (*rule assms*(*1*))

interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
from *cat-prod-2-is-arrI*[*OF assms*] **have** $[g, f]_\circ \in_\circ (\mathfrak{B} \times_C \mathfrak{C})(\!|Arr|\!)$ **by** *auto*
with *assms* **show** *?thesis*
unfolding *cf-array-components* **by** (*simp add*: *nat-omega-simps cat-cs-simps*)
qed

lemma *cf-array-ArrMap-vrange*:
assumes *category* α $\mathfrak{B}$
and *category* α $\mathfrak{C}$
and $\bigwedge c.\ c \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{F}\ c : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\bigwedge b.\ b \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow \mathfrak{G}\ b : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and [*cat-cs-simps*]:
$\bigwedge b\ c.\ b \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow c \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{G}\ b(\!|ObjMap|\!)(\!|c|\!) = \mathfrak{F}\ c(\!|ObjMap|\!)(\!|b|\!)$
shows $\mathcal{R}_\circ$ (*cf-array* $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}(\!|ArrMap|\!)) \subseteq_\circ \mathfrak{D}(\!|Arr|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-array-ArrMap-vdomain*)
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{BC}$: *category* α $\langle\mathfrak{B} \times_C \mathfrak{C}\rangle$
by (*simp add*: $\mathfrak{B}$.*category-axioms* $\mathfrak{C}$.*category-axioms category-cat-prod-2*)
fix *gf* **assume** *prems*: $gf \in_\circ (\mathfrak{B} \times_C \mathfrak{C})(\!|Arr|\!)$
then obtain *bc b'c'* **where** *gf*: $gf : bc \mapsto_{\mathfrak{B} \times_C \mathfrak{C}} b'c'$ **by** *auto*
then obtain *g f b c b' c'*
where *gf-def*: $gf = [g, f]_\circ$
and $bc = [b, c]_\circ$
and $b'c' = [b', c']_\circ$
and *g*: $g : b \mapsto_{\mathfrak{B}} b'$
and *f*: $f : c \mapsto_{\mathfrak{C}} c'$
by (*elim cat-prod-2-is-arrE*[*OF assms*(*1,2*)])
then have *b*: $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and *b'*: $b' \in_\circ \mathfrak{B}(\!|Obj|\!)$
and *c*: $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
and *c'*: $c' \in_\circ \mathfrak{C}(\!|Obj|\!)$
by *auto*
interpret $\mathfrak{G}b$: *is-functor* α $\mathfrak{C}$ $\mathfrak{D}$ $\langle\mathfrak{G}\ b\rangle$ **by** (*rule assms*(*4*)[*OF b*])
interpret $\mathfrak{F}c$: *is-functor* α $\mathfrak{B}$ $\mathfrak{D}$ $\langle\mathfrak{F}\ c\rangle$ **by** (*rule assms*(*3*)[*OF c*])
interpret $\mathfrak{G}b'$: *is-functor* α $\mathfrak{C}$ $\mathfrak{D}$ $\langle\mathfrak{G}\ b'\rangle$ **by** (*rule assms*(*4*)[*OF b'*])
interpret $\mathfrak{F}c'$: *is-functor* α $\mathfrak{B}$ $\mathfrak{D}$ $\langle\mathfrak{F}\ c'\rangle$ **by** (*rule assms*(*3*)[*OF c'*])
from
$\mathfrak{G}b$.*is-functor-axioms*
$\mathfrak{F}c$.*is-functor-axioms*
$\mathfrak{G}b'$.*is-functor-axioms*
$\mathfrak{F}c'$.*is-functor-axioms*
$\mathfrak{G}b$.*HomCod.category-axioms*
g f
have $\mathfrak{G}\ b'(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{D}} \mathfrak{F}\ c(\!|ArrMap|\!)(\!|g|\!) \in_\circ \mathfrak{D}(\!|Arr|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
with *g f prems* **show** *cf-array* $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}(\!|ArrMap|\!)(\!|gf|\!) \in_\circ \mathfrak{D}(\!|Arr|\!)$
unfolding *gf-def cf-array-components*
by (*simp add*: *nat-omega-simps cat-cs-simps*)
qed (*simp add*: *cf-array-ArrMap-vsv*)

8.18.4 Array bifunctor is a bifunctor

lemma *cf-array-specification*:
— See Proposition 1 from Chapter II-3 in [7].
assumes *category* α $\mathfrak{B}$
and *category* α $\mathfrak{C}$
and *category* α $\mathfrak{D}$

and $\bigwedge c.\ c \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{F}\ c : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\bigwedge b.\ b \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow \mathfrak{G}\ b : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\bigwedge b\ c.\ b \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow c \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow \mathfrak{G}\ b(\!|ObjMap|\!)(\!|c|\!) = \mathfrak{F}\ c(\!|ObjMap|\!)(\!|b|\!)$
and
$\bigwedge b\ c\ b'\ c'\ f\ g.\ [\![\ f : b \mapsto_{\mathfrak{B}} b';\ g : c \mapsto_{\mathfrak{C}} c'\ ]\!] \Longrightarrow$
$\mathfrak{G}\ b'(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{D}} \mathfrak{F}\ c(\!|ArrMap|\!)(\!|f|\!) =$
$\mathfrak{F}\ c'(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{D}} \mathfrak{G}\ b(\!|ArrMap|\!)(\!|g|\!)$

shows *cf-array-is-functor*: *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and *cf-array-ObjMap-app-fst*: $\bigwedge b\ c.\ [\![\ b \in_\circ \mathfrak{B}(\!|Obj|\!);\ c \in_\circ \mathfrak{C}(\!|Obj|\!)\ ]\!] \Longrightarrow$
cf-array $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)(\!|b,\ c|\!)_\bullet = \mathfrak{F}\ c(\!|ObjMap|\!)(\!|b|\!)$
and *cf-array-ObjMap-app-snd*: $\bigwedge b\ c.\ [\![\ b \in_\circ \mathfrak{B}(\!|Obj|\!);\ c \in_\circ \mathfrak{C}(\!|Obj|\!)\ ]\!] \Longrightarrow$
cf-array $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)(\!|b,\ c|\!)_\bullet = \mathfrak{G}\ b(\!|ObjMap|\!)(\!|c|\!)$
and *cf-array-ArrMap-app-fst*: $\bigwedge a\ b\ f\ c.\ [\![\ f : a \mapsto_{\mathfrak{B}} b;\ c \in_\circ \mathfrak{C}(\!|Obj|\!)]\!] \Longrightarrow$
cf-array $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ArrMap|\!)(\!|f,\ \mathfrak{C}(\!|CId|\!)(\!|c|\!)|\!)_\bullet = \mathfrak{F}\ c(\!|ArrMap|\!)(\!|f|\!)$
and *cf-array-ArrMap-app-snd*: $\bigwedge a\ b\ g\ c.\ [\![\ g : a \mapsto_{\mathfrak{C}} b;\ c \in_\circ \mathfrak{B}(\!|Obj|\!)\ ]\!] \Longrightarrow$
cf-array $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ArrMap|\!)(\!|\mathfrak{B}(\!|CId|\!)(\!|c|\!),\ g|\!)_\bullet = \mathfrak{G}\ c(\!|ArrMap|\!)(\!|g|\!)$

proof–

interpret $\mathfrak{B}$: *category* $\alpha\ \mathfrak{B}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{C}$: *category* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{D}$: *category* $\alpha\ \mathfrak{D}$ **by** (*rule assms*(*3*))

from *assms*(*4*) **have** [*cat-cs-intros*]: $\mathfrak{F}\ c : \mathfrak{B}' \mapsto\mapsto_{C\alpha'} \mathfrak{D}'$
if $c \in_\circ \mathfrak{C}(\!|Obj|\!)\ \mathfrak{B}' = \mathfrak{B}\ \mathfrak{D}' = \mathfrak{D}\ \alpha' = \alpha$ **for** $\alpha'\ c\ \mathfrak{B}'\ \mathfrak{D}'$
using *that*(*1*) **unfolding** *that*(*2–4*) **by** (*intro assms*(*4*))
from *assms*(*4*) **have** [*cat-cs-intros*]: $\mathfrak{G}\ c : \mathfrak{C}' \mapsto\mapsto_{C\alpha'} \mathfrak{D}'$
if $c \in_\circ \mathfrak{B}(\!|Obj|\!)\ \mathfrak{C}' = \mathfrak{C}\ \mathfrak{D}' = \mathfrak{D}\ \alpha' = \alpha$ **for** $\alpha'\ c\ \mathfrak{C}'\ \mathfrak{D}'$
using *that*(*1*) **unfolding** *that*(*2–4*) **by** (*intro assms*(*5*))

show *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof(*intro is-functorI'*)
show *vfsequence* (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}$) **unfolding** *cf-array-def* **by** *auto*
from *assms*(*1,2*) **show** *category* α ($\mathfrak{B} \times_C \mathfrak{C}$)
by (*simp add*: *category-cat-prod-2*)
show *vcard* (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}$) = $4_{\mathbb{N}}$
unfolding *cf-array-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)$) $\subseteq_\circ \mathfrak{D}(\!|Obj|\!)$
by (*rule cf-array-ObjMap-vrange*) (*auto simp*: *assms intro*: *cat-cs-intros*)
show *cf-array-is-arrI*: *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ArrMap|\!)(\!|ff'|\!)$:
cf-array $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)(\!|aa'|\!) \mapsto_{\mathfrak{D}}$ *cf-array* $\mathfrak{B}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)(\!|bb'|\!)$
if *ff'*: $ff' : aa' \mapsto_{\mathfrak{B} \times_C \mathfrak{C}} bb'$ **for** $aa'\ bb'\ ff'$
proof–
obtain $f\ f'\ a\ a'\ b\ b'$
where *ff'-def*: $ff' = [f,\ f']_\circ$
and *aa'-def*: $aa' = [a,\ a']_\circ$
and *bb'-def*: $bb' = [b,\ b']_\circ$
and *f*: $f : a \mapsto_{\mathfrak{B}} b$
and *f'*: $f' : a' \mapsto_{\mathfrak{C}} b'$
by (*elim cat-prod-2-is-arrE*[*OF* $\mathfrak{B}$.*category-axioms* $\mathfrak{C}$.*category-axioms ff'*])
then have *a*: $a \in_\circ \mathfrak{B}(\!|Obj|\!)$
and *b*: $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and *a'*: $a' \in_\circ \mathfrak{C}(\!|Obj|\!)$
and *b'*: $b' \in_\circ \mathfrak{C}(\!|Obj|\!)$
by *auto*
from f' *assms*(*5*)[*OF a*] *a* **have**
$\mathfrak{G}\ a(\!|ArrMap|\!)(\!|f'|\!) : \mathfrak{F}\ a'(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{D}} \mathfrak{F}\ b'(\!|ObjMap|\!)(\!|a|\!)$
by (*cs-concl* **cs-simp**: *assms*(*6*)[*symmetric*] **cs-intro**: *cat-cs-intros*)
with *assms*(*1–3*) $f\ f'$ *assms*(*4*)[*OF b'*] **show** *?thesis*

```
    unfolding ff'-def aa'-def bb'-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps assms(6)
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
qed
show cf-array 𝔅 ℭ 𝔇 𝔉 𝔊(|ArrMap|)(|gg' ∘_A𝔅 ×_C ℭ ff'|) =
  cf-array 𝔅 ℭ 𝔇 𝔉 𝔊(|ArrMap|)(|gg'|) ∘_A𝔇 cf-array 𝔅 ℭ 𝔇 𝔉 𝔊(|ArrMap|)(|ff'|)
  if gg': gg' : bb' ↦_𝔅 ×_C ℭ cc' and ff': ff' : aa' ↦_𝔅 ×_C ℭ bb'
  for bb' cc' gg' aa' ff'
proof–
  obtain g g' b b' c c'
    where gg'-def: gg' = [g, g']∘
      and bb'-def: bb' = [b, b']∘
      and cc'-def: cc' = [c, c']∘
      and g: g : b ↦_𝔅 c
      and g': g' : b' ↦_ℭ c'
    by (elim cat-prod-2-is-arrE[OF 𝔅.category-axioms ℭ.category-axioms gg'])
  moreover obtain f f' a a' b'' b'''
    where ff'-def: ff' = [f, f']∘
      and aa'-def: aa' = [a, a']∘
      and bb' = [b'', b''']∘
      and f: f : a ↦_𝔅 b''
      and f': f' : a' ↦_ℭ b'''
    by (elim cat-prod-2-is-arrE[OF 𝔅.category-axioms ℭ.category-axioms ff'])
  ultimately have f: f : a ↦_𝔅 b and f': f' : a' ↦_ℭ b' by auto
  with g have a: a ∈_∘ 𝔅(|Obj|)
    and b: b ∈_∘ 𝔅(|Obj|)
    and c: c ∈_∘ 𝔅(|Obj|)
    and a': a' ∈_∘ ℭ(|Obj|)
    and b': b' ∈_∘ ℭ(|Obj|)
    and c': b' ∈_∘ ℭ(|Obj|)
    by auto
  from f' assms(5)[OF a] a have 𝔊a-f':
    𝔊 a(|ArrMap|)(|f'|) : 𝔉 a'(|ObjMap|)(|a|) ↦_𝔇 𝔉 b'(|ObjMap|)(|a|)
    by (cs-concl cs-simp: assms(6)[symmetric] cs-intro: cat-cs-intros)
  from f' b assms(5)[OF b] have 𝔊b-f':
    𝔊 b(|ArrMap|)(|f'|) : 𝔉 a'(|ObjMap|)(|b|) ↦_𝔇 𝔉 b'(|ObjMap|)(|b|)
    by (cs-concl cs-simp: assms(6)[symmetric] cs-intro: cat-cs-intros)
  from f' c assms(5)[OF c] have 𝔊c-f':
    𝔊 c(|ArrMap|)(|f'|) : 𝔉 a'(|ObjMap|)(|c|) ↦_𝔇 𝔉 b'(|ObjMap|)(|c|)
    by (cs-concl cs-simp: assms(6)[symmetric] cs-intro: cat-cs-intros)
  have
    𝔉 b'(|ArrMap|)(|g|) ∘_A𝔇 (𝔉 b'(|ArrMap|)(|f|) ∘_A𝔇 𝔊 a(|ArrMap|)(|f'|)) =
      (𝔊 c(|ArrMap|)(|f'|) ∘_A𝔇 𝔉 a'(|ArrMap|)(|g|)) ∘_A𝔇 𝔉 a'(|ArrMap|)(|f|)
    using f' f g 𝔊b-f' assms(4)[OF a'] assms(4)[OF b']
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps assms(7) cs-intro: cat-cs-intros
      )
  also have ... =
    𝔊 c(|ArrMap|)(|f'|) ∘_A𝔇 (𝔉 a'(|ArrMap|)(|g|) ∘_A𝔇 𝔉 a'(|ArrMap|)(|f|))
    using assms(2) f f' g g' assms(4)[OF a'] assms(5)[OF c]
    by (cs-concl cs-simp: assms(6) cat-cs-simps cs-intro: cat-cs-intros)
  finally have [cat-cs-simps]:
```

```
      𝔉 b′(|ArrMap|)(|g|) ∘_A𝔇 (𝔉 b′(|ArrMap|)(|f|) ∘_A𝔇 𝔊 a(|ArrMap|)(|f′|)) =
        𝔊 c(|ArrMap|)(|f′|) ∘_A𝔇 (𝔉 a′(|ArrMap|)(|g|) ∘_A𝔇 𝔉 a′(|ArrMap|)(|f|))
      by simp
    show ?thesis
      using
        𝔊a-f′ 𝔊c-f′
        f f′
        g g′
        assms(1,2)
        assms(4)[OF a′]
        assms(4)[OF c′]
        assms(5)[OF c]
      unfolding gg′-def ff′-def aa′-def bb′-def cc′-def
      by
        (
          cs-concl
            cs-simp: assms(6,7) cat-prod-cs-simps cat-cs-simps
            cs-intro: cat-prod-cs-intros cat-cs-intros
        )
  qed
  show cf-array 𝔄 ℭ 𝔇 𝔉 𝔊(|ArrMap|)(|(𝔄 ×_C ℭ)(|CId|)(|cc′|)|) =
    𝔇(|CId|)(|cf-array 𝔄 ℭ 𝔇 𝔉 𝔊(|ObjMap|)(|cc′|)|)
    if cc′ ∈_∘ (𝔄 ×_C ℭ)(|Obj|) for cc′
  proof–
    from that obtain c c′
      where cc′-def: cc′ = [c, c′]_∘
        and c: c ∈_∘ 𝔄(|Obj|)
        and c′: c′ ∈_∘ ℭ(|Obj|)
      by (elim cat-prod-2-ObjE[rotated 2]) (auto intro: cat-cs-intros)
    from assms(1,2,3) c c′ assms(4)[OF c′] assms(5)[OF c] show ?thesis
      unfolding cc′-def
      by
        (
          cs-concl
            cs-simp: cat-prod-cs-simps cat-cs-simps assms(6)
            cs-intro: cat-cs-intros cat-prod-cs-intros
        )
  qed
qed (auto simp: cf-array-components cat-cs-intros)

show cf-array 𝔄 ℭ 𝔇 𝔉 𝔊(|ObjMap|)(|b, c|)_• = 𝔉 c(|ObjMap|)(|b|)
  if b ∈_∘ 𝔄(|Obj|) and c ∈_∘ ℭ(|Obj|) for b c
  using that assms(1,2,3)
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps assms(6) cs-intro: cat-prod-cs-intros
    )
show cf-array 𝔄 ℭ 𝔇 𝔉 𝔊(|ObjMap|)(|b, c|)_• = 𝔊 b(|ObjMap|)(|c|)
  if b ∈_∘ 𝔄(|Obj|) and c ∈_∘ ℭ(|Obj|) for b c
  using that assms(1,2,3)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-prod-cs-intros)
show cf-array 𝔄 ℭ 𝔇 𝔉 𝔊(|ArrMap|)(|f, ℭ(|CId|)(|c|)|)_• = 𝔉 c(|ArrMap|)(|f|)
  if f: f : a ↦_𝔄 b and c: c ∈_∘ ℭ(|Obj|) for a b f c
proof–
  from f have a ∈_∘ 𝔄(|Obj|) and b ∈_∘ 𝔄(|Obj|) by auto
  from assms(5)[OF this(1)] assms(5)[OF this(2)] assms(4)[OF c] show ?thesis
    using assms(1,2,3) f c
```

by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps assms*(*6*) **cs-intro**: *cat-cs-intros*
)
qed

show *cf-array* $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|$\mathfrak{B}$(|*CId*|)(|*c*|), *g*|)$_{\bullet}$ = $\mathfrak{G}$ *c*(|*ArrMap*|)(|*g*|)
if *g*: *g* : *a* $\mapsto_{\mathfrak{C}}$ *b* **and** *c*: *c* $\in_{\circ}$ $\mathfrak{B}$(|*Obj*|) **for** *a b g c*
proof–
from *g* **have** *a* $\in_{\circ}$ $\mathfrak{C}$(|*Obj*|) **and** *b* $\in_{\circ}$ $\mathfrak{C}$(|*Obj*|) **by** *auto*
from *assms*(*4*)[*OF this*(*1*)] *assms*(*4*)[*OF this*(*2*)] *assms*(*5*)[*OF c*] **show** *?thesis*
using *assms*(*1,2,3*) *g c*
by
(
cs-concl
cs-simp: *cat-cs-simps assms*(*6*)[*symmetric*] **cs-intro**: *cat-cs-intros*
)
qed

qed

8.19 Composition of a covariant bifunctor and covariant functors

8.19.1 Definition and elementary properties.

definition *cf-bcomp* :: *V* ⇒ *V* ⇒ *V* ⇒ *V*
where *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$ =
[
(
$\lambda a \in_{\circ} (\mathfrak{F}(|HomDom|) \times_C \mathfrak{G}(|HomDom|))(|Obj|)$.
$\mathfrak{S}(|ObjMap|)(|\mathfrak{F}(|ObjMap|)(|vpfst\ a|), \mathfrak{G}(|ObjMap|)(|vpsnd\ a|)|)_{\bullet}$
),
(
$\lambda f \in_{\circ} (\mathfrak{F}(|HomDom|) \times_C \mathfrak{G}(|HomDom|))(|Arr|)$.
$\mathfrak{S}(|ArrMap|)(|\mathfrak{F}(|ArrMap|)(|vpfst\ f|), \mathfrak{G}(|ArrMap|)(|vpsnd\ f|)|)_{\bullet}$
),
$\mathfrak{F}(|HomDom|) \times_C \mathfrak{G}(|HomDom|)$,
$\mathfrak{S}(|HomCod|)$
]$_{\circ}$

Components.

lemma *cf-bcomp-components*:
shows *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|) =
(
$\lambda a \in_{\circ} (\mathfrak{F}(|HomDom|) \times_C \mathfrak{G}(|HomDom|))(|Obj|)$.
$\mathfrak{S}(|ObjMap|)(|\mathfrak{F}(|ObjMap|)(|vpfst\ a|), \mathfrak{G}(|ObjMap|)(|vpsnd\ a|)|)_{\bullet}$
)
and *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|) =
(
$\lambda f \in_{\circ} (\mathfrak{F}(|HomDom|) \times_C \mathfrak{G}(|HomDom|))(|Arr|)$.
$\mathfrak{S}(|ArrMap|)(|\mathfrak{F}(|ArrMap|)(|vpfst\ f|), \mathfrak{G}(|ArrMap|)(|vpsnd\ f|)|)_{\bullet}$
)
and *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*HomDom*|) = $\mathfrak{F}(|HomDom|) \times_C \mathfrak{G}(|HomDom|)$
and *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*HomCod*|) = $\mathfrak{S}(|HomCod|)$
unfolding *cf-bcomp-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

8.19.2 Object map

lemma *cf-bcomp-ObjMap-vsv*: *vsv* (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|))
unfolding *cf-bcomp-components* **by** *simp*

lemma *cf-bcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)) = ($\mathfrak{B}' \times_C \mathfrak{C}'$)(|*Obj*|)
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*)
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*)
show *?thesis* **unfolding** *cf-bcomp-components* **by** (*simp add*: *cat-cs-simps*)
qed

lemma *cf-bcomp-ObjMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $[a, b]_\circ \in_\circ (\mathfrak{B}' \times_C \mathfrak{C}')$(|*Obj*|)
shows *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)(|a, b|)$_\bullet$ = $\mathfrak{S}$(|*ObjMap*|)(|$\mathfrak{F}$(|*ObjMap*|)(|a|), $\mathfrak{G}$(|*ObjMap*|)(|b|)|)$_\bullet$
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
from *assms* **show** *?thesis*
unfolding *cf-bcomp-components*
by (*simp-all add*: *cat-cs-simps nat-omega-simps*)
qed

lemma *cf-bcomp-ObjMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)) $\subseteq_\circ$ $\mathfrak{D}$(|*Obj*|)
proof
(
rule vsv.vsv-vrange-vsubset,
unfold cf-bcomp-ObjMap-vdomain[*OF assms*(*1*,*2*)]
)
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
show *vsv* (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)) **by** (*rule cf-bcomp-ObjMap-vsv*)
fix *bc* **assume** *bc* $\in_\circ$ ($\mathfrak{B}' \times_C \mathfrak{C}'$)(|*Obj*|)
with $\mathfrak{F}$.*HomDom.category-axioms* $\mathfrak{G}$.*HomDom.category-axioms* **obtain** *b c*
where *bc-def*: *bc* = $[b, c]_\circ$ **and** *b*: *b* $\in_\circ$ $\mathfrak{B}'$(|*Obj*|) **and** *c*: *c* $\in_\circ$ $\mathfrak{C}'$(|*Obj*|)
by (*elim cat-prod-2-ObjE*[*rotated* −*1*])
from *assms b c* **show** *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)(|*bc*|) $\in_\circ$ $\mathfrak{D}$(|*Obj*|)
unfolding *bc-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
qed

8.19.3 Arrow map

lemma *cf-bcomp-ArrMap-vsv*: *vsv* (*cf-bcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(|*ArrMap*|))
unfolding *cf-bcomp-components* **by** *simp*

lemma *cf-bcomp-ArrMap-vdomain*[*cat-cs-simps*]:

assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ\ (\textit{cf-bcomp}\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|\textit{ArrMap}|\!))\ =\ (\mathfrak{B}' \times_C \mathfrak{C}')(\!|\textit{Arr}|\!)$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}'\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}'\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
show *?thesis* **unfolding** *cf-bcomp-components* **by** (*simp add*: *cat-cs-simps*)
qed

lemma *cf-bcomp-ArrMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $[g, f]_\circ \in_\circ (\mathfrak{B}' \times_C \mathfrak{C}')(\!|\textit{Arr}|\!)$
shows $\textit{cf-bcomp}\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|\textit{ArrMap}|\!)(\!|g, f|\!)_\bullet = \mathfrak{S}(\!|\textit{ArrMap}|\!)(\!|\mathfrak{F}(\!|\textit{ArrMap}|\!)(\!|g|\!),\ \mathfrak{G}(\!|\textit{ArrMap}|\!)(\!|f|\!)|\!)_\bullet$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}'\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}'\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
from *assms* **show** *?thesis*
unfolding *cf-bcomp-components* **by** (*simp-all add*: *nat-omega-simps cat-cs-simps*)
qed

lemma *cf-bcomp-ArrMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ\ (\textit{cf-bcomp}\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|\textit{ArrMap}|\!)) \subseteq_\circ \mathfrak{D}(\!|\textit{Arr}|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cf-bcomp-ArrMap-vdomain*[*OF assms*(*1,2*)])
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}'\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}'\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
fix *gf* **assume** $gf \in_\circ (\mathfrak{B}' \times_C \mathfrak{C}')(\!|\textit{Arr}|\!)$
with $\mathfrak{F}$.*HomDom.category-axioms* $\mathfrak{G}$.*HomDom.category-axioms* **obtain** *g f*
where *gf-def*: $gf = [g, f]_\circ$ **and** *g*: $g \in_\circ \mathfrak{B}'(\!|\textit{Arr}|\!)$ **and** *f*: $f \in_\circ \mathfrak{C}'(\!|\textit{Arr}|\!)$
by (*elim cat-prod-2-ArrE*[*rotated −1*])
from *g* **obtain** *a b* **where** *g*: $g : a \mapsto_{\mathfrak{B}'} b$ **by** *auto*
from *f* **obtain** $a'\ b'$ **where** *f*: $f : a' \mapsto_{\mathfrak{C}'} b'$ **by** *auto*
from *assms g f* **show** $\textit{cf-bcomp}\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|\textit{ArrMap}|\!)(\!|gf|\!) \in_\circ \mathfrak{D}(\!|\textit{Arr}|\!)$
unfolding *gf-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
qed (*simp add*: *cf-bcomp-ArrMap-vsv*)

8.19.4 Composition of a covariant bifunctor and covariant functors is a functor

lemma *cf-bcomp-is-functor*:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\textit{cf-bcomp}\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G} : \mathfrak{B}' \times_C \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
proof–

interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}'\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}'\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{S}$: *is-functor* $\alpha\ \langle\mathfrak{B} \times_C \mathfrak{C}\rangle\ \mathfrak{D}\ \mathfrak{S}$ **by** (*rule assms*(*3*))

show *?thesis*
proof(*intro is-functorI'*)

show *vfsequence* (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$) **unfolding** *cf-bcomp-def* **by** *simp*
show *category* α ($\mathfrak{B}' \times_C \mathfrak{C}'$)
by
(
simp add:
$\mathfrak{F}$.*HomDom.category-axioms*
$\mathfrak{G}$.*HomDom.category-axioms*
category-cat-prod-2
)
show *vcard* (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$) = $\mathcal{4}_{\mathbb{N}}$
unfolding *cf-bcomp-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** $\mathcal{R}_\circ$ (*cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)) $\subseteq_\circ$ $\mathfrak{D}$(|*Obj*|)
by (*rule cf-bcomp-ObjMap-vrange*)
show *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|ff'|) :
cf-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)(|aa'|) $\mapsto_{\mathfrak{D}}$ *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)(|bb'|)
if ff': ff' : $aa' \mapsto_{\mathfrak{B}' \times_C \mathfrak{C}'} bb'$ **for** aa' bb' ff'
proof–
obtain f f' a a' b b'
where ff'-*def*: $ff' = [f, f']_\circ$
and aa'-*def*: $aa' = [a, a']_\circ$
and bb'-*def*: $bb' = [b, b']_\circ$
and f: $f : a \mapsto_{\mathfrak{B}'} b$
and f': $f' : a' \mapsto_{\mathfrak{C}'} b'$
by
(
elim
cat-prod-2-is-arrE[
OF $\mathfrak{F}$.*HomDom.category-axioms* $\mathfrak{G}$.*HomDom.category-axioms* ff'
]
)
from *assms* f f' **show** *?thesis*
unfolding ff'-*def* aa'-*def* bb'-*def*
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
qed
show *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|$gg' \circ_{A\mathfrak{B}' \times_C \mathfrak{C}'} ff'$|) =
cf-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|gg'|) $\circ_{A\mathfrak{D}}$ *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|ff'|)
if gg': $gg' : bb' \mapsto_{\mathfrak{B}' \times_C \mathfrak{C}'} cc'$
and ff': $ff' : aa' \mapsto_{\mathfrak{B}' \times_C \mathfrak{C}'} bb'$
for bb' cc' gg' aa' ff'
proof–
obtain g g' b b' c c'
where gg'-*def*: $gg' = [g, g']_\circ$
and bb'-*def*: $bb' = [b, b']_\circ$
and cc'-*def*: $cc' = [c, c']_\circ$
and g: $g : b \mapsto_{\mathfrak{B}'} c$
and g': $g' : b' \mapsto_{\mathfrak{C}'} c'$
by
(
elim cat-prod-2-is-arrE[
OF $\mathfrak{F}$.*HomDom.category-axioms* $\mathfrak{G}$.*HomDom.category-axioms* gg'
]
)
moreover obtain f f' a a' b'' b'''
where ff'-*def*: $ff' = [f, f']_\circ$

```
      and aa'-def: aa' = [a, a']∘
      and bb' = [b'', b''']∘
      and f: f : a ↦_𝔅' b''
      and f': f' : a' ↦_𝔈' b'''
    by
      (
        elim cat-prod-2-is-arrE[
          OF 𝔉.HomDom.category-axioms 𝔊.HomDom.category-axioms ff'
          ]
      )
  ultimately have f: f : a ↦_𝔅' b and f': f' : a' ↦_𝔈' b' by auto
  from assms f f' g g' have [cat-cs-simps]:
    [𝔉(|ArrMap|)(|g|) ∘_A𝔅 𝔉(|ArrMap|)(|f|), 𝔊(|ArrMap|)(|g'|) ∘_A𝔈 𝔊(|ArrMap|)(|f'|)]∘ =
      [𝔉(|ArrMap|)(|g|), 𝔊(|ArrMap|)(|g'|)]∘ ∘_A𝔅 ×_C 𝔈 [𝔉(|ArrMap|)(|f|), 𝔊(|ArrMap|)(|f'|)]∘
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-prod-cs-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  from assms f f' g g' show ?thesis
    unfolding gg'-def ff'-def aa'-def bb'-def cc'-def
    by
      (
        cs-concl
          cs-simp: cat-prod-cs-simps cat-cs-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
qed
show
  cf-bcomp 𝔖 𝔉 𝔊(|ArrMap|)(|(𝔅' ×_C 𝔈')(|CId|)(|cc'|)|) =
    𝔇(|CId|)(|cf-bcomp 𝔖 𝔉 𝔊(|ObjMap|)(|cc'|)|)
  if cc' ∈∘ (𝔅' ×_C 𝔈')(|Obj|) for cc'
proof–
  from that obtain c c'
    where cc'-def: cc' = [c, c']∘
      and c: c ∈∘ 𝔅'(|Obj|)
      and c': c' ∈∘ 𝔈'(|Obj|)
    by (elim cat-prod-2-ObjE[rotated 2]) (auto intro: cat-cs-intros)
  from assms c c' have [cat-cs-simps]:
    [𝔅(|CId|)(|𝔉(|ObjMap|)(|c|)|), 𝔈(|CId|)(|𝔊(|ObjMap|)(|c'|)|)]∘ =
      (𝔅 ×_C 𝔈)(|CId|)(|𝔉(|ObjMap|)(|c|), 𝔊(|ObjMap|)(|c'|)|)∙
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-prod-cs-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  from assms c c' show ?thesis
    unfolding cc'-def
    by
      (
        cs-concl
          cs-simp: cat-prod-cs-simps cat-cs-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
qed
qed (auto simp: cf-bcomp-components cat-cs-intros cat-cs-simps)
```

qed

lemma *cf-bcomp-is-functor'*[*cat-cs-intros*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{A}' = \mathfrak{B}' \times_C \mathfrak{C}'$
shows *cf-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-bcomp-is-functor*)

8.20 Composition of a contracovariant bifunctor and covariant functors

The term *contracovariant bifunctor* is used to refer to a bifunctor that is contravariant in the first argument and covariant in the second argument.

definition *cf-cn-cov-bcomp* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}$ =
[
(
$\lambda a \in_\circ (op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)) \times_C \mathfrak{G}(\!|HomDom|\!))(\!|Obj|\!).$
$\mathfrak{S}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|vpfst\ a|\!), \mathfrak{G}(\!|ObjMap|\!)(\!|vpsnd\ a|\!)|\!)_\bullet$
),
(
$\lambda f \in_\circ (op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)) \times_C \mathfrak{G}(\!|HomDom|\!))(\!|Arr|\!).$
$\mathfrak{S}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|vpfst\ f|\!), \mathfrak{G}(\!|ArrMap|\!)(\!|vpsnd\ f|\!)|\!)_\bullet$
),
$op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)) \times_C \mathfrak{G}(\!|HomDom|\!),$
$\mathfrak{S}(\!|HomCod|\!)$
]$_\circ$

Components.

lemma *cf-cn-cov-bcomp-components*:
shows *cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)$ =
(
$\lambda a \in_\circ (op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)) \times_C \mathfrak{G}(\!|HomDom|\!))(\!|Obj|\!).$
$\mathfrak{S}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|vpfst\ a|\!), \mathfrak{G}(\!|ObjMap|\!)(\!|vpsnd\ a|\!)|\!)_\bullet$
)
and *cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|ArrMap|\!)$ =
(
$\lambda f \in_\circ (op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)) \times_C \mathfrak{G}(\!|HomDom|\!))(\!|Arr|\!).$
$\mathfrak{S}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|vpfst\ f|\!), \mathfrak{G}(\!|ArrMap|\!)(\!|vpsnd\ f|\!)|\!)_\bullet$
)
and *cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|HomDom|\!) = op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)) \times_C \mathfrak{G}(\!|HomDom|\!)$
and *cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|HomCod|\!) = \mathfrak{S}(\!|HomCod|\!)$
unfolding *cf-cn-cov-bcomp-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

8.20.1 Object map

lemma *cf-cn-cov-bcomp-ObjMap-vsv*: *vsv* (*cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)$)
unfolding *cf-cn-cov-bcomp-components* **by** *simp*

lemma *cf-cn-cov-bcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}(\!|ObjMap|\!)$) = (*op-cat* $\mathfrak{B}' \times_C \mathfrak{C}'$)$(\!|Obj|\!)$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}'\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(*1*))

interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
show *?thesis*
unfolding *cf-cn-cov-bcomp-components*
by (*simp add*: *nat-omega-simps cat-cs-simps*)
qed

lemma *cf-cn-cov-bcomp-ObjMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $[a, b]_\circ \in_\circ (\textit{op-cat}\ \mathfrak{B}' \times_C \mathfrak{C}')(\!|\textit{Obj}|\!)$
shows
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}(\!|\textit{ObjMap}|\!)(\!|a, b|\!)_\bullet$ =
$\mathfrak{S}(\!|\textit{ObjMap}|\!)(\!|\mathfrak{F}(\!|\textit{ObjMap}|\!)(\!|a|\!), \mathfrak{G}(\!|\textit{ObjMap}|\!)(\!|b|\!)|\!)_\bullet$
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
from *assms* **show** *?thesis*
unfolding *cf-cn-cov-bcomp-components*
by (*simp-all add*: *cat-cs-simps nat-omega-simps*)
qed

lemma *cf-cn-cov-bcomp-ObjMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \textit{op-cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}(\!|\textit{ObjMap}|\!)$) $\subseteq_\circ \mathfrak{D}(\!|\textit{Obj}|\!)$
proof
(
rule vsv.vsv-vrange-vsubset,
unfold cf-cn-cov-bcomp-ObjMap-vdomain[*OF assms*(*1,2*)]
)
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
show *vsv* (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}(\!|\textit{ObjMap}|\!)$)
by (*rule cf-cn-cov-bcomp-ObjMap-vsv*)
fix *bc* **assume** *bc* $\in_\circ$ (*op-cat* $\mathfrak{B}' \times_C \mathfrak{C}'$)$(\!|\textit{Obj}|\!)$
with $\mathfrak{F}$.*HomDom.category-op* $\mathfrak{G}$.*HomDom.category-axioms* **obtain** *b c*
where *bc-def*: *bc* = $[b, c]_\circ$
and *b*: $b \in_\circ \textit{op-cat}\ \mathfrak{B}'(\!|\textit{Obj}|\!)$
and *c*: $c \in_\circ \mathfrak{C}'(\!|\textit{Obj}|\!)$
by (*elim cat-prod-2-ObjE*[*rotated* −*1*])
from *assms b c* **show** *cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}(\!|\textit{ObjMap}|\!)(\!|bc|\!) \in_\circ \mathfrak{D}(\!|\textit{Obj}|\!)$
unfolding *bc-def cat-op-simps*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed

8.20.2 Arrow map

lemma *cf-cn-cov-bcomp-ArrMap-vsv*: *vsv* (*cf-cn-cov-bcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|\textit{ArrMap}|\!)$)
unfolding *cf-cn-cov-bcomp-components* **by** *simp*

lemma *cf-cn-cov-bcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$

shows $\mathcal{D}_{\circ}$ (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)) = (*op-cat* $\mathfrak{B}'$ $\times_C$ $\mathfrak{C}'$)(|*Arr*|)
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
show *?thesis* **unfolding** *cf-cn-cov-bcomp-components* **by** (*simp add*: *cat-cs-simps*)
qed

lemma *cf-cn-cov-bcomp-ArrMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $[g, f]_{\circ} \in_{\circ}$ (*op-cat* $\mathfrak{B}'$ $\times_C$ $\mathfrak{C}'$)(|*Arr*|)
shows *cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|g, f|)$_{\bullet}$ =
$\mathfrak{S}$(|*ArrMap*|)(|$\mathfrak{F}$(|*ArrMap*|)(|g|), $\mathfrak{G}$(|*ArrMap*|)(|f|)|)$_{\bullet}$
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
from *assms* **show** *?thesis*
unfolding *cf-cn-cov-bcomp-components*
by (*simp-all add*: *nat-omega-simps cat-cs-simps*)
qed

lemma *cf-cn-cov-bcomp-ArrMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} :$ *op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_{\circ}$ (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)) $\subseteq_{\circ}$ $\mathfrak{D}$(|*Arr*|)
proof
(
rule vsv.vsv-vrange-vsubset,
unfold cf-cn-cov-bcomp-ArrMap-vdomain[*OF assms*(*1,2*)]
)
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
fix *gf* **assume** *gf* $\in_{\circ}$ (*op-cat* $\mathfrak{B}'$ $\times_C$ $\mathfrak{C}'$)(|*Arr*|)
with $\mathfrak{F}$.*HomDom.category-op* $\mathfrak{G}$.*HomDom.category-axioms* **obtain** *g f*
where *gf-def*: *gf* = $[g, f]_{\circ}$
and *g*: $g \in_{\circ}$ *op-cat* $\mathfrak{B}'$(|*Arr*|)
and *f*: $f \in_{\circ}$ $\mathfrak{C}'$(|*Arr*|)
by (*elim cat-prod-2-ArrE*[*rotated* −*1*])
from *g* **obtain** *a b* **where** *g*: $g : a \mapsto_{\mathfrak{B}'} b$ **unfolding** *cat-op-simps* **by** *auto*
from *f* **obtain** a' b' **where** *f*: $f : a' \mapsto_{\mathfrak{C}'} b'$ **by** *auto*
from *assms g f* **show** *cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|*gf*|) $\in_{\circ}$ $\mathfrak{D}$(|*Arr*|)
unfolding *gf-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed (*rule cf-cn-cov-bcomp-ArrMap-vsv*)

8.20.3 Composition of a contracovariant bifunctor and functors is a functor

lemma *cf-cn-cov-bcomp-is-functor*:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} :$ *op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows *cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$: *op-cat* $\mathfrak{B}'$ $\times_C$ $\mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$

proof–

interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}'$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{C}'$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{S}$: *is-functor* α ‹*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$› $\mathfrak{D}$ $\mathfrak{S}$ **by** (*rule assms*(*3*))

show *?thesis*
proof(*intro is-functorI'*)
show *vfsequence* (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$)
unfolding *cf-cn-cov-bcomp-def* **by** *simp*
show *category* α (*op-cat* $\mathfrak{B}'$ $\times_C$ $\mathfrak{C}'$)
by
(
simp add:
$\mathfrak{F}$.*HomDom.category-op* $\mathfrak{G}$.*HomDom.category-axioms category-cat-prod-2*
)
show *vcard* (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$) = $\mathcal{4}_{\mathbb{N}}$
unfolding *cf-cn-cov-bcomp-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** $\mathcal{R}_\circ$ (*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)) $\subseteq_\circ$ $\mathfrak{D}$(|*Obj*|)
by (*rule cf-cn-cov-bcomp-ObjMap-vrange*)
show
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|ff'|) :
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)(|aa'|) $\mapsto_{\mathfrak{D}}$
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ObjMap*|)(|bb'|)
if ff': ff' : aa' $\mapsto_{op\text{-}cat\ \mathfrak{B}'\ \times_C\ \mathfrak{C}'}$ bb' **for** aa' bb' ff'
proof–
obtain f f' a a' b b'
where ff'-*def*: ff' = [f, f']$_\circ$
and aa'-*def*: aa' = [a, a']$_\circ$
and bb'-*def*: bb' = [b, b']$_\circ$
and f: f : a $\mapsto_{op\text{-}cat\ \mathfrak{B}'}$ b
and f': f' : a' $\mapsto_{\mathfrak{C}'}$ b'
by
(
elim
cat-prod-2-is-arrE[
OF $\mathfrak{F}$.*HomDom.category-op* $\mathfrak{G}$.*HomDom.category-axioms* ff'
]
)
from *assms* f f' **show** *?thesis*
unfolding ff'-*def* aa'-*def* bb'-*def cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed
show
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|gg' $\circ_{A\ op\text{-}cat\ \mathfrak{B}'\ \times_C\ \mathfrak{C}'}$ ff'|) =
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|gg'|) $\circ_{A\ \mathfrak{D}}$
cf-cn-cov-bcomp $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}$(|*ArrMap*|)(|ff'|)
if gg': gg' : bb' $\mapsto_{op\text{-}cat\ \mathfrak{B}'\ \times_C\ \mathfrak{C}'}$ cc'
and ff': ff' : aa' $\mapsto_{op\text{-}cat\ \mathfrak{B}'\ \times_C\ \mathfrak{C}'}$ bb'
for bb' cc' gg' aa' ff'
proof–
obtain g g' b b' c c'

```
    where gg′-def: gg′ = [g, g′]∘
      and bb′-def: bb′ = [b, b′]∘
      and cc′-def: cc′ = [c, c′]∘
      and g: g : b ↦_{op-cat 𝔅′} c
      and g′: g′ : b′ ↦_{𝔈′} c′
    by
      (
        elim cat-prod-2-is-arrE[
          OF 𝔉.HomDom.category-op 𝔊.HomDom.category-axioms gg′
          ]
      )
  moreover obtain f f′ a a′ b″ b‴
    where ff′-def: ff′ = [f, f′]∘
      and aa′-def: aa′ = [a, a′]∘
      and bb′ = [b″, b‴]∘
      and f: f : a ↦_{op-cat 𝔅′} b″
      and f′ : a′ ↦_{𝔈′} b‴
    by
      (
        elim cat-prod-2-is-arrE[
          OF 𝔉.HomDom.category-op 𝔊.HomDom.category-axioms ff′
          ]
      )
  ultimately have f: f : a ↦_{op-cat 𝔅′} b and f′: f′ : a′ ↦_{𝔈′} b′
    by auto
  from assms f f′ g g′ have [cat-cs-simps]:
    [
      𝔉(|ArrMap|)(|f|) ∘_{A 𝔅} 𝔉(|ArrMap|)(|g|),
      𝔊(|ArrMap|)(|g′|) ∘_{A 𝔈} 𝔊(|ArrMap|)(|f′|)
     ]∘ =
      [𝔉(|ArrMap|)(|g|), 𝔊(|ArrMap|)(|g′|)]∘ ∘_{A op-cat 𝔅 ×_C 𝔈}
      [𝔉(|ArrMap|)(|f|), 𝔊(|ArrMap|)(|f′|)]∘
    unfolding cat-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-prod-cs-simps cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  from assms f f′ g g′ show ?thesis
    unfolding gg′-def ff′-def aa′-def bb′-def cc′-def cat-op-simps
    by
      (
        cs-concl
          cs-simp: cat-prod-cs-simps cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed
show
  cf-cn-cov-bcomp 𝔖 𝔉 𝔊(|ArrMap|)(|(op-cat 𝔅′ ×_C 𝔈′)(|CId|)(|cc′|)|) =
    𝔇(|CId|)(|cf-cn-cov-bcomp 𝔖 𝔉 𝔊(|ObjMap|)(|cc′|)|)
  if cc′ ∈_∘ (op-cat 𝔅′ ×_C 𝔈′)(|Obj|) for cc′
proof–
  from that obtain c c′
    where cc′-def: cc′ = [c, c′]∘
      and c: c ∈_∘ op-cat 𝔅′(|Obj|)
      and c′: c′ ∈_∘ 𝔈′(|Obj|)
```

by (*elim cat-prod-2-ObjE*[*rotated 2*])
(*auto intro*: *cat-cs-intros*)
from *assms c c′* **have** [*cat-cs-simps*]:
$[\mathfrak{B}(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!)|\!),\ \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{G}(\!|ObjMap|\!)(\!|c'|\!)|\!)]_\circ =$
$(op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C})(\!|CId|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|c|\!),\ \mathfrak{G}(\!|ObjMap|\!)(\!|c'|\!)|\!)_\bullet$
unfolding *cat-op-simps*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-prod-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
from *assms c c′* **show** *?thesis*
unfolding *cc′-def cat-op-simps*
by
(
cs-concl
cs-simp: *cat-prod-cs-simps cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed
qed (*auto simp*: *cf-cn-cov-bcomp-components cat-cs-simps intro*: *cat-cs-intros*)

qed

lemma *cf-cn-cov-bcomp-is-functor′*[*cat-cs-intros*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{A}' = op\text{-}cat\ \mathfrak{B}' \times_C \mathfrak{C}'$
shows *cf-cn-cov-bcomp* $\mathfrak{S}\ \mathfrak{F}\ \mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-cn-cov-bcomp-is-functor*)

8.20.4 Projection of a contracovariant bifunctor and functors

lemma *cf-cn-cov-bcomp-bifunctor-proj-snd*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $b \in_\circ \mathfrak{B}'(\!|Obj|\!)$
shows
$cf\text{-}cn\text{-}cov\text{-}bcomp\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF} =$
$(\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),-)_{CF}) \circ_{CF} \mathfrak{G}$
proof(*rule cf-eqI*)
from *assms* **show** [*intro*]:
$cf\text{-}cn\text{-}cov\text{-}bcomp\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
$(\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),-)_{CF}) \circ_{CF} \mathfrak{G} : \mathfrak{C}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros*)+
from *assms* **have** *ObjMap-dom-lhs*:
$\mathcal{D}_\circ\ ((cf\text{-}cn\text{-}cov\text{-}bcomp\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF})(\!|ObjMap|\!)) = \mathfrak{C}'(\!|Obj|\!)$
and *ObjMap-dom-rhs*:
$\mathcal{D}_\circ\ (((\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),-)_{CF}) \circ_{CF} \mathfrak{G})(\!|ObjMap|\!)) = \mathfrak{C}'(\!|Obj|\!)$
and *ArrMap-dom-lhs*:
$\mathcal{D}_\circ\ ((cf\text{-}cn\text{-}cov\text{-}bcomp\ \mathfrak{S}\ \mathfrak{F}\ \mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF})(\!|ArrMap|\!)) = \mathfrak{C}'(\!|Arr|\!)$
and *ArrMap-dom-rhs*:
$\mathcal{D}_\circ\ (((\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),-)_{CF}) \circ_{CF} \mathfrak{G})(\!|ArrMap|\!)) = \mathfrak{C}'(\!|Arr|\!)$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros* **cs-simp**: *cat-cs-simps*)+

show
(*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF}$)(⟦*ObjMap*⟧) =
((($\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(⟦ObjMap⟧)(⟦b⟧),-)_{CF}$) $\circ_{CF}$ $\mathfrak{G}$)(⟦*ObjMap*⟧)
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
fix *a* **assume** *a* $\in_\circ$ $\mathfrak{C}'$(⟦*Obj*⟧)
with *assms* **show**
(*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF}$)(⟦*ObjMap*⟧)(⟦*a*⟧) =
((($\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(⟦ObjMap⟧)(⟦b⟧),-)_{CF}$) $\circ_{CF}$ $\mathfrak{G}$)(⟦*ObjMap*⟧)(⟦*a*⟧)
by
(
cs-concl
cs-simp: *cat-prod-cs-simps cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed (*auto intro*: *is-functor.cf-ObjMap-vsv*)
show
(*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF}$)(⟦*ArrMap*⟧) =
((($\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(⟦ObjMap⟧)(⟦b⟧),-)_{CF}$) $\circ_{CF}$ $\mathfrak{G}$)(⟦*ArrMap*⟧)
proof(*rule vsv-eqI*, *unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
fix *f* **assume** *f* $\in_\circ$ $\mathfrak{C}'$(⟦*Arr*⟧)
then obtain *a'* *b'* **where** *f* : *a'* $\mapsto_{\mathfrak{C}'}$ *b'* **by** (*auto intro*: *is-arrI*)
with *assms* **show**
(*cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ $\mathfrak{G}_{op\text{-}cat\ \mathfrak{B}',\mathfrak{C}'}(b,-)_{CF}$)(⟦*ArrMap*⟧)(⟦*f*⟧) =
((($\mathfrak{S}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(⟦ObjMap⟧)(⟦b⟧),-)_{CF}$) $\circ_{CF}$ $\mathfrak{G}$)(⟦*ArrMap*⟧)(⟦*f*⟧)
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed (*auto intro*: *is-functor.cf-ArrMap-vsv*)
qed *simp-all*

8.21 Composition of a covariant bifunctor and a covariant functor

8.21.1 Definition and elementary properties

definition *cf-lcomp* :: *V* ⇒ *V* ⇒ *V* ⇒ *V*
where *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$ = *cf-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ (*cf-id* $\mathfrak{C}$)

definition *cf-rcomp* :: *V* ⇒ *V* ⇒ *V* ⇒ *V*
where *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$ = *cf-bcomp* $\mathfrak{S}$ (*cf-id* $\mathfrak{B}$) $\mathfrak{G}$

Components.

lemma *cf-lcomp-components*:
shows *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(⟦*HomDom*⟧) = $\mathfrak{F}$(⟦*HomDom*⟧) $\times_C$ $\mathfrak{C}$
and *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(⟦*HomCod*⟧) = $\mathfrak{S}$(⟦*HomCod*⟧)
unfolding *cf-lcomp-def cf-bcomp-components dghm-id-components* **by** *simp-all*

lemma *cf-rcomp-components*:
shows *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$(⟦*HomDom*⟧) = $\mathfrak{B}$ $\times_C$ $\mathfrak{G}$(⟦*HomDom*⟧)
and *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$(⟦*HomCod*⟧) = $\mathfrak{S}$(⟦*HomCod*⟧)
unfolding *cf-rcomp-def cf-bcomp-components dghm-id-components* **by** *simp-all*

8.21.2 Object map

lemma *cf-lcomp-ObjMap-vsv*: *vsv* (*cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(⟦*ObjMap*⟧))
unfolding *cf-lcomp-def* **by** (*rule cf-bcomp-ObjMap-vsv*)

lemma *cf-rcomp-ObjMap-vsv*: *vsv* (*cf-rcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(|*ObjMap*|))
unfolding *cf-rcomp-def* **by** (*rule cf-bcomp-ObjMap-vsv*)

lemma *cf-lcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ$ (*cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(|*ObjMap*|)) = ($\mathfrak{A}$ $\times_C$ $\mathfrak{C}$)(|*Obj*|)
using *assms*
unfolding *cf-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-rcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$(|*ObjMap*|)) = ($\mathfrak{B}$ $\times_C$ $\mathfrak{A}$)(|*Obj*|)
using *assms*
unfolding *cf-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-lcomp-ObjMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $a \in_\circ \mathfrak{A}$(|*Obj*|)
and $c \in_\circ \mathfrak{C}$(|*Obj*|)
shows *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(|*ObjMap*|)(|a, c|)$_\bullet$ = $\mathfrak{S}$(|*ObjMap*|)(|$\mathfrak{F}$(|*ObjMap*|)(|a|), c|)$_\bullet$
using *assms*
unfolding *cf-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*)

lemma *cf-rcomp-ObjMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}$(|*Obj*|)
and $a \in_\circ \mathfrak{A}$(|*Obj*|)
shows *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$(|*ObjMap*|)(|b, a|)$_\bullet$ = $\mathfrak{S}$(|*ObjMap*|)(|b, $\mathfrak{G}$(|*ObjMap*|)(|a|)|)$_\bullet$
using *assms*
unfolding *cf-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*)

lemma *cf-lcomp-ObjMap-vrange*:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$(|*ObjMap*|)) $\subseteq_\circ$ $\mathfrak{D}$(|*Obj*|)
using *assms*
unfolding *cf-lcomp-def*
by (*intro cf-bcomp-ObjMap-vrange*) (*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-rcomp-ObjMap-vrange*:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$(|*ObjMap*|)) $\subseteq_\circ$ $\mathfrak{D}$(|*Obj*|)
using *assms*
unfolding *cf-rcomp-def*
by (*intro cf-bcomp-ObjMap-vrange*) (*cs-concl* **cs-intro**: *cat-cs-intros*)+

8.21.3 Arrow map

lemma *cf-lcomp-ArrMap-vsv*: *vsv* (*cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ArrMap|\!)$)
unfolding *cf-lcomp-def* **by** (*rule cf-bcomp-ArrMap-vsv*)

lemma *cf-rcomp-ArrMap-vsv*: *vsv* (*cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|ArrMap|\!)$)
unfolding *cf-rcomp-def* **by** (*rule cf-bcomp-ArrMap-vsv*)

lemma *cf-lcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ$ (*cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ArrMap|\!)$) = $(\mathfrak{A} \times_C \mathfrak{C})(\!|Arr|\!)$
using *assms*
unfolding *cf-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-rcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|ArrMap|\!)$) = $(\mathfrak{B} \times_C \mathfrak{A})(\!|Arr|\!)$
using *assms*
unfolding *cf-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-lcomp-ArrMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $f \in_\circ \mathfrak{A}(\!|Arr|\!)$
and $g \in_\circ \mathfrak{C}(\!|Arr|\!)$
shows *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ArrMap|\!)(\!|f,\ g|\!)_\bullet$ = $\mathfrak{S}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!),\ g|\!)_\bullet$
using *assms*
unfolding *cf-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*)

lemma *cf-rcomp-ArrMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
and $g \in_\circ \mathfrak{A}(\!|Arr|\!)$
shows *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|ArrMap|\!)(\!|f,\ g|\!)_\bullet$ = $\mathfrak{S}(\!|ArrMap|\!)(\!|f,\ \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!)|\!)_\bullet$
using *assms*
unfolding *cf-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*)

lemma *cf-lcomp-ArrMap-vrange*:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ArrMap|\!)$) $\subseteq_\circ$ $\mathfrak{D}(\!|Arr|\!)$
using *assms*
unfolding *cf-lcomp-def*
by (*intro cf-bcomp-ArrMap-vrange*) (*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-rcomp-ArrMap-vrange*:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|ArrMap|\!)$) $\subseteq_\circ$ $\mathfrak{D}(\!|Arr|\!)$
using *assms*
unfolding *cf-rcomp-def*

by (*intro cf-bcomp-ArrMap-vrange*) (*cs-concl* **cs-intro**: *cat-cs-intros*)+

8.21.4 Composition of a covariant bifunctor and a covariant functor is a functor

lemma *cf-lcomp-is-functor*:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F} : \mathfrak{A} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*
unfolding *cf-lcomp-def*
by (*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-lcomp-is-functor'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{A}' = \mathfrak{A} \times_C \mathfrak{C}$
shows *cf-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-lcomp-is-functor*)

lemma *cf-rcomp-is-functor*:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G} : \mathfrak{B} \times_C \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*
unfolding *cf-rcomp-def*
by (*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-rcomp-is-functor'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{A}' = \mathfrak{B} \times_C \mathfrak{A}$
shows *cf-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-rcomp-is-functor*)

8.22 Composition of a contracovariant bifunctor and a covariant functor

definition *cf-cn-cov-lcomp* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}$ = *cf-cn-cov-bcomp* $\mathfrak{S}$ $\mathfrak{F}$ (*cf-id* $\mathfrak{C}$)

definition *cf-cn-cov-rcomp* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}$ = *cf-cn-cov-bcomp* $\mathfrak{S}$ (*cf-id* $\mathfrak{B}$) $\mathfrak{G}$

Components.

lemma *cf-cn-cov-lcomp-components*:
shows *cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\lfloor HomDom \rfloor)$ = *op-cat* $(\mathfrak{F}(\lfloor HomDom \rfloor)) \times_C \mathfrak{C}$
and *cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\lfloor HomCod \rfloor) = \mathfrak{S}(\lfloor HomCod \rfloor)$
unfolding *cf-cn-cov-lcomp-def cf-cn-cov-bcomp-components dghm-id-components*
by *simp-all*

lemma *cf-cn-cov-rcomp-components*:
shows *cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\lfloor HomDom \rfloor)$ = *op-cat* $\mathfrak{B} \times_C \mathfrak{G}(\lfloor HomDom \rfloor)$
and *cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\lfloor HomCod \rfloor) = \mathfrak{S}(\lfloor HomCod \rfloor)$
unfolding *cf-cn-cov-rcomp-def cf-cn-cov-bcomp-components dghm-id-components*
by *simp-all*

8.22.1 Object map

lemma *cf-cn-cov-lcomp-ObjMap-vsv*: *vsv* (*cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ObjMap|\!)$)
unfolding *cf-cn-cov-lcomp-def* **by** (*rule cf-cn-cov-bcomp-ObjMap-vsv*)

lemma *cf-cn-cov-rcomp-ObjMap-vsv*: *vsv* (*cf-cn-cov-rcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ObjMap|\!)$)
unfolding *cf-cn-cov-rcomp-def* **by** (*rule cf-cn-cov-bcomp-ObjMap-vsv*)

lemma *cf-cn-cov-lcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ$ (*cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ObjMap|\!)$) = (*op-cat* $\mathfrak{A}$ $\times_C$ $\mathfrak{C}$)$(\!|Obj|\!)$
using *assms*
unfolding *cf-cn-cov-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-cn-cov-rcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|ObjMap|\!)$) = (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{A}$)$(\!|Obj|\!)$
using *assms*
unfolding *cf-cn-cov-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-cn-cov-lcomp-ObjMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $a \in_\circ$ *op-cat* $\mathfrak{A}(\!|Obj|\!)$
and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|a,\ c|\!)_\bullet$ = $\mathfrak{S}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!),\ c|\!)_\bullet$
using *assms*
unfolding *cf-cn-cov-lcomp-def cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)

lemma *cf-cn-cov-rcomp-ObjMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ$ *op-cat* $\mathfrak{B}(\!|Obj|\!)$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows *cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|ObjMap|\!)(\!|b,\ a|\!)_\bullet$ = $\mathfrak{S}(\!|ObjMap|\!)(\!|b,\ \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)|\!)_\bullet$
using *assms*
unfolding *cf-cn-cov-rcomp-def cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)

lemma *cf-cn-cov-lcomp-ObjMap-vrange*:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S}$: *op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|ObjMap|\!)$) $\subseteq_\circ$ $\mathfrak{D}(\!|Obj|\!)$
using *assms*

unfolding *cf-cn-cov-lcomp-def*
by (*intro cf-cn-cov-bcomp-ObjMap-vrange*)
(*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-cn-cov-rcomp-ObjMap-vrange*:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ$ (*cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|\mathit{ObjMap}|\!)$) $\subseteq_\circ \mathfrak{D}(\!|\mathit{Obj}|\!)$
using *assms*
unfolding *cf-cn-cov-rcomp-def*
by (*intro cf-cn-cov-bcomp-ObjMap-vrange*)
(*cs-concl* **cs-intro**: *cat-cs-intros*)+

8.22.2 Arrow map

lemma *cf-cn-cov-lcomp-ArrMap-vsv*: *vsv* (*cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|\mathit{ArrMap}|\!)$)
unfolding *cf-cn-cov-lcomp-def* **by** (*rule cf-cn-cov-bcomp-ArrMap-vsv*)

lemma *cf-cn-cov-rcomp-ArrMap-vsv*: *vsv* (*cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|\mathit{ArrMap}|\!)$)
unfolding *cf-cn-cov-rcomp-def* **by** (*rule cf-cn-cov-bcomp-ArrMap-vsv*)

lemma *cf-cn-cov-lcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ$ (*cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|\mathit{ArrMap}|\!)$) = (*op-cat* $\mathfrak{A} \times_C \mathfrak{C}$)$(\!|\mathit{Arr}|\!)$
using *assms*
unfolding *cf-cn-cov-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-cn-cov-rcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|\mathit{ArrMap}|\!)$) = (*op-cat* $\mathfrak{B} \times_C \mathfrak{A}$)$(\!|\mathit{Arr}|\!)$
using *assms*
unfolding *cf-cn-cov-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-cn-cov-lcomp-ArrMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $f \in_\circ \mathit{op\text{-}cat}\ \mathfrak{A}(\!|\mathit{Arr}|\!)$
and $g \in_\circ \mathfrak{C}(\!|\mathit{Arr}|\!)$
shows *cf-cn-cov-lcomp* $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{F}(\!|\mathit{ArrMap}|\!)(\!|f,\ g|\!)_\bullet$ = $\mathfrak{S}(\!|\mathit{ArrMap}|\!)(\!|\mathfrak{F}(\!|\mathit{ArrMap}|\!)(\!|f|\!),\ g|\!)_\bullet$
using *assms*
unfolding *cf-cn-cov-lcomp-def cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)

lemma *cf-cn-cov-rcomp-ArrMap-app*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $f \in_\circ \mathit{op\text{-}cat}\ \mathfrak{B}(\!|\mathit{Arr}|\!)$
and $g \in_\circ \mathfrak{A}(\!|\mathit{Arr}|\!)$
shows *cf-cn-cov-rcomp* $\mathfrak{B}$ $\mathfrak{S}$ $\mathfrak{G}(\!|\mathit{ArrMap}|\!)(\!|f,\ g|\!)_\bullet$ = $\mathfrak{S}(\!|\mathit{ArrMap}|\!)(\!|f,\ \mathfrak{G}(\!|\mathit{ArrMap}|\!)(\!|g|\!)|\!)_\bullet$
using *assms*

unfolding *cf-cn-cov-rcomp-def cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)

lemma *cf-cn-cov-lcomp-ArrMap-vrange*:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ\ (\mathit{cf\text{-}cn\text{-}cov\text{-}lcomp}\ \mathfrak{C}\ \mathfrak{S}\ \mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor)) \subseteq_\circ \mathfrak{D}(\lfloor \mathit{Arr} \rfloor)$
using *assms*
unfolding *cf-cn-cov-lcomp-def*
by (*intro cf-cn-cov-bcomp-ArrMap-vrange*)
(*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-cn-cov-rcomp-ArrMap-vrange*:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathcal{R}_\circ\ (\mathit{cf\text{-}cn\text{-}cov\text{-}rcomp}\ \mathfrak{B}\ \mathfrak{S}\ \mathfrak{G}(\lfloor \mathit{ArrMap} \rfloor)) \subseteq_\circ \mathfrak{D}(\lfloor \mathit{Arr} \rfloor)$
using *assms*
unfolding *cf-cn-cov-rcomp-def cat-op-simps*
by (*intro cf-cn-cov-bcomp-ArrMap-vrange*)
(*cs-concl* **cs-intro**: *cat-cs-intros*)+

8.22.3 Composition of a contracovariant bifunctor and a covariant functor is a functor

lemma *cf-cn-cov-lcomp-is-functor*:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathit{cf\text{-}cn\text{-}cov\text{-}lcomp}\ \mathfrak{C}\ \mathfrak{S}\ \mathfrak{F} : \mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*
unfolding *cf-cn-cov-lcomp-def cat-op-simps*
by (*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-cn-cov-lcomp-is-functor'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{AC} = \mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{C}$
shows $\mathit{cf\text{-}cn\text{-}cov\text{-}lcomp}\ \mathfrak{C}\ \mathfrak{S}\ \mathfrak{F} : \mathfrak{AC} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-cn-cov-lcomp-is-functor*)

lemma *cf-cn-cov-rcomp-is-functor*:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
shows $\mathit{cf\text{-}cn\text{-}cov\text{-}rcomp}\ \mathfrak{B}\ \mathfrak{S}\ \mathfrak{G} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*
unfolding *cf-cn-cov-rcomp-def cat-op-simps*
by (*cs-concl* **cs-intro**: *cat-cs-intros*)+

lemma *cf-cn-cov-rcomp-is-functor'*[*cat-cs-intros*]:

assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $\mathfrak{BA} = \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{A}$
shows *cf-cn-cov-rcomp* $\mathfrak{B}\ \mathfrak{S}\ \mathfrak{G} : \mathfrak{BA} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
using *assms*(*1–3*) **unfolding** *assms*(*4*) **by** (*rule cf-cn-cov-rcomp-is-functor*)

8.22.4 Projection of a composition of a contracovariant bifunctor and a covariant functor

lemma *cf-cn-cov-rcomp-bifunctor-proj-snd*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $b \in_\circ \mathfrak{B}(\!|\mathit{Obj}|\!)$
shows
cf-cn-cov-rcomp $\mathfrak{B}\ \mathfrak{S}\ \mathfrak{G}_{\mathit{op\text{-}cat}\ \mathfrak{B},\mathfrak{A}}(b,-)_{CF} =$
$(\mathfrak{S}_{\mathit{op\text{-}cat}\ \mathfrak{B},\mathfrak{C}}(b,-)_{CF}) \circ_{CF} \mathfrak{G}$
using *assms*
unfolding *cf-cn-cov-rcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-cn-cov-lcomp-bifunctor-proj-snd*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{S} : \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
and $b \in_\circ \mathfrak{A}(\!|\mathit{Obj}|\!)$
shows
cf-cn-cov-lcomp $\mathfrak{C}\ \mathfrak{S}\ \mathfrak{F}_{\mathit{op\text{-}cat}\ \mathfrak{A},\mathfrak{C}}(b,-)_{CF} =$
$(\mathfrak{S}_{\mathit{op\text{-}cat}\ \mathfrak{B},\mathfrak{C}}(\mathfrak{F}(\!|\mathit{ObjMap}|\!)(\!|b|\!),-)_{CF})$
using *assms*
unfolding *cf-cn-cov-lcomp-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*)

8.23 Composition of bifunctors

8.23.1 Definitions and elementary properties

definition *cf-blcomp* :: $V \Rightarrow V$
where *cf-blcomp* $\mathfrak{S} =$
cf-lcomp $(\mathfrak{S}(\!|\mathit{HomCod}|\!))\ \mathfrak{S}\ \mathfrak{S} \circ_{CF}$
cf-cat-prod-21-of-3 $(\mathfrak{S}(\!|\mathit{HomCod}|\!))\ (\mathfrak{S}(\!|\mathit{HomCod}|\!))\ (\mathfrak{S}(\!|\mathit{HomCod}|\!))$

definition *cf-brcomp* :: $V \Rightarrow V$
where *cf-brcomp* $\mathfrak{S} =$
cf-rcomp $(\mathfrak{S}(\!|\mathit{HomCod}|\!))\ \mathfrak{S}\ \mathfrak{S} \circ_{CF}$
cf-cat-prod-12-of-3 $(\mathfrak{S}(\!|\mathit{HomCod}|\!))\ (\mathfrak{S}(\!|\mathit{HomCod}|\!))\ (\mathfrak{S}(\!|\mathit{HomCod}|\!))$

Alternative forms of the definitions.

lemma *cf-blcomp-def'*:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *cf-blcomp* $\mathfrak{S} =$ *cf-lcomp* $\mathfrak{C}\ \mathfrak{S}\ \mathfrak{S} \circ_{CF}$ *cf-cat-prod-21-of-3* $\mathfrak{C}\ \mathfrak{C}\ \mathfrak{C}$
proof–
interpret $\mathfrak{S}$: *is-functor* α $\langle\mathfrak{C} \times_C \mathfrak{C}\rangle\ \mathfrak{C}\ \mathfrak{S}$ **by** (*rule assms*)
show *?thesis*
by
(
cs-concl **cs-shallow**

```
      cs-simp: cat-cs-simps cf-blcomp-def cs-intro: cat-cs-intros
    )
qed

lemma cf-brcomp-def':
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ
  shows cf-brcomp 𝔖 = cf-rcomp ℭ 𝔖 𝔖 ∘_CF cf-cat-prod-12-of-3 ℭ ℭ ℭ
proof–
  interpret 𝔖: is-functor α ‹ℭ ×_C ℭ› ℭ 𝔖 by (rule assms)
  show ?thesis
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cf-brcomp-def cs-intro: cat-cs-intros
      )
qed
```

8.23.2 Compositions of bifunctors are functors

```
lemma cf-blcomp-is-functor:
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ
  shows cf-blcomp 𝔖 : ℭ ×_C3 ℭ ×_C3 ℭ ↦↦_Cα ℭ
proof–
  interpret 𝔖: is-functor α ‹ℭ ×_C ℭ› ℭ 𝔖 by (rule assms)
  show ?thesis
    by (cs-concl cs-simp: cat-cs-simps cf-blcomp-def' cs-intro: cat-cs-intros)
qed

lemma cf-blcomp-is-functor'[cat-cs-intros]:
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ and 𝔄' = ℭ ×_C3 ℭ ×_C3 ℭ
  shows cf-blcomp 𝔖 : 𝔄' ↦↦_Cα ℭ
  using assms(1) unfolding assms(2) by (rule cf-blcomp-is-functor)

lemma cf-brcomp-is-functor:
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ
  shows cf-brcomp 𝔖 : ℭ ×_C3 ℭ ×_C3 ℭ ↦↦_Cα ℭ
proof–
  interpret 𝔖: is-functor α ‹ℭ ×_C ℭ› ℭ 𝔖 by (rule assms)
  show ?thesis
    by (cs-concl cs-simp: cat-cs-simps cf-brcomp-def' cs-intro: cat-cs-intros)
qed

lemma cf-brcomp-is-functor'[cat-cs-intros]:
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ and 𝔄' = ℭ ×_C3 ℭ ×_C3 ℭ
  shows cf-brcomp 𝔖 : 𝔄' ↦↦_Cα ℭ
  using assms(1) unfolding assms(2) by (rule cf-brcomp-is-functor)
```

8.23.3 Object map

```
lemma cf-blcomp-ObjMap-vsv[cat-cs-intros]:
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ
  shows vsv (cf-blcomp 𝔖(|ObjMap|))
proof–
  interpret cf-blcomp: is-functor α ‹ℭ ×_C3 ℭ ×_C3 ℭ› ℭ ‹cf-blcomp 𝔖›
    by (rule cf-blcomp-is-functor[OF assms])
  show ?thesis by auto
qed
```

lemma *cf-brcomp-ObjMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *vsv* (*cf-brcomp* $\mathfrak{S}(\!|ObjMap|\!)$)
proof–
interpret *cf-brcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-brcomp* $\mathfrak{S}$›
by (*rule cf-brcomp-is-functor*[*OF assms*])
show *?thesis* **by** *auto*
qed

lemma *cf-blcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-blcomp* $\mathfrak{S}(\!|ObjMap|\!)$) = $(\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C})(\!|Obj|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{C} \times_C \mathfrak{C}$› $\mathfrak{C}$ $\mathfrak{S}$ **by** (*rule assms*)
interpret *cf-blcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-blcomp* $\mathfrak{S}$›
by (*rule cf-blcomp-is-functor*[*OF assms*])
show *?thesis*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *cf-brcomp-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-brcomp* $\mathfrak{S}(\!|ObjMap|\!)$) = $(\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C})(\!|Obj|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{C} \times_C \mathfrak{C}$› $\mathfrak{C}$ $\mathfrak{S}$ **by** (*rule assms*)
interpret *cf-brcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-brcomp* $\mathfrak{S}$›
by (*rule cf-brcomp-is-functor*[*OF assms*])
show *?thesis*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *cf-blcomp-ObjMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $A = [a, b, c]_\circ$
and $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *cf-blcomp* $\mathfrak{S}(\!|ObjMap|\!)(\!|A|\!)$ = $(a \otimes_{HM.O\mathfrak{S}} b) \otimes_{HM.O\mathfrak{S}} c$
proof–
interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{C} \times_C \mathfrak{C}$› $\mathfrak{C}$ $\mathfrak{S}$ **by** (*rule assms*)
interpret *cf-blcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-blcomp* $\mathfrak{S}$›
by (*rule cf-blcomp-is-functor*[*OF assms*(*1*)])
from *assms*(*3–5*) **show** *?thesis*
unfolding *assms*(*2*)
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-prod-cs-simps cf-blcomp-def'*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed

lemma *cf-brcomp-ObjMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $A = [a, b, c]_\circ$
and $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$

shows *cf-brcomp* $\mathfrak{S}(\!|ObjMap|\!)(\!|A|\!) = a \otimes_{HM.O\mathfrak{S}} (b \otimes_{HM.O\mathfrak{S}} c)$
proof–
interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{C} \times_C \mathfrak{C}$› $\mathfrak{C}$ $\mathfrak{S}$ **by** (*rule assms*)
interpret *cf-brcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-brcomp* $\mathfrak{S}$›
by (*rule cf-brcomp-is-functor*[*OF assms*(*1*)])
from *assms*(*3–5*) **show** *?thesis*
unfolding *assms*(*2*)
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-prod-cs-simps cf-brcomp-def'*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed

8.23.4 Arrow map

lemma *cf-blcomp-ArrMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *vsv* (*cf-blcomp* $\mathfrak{S}(\!|ArrMap|\!)$)
proof–
interpret *cf-blcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-blcomp* $\mathfrak{S}$›
by (*rule cf-blcomp-is-functor*[*OF assms*])
show *?thesis* **by** *auto*
qed

lemma *cf-brcomp-ArrMap-vsv*[*cat-cs-intros*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *vsv* (*cf-brcomp* $\mathfrak{S}(\!|ArrMap|\!)$)
proof–
interpret *cf-brcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-brcomp* $\mathfrak{S}$›
by (*rule cf-brcomp-is-functor*[*OF assms*])
show *?thesis* **by** *auto*
qed

lemma *cf-blcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-blcomp* $\mathfrak{S}(\!|ArrMap|\!)$) = $(\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C})(\!|Arr|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{C} \times_C \mathfrak{C}$› $\mathfrak{C}$ $\mathfrak{S}$ **by** (*rule assms*)
interpret *cf-blcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-blcomp* $\mathfrak{S}$›
by (*rule cf-blcomp-is-functor*[*OF assms*])
show *?thesis*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *cf-brcomp-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{S} : \mathfrak{C} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*cf-brcomp* $\mathfrak{S}(\!|ArrMap|\!)$) = $(\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C})(\!|Arr|\!)$
proof–
interpret $\mathfrak{S}$: *is-functor* α ‹$\mathfrak{C} \times_C \mathfrak{C}$› $\mathfrak{C}$ $\mathfrak{S}$ **by** (*rule assms*)
interpret *cf-brcomp*: *is-functor* α ‹$\mathfrak{C} \times_{C3} \mathfrak{C} \times_{C3} \mathfrak{C}$› $\mathfrak{C}$ ‹*cf-brcomp* $\mathfrak{S}$›
by (*rule cf-brcomp-is-functor*[*OF assms*])
show *?thesis*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *cf-blcomp-ArrMap-app*[*cat-cs-simps*]:

```
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ
    and F = [h, g, f]∘
    and h ∈∘ ℭ⦇Arr⦈
    and g ∈∘ ℭ⦇Arr⦈
    and f ∈∘ ℭ⦇Arr⦈
  shows cf-blcomp 𝔖⦇ArrMap⦈⦇F⦈ = (h ⊗_HM.A𝔖 g) ⊗_HM.A𝔖 f
proof–
  interpret 𝔖: is-functor α ‹ℭ ×_C ℭ› ℭ 𝔖 by (rule assms)
  interpret cf-blcomp: is-functor α ‹ℭ ×_C3 ℭ ×_C3 ℭ› ℭ ‹cf-blcomp 𝔖›
    by (rule cf-blcomp-is-functor[OF assms(1)])
  from assms(3–5) show ?thesis
    unfolding assms(2)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-prod-cs-simps cf-blcomp-def'
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
qed

lemma cf-brcomp-ArrMap-app[cat-cs-simps]:
  assumes 𝔖 : ℭ ×_C ℭ ↦↦_Cα ℭ
    and F = [h, g, f]∘
    and h ∈∘ ℭ⦇Arr⦈
    and g ∈∘ ℭ⦇Arr⦈
    and f ∈∘ ℭ⦇Arr⦈
  shows cf-brcomp 𝔖⦇ArrMap⦈⦇F⦈ = h ⊗_HM.A𝔖 (g ⊗_HM.A𝔖 f)
proof–
  interpret 𝔖: is-functor α ‹ℭ ×_C ℭ› ℭ 𝔖 by (rule assms)
  interpret cf-brcomp: is-functor α ‹ℭ ×_C3 ℭ ×_C3 ℭ› ℭ ‹cf-brcomp 𝔖›
    by (rule cf-brcomp-is-functor[OF assms(1)])
  from assms(3–5) show ?thesis
    unfolding assms(2)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-prod-cs-simps cf-brcomp-def'
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
qed
```

8.24 Binatural transformation

8.24.1 Definitions and elementary properties

In this work, a *binatural transformation* is used to denote a natural transformation of bifunctors.

```
definition bnt-proj-fst :: V ⇒ V ⇒ V ⇒ V ⇒ V
  (‹(-,-/'(/-,-/'))/_NTCF›) [51, 51, 51, 51] 51)
  where 𝔑_𝔄,𝔅(-,b)_NTCF =
    [
      (λa∈∘𝔄⦇Obj⦈. 𝔑⦇NTMap⦈⦇a, b⦈•),
      𝔑⦇NTDom⦈_𝔄,𝔅(-,b)_CF,
      𝔑⦇NTCod⦈_𝔄,𝔅(-,b)_CF,
      𝔄,
      𝔑⦇NTDGCod⦈
    ]∘

definition bnt-proj-snd :: V ⇒ V ⇒ V ⇒ V ⇒ V
```

$(\langle(\text{-},\text{-}/'(/\text{-},-/')/_{NTCF})\rangle\ [51,\ 51,\ 51,\ 51]\ 51)$
where $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF}$ =
[
$(\lambda b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor).\ \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ b \rfloor_\bullet)$,
$\mathfrak{N}(\lfloor NTDom \rfloor)_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$,
$\mathfrak{N}(\lfloor NTCod \rfloor)_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$,
$\mathfrak{B}$,
$\mathfrak{N}(\lfloor NTDGCod \rfloor)$
]$_\circ$

Components

lemma *bnt-proj-fst-components*:
shows $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\lfloor NTMap \rfloor) = (\lambda a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor).\ \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ b \rfloor_\bullet)$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\lfloor NTDom \rfloor) = \mathfrak{N}(\lfloor NTDom \rfloor)_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\lfloor NTCod \rfloor) = \mathfrak{N}(\lfloor NTCod \rfloor)_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\lfloor NTDGCod \rfloor) = \mathfrak{N}(\lfloor NTDGCod \rfloor)$
unfolding *bnt-proj-fst-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *bnt-proj-snd-components*:
shows $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\lfloor NTMap \rfloor) = (\lambda b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor).\ \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ b \rfloor_\bullet)$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\lfloor NTDom \rfloor) = \mathfrak{N}(\lfloor NTDom \rfloor)_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\lfloor NTCod \rfloor) = \mathfrak{N}(\lfloor NTCod \rfloor)_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\lfloor NTDGDom \rfloor) = \mathfrak{B}$
and $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\lfloor NTDGCod \rfloor) = \mathfrak{N}(\lfloor NTDGCod \rfloor)$
unfolding *bnt-proj-snd-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

8.24.2 Natural transformation maps

mk-VLambda *bnt-proj-fst-components*(*1*)[*folded VLambda-vconst-on*]
|*vsv bnt-proj-fst-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain bnt-proj-fst-NTMap-vdomain*[*cat-cs-simps*]|
|*app bnt-proj-fst-NTMap-app*[*cat-cs-simps*]|

lemma *bnt-proj-fst-vrange*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
shows $\mathcal{R}_\circ\ ((\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\lfloor NTMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof–
interpret $\mathfrak{N}$: *is-ntcf* α $\langle\mathfrak{A} \times_C \mathfrak{B}\rangle$ $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{S}'$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
show *?thesis*
unfolding *bnt-proj-fst-components*
proof(*rule vrange-VLambda-vsubset*)
fix *a* **assume** $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
with *assms* **show** $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ b \rfloor_\bullet \in_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-prod-cs-intros*)
qed
qed

mk-VLambda *bnt-proj-snd-components*(*1*)[*folded VLambda-vconst-on*]
|*vsv bnt-proj-snd-NTMap-vsv*[*intro*]|
|*vdomain bnt-proj-snd-NTMap-vdomain*[*cat-cs-simps*]|
|*app bnt-proj-snd-NTMap-app*[*cat-cs-simps*]|

lemma *bnt-proj-snd-vrange*:

assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\mathcal{R}_\circ\ ((\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\!|NTMap|\!)) \subseteq_\circ \mathfrak{C}(\!|Arr|\!)$
proof–
interpret $\mathfrak{N}$: *is-ntcf* α $\langle \mathfrak{A} \times_C \mathfrak{B} \rangle$ $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{S}'$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
show *?thesis*
unfolding *bnt-proj-snd-components*
proof(*rule vrange-VLambda-vsubset*)
fix b **assume** $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
with *assms* **show** $\mathfrak{N}(\!|NTMap|\!)(\!|a,\ b|\!)_\bullet \in_\circ \mathfrak{C}(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-prod-cs-intros*)
qed
qed

8.24.3 Binatural transformation projection is a natural transformation

lemma *bnt-proj-snd-is-ntcf*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF} : \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} \mapsto_{CF} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{N}$: *is-ntcf* α $\langle \mathfrak{A} \times_C \mathfrak{B} \rangle$ $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{S}'$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
show *?thesis*
proof(*intro is-ntcfI'*)
show *vfsequence* $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})$ **unfolding** *bnt-proj-snd-def* **by** *simp*
show *vcard* $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF}) = 5_{\mathbb{N}}$
unfolding *bnt-proj-snd-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms* **show** $\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\!|NTMap|\!)(\!|b|\!)$:
$(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)(\!|b|\!) \mapsto_{\mathfrak{C}} (\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)(\!|b|\!)$
if $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **for** b
using *that assms*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
show $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\!|NTMap|\!)(\!|b|\!) \circ_{A\mathfrak{C}} (\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ArrMap|\!)(\!|f|\!) =$
$(\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{C}} (\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\!|NTMap|\!)(\!|a'|\!)$
if $f : a' \mapsto_{\mathfrak{B}} b$ **for** a' b f
using *that assms*
by
(
cs-concl
cs-simp: *is-ntcf.ntcf-Comp-commute cat-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed (*auto simp*: *bnt-proj-snd-components cat-cs-simps*)

qed

lemma *bnt-proj-snd-is-ntcf'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $\mathfrak{F} = \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$
and $\mathfrak{G} = \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$
shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
using *assms* **by** (*auto intro*: *bnt-proj-snd-is-ntcf*)

lemma *bnt-proj-fst-is-ntcf*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF} : \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} \mapsto_{CF} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{N}$: *is-ntcf* α $\langle\mathfrak{A} \times_C \mathfrak{B}\rangle$ $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{S}'$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
show *?thesis*
proof(*intro is-ntcfI'*)
show *vfsequence* $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})$ **unfolding** *bnt-proj-fst-def* **by** *simp*
show *vcard* $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF}) = 5_{\mathbb{N}}$
unfolding *bnt-proj-fst-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **show** $\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\!|NTMap|\!)(\!|a|\!) :$
$(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} (\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ObjMap|\!)(\!|a|\!)$
if $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **for** a
using *that assms*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
show $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\!|NTMap|\!)(\!|b'|\!) \circ_{A\mathfrak{C}} (\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ArrMap|\!)(\!|f|\!) =$
$(\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{C}} (\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\!|NTMap|\!)(\!|a|\!)$
if $f : a \mapsto_{\mathfrak{A}} b'$ **for** a b' f
using *that assms*
by
(
cs-concl
cs-simp: *is-ntcf.ntcf-Comp-commute cat-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed (*auto simp*: *bnt-proj-fst-components cat-cs-simps*)
qed

lemma *bnt-proj-fst-is-ntcf'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

and $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
and $\mathfrak{F} = \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
and $\mathfrak{G} = \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
and $\mathfrak{A}' = \mathfrak{A}$
shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
using *assms*(*1–4*) **unfolding** *assms*(*5–7*) **by** (*rule bnt-proj-fst-is-ntcf*)

8.24.4 Array binatural transformation is a natural transformation

lemma *ntcf-array-is-ntcf*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{S} : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and *vfsequence* $\mathfrak{N}$
and *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
and $\mathfrak{N}(\!|NTDom|\!) = \mathfrak{S}$
and $\mathfrak{N}(\!|NTCod|\!) = \mathfrak{S}'$
and $\mathfrak{N}(\!|NTDGDom|\!) = \mathfrak{A} \times_C \mathfrak{B}$
and $\mathfrak{N}(\!|NTDGCod|\!) = \mathfrak{C}$
and *vsv* $(\mathfrak{N}(\!|NTMap|\!))$
and $\mathcal{D}_{\circ}(\mathfrak{N}(\!|NTMap|\!)) = (\mathfrak{A} \times_C \mathfrak{B})(\!|Obj|\!)$
and $\bigwedge a\ b.\ [\![\ a \in_{\circ} \mathfrak{A}(\!|Obj|\!);\ b \in_{\circ} \mathfrak{B}(\!|Obj|\!)\ ]\!] \Longrightarrow$
$\mathfrak{N}(\!|NTMap|\!)(\!|a,\ b|\!)_{\bullet} : \mathfrak{S}(\!|ObjMap|\!)(\!|a,\ b|\!)_{\bullet} \mapsto_{\mathfrak{C}} \mathfrak{S}'(\!|ObjMap|\!)(\!|a,\ b|\!)_{\bullet}$
and $\bigwedge a.\ a \in_{\circ} \mathfrak{A}(\!|Obj|\!) \Longrightarrow$
$\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF} : \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} \mapsto_{CF} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\bigwedge b.\ b \in_{\circ} \mathfrak{B}(\!|Obj|\!) \Longrightarrow$
$\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF} : \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} \mapsto_{CF} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{N}$: *vsv* $\langle \mathfrak{N}(\!|NTMap|\!) \rangle$ **by** (*rule assms*(*11*))

have [*cat-cs-intros*]:
$[\![\ a \in_{\circ} \mathfrak{A}(\!|Obj|\!);\ b \in_{\circ} \mathfrak{B}(\!|Obj|\!);\ A = \mathfrak{S}(\!|ObjMap|\!)(\!|a,\ b|\!)_{\bullet};\ B = \mathfrak{S}'(\!|ObjMap|\!)(\!|a,\ b|\!)_{\bullet}\ ]\!] \Longrightarrow$
$\mathfrak{N}(\!|NTMap|\!)(\!|a,\ b|\!)_{\bullet} : A \mapsto_{\mathfrak{C}} B$
for *a b A B*
by (*auto intro*: *assms*(*13*))

show *?thesis*
proof(*intro is-ntcfI'*)

show $\mathfrak{N}(\!|NTMap|\!)(\!|ab|\!) : \mathfrak{S}(\!|ObjMap|\!)(\!|ab|\!) \mapsto_{\mathfrak{C}} \mathfrak{S}'(\!|ObjMap|\!)(\!|ab|\!)$
if $ab \in_{\circ} (\mathfrak{A} \times_C \mathfrak{B})(\!|Obj|\!)$ **for** *ab*
proof–
from *that* **obtain** *a b*
where *ab-def*: $ab = [a,\ b]_{\circ}$ **and** *a*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ **and** *b*: $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
by (*elim cat-prod-2-ObjE*[*OF assms*(*1,2*)])
from *a b* **show** *?thesis* **unfolding** *ab-def* **by** (*rule assms*(*13*))
qed

show
$\mathfrak{N}(\!|NTMap|\!)(\!|a'b'|\!) \circ_{A\mathfrak{C}} \mathfrak{S}(\!|ArrMap|\!)(\!|gf|\!) = \mathfrak{S}'(\!|ArrMap|\!)(\!|gf|\!) \circ_{A\mathfrak{C}} \mathfrak{N}(\!|NTMap|\!)(\!|ab|\!)$
if $gf : ab \mapsto_{\mathfrak{A} \times_C \mathfrak{B}} a'b'$ **for** *ab a'b' gf*
proof–
from *that* **obtain** *g f a b a' b'*

```
  where gf-def: gf = [g, f]∘
    and ab-def: ab = [a, b]∘
    and a'b'-def: a'b' = [a', b']∘
    and g: g : a ↦⇘𝔄⇙ a'
    and f: f : b ↦⇘𝔅⇙ b'
  by (elim cat-prod-2-is-arrE[OF assms(1,2)])
then have a: a ∈∘ 𝔄(|Obj|)
  and a': a' ∈∘ 𝔄(|Obj|)
  and b: b ∈∘ 𝔅(|Obj|)
  and b': b' ∈∘ 𝔅(|Obj|)
  by auto
show ?thesis
  unfolding gf-def ab-def a'b'-def
proof–
  from is-ntcfD'(13)[OF assms(15)[OF b] g] g f assms(1,2,3,4)
  have [cat-cs-simps]:
    (𝔖'(|ArrMap|)(|g, 𝔅(|CId|)(|b|)|)• ∘⇘A ℭ⇙ 𝔑(|NTMap|)(|a, b|)•) =
      (𝔑(|NTMap|)(|a', b|)• ∘⇘A ℭ⇙ 𝔖(|ArrMap|)(|g, 𝔅(|CId|)(|b|)|)•)
    by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros) auto
  from is-ntcfD'(13)[OF assms(14)[OF a'] f] g f assms(1,2)
  have 𝔖'𝔑:
    𝔖'(|ArrMap|)(|𝔄(|CId|)(|a'|), f|)• ∘⇘A ℭ⇙ 𝔑(|NTMap|)(|a', b|)• =
      𝔑(|NTMap|)(|a', b'|)• ∘⇘A ℭ⇙ 𝔖(|ArrMap|)(|𝔄(|CId|)(|a'|),f|)•
    by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros) auto
  from g f assms(1–4) have [cat-cs-simps]:
    𝔖'(|ArrMap|)(|𝔄(|CId|)(|a'|), f|)• ∘⇘A ℭ⇙ (𝔑(|NTMap|)(|a', b|)• ∘⇘A ℭ⇙ q) =
      𝔑(|NTMap|)(|a', b'|)• ∘⇘A ℭ⇙ (𝔖(|ArrMap|)(|𝔄(|CId|)(|a'|),f|)• ∘⇘A ℭ⇙ q)
    if q : r ↦⇘ℭ⇙ 𝔖(|ObjMap|)(|a', b|)• for q r
    using that
    by
      (
        cs-concl
          cs-simp: 𝔖'𝔑 category.cat-Comp-assoc[symmetric]
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )

  from assms(1–4) g f have
    𝔖'(|ArrMap|)(|𝔄(|CId|)(|a'|), f|)• ∘⇘A ℭ⇙ 𝔖'(|ArrMap|)(|g, 𝔅(|CId|)(|b|)|)• =
      𝔖'(|ArrMap|)(|[𝔄(|CId|)(|a'|), f]∘ ∘⇘A 𝔄 ×C 𝔅⇙ [g, 𝔅(|CId|)(|b|)]∘|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  also from assms(1–4) g f have ... = 𝔖'(|ArrMap|)(|g, f|)•
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-prod-cs-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  finally have 𝔖'-gf: 𝔖'(|ArrMap|)(|g, f|)• =
    𝔖'(|ArrMap|)(|𝔄(|CId|)(|a'|), f|)• ∘⇘A ℭ⇙ 𝔖'(|ArrMap|)(|g, 𝔅(|CId|)(|b|)|)•
    by simp
  from assms(1–4) g f have
    𝔖(|ArrMap|)(|𝔄(|CId|)(|a'|), f|)• ∘⇘A ℭ⇙ 𝔖(|ArrMap|)(|g, 𝔅(|CId|)(|b|)|)• =
      𝔖(|ArrMap|)(|[𝔄(|CId|)(|a'|), f]∘ ∘⇘A 𝔄 ×C 𝔅⇙ [g, 𝔅(|CId|)(|b|)]∘|)
    by
```

(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
also from *assms*(*1–4*) *g f* **have** ... = $\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_{\bullet}$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-prod-cs-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
finally have $\mathfrak{S}$-*gf*: $\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_{\bullet}$ =
$\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{A}(\lfloor CId \rfloor)(\lfloor a' \rfloor), f \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, \mathfrak{B}(\lfloor CId \rfloor)(\lfloor b \rfloor) \rfloor)_{\bullet}$
by *simp*
from *assms*(*1–4*) *g f assms*(*13*)[*OF a b*] *assms*(*13*)[*OF a′ b*] **have**
$\mathfrak{S}'(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a, b \rfloor)_{\bullet}$ =
$(\mathfrak{S}'(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{A}(\lfloor CId \rfloor)(\lfloor a' \rfloor), f \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a', b \rfloor)_{\bullet}) \circ_{A\mathfrak{C}}$
$\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, \mathfrak{B}(\lfloor CId \rfloor)(\lfloor b \rfloor) \rfloor)_{\bullet}$
unfolding $\mathfrak{S}'$-*gf*
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
also from *assms*(*1–4*) *g f* **have**
... = $(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a', b' \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{A}(\lfloor CId \rfloor)(\lfloor a' \rfloor), f \rfloor)_{\bullet}) \circ_{A\mathfrak{C}}$
$\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, \mathfrak{B}(\lfloor CId \rfloor)(\lfloor b \rfloor) \rfloor)_{\bullet}$
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
also from *assms*(*1–4*) *g f assms*(*13*)[*OF a′ b′*] **have**
... = $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a', b' \rfloor)_{\bullet} \circ_{A\mathfrak{C}}$
$(\mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{A}(\lfloor CId \rfloor)(\lfloor a' \rfloor), f \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, \mathfrak{B}(\lfloor CId \rfloor)(\lfloor b \rfloor) \rfloor)_{\bullet})$
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
also from *assms*(*1–4*) *g f assms*(*13*)[*OF a′ b′*] **have**
... = $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a', b' \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_{\bullet}$
unfolding $\mathfrak{S}$-*gf*[*symmetric*] **by** *simp*
finally show
$\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a', b' \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{S}(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_{\bullet}$ =
$\mathfrak{S}'(\lfloor ArrMap \rfloor)(\lfloor g, f \rfloor)_{\bullet} \circ_{A\mathfrak{C}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a, b \rfloor)_{\bullet}$
by *simp*
qed
qed

qed (*auto simp*: *assms*)

qed

8.24.5 Binatural transformation projections and isomorphisms

lemma *is-iso-ntcf-if-bnt-proj-snd-is-iso-ntcf*:
assumes *category* α $\mathfrak{A}$

```
    and category α 𝔅
    and 𝔑 : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    and ⋀a. a ∈₀ 𝔄(|Obj|) ⟹
      𝔑_𝔄,𝔅(a,-)_NTCF : 𝔖_𝔄,𝔅(a,-)_CF ↦_CF.iso 𝔖'_𝔄,𝔅(a,-)_CF : 𝔅 ↦↦_Cα ℭ
  shows 𝔑 : 𝔖 ↦_CF.iso 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
proof–
  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔅: category α 𝔅 by (rule assms(2))
  show ?thesis
  proof(intro is-iso-ntcfI)
    show 𝔑 : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ by (rule assms(3))
    fix ab assume ab ∈₀ (𝔄 ×_C 𝔅)(|Obj|)
    then obtain a b
      where ab-def: ab = [a, b]₀ and a: a ∈₀ 𝔄(|Obj|) and b: b ∈₀ 𝔅(|Obj|)
      by (elim cat-prod-2-ObjE[OF assms(1,2)])
    interpret 𝔑a: is-iso-ntcf
      α 𝔅 ℭ ‹𝔖_𝔄,𝔅(a,-)_CF› ‹𝔖'_𝔄,𝔅(a,-)_CF› ‹𝔑_𝔄,𝔅(a,-)_NTCF›
      by (rule assms(4)[OF a])
    from b have 𝔑ab: 𝔑(|NTMap|)(|a, b|)• = (𝔑_𝔄,𝔅(a,-)_NTCF)(|NTMap|)(|b|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from 𝔑a.iso-ntcf-is-iso-arr[OF b] assms(1,2,3) a b show
      𝔑(|NTMap|)(|ab|) : 𝔖(|ObjMap|)(|ab|) ↦_iso ℭ 𝔖'(|ObjMap|)(|ab|)
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps ab-def cs-intro: cat-prod-cs-intros
        )
  qed
qed

lemma is-iso-ntcf-if-bnt-proj-fst-is-iso-ntcf:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    and ⋀b. b ∈₀ 𝔅(|Obj|) ⟹
      𝔑_𝔄,𝔅(-,b)_NTCF : 𝔖_𝔄,𝔅(-,b)_CF ↦_CF.iso 𝔖'_𝔄,𝔅(-,b)_CF : 𝔄 ↦↦_Cα ℭ
  shows 𝔑 : 𝔖 ↦_CF.iso 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
proof–
  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔅: category α 𝔅 by (rule assms(2))
  show ?thesis
  proof(intro is-iso-ntcfI)
    show 𝔑 : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ by (rule assms(3))
    fix ab assume ab ∈₀ (𝔄 ×_C 𝔅)(|Obj|)
    then obtain a b
      where ab-def: ab = [a, b]₀ and a: a ∈₀ 𝔄(|Obj|) and b: b ∈₀ 𝔅(|Obj|)
      by (elim cat-prod-2-ObjE[OF assms(1,2)])
    interpret 𝔑a: is-iso-ntcf
      α 𝔄 ℭ ‹𝔖_𝔄,𝔅(-,b)_CF› ‹𝔖'_𝔄,𝔅(-,b)_CF› ‹𝔑_𝔄,𝔅(-,b)_NTCF›
      by (rule assms(4)[OF b])
    from b have 𝔑ab: 𝔑(|NTMap|)(|a, b|)• = (𝔑_𝔄,𝔅(a,-)_NTCF)(|NTMap|)(|b|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from 𝔑a.iso-ntcf-is-iso-arr[OF a] assms(1,2,3) a b show
      𝔑(|NTMap|)(|ab|) : 𝔖(|ObjMap|)(|ab|) ↦_iso ℭ 𝔖'(|ObjMap|)(|ab|)
      unfolding ab-def
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-prod-cs-intros)
  qed
```

qed

lemma *bnt-proj-snd-is-iso-ntcf-if-is-iso-ntcf*:
  assumes *category* α $\mathfrak{A}$
    and *category* α $\mathfrak{B}$
    and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF.iso} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
  shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF} :$
    $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} \mapsto_{CF.iso} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-iso-ntcfI*)
  from *assms* **show** $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF} :$
    $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} \mapsto_{CF} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
  show $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF})(\!|NTMap|\!)(\!|b|\!) :$
    $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)(\!|b|\!) \mapsto_{iso\mathfrak{C}} (\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF})(\!|ObjMap|\!)(\!|b|\!)$
    if $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **for** b
    using *assms that*
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps* **cs-intro**: *cat-prod-cs-intros cat-arrow-cs-intros*
)
qed

lemma *bnt-proj-snd-is-iso-ntcf-if-is-iso-ntcf'*[*cat-cs-intros*]:
  assumes *category* α $\mathfrak{A}$
    and *category* α $\mathfrak{B}$
    and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF.iso} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{F} = \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$
    and $\mathfrak{G} = \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(a,-)_{CF}$
    and $\mathfrak{B}' = \mathfrak{B}$
    and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
  shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{B}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  unfolding *assms*(*4–6*)
  by (*rule bnt-proj-snd-is-iso-ntcf-if-is-iso-ntcf*[*OF assms*(*1–3*,*7*)])

lemma *bnt-proj-fst-is-iso-ntcf-if-is-iso-ntcf*:
  assumes *category* α $\mathfrak{A}$
    and *category* α $\mathfrak{B}$
    and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF.iso} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
  shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF} :$
    $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} \mapsto_{CF.iso} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*intro is-iso-ntcfI*)
  from *assms* **show** $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF} :$
    $\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} \mapsto_{CF} \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
  show $(\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF})(\!|NTMap|\!)(\!|a|\!) :$
    $(\mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ObjMap|\!)(\!|a|\!) \mapsto_{iso\mathfrak{C}} (\mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF})(\!|ObjMap|\!)(\!|a|\!)$
    if $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **for** a
    using *assms that*
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps*
          cs-intro: *cat-prod-cs-intros cat-arrow-cs-intros*
)

qed

lemma *bnt-proj-fst-is-iso-ntcf-if-is-iso-ntcf'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF.iso} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{F} = \mathfrak{S}_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
and $\mathfrak{G} = \mathfrak{S}'_{\mathfrak{A},\mathfrak{B}}(-,b)_{CF}$
and $\mathfrak{A}' = \mathfrak{A}$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(-,b)_{NTCF} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{C}$
unfolding *assms*(*4–6*)
by (*rule bnt-proj-fst-is-iso-ntcf-if-is-iso-ntcf*[*OF assms*(*1–3*,*7*)])

8.25 Binatural transformation flip

8.25.1 Definition and elementary properties

definition *bnt-flip* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$ =
[
fflip ($\mathfrak{N}(\!|NTMap|\!)$),
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ ($\mathfrak{N}(\!|NTDom|\!)$),
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ ($\mathfrak{N}(\!|NTCod|\!)$),
$\mathfrak{B} \times_C \mathfrak{A}$,
$\mathfrak{N}(\!|NTDGCod|\!)$
]$_\circ$

Components.

lemma *bnt-flip-components*:
shows *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\!|NTMap|\!)$ = *fflip* ($\mathfrak{N}(\!|NTMap|\!)$)
and *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\!|NTDom|\!)$ = *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ ($\mathfrak{N}(\!|NTDom|\!)$)
and *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\!|NTCod|\!)$ = *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ ($\mathfrak{N}(\!|NTCod|\!)$)
and *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\!|NTDGDom|\!)$ = $\mathfrak{B} \times_C \mathfrak{A}$
and *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\!|NTDGCod|\!)$ = $\mathfrak{N}(\!|NTDGCod|\!)$
unfolding *bnt-flip-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{S}'$ $\mathfrak{N}$
assumes $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{N}$: *is-ntcf* α ‹$\mathfrak{A} \times_C \mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{S}$ $\mathfrak{S}'$ $\mathfrak{N}$ **by** (*rule* $\mathfrak{N}$)

lemmas *bnt-flip-components'* =
bnt-flip-components[**where** $\mathfrak{A}$=$\mathfrak{A}$ **and** $\mathfrak{B}$=$\mathfrak{B}$ **and** $\mathfrak{N}$=$\mathfrak{N}$, *unfolded cat-cs-simps*]

lemmas [*cat-cs-simps*] = *bnt-flip-components'*(*2–5*)

end

8.25.2 Natural transformation map

lemma *bnt-flip-NTMap-vsv*[*cat-cs-intros*]: *vsv* (*bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\!|NTMap|\!)$)
unfolding *bnt-flip-components* **by** (*rule fflip-vsv*)

lemma *bnt-flip-NTMap-app*:
assumes *category* α $\mathfrak{A}$

```
    and category α 𝔅
    and 𝔑 : 𝔊 ↦_CF 𝔊' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    and a ∈_∘ 𝔄(|Obj|)
    and b ∈_∘ 𝔅(|Obj|)
  shows bnt-flip 𝔄 𝔅 𝔑(|NTMap|)(|b, a|)_• = 𝔑(|NTMap|)(|a, b|)_•
  using assms
  unfolding bnt-flip-components
  by
    (
      cs-concl cs-shallow
        cs-simp: V-cs-simps cat-cs-simps cs-intro: cat-prod-cs-intros
    )

lemma bnt-flip-NTMap-app'[cat-cs-simps]:
  assumes ba = [b, a]_∘
    and category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔊 ↦_CF 𝔊' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    and a ∈_∘ 𝔄(|Obj|)
    and b ∈_∘ 𝔅(|Obj|)
  shows bnt-flip 𝔄 𝔅 𝔑(|NTMap|)(|ba|) = 𝔑(|NTMap|)(|a, b|)_•
  using assms(2–6) unfolding assms(1) by (rule bnt-flip-NTMap-app)

lemma bnt-flip-NTMap-vdomain[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔊 ↦_CF 𝔊' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
  shows 𝒟_∘ (bnt-flip 𝔄 𝔅 𝔑(|NTMap|)) = (𝔅 ×_C 𝔄)(|Obj|)
  using assms
  unfolding bnt-flip-components
  by (cs-concl cs-shallow cs-simp: V-cs-simps cat-cs-simps)

lemma bnt-flip-NTMap-vrange[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔊 ↦_CF 𝔊' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
  shows ℛ_∘ (bnt-flip 𝔄 𝔅 𝔑(|NTMap|)) = ℛ_∘ (𝔑(|NTMap|))
proof–

  interpret 𝔑: is-ntcf α ‹𝔄 ×_C 𝔅› ℭ 𝔊 𝔊' 𝔑 by (rule assms(3))

  show ?thesis
  proof(intro vsubset-antisym)

    show ℛ_∘ (bnt-flip 𝔄 𝔅 𝔑(|NTMap|)) ⊆_∘ ℛ_∘ (𝔑(|NTMap|))
    proof
      (
        intro vsv.vsv-vrange-vsubset,
        unfold bnt-flip-NTMap-vdomain[OF assms]
      )
      fix ba assume ba ∈_∘ (𝔅 ×_C 𝔄)(|Obj|)
      then obtain a b
        where ba-def: ba = [b, a]_∘
          and b: b ∈_∘ 𝔅(|Obj|)
          and a: a ∈_∘ 𝔄(|Obj|)
        by (elim cat-prod-2-ObjE[OF assms(2,1)])
      from 𝔑.ntcf-NTMap-vsv assms a b show
        bnt-flip 𝔄 𝔅 𝔑(|NTMap|)(|ba|) ∈_∘ ℛ_∘ (𝔑(|NTMap|))
```

```
      unfolding ba-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-prod-cs-intros
        )
    qed (cs-concl cs-shallow cs-intro: cat-cs-intros)

    show ℛ∘ (𝔑(|NTMap|)) ⊆∘ ℛ∘ (bnt-flip 𝔄 𝔅 𝔑(|NTMap|))
    proof(intro vsv.vsv-vrange-vsubset, unfold 𝔑.ntcf-NTMap-vdomain)
      fix ab assume prems: ab ∈∘ (𝔄 ×C 𝔅)(|Obj|)
      then obtain a b
        where ab-def: ab = [a, b]∘
          and a: a ∈∘ 𝔄(|Obj|)
          and b: b ∈∘ 𝔅(|Obj|)
        by (elim cat-prod-2-ObjE[OF assms(1,2)])
      from assms a b have ba: [b, a]∘ ∈∘ (𝔅 ×C 𝔄)(|Obj|)
        by (cs-concl cs-shallow cs-intro: cat-prod-cs-intros)
      from assms bnt-flip-NTMap-vsv prems a b ba show
        𝔑(|NTMap|)(|ab|) ∈∘ ℛ∘ (bnt-flip 𝔄 𝔅 𝔑(|NTMap|))
        unfolding ab-def
        by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: V-cs-intros)
    qed auto

  qed

qed
```

8.25.3 Binatural transformation flip natural transformation map

```
lemma bnt-flip-NTMap-is-ntcf:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔖 ↦↦CF 𝔖' : 𝔄 ×C 𝔅 ↦↦↦Cα ℭ
  shows bnt-flip 𝔄 𝔅 𝔑 :
    bifunctor-flip 𝔄 𝔅 𝔖 ↦↦CF bifunctor-flip 𝔄 𝔅 𝔖' :
    𝔅 ×C 𝔄 ↦↦↦Cα ℭ
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔅: category α 𝔅 by (rule assms(2))

  interpret 𝔑: is-ntcf α ‹𝔄 ×C 𝔅› ℭ 𝔖 𝔖' 𝔑 by (rule assms(3))

  show ?thesis
  proof(intro is-ntcfI')
    show vfsequence (bnt-flip 𝔄 𝔅 𝔑) unfolding bnt-flip-def by simp
    show vcard (bnt-flip 𝔄 𝔅 𝔑) = 5ℕ
      unfolding bnt-flip-def by (simp add: nat-omega-simps)
    show bnt-flip 𝔄 𝔅 𝔑(|NTMap|)(|ba|) :
      bifunctor-flip 𝔄 𝔅 𝔖(|ObjMap|)(|ba|) ↦ℭ
      bifunctor-flip 𝔄 𝔅 𝔖'(|ObjMap|)(|ba|)
      if ba ∈∘ (𝔅 ×C 𝔄)(|Obj|) for ba
    proof–
      from that obtain b a
        where ba-def: ba = [b, a]∘
          and b: b ∈∘ 𝔅(|Obj|)
          and a: a ∈∘ 𝔄(|Obj|)
```

```
      by (elim cat-prod-2-ObjE[rotated 2]) (auto intro: cat-cs-intros)
    from assms a b show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps ba-def
            cs-intro: cat-cs-intros cat-prod-cs-intros
        )
  qed
  show
    bnt-flip 𝔄 𝔅 𝔑(|NTMap|)(|b′a′|) ∘_A ℭ bifunctor-flip 𝔄 𝔅 𝔖(|ArrMap|)(|gf|) =
      bifunctor-flip 𝔄 𝔅 𝔖′(|ArrMap|)(|gf|) ∘_A ℭ bnt-flip 𝔄 𝔅 𝔑(|NTMap|)(|ba|)
    if gf : ba ↦_𝔅 ×_C 𝔄 b′a′ for ba b′a′ gf
  proof–
    from that obtain g f a b a′ b′
      where gf-def: gf = [g, f]∘
        and ba-def: ba = [b, a]∘
        and b′a′-def: b′a′ = [b′, a′]∘
        and g: g : b ↦_𝔅 b′
        and f: f : a ↦_𝔄 a′
      by (elim cat-prod-2-is-arrE[OF assms(2,1)])
    from assms g f show ?thesis
      unfolding gf-def ba-def b′a′-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-cs-simps 𝔑.ntcf-Comp-commute
            cs-intro: cat-cs-intros cat-prod-cs-intros
        )
  qed

qed
  (
    use assms in
      ‹cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros›
  )+

qed

lemma bnt-flip-NTMap-is-ntcf'[cat-cs-intros]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔖 ↦_CF 𝔖′ : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    and 𝒯 = bifunctor-flip 𝔄 𝔅 𝔖
    and 𝒯′ = bifunctor-flip 𝔄 𝔅 𝔖′
    and 𝔇 = 𝔅 ×_C 𝔄
  shows bnt-flip 𝔄 𝔅 𝔑 : 𝒯 ↦_CF 𝒯′ : 𝔇 ↦↦_Cα ℭ
  using assms(1–3) unfolding assms(4–6) by (intro bnt-flip-NTMap-is-ntcf)
```

8.25.4 Double-flip of a binatural transformation

```
lemma bnt-flip-flip[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔖 ↦_CF 𝔖′ : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
  shows bnt-flip 𝔅 𝔄 (bnt-flip 𝔄 𝔅 𝔑) = 𝔑
proof(rule ntcf-eqI)
  interpret 𝔄: category α 𝔄 by (rule assms(1))
```

```
  interpret 𝔅: category α 𝔅 by (rule assms(2))
  interpret 𝔑: is-ntcf α ‹𝔄 ×_C 𝔅› ℭ 𝔖 𝔖' 𝔑 by (rule assms(3))
  from assms show
    bnt-flip 𝔅 𝔄 (bnt-flip 𝔄 𝔅 𝔑) : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  then have dom-lhs:
    𝒟∘ (bnt-flip 𝔅 𝔄 (bnt-flip 𝔄 𝔅 𝔑)(|NTMap|)) = (𝔄 ×_C 𝔅)(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps)
  show 𝔑 : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ by (rule assms(3))
  then have dom-rhs: 𝒟∘ (𝔑(|NTMap|)) = (𝔄 ×_C 𝔅)(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show bnt-flip 𝔅 𝔄 (bnt-flip 𝔄 𝔅 𝔑)(|NTMap|) = 𝔑(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix ab assume ab ∈∘ (𝔄 ×_C 𝔅)(|Obj|)
    then obtain a b
      where ab-def: ab = [a, b]∘
        and a: a ∈∘ 𝔄(|Obj|)
        and b: b ∈∘ 𝔅(|Obj|)
      by (rule cat-prod-2-ObjE[OF assms(1,2)])
    from assms a b show
      bnt-flip 𝔅 𝔄 (bnt-flip 𝔄 𝔅 𝔑)(|NTMap|)(|ab|) = 𝔑(|NTMap|)(|ab|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps ab-def cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-shallow cs-intro: V-cs-intros cat-cs-intros)+
qed simp-all
```

8.25.5 A projection of a flip of a binatural transformation

```
lemma bnt-flip-proj-snd[cat-cs-simps]:
  assumes category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔖 ↦_CF 𝔖' : 𝔄 ×_C 𝔅 ↦↦_Cα ℭ
    and b ∈∘ 𝔅(|Obj|)
  shows bnt-flip 𝔄 𝔅 𝔑_𝔅,𝔄(b,−)_NTCF = 𝔑_𝔄,𝔅(−,b)_NTCF
proof(rule ntcf-eqI)
  from assms show bnt-flip 𝔄 𝔅 𝔑_𝔅,𝔄(b,−)_NTCF :
    bifunctor-flip 𝔄 𝔅 𝔖_𝔅,𝔄(b,−)_CF ↦_CF bifunctor-flip 𝔄 𝔅 𝔖'_𝔅,𝔄(b,−)_CF :
    𝔄 ↦↦_Cα ℭ
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  from assms show 𝔑_𝔄,𝔅(−,b)_NTCF :
    bifunctor-flip 𝔄 𝔅 𝔖_𝔅,𝔄(b,−)_CF ↦_CF bifunctor-flip 𝔄 𝔅 𝔖'_𝔅,𝔄(b,−)_CF :
    𝔄 ↦↦_Cα ℭ
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms have dom-lhs:
    𝒟∘ ((bnt-flip 𝔄 𝔅 𝔑_𝔅,𝔄(b,−)_NTCF)(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms have dom-rhs: 𝒟∘ ((𝔑_𝔄,𝔅(−,b)_NTCF)(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show (bnt-flip 𝔄 𝔅 𝔑_𝔅,𝔄(b,−)_NTCF)(|NTMap|) = (𝔑_𝔄,𝔅(−,b)_NTCF)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume a ∈∘ 𝔄(|Obj|)
    with assms show
      (bnt-flip 𝔄 𝔅 𝔑_𝔅,𝔄(b,−)_NTCF)(|NTMap|)(|a|) = (𝔑_𝔄,𝔅(−,b)_NTCF)(|NTMap|)(|a|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  qed (auto simp: cat-cs-intros)
```

qed *simp-all*

lemma *bnt-flip-proj-fst*[*cat-cs-simps*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ \mathfrak{A}(|Obj|)$
shows *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}_{\mathfrak{B},\mathfrak{A}}(-,a)_{NTCF} = \mathfrak{N}_{\mathfrak{A},\mathfrak{B}}(a,-)_{NTCF}$
proof–
from *assms* **have** *f-*$\mathfrak{N}$:
bnt-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$:
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}$ $\mapsto_{CF}$ *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}'$:
$\mathfrak{B} \times_C \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show *?thesis*
by
(
rule
bnt-flip-proj-snd
[
OF assms(*2*,*1*) *f-*$\mathfrak{N}$ *assms*(*4*),
unfolded bnt-flip-flip[*OF assms*(*1*,*2*,*3*)],
symmetric
]
)
qed

8.25.6 A flip of a binatural isomorphism

lemma *bnt-flip-is-iso-ntcf*:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF.iso} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$:
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}$ $\mapsto_{CF.iso}$ *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}'$:
$\mathfrak{B} \times_C \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof(*rule is-iso-ntcf-if-bnt-proj-snd-is-iso-ntcf*)
from *assms* **show** *f-*$\mathfrak{N}$: *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$:
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}$ $\mapsto_{CF}$ *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}'$:
$\mathfrak{B} \times_C \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
fix *a* **assume** $a \in_\circ \mathfrak{B}(|Obj|)$
with *assms f-*$\mathfrak{N}$ **show**
bnt-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}_{\mathfrak{B},\mathfrak{A}}(a,-)_{NTCF}$:
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}_{\mathfrak{B},\mathfrak{A}}(a,-)_{CF} \mapsto_{CF.iso}$
bifunctor-flip $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}'_{\mathfrak{B},\mathfrak{A}}(a,-)_{CF}$:
$\mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros ntcf-cs-intros*)
qed (*simp-all add*: *assms*)

lemma *bnt-flip-is-iso-ntcf'*[*cat-cs-intros*]:
assumes *category* α $\mathfrak{A}$
and *category* α $\mathfrak{B}$
and $\mathfrak{N} : \mathfrak{S} \mapsto_{CF.iso} \mathfrak{S}' : \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{F} =$ *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}$
and $\mathfrak{G} =$ *bifunctor-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{S}'$
and $\mathfrak{D} = \mathfrak{B} \times_C \mathfrak{A}$
shows *bnt-flip* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{D} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

using *bnt-flip-is-iso-ntcf*[*OF assms*(*1–3*)] **unfolding** *assms*(*4–6*) **by** *simp*

9 Subcategory

9.1 Background

named-theorems *cat-sub-cs-intros*
named-theorems *cat-sub-bw-cs-intros*
named-theorems *cat-sub-fw-cs-intros*
named-theorems *cat-sub-bw-cs-simps*

9.2 Simple subcategory

9.2.1 Definition and elementary properties

See Chapter I-3 in [7].

locale *subcategory* = *sdg*: *category* α $\mathfrak{B}$ + *dg*: *category* α $\mathfrak{C}$ **for** α $\mathfrak{B}$ $\mathfrak{C}$ +
  assumes *subcat-subsemicategory*: *cat-smc* $\mathfrak{B}$ $\subseteq_{SMC\alpha}$ *cat-smc* $\mathfrak{C}$
    and *subcat-CId*: $a \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow \mathfrak{B}(\!|CId|\!)(\!|a|\!) = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$

abbreviation *is-subcategory* ($\langle$(-/ $\subseteq_{C^1}$ -)$\rangle$ [51, 51] 50)
  where $\mathfrak{B} \subseteq_{C\alpha} \mathfrak{C} \equiv$ *subcategory* α $\mathfrak{B}$ $\mathfrak{C}$

Rules.

lemma (**in** *subcategory*) *subcategory-axioms'*[*cat-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
  shows $\mathfrak{B}' \subseteq_{C\alpha'} \mathfrak{C}$
  unfolding *assms* **by** (*rule subcategory-axioms*)

lemma (**in** *subcategory*) *subcategory-axioms''*[*cat-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows $\mathfrak{B} \subseteq_{C\alpha'} \mathfrak{C}'$
  unfolding *assms* **by** (*rule subcategory-axioms*)

mk-ide rf *subcategory-def*[*unfolded subcategory-axioms-def*]
  |*intro subcategoryI*[*intro*!]|
  |*dest subcategoryD*[*dest*]|
  |*elim subcategoryE*[*elim*!]|

lemmas [*cat-sub-cs-intros*] = *subcategoryD*(*1*,*2*)

lemma *subcategoryI'*:
  assumes *category* α $\mathfrak{B}$
    and *category* α $\mathfrak{C}$
    and $\bigwedge a.\ a \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow a \in_\circ \mathfrak{C}(\!|Obj|\!)$
    and $\bigwedge a\ b\ f.\ f : a \mapsto_{\mathfrak{B}} b \Longrightarrow f : a \mapsto_{\mathfrak{C}} b$
    and $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{B}} c;\ f : a \mapsto_{\mathfrak{B}} b\ ]\!] \Longrightarrow$
      $g \circ_{A\mathfrak{B}} f = g \circ_{A\mathfrak{C}} f$
    and $\bigwedge a.\ a \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow \mathfrak{B}(\!|CId|\!)(\!|a|\!) = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
  shows $\mathfrak{B} \subseteq_{C\alpha} \mathfrak{C}$
proof–
  interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
  show *?thesis*
    by
      (
        intro subcategoryI subsemicategoryI',
        unfold slicing-simps;
        (*intro* $\mathfrak{B}$.*cat-semicategory* $\mathfrak{C}$.*cat-semicategory assms*)*?*
)

qed

A subcategory is a subsemicategory.

context *subcategory*
begin

interpretation *subsmc*: *subsemicategory* α ‹*cat-smc* $\mathfrak{B}$› ‹*cat-smc* $\mathfrak{C}$›
  by (*rule subcat-subsemicategory*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
  subcat-Obj-vsubset = *subsmc.subsmc-Obj-vsubset*
  and *subcat-is-arr-vsubset* = *subsmc.subsmc-is-arr-vsubset*
  and *subcat-subdigraph-op-dg-op-dg* = *subsmc.subsmc-subdigraph-op-dg-op-dg*
  and *subcat-objD* = *subsmc.subsmc-objD*
  and *subcat-arrD* = *subsmc.subsmc-arrD*
  and *subcat-dom-simp* = *subsmc.subsmc-dom-simp*
  and *subcat-cod-simp* = *subsmc.subsmc-cod-simp*
  and *subcat-is-arrD* = *subsmc.subsmc-is-arrD*

lemmas-with [*unfolded slicing-simps slicing-commute*]:
  subcat-Comp-simp = *subsmc.subsmc-Comp-simp*
  and *subcat-is-idem-arrD* = *subsmc.subsmc-is-idem-arrD*

end

lemmas [*cat-sub-fw-cs-intros*] =
  subcategory.subcat-Obj-vsubset
  subcategory.subcat-is-arr-vsubset
  subcategory.subcat-objD
  subcategory.subcat-arrD
  subcategory.subcat-is-arrD

lemmas [*cat-sub-bw-cs-simps*] =
  subcategory.subcat-dom-simp
  subcategory.subcat-cod-simp

lemmas [*cat-sub-fw-cs-intros*] =
  subcategory.subcat-is-idem-arrD

lemmas [*cat-sub-bw-cs-simps*] =
  subcategory.subcat-Comp-simp

The opposite subcategory.

lemma (**in** *subcategory*) *subcat-subcategory-op-cat*: *op-cat* $\mathfrak{B}$ $\subseteq_{C\alpha}$ *op-cat* $\mathfrak{C}$
proof(*rule subcategoryI*)
  show *cat-smc* (*op-cat* $\mathfrak{B}$) $\subseteq_{SMC\alpha}$ *cat-smc* (*op-cat* $\mathfrak{C}$)
    unfolding *slicing-commute*[*symmetric*]
    by (*intro subsmc-subsemicategory-op-smc subcat-subsemicategory*)
qed (*simp-all add*: *sdg.category-op dg.category-op cat-op-simps subcat-CId*)

lemmas *subcat-subcategory-op-cat*[*intro*] = *subcategory.subcat-subcategory-op-cat*

Elementary properties.

lemma (**in** *subcategory*) *subcat-CId-is-arr*[*intro*]:
  assumes $a \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
  shows $\mathfrak{C}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{B}} a$
proof–

from *assms* **have** $\mathfrak{BC}$: $\mathfrak{B}(\!|CId|\!)(\!|a|\!) = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$ **by** (*simp add*: *subcat-CId*)
from *assms* **have** $\mathfrak{B}(\!|CId|\!)(\!|a|\!) : a \mapsto_{\mathfrak{B}} a$ **by** (*auto intro*: *cat-cs-intros*)
then show *?thesis* **unfolding** $\mathfrak{BC}$ **by** *simp*
qed

Further rules.

lemma (**in** *subcategory*) *subcat-CId-simp*[*cat-sub-bw-cs-simps*]:
assumes $a \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathfrak{B}(\!|CId|\!)(\!|a|\!) = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
using *assms* **by** (*simp add*: *subcat-CId*)

lemmas [*cat-sub-bw-cs-simps*] = *subcategory.subcat-CId-simp*

lemma (**in** *subcategory*) *subcat-is-right-inverseD*[*cat-sub-fw-cs-intros*]:
assumes *is-right-inverse* $\mathfrak{B}$ *g f*
shows *is-right-inverse* $\mathfrak{C}$ *g f*
using *assms subcategory-axioms*
by (*elim is-right-inverseE*, *intro is-right-inverseI*)
(
cs-concl
cs-simp: *cat-sub-bw-cs-simps*[*symmetric*]
cs-intro: *cat-sub-fw-cs-intros cat-cs-intros cat-sub-cs-intros*
)

lemmas [*cat-sub-fw-cs-intros*] = *subcategory.subcat-is-right-inverseD*

lemma (**in** *subcategory*) *subcat-is-left-inverseD*[*cat-sub-fw-cs-intros*]:
assumes *is-left-inverse* $\mathfrak{B}$ *g f*
shows *is-left-inverse* $\mathfrak{C}$ *g f*
proof–
have *op-cat* $\mathfrak{B} \subseteq_{C\alpha}$ *op-cat* $\mathfrak{C}$ **by** (*simp add*: *subcat-subcategory-op-cat*)
from *subcategory.subcat-is-right-inverseD*[*OF this*] **show** *?thesis*
unfolding *cat-op-simps* **using** *assms*.
qed

lemmas [*cat-sub-fw-cs-intros*] = *subcategory.subcat-is-left-inverseD*

lemma (**in** *subcategory*) *subcat-is-inverseD*[*cat-sub-fw-cs-intros*]:
assumes *is-inverse* $\mathfrak{B}$ *g f*
shows *is-inverse* $\mathfrak{C}$ *g f*
using *assms subcategory-axioms*
by (*elim is-inverseE*, *intro is-inverseI*)
(
cs-concl
cs-simp: *cat-sub-bw-cs-simps*[*symmetric*]
cs-intro: *cat-sub-fw-cs-intros cat-cs-intros cat-sub-cs-intros*
)

lemmas [*cat-sub-fw-cs-intros*] = *subcategory.subcat-is-inverseD*

lemma (**in** *subcategory*) *subcat-is-iso-arrD*[*cat-sub-fw-cs-intros*]:
assumes $f : a \mapsto_{iso\mathfrak{B}} b$
shows $f : a \mapsto_{iso\mathfrak{C}} b$
proof(*intro is-iso-arrI*)
from *subcategory-axioms is-iso-arrD*(*1*)[*OF assms*] **show** $f : a \mapsto_{\mathfrak{C}} b$
by
(
cs-concl **cs-shallow**

```
      cs-simp: cat-sub-bw-cs-simps[symmetric] cs-intro: cat-sub-fw-cs-intros
    )
  from assms have is-inverse 𝔅 (f⁻¹_C𝔅) f
    by (rule sdg.cat-the-inverse-is-inverse)
  with subcategory-axioms show is-inverse ℭ (f⁻¹_C𝔅) f
    by (elim is-inverseE, intro is-inverseI)
      (
        cs-concl
          cs-simp: cat-sub-bw-cs-simps[symmetric]
          cs-intro: cat-sub-fw-cs-intros cat-cs-intros
      )
qed

lemmas [cat-sub-fw-cs-intros] = subcategory.subcat-is-iso-arrD

lemma (in subcategory) subcat-the-inverse-simp[cat-sub-bw-cs-simps]:
  assumes f : a ↦_iso𝔅 b
  shows f⁻¹_C𝔅 = f⁻¹_Cℭ
proof–
  from assms have is-inverse 𝔅 (f⁻¹_C𝔅) f
    by (auto dest: sdg.cat-the-inverse-is-inverse)
  with subcategory-axioms have inv-f𝔅: is-inverse ℭ (f⁻¹_C𝔅) f
    by (auto dest: cat-sub-fw-cs-intros)
  from assms have f : a ↦_isoℭ b by (auto dest: cat-sub-fw-cs-intros)
  then have inv-fℭ: is-inverse ℭ (f⁻¹_Cℭ) f
    by (auto dest: dg.cat-the-inverse-is-inverse)
  from inv-f𝔅 inv-fℭ show ?thesis by (intro dg.cat-is-inverse-eq)
qed

lemmas [cat-sub-bw-cs-simps] = subcategory.subcat-the-inverse-simp

lemma (in subcategory) subcat-obj-isoD:
  assumes a ≈_obj𝔅 b
  shows a ≈_objℭ b
  using assms subcategory-axioms
  by (elim obj-isoE)
    (
      cs-concl cs-shallow
        cs-simp: cat-sub-bw-cs-simps cs-intro: obj-isoI cat-sub-fw-cs-intros
    )

lemmas [cat-sub-fw-cs-intros] = subcategory.subcat-obj-isoD
```

9.2.2 Subcategory relation is a partial order

```
lemma subcat-refl:
  assumes category α 𝔄
  shows 𝔄 ⊆_Cα 𝔄
proof–
  interpret category α 𝔄 by (rule assms)
  show ?thesis
    by (auto intro: cat-cs-intros slicing-intros subdg-refl subsemicategoryI)
qed

lemma subcat-trans:
  assumes 𝔄 ⊆_Cα 𝔅 and 𝔅 ⊆_Cα ℭ
  shows 𝔄 ⊆_Cα ℭ
proof–
```

```
  interpret 𝔄𝔅: subcategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅ℭ: subcategory α 𝔅 ℭ by (rule assms(2))
  show ?thesis
  proof(rule subcategoryI)
    show cat-smc 𝔄 ⊆_{SMCα} cat-smc ℭ
      by
        (
          meson
            𝔄𝔅.subcat-subsemicategory
            𝔅ℭ.subcat-subsemicategory
            subsmc-trans
        )
  qed
    (
      use 𝔄𝔅.subcategory-axioms 𝔅ℭ.subcategory-axioms in
        ‹auto simp: 𝔄𝔅.subcat-Obj-vsubset cat-sub-bw-cs-simps›
    )
qed

lemma subcat-antisym:
  assumes 𝔄 ⊆_{Cα} 𝔅 and 𝔅 ⊆_{Cα} 𝔄
  shows 𝔄 = 𝔅
proof–
  interpret 𝔄𝔅: subcategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅𝔄: subcategory α 𝔅 𝔄 by (rule assms(2))
  show ?thesis
  proof(rule cat-eqI)
    from
      subsmc-antisym[
        OF 𝔄𝔅.subcat-subsemicategory 𝔅𝔄.subcat-subsemicategory
        ]
    have
      cat-smc 𝔄(|Obj|) = cat-smc 𝔅(|Obj|) cat-smc 𝔄(|Arr|) = cat-smc 𝔅(|Arr|)
      by simp-all
    then show Obj: 𝔄(|Obj|) = 𝔅(|Obj|) and Arr: 𝔄(|Arr|) = 𝔅(|Arr|)
      unfolding slicing-simps by simp-all
    show 𝔄(|Dom|) = 𝔅(|Dom|)
      by (rule vsv-eqI) (auto simp: 𝔄𝔅.subcat-dom-simp Arr cat-cs-simps)
    show 𝔄(|Cod|) = 𝔅(|Cod|)
      by (rule vsv-eqI) (auto simp: 𝔅𝔄.subcat-cod-simp Arr cat-cs-simps)
    have cat-smc 𝔄 ⊆_{SMCα} cat-smc 𝔅 cat-smc 𝔅 ⊆_{SMCα} cat-smc 𝔄
      by (simp-all add: 𝔄𝔅.subcat-subsemicategory 𝔅𝔄.subcat-subsemicategory)
    from subsmc-antisym[OF this] have cat-smc 𝔄 = cat-smc 𝔅 .
    then have cat-smc 𝔄(|Comp|) = cat-smc 𝔅(|Comp|) by auto
    then show 𝔄(|Comp|) = 𝔅(|Comp|) unfolding slicing-simps by simp
    show 𝔄(|CId|) = 𝔅(|CId|)
      by (rule vsv-eqI) (auto simp: Obj 𝔄𝔅.subcat-CId-simp cat-cs-simps)
  qed (auto intro: cat-cs-intros)
qed
```

9.3 Inclusion functor

9.3.1 Definition and elementary properties

See Chapter I-3 in [7].

```
abbreviation (input) cf-inc :: V ⇒ V ⇒ V
  where cf-inc ≡ dghm-inc
```

Slicing.

lemma *dghm-smcf-inc*[*slicing-commute*]:
dghm-inc (*cat-smc* $\mathfrak{B}$) (*cat-smc* $\mathfrak{C}$) = *cf-smcf* (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$)
unfolding *cf-smcf-def dghm-inc-def cat-smc-def dg-field-simps dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Elementary properties.

lemmas [*cat-cs-simps*] =
dghm-inc-ObjMap-app
dghm-inc-ArrMap-app

9.3.2 Canonical inclusion functor associated with a subcategory

sublocale *subcategory* ⊆ *inc*: *is-ft-functor* α $\mathfrak{B}$ $\mathfrak{C}$ ‹*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$›
proof(*rule is-ft-functorI*)
interpret *subsmc*: *subsemicategory* α ‹*cat-smc* $\mathfrak{B}$› ‹*cat-smc* $\mathfrak{C}$›
by (*rule subcat-subsemicategory*)
show *cf-inc* $\mathfrak{B}$ $\mathfrak{C}$: $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
proof(*rule is-functorI*)
show *vfsequence* (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$) **unfolding** *dghm-inc-def* **by** *auto*
show *vcard* (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$) = $\mathfrak{4}_{\mathbb{N}}$
unfolding *dghm-inc-def* **by** (*simp add*: *nat-omega-simps*)
from *sdg.cat-CId-is-arr subcat-CId-simp* **show** *c* ∈$_\circ$ $\mathfrak{B}$⦇*Obj*⦈ ⟹
cf-inc $\mathfrak{B}$ $\mathfrak{C}$⦇*ArrMap*⦈⦇$\mathfrak{B}$⦇*CId*⦈⦇*c*⦈⦈ = $\mathfrak{C}$⦇*CId*⦈⦇*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$⦇*ObjMap*⦈⦇*c*⦈⦈
for *c*
unfolding *dghm-inc-components* **by** *force*
from *subsmc.inc.is-ft-semifunctor-axioms* **show**
cf-smcf (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$) : *cat-smc* $\mathfrak{B}$ $\mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{C}$
unfolding *slicing-commute* **by** *auto*
qed (*auto simp*: *dghm-inc-components cat-cs-intros*)
from *subsmc.inc.is-ft-semifunctor-axioms* **show**
cf-smcf (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$) : *cat-smc* $\mathfrak{B}$ $\mapsto\mapsto_{SMC.faithful\alpha}$ *cat-smc* $\mathfrak{C}$
unfolding *slicing-commute* **by** *auto*
qed

lemmas (**in** *subcategory*) *subcat-cf-inc-is-ft-functor* = *inc.is-ft-functor-axioms*

9.3.3 Inclusion functor for the opposite categories

lemma (**in** *subcategory*) *subcat-cf-inc-op-cat-is-functor*:
cf-inc (*op-cat* $\mathfrak{B}$) (*op-cat* $\mathfrak{C}$) : *op-cat* $\mathfrak{B}$ $\mapsto\mapsto_{C.faithful\alpha}$ *op-cat* $\mathfrak{C}$
by
(
intro
subcategory.subcat-cf-inc-is-ft-functor
subcat-subcategory-op-cat
)

lemma (**in** *subcategory*) *subcat-op-cat-cf-inc*:
cf-inc (*op-cat* $\mathfrak{B}$) (*op-cat* $\mathfrak{C}$) = *op-cf* (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$)
by (*rule cf-eqI*)
(
auto
simp:
cat-op-simps
dghm-inc-components
subcat-cf-inc-op-cat-is-functor
is-ft-functor.axioms(*1*)

intro: *cat-op-intros*
)

9.4 Full subcategory

See Chapter I-3 in [7].

locale *fl-subcategory* = *subcategory* +
assumes *fl-subcat-fl-subsemicategory*: *cat-smc* $\mathfrak{B}$ $\subseteq_{SMC.full\alpha}$ *cat-smc* $\mathfrak{C}$

abbreviation *is-fl-subcategory* (‹(-/ $\subseteq_{C.full1}$ -)› [*51*, *51*] *50*)
where $\mathfrak{B}$ $\subseteq_{C.full\alpha}$ $\mathfrak{C}$ ≡ *fl-subcategory* α $\mathfrak{B}$ $\mathfrak{C}$

Rules.

mk-ide rf *fl-subcategory-def*[*unfolded fl-subcategory-axioms-def*]
|*intro fl-subcategoryI*|
|*dest fl-subcategoryD*[*dest*]|
|*elim fl-subcategoryE*[*elim*!]|

lemmas [*cat-sub-cs-intros*] = *fl-subcategoryD*(*1*)

Elementary properties.

sublocale *fl-subcategory* ⊆ *inc*: *is-fl-functor* α $\mathfrak{B}$ $\mathfrak{C}$ ‹*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$›
proof(*rule is-fl-functorI*)
interpret *fl-subsemicategory* α ‹*cat-smc* $\mathfrak{B}$› ‹*cat-smc* $\mathfrak{C}$›
by (*rule fl-subcat-fl-subsemicategory*)
from *inc.is-fl-semifunctor-axioms* **show**
cf-smcf (*cf-inc* $\mathfrak{B}$ $\mathfrak{C}$) : *cat-smc* $\mathfrak{B}$ $\mapsto\mapsto_{SMC.full\alpha}$ *cat-smc* $\mathfrak{C}$
unfolding *slicing-commute* **by** *simp*
qed (*rule inc.is-functor-axioms*)

9.5 Wide subcategory

9.5.1 Definition and elementary properties

See [1][3].

locale *wide-subcategory* = *subcategory* +
assumes *wide-subcat-wide-subsemicategory*: *cat-smc* $\mathfrak{B}$ $\subseteq_{SMC.wide\alpha}$ *cat-smc* $\mathfrak{C}$

abbreviation *is-wide-subcategory* (‹(-/ $\subseteq_{C.wide1}$ -)› [*51*, *51*] *50*)
where $\mathfrak{B}$ $\subseteq_{C.wide\alpha}$ $\mathfrak{C}$ ≡ *wide-subcategory* α $\mathfrak{B}$ $\mathfrak{C}$

Rules.

mk-ide rf *wide-subcategory-def*[*unfolded wide-subcategory-axioms-def*]
|*intro wide-subcategoryI*|
|*dest wide-subcategoryD*[*dest*]|
|*elim wide-subcategoryE*[*elim*!]|

lemmas [*cat-sub-cs-intros*] = *wide-subcategoryD*(*1*)

Wide subcategory is wide subsemicategory.

context *wide-subcategory*
begin

interpretation *wide-subsmc*: *wide-subsemicategory* α ‹*cat-smc* $\mathfrak{B}$› ‹*cat-smc* $\mathfrak{C}$›

[3] https://ncatlab.org/nlab/show/wide+subcategory

by (*rule wide-subcat-wide-subsemicategory*)

lemmas-with [*unfolded slicing-simps*]:
wide-subcat-Obj[*dg-sub-bw-cs-intros*] = *wide-subsmc.wide-subsmc-Obj*
and *wide-subcat-obj-eq*[*dg-sub-bw-cs-simps*] = *wide-subsmc.wide-subsmc-obj-eq*

end

lemmas [*cat-sub-bw-cs-simps*] = *wide-subcategory.wide-subcat-obj-eq*[*symmetric*]
lemmas [*cat-sub-bw-cs-simps*] = *wide-subsemicategory.wide-subsmc-obj-eq*

9.5.2 The wide subcategory relation is a partial order

lemma *wide-subcat-refl*:
assumes *category* α $\mathfrak{A}$
shows $\mathfrak{A} \subseteq_{C.wide\alpha} \mathfrak{A}$
proof–
interpret *category* α $\mathfrak{A}$ **by** (*rule assms*)
show *?thesis*
by
(
auto intro:
assms
slicing-intros
wide-subsmc-refl
wide-subcategoryI
subsmc-refl
)
qed

lemma *wide-subcat-trans*[*trans*]:
assumes $\mathfrak{A} \subseteq_{C.wide\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{C.wide\alpha} \mathfrak{C}$
shows $\mathfrak{A} \subseteq_{C.wide\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{AB}$: *wide-subcategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{BC}$: *wide-subcategory* α $\mathfrak{B}$ $\mathfrak{C}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
intro
wide-subcategoryI
subcat-trans[*OF* $\mathfrak{AB}$.*subcategory-axioms* $\mathfrak{BC}$.*subcategory-axioms*],
rule wide-subsmc-trans,
rule $\mathfrak{AB}$.*wide-subcat-wide-subsemicategory*,
rule $\mathfrak{BC}$.*wide-subcat-wide-subsemicategory*
)
qed

lemma *wide-subcat-antisym*:
assumes $\mathfrak{A} \subseteq_{C.wide\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{C.wide\alpha} \mathfrak{A}$
shows $\mathfrak{A} = \mathfrak{B}$
proof–
interpret $\mathfrak{AB}$: *wide-subcategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{BA}$: *wide-subcategory* α $\mathfrak{B}$ $\mathfrak{A}$ **by** (*rule assms*(*2*))
show *?thesis*
by (*rule subcat-antisym*[*OF* $\mathfrak{AB}$.*subcategory-axioms* $\mathfrak{BA}$.*subcategory-axioms*])
qed

9.6 Replete subcategory

9.6.1 Definition and elementary properties

See nLab [1][4].

locale *replete-subcategory* = *subcategory* α $\mathfrak{B}$ $\mathfrak{C}$ **for** α $\mathfrak{B}$ $\mathfrak{C}$ +
  assumes *rep-subcat-is-iso-arr-is-arr*:
    $a \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow f : a \mapsto_{iso\mathfrak{C}} b \Longrightarrow f : a \mapsto_{\mathfrak{B}} b$

abbreviation *is-replete-subcategory* $(\langle(\text{-}/\ \subseteq_{C.rep^1}\ \text{-})\rangle\ [51,\ 51]\ 50)$
  where $\mathfrak{B} \subseteq_{C.rep\alpha} \mathfrak{C} \equiv$ *replete-subcategory* α $\mathfrak{B}$ $\mathfrak{C}$

Rules.

mk-ide rf *replete-subcategory-def*[*unfolded replete-subcategory-axioms-def*]
  |*intro replete-subcategoryI*|
  |*dest replete-subcategoryD*[*dest*]|
  |*elim replete-subcategoryE*[*elim!*]|

lemmas [*cat-sub-cs-intros*] = *replete-subcategoryD*(*1*)

Elementary properties.

lemma (**in** *replete-subcategory*)
  rep-subcat-is-iso-arr-is-iso-arr-left:
  assumes $a \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** $f : a \mapsto_{iso\mathfrak{C}} b$
  shows $f : a \mapsto_{iso\mathfrak{B}} b$
proof(*intro is-iso-arrI is-inverseI*)
  from *assms* **show** *f*: $f : a \mapsto_{\mathfrak{B}} b$
    by (*auto intro*: *rep-subcat-is-iso-arr-is-arr*)
  have ${f^{-1}}_{C\mathfrak{C}} : b \mapsto_{iso\mathfrak{C}} a$
    by (*rule dg.cat-the-inverse-is-iso-arr*[*OF assms*(*2*)])
  with *f* **show** *inv-f*: ${f^{-1}}_{C\mathfrak{C}} : b \mapsto_{\mathfrak{B}} a$
    by (*auto intro*: *rep-subcat-is-iso-arr-is-arr*)
  show $f : a \mapsto_{\mathfrak{B}} b$ **by** (*rule f*)
  from *dg.category-axioms assms* **have** [*cat-sub-bw-cs-simps*]:
    ${f^{-1}}_{C\mathfrak{C}} \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
  from *dg.category-axioms assms* **have** [*cat-sub-bw-cs-simps*]:
    $f \circ_{A\mathfrak{C}} {f^{-1}}_{C\mathfrak{C}} = \mathfrak{C}(\!|CId|\!)(\!|b|\!)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
  from *subcategory-axioms f inv-f* **show** ${f^{-1}}_{C\mathfrak{C}} \circ_{A\mathfrak{B}} f = \mathfrak{B}(\!|CId|\!)(\!|a|\!)$
    by (*cs-concl* **cs-simp**: *cat-sub-bw-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *subcategory-axioms f inv-f* **show** $f \circ_{A\mathfrak{B}} {f^{-1}}_{C\mathfrak{C}} = \mathfrak{B}(\!|CId|\!)(\!|b|\!)$
    by (*cs-concl* **cs-simp**: *cat-sub-bw-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma (**in** *replete-subcategory*)
  rep-subcat-is-iso-arr-is-iso-arr-right:
  assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** $f : a \mapsto_{iso\mathfrak{C}} b$
  shows $f : a \mapsto_{iso\mathfrak{B}} b$
proof–
  from *assms*(*2*) **have** ${f^{-1}}_{C\mathfrak{C}} : b \mapsto_{iso\mathfrak{C}} a$
    by (*rule dg.cat-the-inverse-is-iso-arr*)
  with *assms*(*1*) **have** *inv-f*: ${f^{-1}}_{C\mathfrak{C}} : b \mapsto_{iso\mathfrak{B}} a$
    by (*intro rep-subcat-is-iso-arr-is-iso-arr-left*)
  then have $({f^{-1}}_{C\mathfrak{C}})^{-1}{}_{C\mathfrak{B}} : a \mapsto_{iso\mathfrak{B}} b$
    by (*rule sdg.cat-the-inverse-is-iso-arr*)

[4] https://ncatlab.org/nlab/show/replete+subcategory

moreover from *replete-subcategory-axioms assms inv-f* **have** $(f^{-1}{}_{C\mathfrak{C}})^{-1}{}_{C\mathfrak{B}} = f$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-sub-bw-cs-simps cat-cs-simps* **cs-intro**: *cat-cs-intros*
)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *replete-subcategory*)
rep-subcat-is-iso-arr-is-iso-arr-left-iff:
assumes $a \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $f : a \mapsto_{iso\mathfrak{B}} b \longleftrightarrow f : a \mapsto_{iso\mathfrak{C}} b$
using *assms replete-subcategory-axioms*
by (*intro iffI*)
(
cs-concl **cs-intro**:
rep-subcat-is-iso-arr-is-iso-arr-left
cat-sub-fw-cs-intros
)

lemma (**in** *replete-subcategory*)
rep-subcat-is-iso-arr-is-iso-arr-right-iff:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $f : a \mapsto_{iso\mathfrak{B}} b \longleftrightarrow f : a \mapsto_{iso\mathfrak{C}} b$
using *assms replete-subcategory-axioms*
by (*intro iffI*)
(
cs-concl **cs-intro**:
rep-subcat-is-iso-arr-is-iso-arr-right
cat-sub-fw-cs-intros
)

9.6.2 The replete subcategory relation is a partial order

lemma *rep-subcat-refl*:
assumes *category* α $\mathfrak{A}$
shows $\mathfrak{A} \subseteq_{C.rep\alpha} \mathfrak{A}$
proof–
interpret *category* α $\mathfrak{A}$ **by** (*rule assms*)
show *?thesis*
by (*intro replete-subcategoryI subcat-refl assms is-iso-arrD*(*1*))
qed

lemma *rep-subcat-trans*[*trans*]:
assumes $\mathfrak{A} \subseteq_{C.rep\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{C.rep\alpha} \mathfrak{C}$
shows $\mathfrak{A} \subseteq_{C.rep\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{AB}$: *replete-subcategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{BC}$: *replete-subcategory* α $\mathfrak{B}$ $\mathfrak{C}$ **by** (*rule assms*(*2*))
show *?thesis*
proof
(
intro
replete-subcategoryI
subcat-trans[*OF* $\mathfrak{AB}$.*subcategory-axioms* $\mathfrak{BC}$.*subcategory-axioms*]
)
fix a b f **assume** *prems*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ $f : a \mapsto_{iso\mathfrak{C}} b$

```
    have b ∈∘ 𝔅(|Obj|)
      by
        (
          rule 𝔄𝔅.dg.cat-is-arrD(3)
            [
              OF 𝔅ℭ.rep-subcat-is-iso-arr-is-arr[
                OF 𝔄𝔅.subcat-objD[OF prems(1)] prems(2)
                ]
            ]
        )
    then have f : a ↦_iso𝔅 b
      by
        (
          rule 𝔅ℭ.rep-subcat-is-iso-arr-is-iso-arr-right[
            OF - prems(2)
            ]
        )
    then show f : a ↦𝔄 b
      by (rule 𝔄𝔅.rep-subcat-is-iso-arr-is-arr[OF prems(1)])
  qed
qed

lemma rep-subcat-antisym:
  assumes 𝔄 ⊆_C.repα 𝔅 and 𝔅 ⊆_C.repα 𝔄
  shows 𝔄 = 𝔅
proof–
  interpret 𝔄𝔅: replete-subcategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅𝔄: replete-subcategory α 𝔅 𝔄 by (rule assms(2))
  show ?thesis
    by (rule subcat-antisym[OF 𝔄𝔅.subcategory-axioms 𝔅𝔄.subcategory-axioms])
qed
```

9.7 Wide replete subcategory

9.7.1 Definition and elementary properties

```
locale wide-replete-subcategory =
  wide-subcategory α 𝔅 ℭ + replete-subcategory α 𝔅 ℭ for α 𝔅 ℭ

abbreviation is-wide-replete-subcategory (‹(-/ ⊆_C.wr1 -)› [51, 51] 50)
  where 𝔅 ⊆_C.wrα ℭ ≡ wide-replete-subcategory α 𝔅 ℭ
```

Rules.

```
mk-ide rf wide-replete-subcategory-def
  |intro wide-replete-subcategoryI|
  |dest wide-replete-subcategoryD[dest]|
  |elim wide-replete-subcategoryE[elim!]|

lemmas [cat-sub-cs-intros] = wide-replete-subcategoryD
```

Wide replete subcategory preserves isomorphisms.

```
lemma (in wide-replete-subcategory)
  wr-subcat-is-iso-arr-is-iso-arr:
  f : a ↦_iso𝔅 b ⟷ f : a ↦_isoℭ b
proof(rule iffI)
  assume prems: f : a ↦_isoℭ b
  then have a ∈∘ ℭ(|Obj|) by auto
  then have a: a ∈∘ 𝔅(|Obj|) by (simp add: wide-subcat-obj-eq)
```

```
  show f : a ↦_{iso}𝔅 b
    by (intro rep-subcat-is-iso-arr-is-iso-arr-left[OF a prems])
qed
  (
    use wide-replete-subcategory-axioms in
      ‹cs-concl cs-shallow cs-intro: cat-sub-fw-cs-intros ›
  )

lemmas [cat-sub-bw-cs-simps] =
  wide-replete-subcategory.wr-subcat-is-iso-arr-is-iso-arr
```

9.7.2 The wide replete subcategory relation is a partial order

```
lemma wr-subcat-refl:
  assumes category α 𝔄
  shows 𝔄 ⊆_{C.wr α} 𝔄
  by (intro wide-replete-subcategoryI wide-subcat-refl rep-subcat-refl assms)

lemma wr-subcat-trans[trans]:
  assumes 𝔄 ⊆_{C.wr α} 𝔅 and 𝔅 ⊆_{C.wr α} ℭ
  shows 𝔄 ⊆_{C.wr α} ℭ
proof–
  interpret 𝔄𝔅: wide-replete-subcategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅ℭ: wide-replete-subcategory α 𝔅 ℭ by (rule assms(2))
  show ?thesis
    by
      (
        intro wide-replete-subcategoryI,
        rule wide-subcat-trans,
        rule 𝔄𝔅.wide-subcategory-axioms,
        rule 𝔅ℭ.wide-subcategory-axioms,
        rule rep-subcat-trans,
        rule 𝔄𝔅.replete-subcategory-axioms,
        rule 𝔅ℭ.replete-subcategory-axioms
      )
qed

lemma wr-subcat-antisym:
  assumes 𝔄 ⊆_{C.wr α} 𝔅 and 𝔅 ⊆_{C.wr α} 𝔄
  shows 𝔄 = 𝔅
proof–
  interpret 𝔄𝔅: wide-replete-subcategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅𝔄: wide-replete-subcategory α 𝔅 𝔄 by (rule assms(2))
  show ?thesis
    by (rule subcat-antisym[OF 𝔄𝔅.subcategory-axioms 𝔅𝔄.subcategory-axioms])
qed
```

10 Simple categories

10.1 Background

The section presents a variety of simple categories, (such as the empty category *0* and the singleton category *1*) and functors between them (see [7] for further information).

10.2 Empty category *0*

10.2.1 Definition and elementary properties

See Chapter I-2 in [7].

definition *cat-0* :: *V*
  where *cat-0* = [*0*, *0*, *0*, *0*, *0*, *0*]$_\circ$

Components.

lemma *cat-0-components*:
  shows *cat-0*(|*Obj*|) = *0*
    and *cat-0*(|*Arr*|) = *0*
    and *cat-0*(|*Dom*|) = *0*
    and *cat-0*(|*Cod*|) = *0*
    and *cat-0*(|*Comp*|) = *0*
    and *cat-0*(|*CId*|) = *0*
  unfolding *cat-0-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-cat-0*: *cat-smc cat-0* = *smc-0*
  unfolding *cat-smc-def cat-0-def smc-0-def dg-field-simps*
  by (*simp add*: *nat-omega-simps*)

lemmas-with (**in** $\mathcal{Z}$) [*folded cat-smc-cat-0*, *unfolded slicing-simps*]:
  cat-0-is-arr-iff = *smc-0-is-arr-iff*

10.2.2 *0* is a category

lemma (**in** $\mathcal{Z}$) *category-cat-0*[*cat-cs-intros*]: *category* α *cat-0*
proof(*intro categoryI*)
  show *vfsequence cat-0 vcard cat-0* = $6_{\mathbb{N}}$
    by (*simp-all add*: *cat-0-def nat-omega-simps*)
qed
  (
    auto simp:
      $\mathcal{Z}$*-axioms*
      cat-0-components
      cat-0-is-arr-iff
      cat-smc-cat-0
      $\mathcal{Z}$*.semicategory-smc-0*
)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$*.category-cat-0*

10.2.3 Opposite of the category *0*

lemma *op-cat-cat-0*[*cat-op-simps*]: *op-cat* (*cat-0*) = *cat-0*
proof(*rule cat-smc-eqI*)
  define β **where** $\beta = \omega + \omega$
  interpret β: $\mathcal{Z}$ β **unfolding** β*-def* **by** (*rule* $\mathcal{Z}$*-*$\omega\omega$)

show *category* β (*op-cat cat-0*)
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)
show *category* β *cat-0* **by** (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
qed
(
simp-all add:
cat-0-components op-cat-components cat-smc-cat-0
slicing-commute[*symmetric*] *smc-op-simps*
)

10.3 Empty functors

10.3.1 Definition and elementary properties

definition *cf-0* :: $V \Rightarrow V$
where *cf-0* $\mathfrak{A}$ = [*0*, *0*, *cat-0*, $\mathfrak{A}$]$_\circ$

Components.

lemma *cf-0-components*:
shows *cf-0* $\mathfrak{A}(\!|$*ObjMap*$|\!)$ = *0*
and *cf-0* $\mathfrak{A}(\!|$*ArrMap*$|\!)$ = *0*
and *cf-0* $\mathfrak{A}(\!|$*HomDom*$|\!)$ = *cat-0*
and *cf-0* $\mathfrak{A}(\!|$*HomCod*$|\!)$ = $\mathfrak{A}$
unfolding *cf-0-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cf-smcf-cf-0*: *cf-smcf* (*cf-0* $\mathfrak{A}$) = *smcf-0* (*cat-smc* $\mathfrak{A}$)
unfolding
dg-field-simps dghm-field-simps
cf-smcf-def cf-0-def smc-0-def cat-0-def smcf-0-def cat-smc-def
by (*simp add*: *nat-omega-simps*)

Opposite empty category homomorphism.

lemma *op-cf-cf-0*: *op-cf* (*cf-0* $\mathfrak{C}$) = *cf-0* (*op-cat* $\mathfrak{C}$)
unfolding
cf-0-def op-cat-def op-cf-def cat-0-def dghm-field-simps dg-field-simps
by (*simp add*: *nat-omega-simps*)

10.3.2 Object map

lemma *cf-0-ObjMap-vsv*[*cat-cs-intros*]: *vsv* (*cf-0* $\mathfrak{C}(\!|$*ObjMap*$|\!)$)
unfolding *cf-0-components* **by** *simp*

10.3.3 Arrow map

lemma *cf-0-ArrMap-vsv*[*cat-cs-intros*]: *vsv* (*cf-0* $\mathfrak{C}(\!|$*ArrMap*$|\!)$)
unfolding *cf-0-components* **by** *simp*

10.3.4 Empty functor is a faithful functor

lemma *cf-0-is-ft-functor*:
assumes *category* α $\mathfrak{A}$
shows *cf-0* $\mathfrak{A}$: *cat-0* $\mapsto\mapsto_{C.faithful\alpha}$ $\mathfrak{A}$
proof(*rule is-ft-functorI*)
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
show *cf-0* $\mathfrak{A}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
proof(*rule is-functorI*, *unfold cat-smc-cat-0 cf-smcf-cf-0*)
show *vfsequence* (*cf-0* $\mathfrak{A}$) **unfolding** *cf-0-def* **by** *simp*

```
    show vcard (cf-0 𝔄) = ℵ₀
      unfolding cf-0-def by (simp add: nat-omega-simps)
    from 𝒵.smcf-0-is-ft-semifunctor assms show
      smcf-0 (cat-smc 𝔄) : smc-0 ↦↦_{SMCα} cat-smc 𝔄
      by auto
  qed (auto simp: assms 𝔄.category-cat-0 cat-0-components cf-0-components)
  show cf-smcf (cf-0 𝔄) : cat-smc cat-0 ↦↦_{SMC.faithfulα} cat-smc 𝔄
    by
      (
        auto simp:
          assms
          𝔄.𝒵-axioms
          𝔄.smcf-0-is-ft-semifunctor
          category.cat-semicategory
          cf-smcf-cf-0
          cat-smc-cat-0
      )
qed

lemma cf-0-is-ft-functor'[cf-cs-intros]:
  assumes category α 𝔄
    and 𝔅' = 𝔄
    and 𝔄' = cat-0
  shows cf-0 𝔄 : 𝔄' ↦↦_{C.faithfulα} 𝔅'
  using assms(1) unfolding assms(2,3) by (rule cf-0-is-ft-functor)

lemma cf-0-is-functor:
  assumes category α 𝔄
  shows cf-0 𝔄 : cat-0 ↦↦_{Cα} 𝔄
  using cf-0-is-ft-functor[OF assms] by auto

lemma cf-0-is-functor'[cat-cs-intros]:
  assumes category α 𝔄
    and 𝔅' = 𝔄
    and 𝔄' = cat-0
  shows cf-0 𝔄 : 𝔄' ↦↦_{Cα} 𝔅'
  using assms(1) unfolding assms(2,3) by (rule cf-0-is-functor)
```

10.3.5 Further properties

```
lemma is-functor-is-cf-0-if-cat-0:
  assumes 𝔉 : cat-0 ↦↦_{Cα} ℭ
  shows 𝔉 = cf-0 ℭ
proof(rule cf-smcf-eqI)
  interpret 𝔉: is-functor α cat-0 ℭ 𝔉 by (rule assms(1))
  show 𝔉 : cat-0 ↦↦_{Cα} ℭ by (rule assms(1))
  then have dom-lhs: 𝒟∘ (𝔉(|ObjMap|)) = 0 𝒟∘ (𝔉(|ArrMap|)) = 0
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-0-components)+
  show cf-0 ℭ : cat-0 ↦↦_{Cα} ℭ
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show cf-smcf 𝔉 = cf-smcf (cf-0 ℭ)
    unfolding cf-smcf-cf-0
    by
      (
        rule is-semifunctor-is-smcf-0-if-smc-0,
        rule 𝔉.cf-is-semifunctor[unfolded slicing-simps cat-smc-cat-0]
      )
qed simp-all
```

lemma (in *is-functor*) *cf-comp-cf-cf-0*[*cat-cs-simps*]: $\mathfrak{F} \circ_{CF}$ *cf-0* $\mathfrak{A}$ = *cf-0* $\mathfrak{B}$
proof(*rule cf-eqI*)
  show $\mathfrak{F} \circ_{CF}$ *cf-0* $\mathfrak{A}$: *cat-0* $\mapsto\!\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$ **by** (*cs-concl* **cs-intro**: *cat-cs-intros*)
  then have *ObjMap-dom-lhs*: $\mathcal{D}_\circ$ (($\mathfrak{F} \circ_{CF}$ *cf-0* $\mathfrak{A}$)(⟦*ObjMap*⟧)) = *cat-0*(⟦*Obj*⟧)
    and *ArrMap-dom-lhs*: $\mathcal{D}_\circ$ (($\mathfrak{F} \circ_{CF}$ *cf-0* $\mathfrak{A}$)(|*ArrMap*|)) = *cat-0*(|*Arr*|)
    by (*cs-concl* **cs-simp**: *cat-cs-simps*)+
  show *cf-0* $\mathfrak{B}$: *cat-0* $\mapsto\!\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
    by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
  then have *ObjMap-dom-rhs*: $\mathcal{D}_\circ$ (*cf-0* $\mathfrak{B}$(⟦*ObjMap*⟧)) = *cat-0*(⟦*Obj*⟧)
    and *ArrMap-dom-rhs*: $\mathcal{D}_\circ$ (*cf-0* $\mathfrak{B}$(|*ArrMap*|)) = *cat-0*(|*Arr*|)
    by (*cs-concl* **cs-simp**: *cat-cs-simps*)+
  show ($\mathfrak{F} \circ_{CF}$ *cf-0* $\mathfrak{A}$)(⟦*ObjMap*⟧) = *cf-0* $\mathfrak{B}$(⟦*ObjMap*⟧)
    by
      (
        rule vsv-eqI,
        unfold ObjMap-dom-lhs ObjMap-dom-rhs ArrMap-dom-lhs ArrMap-dom-rhs
)
      (*auto simp*: *cat-0-components intro*: *cat-cs-intros*)
  show ($\mathfrak{F} \circ_{CF}$ *cf-0* $\mathfrak{A}$)(|*ArrMap*|) = *cf-0* $\mathfrak{B}$(|*ArrMap*|)
    by
      (
        rule vsv-eqI,
        unfold ObjMap-dom-lhs ObjMap-dom-rhs ArrMap-dom-lhs ArrMap-dom-rhs
)
      (*auto simp*: *cat-0-components intro*: *cat-cs-intros*)
qed *simp-all*

lemmas [*cat-cs-simps*] = *is-functor.cf-comp-cf-cf-0*

10.4 Empty natural transformation

10.4.1 Definition and elementary properties

See Chapter X-1 in [7].

definition *ntcf-0* :: *V* ⇒ *V*
  where *ntcf-0* $\mathfrak{C}$ = [*0*, *cf-0* $\mathfrak{C}$, *cf-0* $\mathfrak{C}$, *cat-0*, $\mathfrak{C}$]$_\circ$

Components.

lemma *ntcf-0-components*:
  shows *ntcf-0* $\mathfrak{C}$(|*NTMap*|) = *0*
    and [*cat-cs-simps*]: *ntcf-0* $\mathfrak{C}$(|*NTDom*|) = *cf-0* $\mathfrak{C}$
    and [*cat-cs-simps*]: *ntcf-0* $\mathfrak{C}$(|*NTCod*|) = *cf-0* $\mathfrak{C}$
    and [*cat-cs-simps*]: *ntcf-0* $\mathfrak{C}$(|*NTDGDom*|) = *cat-0*
    and [*cat-cs-simps*]: *ntcf-0* $\mathfrak{C}$(|*NTDGCod*|) = $\mathfrak{C}$
  unfolding *ntcf-0-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *ntcf-ntsmcf-ntcf-0*: *ntcf-ntsmcf* (*ntcf-0* $\mathfrak{A}$) = *ntsmcf-0* (*cat-smc* $\mathfrak{A}$)
  unfolding
    ntcf-ntsmcf-def ntcf-0-def ntsmcf-0-def cat-smc-def
    cf-smcf-def smcf-0-def cf-0-def cat-0-def smc-0-def
    dg-field-simps dghm-field-simps nt-field-simps
  by (*simp add*: *nat-omega-simps*)

Duality.

lemma *op-ntcf-ntcf-0*: *op-ntcf* (*ntcf-0* $\mathfrak{C}$) = *ntcf-0* (*op-cat* $\mathfrak{C}$)

by
(
simp-all add:
op-ntcf-def ntcf-0-def op-cat-def op-cf-cf-0 cat-0-def
nt-field-simps dg-field-simps nat-omega-simps
)

10.4.2 Natural transformation map

lemma *ntcf-0-NTMap-vsv*[*cat-cs-intros*]: *vsv* (*ntcf-0* $\mathfrak{C}$(|*NTMap*|))
unfolding *ntcf-0-components* **by** *simp*

lemma *ntcf-0-NTMap-vdomain*[*cat-cs-simps*]: $\mathcal{D}_\circ$ (*ntcf-0* $\mathfrak{C}$(|*NTMap*|)) = *0*
unfolding *ntcf-0-components* **by** *simp*

lemma *ntcf-0-NTMap-vrange*[*cat-cs-simps*]: $\mathcal{R}_\circ$ (*ntcf-0* $\mathfrak{C}$(|*NTMap*|)) = *0*
unfolding *ntcf-0-components* **by** *simp*

10.4.3 Empty natural transformation is a natural transformation

lemma (**in** *category*) *cat-ntcf-0-is-ntcfI*:
ntcf-0 $\mathfrak{C}$: *cf-0* $\mathfrak{C}$ $\mapsto_{CF}$ *cf-0* $\mathfrak{C}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
proof(*intro is-ntcfI*)
show *vfsequence* (*ntcf-0* $\mathfrak{C}$) **unfolding** *ntcf-0-def* **by** *simp*
show *vcard* (*ntcf-0* $\mathfrak{C}$) = $5_{\mathbb{N}}$
unfolding *ntcf-0-def* **by** (*simp add*: *nat-omega-simps*)
show *ntcf-ntsmcf* (*ntcf-0* $\mathfrak{C}$) :
cf-smcf (*cf-0* $\mathfrak{C}$) $\mapsto_{SMCF}$ *cf-smcf* (*cf-0* $\mathfrak{C}$) :
cat-smc cat-0 $\mapsto\mapsto_{SMC\alpha}$ *cat-smc* $\mathfrak{C}$
unfolding *ntcf-ntsmcf-ntcf-0 cf-smcf-cf-0 cat-smc-cat-0*
by (*cs-concl* **cs-shallow cs-intro**: *smc-cs-intros slicing-intros*)
qed (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)+

lemma (**in** *category*) *cat-ntcf-0-is-ntcfI'*[*cat-cs-intros*]:
assumes $\mathfrak{F}'$ = *cf-0* $\mathfrak{C}$
and $\mathfrak{G}'$ = *cf-0* $\mathfrak{C}$
and $\mathfrak{A}'$ = *cat-0*
and $\mathfrak{B}'$ = $\mathfrak{C}$
and $\mathfrak{F}'$ = $\mathfrak{F}$
and $\mathfrak{G}'$ = $\mathfrak{G}$
shows *ntcf-0* $\mathfrak{C}$: $\mathfrak{F}'$ $\mapsto_{CF}$ $\mathfrak{G}'$: $\mathfrak{A}'$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule cat-ntcf-0-is-ntcfI*)

lemmas [*cat-cs-intros*] = *category.cat-ntcf-0-is-ntcfI'*

lemma *is-ntcf-is-ntcf-0-if-cat-0*:
assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
shows $\mathfrak{N}$ = *ntcf-0* $\mathfrak{C}$ **and** $\mathfrak{F}$ = *cf-0* $\mathfrak{C}$ **and** $\mathfrak{G}$ = *cf-0* $\mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-ntcf* α *cat-0* $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
note *is-ntsmcf-is-ntsmcf-0-if-smc-0* = *is-ntsmcf-is-ntsmcf-0-if-smc-0*
[
OF $\mathfrak{N}$.*ntcf-is-ntsmcf*[*unfolded cat-smc-cat-0*],
folded smcf-dghm-smcf-0 ntsmcf-tdghm-ntsmcf-0
]
show $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-0* $\mathfrak{C}$ **and** $\mathfrak{G}$-*def*: $\mathfrak{G}$ = *cf-0* $\mathfrak{C}$
by (*all*‹*intro is-functor-is-cf-0-if-cat-0*›)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)+

```
  show 𝔑 = ntcf-0 𝔈
  proof(rule ntcf-ntsmcf-eqI)
    show 𝔑 : 𝔉 ↦_CF 𝔊 : cat-0 ↦↦_Cα 𝔈 by (rule assms(1))
    show ntcf-0 𝔈 : 𝔉 ↦_CF 𝔊 : cat-0 ↦↦_Cα 𝔈
      by (cs-concl cs-shallow cs-simp: 𝔉-def 𝔊-def cs-intro: cat-cs-intros)
  qed
    (
      simp-all add:
        𝔉-def 𝔊-def is-ntsmcf-is-ntsmcf-0-if-smc-0 ntcf-ntsmcf-ntcf-0
    )
qed
```

10.4.4 Further properties

```
lemma ntcf-vcomp-ntcf-ntcf-0[cat-cs-simps]:
  assumes 𝔑 : 𝔉 ↦_CF 𝔊 : cat-0 ↦↦_Cα 𝔈
  shows 𝔑 ∙_NTCF ntcf-0 𝔈 = ntcf-0 𝔈
proof(rule ntcf-ntsmcf-eqI)
  interpret 𝔑: is-ntcf α cat-0 𝔈 𝔉 𝔊 𝔑 by (rule assms(1))
  show 𝔑 ∙_NTCF ntcf-0 𝔈 : cf-0 𝔈 ↦_CF cf-0 𝔈 : cat-0 ↦↦_Cα 𝔈
    unfolding is-ntcf-is-ntcf-0-if-cat-0[OF assms]
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ntcf-0 𝔈 : cf-0 𝔈 ↦_CF cf-0 𝔈 : cat-0 ↦↦_Cα 𝔈
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ntcf-ntsmcf (𝔑 ∙_NTSMCF ntcf-0 𝔈) = ntcf-ntsmcf (ntcf-0 𝔈)
    unfolding
      slicing-commute[symmetric]
      ntsmcf-vcomp-ntsmcf-ntsmcf-0
        [
          OF 𝔑.ntcf-is-ntsmcf[unfolded cat-smc-cat-0],
          folded ntcf-ntsmcf-ntcf-0
        ]
      ..
qed simp-all

lemma ntcf-vcomp-ntcf-0-ntcf[cat-cs-simps]:
  assumes 𝔑 : 𝔉 ↦_CF 𝔊 : cat-0 ↦↦_Cα 𝔈
  shows ntcf-0 𝔈 ∙_NTCF 𝔑 = ntcf-0 𝔈
proof(rule ntcf-ntsmcf-eqI)
  interpret 𝔑: is-ntcf α cat-0 𝔈 𝔉 𝔊 𝔑 by (rule assms(1))
  show ntcf-0 𝔈 ∙_NTCF 𝔑 : cf-0 𝔈 ↦_CF cf-0 𝔈 : cat-0 ↦↦_Cα 𝔈
    unfolding is-ntcf-is-ntcf-0-if-cat-0[OF assms]
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ntcf-0 𝔈 : cf-0 𝔈 ↦_CF cf-0 𝔈 : cat-0 ↦↦_Cα 𝔈
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  show ntcf-ntsmcf (ntcf-0 𝔈 ∙_NTCF 𝔑) = ntcf-ntsmcf (ntcf-0 𝔈)
    unfolding
      slicing-commute[symmetric]
      ntsmcf-vcomp-ntsmcf-0-ntsmcf
        [
          OF 𝔑.ntcf-is-ntsmcf[unfolded cat-smc-cat-0],
          folded ntcf-ntsmcf-ntcf-0
        ]
      ..
  qed simp-all

lemma (in is-functor) cf-ntcf-comp-cf-ntcf-0[cat-cs-simps]:
  𝔉 ∘_CF-NTCF ntcf-0 𝔄 = ntcf-0 𝔅
```

proof(*rule ntcf-eqI*)
show $\mathfrak{F} \circ_{CF-NTCF} ntcf\text{-}0\ \mathfrak{A} : cf\text{-}0\ \mathfrak{B} \mapsto_{CF} cf\text{-}0\ \mathfrak{B} : cat\text{-}0 \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_\circ\ ((\mathfrak{F} \circ_{CF-NTCF} ntcf\text{-}0\ \mathfrak{A})(\lfloor NTMap \rfloor)) = cat\text{-}0(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
show $ntcf\text{-}0\ \mathfrak{B} : cf\text{-}0\ \mathfrak{B} \mapsto_{CF} cf\text{-}0\ \mathfrak{B} : cat\text{-}0 \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_\circ\ (ntcf\text{-}0\ \mathfrak{B}(\lfloor NTMap \rfloor)) = cat\text{-}0(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
show $(\mathfrak{F} \circ_{CF-NTCF} ntcf\text{-}0\ \mathfrak{A})(\lfloor NTMap \rfloor) = ntcf\text{-}0\ \mathfrak{B}(\lfloor NTMap \rfloor)$
by (*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
(*auto simp*: *cat-0-components intro*!: *cat-cs-intros*)+
qed *simp-all*

lemmas [*cat-cs-simps*] = *is-functor.cf-ntcf-comp-cf-ntcf-0*

10.5 *1*: category with one object and one arrow

10.5.1 Definition and elementary properties

See Chapter I-2 in [7].

definition *cat-1* :: $V \Rightarrow V \Rightarrow V$
where *cat-1* $\mathfrak{a}\ \mathfrak{f}$ =
[
set $\{\mathfrak{a}\}$,
set $\{\mathfrak{f}\}$,
set $\{\langle \mathfrak{f}, \mathfrak{a} \rangle\}$,
set $\{\langle \mathfrak{f}, \mathfrak{a} \rangle\}$,
set $\{\langle [\mathfrak{f}, \mathfrak{f}]_\circ, \mathfrak{f} \rangle\}$,
set $\{\langle \mathfrak{a}, \mathfrak{f} \rangle\}$
]$_\circ$

Components.

lemma *cat-1-components*:
shows *cat-1* $\mathfrak{a}\ \mathfrak{f}(\lfloor Obj \rfloor)$ = *set* $\{\mathfrak{a}\}$
and *cat-1* $\mathfrak{a}\ \mathfrak{f}(\lfloor Arr \rfloor)$ = *set* $\{\mathfrak{f}\}$
and *cat-1* $\mathfrak{a}\ \mathfrak{f}(\lfloor Dom \rfloor)$ = *set* $\{\langle \mathfrak{f}, \mathfrak{a} \rangle\}$
and *cat-1* $\mathfrak{a}\ \mathfrak{f}(\lfloor Cod \rfloor)$ = *set* $\{\langle \mathfrak{f}, \mathfrak{a} \rangle\}$
and *cat-1* $\mathfrak{a}\ \mathfrak{f}(\lfloor Comp \rfloor)$ = *set* $\{\langle [\mathfrak{f}, \mathfrak{f}]_\circ, \mathfrak{f} \rangle\}$
and *cat-1* $\mathfrak{a}\ \mathfrak{f}(\lfloor CId \rfloor)$ = *set* $\{\langle \mathfrak{a}, \mathfrak{f} \rangle\}$
unfolding *cat-1-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-cat-1*: *cat-smc* (*cat-1* $\mathfrak{a}\ \mathfrak{f}$) = *smc-1* $\mathfrak{a}\ \mathfrak{f}$
unfolding *cat-smc-def cat-1-def smc-1-def dg-field-simps*
by (*simp add*: *nat-omega-simps*)

lemmas-with [*folded smc-cat-1*, *unfolded slicing-simps*]:
cat-1-is-arrI = *smc-1-is-arrI*
and *cat-1-is-arrD* = *smc-1-is-arrD*
and *cat-1-is-arrE* = *smc-1-is-arrE*
and *cat-1-is-arr-iff* = *smc-1-is-arr-iff*
and *cat-1-Comp-app*[*cat-cs-simps*] = *smc-1-Comp-app*

10.5.2 Object

lemma *cat-1-ObjI*[*cat-cs-intros*]:

assumes $a = \mathfrak{a}$
shows $a \in_{\circ} \mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f}\ (\!|\mathit{Obj}|\!)$
unfolding *cat-1-components*(*1*) *assms* **by** *simp*

10.5.3 Identity

lemma *cat-1-CId-app*: $\mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f}(\!|\mathit{CId}|\!)(\!|\mathfrak{a}|\!) = \mathfrak{f}$
unfolding *cat-1-components* **by** *simp*

10.5.4 *1* is a category

lemma (**in** $\mathcal{Z}$) *category-cat-1*:
assumes $\mathfrak{a} \in_{\circ} \mathit{Vset}\ \alpha$ **and** $\mathfrak{f} \in_{\circ} \mathit{Vset}\ \alpha$
shows *category* α (*cat-1* $\mathfrak{a}\ \mathfrak{f}$)
proof(*intro categoryI*, *unfold smc-cat-1*)
show *vfsequence* (*cat-1* $\mathfrak{a}\ \mathfrak{f}$)
unfolding *cat-1-def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* (*cat-1* $\mathfrak{a}\ \mathfrak{f}$) = $6_{\mathbb{N}}$
unfolding *cat-1-def* **by** (*simp add*: *nat-omega-simps*)
qed (*auto simp*: *assms semicategory-smc-1 cat-1-is-arr-iff cat-1-components*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$.*category-cat-1*

lemma (**in** $\mathcal{Z}$) *finite-category-cat-1*:
assumes $\mathfrak{a} \in_{\circ} \mathit{Vset}\ \alpha$ **and** $\mathfrak{f} \in_{\circ} \mathit{Vset}\ \alpha$
shows *finite-category* α (*cat-1* $\mathfrak{a}\ \mathfrak{f}$)
by (*intro finite-categoryI'*)
(*auto simp*: *cat-1-components intro*: *category-cat-1*[*OF assms*])

lemmas [*cat-small-cs-intros*] = $\mathcal{Z}$.*finite-category-cat-1*

10.5.5 Opposite of the category *1*

lemma (**in** $\mathcal{Z}$) *cat-1-op*[*cat-op-simps*]:
assumes $\mathfrak{a} \in_{\circ} \mathit{Vset}\ \alpha$ **and** $\mathfrak{f} \in_{\circ} \mathit{Vset}\ \alpha$
shows *op-cat* (*cat-1* $\mathfrak{a}\ \mathfrak{f}$) = *cat-1* $\mathfrak{a}\ \mathfrak{f}$
proof(*rule cat-eqI*, *unfold cat-op-simps*)
from *assms* **show** *category* α (*op-cat* (*cat-1* $\mathfrak{a}\ \mathfrak{f}$))
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)
from *assms* **show** *category* α (*cat-1* $\mathfrak{a}\ \mathfrak{f}$)
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show *op-cat* (*cat-1* $\mathfrak{a}\ \mathfrak{f}$)$(\!|\mathit{Comp}|\!)$ = *cat-1* $\mathfrak{a}\ \mathfrak{f}(\!|\mathit{Comp}|\!)$
unfolding *cat-1-components op-cat-components fflip-vsingleton* **..**
qed (*simp-all add*: *cat-1-components*)

lemma (**in** $\mathcal{Z}$) *cat-1-op-0*[*cat-op-simps*]: *op-cat* (*cat-1 0 0*) = *cat-1 0 0*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *V-cs-intros cat-cs-intros*
)

10.5.6 Further properties

lemma *cf-const-if-HomCod-is-cat-1*:
assumes $\mathfrak{K} : \mathfrak{B} \mapsto\!\!\mapsto_{C\alpha} \mathit{cat\text{-}1}\ \mathfrak{a}\ \mathfrak{f}$
shows $\mathfrak{K}$ = *cf-const* $\mathfrak{B}$ (*cat-1* $\mathfrak{a}\ \mathfrak{f}$) $\mathfrak{a}$
proof(*rule cf-eqI*)
interpret $\mathfrak{K}$: *is-functor* $\alpha\ \mathfrak{B}$ ‹*cat-1* $\mathfrak{a}\ \mathfrak{f}$› $\mathfrak{K}$ **by** (*rule assms*(*1*))

```
  show cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞 : 𝔙 ↦↦_Cα cat-1 𝔞 𝔣
    by (cs-concl cs-intro: cat-cs-intros)
  have ObjMap-dom-lhs: 𝒟∘ (ℜ(|ObjMap|)) = 𝔙(|Obj|) by (simp add: cat-cs-simps)
  have ObjMap-dom-rhs: 𝒟∘ (cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞(|ObjMap|)) = 𝔙(|Obj|)
    by (simp add: cat-cs-simps)
  have ArrMap-dom-lhs: 𝒟∘ (ℜ(|ArrMap|)) = 𝔙(|Arr|) by (simp add: cat-cs-simps)
  have ArrMap-dom-rhs: 𝒟∘ (cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞(|ArrMap|)) = 𝔙(|Arr|)
    by (simp add: cat-cs-simps)
  show ℜ(|ObjMap|) = cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞(|ObjMap|)
  proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
    fix a assume prems: a ∈∘ 𝔙(|Obj|)
    then have ℜ(|ObjMap|)(|a|) ∈∘ cat-1 𝔞 𝔣(|Obj|)
      by (auto intro: ℜ.cf-ObjMap-app-in-HomCod-Obj)
    then have ℜ(|ObjMap|)(|a|) = 𝔞 by (auto simp: cat-1-components)
    with prems show ℜ(|ObjMap|)(|a|) = cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞(|ObjMap|)(|a|)
      by (auto simp: cat-cs-simps)
  qed (auto intro: cat-cs-intros)
  show ℜ(|ArrMap|) = cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix a assume prems: a ∈∘ 𝔙(|Arr|)
    then have ℜ(|ArrMap|)(|a|) ∈∘ cat-1 𝔞 𝔣(|Arr|)
      by (auto intro: ℜ.cf-ArrMap-app-in-HomCod-Arr)
    then have ℜ(|ArrMap|)(|a|) = 𝔣 by (auto simp: cat-1-components)
    with prems show ℜ(|ArrMap|)(|a|) = cf-const 𝔙 (cat-1 𝔞 𝔣) 𝔞(|ArrMap|)(|a|)
      by (auto simp: cat-1-CId-app cat-cs-simps)
  qed (auto intro: cat-cs-intros)
qed (simp-all add: assms)

lemma cf-const-if-HomDom-is-cat-1:
  assumes ℜ : cat-1 𝔞 𝔣 ↦↦_Cα ℭ
  shows ℜ = cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))
proof–

  interpret ℜ: is-functor α ‹cat-1 𝔞 𝔣› ℭ ℜ by (rule assms(1))

  from cat-1-components(1) have 𝔞: 𝔞 ∈∘ Vset α
    by (auto simp: ℜ.HomDom.cat-in-Obj-in-Vset)
  from cat-1-components(2) have 𝔣: 𝔣 ∈∘ Vset α
    by (auto simp: ℜ.HomDom.cat-in-Arr-in-Vset)

  from 𝔞 𝔣 interpret cf-1:
    is-tm-functor α ‹cat-1 𝔞 𝔣› ℭ ‹cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))›
    by (cs-concl cs-intro: cat-small-cs-intros cat-cs-intros)

  show ?thesis
  proof(rule cf-eqI)
    show cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|)) : cat-1 𝔞 𝔣 ↦↦_Cα ℭ
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    have ObjMap-dom-lhs: 𝒟∘ (ℜ(|ObjMap|)) = set {𝔞}
      by (simp add: cat-cs-simps cat-1-components)
    have ObjMap-dom-rhs:
      𝒟∘ (cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))(|ObjMap|)) = set {𝔞}
      by (simp add: cat-cs-simps cat-1-components)
    show ℜ(|ObjMap|) = cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))(|ObjMap|)
    proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
      fix a assume a ∈∘ set {𝔞}
      then have a-def: a = 𝔞 by simp
      show ℜ(|ObjMap|)(|a|) = cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))(|ObjMap|)(|a|)
```

```
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-1-components(1) cat-cs-simps a-def
            cs-intro: V-cs-intros
        )
    qed auto

    have ArrMap-dom-lhs: 𝒟₀ (ℜ(|ArrMap|)) = set {𝔣}
      by (simp add: cat-cs-simps cat-1-components)
    have ArrMap-dom-rhs:
      𝒟₀ (cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))(|ArrMap|)) = set {𝔣}
      by (simp add: cat-cs-simps cat-1-components)

    show ℜ(|ArrMap|) = cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))(|ArrMap|)
    proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
      fix f assume f ∈₀ set {𝔣}
      then have f-def: f = 𝔣 by simp
      show ℜ(|ArrMap|)(|f|) = cf-const (cat-1 𝔞 𝔣) ℭ (ℜ(|ObjMap|)(|𝔞|))(|ArrMap|)(|f|)
        unfolding f-def
        by (subst cat-1-CId-app[symmetric, of 𝔣 𝔞])
          (
            cs-concl cs-shallow
              cs-simp: cat-1-components(1,2) cat-cs-simps
              cs-intro: V-cs-intros cat-cs-intros
          )
    qed auto

  qed (simp-all add: assms)

qed
```

11 Discrete category

11.1 Abstract discrete category

named-theorems *cat-discrete-cs-simps*
named-theorems *cat-discrete-cs-intros*

11.1.1 Definition and elementary properties

See Chapter I-2 in [7].

locale *cat-discrete* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *cat-discrete-Arr*: $f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \Longrightarrow f \in_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{C}(\!|CId|\!))$

Rules.

lemma (**in** *cat-discrete*)
  assumes $\alpha' = \alpha$ $\mathfrak{C}' = \mathfrak{C}$
  shows *cat-discrete* α' $\mathfrak{C}'$
  unfolding *assms* **by** (*rule cat-discrete-axioms*)

mk-ide rf *cat-discrete-def*[*unfolded cat-discrete-axioms-def*]
  |*intro cat-discreteI*|
  |*dest cat-discreteD*[*dest*]|
  |*elim cat-discreteE*[*elim*]|

lemmas [*cat-discrete-cs-intros*] = *cat-discreteD*(*1*)

Elementary properties.

lemma (**in** *cat-discrete*) *cat-discrete-is-arrD*[*dest*]:
  assumes $f : a \mapsto_{\mathfrak{C}} b$
  shows $b = a$ **and** $f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
proof–
  from *assms cat-discrete-Arr* **have** $f \in_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{C}(\!|CId|\!))$
    by (*auto simp*: *cat-cs-simps*)
  with *cat-CId-vdomain* **obtain** a' **where** *f-def*: $f = \mathfrak{C}(\!|CId|\!)(\!|a'|\!)$ **and** $a' \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
    by (*blast dest*: *CId.vrange-atD*)
  then have $f : a' \mapsto_{\mathfrak{C}} a'$ **by** (*auto intro*: *cat-CId-is-arr'*)
  with *assms* **have** $a = a'$ **and** $b = a'$ **by** *blast+*
  with *f-def* **show** $b = a$ **and** $f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$ **by** *auto*
qed

lemma (**in** *cat-discrete*) *cat-discrete-is-arrE*[*elim*]:
  assumes $f : b \mapsto_{\mathfrak{C}} c$
  obtains a **where** $f : a \mapsto_{\mathfrak{C}} a$ **and** $f = \mathfrak{C}(\!|CId|\!)(\!|a|\!)$
  using *assms* **by** *auto*

11.2 The discrete category

As explained in Chapter I-2 in [7], every discrete category is identified with its set of objects. In this work, it is assumed that the set of objects and the set of arrows in the canonical discrete category coincide; the domain and the codomain functions are identities.

11.2.1 Definition and elementary properties

definition *the-cat-discrete* :: $V \Rightarrow V$ $(\langle\cdot_C\rangle)$
  where $:_C\ I = [I, I, \textit{vid-on } I, \textit{vid-on } I, (\lambda fg \in_{\circ} \textit{fid-on } I.\ fg(\!|0|\!)), \textit{vid-on } I]_{\circ}$

Components.

lemma *the-cat-discrete-components*:
shows $:_C\ I(\!|Obj|\!) = I$
and $:_C\ I(\!|Arr|\!) = I$
and $:_C\ I(\!|Dom|\!) = vid\text{-}on\ I$
and $:_C\ I(\!|Cod|\!) = vid\text{-}on\ I$
and $:_C\ I(\!|Comp|\!) = (\lambda fg{\in_\circ}fid\text{-}on\ I.\ fg(\!|0|\!))$
and $:_C\ I(\!|CId|\!) = vid\text{-}on\ I$
unfolding *the-cat-discrete-def dg-field-simps*
by (*simp-all add*: *nat-omega-simps*)

11.2.2 Domain

mk-VLambda *the-cat-discrete-components*(*3*)[*folded VLambda-vid-on*]
|*vsv the-cat-discrete-Dom-vsv*[*cat-discrete-cs-intros*]|
|*vdomain the-cat-discrete-Dom-vdomain*[*cat-discrete-cs-simps*]|
|*app the-cat-discrete-Dom-app*[*cat-discrete-cs-simps*]|

11.2.3 Codomain

mk-VLambda *the-cat-discrete-components*(*4*)[*folded VLambda-vid-on*]
|*vsv the-cat-discrete-Cod-vsv*[*cat-discrete-cs-intros*]|
|*vdomain the-cat-discrete-Cod-vdomain*[*cat-discrete-cs-simps*]|
|*app the-cat-discrete-Cod-app*[*cat-discrete-cs-simps*]|

11.2.4 Composition

lemma *the-cat-discrete-Comp-vsv*[*cat-discrete-cs-intros*]: *vsv* $(:_C\ I(\!|Comp|\!))$
unfolding *the-cat-discrete-components* **by** *simp*

lemma *the-cat-discrete-Comp-vdomain*: $\mathcal{D}_\circ\ (:_C\ I(\!|Comp|\!)) = fid\text{-}on\ I$
unfolding *the-cat-discrete-components* **by** *simp*

lemma *the-cat-discrete-Comp-vrange*:
$\mathcal{R}_\circ\ (:_C\ I(\!|Comp|\!)) = I$
proof(*intro vsubset-antisym vsubsetI*)
fix *f* **assume** $f \in_\circ \mathcal{R}_\circ\ (:_C\ I(\!|Comp|\!))$
then obtain *gg* **where** *f-def*: $f = :_C\ I(\!|Comp|\!)(\!|gg|\!)$ **and** *gg*: $gg \in_\circ fid\text{-}on\ I$
unfolding *the-cat-discrete-components* **by** *auto*
from *gg* **show** $f \in_\circ I$
unfolding *f-def the-cat-discrete-components* **by** *clarsimp*
next
fix *f* **assume** $f \in_\circ I$
then have $[f, f]_\circ \in_\circ fid\text{-}on\ I$ **by** *clarsimp*
moreover then have $f = :_C\ I(\!|Comp|\!)(\!|f, f|\!)_\bullet$
unfolding *the-cat-discrete-components* **by** *simp*
ultimately show $f \in_\circ \mathcal{R}_\circ\ (:_C\ I(\!|Comp|\!))$
unfolding *the-cat-discrete-components*
by (*metis rel-VLambda.vsv-vimageI2 vdomain-VLambda*)
qed

lemma *the-cat-discrete-Comp-app*[*cat-discrete-cs-simps*]:
assumes $i \in_\circ I$
shows $i \circ_{A:_C\ I} i = i$
proof–
from *assms* **have** $[i, i]_\circ \in_\circ fid\text{-}on\ I$ **by** *clarsimp*
then show *?thesis* **unfolding** *the-cat-discrete-components* **by** *simp*
qed

11.2.5 Identity

mk-VLambda *the-cat-discrete-components*(*6*)[*folded VLambda-vid-on*]
|*vsv the-cat-discrete-CId-vsv*[*cat-discrete-cs-intros*]|
|*vdomain the-cat-discrete-CId-vdomain*[*cat-discrete-cs-simps*]|
|*app the-cat-discrete-CId-app*[*cat-discrete-cs-simps*]|

11.2.6 Arrow with a domain and a codomain

lemma *the-cat-discrete-is-arrI*:
assumes $i \in_{\circ} I$
shows $i : i \mapsto_{:_C\ I} i$
using *assms* **unfolding** *is-arr-def the-cat-discrete-components* **by** *simp*

lemma *the-cat-discrete-is-arrI'*[*cat-discrete-cs-intros*]:
assumes $i \in_{\circ} I$
and $a = i$
and $b = i$
shows $i : a \mapsto_{:_C\ I} b$
using *assms*(*1*) **unfolding** *assms*(*2*,*3*) **by** (*rule the-cat-discrete-is-arrI*)

lemma *the-cat-discrete-is-arrD*:
assumes $f : a \mapsto_{:_C\ I} b$
shows $f : f \mapsto_{:_C\ I} f$
and $a : a \mapsto_{:_C\ I} a$
and $b : b \mapsto_{:_C\ I} b$
and $f \in_{\circ} I$
and $a \in_{\circ} I$
and $b \in_{\circ} I$
and $f = a$
and $f = b$
and $b = a$
using *assms* **unfolding** *is-arr-def the-cat-discrete-components* **by** *force+*

11.2.7 The discrete category is a discrete category

lemma (**in** $\mathcal{Z}$) *cat-discrete-the-cat-discrete*:
assumes $I \subseteq_{\circ} Vset\ \alpha$
shows *cat-discrete* α $(:_C\ I)$
proof(*intro cat-discreteI categoryI'*)
show *vfsequence* $(:_C\ I)$ **unfolding** *the-cat-discrete-def* **by** *simp*
show *vcard* $(:_C\ I) = 6_{\mathbb{N}}$
unfolding *the-cat-discrete-def* **by** (*simp add*: *nat-omega-simps*)
show $gf \in_{\circ} \mathcal{D}_{\circ}\ (:_C\ I(\!|Comp|\!)) \longleftrightarrow$
$(\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{:_C\ I} c \wedge f : a \mapsto_{:_C\ I} b)$
for *gf*
unfolding *the-cat-discrete-Comp-vdomain*
proof
assume $gf \in_{\circ}$ *fid-on I*
then obtain *a* **where** $gf = [a, a]_{\circ}$ **and** $a \in_{\circ} I$ **by** *clarsimp*
moreover then have $a : a \mapsto_{:_C\ I} a$
by (*auto intro*: *the-cat-discrete-is-arrI*)
ultimately show
$\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{:_C\ I} c \wedge f : a \mapsto_{:_C\ I} b$
by *auto*
next
assume $\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{:_C\ I} c \wedge f : a \mapsto_{:_C\ I} b$
then obtain *g f b c a* **where** *gf-def*: $gf = [g, f]_{\circ}$
and *g*: $g : b \mapsto_{:_C\ I} c$

```
    and f: f : a ↦_:C I b
    by clarsimp
  then have g = f by (metis is-arrE the-cat-discrete-is-arrD(1))
  with the-cat-discrete-is-arrD(4)[OF f] show gf ∈∘ fid-on I
    unfolding gf-def by clarsimp
qed
show g ∘_A:C I f : a ↦_:C I c if g : b ↦_:C I c and f : a ↦_:C I b
  for g b c f a
proof–
  from that have fba: f = a b = a and a: a ∈∘ I
    unfolding the-cat-discrete-is-arrD[OF that(2)] by (simp-all add: ‹a ∈∘ I›)
  from that have gcb: g = b c = b
    unfolding the-cat-discrete-is-arrD[OF that(1)] by simp-all
  from a show ?thesis
    unfolding fba gcb
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-discrete-cs-simps cs-intro: cat-discrete-cs-intros
      )
qed
show h ∘_A:C I g ∘_A:C I f = h ∘_A:C I (g ∘_A:C I f)
  if h : c ↦_:C I d and g : b ↦_:C I c and f : a ↦_:C I b
  for h c d g b f a
proof–
  from that have fba: f = a b = a and a: a ∈∘ I
    unfolding the-cat-discrete-is-arrD[OF that(3)] by (simp-all add: ‹a ∈∘ I›)
  from that have gcb: g = b c = b
    unfolding the-cat-discrete-is-arrD[OF that(2)] by simp-all
  from that have hcd: h = c d = c
    unfolding the-cat-discrete-is-arrD[OF that(1)] by simp-all
  from a show ?thesis
    unfolding fba gcb hcd
    by (cs-concl cs-shallow cs-simp: cat-discrete-cs-simps)
qed
show :C I (|CId|) (|b|) ∘_A:C I f = f if f : a ↦_:C I b for f a b
proof–
  from that have fba: f = a b = a and a: a ∈∘ I
    unfolding the-cat-discrete-is-arrD[OF that] by (simp-all add: ‹a ∈∘ I›)
  from a show ?thesis
    by (cs-concl cs-shallow cs-simp: cat-discrete-cs-simps fba)
qed
show f ∘_A:C I :C I (|CId|) (|b|) = f if f : b ↦_:C I c for f b c
proof–
  from that have fba: f = b c = b and b: b ∈∘ I
    unfolding the-cat-discrete-is-arrD[OF that] by (simp-all add: ‹b ∈∘ I›)
  from b show ?thesis
    by (cs-concl cs-shallow cs-simp: cat-discrete-cs-simps fba)
qed
show :C I (|CId|) (|a|) : a ↦_:C I a
  if a ∈∘ :C I (|Obj|) for a
  using that
  by (auto simp: the-cat-discrete-components intro: cat-discrete-cs-intros)
show ⋃∘((λa∈∘A. ⋃∘(VLambda B (Hom (:C I) a) `∘ B)) `∘ A) ∈∘ Vset α
  if A ⊆∘ :C I (|Obj|)
    and B ⊆∘ :C I (|Obj|)
    and A ∈∘ Vset α
    and B ∈∘ Vset α
```

for A B
proof–
have $(\bigcup_{\circ} a \in_{\circ} A.\ \bigcup_{\circ} b \in_{\circ} B.\ \mathit{Hom}\ (:_C\ I)\ a\ b) \subseteq_{\circ} A \cup_{\circ} B$
proof(*intro vsubsetI*, *elim vifunionE*, *unfold in-Hom-iff*)
fix i j f **assume** *prems*: $i \in_{\circ} A$ $j \in_{\circ} B$ $f : i \mapsto_{:_C\ I} j$
then show $f \in_{\circ} A \cup_{\circ} B$
unfolding *the-cat-discrete-is-arrD*[*OF prems*(*3*)] **by** *simp*
qed
moreover have $A \cup_{\circ} B \in_{\circ} \mathit{Vset}\ \alpha$ **by** (*simp add*: *that*(*3*,*4*) *vunion-in-VsetI*)
ultimately show $(\bigcup_{\circ} a \in_{\circ} A.\ \bigcup_{\circ} b \in_{\circ} B.\ \mathit{Hom}\ (:_C\ I)\ a\ b) \in_{\circ} \mathit{Vset}\ \alpha$
by (*auto simp*: *vsubset-in-VsetI*)
qed
qed (*auto simp*: *assms the-cat-discrete-components intro*: *cat-cs-intros*)

lemmas [*cat-discrete-cs-intros*] = $\mathcal{Z}$.*cat-discrete-the-cat-discrete*

11.2.8 Opposite discrete category

lemma (**in** $\mathcal{Z}$) *the-cat-discrete-op*[*cat-op-simps*]:
assumes $I \subseteq_{\circ} \mathit{Vset}\ \alpha$
shows *op-cat* $(:_C\ I) = :_C\ I$
proof(*rule cat-eqI*[*of* α])
from *assms* **show** *dI*: *category* α $(:_C\ I)$
by (*cs-concl* **cs-intro**: *cat-discrete-the-cat-discrete cat-discrete-cs-intros*)
then show *op-dI*: *category* α (*op-cat* $(:_C\ I)$)
by (*cs-concl* **cs-shallow cs-intro**: *cat-op-intros*)
interpret *category* α ‹*op-cat* $(:_C\ I)$› **by** (*rule op-dI*)
show *op-cat* $(:_C\ I)(\!|\mathit{Comp}|\!) = :_C\ I(\!|\mathit{Comp}|\!)$
proof(*rule vsv-eqI*)
show $\mathcal{D}_{\circ}$ (*op-cat* $(:_C\ I)(\!|\mathit{Comp}|\!)$) = $\mathcal{D}_{\circ}$ $(:_C\ I(\!|\mathit{Comp}|\!))$
by (*simp add*: *the-cat-discrete-components op-cat-components*)
fix *gf* **assume** *gf* $\in_{\circ}$ $\mathcal{D}_{\circ}$ (*op-cat* $(:_C\ I)(\!|\mathit{Comp}|\!)$)
then have *gf* $\in_{\circ}$ *fid-on I*
by (*simp add*: *the-cat-discrete-components op-cat-components*)
then obtain *h* **where** *gf-def*: *gf* = $[h, h]_{\circ}$ **and** *h*: $h \in_{\circ} I$ **by** *clarsimp*
from *dI h* **show** *op-cat* $(:_C\ I)(\!|\mathit{Comp}|\!)(\!|\mathit{gf}|\!) = :_C\ I(\!|\mathit{Comp}|\!)(\!|\mathit{gf}|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps gf-def* **cs-intro**: *cat-discrete-cs-intros*
)
qed (*auto intro*: *cat-discrete-cs-intros*)
qed (*unfold the-cat-discrete-components op-cat-components*, *simp-all*)

11.3 Discrete functor

11.3.1 Local assumptions for the discrete functor

See Chapter III in [7]).

locale *cf-discrete* = *category* α $\mathfrak{C}$ **for** α I F $\mathfrak{C}$ +
assumes *cf-discrete-selector-vrange*[*cat-discrete-cs-intros*]:
$i \in_{\circ} I \Longrightarrow F\ i \in_{\circ} \mathfrak{C}(\!|\mathit{Obj}|\!)$
and *cf-discrete-vdomain-vsubset-Vset*: $I \subseteq_{\circ} \mathit{Vset}\ \alpha$

lemmas (**in** *cf-discrete*) *cf-discrete-category* = *category-axioms*

lemmas [*cat-discrete-cs-intros*] = *cf-discrete.cf-discrete-category*

Rules.

lemma (in *cf-discrete*) *cf-discrete-axioms'*[*cat-discrete-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $I' = I$ **and** $F' = F$
shows *cf-discrete* α' I' F' $\mathfrak{C}$
unfolding *assms* **by** (*rule cf-discrete-axioms*)

mk-ide rf *cf-discrete-def*[*unfolded cf-discrete-axioms-def*]
|*intro cf-discreteI*|
|*dest cf-discreteD*[*dest*]|
|*elim cf-discreteE*[*elim*]|

Elementary properties.

lemma (in *cf-discrete*) *cf-discrete-is-functor-cf-CId-selector-is-arr*:
assumes $i \in_\circ I$
shows $\mathfrak{C}(\!|CId|\!)(\!|F\ i|\!) : F\ i \mapsto_{\mathfrak{C}} F\ i$
using *assms* **by** (*meson cat-CId-is-arr' cf-discreteD*(*2*) *cf-discrete-axioms*)

lemma (in *cf-discrete*)
cf-discrete-is-functor-cf-CId-selector-is-arr'[*cat-discrete-cs-intros*]:
assumes $i \in_\circ I$ **and** $a = F\ i$ **and** $b = F\ i$
shows $\mathfrak{C}(\!|CId|\!)(\!|F\ i|\!) : a \mapsto_{\mathfrak{C}} b$
using *assms*(*1*)
unfolding *assms*(*2*,*3*)
by (*rule cf-discrete-is-functor-cf-CId-selector-is-arr*)

11.3.2 Definition and elementary properties

definition *the-cf-discrete* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow V$ ($\langle\!:\!\rightarrow\!:\!\rangle$)
where $:\!\rightarrow: I\ F\ \mathfrak{C} = [\ VLambda\ I\ F,\ (\lambda i \in_\circ I.\ \mathfrak{C}(\!|CId|\!)(\!|F\ i|\!)),\ :_C\ I,\ \mathfrak{C}]_\circ$

Components.

lemma *the-cf-discrete-components*:
shows $:\!\rightarrow: I\ F\ \mathfrak{C}(\!|ObjMap|\!) = (\lambda i \in_\circ I.\ F\ i)$
and $:\!\rightarrow: I\ F\ \mathfrak{C}(\!|ArrMap|\!) = (\lambda i \in_\circ I.\ \mathfrak{C}(\!|CId|\!)(\!|F\ i|\!))$
and [*cat-discrete-cs-simps*]: $:\!\rightarrow: I\ F\ \mathfrak{C}(\!|HomDom|\!) = :_C\ I$
and [*cat-discrete-cs-simps*]: $:\!\rightarrow: I\ F\ \mathfrak{C}(\!|HomCod|\!) = \mathfrak{C}$
unfolding *the-cf-discrete-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

11.3.3 Object map

mk-VLambda *the-cf-discrete-components*(*1*)
|*vsv the-cf-discrete-ObjMap-vsv*[*cat-discrete-cs-intros*]|
|*vdomain the-cf-discrete-ObjMap-vdomain*[*cat-discrete-cs-simps*]|
|*app the-cf-discrete-ObjMap-app*[*cat-discrete-cs-simps*]|

lemma (in *cf-discrete*) *cf-discrete-the-cf-discrete-ObjMap-vrange*:
$\mathcal{R}_\circ\ (:\!\rightarrow: I\ F\ \mathfrak{C}(\!|ObjMap|\!)) \subseteq_\circ \mathfrak{C}(\!|Obj|\!)$
using *cf-discrete-is-functor-cf-CId-selector-is-arr*
unfolding *the-cf-discrete-components*
by (*intro vrange-VLambda-vsubset*) *auto*

11.3.4 Arrow map

mk-VLambda *the-cf-discrete-components*(*2*)
|*vsv the-cf-discrete-ArrMap-vsv*[*cat-discrete-cs-intros*]|
|*vdomain the-cf-discrete-ArrMap-vdomain*[*cat-discrete-cs-simps*]|
|*app the-cf-discrete-ArrMap-app*[*cat-discrete-cs-simps*]|

lemma (**in** *cf-discrete*) *cf-discrete-the-cf-discrete-ArrMap-vrange*:
 $\mathcal{R}_{\circ}$ (:→: *I F* $\mathfrak{C}$(|*ArrMap*|)) $\subseteq_{\circ}$ $\mathfrak{C}$(|*Arr*|)
 using *cf-discrete-is-functor-cf-CId-selector-is-arr*
 unfolding *the-cf-discrete-components*
 by (*intro vrange-VLambda-vsubset*) (*auto simp*: *cf-discrete-selector-vrange*)

11.3.5 Discrete functor is a functor

lemma (**in** *cf-discrete*) *cf-discrete-the-cf-discrete-is-functor*:
 :→: *I F* $\mathfrak{C}$: :$_C$ *I* ↦↦$_{C\alpha}$ $\mathfrak{C}$
proof(*intro is-functorI′*)
 show *vfsequence* (:→: *I F* $\mathfrak{C}$) **unfolding** *the-cf-discrete-def* **by** *simp*
 show *category* α (:$_C$ *I*)
  by
   (
    simp add:
     cat-discrete-the-cat-discrete
     cf-discrete-vdomain-vsubset-Vset
     cat-discrete.axioms(*1*)
)
 show *vcard* (:→: *I F* $\mathfrak{C}$) = $\mathcal{4}_{\mathbb{N}}$
  unfolding *the-cf-discrete-def* **by** (*simp add*: *nat-omega-simps*)
 show
  :→: *I F* $\mathfrak{C}$(|*ArrMap*|)(|*f*|) : :→: *I F* $\mathfrak{C}$(|*ObjMap*|)(|*a*|) ↦$_{\mathfrak{C}}$:→: *I F* $\mathfrak{C}$(|*ObjMap*|)(|*b*|)
  if *f* : *a* ↦$_{:C}$ *I* *b* **for** *f a b*
 proof–
  from *that* **have** *fba*: *f* = *a b* = *a* **and** *a*: *a* ∈$_{\circ}$ *I*
   unfolding *the-cat-discrete-is-arrD*[*OF that*] **by** (*simp-all add*: ‹*a* ∈$_{\circ}$ *I*›)
  from *that* ‹*a* ∈$_{\circ}$ *I*› **show** *?thesis*
   by
    (
     cs-concl **cs-shallow**
      cs-simp: *cat-discrete-cs-simps fba* **cs-intro**: *cat-discrete-cs-intros*
)
 qed
 show :→: *I F* $\mathfrak{C}$(|*ArrMap*|)(|*g* ∘$_{A:C}$ *I f*|) =
  :→: *I F* $\mathfrak{C}$(|*ArrMap*|)(|*g*|) ∘$_{A\mathfrak{C}}$:→: *I F* $\mathfrak{C}$(|*ArrMap*|)(|*f*|)
  if *g* : *b* ↦$_{:C}$ *I* *c* **and** *f* : *a* ↦$_{:C}$ *I* *b* **for** *g b c f a*
 proof–
  from *that* **have** *gfacb*: *f* = *a a* = *b g* = *b c* = *b* **and** *b*: *b* ∈$_{\circ}$ *I*
   by
    (
     simp-all add:
      the-cat-discrete-is-arrD(*8–9*)[*OF that*(*1*)]
      the-cat-discrete-is-arrD(*5–9*)[*OF that*(*2*)]
)
  have *F b* ∈$_{\circ}$ $\mathfrak{C}$(|*Obj*|) **by** (*simp add*: *b cf-discrete-selector-vrange*)
  from *b category-axioms this* **show** *?thesis*
   using *that*
   unfolding *gfacb*
   by
    (
     cs-concl **cs-shallow**
      cs-simp: *cat-cs-simps cat-discrete-cs-simps* **cs-intro**: *cat-cs-intros*
)
 qed
 show :→: *I F* $\mathfrak{C}$(|*ArrMap*|)(|:$_C$ *I*(|*CId*|)(|*c*|)|) = $\mathfrak{C}$(|*CId*|)(|:→: *I F* $\mathfrak{C}$(|*ObjMap*|)(|*c*|)|)

```
    if c ∈∘ :C I(|Obj|) for c
    using that
    unfolding the-cat-discrete-components(1)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-discrete-cs-simps cs-intro: cat-cs-intros
      )
qed
  (
    auto simp:
      the-cf-discrete-components
      the-cat-discrete-components
      cat-cs-intros
      cat-discrete-cs-intros
  )

lemma (in cf-discrete) cf-discrete-the-cf-discrete-is-functor':
  assumes 𝔄' = :C I and 𝔈' = 𝔈
  shows :→: I F 𝔈 : 𝔄' ↦↦CO 𝔈'
  unfolding assms by (rule cf-discrete-the-cf-discrete-is-functor)

lemmas [cat-discrete-cs-intros] =
  cf-discrete.cf-discrete-the-cf-discrete-is-functor'
```

11.3.6 Uniqueness of the discrete category

```
lemma (in cat-discrete) cat-discrete-iso-the-cat-discrete:
  assumes I ⊆∘ Vset α and I ≈∘ 𝔈(|Obj|)
  obtains F where :→: I F 𝔈 : :C I ↦↦C.isoO 𝔈
proof–

  from assms obtain F where v11-f: v11 F
    and dr[simp]: 𝒟∘ F = I ℛ∘ F = 𝔈(|Obj|)
    by auto
  let ?F = λi. F(|i|)
  interpret F: v11 F by (rule v11-f)
  from assms(1) interpret 𝔈: cf-discrete α I ?F 𝔈
    apply(intro cf-discreteI)
    unfolding dr[symmetric]
    by (cs-concl cs-shallow cs-intro: V-cs-intros cat-cs-intros)+
  have :→: I ?F 𝔈 : :C I ↦↦C.isoO 𝔈
  proof(intro is-iso-functorI')
    from 𝔈.cf-discrete-selector-vrange show
      :→: I ?F 𝔈 : :C I ↦↦CO 𝔈
      by (intro cf-discrete.cf-discrete-the-cf-discrete-is-functor cf-discreteI)
        (auto simp: category-axioms assms(1))
    show v11 (:→: I ?F 𝔈(|ArrMap|))
    proof(rule vsv.vsv-valeq-v11I, unfold the-cf-discrete-ArrMap-vdomain)
      fix i j assume prems:
        i ∈∘ I j ∈∘ I :→: I ?F 𝔈(|ArrMap|)(|i|) = :→: I ?F 𝔈(|ArrMap|)(|j|)
      from prems(3) have 𝔈(|CId|)(|?F i|) = 𝔈(|CId|)(|?F j|)
        unfolding
          the-cf-discrete-ArrMap-app[OF prems(1)]
          the-cf-discrete-ArrMap-app[OF prems(2)].
      then have ?F i = ?F j
        by
          (
```

```
        metis
          𝔈.cf-discrete-is-functor-cf-CId-selector-is-arr
          prems(1,2)
          cat-is-arrD(4)
      )
    with F.v11-eq-iff prems show i = j by simp
  qed (simp add: the-cf-discrete-components)
  show ℛ∘ (:→: I ?F 𝔈(|ArrMap|)) = 𝔈(|Arr|)
  proof(intro vsubset-antisym vsubsetI)
    fix f assume f ∈∘ ℛ∘ (:→: I ?F 𝔈(|ArrMap|))
    with 𝔈.cf-discrete-the-cf-discrete-ArrMap-vrange show f ∈∘ 𝔈(|Arr|)
      by auto
  next
    fix f assume f ∈∘ 𝔈(|Arr|)
    then obtain a b where f : a ↦𝔈 b by auto
    then obtain a where f-def: f = 𝔈(|CId|)(|a|) and a: a ∈∘ 𝔈(|Obj|) by auto
    from a F.vrange-atD dr obtain i where a-def: a = ?F i and i: i ∈∘ I
      by blast
    from a i show f ∈∘ ℛ∘ (:→: I ?F 𝔈(|ArrMap|))
      unfolding a-def f-def the-cf-discrete-components by auto
  qed
qed (auto simp: v11-f the-cf-discrete-components)
with that show ?thesis by simp

qed
```

11.3.7 Opposite discrete functor

```
lemma (in cf-discrete) cf-discrete-the-cf-discrete-op[cat-op-simps]:
  op-cf (:→: I F 𝔈) = :→: I F (op-cat 𝔈)
proof(rule cf-eqI)
  from cf-discrete-vdomain-vsubset-Vset show
    op-cf (:→: I F 𝔈) : :C I ↦↦→Cα op-cat 𝔈
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps cs-intro: cat-op-intros cat-discrete-cs-intros
      )
  show :→: I F (op-cat 𝔈) : :C I ↦↦→Cα op-cat 𝔈
  proof(intro cf-discrete.cf-discrete-the-cf-discrete-is-functor cf-discreteI)
    fix i assume i ∈∘ I
    then show F i ∈∘ op-cat 𝔈(|Obj|)
      by (simp add: cat-op-simps cf-discrete-selector-vrange)
  qed (intro cf-discrete-vdomain-vsubset-Vset cat-cs-intros)+
qed (unfold cat-op-simps the-cf-discrete-components, simp-all)

lemmas [cat-op-simps] = cf-discrete.cf-discrete-the-cf-discrete-op

lemma (in cf-discrete) cf-discrete-op[cat-op-intros]:
  cf-discrete α I F (op-cat 𝔈)
proof(intro cf-discreteI)
  show category α (op-cat 𝔈)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  fix i assume i ∈∘ I
  then show F i ∈∘ op-cat 𝔈(|Obj|)
    by
      (
        cs-concl cs-shallow
```

cs-simp: *cat-op-simps* **cs-intro**: *cat-discrete-cs-intros*
)
qed (*intro cf-discrete-vdomain-vsubset-Vset*)

lemmas [*cat-op-intros*] = *cf-discrete.cf-discrete-op*

11.4 Tiny discrete category

11.4.1 Background

named-theorems *cat-small-discrete-cs-simps*
named-theorems *cat-small-discrete-cs-intros*

lemmas [*cat-small-discrete-cs-simps*] = *cat-discrete-cs-simps*
lemmas [*cat-small-discrete-cs-intros*] = *cat-discrete-cs-intros*

11.4.2 Definition and elementary properties

locale *tiny-cat-discrete* = *cat-discrete* α $\mathfrak{C}$ + *tiny-category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$

Rules.

lemma (**in** *tiny-cat-discrete*) *tiny-cat-discrete-axioms'*[*cat-discrete-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows *tiny-cat-discrete* α' $\mathfrak{C}'$
unfolding *assms* **by** (*rule tiny-cat-discrete-axioms*)

mk-ide rf *tiny-cat-discrete-def*
|*intro tiny-cat-discreteI*|
|*dest tiny-cat-discreteD*[*dest*]|
|*elim tiny-cat-discreteE*[*elim*]|

lemmas [*cat-small-discrete-cs-intros*] = *tiny-cat-discreteD*

lemma *tiny-cat-discreteI'*:
assumes *tiny-category* α $\mathfrak{C}$ **and** $\bigwedge f.\ f \in_\circ \mathfrak{C}(\!|Arr|\!) \Longrightarrow f \in_\circ \mathcal{R}_\circ\ (\mathfrak{C}(\!|CId|\!))$
shows *tiny-cat-discrete* α $\mathfrak{C}$
proof(*intro tiny-cat-discreteI cat-discreteI*)
interpret *tiny-category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))
show *category* α $\mathfrak{C}$ **by** (*auto intro*: *tiny-cat-category*)
show $f \in_\circ \mathcal{R}_\circ\ (\mathfrak{C}(\!|CId|\!))$ **if** $f \in_\circ \mathfrak{C}(\!|Arr|\!)$ **for** f **using** *that* **by** (*rule assms*(*2*))
qed (*auto intro*: *assms*(*1*))

11.4.3 The discrete category is a tiny category

lemma (**in** $\mathcal{Z}$) *tiny-cat-discrete-the-cat-discrete*[*cat-small-discrete-cs-intros*]:
assumes $I \in_\circ Vset\ \alpha$
shows *tiny-cat-discrete* α ($:_C$ I)
proof(*intro tiny-cat-discreteI cat-discrete-the-cat-discrete*)
from *assms* **show** $I \subseteq_\circ Vset\ \alpha$ **by** *auto*
then interpret *cat-discrete* α ‹$:_C$ I› **by** (*intro cat-discrete-the-cat-discrete*)
show *tiny-category* α ($:_C$ I)
by (*intro tiny-categoryI'*, *unfold the-cat-discrete-components*)
(*auto intro*: *cat-cs-intros assms*)
qed

lemmas [*cat-small-discrete-cs-intros*] = $\mathcal{Z}$.*cat-discrete-the-cat-discrete*

11.5 Discrete functor with tiny maps

11.5.1 Definition and elementary properties

locale *tm-cf-discrete* = *category* α $\mathfrak{C}$ **for** α I F $\mathfrak{C}$ +
  assumes *tm-cf-discrete-selector-vrange*[*cat-small-discrete-cs-intros*]:
    $i \in_\circ I \Longrightarrow F\ i \in_\circ \mathfrak{C}(\!|Obj|\!)$
    and *tm-cf-discrete-ObjMap-in-Vset*: *VLambda I F* $\in_\circ$ *Vset* α
    and *tm-cf-discrete-ArrMap-in-Vset*: $(\lambda i \in_\circ I.\ \mathfrak{C}(\!|CId|\!)(\!|F\ i|\!)) \in_\circ$ *Vset* α

Rules.

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-axioms'*[*cat-small-discrete-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $I' = I$ **and** $F' = F$
  shows *tm-cf-discrete* α' I' F' $\mathfrak{C}$
  unfolding *assms* **by** (*rule tm-cf-discrete-axioms*)

mk-ide rf *tm-cf-discrete-def*[*unfolded tm-cf-discrete-axioms-def*]
  |*intro tm-cf-discreteI*|
  |*dest tm-cf-discreteD*[*dest*]|
  |*elim tm-cf-discreteE*[*elim*]|

lemma *tm-cf-discreteI'*:
  assumes *cf-discrete* α I F $\mathfrak{C}$
    and $(\lambda i \in_\circ I.\ F\ i) \in_\circ$ *Vset* α
    and $(\lambda i \in_\circ I.\ \mathfrak{C}(\!|CId|\!)(\!|F\ i|\!)) \in_\circ$ *Vset* α
  shows *tm-cf-discrete* α I F $\mathfrak{C}$
proof–
  interpret *cf-discrete* α I F $\mathfrak{C}$ **by** (*rule assms*(*1*))
  show *?thesis*
    by (*intro tm-cf-discreteI*)
      (*auto intro*: *assms cf-discrete-selector-vrange cat-cs-intros*)
qed

Elementary properties.

sublocale *tm-cf-discrete* ⊆ *cf-discrete*
proof(*intro cf-discreteI*)
  from *tm-cf-discrete-ObjMap-in-Vset* **have** $\mathcal{D}_\circ\ (\lambda i \in_\circ I.\ F\ i) \in_\circ$ *Vset* α
    by (*cs-concl* **cs-shallow cs-intro**: *vdomain-in-VsetI*)
  then show $I \subseteq_\circ$ *Vset* α **by** *auto*
qed (*auto intro*: *cat-cs-intros tm-cf-discrete-selector-vrange*)

lemmas (**in** *tm-cf-discrete*) *tm-cf-discrete-is-cf-discrete-axioms* =
  cf-discrete-axioms

lemmas [*cat-small-discrete-cs-intros*] =
  tm-cf-discrete.tm-cf-discrete-is-cf-discrete-axioms

lemma (**in** *tm-cf-discrete*)
  tm-cf-discrete-index-in-Vset[*cat-small-discrete-cs-intros*]:
  $I \in_\circ$ *Vset* α
proof–
  from *tm-cf-discrete-ObjMap-in-Vset* **have** $\mathcal{D}_\circ\ (\lambda i \in_\circ I.\ F\ i) \in_\circ$ *Vset* α
    by (*cs-concl* **cs-shallow cs-intro**: *vdomain-in-VsetI*)
  then show *?thesis* **by** *simp*
qed

11.5.2 Opposite discrete functor with tiny maps

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-op*[*cat-op-intros*]:

tm-cf-discrete α I F (*op-cat* $\mathfrak{C}$)
using *tm-cf-discrete-ObjMap-in-Vset* *tm-cf-discrete-ArrMap-in-Vset*
by (*intro* *tm-cf-discreteI'* *cf-discrete-op*) (*auto simp*: *cat-op-simps*)

lemmas [*cat-op-intros*] = *tm-cf-discrete.tm-cf-discrete-op*

11.5.3 Discrete functor with tiny maps is a functor with tiny maps

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-the-cf-discrete-is-tm-functor*:
$:\!\!\rightarrow: I\ F\ \mathfrak{C} : :_C\ I \mapsto\!\!\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*intro* *is-tm-functorI'* *cf-discrete-the-cf-discrete-is-functor*)
(
auto simp:
the-cf-discrete-components
tm-cf-discrete-ObjMap-in-Vset
tm-cf-discrete-ArrMap-in-Vset
)

lemma (**in** *tm-cf-discrete*) *tm-cf-discrete-the-cf-discrete-is-tm-functor'*:
assumes $\mathfrak{A}' = :_C\ I$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $:\!\!\rightarrow: I\ F\ \mathfrak{C} : \mathfrak{A}' \mapsto\!\!\mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule* *tm-cf-discrete-the-cf-discrete-is-tm-functor*)

lemmas [*cat-discrete-cs-intros*] =
tm-cf-discrete.tm-cf-discrete-the-cf-discrete-is-tm-functor'

12 →•← and ←•→: cospan and span

12.1 Background

General information about →•← and ←•→ (also known as cospans and spans, respectively) can be found in in Chapters III-3 and III-4 in [7], as well as nLab [1][56].

named-theorems *cat-ss-cs-simps*
named-theorems *cat-ss-cs-intros*

named-theorems *cat-ss-elem-simps*

definition $\mathfrak{o}_{SS}$ **where** [*cat-ss-elem-simps*]: $\mathfrak{o}_{SS} = 0$
definition $\mathfrak{a}_{SS}$ **where** [*cat-ss-elem-simps*]: $\mathfrak{a}_{SS} = 1_{\mathbb{N}}$
definition $\mathfrak{b}_{SS}$ **where** [*cat-ss-elem-simps*]: $\mathfrak{b}_{SS} = 2_{\mathbb{N}}$
definition $\mathfrak{g}_{SS}$ **where** [*cat-ss-elem-simps*]: $\mathfrak{g}_{SS} = 3_{\mathbb{N}}$
definition $\mathfrak{f}_{SS}$ **where** [*cat-ss-elem-simps*]: $\mathfrak{f}_{SS} = 4_{\mathbb{N}}$

lemma *cat-ss-ineq*:
shows *cat-ss-*$\mathfrak{ab}$[*cat-ss-cs-intros*]: $\mathfrak{a}_{SS} \neq \mathfrak{b}_{SS}$
and *cat-ss-*$\mathfrak{ao}$[*cat-ss-cs-intros*]: $\mathfrak{a}_{SS} \neq \mathfrak{o}_{SS}$
and *cat-ss-*$\mathfrak{bo}$[*cat-ss-cs-intros*]: $\mathfrak{b}_{SS} \neq \mathfrak{o}_{SS}$
and *cat-ss-*$\mathfrak{gf}$[*cat-ss-cs-intros*]: $\mathfrak{g}_{SS} \neq \mathfrak{f}_{SS}$
and *cat-ss-*$\mathfrak{ga}$[*cat-ss-cs-intros*]: $\mathfrak{g}_{SS} \neq \mathfrak{a}_{SS}$
and *cat-ss-*$\mathfrak{gb}$[*cat-ss-cs-intros*]: $\mathfrak{g}_{SS} \neq \mathfrak{b}_{SS}$
and *cat-ss-*$\mathfrak{go}$[*cat-ss-cs-intros*]: $\mathfrak{g}_{SS} \neq \mathfrak{o}_{SS}$
and *cat-ss-*$\mathfrak{fa}$[*cat-ss-cs-intros*]: $\mathfrak{f}_{SS} \neq \mathfrak{a}_{SS}$
and *cat-ss-*$\mathfrak{fb}$[*cat-ss-cs-intros*]: $\mathfrak{f}_{SS} \neq \mathfrak{b}_{SS}$
and *cat-ss-*$\mathfrak{fo}$[*cat-ss-cs-intros*]: $\mathfrak{f}_{SS} \neq \mathfrak{o}_{SS}$
unfolding *cat-ss-elem-simps* **by** *simp-all*

lemma (in $\mathcal{Z}$**)**
shows *cat-ss-*$\mathfrak{a}$[*cat-ss-cs-intros*]: $\mathfrak{a}_{SS} \in_{\circ} \mathit{Vset}\ \alpha$
and *cat-ss-*$\mathfrak{b}$[*cat-ss-cs-intros*]: $\mathfrak{b}_{SS} \in_{\circ} \mathit{Vset}\ \alpha$
and *cat-ss-*$\mathfrak{o}$[*cat-ss-cs-intros*]: $\mathfrak{o}_{SS} \in_{\circ} \mathit{Vset}\ \alpha$
and *cat-ss-*$\mathfrak{g}$[*cat-ss-cs-intros*]: $\mathfrak{g}_{SS} \in_{\circ} \mathit{Vset}\ \alpha$
and *cat-ss-*$\mathfrak{f}$[*cat-ss-cs-intros*]: $\mathfrak{f}_{SS} \in_{\circ} \mathit{Vset}\ \alpha$
unfolding *cat-ss-elem-simps* **by** *simp-all*

12.2 Composable arrows in →•← and ←•→

abbreviation *cat-scospan-composable* :: *V*
where *cat-scospan-composable* ≡
$(\mathit{set}\ \{\mathfrak{o}_{SS}\} \times_{\bullet} \mathit{set}\ \{\mathfrak{o}_{SS}, \mathfrak{g}_{SS}, \mathfrak{f}_{SS}\}) \cup_{\circ}$
$(\mathit{set}\ \{\mathfrak{g}_{SS}, \mathfrak{a}_{SS}\} \times_{\bullet} \mathit{set}\ \{\mathfrak{a}_{SS}\}) \cup_{\circ}$
$(\mathit{set}\ \{\mathfrak{f}_{SS}, \mathfrak{b}_{SS}\} \times_{\bullet} \mathit{set}\ \{\mathfrak{b}_{SS}\})$

abbreviation *cat-sspan-composable* :: *V*
where *cat-sspan-composable* ≡ $(\mathit{cat\text{-}scospan\text{-}composable})^{-1_{\bullet}}$

Rules.

lemma *cat-scospan-composable-*$\mathfrak{oo}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $[g, f]_{\circ} \in_{\circ} \mathit{cat\text{-}scospan\text{-}composable}$
using *assms* **by** *auto*

[5] https://ncatlab.org/nlab/show/cospan
[6] https://ncatlab.org/nlab/show/span

lemma *cat-scospan-composable-*$\mathfrak{og}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{g}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
using *assms* **by** *auto*

lemma *cat-scospan-composable-*$\mathfrak{of}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
using *assms* **by** *auto*

lemma *cat-scospan-composable-*$\mathfrak{ga}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{g}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
using *assms* **by** *auto*

lemma *cat-scospan-composable-*$\mathfrak{fb}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{f}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
using *assms* **by** *auto*

lemma *cat-scospan-composable-*$\mathfrak{aa}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
using *assms* **by** *auto*

lemma *cat-scospan-composable-*$\mathfrak{bb}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
using *assms* **by** *auto*

lemma *cat-scospan-composableE*:
assumes $[g, f]_\circ \in_\circ$ *cat-scospan-composable*
obtains $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
| $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{g}_{SS}$
| $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
| $g = \mathfrak{g}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
| $g = \mathfrak{f}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
| $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
| $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
using *assms that* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{oo}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{go}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{g}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{fo}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{f}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{ag}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{g}_{SS}$

shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{bf}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{aa}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composable-*$\mathfrak{bb}$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
using *assms* **by** *auto*

lemma *cat-sspan-composableE*:
assumes $[g, f]_\circ \in_\circ$ *cat-sspan-composable*
obtains $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
| $g = \mathfrak{g}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
| $g = \mathfrak{f}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
| $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{g}_{SS}$
| $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
| $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
| $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
using *assms that* **by** *auto*

12.3 Categories $\rightarrow\!\cdot\!\leftarrow$ and $\leftarrow\!\cdot\!\rightarrow$

12.3.1 Definition and elementary properties

See Chapter III-3 and Chapter III-4 in [7].

definition *the-cat-scospan* :: V ($\langle\rightarrow\!\cdot\!\leftarrow_C\rangle$)
where $\rightarrow\!\cdot\!\leftarrow_C =$
[
set $\{\mathfrak{a}_{SS}, \mathfrak{b}_{SS}, \mathfrak{o}_{SS}\}$,
set $\{\mathfrak{a}_{SS}, \mathfrak{g}_{SS}, \mathfrak{o}_{SS}, \mathfrak{f}_{SS}, \mathfrak{b}_{SS}\}$,
(
$\lambda x \in_\circ$ *set* $\{\mathfrak{a}_{SS}, \mathfrak{g}_{SS}, \mathfrak{o}_{SS}, \mathfrak{f}_{SS}, \mathfrak{b}_{SS}\}$.
if $x = \mathfrak{a}_{SS} \Rightarrow \mathfrak{a}_{SS}$
| $x = \mathfrak{b}_{SS} \Rightarrow \mathfrak{b}_{SS}$
| $x = \mathfrak{g}_{SS} \Rightarrow \mathfrak{a}_{SS}$
| $x = \mathfrak{f}_{SS} \Rightarrow \mathfrak{b}_{SS}$
| *otherwise* $\Rightarrow \mathfrak{o}_{SS}$
),
(
$\lambda x \in_\circ$ *set* $\{\mathfrak{a}_{SS}, \mathfrak{g}_{SS}, \mathfrak{o}_{SS}, \mathfrak{f}_{SS}, \mathfrak{b}_{SS}\}$.
if $x = \mathfrak{a}_{SS} \Rightarrow \mathfrak{a}_{SS}$
| $x = \mathfrak{b}_{SS} \Rightarrow \mathfrak{b}_{SS}$
| *otherwise* $\Rightarrow \mathfrak{o}_{SS}$
),
(
$\lambda gf \in_\circ$ *cat-scospan-composable*.
if $gf = [\mathfrak{o}_{SS}, \mathfrak{g}_{SS}]_\circ \Rightarrow \mathfrak{g}_{SS}$
| $gf = [\mathfrak{o}_{SS}, \mathfrak{f}_{SS}]_\circ \Rightarrow \mathfrak{f}_{SS}$
| *otherwise* $\Rightarrow gf(\!|0|\!)$

```
    ),
    vid-on (set {𝔞_SS, 𝔟_SS, 𝔬_SS})
  ]∘

definition the-cat-sspan :: V (⟨←•→_C⟩)
  where ←•→_C =
  [
    set {𝔞_SS, 𝔟_SS, 𝔬_SS},
    set {𝔞_SS, 𝔤_SS, 𝔬_SS, 𝔣_SS, 𝔟_SS},
    (
      λx∈∘set {𝔞_SS, 𝔤_SS, 𝔬_SS, 𝔣_SS, 𝔟_SS}.
       if x = 𝔞_SS ⇒ 𝔞_SS
        | x = 𝔟_SS ⇒ 𝔟_SS
        | otherwise ⇒ 𝔬_SS
    ),
    (
      λx∈∘set {𝔞_SS, 𝔤_SS, 𝔬_SS, 𝔣_SS, 𝔟_SS}.
       if x = 𝔞_SS ⇒ 𝔞_SS
        | x = 𝔟_SS ⇒ 𝔟_SS
        | x = 𝔤_SS ⇒ 𝔞_SS
        | x = 𝔣_SS ⇒ 𝔟_SS
        | otherwise ⇒ 𝔬_SS
    ),
    (
      λgf∈∘cat-sspan-composable.
       if gf = [𝔞_SS, 𝔤_SS]∘ ⇒ 𝔤_SS
        | gf = [𝔟_SS, 𝔣_SS]∘ ⇒ 𝔣_SS
        | otherwise ⇒ gf(|0|)
    ),
    vid-on (set {𝔞_SS, 𝔟_SS, 𝔬_SS})
  ]∘
```

Components.

```
lemma the-cat-scospan-components:
  shows →•←_C(|Obj|) = set {𝔞_SS, 𝔟_SS, 𝔬_SS}
    and →•←_C(|Arr|) = set {𝔞_SS, 𝔤_SS, 𝔬_SS, 𝔣_SS, 𝔟_SS}
    and →•←_C(|Dom|) =
      (
        λx∈∘set {𝔞_SS, 𝔤_SS, 𝔬_SS, 𝔣_SS, 𝔟_SS}.
         if x = 𝔞_SS ⇒ 𝔞_SS
          | x = 𝔟_SS ⇒ 𝔟_SS
          | x = 𝔤_SS ⇒ 𝔞_SS
          | x = 𝔣_SS ⇒ 𝔟_SS
          | otherwise ⇒ 𝔬_SS
      )
    and →•←_C(|Cod|) =
      (
        λx∈∘set {𝔞_SS, 𝔤_SS, 𝔬_SS, 𝔣_SS, 𝔟_SS}.
         if x = 𝔞_SS ⇒ 𝔞_SS
          | x = 𝔟_SS ⇒ 𝔟_SS
          | otherwise ⇒ 𝔬_SS
      )
    and →•←_C(|Comp|) =
      (
        λgf∈∘cat-scospan-composable.
         if gf = [𝔬_SS, 𝔤_SS]∘ ⇒ 𝔤_SS
          | gf = [𝔬_SS, 𝔣_SS]∘ ⇒ 𝔣_SS
          | otherwise ⇒ gf(|0|)
```

)
and $\rightarrow\!\cdot\!\leftarrow_C(\!|CId|\!) = vid\text{-}on\ (set\ \{\mathfrak{a}_{SS},\ \mathfrak{b}_{SS},\ \mathfrak{o}_{SS}\})$
unfolding *the-cat-scospan-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *the-cat-sspan-components*:
shows $\leftarrow\!\cdot\!\rightarrow_C(\!|Obj|\!) = set\ \{\mathfrak{a}_{SS},\ \mathfrak{b}_{SS},\ \mathfrak{o}_{SS}\}$
and $\leftarrow\!\cdot\!\rightarrow_C(\!|Arr|\!) = set\ \{\mathfrak{a}_{SS},\ \mathfrak{g}_{SS},\ \mathfrak{o}_{SS},\ \mathfrak{f}_{SS},\ \mathfrak{b}_{SS}\}$
and $\leftarrow\!\cdot\!\rightarrow_C(\!|Dom|\!) =$
(
$\lambda x \in_\circ set\ \{\mathfrak{a}_{SS},\ \mathfrak{g}_{SS},\ \mathfrak{o}_{SS},\ \mathfrak{f}_{SS},\ \mathfrak{b}_{SS}\}.$
if $x = \mathfrak{a}_{SS} \Rightarrow \mathfrak{a}_{SS}$
$\mid x = \mathfrak{b}_{SS} \Rightarrow \mathfrak{b}_{SS}$
$\mid otherwise \Rightarrow \mathfrak{o}_{SS}$
)
and $\leftarrow\!\cdot\!\rightarrow_C(\!|Cod|\!) =$
(
$\lambda x \in_\circ set\ \{\mathfrak{a}_{SS},\ \mathfrak{g}_{SS},\ \mathfrak{o}_{SS},\ \mathfrak{f}_{SS},\ \mathfrak{b}_{SS}\}.$
if $x = \mathfrak{a}_{SS} \Rightarrow \mathfrak{a}_{SS}$
$\mid x = \mathfrak{b}_{SS} \Rightarrow \mathfrak{b}_{SS}$
$\mid x = \mathfrak{g}_{SS} \Rightarrow \mathfrak{a}_{SS}$
$\mid x = \mathfrak{f}_{SS} \Rightarrow \mathfrak{b}_{SS}$
$\mid otherwise \Rightarrow \mathfrak{o}_{SS}$
)
and $\leftarrow\!\cdot\!\rightarrow_C(\!|Comp|\!) =$
(
$\lambda gf \in_\circ cat\text{-}sspan\text{-}composable.$
if $gf = [\mathfrak{a}_{SS},\ \mathfrak{g}_{SS}]_\circ \Rightarrow \mathfrak{g}_{SS}$
$\mid gf = [\mathfrak{b}_{SS},\ \mathfrak{f}_{SS}]_\circ \Rightarrow \mathfrak{f}_{SS}$
$\mid otherwise \Rightarrow gf(\!|0|\!)$
)
and $\leftarrow\!\cdot\!\rightarrow_C(\!|CId|\!) = vid\text{-}on\ (set\ \{\mathfrak{a}_{SS},\ \mathfrak{b}_{SS},\ \mathfrak{o}_{SS}\})$
unfolding *the-cat-sspan-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Elementary properties.

lemma *the-cat-scospan-components-vsv*[*cat-ss-cs-intros*]: *vsv* $(\rightarrow\!\cdot\!\leftarrow_C)$
unfolding *the-cat-scospan-def* **by** *auto*

lemma *the-cat-sspan-components-vsv*[*cat-ss-cs-intros*]: *vsv* $(\leftarrow\!\cdot\!\rightarrow_C)$
unfolding *the-cat-sspan-def* **by** *auto*

12.3.2 Objects

lemma *the-cat-scospan-Obj-*$\mathfrak{o}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{o}_{SS}$
shows $a \in_\circ \rightarrow\!\cdot\!\leftarrow_C(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-Obj-*$\mathfrak{a}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{a}_{SS}$
shows $a \in_\circ \rightarrow\!\cdot\!\leftarrow_C(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-Obj-*$\mathfrak{b}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{b}_{SS}$
shows $a \in_\circ \rightarrow\!\cdot\!\leftarrow_C(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-ObjE*:

assumes $a \in_{\circ} \to\bullet\leftarrow_{C}(\!|Obj|\!)$
obtains $\langle a = \mathfrak{o}_{SS} \rangle \mid \langle a = \mathfrak{a}_{SS} \rangle \mid \langle a = \mathfrak{b}_{SS} \rangle$
using *assms* **unfolding** *the-cat-scospan-components* **by** *auto*

lemma *the-cat-sspan-Obj-*$\mathfrak{o}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{o}_{SS}$
shows $a \in_{\circ} \leftarrow\bullet\rightarrow_{C}(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-Obj-*$\mathfrak{a}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{a}_{SS}$
shows $a \in_{\circ} \leftarrow\bullet\rightarrow_{C}(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-Obj-*$\mathfrak{b}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{b}_{SS}$
shows $a \in_{\circ} \leftarrow\bullet\rightarrow_{C}(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-ObjE*:
assumes $a \in_{\circ} \leftarrow\bullet\rightarrow_{C}(\!|Obj|\!)$
obtains $\langle a = \mathfrak{o}_{SS} \rangle \mid \langle a = \mathfrak{a}_{SS} \rangle \mid \langle a = \mathfrak{b}_{SS} \rangle$
using *assms* **unfolding** *the-cat-sspan-components* **by** *auto*

12.3.3 Arrows

lemma *the-cat-scospan-Arr-*$\mathfrak{a}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{a}_{SS}$
shows $a \in_{\circ} \to\bullet\leftarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-Arr-*$\mathfrak{b}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{b}_{SS}$
shows $a \in_{\circ} \to\bullet\leftarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-Arr-*$\mathfrak{o}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{o}_{SS}$
shows $a \in_{\circ} \to\bullet\leftarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-Arr-*$\mathfrak{g}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{g}_{SS}$
shows $a \in_{\circ} \to\bullet\leftarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-Arr-*$\mathfrak{f}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{f}_{SS}$
shows $a \in_{\circ} \to\bullet\leftarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-scospan-components* **by** *simp*

lemma *the-cat-scospan-ArrE*:
assumes $f \in_{\circ} \to\bullet\leftarrow_{C}(\!|Arr|\!)$
obtains $\langle f = \mathfrak{a}_{SS} \rangle \mid \langle f = \mathfrak{b}_{SS} \rangle \mid \langle f = \mathfrak{o}_{SS} \rangle \mid \langle f = \mathfrak{g}_{SS} \rangle \mid \langle f = \mathfrak{f}_{SS} \rangle$
using *assms* **unfolding** *the-cat-scospan-components* **by** *auto*

lemma *the-cat-sspan-Arr-*$\mathfrak{a}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{a}_{SS}$

shows $a \in_{\circ} \leftarrow\!\cdot\!\rightarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-Arr-*$\mathfrak{b}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{b}_{SS}$
shows $a \in_{\circ} \leftarrow\!\cdot\!\rightarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-Arr-*$\mathfrak{o}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{o}_{SS}$
shows $a \in_{\circ} \leftarrow\!\cdot\!\rightarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-Arr-*$\mathfrak{g}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{g}_{SS}$
shows $a \in_{\circ} \leftarrow\!\cdot\!\rightarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-Arr-*$\mathfrak{f}I$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{f}_{SS}$
shows $a \in_{\circ} \leftarrow\!\cdot\!\rightarrow_{C}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-sspan-components* **by** *simp*

lemma *the-cat-sspan-ArrE*:
assumes $f \in_{\circ} \leftarrow\!\cdot\!\rightarrow_{C}(\!|Arr|\!)$
obtains $\langle f = \mathfrak{a}_{SS}\rangle \mid \langle f = \mathfrak{b}_{SS}\rangle \mid \langle f = \mathfrak{o}_{SS}\rangle \mid \langle f = \mathfrak{g}_{SS}\rangle \mid \langle f = \mathfrak{f}_{SS}\rangle$
using *assms* **unfolding** *the-cat-sspan-components* **by** *auto*

12.3.4 Domain

mk-VLambda *the-cat-scospan-components*(*3*)
|*vsv the-cat-scospan-Dom-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-scospan-Dom-vdomain*[*cat-ss-cs-simps*]|

lemma *the-cat-scospan-Dom-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{a}_{SS}$
shows $\rightarrow\!\cdot\!\leftarrow_{C}(\!|Dom|\!)(\!|f|\!) = \mathfrak{a}_{SS}$
unfolding *the-cat-scospan-components assms* **by** *simp*

lemma *the-cat-scospan-Dom-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{b}_{SS}$
shows $\rightarrow\!\cdot\!\leftarrow_{C}(\!|Dom|\!)(\!|f|\!) = \mathfrak{b}_{SS}$
unfolding *the-cat-scospan-components assms* **by** *simp*

lemma *the-cat-scospan-Dom-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{o}_{SS}$
shows $\rightarrow\!\cdot\!\leftarrow_{C}(\!|Dom|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-scospan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-scospan-Dom-app-*$\mathfrak{g}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{g}_{SS}$
shows $\rightarrow\!\cdot\!\leftarrow_{C}(\!|Dom|\!)(\!|f|\!) = \mathfrak{a}_{SS}$
unfolding *the-cat-scospan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-scospan-Dom-app-*$\mathfrak{f}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{f}_{SS}$
shows $\rightarrow\!\cdot\!\leftarrow_{C}(\!|Dom|\!)(\!|f|\!) = \mathfrak{b}_{SS}$
unfolding *the-cat-scospan-components assms* **using** *cat-ss-ineq* **by** *auto*

mk-VLambda *the-cat-sspan-components*(*3*)
|*vsv the-cat-sspan-Dom-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-sspan-Dom-vdomain*[*cat-ss-cs-simps*]|

lemma *the-cat-sspan-Dom-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{a}_{SS}$
shows $\leftarrow\!\!\cdot\!\!\rightarrow_C(\!|Dom|\!)(\!|f|\!) = \mathfrak{a}_{SS}$
unfolding *the-cat-sspan-components assms* **by** *simp*

lemma *the-cat-sspan-Dom-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{b}_{SS}$
shows $\leftarrow\!\!\cdot\!\!\rightarrow_C(\!|Dom|\!)(\!|f|\!) = \mathfrak{b}_{SS}$
unfolding *the-cat-sspan-components assms* **by** *simp*

lemma *the-cat-sspan-Dom-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{o}_{SS}$
shows $\leftarrow\!\!\cdot\!\!\rightarrow_C(\!|Dom|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-sspan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-sspan-Dom-app-*$\mathfrak{g}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{g}_{SS}$
shows $\leftarrow\!\!\cdot\!\!\rightarrow_C(\!|Dom|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-sspan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-sspan-Dom-app-*$\mathfrak{f}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{f}_{SS}$
shows $\leftarrow\!\!\cdot\!\!\rightarrow_C(\!|Dom|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-sspan-components assms* **using** *cat-ss-ineq* **by** *auto*

12.3.5 Codomain

mk-VLambda *the-cat-scospan-components*(*4*)
|*vsv the-cat-scospan-Cod-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-scospan-Cod-vdomain*[*cat-ss-cs-simps*]|

lemma *the-cat-scospan-Cod-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{a}_{SS}$
shows $\rightarrow\!\!\cdot\!\!\leftarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{a}_{SS}$
unfolding *the-cat-scospan-components assms* **by** *simp*

lemma *the-cat-scospan-Cod-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{b}_{SS}$
shows $\rightarrow\!\!\cdot\!\!\leftarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{b}_{SS}$
unfolding *the-cat-scospan-components assms* **by** *simp*

lemma *the-cat-scospan-Cod-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{o}_{SS}$
shows $\rightarrow\!\!\cdot\!\!\leftarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-scospan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-scospan-Cod-app-*$\mathfrak{g}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{g}_{SS}$
shows $\rightarrow\!\!\cdot\!\!\leftarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-scospan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-scospan-Cod-app-*$\mathfrak{f}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{f}_{SS}$

shows $\rightarrow\cdot\leftarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-scospan-components assms* **using** *cat-ss-ineq* **by** *auto*

mk-VLambda *the-cat-sspan-components*(4)
|*vsv the-cat-sspan-Cod-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-sspan-Cod-vdomain*[*cat-ss-cs-simps*]|

lemma *the-cat-sspan-Cod-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{a}_{SS}$
shows $\leftarrow\cdot\rightarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{a}_{SS}$
unfolding *the-cat-sspan-components assms* **by** *simp*

lemma *the-cat-sspan-Cod-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{b}_{SS}$
shows $\leftarrow\cdot\rightarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{b}_{SS}$
unfolding *the-cat-sspan-components assms* **by** *simp*

lemma *the-cat-sspan-Cod-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{o}_{SS}$
shows $\leftarrow\cdot\rightarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{o}_{SS}$
unfolding *the-cat-sspan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-sspan-Cod-app-*$\mathfrak{g}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{g}_{SS}$
shows $\leftarrow\cdot\rightarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{a}_{SS}$
unfolding *the-cat-sspan-components assms* **using** *cat-ss-ineq* **by** *auto*

lemma *the-cat-sspan-Cod-app-*$\mathfrak{f}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{f}_{SS}$
shows $\leftarrow\cdot\rightarrow_C(\!|Cod|\!)(\!|f|\!) = \mathfrak{b}_{SS}$
unfolding *the-cat-sspan-components assms* **using** *cat-ss-ineq* **by** *auto*

12.3.6 Composition

mk-VLambda *the-cat-scospan-components*(5)
|*vsv the-cat-scospan-Comp-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-scospan-Comp-vdomain*[*cat-ss-cs-simps*]|

lemma *the-cat-scospan-Comp-app-*$\mathfrak{a}\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $g \circ_{A\rightarrow\cdot\leftarrow_C} f = g$ $g \circ_{A\rightarrow\cdot\leftarrow_C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
with *assms* **show** $g \circ_{A\rightarrow\cdot\leftarrow_C} f = g$ $g \circ_{A\rightarrow\cdot\leftarrow_C} f = f$
unfolding *the-cat-scospan-components*(5) **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-scospan-Comp-app-*$\mathfrak{b}\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $g \circ_{A\rightarrow\cdot\leftarrow_C} f = g$ $g \circ_{A\rightarrow\cdot\leftarrow_C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
with *assms* **show** $g \circ_{A\rightarrow\cdot\leftarrow_C} f = g$ $g \circ_{A\rightarrow\cdot\leftarrow_C} f = f$
unfolding *the-cat-scospan-components*(5) **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-scospan-Comp-app-*$\mathfrak{o}\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$

shows $g \circ_{A\to\bullet\leftarrow C} f = g$ $g \circ_{A\to\bullet\leftarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
with *assms* **show** $g \circ_{A\to\bullet\leftarrow C} f = g$ $g \circ_{A\to\bullet\leftarrow C} f = f$
unfolding *the-cat-scospan-components*(*5*) **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-scospan-Comp-app-*$\mathfrak{og}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{g}_{SS}$
shows $g \circ_{A\to\bullet\leftarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
then show $g \circ_{A\to\bullet\leftarrow C} f = f$
unfolding *the-cat-scospan-components*(*5*) *assms* **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-scospan-Comp-app-*$\mathfrak{of}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
shows $g \circ_{A\to\bullet\leftarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
then show $g \circ_{A\to\bullet\leftarrow C} f = f$
unfolding *the-cat-scospan-components*(*5*) *assms* **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-scospan-Comp-app-*$\mathfrak{ga}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{g}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $g \circ_{A\to\bullet\leftarrow C} f = g$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
then show $g \circ_{A\to\bullet\leftarrow C} f = g$
unfolding *the-cat-scospan-components*(*5*) *assms*
using *cat-ss-ineq*
by (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-scospan-Comp-app-*$\mathfrak{fb}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{f}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $g \circ_{A\to\bullet\leftarrow C} f = g$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-scospan-composable* **by** *auto*
then show $g \circ_{A\to\bullet\leftarrow C} f = g$
unfolding *the-cat-scospan-components*(*5*) *assms*
using *cat-ss-ineq*
by (*auto simp*: *nat-omega-simps*)
qed

mk-VLambda *the-cat-sspan-components*(*5*)
|*vsv the-cat-sspan-Comp-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-sspan-Comp-vdomain*[*cat-ss-cs-simps*]|

lemma *the-cat-sspan-Comp-app-*$\mathfrak{aa}$[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $g \circ_{A\leftarrow\bullet\to C} f = g$ $g \circ_{A\leftarrow\bullet\to C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
with *assms* **show** $g \circ_{A\leftarrow\bullet\to C} f = g$ $g \circ_{A\leftarrow\bullet\to C} f = f$
unfolding *the-cat-sspan-components*(*5*) **by** (*auto simp*: *nat-omega-simps*)

qed

lemma *the-cat-sspan-Comp-app-𝔟𝔟*[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$ $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
with *assms* **show** $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$ $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
unfolding *the-cat-sspan-components*(*5*) **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-sspan-Comp-app-𝔬𝔬*[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$ $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
with *assms* **show** $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$ $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
unfolding *the-cat-sspan-components*(*5*) **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-sspan-Comp-app-𝔞𝔤*[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{g}_{SS}$
shows $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
then show $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
unfolding *the-cat-sspan-components*(*5*) *assms* **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-sspan-Comp-app-𝔟𝔣*[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
shows $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
then show $g \circ_{A \leftarrow \bullet \rightarrow C} f = f$
unfolding *the-cat-sspan-components*(*5*) *assms* **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-sspan-Comp-app-𝔤𝔬*[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{g}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
then show $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$
unfolding *the-cat-sspan-components*(*5*) *assms*
using *cat-ss-ineq*
by (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-sspan-Comp-app-𝔣𝔬*[*cat-ss-cs-simps*]:
assumes $g = \mathfrak{f}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-sspan-composable* **by** *auto*
then show $g \circ_{A \leftarrow \bullet \rightarrow C} f = g$
unfolding *the-cat-sspan-components*(*5*) *assms*
using *cat-ss-ineq*
by (*auto simp*: *nat-omega-simps*)

qed

12.3.7 Identity

mk-VLambda *the-cat-scospan-components*(*6*)[*folded VLambda-vid-on*]
|*vsv the-cat-scospan-CId-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-scospan-CId-vdomain*[*cat-ss-cs-simps*]|
|*app the-cat-scospan-CId-app*[*cat-ss-cs-simps*]|

mk-VLambda *the-cat-sspan-components*(*6*)[*folded VLambda-vid-on*]
|*vsv the-cat-sspan-CId-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cat-sspan-CId-vdomain*[*cat-ss-cs-simps*]|
|*app the-cat-sspan-CId-app*[*cat-ss-cs-simps*]|

12.3.8 Arrow with a domain and a codomain

lemma *the-cat-scospan-is-arr-*$\mathfrak{aaa}$[*cat-ss-cs-intros*]:
assumes $a' = \mathfrak{a}_{SS}$ **and** $b' = \mathfrak{a}_{SS}$ **and** $f = \mathfrak{a}_{SS}$
shows $f : a' \mapsto_{\rightarrow\cdot\leftarrow_C} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\rightarrow\cdot\leftarrow_C(\!|Dom|\!)(\!|\mathfrak{a}_{SS}|\!) = \mathfrak{a}_{SS}$ $\rightarrow\cdot\leftarrow_C(\!|Cod|\!)(\!|\mathfrak{a}_{SS}|\!) = \mathfrak{a}_{SS}$
by (*cs-concl* **cs-simp**: *cat-ss-cs-simps*)+
qed (*auto simp*: *the-cat-scospan-components*)

lemma *the-cat-scospan-is-arr-*$\mathfrak{bbb}$[*cat-ss-cs-intros*]:
assumes $a' = \mathfrak{b}_{SS}$ **and** $b' = \mathfrak{b}_{SS}$ **and** $f = \mathfrak{b}_{SS}$
shows $f : a' \mapsto_{\rightarrow\cdot\leftarrow_C} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\rightarrow\cdot\leftarrow_C(\!|Dom|\!)(\!|\mathfrak{b}_{SS}|\!) = \mathfrak{b}_{SS}$ $\rightarrow\cdot\leftarrow_C(\!|Cod|\!)(\!|\mathfrak{b}_{SS}|\!) = \mathfrak{b}_{SS}$
by (*cs-concl* **cs-simp**: *cat-ss-cs-simps*)+
qed (*auto simp*: *the-cat-scospan-components*)

lemma *the-cat-scospan-is-arr-*$\mathfrak{ooo}$[*cat-ss-cs-intros*]:
assumes $a' = \mathfrak{o}_{SS}$ **and** $b' = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{o}_{SS}$
shows $f : a' \mapsto_{\rightarrow\cdot\leftarrow_C} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\rightarrow\cdot\leftarrow_C(\!|Dom|\!)(\!|\mathfrak{o}_{SS}|\!) = \mathfrak{o}_{SS}$ $\rightarrow\cdot\leftarrow_C(\!|Cod|\!)(\!|\mathfrak{o}_{SS}|\!) = \mathfrak{o}_{SS}$
by (*cs-concl* **cs-simp**: *cat-ss-cs-simps*)+
qed (*auto simp*: *the-cat-scospan-components*)

lemma *the-cat-scospan-is-arr-*$\mathfrak{aog}$[*cat-ss-cs-intros*]:
assumes $a' = \mathfrak{a}_{SS}$ **and** $b' = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{g}_{SS}$
shows $f : a' \mapsto_{\rightarrow\cdot\leftarrow_C} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\rightarrow\cdot\leftarrow_C(\!|Dom|\!)(\!|\mathfrak{g}_{SS}|\!) = \mathfrak{a}_{SS}$ $\rightarrow\cdot\leftarrow_C(\!|Cod|\!)(\!|\mathfrak{g}_{SS}|\!) = \mathfrak{o}_{SS}$
by (*cs-concl* **cs-simp**: *cat-ss-cs-simps*)+
qed (*auto simp*: *the-cat-scospan-components*)

lemma *the-cat-scospan-is-arr-*$\mathfrak{bof}$[*cat-ss-cs-intros*]:
assumes $a' = \mathfrak{b}_{SS}$ **and** $b' = \mathfrak{o}_{SS}$ **and** $f = \mathfrak{f}_{SS}$
shows $f : a' \mapsto_{\rightarrow\cdot\leftarrow_C} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\rightarrow\cdot\leftarrow_C(\!|Dom|\!)(\!|\mathfrak{f}_{SS}|\!) = \mathfrak{b}_{SS}$ $\rightarrow\cdot\leftarrow_C(\!|Cod|\!)(\!|\mathfrak{f}_{SS}|\!) = \mathfrak{o}_{SS}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-ss-cs-simps*)+
qed (*auto simp*: *the-cat-scospan-components*)

lemma *the-cat-scospan-is-arrE*:
assumes $f' : a' \mapsto_{\rightarrow\cdot\leftarrow_C} b'$

obtains $a' = \mathfrak{a}_{SS}$ **and** $b' = \mathfrak{a}_{SS}$ **and** $f' = \mathfrak{a}_{SS}$
| $a' = \mathfrak{b}_{SS}$ **and** $b' = \mathfrak{b}_{SS}$ **and** $f' = \mathfrak{b}_{SS}$
| $a' = \mathfrak{o}_{SS}$ **and** $b' = \mathfrak{o}_{SS}$ **and** $f' = \mathfrak{o}_{SS}$
| $a' = \mathfrak{a}_{SS}$ **and** $b' = \mathfrak{o}_{SS}$ **and** $f' = \mathfrak{g}_{SS}$
| $a' = \mathfrak{b}_{SS}$ **and** $b' = \mathfrak{o}_{SS}$ **and** $f' = \mathfrak{f}_{SS}$
proof–
note f = *is-arrD*[*OF assms*]
from $f(1)$ **consider** ($\mathfrak{a}_{SS}$) ‹$f' = \mathfrak{a}_{SS}$›
| ($\mathfrak{b}_{SS}$) ‹$f' = \mathfrak{b}_{SS}$›
| ($\mathfrak{o}_{SS}$) ‹$f' = \mathfrak{o}_{SS}$›
| ($\mathfrak{g}_{SS}$) ‹$f' = \mathfrak{g}_{SS}$›
| ($\mathfrak{f}_{SS}$) ‹$f' = \mathfrak{f}_{SS}$›
by (*elim the-cat-scospan-ArrE*)
then show *?thesis*
proof *cases*
case $\mathfrak{a}_{SS}$
moreover from $f(2,3)$[*unfolded* $\mathfrak{a}_{SS}$, *symmetric*] **have** $a' = \mathfrak{a}_{SS}$ $b' = \mathfrak{a}_{SS}$
by (*simp-all add*: *cat-ss-cs-simps*)
ultimately show *?thesis* **using** *that* **by** *auto*
next
case $\mathfrak{b}_{SS}$
moreover from $f(2,3)$[*unfolded* $\mathfrak{b}_{SS}$, *symmetric*] **have** $a' = \mathfrak{b}_{SS}$ $b' = \mathfrak{b}_{SS}$
by (*simp-all add*: *cat-ss-cs-simps*)
ultimately show *?thesis* **using** *that* **by** *auto*
next
case $\mathfrak{o}_{SS}$
moreover from $f(2,3)$[*unfolded* $\mathfrak{o}_{SS}$, *symmetric*] **have** $a' = \mathfrak{o}_{SS}$ $b' = \mathfrak{o}_{SS}$
by (*simp-all add*: *cat-ss-cs-simps*)
ultimately show *?thesis* **using** *that* **by** *auto*
next
case $\mathfrak{g}_{SS}$
moreover have $a' = \mathfrak{a}_{SS}$ $b' = \mathfrak{o}_{SS}$
by (*simp-all add*: $f(2,3)$[*unfolded* $\mathfrak{g}_{SS}$, *symmetric*] *cat-ss-cs-simps*)
ultimately show *?thesis* **using** *that* **by** *auto*
next
case $\mathfrak{f}_{SS}$
moreover have $a' = \mathfrak{b}_{SS}$ $b' = \mathfrak{o}_{SS}$
by (*simp-all add*: $f(2,3)$[*unfolded* $\mathfrak{f}_{SS}$, *symmetric*] *cat-ss-cs-simps*)
ultimately show *?thesis* **using** *that* **by** *auto*
qed
qed

12.3.9 $\rightarrow\cdot\leftarrow$ is a finite category

lemma (**in** $\mathcal{Z}$) *finite-category-the-cat-scospan*[*cat-ss-cs-intros*]:
finite-category α ($\rightarrow\cdot\leftarrow_C$)
proof(*intro finite-categoryI'' tiny-categoryI''*)
show *vfsequence* ($\rightarrow\cdot\leftarrow_C$) **unfolding** *the-cat-scospan-def* **by** *simp*
show *vcard* ($\rightarrow\cdot\leftarrow_C$) = $6_{\mathbf{N}}$
unfolding *the-cat-scospan-def* **by** (*simp-all add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ ($\rightarrow\cdot\leftarrow_C(\!|Dom|\!)$) $\subseteq_\circ$ $\rightarrow\cdot\leftarrow_C(\!|Obj|\!)$ **by** (*auto simp*: *the-cat-scospan-components*)
show $\mathcal{R}_\circ$ ($\rightarrow\cdot\leftarrow_C(\!|Cod|\!)$) $\subseteq_\circ$ $\rightarrow\cdot\leftarrow_C(\!|Obj|\!)$ **by** (*auto simp*: *the-cat-scospan-components*)
show (*gf* $\in_\circ$ $\mathcal{D}_\circ$ ($\rightarrow\cdot\leftarrow_C(\!|Comp|\!)$)) =
($\exists\, g\ f\ b\ c\ a.$ *gf* = $[g, f]_\circ \wedge g : b \mapsto_{\rightarrow\cdot\leftarrow_C} c \wedge f : a \mapsto_{\rightarrow\cdot\leftarrow_C} b$)
for *gf*
unfolding *the-cat-scospan-Comp-vdomain*
proof
assume *prems*: *gf* $\in_\circ$ *cat-scospan-composable*

```
  then obtain g f where gf-def: gf = [g, f]∘ by auto
  from prems show
    ∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦→•←C c ∧ f : a ↦→•←C b
    unfolding gf-def
    by
      (
        cases rule: cat-scospan-composableE;
        (intro exI conjI)?;
        cs-concl-step?;
        (simp only:)?,
        all‹intro is-arrI, unfold the-cat-scospan-components(2)›
      )
      (cs-concl cs-simp: cat-ss-cs-simps V-cs-simps cs-intro: V-cs-intros)+
next
  assume prems:
    ∃ g f b′ c′ a′. gf = [g, f]∘ ∧ g : b′ ↦→•←C c′ ∧ f : a′ ↦→•←C b′
  then obtain g f b c a
    where gf-def: gf = [g, f]∘
      and g: g : b ↦→•←C c
      and f: f : a ↦→•←C b
    by clarsimp
  from g f show gf ∈∘ cat-scospan-composable
    unfolding gf-def
    by (elim the-cat-scospan-is-arrE) (auto simp: cat-ss-cs-intros)
qed
show D∘ (→•←C(|CId|)) = →•←C(|Obj|)
  by (simp add: cat-ss-cs-simps the-cat-scospan-components)
show g ∘A→•←C f : a ↦→•←C c
  if g : b ↦→•←C c and f : a ↦→•←C b for b c g a f
  using that
  by (elim the-cat-scospan-is-arrE; simp only:)
    (
      all‹
        solves‹simp add: cat-ss-ineq cat-ss-ineq[symmetric]› |
        cs-concl cs-simp: cat-ss-cs-simps cs-intro: cat-ss-cs-intros
      ›
    )
show h ∘A→•←C g ∘A→•←C f = h ∘A→•←C (g ∘A→•←C f)
  if h : c ↦→•←C d and g : b ↦→•←C c and f : a ↦→•←C b
  for c d h b g a f
  using that
  by (elim the-cat-scospan-is-arrE; simp only:)
    (
      all‹
        solves‹simp only: cat-ss-ineq cat-ss-ineq[symmetric]› |
        cs-concl cs-simp: cat-ss-cs-simps cs-intro: cat-ss-cs-intros
        ›
    )
show →•←C(|CId|)(|a|) : a ↦→•←C a if a ∈∘ →•←C(|Obj|) for a
  using that
  by (elim the-cat-scospan-ObjE)
    (
      all‹
        cs-concl
          cs-simp: V-cs-simps cat-ss-cs-simps
          cs-intro: V-cs-intros cat-ss-cs-intros
      ›
    )
```

```
  show →•←_C(|CId|)(|b|) ∘_{A→•←_C} f = f if f : a ↦_{→•←_C} b for a b f
    using that
    by (elim the-cat-scospan-is-arrE)
      (
        cs-concl
          cs-simp: V-cs-simps cat-ss-cs-simps
          cs-intro: V-cs-intros cat-ss-cs-intros
      )+
  show f ∘_{A→•←_C} →•←_C(|CId|)(|b|) = f if f : b ↦_{→•←_C} c for b c f
    using that
    by (elim the-cat-scospan-is-arrE)
      (
        cs-concl
          cs-simp: V-cs-simps cat-ss-cs-simps
          cs-intro: V-cs-intros cat-ss-cs-intros
      )+
qed
  (
    cs-concl
      cs-simp: V-cs-simps cat-ss-cs-simps the-cat-scospan-components(1,2)
      cs-intro: cat-cs-intros cat-ss-cs-intros V-cs-intros
  )+

lemmas [cat-ss-cs-intros] = Z.finite-category-the-cat-scospan
```

12.3.10 Duality for →•← and ←•→

```
lemma the-cat-scospan-op[cat-op-simps]: op-cat (→•←_C) = ←•→_C
proof–
  have dom-lhs: D∘ (op-cat (→•←_C)) = 6_ℕ
    unfolding op-cat-def by (simp add: nat-omega-simps)
  have dom-rhs: D∘ (←•→_C) = 6_ℕ
    unfolding the-cat-sspan-def by (simp add: nat-omega-simps)
  show ?thesis
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    show a ∈∘ 6_ℕ ⟹ op-cat (→•←_C)(|a|) = ←•→_C(|a|) for a
    proof
      (
        elim-in-numeral,
        fold dg-field-simps,
        unfold op-cat-components;
        rule sym
      )
      show ←•→_C(|Comp|) = fflip (→•←_C(|Comp|))
      proof(rule vsv-eqI, unfold cat-ss-cs-simps vdomain-fflip)
        fix gf assume prems: gf ∈∘ cat-sspan-composable
        then obtain g f where gf-def: gf = [g, f]∘ by auto
        from prems have fg: [f, g]∘ ∈∘ cat-scospan-composable
          unfolding gf-def by auto
        have [cat-ss-cs-simps]: g ∘_{A←•→_C} f = f ∘_{A→•←_C} g
          if [f, g]∘ ∈∘ cat-scospan-composable
          using that
          by (elim cat-scospan-composableE; simp only:)
            (cs-concl cs-simp: cat-ss-cs-simps cs-intro: cat-ss-cs-intros)+
        from fg show
          ←•→_C(|Comp|)(|gf|) = fflip (→•←_C(|Comp|))(|gf|)
          unfolding gf-def
          by (cs-concl cs-shallow cs-simp: cat-ss-cs-simps fflip-app)
```

qed (*auto intro*: *fflip-vsv cat-ss-cs-intros*)
qed (*unfold the-cat-sspan-components the-cat-scospan-components*, *simp-all*)
qed (*auto intro*: *cat-op-intros cat-ss-cs-intros*)
qed

lemma (**in** $\mathcal{Z}$) *the-cat-sspan-op*[*cat-op-simps*]: *op-cat* $(\leftarrow\bullet\rightarrow_C) = \rightarrow\bullet\leftarrow_C$
proof–
interpret *scospan*: *finite-category* α $\langle\rightarrow\bullet\leftarrow_C\rangle$
by (*rule finite-category-the-cat-scospan*)
interpret *sspan*: *finite-category* α $\langle\leftarrow\bullet\rightarrow_C\rangle$
by (*rule scospan.finite-category-op*[*unfolded cat-op-simps*])
from *the-cat-scospan-op* **have** *op-cat* $(\leftarrow\bullet\rightarrow_C)$ = *op-cat* (*op-cat* $(\rightarrow\bullet\leftarrow_C)$)
by *simp*
also have ... $= \rightarrow\bullet\leftarrow_C$ **by** (*cs-concl* **cs-shallow cs-simp**: *cat-op-simps*)
finally show *?thesis* **by** *auto*
qed

lemmas [*cat-op-simps*] = $\mathcal{Z}$.*the-cat-sspan-op*

12.3.11 $\leftarrow\bullet\rightarrow$ is a finite category

lemma (**in** $\mathcal{Z}$) *finite-category-the-cat-sspan*[*cat-ss-cs-intros*]:
finite-category α $(\leftarrow\bullet\rightarrow_C)$
proof–
interpret *scospan*: *finite-category* α $\langle\rightarrow\bullet\leftarrow_C\rangle$
by (*rule finite-category-the-cat-scospan*)
show *?thesis* **by** (*rule scospan.finite-category-op*[*unfolded cat-op-simps*])
qed

12.4 Local assumptions for functors from $\rightarrow\bullet\leftarrow$ and $\leftarrow\bullet\rightarrow$

The functors from $\rightarrow\bullet\leftarrow$ and $\leftarrow\bullet\rightarrow$ are introduced as convenient abstractions for the definition of the pullbacks and the pushouts (e.g., see Chapter III-3 and Chapter III-4 in [7]).

12.4.1 Definitions and elementary properties

locale *cf-scospan* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ $\mathfrak{C}$ +
assumes *cf-scospan-*$\mathfrak{g}$[*cat-ss-cs-intros*]: $\mathfrak{g} : \mathfrak{a} \mapsto_{\mathfrak{C}} \mathfrak{o}$
and *cf-scospan-*$\mathfrak{f}$[*cat-ss-cs-intros*]: $\mathfrak{f} : \mathfrak{b} \mapsto_{\mathfrak{C}} \mathfrak{o}$

lemma (**in** *cf-scospan*) *cf-scospan-*$\mathfrak{g}'$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{a}$ **and** $b = \mathfrak{o}$
shows $\mathfrak{g} : a \mapsto_{\mathfrak{C}} b$
unfolding *assms* **by** (*rule cf-scospan-*$\mathfrak{g}$)

lemma (**in** *cf-scospan*) *cf-scospan-*$\mathfrak{g}''$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{g}$ **and** $b = \mathfrak{o}$
shows $g : \mathfrak{a} \mapsto_{\mathfrak{C}} b$
unfolding *assms* **by** (*rule cf-scospan-*$\mathfrak{g}$)

lemma (**in** *cf-scospan*) *cf-scospan-*$\mathfrak{g}'''$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{g}$ **and** $a = \mathfrak{a}$
shows $g : a \mapsto_{\mathfrak{C}} \mathfrak{o}$
unfolding *assms* **by** (*rule cf-scospan-*$\mathfrak{g}$)

lemma (**in** *cf-scospan*) *cf-scospan-*$\mathfrak{f}'$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{b}$ **and** $b = \mathfrak{o}$
shows $\mathfrak{f} : a \mapsto_{\mathfrak{C}} b$

unfolding *assms* **by** (*rule cf-scospan-*$\mathfrak{f}$)

lemma (**in** *cf-scospan*) *cf-scospan-*$\mathfrak{f}''$[*cat-ss-cs-intros*]:
assumes $f = \mathfrak{f}$ **and** $b = \mathfrak{o}$
shows $f : \mathfrak{b} \mapsto_{\mathfrak{C}} b$
unfolding *assms* **by** (*rule cf-scospan-*$\mathfrak{f}$)

lemma (**in** *cf-scospan*) *cf-scospan-*$\mathfrak{f}'''$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{f}$ **and** $b = \mathfrak{b}$
shows $g : b \mapsto_{\mathfrak{C}} \mathfrak{o}$
unfolding *assms* **by** (*rule cf-scospan-*$\mathfrak{f}$)

locale *cf-sspan* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ **and** $\mathfrak{C}$ +
assumes *cf-sspan-*$\mathfrak{g}$[*cat-ss-cs-intros*]: $\mathfrak{g} : \mathfrak{o} \mapsto_{\mathfrak{C}} \mathfrak{a}$
and *cf-sspan-*$\mathfrak{f}$[*cat-ss-cs-intros*]: $\mathfrak{f} : \mathfrak{o} \mapsto_{\mathfrak{C}} \mathfrak{b}$

lemma (**in** *cf-sspan*) *cf-sspan-*$\mathfrak{g}'$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{o}$ **and** $b = \mathfrak{a}$
shows $\mathfrak{g} : a \mapsto_{\mathfrak{C}} b$
unfolding *assms* **by** (*rule cf-sspan-*$\mathfrak{g}$)

lemma (**in** *cf-sspan*) *cf-sspan-*$\mathfrak{g}''$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{g}$ **and** $a = \mathfrak{a}$
shows $g : \mathfrak{o} \mapsto_{\mathfrak{C}} a$
unfolding *assms* **by** (*rule cf-sspan-*$\mathfrak{g}$)

lemma (**in** *cf-sspan*) *cf-sspan-*$\mathfrak{g}'''$[*cat-ss-cs-intros*]:
assumes $g = \mathfrak{g}$ **and** $a = \mathfrak{o}$
shows $g : a \mapsto_{\mathfrak{C}} \mathfrak{a}$
unfolding *assms* **by** (*rule cf-sspan-*$\mathfrak{g}$)

lemma (**in** *cf-sspan*) *cf-sspan-*$\mathfrak{f}'$[*cat-ss-cs-intros*]:
assumes $a = \mathfrak{o}$ **and** $b = \mathfrak{b}$
shows $\mathfrak{f} : a \mapsto_{\mathfrak{C}} b$
unfolding *assms* **by** (*rule cf-sspan-*$\mathfrak{f}$)

lemma (**in** *cf-sspan*) *cf-sspan-*$\mathfrak{f}''$[*cat-ss-cs-intros*]:
assumes $f = \mathfrak{f}$ **and** $b = \mathfrak{b}$
shows $f : \mathfrak{o} \mapsto_{\mathfrak{C}} b$
unfolding *assms* **by** (*rule cf-sspan-*$\mathfrak{f}$)

lemma (**in** *cf-sspan*) *cf-sspan-*$\mathfrak{f}'''$[*cat-ss-cs-intros*]:
assumes $f = \mathfrak{f}$ **and** $b = \mathfrak{o}$
shows $f : b \mapsto_{\mathfrak{C}} \mathfrak{b}$
unfolding *assms* **by** (*rule cf-sspan-*$\mathfrak{f}$)

Rules.

lemmas (**in** *cf-scospan*) [*cat-ss-cs-intros*] = *cf-scospan-axioms*

mk-ide rf *cf-scospan-def*[*unfolded cf-scospan-axioms-def*]
|*intro cf-scospanI*|
|*dest cf-scospanD*[*dest*]|
|*elim cf-scospanE*[*elim*]|

lemmas [*cat-ss-cs-intros*] = *cf-scospanD*(*1*)

lemmas (**in** *cf-sspan*) [*cat-ss-cs-intros*] = *cf-sspan-axioms*

mk-ide rf *cf-sspan-def*[*unfolded cf-sspan-axioms-def*]
|*intro cf-sspanI*|
|*dest cf-sspanD*[*dest*]|
|*elim cf-sspanE*[*elim*]|

Duality.

lemma (**in** *cf-scospan*) *cf-sspan-op*[*cat-op-intros*]:
cf-sspan α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ (*op-cat* $\mathfrak{C}$)
by (*intro cf-sspanI*, *unfold cat-op-simps*)
(*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros cat-ss-cs-intros*)+

lemmas [*cat-op-intros*] = *cf-scospan.cf-sspan-op*

lemma (**in** *cf-sspan*) *cf-scospan-op*[*cat-op-intros*]:
cf-scospan α $\mathfrak{a}$ $\mathfrak{g}$ $\mathfrak{o}$ $\mathfrak{f}$ $\mathfrak{b}$ (*op-cat* $\mathfrak{C}$)
by (*intro cf-scospanI*, *unfold cat-op-simps*)
(*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros cat-ss-cs-intros*)+

lemmas [*cat-op-intros*] = *cf-sspan.cf-scospan-op*

12.5 Functors from →•← and ←•→

12.5.1 Definition and elementary properties

definition *the-cf-scospan* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
($\langle\langle$-→-→-←-←-$\rangle_{CF^{1\rangle}}$ [*51*, *51*, *51*, *51*, *51*] *999*)
where $\langle\mathfrak{a}{\to}\mathfrak{g}{\to}\mathfrak{o}{\leftarrow}\mathfrak{f}{\leftarrow}\mathfrak{b}\rangle_{CF\mathfrak{C}}$ =
[
(
$\lambda a{\in_\circ}{\to}\bullet{\leftarrow}_C(\!|Obj|\!)$.
if $a = \mathfrak{a}_{SS} \Rightarrow \mathfrak{a}$
| $a = \mathfrak{b}_{SS} \Rightarrow \mathfrak{b}$
| *otherwise* $\Rightarrow \mathfrak{o}$
),
(
$\lambda f{\in_\circ}{\to}\bullet{\leftarrow}_C(\!|Arr|\!)$.
if $f = \mathfrak{a}_{SS} \Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}|\!)$
| $f = \mathfrak{b}_{SS} \Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}|\!)$
| $f = \mathfrak{g}_{SS} \Rightarrow \mathfrak{g}$
| $f = \mathfrak{f}_{SS} \Rightarrow \mathfrak{f}$
| *otherwise* $\Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{o}|\!)$
),
${\to}\bullet{\leftarrow}_C$,
$\mathfrak{C}$
]$_\circ$

definition *the-cf-sspan* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
($\langle\langle$-←-←-→-→-$\rangle_{CF^{1\rangle}}$ [*51*, *51*, *51*, *51*, *51*] *999*)
where $\langle\mathfrak{a}{\leftarrow}\mathfrak{g}{\leftarrow}\mathfrak{o}{\to}\mathfrak{f}{\to}\mathfrak{b}\rangle_{CF\mathfrak{C}}$ =
[
(
$\lambda a{\in_\circ}{\leftarrow}\bullet{\to}_C(\!|Obj|\!)$.
if $a = \mathfrak{a}_{SS} \Rightarrow \mathfrak{a}$
| $a = \mathfrak{b}_{SS} \Rightarrow \mathfrak{b}$
| *otherwise* $\Rightarrow \mathfrak{o}$
),
(
$\lambda f{\in_\circ}{\leftarrow}\bullet{\to}_C(\!|Arr|\!)$.

```
      if f = 𝔞SS ⇒ 𝔈(|CId|)(|𝔞|)
       | f = 𝔟SS ⇒ 𝔈(|CId|)(|𝔟|)
       | f = 𝔤SS ⇒ 𝔤
       | f = 𝔣SS ⇒ 𝔣
       | otherwise ⇒ 𝔈(|CId|)(|𝔬|)
    ),
    ←•→C,
    𝔈
  ]∘
```

Components.

lemma *the-cf-scospan-components*:
shows $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor ObjMap \rfloor) =$
```
  (
    λa∈∘→•←C(|Obj|).
     if a = 𝔞SS ⇒ 𝔞
      | a = 𝔟SS ⇒ 𝔟
      | otherwise ⇒ 𝔬
  )
```
and $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor ArrMap \rfloor) =$
```
    (
      λf∈∘→•←C(|Arr|).
       if f = 𝔞SS ⇒ 𝔈(|CId|)(|𝔞|)
        | f = 𝔟SS ⇒ 𝔈(|CId|)(|𝔟|)
        | f = 𝔤SS ⇒ 𝔤
        | f = 𝔣SS ⇒ 𝔣
        | otherwise ⇒ 𝔈(|CId|)(|𝔬|)
    )
```
and [*cat-ss-cs-simps*]: $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor HomDom \rfloor) = \to \bullet \leftarrow_C$
and [*cat-ss-cs-simps*]: $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor HomCod \rfloor) = \mathfrak{C}$
unfolding *the-cf-scospan-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *the-cf-sspan-components*:
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor ObjMap \rfloor) =$
```
  (
    λa∈∘←•→C(|Obj|).
     if a = 𝔞SS ⇒ 𝔞
      | a = 𝔟SS ⇒ 𝔟
      | otherwise ⇒ 𝔬
  )
```
and $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor ArrMap \rfloor) =$
```
    (
      λf∈∘←•→C(|Arr|).
       if f = 𝔞SS ⇒ 𝔈(|CId|)(|𝔞|)
        | f = 𝔟SS ⇒ 𝔈(|CId|)(|𝔟|)
        | f = 𝔤SS ⇒ 𝔤
        | f = 𝔣SS ⇒ 𝔣
        | otherwise ⇒ 𝔈(|CId|)(|𝔬|)
    )
```
and [*cat-ss-cs-simps*]: $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor HomDom \rfloor) = \leftarrow \bullet \to_C$
and [*cat-ss-cs-simps*]: $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF} \mathfrak{C}(\lfloor HomCod \rfloor) = \mathfrak{C}$
unfolding *the-cf-sspan-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Elementary properties.

lemma *the-cf-scospan-components-vsv*[*cat-ss-cs-intros*]: *vsv* $(\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF} \mathfrak{C})$
unfolding *the-cf-scospan-def* **by** *auto*

lemma *the-cf-sspan-components-vsv*[*cat-ss-cs-intros*]: *vsv* (⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩$_{CF\mathfrak{C}}$)
unfolding *the-cf-sspan-def* **by** *auto*

12.5.2 Object map.

mk-VLambda *the-cf-scospan-components*(*1*)
|*vsv the-cf-scospan-ObjMap-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cf-scospan-ObjMap-vdomain*[*cat-ss-cs-simps*]|
|*app the-cf-scospan-ObjMap-app*|

lemma *the-cf-scospan-ObjMap-app-𝔞*[*cat-ss-cs-simps*]:
assumes *x* = 𝔞$_{SS}$
shows ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩$_{CF\mathfrak{C}}$(|*ObjMap*|)(|*x*|) = 𝔞
by
(
cs-concl
cs-simp: *the-cf-scospan-ObjMap-app V-cs-simps assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ObjMap-app-𝔟*[*cat-ss-cs-simps*]:
assumes *x* = 𝔟$_{SS}$
shows ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩$_{CF\mathfrak{C}}$(|*ObjMap*|)(|*x*|) = 𝔟
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-scospan-ObjMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ObjMap-app-𝔬*[*cat-ss-cs-simps*]:
assumes *x* = 𝔬$_{SS}$
shows ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩$_{CF\mathfrak{C}}$(|*ObjMap*|)(|*x*|) = 𝔬
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-scospan-ObjMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ObjMap-vrange*:
$\mathcal{R}_\circ$ (⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩$_{CF\mathfrak{C}}$(|*ObjMap*|)) $\subseteq_\circ$ 𝔈(|*Obj*|)
proof
(
intro vsv.vsv-vrange-vsubset,
unfold the-cf-scospan-ObjMap-vdomain,
intro the-cf-scospan-ObjMap-vsv
)
fix *a* **assume** *a* $\in_\circ$ →•←$_C$(|*Obj*|)
then consider ‹*a* = 𝔞$_{SS}$› | ‹*a* = 𝔟$_{SS}$› | ‹*a* = 𝔬$_{SS}$›
unfolding *the-cat-scospan-components* **by** *auto*
then show ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩$_{CF\mathfrak{C}}$(|*ObjMap*|)(|*a*|) $\in_\circ$ 𝔈(|*Obj*|)
by *cases*
(
cs-concl
cs-simp: *cat-ss-cs-simps* **cs-intro**: *cat-cs-intros cat-ss-cs-intros*

)+
qed

mk-VLambda *the-cf-sspan-components*(*1*)
|*vsv the-cf-sspan-ObjMap-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cf-sspan-ObjMap-vdomain*[*cat-ss-cs-simps*]|
|*app the-cf-sspan-ObjMap-app*|

lemma *the-cf-sspan-ObjMap-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $x = \mathfrak{a}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) = \mathfrak{a}$
by
(
cs-concl
cs-simp: *the-cf-sspan-ObjMap-app V-cs-simps assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ObjMap-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $x = \mathfrak{b}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) = \mathfrak{b}$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ObjMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ObjMap-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $x = \mathfrak{o}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor) = \mathfrak{o}$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ObjMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ObjMap-vrange*:
$\mathcal{R}_{\circ}\ (\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)) \subseteq_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
proof
(
intro vsv.vsv-vrange-vsubset,
unfold the-cf-sspan-ObjMap-vdomain,
intro the-cf-sspan-ObjMap-vsv
)
fix *a* **assume** $a \in_{\circ} \leftarrow\!\bullet\!\rightarrow_C(\lfloor Obj \rfloor)$
then consider $\langle a = \mathfrak{a}_{SS} \rangle \mid \langle a = \mathfrak{b}_{SS} \rangle \mid \langle a = \mathfrak{o}_{SS} \rangle$
unfolding *the-cat-sspan-components* **by** *auto*
then show $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
by *cases*
(
cs-concl
cs-simp: *cat-ss-cs-simps* **cs-intro**: *cat-cs-intros cat-ss-cs-intros*
)+
qed

12.5.3 Arrow map.

mk-VLambda *the-cf-scospan-components*(*2*)
|*vsv the-cf-scospan-ArrMap-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cf-scospan-ArrMap-vdomain*[*cat-ss-cs-simps*]|
|*app the-cf-scospan-ArrMap-app*|

lemma (**in** *cf-scospan*) *the-cf-scospan-ArrMap-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{o}_{SS}$
shows $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(|ArrMap|)(|f|) = \mathfrak{C}(|CId|)(|\mathfrak{o}|)$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-scospan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ArrMap-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{a}_{SS}$
shows $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(|ArrMap|)(|f|) = \mathfrak{C}(|CId|)(|\mathfrak{a}|)$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-scospan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ArrMap-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{b}_{SS}$
shows $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(|ArrMap|)(|f|) = \mathfrak{C}(|CId|)(|\mathfrak{b}|)$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-scospan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ArrMap-app-*$\mathfrak{g}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{g}_{SS}$
shows $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(|ArrMap|)(|f|) = \mathfrak{g}$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-scospan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ArrMap-app-*$\mathfrak{f}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{f}_{SS}$
shows $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(|ArrMap|)(|f|) = \mathfrak{f}$
using *cat-ss-ineq*
by
(
cs-concl

cs-simp: *V-cs-simps the-cf-scospan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-scospan*) *the-cf-scospan-ArrMap-vrange*:
$\mathcal{R}_{\circ}$ ($\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)$) $\subseteq_{\circ}$ $\mathfrak{C}(\lfloor Arr \rfloor)$
proof
(
intro vsv.vsv-vrange-vsubset,
unfold the-cf-scospan-ArrMap-vdomain,
intro the-cf-scospan-ArrMap-vsv
)
fix a **assume** $a \in_{\circ} {\to\cdot\leftarrow}_{C}(\lfloor Arr \rfloor)$
then consider $\langle a = \mathfrak{a}_{SS} \rangle$ | $\langle a = \mathfrak{b}_{SS} \rangle$ | $\langle a = \mathfrak{o}_{SS} \rangle$ | $\langle a = \mathfrak{g}_{SS} \rangle$ | $\langle a = \mathfrak{f}_{SS} \rangle$
unfolding *the-cat-scospan-components* **by** *auto*
then show $\langle \mathfrak{a} \to \mathfrak{g} \to \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor a \rfloor) \in_{\circ} \mathfrak{C}(\lfloor Arr \rfloor)$
by *cases*
(
cs-concl
cs-simp: *cat-ss-cs-simps* **cs-intro**: *cat-cs-intros cat-ss-cs-intros*
)+
qed

mk-VLambda *the-cf-sspan-components*(*2*)
|*vsv the-cf-sspan-ArrMap-vsv*[*cat-ss-cs-intros*]|
|*vdomain the-cf-sspan-ArrMap-vdomain*[*cat-ss-cs-simps*]|
|*app the-cf-sspan-ArrMap-app*|

lemma (**in** *cf-sspan*) *the-cf-sspan-ArrMap-app-*$\mathfrak{o}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{o}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor \mathfrak{o} \rfloor)$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ArrMap-app-*$\mathfrak{a}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{a}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor \mathfrak{a} \rfloor)$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ArrMap-app-*$\mathfrak{b}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{b}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \to \mathfrak{f} \to \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{C}(\lfloor CId \rfloor)(\lfloor \mathfrak{b} \rfloor)$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ArrMap-app assms*

cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ArrMap-app-*$\mathfrak{g}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{g}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{g}$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ArrMap-app-*$\mathfrak{f}$[*cat-ss-cs-simps*]:
assumes $f = \mathfrak{f}_{SS}$
shows $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) = \mathfrak{f}$
using *cat-ss-ineq*
by
(
cs-concl
cs-simp: *V-cs-simps the-cf-sspan-ArrMap-app assms*
cs-intro: *cat-ss-cs-intros*
)

lemma (**in** *cf-sspan*) *the-cf-sspan-ArrMap-vrange*:
$\mathcal{R}_\circ\ (\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof
(
intro vsv.vsv-vrange-vsubset,
unfold the-cf-sspan-ArrMap-vdomain,
intro the-cf-sspan-ArrMap-vsv
)
fix a **assume** $a \in_\circ \leftarrow\!\bullet\!\rightarrow_C(\lfloor Arr \rfloor)$
then consider $\langle a = \mathfrak{a}_{SS} \rangle \mid \langle a = \mathfrak{b}_{SS} \rangle \mid \langle a = \mathfrak{o}_{SS} \rangle \mid \langle a = \mathfrak{g}_{SS} \rangle \mid \langle a = \mathfrak{f}_{SS} \rangle$
unfolding *the-cat-sspan-components* **by** *auto*
then show $\langle \mathfrak{a} \leftarrow \mathfrak{g} \leftarrow \mathfrak{o} \rightarrow \mathfrak{f} \rightarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor a \rfloor) \in_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
by *cases*
(
cs-concl
cs-simp: *cat-ss-cs-simps* **cs-intro**: *cat-cs-intros cat-ss-cs-intros*
)+
qed

12.5.4 Functor from →•← is a functor

lemma (**in** *cf-scospan*) *cf-scospan-the-cf-scospan-is-tm-functor*:
$\langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}} : \rightarrow\!\bullet\!\leftarrow_C \mapsto\!\mapsto_{C.tm\alpha} \mathfrak{C}$
proof(*intro is-functor.cf-is-tm-functor-if-HomDom-finite-category is-functorI'*)
show *vfsequence* $(\langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}})$
unfolding *the-cf-scospan-def* **by** *auto*
show *vcard* $(\langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}) = \aleph_{\mathbb{N}}$
unfolding *the-cf-scospan-def* **by** (*simp add*: *nat-omega-simps*)
show $\langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) :$
$\langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{C}} \langle \mathfrak{a} \rightarrow \mathfrak{g} \rightarrow \mathfrak{o} \leftarrow \mathfrak{f} \leftarrow \mathfrak{b} \rangle_{CF\mathfrak{C}}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
if $f : a \mapsto_{\rightarrow\bullet\leftarrow_C} b$ **for** $a\ b\ f$
using *that*
by (*cases rule*: *the-cat-scospan-is-arrE*; *simp only*:)

```
    (
      cs-concl
        cs-simp: cat-ss-cs-simps cs-intro: cat-cs-intros cat-ss-cs-intros
    )+
  show ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈}(|ArrMap|)(|g ∘_{A→•←_C} f|) =
    ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈}(|ArrMap|)(|g|) ∘_{A𝔈} ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈}(|ArrMap|)(|f|)
    if g : b ↦_{→•←_C} c and f : a ↦_{→•←_C} b for b c g a f
    using that
    by (elim the-cat-scospan-is-arrE)
      (
        all‹simp only:›,
        all‹
          solves‹simp add: cat-ss-ineq cat-ss-ineq[symmetric]› |
          cs-concl
            cs-simp: cat-cs-simps cat-ss-cs-simps
            cs-intro: cat-cs-intros cat-ss-cs-intros
          ›
      )
  show
    ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈}(|ArrMap|)(|→•←_C(|CId|)(|c|)|) =
      𝔈(|CId|)(|⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈}(|ObjMap|)(|c|)|)
    if c ∈_∘ →•←_C(|Obj|) for c
    using that
    by (elim the-cat-scospan-ObjE; simp only:)
      (
        cs-concl
          cs-simp: V-cs-simps cat-ss-cs-simps
          cs-intro: V-cs-intros cat-ss-cs-intros
      )+

qed
  (
    cs-concl
      cs-simp: cat-ss-cs-simps
      cs-intro:
        the-cf-scospan-ObjMap-vrange
        cat-ss-cs-intros cat-cs-intros cat-small-cs-intros
  )+

lemma (in cf-scospan) cf-scospan-the-cf-scospan-is-tm-functor':
  assumes 𝔄' = →•←_C and 𝔈' = 𝔈
  shows ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈} : 𝔄' ↦↦_{C.tmα} 𝔈'
  unfolding assms by (rule cf-scospan-the-cf-scospan-is-tm-functor)

lemmas [cat-ss-cs-intros] = cf-scospan.cf-scospan-the-cf-scospan-is-tm-functor
```

12.5.5 Duality for the functors from →•← and ←•→

```
lemma op-cf-cf-scospan[cat-op-simps]:
  op-cf (⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈}) = ⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_{CF op-cat 𝔈}
proof–
  have dom-lhs: 𝒟_∘ (op-cf (⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈})) = 4_ℕ
    unfolding op-cf-def by (simp add: nat-omega-simps)
  have dom-rhs: 𝒟_∘ (⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_{CF op-cat 𝔈}) = 4_ℕ
    unfolding the-cf-sspan-def by (simp add: nat-omega-simps)
  show ?thesis
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    show op-cf (⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩_{CF𝔈})(|a|) = ⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩_{CF op-cat 𝔈}(|a|)
```

```
    if a ∈∘ 4ℕ for a
    using that
    by
      (
        elim-in-numeral,
        fold dghm-field-simps,
        unfold cat-op-simps the-cf-sspan-components the-cf-scospan-components
      )
      (
        simp-all add:
          the-cat-scospan-components(1,2)
          the-cat-sspan-components(1,2)
          cat-op-simps
      )
  qed (auto intro: cat-op-intros cat-ss-cs-intros)
qed

lemma (in 𝒵) op-cf-cf-scospan[cat-op-simps]:
  op-cf (⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩CF𝔈) = ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩CF op-cat 𝔈
proof–
  have dom-lhs: 𝒟∘ (op-cf (⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩CF𝔈)) = 4ℕ
    unfolding op-cf-def by (simp add: nat-omega-simps)
  have dom-rhs: 𝒟∘ (⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩CF op-cat 𝔈) = 4ℕ
    unfolding the-cf-scospan-def by (simp add: nat-omega-simps)
  show ?thesis
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    show op-cf (⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩CF𝔈)(⌊a⌋) = ⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩CF op-cat 𝔈(⌊a⌋)
      if a ∈∘ 4ℕ for a
      using that
      by
        (
          elim-in-numeral,
          fold dghm-field-simps,
          unfold cat-op-simps the-cf-sspan-components the-cf-scospan-components
        )
        (
          simp-all add:
            the-cat-scospan-components(1,2)
            the-cat-sspan-components(1,2)
            cat-op-simps
        )
  qed (auto intro: cat-op-intros cat-ss-cs-intros)
qed

lemmas [cat-op-simps] = 𝒵.op-cf-cf-scospan
```

12.5.6 Functor from ←•→ is a functor

```
lemma (in cf-sspan) cf-sspan-the-cf-sspan-is-tm-functor:
  ⟨𝔞←𝔤←𝔬→𝔣→𝔟⟩CF𝔈 : ←•→C ↦↦C.tmα 𝔈
proof–
  interpret scospan: cf-scospan α 𝔞 𝔤 𝔬 𝔣 𝔟 ‹op-cat 𝔈› by (rule cf-scospan-op)
  interpret scospan:
    is-tm-functor α ‹→•←C› ‹op-cat 𝔈› ‹⟨𝔞→𝔤→𝔬←𝔣←𝔟⟩CF op-cat 𝔈›
    by (rule scospan.cf-scospan-the-cf-scospan-is-tm-functor)
  show ?thesis by (rule scospan.is-tm-functor-op[unfolded cat-op-simps])
qed
```

lemma (**in** *cf-sspan*) *cf-sspan-the-cf-sspan-is-tm-functor'*:
assumes $\mathfrak{A}' = \leftarrow\bullet\rightarrow_C$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\langle \mathfrak{a}\leftarrow\mathfrak{g}\leftarrow\mathfrak{o}\rightarrow\mathfrak{f}\rightarrow\mathfrak{b}\rangle_{CF\mathfrak{C}} : \mathfrak{A}' \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule cf-sspan-the-cf-sspan-is-tm-functor*)

lemmas [*cat-ss-cs-intros*] = *cf-sspan.cf-sspan-the-cf-sspan-is-tm-functor*

13 Categories with parallel arrows between two objects

13.1 Background: category with parallel arrows between two objects

named-theorems *cat-parallel-cs-simps*
named-theorems *cat-parallel-cs-intros*

definition $\mathfrak{a}_{PL} :: V \Rightarrow V$ **where** $\mathfrak{a}_{PL}\ F = set\ \{F,\ 0\}$
definition $\mathfrak{b}_{PL} :: V \Rightarrow V$ **where** $\mathfrak{b}_{PL}\ F = set\ \{F,\ 1_{\mathbb{N}}\}$

lemma *cat-PL-*$\mathfrak{a}$*-nin-F*[*cat-parallel-cs-intros*]: $\mathfrak{a}_{PL}\ F \notin_{\circ} F$
unfolding $\mathfrak{a}_{PL}$*-def* **using** *mem-not-sym* **by** *auto*

lemma *cat-PL-*$\mathfrak{b}$*-nin-F*[*cat-parallel-cs-intros*]: $\mathfrak{b}_{PL}\ F \notin_{\circ} F$
unfolding $\mathfrak{b}_{PL}$*-def* **using** *mem-not-sym* **by** *auto*

lemma *cat-PL-*$\mathfrak{ab}$[*cat-parallel-cs-intros*]: $\mathfrak{a}_{PL}\ F \neq \mathfrak{b}_{PL}\ F$
unfolding $\mathfrak{a}_{PL}$*-def* $\mathfrak{b}_{PL}$*-def* **by** (*simp add*: *Set.doubleton-eq-iff*)

lemmas *cat-PL-*$\mathfrak{ba}$[*cat-parallel-cs-intros*] = *cat-PL-*$\mathfrak{ab}$[*symmetric*]

13.2 Composable arrows for a category with parallel arrows between two objects

definition *cat-parallel-composable* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cat-parallel-composable* $\mathfrak{a}\ \mathfrak{b}\ F \equiv$
$set\ \{[\mathfrak{a},\ \mathfrak{a}]_{\circ},\ [\mathfrak{b},\ \mathfrak{b}]_{\circ}\} \cup_{\circ}$
$(F \times_{\bullet} set\ \{\mathfrak{a}\}) \cup_{\circ}$
$(set\ \{\mathfrak{b}\} \times_{\bullet} F)$

Rules.

lemma *cat-parallel-composable-*$\mathfrak{aa}$[*cat-parallel-cs-intros*]:
assumes $g = \mathfrak{a}$ **and** $f = \mathfrak{a}$
shows $[g,\ f]_{\circ} \in_{\circ}$ *cat-parallel-composable* $\mathfrak{a}\ \mathfrak{b}\ F$
unfolding *assms cat-parallel-composable-def* **by** *auto*

lemma *cat-parallel-composable-*$\mathfrak{bf}$[*cat-parallel-cs-intros*]:
assumes $g = \mathfrak{b}$ **and** $f \in_{\circ} F$
shows $[g,\ f]_{\circ} \in_{\circ}$ *cat-parallel-composable* $\mathfrak{a}\ \mathfrak{b}\ F$
using *assms*(*2*) **unfolding** *assms*(*1*) *cat-parallel-composable-def* **by** *auto*

lemma *cat-parallel-composable-*$\mathfrak{fa}$[*cat-parallel-cs-intros*]:
assumes $g \in_{\circ} F$ **and** $f = \mathfrak{a}$
shows $[g,\ f]_{\circ} \in_{\circ}$ *cat-parallel-composable* $\mathfrak{a}\ \mathfrak{b}\ F$
using *assms*(*1*) **unfolding** *assms*(*2*) *cat-parallel-composable-def* **by** *auto*

lemma *cat-parallel-composable-*$\mathfrak{bb}$[*cat-parallel-cs-intros*]:
assumes $g = \mathfrak{b}$ **and** $f = \mathfrak{b}$
shows $[g,\ f]_{\circ} \in_{\circ}$ *cat-parallel-composable* $\mathfrak{a}\ \mathfrak{b}\ F$
unfolding *assms cat-parallel-composable-def* **by** *auto*

lemma *cat-parallel-composableE*:
assumes $[g,\ f]_{\circ} \in_{\circ}$ *cat-parallel-composable* $\mathfrak{a}\ \mathfrak{b}\ F$
obtains $g = \mathfrak{b}$ **and** $f = \mathfrak{b}$
| $g = \mathfrak{b}$ **and** $f \in_{\circ} F$
| $g \in_{\circ} F$ **and** $f = \mathfrak{a}$
| $g = \mathfrak{a}$ **and** $f = \mathfrak{a}$
using *assms that* **unfolding** *cat-parallel-composable-def* **by** *auto*

Elementary properties.

lemma *cat-parallel-composable-fconverse*:
(*cat-parallel-composable* $\mathfrak{a}$ $\mathfrak{b}$ F)$^{-1}_{\bullet}$ = *cat-parallel-composable* $\mathfrak{b}$ $\mathfrak{a}$ F
unfolding *cat-parallel-composable-def* **by** *auto*

13.3 Local assumptions for a category with parallel arrows between two objects

locale *cat-parallel* = $\mathcal{Z}$ α **for** α +
fixes $\mathfrak{a}$ $\mathfrak{b}$ F
assumes *cat-parallel-*$\mathfrak{ab}$[*cat-parallel-cs-intros*]: $\mathfrak{a} \neq \mathfrak{b}$
and *cat-parallel-*$\mathfrak{a}F$[*cat-parallel-cs-intros*]: $\mathfrak{a} \notin_{\circ} F$
and *cat-parallel-*$\mathfrak{b}F$[*cat-parallel-cs-intros*]: $\mathfrak{b} \notin_{\circ} F$
and *cat-parallel-*$\mathfrak{a}$*-in-Vset*[*cat-parallel-cs-intros*]: $\mathfrak{a} \in_{\circ} Vset\ \alpha$
and *cat-parallel-*$\mathfrak{b}$*-in-Vset*[*cat-parallel-cs-intros*]: $\mathfrak{b} \in_{\circ} Vset\ \alpha$
and *cat-parallel-F-in-Vset*[*cat-parallel-cs-intros*]: $F \in_{\circ} Vset\ \alpha$

lemmas (**in** *cat-parallel*) *cat-parallel-ineq* =
cat-parallel-$\mathfrak{ab}$
cat-parallel-$\mathfrak{a}F$
cat-parallel-$\mathfrak{b}F$

Rules.

lemmas (**in** *cat-parallel*) [*cat-parallel-cs-intros*] = *cat-parallel-axioms*

mk-ide rf *cat-parallel-def*[*unfolded cat-parallel-axioms-def*]
|*intro cat-parallelI*|
|*dest cat-parallelD*[*dest*]|
|*elim cat-parallelE*[*elim*]|

Duality.

lemma (**in** *cat-parallel*) *cat-parallel-op*[*cat-op-intros*]:
cat-parallel α $\mathfrak{b}$ $\mathfrak{a}$ F
by (*intro cat-parallelI*)
(*auto intro*!: *cat-parallel-cs-intros cat-parallel-*$\mathfrak{ab}$[*symmetric*])

Elementary properties.

lemma (**in** $\mathcal{Z}$) *cat-parallel-PL*:
assumes $F \in_{\circ} Vset\ \alpha$
shows *cat-parallel* α ($\mathfrak{a}_{PL}$ F) ($\mathfrak{b}_{PL}$ F) F
proof (*rule cat-parallelI*)
show $\mathfrak{a}_{PL}$ $F \in_{\circ} Vset\ \alpha$
unfolding $\mathfrak{a}_{PL}$*-def* **by** (*intro Limit-vdoubleton-in-VsetI assms*) *auto*
show $\mathfrak{b}_{PL}$ $F \in_{\circ} Vset\ \alpha$
unfolding $\mathfrak{b}_{PL}$*-def*
by (*intro Limit-vdoubleton-in-VsetI ord-of-nat-in-Vset assms*) *simp*
qed (*auto simp*: *assms cat-PL-*$\mathfrak{ab}$ *cat-PL-*$\mathfrak{a}$*-nin-F cat-PL-*$\mathfrak{b}$*-nin-F*)

13.4 $\Uparrow$: category with parallel arrows between two objects

13.4.1 Definition and elementary properties

See Chapter I-2 and Chapter III-3 in [7].

definition *the-cat-parallel* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle\Uparrow_C\rangle$)
where $\Uparrow_C$ $\mathfrak{a}$ $\mathfrak{b}$ F =
[

$set\ \{\mathfrak{a},\ \mathfrak{b}\},$
$set\ \{\mathfrak{a},\ \mathfrak{b}\} \cup_\circ F,$
$(\lambda x \in_\circ set\ \{\mathfrak{a},\ \mathfrak{b}\} \cup_\circ F.\ (x = \mathfrak{b}\ ?\ \mathfrak{b} : \mathfrak{a})),$
$(\lambda x \in_\circ set\ \{\mathfrak{a},\ \mathfrak{b}\} \cup_\circ F.\ (x = \mathfrak{a}\ ?\ \mathfrak{a} : \mathfrak{b})),$
(
$\lambda gf \in_\circ cat\text{-}parallel\text{-}composable\ \mathfrak{a}\ \mathfrak{b}\ F.$
$if\ gf = [\mathfrak{b},\ \mathfrak{b}]_\circ \Rightarrow \mathfrak{b}$
$\mid \exists f.\ gf = [\mathfrak{b},\ f]_\circ \Rightarrow gf(\!|1_{\mathbb{N}}|\!)$
$\mid \exists f.\ gf = [f,\ \mathfrak{a}]_\circ \Rightarrow gf(\!|0|\!)$
$\mid otherwise \Rightarrow \mathfrak{a}$
),
$vid\text{-}on\ (set\ \{\mathfrak{a},\ \mathfrak{b}\})$
$]_\circ$

Components.

lemma *the-cat-parallel-components*:
shows $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!) = set\ \{\mathfrak{a},\ \mathfrak{b}\}$
and $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!) = set\ \{\mathfrak{a},\ \mathfrak{b}\} \cup_\circ F$
and $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Dom|\!) = (\lambda x \in_\circ set\ \{\mathfrak{a},\ \mathfrak{b}\} \cup_\circ F.\ (x = \mathfrak{b}\ ?\ \mathfrak{b} : \mathfrak{a}))$
and $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Cod|\!) = (\lambda x \in_\circ set\ \{\mathfrak{a},\ \mathfrak{b}\} \cup_\circ F.\ (x = \mathfrak{a}\ ?\ \mathfrak{a} : \mathfrak{b}))$
and $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Comp|\!) =$
(
$\lambda gf \in_\circ cat\text{-}parallel\text{-}composable\ \mathfrak{a}\ \mathfrak{b}\ F.$
$if\ gf = [\mathfrak{b},\ \mathfrak{b}]_\circ \Rightarrow \mathfrak{b}$
$\mid \exists f.\ gf = [\mathfrak{b},\ f]_\circ \Rightarrow gf(\!|1_{\mathbb{N}}|\!)$
$\mid \exists f.\ gf = [f,\ \mathfrak{a}]_\circ \Rightarrow gf(\!|0|\!)$
$\mid otherwise \Rightarrow \mathfrak{a}$
)
and $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|CId|\!) = vid\text{-}on\ (set\ \{\mathfrak{a},\ \mathfrak{b}\})$
unfolding *the-cat-parallel-def dg-field-simps*
by (*simp-all add*: *nat-omega-simps*)

13.4.2 Objects

lemma *the-cat-parallel-Obj-*$\mathfrak{a}I$[*cat-parallel-cs-intros*]:
assumes $a = \mathfrak{a}$
shows $a \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-parallel-components* **by** *simp*

lemma *the-cat-parallel-Obj-*$\mathfrak{b}I$[*cat-parallel-cs-intros*]:
assumes $a = \mathfrak{b}$
shows $a \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!)$
using *assms* **unfolding** *the-cat-parallel-components* **by** *simp*

lemma *the-cat-parallel-ObjE*:
assumes $a \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!)$
obtains $a = \mathfrak{a} \mid a = \mathfrak{b}$
using *assms* **unfolding** *the-cat-parallel-components*(*1*) **by** *fastforce*

13.4.3 Arrows

lemma *the-cat-parallel-Arr-*$\mathfrak{a}I$[*cat-parallel-cs-intros*]:
assumes $f = \mathfrak{a}$
shows $f \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-parallel-components* **by** *simp*

lemma *the-cat-parallel-Arr-*$\mathfrak{b}I$[*cat-parallel-cs-intros*]:
assumes $f = \mathfrak{b}$

shows $f \in_{\circ} \Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-parallel-components* **by** *simp*

lemma *the-cat-parallel-Arr-FI*[*cat-parallel-cs-intros*]:
assumes $f \in_{\circ} F$
shows $f \in_{\circ} \Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-parallel-components* **by** *simp*

lemma *the-cat-parallel-ArrE*:
assumes $f \in_{\circ} \Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
obtains $f = \mathfrak{a} \mid f = \mathfrak{b} \mid f \in_{\circ} F$
using *assms that* **unfolding** *the-cat-parallel-components* **by** *auto*

13.4.4 Domain

mk-VLambda *the-cat-parallel-components*(*3*)
|*vsv the-cat-parallel-Dom-vsv*[*cat-parallel-cs-intros*]|
|*vdomain the-cat-parallel-Dom-vdomain*[*cat-parallel-cs-simps*]|

lemma (**in** *cat-parallel*) *the-cat-parallel-Dom-app-*$\mathfrak{b}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{b}$
shows $\Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Dom|\!)(\!|f|\!) = \mathfrak{b}$
unfolding *the-cat-parallel-components assms* **by** *simp*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel.the-cat-parallel-Dom-app-*$\mathfrak{b}$

lemma (**in** *cat-parallel*) *the-cat-parallel-Dom-app-F*[*cat-parallel-cs-simps*]:
assumes $f \in_{\circ} F$
shows $\Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Dom|\!)(\!|f|\!) = \mathfrak{a}$
unfolding *the-cat-parallel-components* **using** *assms cat-parallel-ineq* **by** *auto*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel.the-cat-parallel-Dom-app-F*

lemma (**in** *cat-parallel*) *the-cat-parallel-Dom-app-*$\mathfrak{a}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{a}$
shows $\Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Dom|\!)(\!|f|\!) = \mathfrak{a}$
unfolding *the-cat-parallel-components assms* **by** *auto*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel.the-cat-parallel-Dom-app-*$\mathfrak{a}$

13.4.5 Codomain

mk-VLambda *the-cat-parallel-components*(*4*)
|*vsv the-cat-parallel-Cod-vsv*[*cat-parallel-cs-intros*]|
|*vdomain the-cat-parallel-Cod-vdomain*[*cat-parallel-cs-simps*]|

lemma (**in** *cat-parallel*) *the-cat-parallel-Cod-app-*$\mathfrak{b}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{b}$
shows $\Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Cod|\!)(\!|f|\!) = \mathfrak{b}$
unfolding *the-cat-parallel-components assms* **by** *simp*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel.the-cat-parallel-Cod-app-*$\mathfrak{b}$

lemma (**in** *cat-parallel*) *the-cat-parallel-Cod-app-F*[*cat-parallel-cs-simps*]:
assumes $f \in_{\circ} F$
shows $\Uparrow_C \mathfrak{a}\ \mathfrak{b}\ F(\!|Cod|\!)(\!|f|\!) = \mathfrak{b}$
unfolding *the-cat-parallel-components* **using** *assms cat-parallel-ineq* **by** *auto*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel.the-cat-parallel-Cod-app-F*

lemma (**in** *cat-parallel*) *the-cat-parallel-Cod-app-*𝔞[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{a}$
shows $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Cod \rfloor)(\lfloor f \rfloor) = \mathfrak{a}$
unfolding *the-cat-parallel-components assms* **by** *auto*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel.the-cat-parallel-Cod-app-*𝔞

13.4.6 Composition

mk-VLambda *the-cat-parallel-components*(*5*)
|*vsv the-cat-parallel-Comp-vsv*[*cat-parallel-cs-intros*]|
|*vdomain the-cat-parallel-Comp-vdomain*[*cat-parallel-cs-simps*]|
|*app the-cat-parallel-Comp-app*[*cat-parallel-cs-simps*]|

lemma *the-cat-parallel-Comp-app-*𝔟𝔟[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{b}$ **and** $f = \mathfrak{b}$
shows $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = g\ g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-parallel-composable* 𝔞 𝔟 F
by (*cs-concl* **cs-shallow cs-intro**: *cat-parallel-cs-intros*)
then show $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = g\ g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = f$
unfolding *the-cat-parallel-components*(*5*) *assms*
by (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-parallel-Comp-app-*𝔞𝔞[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{a}$ **and** $f = \mathfrak{a}$
shows $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = g\ g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-parallel-composable* 𝔞 𝔟 F
by (*cs-concl* **cs-intro**: *cat-parallel-cs-intros*)
then show $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = g\ g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = f$
unfolding *the-cat-parallel-components*(*5*) *assms*
by (*auto simp*: *nat-omega-simps*)
qed

lemma *the-cat-parallel-Comp-app-*𝔟F[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{b}$ **and** $f \in_\circ F$
shows $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = f$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-parallel-composable* 𝔞 𝔟 F
by (*cs-concl* **cs-intro**: *cat-parallel-cs-intros*)
then show $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = f$
unfolding *the-cat-parallel-components*(*5*) *assms*
by (*auto simp*: *nat-omega-simps*)
qed

lemma (**in** *cat-parallel*) *the-cat-parallel-Comp-app-F*𝔞[*cat-parallel-cs-simps*]:
assumes $g \in_\circ F$ **and** $f = \mathfrak{a}$
shows $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = g$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *cat-parallel-composable* 𝔞 𝔟 F
by (*cs-concl* **cs-intro**: *cat-parallel-cs-intros*)
then show $g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f = g$
unfolding *the-cat-parallel-components*(*5*)

using *assms cat-parallel-ineq*
by (*auto simp*: *nat-omega-simps*)
qed

13.4.7 Identity

mk-VLambda *the-cat-parallel-components*(*6*)[*unfolded VLambda-vid-on*[*symmetric*]]
|*vsv the-cat-parallel-CId-vsv*[*cat-parallel-cs-intros*]|
|*vdomain the-cat-parallel-CId-vdomain*[*cat-parallel-cs-simps*]|
|*app the-cat-parallel-CId-app*|

lemma *the-cat-parallel-CId-app-*𝔞[*cat-parallel-cs-simps*]:
assumes $a = \mathfrak{a}$
shows $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor CId \rfloor)(\lfloor a \rfloor) = \mathfrak{a}$
unfolding *assms* **by** (*auto simp*: *the-cat-parallel-CId-app*)

lemma *the-cat-parallel-CId-app-*𝔟[*cat-parallel-cs-simps*]:
assumes $a = \mathfrak{b}$
shows $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor CId \rfloor)(\lfloor a \rfloor) = \mathfrak{b}$
unfolding *assms* **by** (*auto simp*: *the-cat-parallel-CId-app*)

13.4.8 Arrow with a domain and a codomain

lemma (**in** *cat-parallel*) *the-cat-parallel-is-arr-*𝔞𝔞𝔞[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{a}$ **and** $b' = \mathfrak{a}$ **and** $f = \mathfrak{a}$
shows $f : a' \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Dom \rfloor)(\lfloor \mathfrak{a} \rfloor) = \mathfrak{a}\ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Cod \rfloor)(\lfloor \mathfrak{a} \rfloor) = \mathfrak{a}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-parallel-cs-simps* **cs-intro**: *V-cs-intros*)+
qed (*auto simp*: *the-cat-parallel-components*)

lemma (**in** *cat-parallel*) *the-cat-parallel-is-arr-*𝔟𝔟𝔟[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{b}$ **and** $b' = \mathfrak{b}$ **and** $f = \mathfrak{b}$
shows $f : a' \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} b'$
proof(*intro is-arrI*, *unfold assms*)
show $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Dom \rfloor)(\lfloor \mathfrak{b} \rfloor) = \mathfrak{b}\ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Cod \rfloor)(\lfloor \mathfrak{b} \rfloor) = \mathfrak{b}$
by (*cs-concl* **cs-simp**: *cat-parallel-cs-simps* **cs-intro**: *V-cs-intros*)+
qed (*auto simp*: *the-cat-parallel-components*)

lemma (**in** *cat-parallel*) *the-cat-parallel-is-arr-*𝔞𝔟*F*[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{a}$ **and** $b' = \mathfrak{b}$ **and** $f \in_\circ F$
shows $f : a' \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} b'$
proof(*intro is-arrI*, *unfold assms*(*1*,*2*))
from *assms*(*3*) **show** $\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Dom \rfloor)(\lfloor f \rfloor) = \mathfrak{a}\ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\lfloor Cod \rfloor)(\lfloor f \rfloor) = \mathfrak{b}$
by (*cs-concl* **cs-simp**: *cat-parallel-cs-simps* **cs-intro**: *V-cs-intros*)+
qed (*auto simp*: *the-cat-parallel-components assms*(*3*))

lemma (**in** *cat-parallel*) *the-cat-parallel-is-arrE*:
assumes $f' : a' \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} b'$
obtains $a' = \mathfrak{a}$ **and** $b' = \mathfrak{a}$ **and** $f' = \mathfrak{a}$
| $a' = \mathfrak{b}$ **and** $b' = \mathfrak{b}$ **and** $f' = \mathfrak{b}$
| $a' = \mathfrak{a}$ **and** $b' = \mathfrak{b}$ **and** $f' \in_\circ F$
proof–
note f = *is-arrD*[*OF assms*]
from f(*1*) **consider** (𝔞) ‹$f' = \mathfrak{a}$› | (𝔟) ‹$f' = \mathfrak{b}$› | (*F*) ‹$f' \in_\circ F$›
unfolding *the-cat-parallel-components*(*2*) **by** *auto*
then show *?thesis*
proof *cases*

```
    case 𝔞
    moreover from f(2)[unfolded 𝔞, symmetric] have a′ = 𝔞
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-parallel-cs-simps cs-intro: V-cs-intros
        )
    moreover from f(3)[unfolded 𝔞, symmetric] have b′ = 𝔞
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-parallel-cs-simps cs-intro: V-cs-intros
        )
    ultimately show ?thesis using that by auto
  next
    case 𝔟
    moreover from f(2)[unfolded 𝔟, symmetric] have a′ = 𝔟
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-parallel-cs-simps cs-intro: V-cs-intros
        )
    moreover from f(3)[unfolded 𝔟, symmetric] have b′ = 𝔟
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-parallel-cs-simps cs-intro: V-cs-intros
        )
    ultimately show ?thesis using that by auto
  next
    case F
    moreover from f(2)[symmetric] F have a′ = 𝔞
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-parallel-cs-simps cs-intro: V-cs-intros
        )
    moreover from f(3)[symmetric] F have b′ = 𝔟
      by (cs-prems cs-shallow cs-simp: cat-parallel-cs-simps)
    ultimately show ?thesis using that by auto
  qed
qed
```

13.4.9 $\Uparrow$ is a category

```
lemma (in cat-parallel) tiny-category-the-cat-parallel[cat-parallel-cs-intros]:
  tiny-category α (⇑C 𝔞 𝔟 F)
proof(intro tiny-categoryI′′)
  show vfsequence (⇑C 𝔞 𝔟 F) unfolding the-cat-parallel-def by simp
  show vcard (⇑C 𝔞 𝔟 F) = 6N
    unfolding the-cat-parallel-def by (simp-all add: nat-omega-simps)
  show ℛ∘ (⇑C 𝔞 𝔟 F(|Dom|)) ⊆∘ ⇑C 𝔞 𝔟 F(|Obj|)
    by (auto simp: the-cat-parallel-components)
  show ℛ∘ (⇑C 𝔞 𝔟 F(|Cod|)) ⊆∘ ⇑C 𝔞 𝔟 F(|Obj|)
    by (auto simp: the-cat-parallel-components)
  show (gf ∈∘ 𝒟∘ (⇑C 𝔞 𝔟 F(|Comp|))) =
    (∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦⇑C 𝔞 𝔟 F c ∧ f : a ↦⇑C 𝔞 𝔟 F b)
    for gf
```

```
  unfolding the-cat-parallel-Comp-vdomain
proof
  assume prems: gf ∈∘ cat-parallel-composable 𝔞 𝔟 F
  then obtain g f where gf-def: gf = [g, f]∘
    unfolding cat-parallel-composable-def by auto
  from prems show
    ∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦⇑C 𝔞 𝔟 F c ∧ f : a ↦⇑C 𝔞 𝔟 F b
    unfolding gf-def
    by
      (
        cases rule: cat-parallel-composableE;
        (intro exI conjI)?;
        cs-concl-step?;
        (simp only:)?,
        all‹intro is-arrI, unfold the-cat-parallel-components(2)›
      )
      (
        cs-concl
          cs-simp: cat-parallel-cs-simps V-cs-simps cs-intro: V-cs-intros
      )+
next
  assume
    ∃ g f b′ c′ a′.
      gf = [g, f]∘ ∧ g : b′ ↦⇑C 𝔞 𝔟 F c′ ∧ f : a′ ↦⇑C 𝔞 𝔟 F b′
  then obtain g f b c a
    where gf-def: gf = [g, f]∘
      and g: g : b ↦⇑C 𝔞 𝔟 F c
      and f: f : a ↦⇑C 𝔞 𝔟 F b
    by clarsimp
  from g f show gf ∈∘ cat-parallel-composable 𝔞 𝔟 F
    unfolding gf-def
    by (elim the-cat-parallel-is-arrE) (auto simp: cat-parallel-cs-intros)
qed
show 𝒟∘ (⇑C 𝔞 𝔟 F(|CId|)) = ⇑C 𝔞 𝔟 F(|Obj|)
  by (simp add: cat-parallel-cs-simps the-cat-parallel-components)
show g ∘A⇑C 𝔞 𝔟 F f : a ↦⇑C 𝔞 𝔟 F c
  if g : b ↦⇑C 𝔞 𝔟 F c and f : a ↦⇑C 𝔞 𝔟 F b for b c g a f
  using that
  by (elim the-cat-parallel-is-arrE; simp only:)
    (
      all‹
        solves‹simp add: cat-parallel-𝔞𝔟[symmetric]› |
        cs-concl cs-simp: cat-parallel-cs-simps
      ›
    )
show
  h ∘A⇑C 𝔞 𝔟 F g ∘A⇑C 𝔞 𝔟 F f =
    h ∘A⇑C 𝔞 𝔟 F (g ∘A⇑C 𝔞 𝔟 F f)
  if h : c ↦⇑C 𝔞 𝔟 F d
    and g : b ↦⇑C 𝔞 𝔟 F c
    and f : a ↦⇑C 𝔞 𝔟 F b
  for c d h b g a f
  using that
  by (elim the-cat-parallel-is-arrE; simp only:)
    (
      all‹
        solves‹simp only: cat-parallel-ineq cat-parallel-𝔞𝔟[symmetric]› |
```

```
        cs-concl
          cs-simp: cat-parallel-cs-simps cs-intro: cat-parallel-cs-intros
        ›
    )
  show ⇑_C 𝔞 𝔟 F(|CId|)(|a|) : a ↦_⇑_C 𝔞 𝔟 F a if a ∈_∘ ⇑_C 𝔞 𝔟 F(|Obj|)
    for a
  proof–
    from that consider ‹a = 𝔞› | ‹a = 𝔟›
      unfolding the-cat-parallel-components(1) by auto
    then show ⇑_C 𝔞 𝔟 F(|CId|)(|a|) : a ↦_⇑_C 𝔞 𝔟 F a
      by cases
        (
          cs-concl
            cs-simp: cat-parallel-cs-simps cs-intro: cat-parallel-cs-intros
        )+
  qed
  show ⇑_C 𝔞 𝔟 F(|CId|)(|b|) ∘_A⇑_C 𝔞 𝔟 F f = f
    if f : a ↦_⇑_C 𝔞 𝔟 F b for a b f
    using that
    by (elim the-cat-parallel-is-arrE)
      (cs-concl cs-simp: cat-parallel-cs-simps cs-intro: cat-parallel-cs-intros)
  show f ∘_A⇑_C 𝔞 𝔟 F ⇑_C 𝔞 𝔟 F(|CId|)(|b|) = f
    if f : b ↦_⇑_C 𝔞 𝔟 F c for b c f
    using that
    by (elim the-cat-parallel-is-arrE)
      (cs-concl cs-simp: cat-parallel-cs-simps cs-intro: cat-parallel-cs-intros)
  show ⇑_C 𝔞 𝔟 F(|Obj|) ∈_∘ Vset α
    by
      (
        cs-concl cs-shallow
          cs-simp: the-cat-parallel-components nat-omega-simps
          cs-intro: V-cs-intros cat-parallel-cs-intros
      )
qed
  (
    cs-concl
      cs-simp:
        nat-omega-simps cat-parallel-cs-simps the-cat-parallel-components(2)
      cs-intro:
        cat-cs-intros
        cat-parallel-cs-intros
        V-cs-intros
        Limit-succ-in-VsetI
  )+

lemmas [cat-parallel-cs-intros] = cat-parallel.tiny-category-the-cat-parallel
```

13.4.10 Opposite parallel category

```
lemma (in cat-parallel) op-cat-the-cat-parallel[cat-op-simps]:
  op-cat (⇑_C 𝔞 𝔟 F) = ⇑_C 𝔟 𝔞 F
proof(rule cat-eqI)
  interpret par: cat-parallel α 𝔟 𝔞 F by (rule cat-parallel-op)
  show 𝔟𝔞: category α (⇑_C 𝔟 𝔞 F)
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-parallel-cs-intros)
  show 𝔞𝔟: category α (op-cat (⇑_C 𝔞 𝔟 F))
    by
      (
```

```
      cs-concl cs-shallow
        cs-intro: cat-small-cs-intros cat-op-intros cat-parallel-cs-intros
    )
  interpret 𝔟𝔞: category α ‹⇑C 𝔟 𝔞 F› by (rule 𝔟𝔞)
  interpret 𝔞𝔟: category α ‹⇑C 𝔞 𝔟 F›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-parallel-cs-intros)
  show op-cat (⇑C 𝔞 𝔟 F)⦇Comp⦈ = ⇑C 𝔟 𝔞 F⦇Comp⦈
  proof(rule vsv-eqI)
    show vsv (op-cat (⇑C 𝔞 𝔟 F)⦇Comp⦈)
      unfolding op-cat-components by (rule fflip-vsv)
    show vsv (⇑C 𝔟 𝔞 F⦇Comp⦈)
      by (cs-concl cs-shallow cs-intro: cat-parallel-cs-intros)
    show [cat-op-simps]: 𝒟∘ (op-cat (⇑C 𝔞 𝔟 F)⦇Comp⦈) = 𝒟∘ (⇑C 𝔟 𝔞 F⦇Comp⦈)
      by
        (
          cs-concl cs-shallow
            cs-simp:
              cat-parallel-composable-fconverse
              op-cat-components(5)
              vdomain-fflip
              cat-parallel-cs-simps
            cs-intro: cat-cs-intros
        )
    fix gf assume gf ∈∘ 𝒟∘ (op-cat (⇑C 𝔞 𝔟 F)⦇Comp⦈)
    then have gf ∈∘ 𝒟∘ (⇑C 𝔟 𝔞 F⦇Comp⦈) unfolding cat-op-simps by simp
    then obtain g f a b c
      where gf-def: gf = [g, f]∘
        and g: g : b ↦⇑C 𝔟 𝔞 F c
        and f: f : a ↦⇑C 𝔟 𝔞 F b
      by auto
    from g f show op-cat (⇑C 𝔞 𝔟 F)⦇Comp⦈⦇gf⦈ = ⇑C 𝔟 𝔞 F⦇Comp⦈⦇gf⦈
      unfolding gf-def
      by (elim par.the-cat-parallel-is-arrE)
        (
          simp add: cat-parallel-cs-intros |
          cs-concl
            cs-simp: cat-op-simps cat-parallel-cs-simps
            cs-intro: cat-cs-intros cat-parallel-cs-intros
        )+
  qed
  show op-cat (⇑C 𝔞 𝔟 F)⦇CId⦈ = ⇑C 𝔟 𝔞 F⦇CId⦈
  proof(unfold cat-op-simps, rule vsv-eqI, unfold cat-parallel-cs-simps)
    fix a assume a ∈∘ set {𝔞, 𝔟}
    then consider ‹a = 𝔞› | ‹a = 𝔟› by auto
    then show ⇑C 𝔞 𝔟 F⦇CId⦈⦇a⦈ = ⇑C 𝔟 𝔞 F⦇CId⦈⦇a⦈
      by cases (cs-concl cs-simp: cat-parallel-cs-simps)+
  qed auto
qed (auto simp: the-cat-parallel-components op-cat-components)

lemmas [cat-op-simps] = cat-parallel.op-cat-the-cat-parallel
```

13.5 Parallel functor

13.5.1 Background

See Chapter III-3 and Chapter III-4 in [7]).

13.5.2 Local assumptions for the parallel functor

locale *cf-parallel* = *cat-parallel* α $\mathfrak{a}$ $\mathfrak{b}$ F + *category* α $\mathfrak{C}$ + F': *vsv* F'
  for α $\mathfrak{a}$ $\mathfrak{b}$ F $\mathfrak{a}'$ $\mathfrak{b}'$ F' $\mathfrak{C}$ :: V +
  assumes *cf-parallel-F'-vdomain*[*cat-parallel-cs-simps*]: $\mathcal{D}_\circ\ F' = F$
    and *cf-parallel-F'*[*cat-parallel-cs-intros*]: $\mathfrak{f} \in_\circ F \Longrightarrow F'(\!|\mathfrak{f}|\!) : \mathfrak{a}' \mapsto_{\mathfrak{C}} \mathfrak{b}'$
    and *cf-parallel-*$\mathfrak{a}'$[*cat-parallel-cs-intros*]: $\mathfrak{a}' \in_\circ \mathfrak{C}(\!|Obj|\!)$
    and *cf-parallel-*$\mathfrak{b}'$[*cat-parallel-cs-intros*]: $\mathfrak{b}' \in_\circ \mathfrak{C}(\!|Obj|\!)$

lemmas (**in** *cf-parallel*) [*cat-parallel-cs-intros*] = *F'.vsv-axioms*

lemma (**in** *cf-parallel*) *cf-parallel-F''*[*cat-parallel-cs-intros*]:
  assumes $\mathfrak{f} \in_\circ F$ **and** $a = \mathfrak{a}'$ **and** $b = \mathfrak{b}'$
  shows $F'(\!|\mathfrak{f}|\!) : a \mapsto_{\mathfrak{C}} b$
  using *assms*(*1*) **unfolding** *assms*(*2–3*) **by** (*rule cf-parallel-F'*)

lemma (**in** *cf-parallel*) *cf-parallel-F'''*[*cat-parallel-cs-intros*]:
  assumes $\mathfrak{f} \in_\circ F$ **and** $f = F'(\!|\mathfrak{f}|\!)$ **and** $b = \mathfrak{b}'$
  shows $f : \mathfrak{a}' \mapsto_{\mathfrak{C}} b$
  using *assms*(*1*) **unfolding** *assms*(*2–3*) **by** (*rule cf-parallel-F'*)

lemma (**in** *cf-parallel*) *cf-parallel-F''''*[*cat-parallel-cs-intros*]:
  assumes $\mathfrak{f} \in_\circ F$ **and** $f = F'(\!|\mathfrak{f}|\!)$ **and** $a = \mathfrak{a}'$
  shows $f : a \mapsto_{\mathfrak{C}} \mathfrak{b}'$
  using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule cf-parallel-F'*)

Rules.

lemma (**in** *cf-parallel*) *cf-parallel-axioms'*[*cat-parallel-cs-intros*]:
  assumes $\alpha' = \alpha$
    and $a'' = \mathfrak{a}$
    and $b'' = \mathfrak{b}$
    and $F'' = F$
    and $a''' = \mathfrak{a}'$
    and $b''' = \mathfrak{b}'$
    and $F''' = F'$
  shows *cf-parallel* α' a'' b'' F'' a''' b''' F''' $\mathfrak{C}$
  unfolding *assms* **by** (*rule cf-parallel-axioms*)

mk-ide rf *cf-parallel-def*[*unfolded cf-parallel-axioms-def*]
  |*intro cf-parallelI*|
  |*dest cf-parallelD*[*dest*]|
  |*elim cf-parallelE*[*elim*]|

lemmas [*cat-parallel-cs-intros*] = *cf-parallelD*(*1,2*)

Duality.

lemma (**in** *cf-parallel*) *cf-parallel-op*[*cat-op-intros*]:
  cf-parallel α $\mathfrak{b}$ $\mathfrak{a}$ F $\mathfrak{b}'$ $\mathfrak{a}'$ F' (*op-cat* $\mathfrak{C}$)
  by (*intro cf-parallelI*, *unfold cat-op-simps insert-commute*)
    (
      cs-concl
        cs-simp: *cat-parallel-cs-simps*
        cs-intro: *cat-parallel-cs-intros cat-cs-intros cat-op-intros*
)+

lemmas [*cat-op-intros*] = *cf-parallel.cf-parallel-op*

13.5.3 Definition and elementary properties

definition *the-cf-parallel* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
($\langle \Uparrow\!\!\rightarrow\!\!\Uparrow_{CF} \rangle$)
where $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F' =$
[
$(\lambda a \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!).\ (a = \mathfrak{a}\ ?\ \mathfrak{a}' : \mathfrak{b}'))$,
(
$\lambda f \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!).$
(
if $f = \mathfrak{a} \Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)$
$|\ f = \mathfrak{b} \Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)$
$|$ *otherwise* $\Rightarrow F'(\!|f|\!)$
)
),
$\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F$,
$\mathfrak{C}$
]$_\circ$

Components.

lemma *the-cf-parallel-components*:
shows $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!) =$
$(\lambda a \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!).\ (a = \mathfrak{a}\ ?\ \mathfrak{a}' : \mathfrak{b}'))$
and $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!) =$
(
$\lambda f \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!).$
(
if $f = \mathfrak{a} \Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)$
$|\ f = \mathfrak{b} \Rightarrow \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)$
$|$ *otherwise* $\Rightarrow F'(\!|f|\!)$
)
)
and [*cat-parallel-cs-simps*]: $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|HomDom|\!) = \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F$
and [*cat-parallel-cs-simps*]: $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|HomCod|\!) = \mathfrak{C}$
unfolding *the-cf-parallel-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

13.5.4 Object map

mk-VLambda *the-cf-parallel-components*(*1*)
|*vsv the-cf-parallel-ObjMap-vsv*[*cat-parallel-cs-intros*]|
|*vdomain the-cf-parallel-ObjMap-vdomain*[*cat-parallel-cs-simps*]|
|*app the-cf-parallel-ObjMap-app*|

lemma (**in** *cf-parallel*) *the-cf-parallel-ObjMap-app-*$\mathfrak{a}$[*cat-parallel-cs-simps*]:
assumes $x = \mathfrak{a}$
shows $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)(\!|x|\!) = \mathfrak{a}'$
by
(
cs-concl
cs-simp:
assms the-cf-parallel-ObjMap-app cat-parallel-cs-simps V-cs-simps
cs-intro: *cat-parallel-cs-intros*
)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel.the-cf-parallel-ObjMap-app-*$\mathfrak{a}$

lemma (**in** *cf-parallel*) *the-cf-parallel-ObjMap-app-*$\mathfrak{b}$[*cat-parallel-cs-simps*]:
assumes $x = \mathfrak{b}$

shows $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)(\!|x|\!) = \mathfrak{b}'$
using *cat-parallel-ineq*
by
(
cs-concl
cs-simp:
assms the-cf-parallel-ObjMap-app cat-parallel-cs-simps V-cs-simps
cs-intro: *cat-parallel-cs-intros*
)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel.the-cf-parallel-ObjMap-app-*$\mathfrak{b}$

lemma (**in** *cf-parallel*) *the-cf-parallel-ObjMap-vrange*:
$\mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)) = set\ \{\mathfrak{a}', \mathfrak{b}'\}$
proof(*intro vsubset-antisym*)
show $\mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)) \subseteq_{\circ} set\ \{\mathfrak{a}', \mathfrak{b}'\}$
unfolding *the-cf-parallel-components*
by (*intro vrange-VLambda-vsubset*)
(*simp-all add*: *cat-parallel-*$\mathfrak{ab}$ *cf-parallel-*$\mathfrak{a}'$ *cf-parallel-*$\mathfrak{b}'$)
show $set\ \{\mathfrak{a}', \mathfrak{b}'\} \subseteq_{\circ} \mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!))$
proof(*rule vsubsetI*)
fix x **assume** *prems*: $x \in_{\circ} set\ \{\mathfrak{a}', \mathfrak{b}'\}$
from *prems* **consider** $\langle x = \mathfrak{a}' \rangle \mid \langle x = \mathfrak{b}' \rangle$ **by** *auto*
moreover have $\mathfrak{a}' \in_{\circ} \mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!))$
by (*rule vsv.vsv-vimageI2'*[*of* - - $\mathfrak{a}$])
(
cs-concl
cs-simp: *cat-parallel-cs-simps* **cs-intro**: *cat-parallel-cs-intros*
)
moreover have $\mathfrak{b}' \in_{\circ} \mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!))$
by (*rule vsv.vsv-vimageI2'*[*of* - - $\mathfrak{b}$])
(
cs-concl
cs-simp: *cat-parallel-cs-simps* **cs-intro**: *cat-parallel-cs-intros*
)
ultimately show $x \in_{\circ} \mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!))$ **by** *auto*
qed
qed

lemma (**in** *cf-parallel*) *the-cf-parallel-ObjMap-vrange-vsubset-Obj*:
$\mathcal{R}_{\circ}\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
unfolding *the-cf-parallel-components*
by (*intro vrange-VLambda-vsubset*)
(*simp-all add*: *cat-parallel-*$\mathfrak{ab}$ *cf-parallel-*$\mathfrak{a}'$ *cf-parallel-*$\mathfrak{b}'$)

13.5.5 Arrow map

mk-VLambda *the-cf-parallel-components*(*2*)
|*vsv the-cf-parallel-ArrMap-vsv*[*cat-parallel-cs-intros*]|
|*vdomain the-cf-parallel-ArrMap-vdomain*[*cat-parallel-cs-simps*]|
|*app the-cf-parallel-ArrMap-app*|

lemma (**in** *cf-parallel*) *the-cf-parallel-ArrMap-app-F*[*cat-parallel-cs-simps*]:
assumes $f \in_{\circ} F$
shows $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!) = F'(\!|f|\!)$
proof–
from *assms* **have** $f \in_{\circ} \Uparrow_{C}\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-parallel-cs-intros a-in-succ-xI*)

from *assms this* **show** *?thesis*
using *cat-parallel-ineq*
by (*auto simp*: *the-cf-parallel-ArrMap-app cat-parallel-cs-simps*)
qed

lemmas [*cat-parallel-cs-simps*] = *cf-parallel.the-cf-parallel-ArrMap-app-F*

lemma (**in** *cf-parallel*) *the-cf-parallel-ArrMap-app-*$\mathfrak{a}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{a}$
shows $\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)$
proof–
from *assms* **have** $f \in_{\circ} \Uparrow_{C}\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
by (*cs-concl* **cs-intro**: *cat-parallel-cs-intros a-in-succ-xI*)
from *this* **show** *?thesis*
using *cat-parallel-ineq*
by (*elim the-cat-parallel-ArrE*; *simp only*: *assms*)
(*auto simp*: *the-cf-parallel-ArrMap-app*)
qed

lemmas [*cat-parallel-cs-simps*] = *cf-parallel.the-cf-parallel-ArrMap-app-*$\mathfrak{a}$

lemma (**in** *cf-parallel*) *the-cf-parallel-ArrMap-app-*$\mathfrak{b}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{b}$
shows $\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)$
proof–
from *assms* **have** $f \in_{\circ} \Uparrow_{C}\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
by (*cs-concl* **cs-intro**: *cat-parallel-cs-intros a-in-succ-xI*)
from *this* **show** *?thesis*
using *cat-parallel-ineq*
by (*elim the-cat-parallel-ArrE*; *simp only*: *assms*)
(*auto simp*: *the-cf-parallel-ArrMap-app*)
qed

lemmas [*cat-parallel-cs-simps*] = *cf-parallel.the-cf-parallel-ArrMap-app-*$\mathfrak{b}$

lemma (**in** *cf-parallel*) *the-cf-parallel-ArrMap-vrange*:
$\mathcal{R}_{\circ}$ ($\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)$) = ($F'$ ‘$_{\circ}$ F) $\cup_{\circ}$ *set* {$\mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)$, $\mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)$}
(**is** ‹$\mathcal{R}_{\circ}$ ($\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)$) = *?FF* $\cup_{\circ}$ *?CID*›)
proof(*intro vsubset-antisym*)

show $\mathcal{R}_{\circ}$ ($\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)$) $\subseteq_{\circ}$ *?FF* $\cup_{\circ}$ *?CID*
proof
(
intro vsv.vsv-vrange-vsubset the-cf-parallel-ArrMap-vsv,
unfold the-cf-parallel-ArrMap-vdomain
)
fix *f* **assume** *prems*: $f \in_{\circ} \Uparrow_{C}\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
from *prems* **show** $\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!) \in_{\circ}$ *?FF* $\cup_{\circ}$ *?CID*
by (*elim the-cat-parallel-ArrE*; (*simp only*:) *?*)
(
cs-concl
cs-simp: *vinsert-set-insert-eq cat-parallel-cs-simps*
cs-intro: *vsv.vsv-vimageI1 V-cs-intros*
)
qed

show *?FF* $\cup_{\circ}$ *?CID* $\subseteq_{\circ}$ $\mathcal{R}_{\circ}$ ($\Uparrow_{\to}\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)$)
proof(*rule vsubsetI*)

fix f **assume** *prems*: $f \in_\circ F' \mathbin{`}_\circ F \cup_\circ$ *set* $\{\mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!), \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)\}$
then consider $\langle f \in_\circ F' \mathbin{`}_\circ F\rangle \mid \langle f = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)\rangle \mid \langle f = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)\rangle$ **by** *auto*
then show $f \in_\circ \mathcal{R}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
proof *cases*
assume $f \in_\circ F' \mathbin{`}_\circ F$
then obtain $\mathfrak{f}$ **where** $\mathfrak{f}$: $\mathfrak{f} \in_\circ F$ **and** *f-def*: $f = F'(\!|\mathfrak{f}|\!)$ **by** *auto*
from $\mathfrak{f}$ **have** *f-def'*: $f = \Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|\mathfrak{f}|\!)$
unfolding *f-def* **by** (*cs-concl* **cs-simp**: *cat-parallel-cs-simps*)
from $\mathfrak{f}$ **have** $\mathfrak{f} \in_\circ \mathcal{D}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-parallel-cs-simps* **cs-intro**: *cat-parallel-cs-intros*
)
then show $f \in_\circ \mathcal{R}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
unfolding *f-def'*
by (*auto simp*: *cat-parallel-cs-intros intro*: *vsv.vsv-vimageI2*)
next
assume *prems*: $f = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)$
have *f-def'*: $f = \Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|\mathfrak{a}|\!)$
by (*cs-concl* **cs-simp**: *cat-parallel-cs-simps prems*)
have $\mathfrak{a} \in_\circ \mathcal{D}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
by
(
cs-concl
cs-simp: *cat-parallel-cs-simps* **cs-intro**: *cat-parallel-cs-intros*
)
then show $f \in_\circ \mathcal{R}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
unfolding *f-def'*
by (*auto simp*: *cat-parallel-cs-intros intro*: *vsv.vsv-vimageI2*)
next
assume *prems*: $f = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)$
have *f-def'*: $f = \Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|\mathfrak{b}|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *V-cs-simps cat-parallel-cs-simps prems*)
have $\mathfrak{b} \in_\circ \mathcal{D}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
by
(
cs-concl
cs-simp: *cat-parallel-cs-simps* **cs-intro**: *cat-parallel-cs-intros*
)
then show $f \in_\circ \mathcal{R}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
unfolding *f-def'*
by (*auto simp*: *cat-parallel-cs-intros intro*: *vsv.vsv-vimageI2*)
qed
qed

qed

lemma (**in** *cf-parallel*) *the-cf-parallel-ArrMap-vrange-vsubset-Arr*:
$\mathcal{R}_\circ\ (\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)) \subseteq_\circ \mathfrak{C}(\!|Arr|\!)$
proof(*intro vsv.vsv-vrange-vsubset*, *unfold cat-parallel-cs-simps*)
show *vsv* $(\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!))$
by (*cs-intro-step cat-parallel-cs-intros*)
fix f **assume** $f \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Arr|\!)$
then show $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!) \in_\circ \mathfrak{C}(\!|Arr|\!)$
by (*elim the-cat-parallel-ArrE*)
(
cs-concl

cs-simp: *cat-parallel-cs-simps*
cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)+
qed

13.5.6 Parallel functor is a functor

lemma (**in** *cf-parallel*) *cf-parallel-the-cf-parallel-is-tm-functor*:
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F' : \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F \mapsto\!\!\mapsto_{C.tm\alpha}\ \mathfrak{C}$
proof(*intro is-tm-functorI' is-functorI'*)

interpret *tcp*: *tiny-category* α ‹$\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F$›
by (*rule tiny-category-the-cat-parallel*)

show *vfsequence* ($\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'$)
unfolding *the-cf-parallel-def* **by** *auto*
show *vcard* ($\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'$) = $4_{\mathbb{N}}$
unfolding *the-cf-parallel-def* **by** (*simp add*: *nat-omega-simps*)
show $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!)$:
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}}$
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)(\!|b|\!)$
if $f : a \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} b$ **for** *a b f*
using *that*
by (*cases rule*: *the-cat-parallel-is-arrE*; *simp only*:)
(
cs-concl
cs-simp: *cat-parallel-cs-simps*
cs-intro: *cat-cs-intros cat-parallel-cs-intros*
)+

show
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|g \circ_{A\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} f|\!) =$
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{C}}$
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|f|\!)$
if $g : b \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} c$ **and** $f : a \mapsto_{\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F} b$ **for** *b c g a f*
using *that*
by (*elim the-cat-parallel-is-arrE*)
(
all‹*simp only*:›,
all‹
solves‹*simp add*: *cat-parallel-ineq cat-parallel-*$\mathfrak{ab}$[*symmetric*]› |
cs-concl
cs-simp: *cat-cs-simps cat-parallel-cs-simps*
cs-intro: *cat-cs-intros cat-parallel-cs-intros*
›
)

show
$\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ArrMap|\!)(\!|\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|CId|\!)(\!|c|\!)|\!) =$
$\mathfrak{C}(\!|CId|\!)(\!|\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!)(\!|c|\!)|\!)$
if $c \in_\circ \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F(\!|Obj|\!)$ **for** *c*
using *that*
by (*elim the-cat-parallel-ObjE*; *simp only*:)
(*cs-concl* **cs-simp**: *cat-parallel-cs-simps*)+

show $\Uparrow\!\!\rightarrow\!\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|ObjMap|\!) \in_\circ Vset\ \alpha$
proof
(

rule vbrelation.vbrelation-Limit-in-VsetI,
unfold
the-cf-parallel-ObjMap-vdomain
the-cf-parallel-ObjMap-vrange
the-cat-parallel-components(*1*);
(*intro Limit-vdoubleton-in-VsetI*)*?*
)
show $\mathfrak{a} \in_\circ \mathit{Vset}\ \alpha\ \mathfrak{b} \in_\circ \mathit{Vset}\ \alpha\ \mathfrak{a}' \in_\circ \mathit{Vset}\ \alpha\ \mathfrak{b}' \in_\circ \mathit{Vset}\ \alpha$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-parallel-cs-intros*)+
qed (*use the-cf-parallel-ObjMap-vsv* **in** *blast*)+

show $\Uparrow\!\!\rightarrow\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'(\!|\mathit{ArrMap}|\!) \in_\circ \mathit{Vset}\ \alpha$
proof
(
rule vbrelation.vbrelation-Limit-in-VsetI,
unfold
the-cf-parallel-ArrMap-vdomain
the-cf-parallel-ArrMap-vrange
the-cat-parallel-components(*1*);
(*intro tcp.tiny-cat-Arr-in-Vset vunion-in-VsetI Limit-vdoubleton-in-VsetI*)*?*
)
show $\mathfrak{C}(\!|\mathit{CId}|\!)(\!|\mathfrak{a}'|\!) \in_\circ \mathit{Vset}\ \alpha\ \mathfrak{C}(\!|\mathit{CId}|\!)(\!|\mathfrak{b}'|\!) \in_\circ \mathit{Vset}\ \alpha$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-parallel-cs-intros*)+
from *cf-parallel-F'* **have** $F' \circ_\circ F \subseteq_\circ \mathit{Hom}\ \mathfrak{C}\ \mathfrak{a}'\ \mathfrak{b}'$
by (*simp add*: *F'.vsv-vimage-vsubsetI*)
moreover have $\mathit{Hom}\ \mathfrak{C}\ \mathfrak{a}'\ \mathfrak{b}' \in_\circ \mathit{Vset}\ \alpha$
by (*auto simp*: *cat-Hom-in-Vset cf-parallel-*$\mathfrak{a}'$ *cf-parallel-*$\mathfrak{b}'$)
ultimately show $F' \circ_\circ F \in_\circ \mathit{Vset}\ \alpha$ **by** *auto*
qed (*use the-cf-parallel-ArrMap-vsv* **in** *blast*)+

qed
(
cs-concl
cs-simp: *cat-parallel-cs-simps*
cs-intro:
the-cf-parallel-ObjMap-vrange-vsubset-Obj
cat-parallel-cs-intros cat-cs-intros cat-small-cs-intros
)+

lemma (**in** *cf-parallel*) *cf-parallel-the-cf-parallel-is-tm-functor'*:
assumes $\mathfrak{A}' = \Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\Uparrow\!\!\rightarrow\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F' : \mathfrak{A}' \mapsto\!\!\mapsto_{C.tm\alpha} \mathfrak{C}'$
unfolding *assms* **by** (*rule cf-parallel-the-cf-parallel-is-tm-functor*)

lemmas [*cat-parallel-cs-intros*] =
cf-parallel.cf-parallel-the-cf-parallel-is-tm-functor'

13.5.7 Opposite parallel functor

lemma (**in** *cf-parallel*) *cf-parallel-the-cf-parallel-op*[*cat-op-simps*]:
op-cf $(\Uparrow\!\!\rightarrow\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F') = \Uparrow\!\!\rightarrow\!\Uparrow_{CF}\ (\mathit{op\text{-}cat}\ \mathfrak{C})\ \mathfrak{b}\ \mathfrak{a}\ F\ \mathfrak{b}'\ \mathfrak{a}'\ F'$
proof–
interpret ↑: *is-tm-functor* $\alpha\ \langle\Uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ F\rangle\ \mathfrak{C}\ \langle\Uparrow\!\!\rightarrow\!\Uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ F\ \mathfrak{a}'\ \mathfrak{b}'\ F'\rangle$
by (*rule cf-parallel-the-cf-parallel-is-tm-functor*)
show *?thesis*
proof
(
rule cf-eqI[*of* $\alpha\ \langle\Uparrow_C\ \mathfrak{b}\ \mathfrak{a}\ F\rangle\ \langle\mathit{op\text{-}cat}\ \mathfrak{C}\rangle$ - $\langle\Uparrow_C\ \mathfrak{b}\ \mathfrak{a}\ F\rangle\ \langle\mathit{op\text{-}cat}\ \mathfrak{C}\rangle$],

```
    unfold cat-op-simps
  )
  show op-cf (⇑→⇑CF 𝔈 𝔞 𝔟 F 𝔞' 𝔟' F') : ⇑C 𝔟 𝔞 F ↦↦Cα op-cat 𝔈
    by (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-op-intros)
  show ⇑→⇑CF (op-cat 𝔈) 𝔟 𝔞 F 𝔟' 𝔞' F' : ⇑C 𝔟 𝔞 F ↦↦Cα op-cat 𝔈
    by
      (
        cs-concl
          cs-intro: cat-op-intros cat-small-cs-intros cat-parallel-cs-intros
      )
  show
    ⇑→⇑CF 𝔈 𝔞 𝔟 F 𝔞' 𝔟' F'(|ObjMap|) =
      ⇑→⇑CF (op-cat 𝔈) 𝔟 𝔞 F 𝔟' 𝔞' F'(|ObjMap|)
  proof
    (
      rule vsv-eqI;
      (intro cat-parallel-cs-intros)?;
      unfold cat-parallel-cs-simps
    )
    fix a assume a ∈∘ ⇑C 𝔞 𝔟 F(|Obj|)
    then consider ‹a = 𝔞› | ‹a = 𝔟› by (elim the-cat-parallel-ObjE) simp
    then show
      ⇑→⇑CF 𝔈 𝔞 𝔟 F 𝔞' 𝔟' F'(|ObjMap|)(|a|) =
        ⇑→⇑CF (op-cat 𝔈) 𝔟 𝔞 F 𝔟' 𝔞' F'(|ObjMap|)(|a|)
      by cases
        (
          cs-concl
            cs-simp: cat-parallel-cs-simps
            cs-intro: cat-parallel-cs-intros cat-op-intros
        )
  qed (auto simp: the-cat-parallel-components)
  show
    ⇑→⇑CF 𝔈 𝔞 𝔟 F 𝔞' 𝔟' F'(|ArrMap|) =
      ⇑→⇑CF (op-cat 𝔈) 𝔟 𝔞 F 𝔟' 𝔞' F'(|ArrMap|)
  proof
    (
      rule vsv-eqI;
      (intro cat-parallel-cs-intros)?;
      unfold cat-parallel-cs-simps
    )
    fix f assume f ∈∘ ⇑C 𝔞 𝔟 F(|Arr|)
    then consider ‹f = 𝔞› | ‹f = 𝔟› | ‹f ∈∘ F›
      by (elim the-cat-parallel-ArrE) simp
    then show
      ⇑→⇑CF 𝔈 𝔞 𝔟 F 𝔞' 𝔟' F'(|ArrMap|)(|f|) =
        ⇑→⇑CF (op-cat 𝔈) 𝔟 𝔞 F 𝔟' 𝔞' F'(|ArrMap|)(|f|)
      by cases
        (
          cs-concl
            cs-simp: cat-parallel-cs-simps cat-op-simps
            cs-intro: cat-parallel-cs-intros cat-op-intros
        )+
  qed (auto simp: the-cat-parallel-components)
 qed simp-all
qed

lemmas [cat-op-simps] = cf-parallel.cf-parallel-the-cf-parallel-op
```

13.6 Background for the definition of a category with two parallel arrows between two objects

The case of two parallel arrows between two objects is treated explicitly because it is prevalent in applications.

definition $\mathfrak{g}_{PL} :: V$ **where** $\mathfrak{g}_{PL} = 0$
definition $\mathfrak{f}_{PL} :: V$ **where** $\mathfrak{f}_{PL} = 1_{\mathbb{N}}$

definition $\mathfrak{a}_{PL2} :: V$ **where** $\mathfrak{a}_{PL2} = \mathfrak{a}_{PL}\ (set\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\})$
definition $\mathfrak{b}_{PL2} :: V$ **where** $\mathfrak{b}_{PL2} = \mathfrak{b}_{PL}\ (set\ \{\mathfrak{g}_{PL}, \mathfrak{f}_{PL}\})$

lemma *cat-PL2-ineq*:
shows *cat-PL2-*$\mathfrak{ab}$[*cat-parallel-cs-intros*]: $\mathfrak{a}_{PL2} \neq \mathfrak{b}_{PL2}$
and *cat-PL2-*$\mathfrak{ag}$[*cat-parallel-cs-intros*]: $\mathfrak{a}_{PL2} \neq \mathfrak{g}_{PL}$
and *cat-PL2-*$\mathfrak{af}$[*cat-parallel-cs-intros*]: $\mathfrak{a}_{PL2} \neq \mathfrak{f}_{PL}$
and *cat-PL2-*$\mathfrak{bg}$[*cat-parallel-cs-intros*]: $\mathfrak{b}_{PL2} \neq \mathfrak{g}_{PL}$
and *cat-PL2-*$\mathfrak{bf}$[*cat-parallel-cs-intros*]: $\mathfrak{b}_{PL2} \neq \mathfrak{f}_{PL}$
and *cat-PL2-*$\mathfrak{gf}$[*cat-parallel-cs-intros*]: $\mathfrak{g}_{PL} \neq \mathfrak{f}_{PL}$
unfolding $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def* $\mathfrak{g}_{PL}$*-def* $\mathfrak{f}_{PL}$*-def* $\mathfrak{a}_{PL}$*-def* $\mathfrak{b}_{PL}$*-def*
by (*simp-all add*: *Set.doubleton-eq-iff one*)

lemma (**in** $\mathcal{Z}$)
shows *cat-PL2-*$\mathfrak{a}$[*cat-parallel-cs-intros*]: $\mathfrak{a}_{PL2} \in_{\circ} Vset\ \alpha$
and *cat-PL2-*$\mathfrak{b}$[*cat-parallel-cs-intros*]: $\mathfrak{b}_{PL2} \in_{\circ} Vset\ \alpha$
and *cat-PL2-*$\mathfrak{g}$[*cat-parallel-cs-intros*]: $\mathfrak{g}_{PL} \in_{\circ} Vset\ \alpha$
and *cat-PL2-*$\mathfrak{f}$[*cat-parallel-cs-intros*]: $\mathfrak{f}_{PL} \in_{\circ} Vset\ \alpha$
unfolding $\mathfrak{a}_{PL}$*-def* $\mathfrak{b}_{PL}$*-def* $\mathfrak{a}_{PL2}$*-def* $\mathfrak{b}_{PL2}$*-def* $\mathfrak{g}_{PL}$*-def* $\mathfrak{f}_{PL}$*-def*
by (*simp-all add*: *Limit-vdoubleton-in-VsetI*)

13.7 Local assumptions for a category with two parallel arrows between two objects

locale *cat-parallel-2* = $\mathcal{Z}\ \alpha$ **for** α +
fixes $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$
assumes *cat-parallel-2-*$\mathfrak{ab}$[*cat-parallel-cs-intros*]: $\mathfrak{a} \neq \mathfrak{b}$
and *cat-parallel-2-*$\mathfrak{ag}$[*cat-parallel-cs-intros*]: $\mathfrak{a} \neq \mathfrak{g}$
and *cat-parallel-2-*$\mathfrak{af}$[*cat-parallel-cs-intros*]: $\mathfrak{a} \neq \mathfrak{f}$
and *cat-parallel-2-*$\mathfrak{bg}$[*cat-parallel-cs-intros*]: $\mathfrak{b} \neq \mathfrak{g}$
and *cat-parallel-2-*$\mathfrak{bf}$[*cat-parallel-cs-intros*]: $\mathfrak{b} \neq \mathfrak{f}$
and *cat-parallel-2-*$\mathfrak{gf}$[*cat-parallel-cs-intros*]: $\mathfrak{g} \neq \mathfrak{f}$
and *cat-parallel-2-*$\mathfrak{a}$*-in-Vset*[*cat-parallel-cs-intros*]: $\mathfrak{a} \in_{\circ} Vset\ \alpha$
and *cat-parallel-2-*$\mathfrak{b}$*-in-Vset*[*cat-parallel-cs-intros*]: $\mathfrak{b} \in_{\circ} Vset\ \alpha$
and *cat-parallel-2-*$\mathfrak{g}$*-in-Vset*[*cat-parallel-cs-intros*]: $\mathfrak{g} \in_{\circ} Vset\ \alpha$
and *cat-parallel-2-*$\mathfrak{f}$*-in-Vset*[*cat-parallel-cs-intros*]: $\mathfrak{f} \in_{\circ} Vset\ \alpha$

lemmas (**in** *cat-parallel-2*) *cat-parallel-ineq* =
cat-parallel-2-$\mathfrak{ab}$
cat-parallel-2-$\mathfrak{ag}$
cat-parallel-2-$\mathfrak{af}$
cat-parallel-2-$\mathfrak{bg}$
cat-parallel-2-$\mathfrak{bf}$
cat-parallel-2-$\mathfrak{gf}$

Rules.

lemmas (**in** *cat-parallel-2*) [*cat-parallel-cs-intros*] = *cat-parallel-2-axioms*

mk-ide rf *cat-parallel-2-def*[*unfolded cat-parallel-2-axioms-def*]

|*intro cat-parallel-2I*|
|*dest cat-parallel-2D*[*dest*]|
|*elim cat-parallel-2E*[*elim*]|

sublocale *cat-parallel-2* ⊆ *cat-parallel* *α* 𝔞 𝔟 ‹*set* {𝔤, 𝔣}›
by *unfold-locales*
(*simp-all add*: *cat-parallel-cs-intros Limit-vdoubleton-in-VsetI*)

Duality.

lemma (**in** *cat-parallel-2*) *cat-parallel-op*[*cat-op-intros*]:
cat-parallel-2 *α* 𝔟 𝔞 𝔣 𝔤
by (*intro cat-parallel-2I*)
(*auto intro*!: *cat-parallel-cs-intros cat-parallel-ineq*[*symmetric*])

13.8 ↑↑: category with two parallel arrows between two objects

13.8.1 Definition and elementary properties

See Chapter I-2 and Chapter III-3 in [7].

definition *the-cat-parallel-2* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* (‹↑↑$_C$›)
where ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣 = ⇑$_C$ 𝔞 𝔟 (*set* {𝔤, 𝔣})

Components.

lemma *the-cat-parallel-2-components*:
shows ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Obj*|) = *set* {𝔞, 𝔟}
and ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Arr*|) = *set* {𝔞, 𝔟, 𝔤, 𝔣}
unfolding *the-cat-parallel-2-def the-cat-parallel-components* **by** *auto*

Elementary properties.

lemma *the-cat-parallel-2-commute*: ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣 = ↑↑$_C$ 𝔞 𝔟 𝔣 𝔤
unfolding *the-cat-parallel-2-def* **by** (*simp add*: *insert-commute*)

lemma *cat-parallel-is-cat-parallel-2*:
assumes *cat-parallel* *α* 𝔞 𝔟 (*set* {𝔤, 𝔣}) **and** 𝔤 ≠ 𝔣
shows *cat-parallel-2* *α* 𝔞 𝔟 𝔤 𝔣
proof–
interpret *cat-parallel* *α* 𝔞 𝔟 ‹*set* {𝔤, 𝔣}› **by** (*rule assms*(*1*))
show *?thesis*
using *cat-parallel-*𝔞*F cat-parallel-*𝔟*F cat-parallel-F-in-Vset assms*
by (*intro cat-parallel-2I*)
(
auto
dest: *vdoubleton-in-VsetD*
simp: *cat-parallel-*𝔞*-in-Vset cat-parallel-*𝔟*-in-Vset*
)
qed

13.8.2 Objects

lemma *the-cat-parallel-2-Obj-*𝔞*I*[*cat-parallel-cs-intros*]:
assumes *a* = 𝔞
shows *a* ∈$_\circ$ ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Obj*|)
unfolding *the-cat-parallel-2-def*
by (*cs-concl* **cs-simp**: *assms* **cs-intro**: *cat-parallel-cs-intros*)

lemma *the-cat-parallel-2-Obj-*𝔟*I*[*cat-parallel-cs-intros*]:
assumes *a* = 𝔟

shows $a \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Obj|\!)$
unfolding *the-cat-parallel-2-def*
by (*cs-concl* **cs-shallow** **cs-simp**: *assms* **cs-intro**: *cat-parallel-cs-intros*)

lemma *the-cat-parallel-2-ObjE*:
assumes $a \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Obj|\!)$
obtains $a = \mathfrak{a} \mid a = \mathfrak{b}$
using *assms* **unfolding** *the-cat-parallel-2-def* **by** (*elim the-cat-parallel-ObjE*)

13.8.3 Arrows

lemma *the-cat-parallel-2-Arr-*$\mathfrak{a}I$[*cat-parallel-cs-intros*]:
assumes $f = \mathfrak{a}$
shows $f \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-parallel-2-def* **by** (*intro the-cat-parallel-Arr-*$\mathfrak{a}I$)

lemma *the-cat-parallel-2-Arr-*$\mathfrak{b}I$[*cat-parallel-cs-intros*]:
assumes $f = \mathfrak{b}$
shows $f \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Arr|\!)$
using *assms* **unfolding** *the-cat-parallel-2-def* **by** (*intro the-cat-parallel-Arr-*$\mathfrak{b}I$)

lemma *the-cat-parallel-2-Arr-*$\mathfrak{g}I$[*cat-parallel-cs-intros*]:
assumes $f = \mathfrak{g}$
shows $f \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Arr|\!)$
unfolding *assms*(*1*) *the-cat-parallel-2-def*
by (*cs-concl* **cs-simp**: *V-cs-simps* **cs-intro**: *V-cs-intros cat-parallel-cs-intros*)

lemma *the-cat-parallel-2-Arr-*$\mathfrak{f}I$[*cat-parallel-cs-intros*]:
assumes $f = \mathfrak{f}$
shows $f \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Arr|\!)$
unfolding *assms*(*1*) *the-cat-parallel-2-def*
by
(
cs-concl **cs-shallow**
cs-simp: *V-cs-simps* **cs-intro**: *V-cs-intros cat-parallel-cs-intros*
)

lemma *the-cat-parallel-2-ArrE*:
assumes $f \in_{\circ} \uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Arr|\!)$
obtains $f = \mathfrak{a} \mid f = \mathfrak{b} \mid f = \mathfrak{g} \mid f = \mathfrak{f}$
using *assms that*
unfolding *the-cat-parallel-2-def*
by (*auto elim*!: *the-cat-parallel-ArrE*)

13.8.4 Domain

lemma *the-cat-parallel-2-Dom-vsv*[*cat-parallel-cs-intros*]: *vsv* $(\uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Dom|\!))$
unfolding *the-cat-parallel-2-def* **by** (*rule the-cat-parallel-Dom-vsv*)

lemma *the-cat-parallel-2-Dom-vdomain*[*cat-parallel-cs-simps*]:
$\mathcal{D}_{\circ}\ (\uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Dom|\!)) = set\ \{\mathfrak{a}, \mathfrak{b}, \mathfrak{g}, \mathfrak{f}\}$
unfolding *the-cat-parallel-2-def the-cat-parallel-Dom-vdomain* **by** *auto*

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Dom-app-*$\mathfrak{b}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{b}$
shows $\uparrow\uparrow_C \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Dom|\!)(\!|f|\!) = \mathfrak{b}$
using *assms*
unfolding *the-cat-parallel-2-def*

by (*simp add*: *the-cat-parallel-Dom-app-*𝔟)

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Dom-app-*𝔟

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Dom-app-*𝔤[*cat-parallel-cs-simps*]:
assumes f = 𝔤
shows ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Dom*|)(|*f*|) = 𝔞
using *assms*
unfolding *the-cat-parallel-2-def*
by (*intro the-cat-parallel-Dom-app-F*) *simp*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Dom-app-*𝔤

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Dom-app-*𝔣[*cat-parallel-cs-simps*]:
assumes f = 𝔣
shows ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Dom*|)(|*f*|) = 𝔞
using *assms*
unfolding *the-cat-parallel-2-def*
by (*intro the-cat-parallel-Dom-app-F*) *simp*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Dom-app-*𝔣

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Dom-app-*𝔞[*cat-parallel-cs-simps*]:
assumes f = 𝔞
shows ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Dom*|)(|*f*|) = 𝔞
using *assms*
unfolding *the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-Dom-app-*𝔞)

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Dom-app-*𝔞

13.8.5 Codomain

lemma *the-cat-parallel-2-Cod-vsv*[*cat-parallel-cs-intros*]: *vsv* (↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Cod*|))
unfolding *the-cat-parallel-2-def* **by** (*rule the-cat-parallel-Cod-vsv*)

lemma *the-cat-parallel-2-Cod-vdomain*[*cat-parallel-cs-simps*]:
$\mathcal{D}_\circ$ (↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Cod*|)) = *set* {𝔞, 𝔟, 𝔤, 𝔣}
unfolding *the-cat-parallel-2-def the-cat-parallel-Cod-vdomain* **by** *auto*

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Cod-app-*𝔟[*cat-parallel-cs-simps*]:
assumes f = 𝔟
shows ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Cod*|)(|*f*|) = 𝔟
using *assms*
unfolding *the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-Cod-app-*𝔟)

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Cod-app-*𝔟

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Cod-app-*𝔤[*cat-parallel-cs-simps*]:
assumes f = 𝔤
shows ↑↑$_C$ 𝔞 𝔟 𝔤 𝔣(|*Cod*|)(|*f*|) = 𝔟
using *assms*
unfolding *the-cat-parallel-2-def*
by (*intro the-cat-parallel-Cod-app-F*) *simp*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Cod-app-*𝔤

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Cod-app-*𝔣[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{f}$
shows $\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Cod|\!)(\!|f|\!) = \mathfrak{b}$
using *assms*
unfolding *the-cat-parallel-2-def*
by (*intro the-cat-parallel-Cod-app-F*) *simp*

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Cod-app-*𝔣

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Cod-app-*𝔞[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{a}$
shows $\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Cod|\!)(\!|f|\!) = \mathfrak{a}$
using *assms*
unfolding *the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-Cod-app-*𝔞)

lemmas [*cat-parallel-cs-simps*] = *cat-parallel-2.the-cat-parallel-2-Cod-app-*𝔞

13.8.6 Composition

lemma *the-cat-parallel-2-Comp-vsv*[*cat-parallel-cs-intros*]:
vsv ($\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|Comp|\!)$)
unfolding *the-cat-parallel-2-def* **by** (*rule the-cat-parallel-Comp-vsv*)

lemma *the-cat-parallel-2-Comp-app-*𝔟𝔟[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{b}$ **and** $f = \mathfrak{b}$
shows $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = g\ g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = f$
proof–
note *gf* = *the-cat-parallel-Comp-app-*𝔟𝔟[*OF assms*, **where** *F*=‹*set* {𝔤, 𝔣}›]
show $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = g\ g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = f$
unfolding *the-cat-parallel-2-def*
subgoal unfolding *gf*(*1*) **by** *simp*
subgoal unfolding *gf*(*2*) **by** *simp*
done
qed

lemma *the-cat-parallel-2-Comp-app-*𝔞𝔞[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{a}$ **and** $f = \mathfrak{a}$
shows $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = g\ g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = f$
proof–
note *gf* = *the-cat-parallel-Comp-app-*𝔞𝔞[*OF assms*, **where** *F*=‹*set* {𝔤, 𝔣}›]
show $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = g\ g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = f$
unfolding *the-cat-parallel-2-def*
subgoal unfolding *gf*(*1*) **by** *simp*
subgoal unfolding *gf*(*2*) **by** *simp*
done
qed

lemma *the-cat-parallel-2-Comp-app-*𝔟𝔤[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{b}$ **and** $f = \mathfrak{g}$
shows $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = f$
unfolding
the-cat-parallel-2-def assms
*the-cat-parallel-Comp-app-*𝔟*F*[
where *F*=‹*set* {𝔤, 𝔣}›, *OF assms*(*1*), *of* 𝔤, *unfolded assms*, *simplified*
]
by *simp*

lemma *the-cat-parallel-2-Comp-app-𝔟𝔣*[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{b}$ **and** $f = \mathfrak{f}$
shows $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = f$
unfolding
the-cat-parallel-2-def assms
the-cat-parallel-Comp-app-𝔟F[
where *F*=‹*set* {𝔤, 𝔣}›, *OF assms*(*1*), *of* 𝔣, *unfolded assms*, *simplified*
]
by *simp*

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Comp-app-𝔤𝔞*[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{g}$ **and** $f = \mathfrak{a}$
shows $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = g$
unfolding
the-cat-parallel-2-def assms
the-cat-parallel-Comp-app-F𝔞[
of 𝔤, *OF - assms*(*2*), *unfolded assms*, *simplified*
]
by *simp*

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-Comp-app-𝔣𝔞*[*cat-parallel-cs-simps*]:
assumes $g = \mathfrak{f}$ **and** $f = \mathfrak{a}$
shows $g \circ_{A\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} f = g$
unfolding
the-cat-parallel-2-def assms
the-cat-parallel-Comp-app-F𝔞[
of 𝔣, *OF - assms*(*2*), *unfolded assms*, *simplified*
]
by *simp*

13.8.7 Identity

lemma *the-cat-parallel-2-CId-vsv*[*cat-parallel-cs-intros*]: *vsv* $(\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|CId|\!))$
unfolding *the-cat-parallel-2-def* **by** (*rule the-cat-parallel-CId-vsv*)

lemma *the-cat-parallel-2-CId-vdomain*[*cat-parallel-cs-simps*]:
$\mathcal{D}_\circ\ (\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|CId|\!)) = set\ \{\mathfrak{a}, \mathfrak{b}\}$
unfolding *the-cat-parallel-2-def* **by** (*rule the-cat-parallel-CId-vdomain*)

lemma *the-cat-parallel-2-CId-app-𝔞*[*cat-parallel-cs-simps*]:
assumes $a = \mathfrak{a}$
shows $\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|CId|\!)(\!|a|\!) = \mathfrak{a}$
unfolding *assms the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-CId-app-𝔞*)

lemma *the-cat-parallel-2-CId-app-𝔟*[*cat-parallel-cs-simps*]:
assumes $a = \mathfrak{b}$
shows $\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}(\!|CId|\!)(\!|a|\!) = \mathfrak{b}$
unfolding *assms the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-CId-app-𝔟*)

13.8.8 Arrow with a domain and a codomain

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-is-arr-𝔞𝔞𝔞*[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{a}$ **and** $b' = \mathfrak{a}$ **and** $f = \mathfrak{a}$
shows $f : a' \mapsto_{\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} b'$
unfolding *assms the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-is-arr-𝔞𝔞𝔞*)

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-is-arr-𝔟𝔟𝔟*[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{b}$ **and** $b' = \mathfrak{b}$ **and** $f = \mathfrak{b}$
shows $f : a' \mapsto_{\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} b'$
unfolding *assms the-cat-parallel-2-def*
by (*simp add*: *the-cat-parallel-is-arr-𝔟𝔟𝔟*)

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-is-arr-𝔞𝔟𝔤*[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{a}$ **and** $b' = \mathfrak{b}$ **and** $f = \mathfrak{g}$
shows $f : a' \mapsto_{\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} b'$
unfolding *assms the-cat-parallel-2-def*
by
(
rule the-cat-parallel-is-arr-𝔞𝔟F[
OF assms(*1*,*2*), *of* $\mathfrak{g}$, *unfolded assms*, *simplified*
]
)

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-is-arr-𝔞𝔟𝔣*[*cat-parallel-cs-intros*]:
assumes $a' = \mathfrak{a}$ **and** $b' = \mathfrak{b}$ **and** $f = \mathfrak{f}$
shows $f : a' \mapsto_{\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} b'$
unfolding *assms the-cat-parallel-2-def*
by
(
rule the-cat-parallel-is-arr-𝔞𝔟F[
OF assms(*1*,*2*), *of* $\mathfrak{f}$, *unfolded assms*, *simplified*
]
)

lemma (**in** *cat-parallel-2*) *the-cat-parallel-2-is-arrE*:
assumes $f' : a' \mapsto_{\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}} b'$
obtains $a' = \mathfrak{a}$ **and** $b' = \mathfrak{a}$ **and** $f' = \mathfrak{a}$
| $a' = \mathfrak{b}$ **and** $b' = \mathfrak{b}$ **and** $f' = \mathfrak{b}$
| $a' = \mathfrak{a}$ **and** $b' = \mathfrak{b}$ **and** $f' = \mathfrak{g}$
| $a' = \mathfrak{a}$ **and** $b' = \mathfrak{b}$ **and** $f' = \mathfrak{f}$
using *assms*
unfolding *the-cat-parallel-2-def*
by (*elim the-cat-parallel-is-arrE*) *auto*

13.8.9 ↑↑ is a category

lemma (**in** *cat-parallel-2*)
finite-category-the-cat-parallel-2[*cat-parallel-cs-intros*]:
finite-category α $(\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f})$
proof(*intro finite-categoryI''*)
show *tiny-category* α $(\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f})$
unfolding *the-cat-parallel-2-def* **by** (*rule tiny-category-the-cat-parallel*)
qed (*auto simp*: *the-cat-parallel-2-components*)

lemmas [*cat-parallel-cs-intros*] =
cat-parallel-2.finite-category-the-cat-parallel-2

13.8.10 Opposite parallel category

lemma (**in** *cat-parallel-2*) *op-cat-the-cat-parallel-2*[*cat-op-simps*]:
op-cat $(\uparrow\uparrow_C\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}) = \uparrow\uparrow_C\ \mathfrak{b}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{g}$
unfolding *the-cat-parallel-2-def cat-op-simps* **by** (*metis doubleton-eq-iff*)

lemmas [*cat-op-simps*] = *cat-parallel-2.op-cat-the-cat-parallel-2*

13.9 Parallel functor for a category with two parallel arrows between two objects

locale *cf-parallel-2* = *cat-parallel-2* α $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ + *category* α $\mathfrak{C}$
  for α $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$ $\mathfrak{C}$ :: V +
  assumes *cf-parallel-*$\mathfrak{g}'$[*cat-parallel-cs-intros*]: $\mathfrak{g}' : \mathfrak{a}' \mapsto_{\mathfrak{C}} \mathfrak{b}'$
    and *cf-parallel-*$\mathfrak{f}'$[*cat-parallel-cs-intros*]: $\mathfrak{f}' : \mathfrak{a}' \mapsto_{\mathfrak{C}} \mathfrak{b}'$

sublocale *cf-parallel-2* ⊆
  cf-parallel α $\mathfrak{a}$ $\mathfrak{b}$ ‹*set* {$\mathfrak{g}$, $\mathfrak{f}$}› $\mathfrak{a}'$ $\mathfrak{b}'$ ‹$\lambda f \in_{\circ}$ *set* {$\mathfrak{g}$, $\mathfrak{f}$}. (f = $\mathfrak{f}$? $\mathfrak{f}'$: $\mathfrak{g}'$)› $\mathfrak{C}$
  by *unfold-locales* (*auto intro*: *cat-parallel-cs-intros cat-cs-intros*)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-*$\mathfrak{g}''$[*cat-parallel-cs-intros*]:
  assumes a = $\mathfrak{a}'$ **and** b = $\mathfrak{b}'$
  shows $\mathfrak{g}' : a \mapsto_{\mathfrak{C}} b$
  unfolding *assms* **by** (*rule cf-parallel-*$\mathfrak{g}'$)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-*$\mathfrak{g}'''$[*cat-parallel-cs-intros*]:
  assumes g = $\mathfrak{g}'$ **and** b = $\mathfrak{b}'$
  shows $g : \mathfrak{a}' \mapsto_{\mathfrak{C}} b$
  unfolding *assms* **by** (*rule cf-parallel-*$\mathfrak{g}'$)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-*$\mathfrak{g}''''$[*cat-parallel-cs-intros*]:
  assumes g = $\mathfrak{g}'$ **and** a = $\mathfrak{a}'$
  shows $g : a \mapsto_{\mathfrak{C}} \mathfrak{b}'$
  unfolding *assms* **by** (*rule cf-parallel-*$\mathfrak{g}'$)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-*$\mathfrak{f}''$[*cat-parallel-cs-intros*]:
  assumes a = $\mathfrak{a}'$ **and** b = $\mathfrak{b}'$
  shows $\mathfrak{f}' : a \mapsto_{\mathfrak{C}} b$
  unfolding *assms* **by** (*rule cf-parallel-*$\mathfrak{f}'$)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-*$\mathfrak{f}'''$[*cat-parallel-cs-intros*]:
  assumes f = $\mathfrak{f}'$ **and** b = $\mathfrak{b}'$
  shows $f : \mathfrak{a}' \mapsto_{\mathfrak{C}} b$
  unfolding *assms* **by** (*rule cf-parallel-*$\mathfrak{f}'$)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-*$\mathfrak{f}''''$[*cat-parallel-cs-intros*]:
  assumes f = $\mathfrak{f}'$ **and** a = $\mathfrak{a}'$
  shows $f : a \mapsto_{\mathfrak{C}} \mathfrak{b}'$
  unfolding *assms* **by** (*rule cf-parallel-*$\mathfrak{f}'$)

Rules.

lemma (**in** *cf-parallel-2*) *cf-parallel-axioms'*[*cat-parallel-cs-intros*]:
  assumes α' = α
    and a = $\mathfrak{a}$
    and b = $\mathfrak{b}$
    and g = $\mathfrak{g}$
    and f = $\mathfrak{f}$
    and a' = $\mathfrak{a}'$
    and b' = $\mathfrak{b}'$
    and g' = $\mathfrak{g}'$
    and f' = $\mathfrak{f}'$
  shows *cf-parallel-2* α' a b g f a' b' g' f' $\mathfrak{C}$
  unfolding *assms* **by** (*rule cf-parallel-2-axioms*)

mk-ide rf *cf-parallel-2-def*[*unfolded cf-parallel-2-axioms-def*]
|*intro cf-parallel-2I*|
|*dest cf-parallel-2D*[*dest*]|
|*elim cf-parallel-2E*[*elim*]|

lemmas [*cat-parallel-cs-intros*] = *cf-parallelD*(*1*,*2*)

Duality.

lemma (**in** *cf-parallel-2*) *cf-parallel-2-op*[*cat-op-intros*]:
cf-parallel-2 α $\mathfrak{b}$ $\mathfrak{a}$ $\mathfrak{f}$ $\mathfrak{g}$ $\mathfrak{b}'$ $\mathfrak{a}'$ $\mathfrak{f}'$ $\mathfrak{g}'$ (*op-cat* $\mathfrak{C}$)
by (*intro cf-parallel-2I*, *unfold cat-op-simps*)
(*cs-concl* **cs-intro**: *cat-parallel-cs-intros cat-cs-intros cat-op-intros*)

lemmas [*cat-op-intros*] = *cf-parallel.cf-parallel-op*

13.9.1 Definition and elementary properties

definition *the-cf-parallel-2* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
(‹$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}$›)
where $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$ =
$\Uparrow\rightarrow\Uparrow_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ (*set* {$\mathfrak{g}$, $\mathfrak{f}$}) $\mathfrak{a}'$ $\mathfrak{b}'$ ($\lambda f \in_\circ$ *set* {$\mathfrak{g}$, $\mathfrak{f}$}. (f = $\mathfrak{f}$ *?* $\mathfrak{f}'$: $\mathfrak{g}'$))

Components.

lemma *the-cf-parallel-2-components*:
shows [*cat-parallel-cs-simps*]:
$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*HomDom*|) = $\uparrow\uparrow_C$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$
and [*cat-parallel-cs-simps*]:
$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*HomCod*|) = $\mathfrak{C}$
unfolding
the-cf-parallel-2-def the-cat-parallel-2-def the-cf-parallel-components
by *simp-all*

Elementary properties.

lemma (**in** *cf-parallel-2*) *cf-parallel-2-the-cf-parallel-2-commute*:
$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$ = $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{f}$ $\mathfrak{g}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{f}'$ $\mathfrak{g}'$
using *cat-parallel-2-*$\mathfrak{gf}$
unfolding *the-cf-parallel-2-def insert-commute*
by (*force simp*: *VLambda-vdoubleton*)

lemma *cf-parallel-is-cf-parallel-2*:
assumes
cf-parallel α $\mathfrak{a}$ $\mathfrak{b}$ (*set* {$\mathfrak{g}$, $\mathfrak{f}$}) $\mathfrak{a}'$ $\mathfrak{b}'$ ($\lambda f \in_\circ$ *set* {$\mathfrak{g}$, $\mathfrak{f}$}. (f = $\mathfrak{f}$ *?* $\mathfrak{f}'$: $\mathfrak{g}'$)) $\mathfrak{C}$
and $\mathfrak{g} \neq \mathfrak{f}$
shows *cf-parallel-2* α $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$ $\mathfrak{C}$
proof–
interpret
cf-parallel α $\mathfrak{a}$ $\mathfrak{b}$ ‹*set* {$\mathfrak{g}$, $\mathfrak{f}$}› $\mathfrak{a}'$ $\mathfrak{b}'$ ‹$\lambda f \in_\circ$ *set* {$\mathfrak{g}$, $\mathfrak{f}$}. (f = $\mathfrak{f}$ *?* $\mathfrak{f}'$: $\mathfrak{g}'$)› $\mathfrak{C}$
by (*rule assms*(*1*))
have $\mathfrak{gf}$: $\mathfrak{g} \in_\circ$ *set* {$\mathfrak{g}$, $\mathfrak{f}$} $\mathfrak{f} \in_\circ$ *set* {$\mathfrak{g}$, $\mathfrak{f}$} **by** *auto*
show *?thesis*
using *cat-parallel-axioms assms*(*2*) *category-axioms* $\mathfrak{gf}$[*THEN cf-parallel-F'*]
by (*intro cf-parallel-2I cat-parallel-is-cat-parallel-2*)
(*auto simp*: *assms*(*2*))
qed

13.9.2 Object map

lemma *the-cf-parallel-2-ObjMap-vsv*[*cat-parallel-cs-intros*]:

vsv (↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ObjMap*|))
unfolding *the-cf-parallel-2-def* **by** (*intro cat-parallel-cs-intros*)

lemma *the-cf-parallel-2-ObjMap-vdomain*[*cat-parallel-cs-simps*]:
$\mathcal{D}_\circ$ (↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ObjMap*|)) = ↑↑$_C$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$(|*Obj*|)
unfolding *the-cf-parallel-2-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-parallel-cs-simps the-cat-parallel-2-def*)

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ObjMap-app-*$\mathfrak{a}$[*cat-parallel-cs-simps*]:
assumes $x = \mathfrak{a}$
shows ↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ObjMap*|)(|x|) = $\mathfrak{a}'$
unfolding *the-cf-parallel-2-def*
by (*cs-concl* **cs-simp**: *assms cat-parallel-cs-simps*)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel-2.the-cf-parallel-2-ObjMap-app-*$\mathfrak{a}$

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ObjMap-app-*$\mathfrak{b}$[*cat-parallel-cs-simps*]:
assumes $x = \mathfrak{b}$
shows ↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ObjMap*|)(|x|) = $\mathfrak{b}'$
unfolding *the-cf-parallel-2-def*
by (*cs-concl* **cs-shallow cs-simp**: *assms cat-parallel-cs-simps*)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel-2.the-cf-parallel-2-ObjMap-app-*$\mathfrak{b}$

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ObjMap-vrange*:
$\mathcal{R}_\circ$ (↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ObjMap*|)) = *set* {$\mathfrak{a}'$, $\mathfrak{b}'$}
unfolding *the-cf-parallel-2-def* **by** (*rule the-cf-parallel-ObjMap-vrange*)

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ObjMap-vrange-vsubset-Obj*:
$\mathcal{R}_\circ$ (↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ObjMap*|)) ⊆$_\circ$ $\mathfrak{C}$(|*Obj*|)
unfolding *the-cf-parallel-2-def*
by (*rule the-cf-parallel-ObjMap-vrange-vsubset-Obj*)

13.9.3 Arrow map

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ArrMap-app-*$\mathfrak{g}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{g}$
shows ↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ArrMap*|)(|f|) = $\mathfrak{g}'$
unfolding *the-cf-parallel-2-def assms*
by
(
cs-concl
cs-simp: *V-cs-simps cat-parallel-cs-simps*
cs-intro: *V-cs-intros cat-parallel-cs-intros*
)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel-2.the-cf-parallel-2-ArrMap-app-*$\mathfrak{g}$

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ArrMap-app-*$\mathfrak{f}$[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{f}$
shows ↑↑→↑↑$_{CF}$ $\mathfrak{C}$ $\mathfrak{a}$ $\mathfrak{b}$ $\mathfrak{g}$ $\mathfrak{f}$ $\mathfrak{a}'$ $\mathfrak{b}'$ $\mathfrak{g}'$ $\mathfrak{f}'$(|*ArrMap*|)(|f|) = $\mathfrak{f}'$
unfolding *the-cf-parallel-2-def assms*
by
(
cs-concl
cs-simp: *V-cs-simps cat-parallel-cs-simps*
cs-intro: *V-cs-intros cat-parallel-cs-intros*
)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel-2.the-cf-parallel-2-ArrMap-app-𝔣*

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ArrMap-app-𝔞*[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{a}$
shows $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}'(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!)$
unfolding *the-cf-parallel-2-def assms*
by (*cs-concl* **cs-simp**: *cat-parallel-cs-simps*)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel-2.the-cf-parallel-2-ArrMap-app-𝔞*

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ArrMap-app-𝔟*[*cat-parallel-cs-simps*]:
assumes $f = \mathfrak{b}$
shows $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}'(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!)$
unfolding *the-cf-parallel-2-def assms*
by (*cs-concl* **cs-shallow cs-simp**: *cat-parallel-cs-simps*)

lemmas [*cat-parallel-cs-simps*] = *cf-parallel-2.the-cf-parallel-2-ArrMap-app-𝔟*

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ArrMap-vrange*:
$\mathcal{R}_{\circ}\ (\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}'(\!|ArrMap|\!)) = set\ \{\mathfrak{C}(\!|CId|\!)(\!|\mathfrak{a}'|\!),\ \mathfrak{C}(\!|CId|\!)(\!|\mathfrak{b}'|\!),\ \mathfrak{f}',\ \mathfrak{g}'\}$
unfolding *the-cf-parallel-2-def the-cf-parallel-ArrMap-vrange*
using *cat-parallel-2-𝔤𝔣*
by (*auto simp*: *app-vimage-iff VLambda-vdoubleton*)

lemma (**in** *cf-parallel-2*) *the-cf-parallel-2-ArrMap-vrange-vsubset-Arr*:
$\mathcal{R}_{\circ}\ (\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}'(\!|ArrMap|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!)$
unfolding *the-cf-parallel-2-def*
by (*rule the-cf-parallel-ArrMap-vrange-vsubset-Arr*)

13.9.4 Parallel functor for a category with two parallel arrows between two objects is a functor

lemma (**in** *cf-parallel-2*) *cf-parallel-2-the-cf-parallel-2-is-tm-functor*:
$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}' : \uparrow\uparrow_{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f} \mapsto\mapsto_{C.tm\alpha}\ \mathfrak{C}$
unfolding *the-cf-parallel-2-def the-cat-parallel-2-def*
by (*rule cf-parallel-the-cf-parallel-is-tm-functor*)

lemma (**in** *cf-parallel-2*) *cf-parallel-2-the-cf-parallel-2-is-tm-functor′*:
assumes $\mathfrak{A}' = \uparrow\uparrow_{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}' : \mathfrak{A}' \mapsto\mapsto_{C.tm\alpha}\ \mathfrak{C}'$
unfolding *assms* **by** (*rule cf-parallel-2-the-cf-parallel-2-is-tm-functor*)

lemmas [*cat-parallel-cs-intros*] =
cf-parallel-2.cf-parallel-2-the-cf-parallel-2-is-tm-functor′

13.9.5 Opposite parallel functor for a category with two parallel arrows between two objects

lemma (**in** *cf-parallel-2*) *cf-parallel-2-the-cf-parallel-2-op*[*cat-op-simps*]:
op-cf $(\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ \mathfrak{C}\ \mathfrak{a}\ \mathfrak{b}\ \mathfrak{g}\ \mathfrak{f}\ \mathfrak{a}'\ \mathfrak{b}'\ \mathfrak{g}'\ \mathfrak{f}') =$
$\uparrow\uparrow\rightarrow\uparrow\uparrow_{CF}\ (op\text{-}cat\ \mathfrak{C})\ \mathfrak{b}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{g}\ \mathfrak{b}'\ \mathfrak{a}'\ \mathfrak{f}'\ \mathfrak{g}'$
using *cat-parallel-2-𝔤𝔣*
unfolding *the-cf-parallel-2-def cf-parallel-the-cf-parallel-op*
by (*auto simp*: *VLambda-vdoubleton insert-commute*)

lemmas [*cat-op-simps*] = *cf-parallel-2.cf-parallel-2-the-cf-parallel-2-op*

14 Comma categories

14.1 Background

named-theorems *cat-comma-cs-simps*
named-theorems *cat-comma-cs-intros*

14.2 Comma category

14.2.1 Definition and elementary properties

See Exercise 1.3.vi in [12] or Chapter II-6 in [7].

definition *cat-comma-Obj* :: $V \Rightarrow V \Rightarrow V$
  where *cat-comma-Obj* $\mathfrak{G}\ \mathfrak{H} \equiv$ *set*
    {
      $[a,\ b,\ f]_\circ \mid a\ b\ f.$
        $a \in_\circ \mathfrak{G}(\!|HomDom|\!)(\!|Obj|\!) \wedge$
        $b \in_\circ \mathfrak{H}(\!|HomDom|\!)(\!|Obj|\!) \wedge$
        $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{G}(\!|HomCod|\!)} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
    }

lemma *small-cat-comma-Obj*[*simp*]:
  small
    {
      $[a,\ b,\ f]_\circ \mid a\ b\ f.$
        $a \in_\circ \mathfrak{A}(\!|Obj|\!) \wedge b \in_\circ \mathfrak{B}(\!|Obj|\!) \wedge f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
    }
  (**is** ‹*small ?abfs*›)
proof–
  define Q **where**
    $Q\ i = (\textit{if } i = 0 \textit{ then } \mathfrak{A}(\!|Obj|\!) \textit{ else if } i = 1_{\mathbb{N}} \textit{ then } \mathfrak{B}(\!|Obj|\!) \textit{ else } \mathfrak{C}(\!|Arr|\!))$
    for i
  have *?abfs* $\subseteq$ *elts* $(\prod_\circ i \in_\circ$ *set* $\{0,\ 1_{\mathbb{N}},\ 2_{\mathbb{N}}\}.\ Q\ i)$
    unfolding *Q-def*
  proof
    (
      intro subsetI,
      unfold mem-Collect-eq,
      elim exE conjE,
      intro vproductI;
      simp only:
)
    fix $a\ b\ f$ **show** $\mathcal{D}_\circ\ [a,\ b,\ f]_\circ =$ *set* $\{0,\ 1_{\mathbb{N}},\ 2_{\mathbb{N}}\}$
      by (*simp add*: *three nat-omega-simps*)
  qed (*force simp*: *nat-omega-simps*)+
  then show *small ?abfs* **by** (*rule down*)
qed

definition *cat-comma-Hom* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B \equiv$ *set*
    {
      $[A,\ B,\ [g,\ h]_\circ]_\circ \mid g\ h.$
        $A \in_\circ$ *cat-comma-Obj* $\mathfrak{G}\ \mathfrak{H} \wedge$
        $B \in_\circ$ *cat-comma-Obj* $\mathfrak{G}\ \mathfrak{H} \wedge$
        $g : A(\!|0|\!) \mapsto_{\mathfrak{G}(\!|HomDom|\!)} B(\!|0|\!) \wedge$
        $h : A(\!|1_{\mathbb{N}}|\!) \mapsto_{\mathfrak{H}(\!|HomDom|\!)} B(\!|1_{\mathbb{N}}|\!) \wedge$
        $B(\!|2_{\mathbb{N}}|\!) \circ_{A\mathfrak{G}(\!|HomCod|\!)} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) =$
        $\mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{G}(\!|HomCod|\!)} A(\!|2_{\mathbb{N}}|\!)$

```
  }

lemma small-cat-comma-Hom[simp]: small
  {
    [A, B, [g, h]∘]∘ | g h.
      A ∈∘ cat-comma-Obj 𝔊 ℌ ∧
      B ∈∘ cat-comma-Obj 𝔊 ℌ ∧
      g : A(|0|) ↦𝔄 B(|0|) ∧
      h : A(|1ℕ|) ↦𝔅 B(|1ℕ|) ∧
      B(|2ℕ|) ∘A𝔈 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘A𝔈 A(|2ℕ|)
  }
  (is ‹small ?abf-a'b'f'-gh›)
proof–
  define Q where
    Q i =
      (
        if i = 0
        then cat-comma-Obj 𝔊 ℌ
        else if i = 1ℕ then cat-comma-Obj 𝔊 ℌ else 𝔄(|Arr|) ×• 𝔅(|Arr|)
      )
    for i
  have ?abf-a'b'f'-gh ⊆ elts (∏∘i∈∘ set {0, 1ℕ, 2ℕ}. Q i)
    unfolding Q-def
  proof
    (
      intro subsetI,
      unfold mem-Collect-eq,
      elim exE conjE,
      intro vproductI;
      simp only:
    )
    fix a b f show 𝒟∘ [a, b, f]∘ = ZFC-in-HOL.set {0, 1ℕ, 2ℕ}
      by (simp add: three nat-omega-simps)
  qed (force simp : nat-omega-simps)+
  then show small ?abf-a'b'f'-gh by (rule down)
qed

definition cat-comma-Arr :: V ⇒ V ⇒ V
  where cat-comma-Arr 𝔊 ℌ ≡
    (
      ⋃∘A∈∘cat-comma-Obj 𝔊 ℌ. ⋃∘B∈∘cat-comma-Obj 𝔊 ℌ.
        cat-comma-Hom 𝔊 ℌ A B
    )

definition cat-comma-composable :: V ⇒ V ⇒ V
  where cat-comma-composable 𝔊 ℌ ≡ set
    {
      [[B, C, G]∘, [A, B, F]∘]∘ | A B C G F.
        [B, C, G]∘ ∈∘ cat-comma-Arr 𝔊 ℌ ∧ [A, B, F]∘ ∈∘ cat-comma-Arr 𝔊 ℌ
    }

lemma small-cat-comma-composable[simp]:
  shows small
    {
      [[B, C, G]∘, [A, B, F]∘]∘ | A B C G F.
        [B, C, G]∘ ∈∘ cat-comma-Arr 𝔊 ℌ ∧ [A, B, F]∘ ∈∘ cat-comma-Arr 𝔊 ℌ
    }
  (is ‹small ?S›)
```

proof(*rule down*)
  show *?S* ⊆ *elts* (*cat-comma-Arr* 𝔊 ℌ ×$_\bullet$ *cat-comma-Arr* 𝔊 ℌ) **by** *auto*
qed

definition *cat-comma* :: *V* ⇒ *V* ⇒ *V* (‹(- $_{CF}$↓$_{CF}$ -)› [*1000*, *1000*] *999*)
  where 𝔊 $_{CF}$↓$_{CF}$ ℌ =
    [
      cat-comma-Obj 𝔊 ℌ,
      cat-comma-Arr 𝔊 ℌ,
      (λ*F*∈$_\circ$*cat-comma-Arr* 𝔊 ℌ. *F*(|*0*|)),
      (λ*F*∈$_\circ$*cat-comma-Arr* 𝔊 ℌ. *F*(|$1_{\mathbb{N}}$|)),
      (
        λ*GF*∈$_\circ$*cat-comma-composable* 𝔊 ℌ.
          [
            GF(|$1_{\mathbb{N}}$|)(|*0*|),
            GF(|*0*|)(|$1_{\mathbb{N}}$|),
            [
              GF(|*0*|)(|$2_{\mathbb{N}}$|)(|*0*|) ∘$_{A\,\mathfrak{G}(|HomDom|)}$ *GF*(|$1_{\mathbb{N}}$|)(|$2_{\mathbb{N}}$|)(|*0*|),
              GF(|*0*|)(|$2_{\mathbb{N}}$|)(|$1_{\mathbb{N}}$|) ∘$_{A\,\mathfrak{H}(|HomDom|)}$ *GF*(|$1_{\mathbb{N}}$|)(|$2_{\mathbb{N}}$|)(|$1_{\mathbb{N}}$|)
            ]$_\circ$
          ]$_\circ$
      ),
      (
        λ*A*∈$_\circ$*cat-comma-Obj* 𝔊 ℌ.
          [*A*, *A*, [𝔊(|*HomDom*|)(|*CId*|)(|*A*(|*0*|)|), ℌ(|*HomDom*|)(|*CId*|)(|*A*(|$1_{\mathbb{N}}$|)|)]$_\circ$]$_\circ$
)
    ]$_\circ$

Components.

lemma *cat-comma-components*:
  shows 𝔊 $_{CF}$↓$_{CF}$ ℌ(|*Obj*|) = *cat-comma-Obj* 𝔊 ℌ
    and 𝔊 $_{CF}$↓$_{CF}$ ℌ(|*Arr*|) = *cat-comma-Arr* 𝔊 ℌ
    and 𝔊 $_{CF}$↓$_{CF}$ ℌ(|*Dom*|) = (λ*F*∈$_\circ$*cat-comma-Arr* 𝔊 ℌ. *F*(|*0*|))
    and 𝔊 $_{CF}$↓$_{CF}$ ℌ(|*Cod*|) = (λ*F*∈$_\circ$*cat-comma-Arr* 𝔊 ℌ. *F*(|$1_{\mathbb{N}}$|))
    and 𝔊 $_{CF}$↓$_{CF}$ ℌ(|*Comp*|) =
      (
        λ*GF*∈$_\circ$*cat-comma-composable* 𝔊 ℌ.
          [
            GF(|$1_{\mathbb{N}}$|)(|*0*|),
            GF(|*0*|)(|$1_{\mathbb{N}}$|),
            [
              GF(|*0*|)(|$2_{\mathbb{N}}$|)(|*0*|) ∘$_{A\,\mathfrak{G}(|HomDom|)}$ *GF*(|$1_{\mathbb{N}}$|)(|$2_{\mathbb{N}}$|)(|*0*|),
              GF(|*0*|)(|$2_{\mathbb{N}}$|)(|$1_{\mathbb{N}}$|) ∘$_{A\,\mathfrak{H}(|HomDom|)}$ *GF*(|$1_{\mathbb{N}}$|)(|$2_{\mathbb{N}}$|)(|$1_{\mathbb{N}}$|)
            ]$_\circ$
          ]$_\circ$
)
    and 𝔊 $_{CF}$↓$_{CF}$ ℌ(|*CId*|) =
      (
        λ*A*∈$_\circ$*cat-comma-Obj* 𝔊 ℌ.
          [*A*, *A*, [𝔊(|*HomDom*|)(|*CId*|)(|*A*(|*0*|)|), ℌ(|*HomDom*|)(|*CId*|)(|*A*(|$1_{\mathbb{N}}$|)|)]$_\circ$]$_\circ$
)
  unfolding *cat-comma-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
  fixes α 𝔄 𝔅 ℭ 𝔊 ℌ
  assumes 𝔊: 𝔊 : 𝔄 ↦↦$_{C\alpha}$ ℭ
    and ℌ: ℌ : 𝔅 ↦↦$_{C\alpha}$ ℭ

begin

interpretation $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)
interpretation $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule* $\mathfrak{H}$)

lemma *cat-comma-Obj-def'*:
cat-comma-Obj $\mathfrak{G}$ $\mathfrak{H}$ ≡ *set*
{
$[a, b, f]_\circ$ | a b f.
$a \in_\circ \mathfrak{A}(\!|Obj|\!) \wedge b \in_\circ \mathfrak{B}(\!|Obj|\!) \wedge f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
}
unfolding *cat-comma-Obj-def cat-cs-simps* **by** *simp*

lemma *cat-comma-Hom-def'*:
cat-comma-Hom $\mathfrak{G}$ $\mathfrak{H}$ A B ≡ *set*
{
$[A, B, [g, h]_\circ]_\circ$ | g h.
$A \in_\circ$ *cat-comma-Obj* $\mathfrak{G}$ $\mathfrak{H}$ ∧
$B \in_\circ$ *cat-comma-Obj* $\mathfrak{G}$ $\mathfrak{H}$ ∧
$g : A(\!|0|\!) \mapsto_{\mathfrak{A}} B(\!|0|\!)$ ∧
$h : A(\!|1_{\mathbb{N}}|\!) \mapsto_{\mathfrak{B}} B(\!|1_{\mathbb{N}}|\!)$ ∧
$B(\!|2_{\mathbb{N}}|\!) \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} A(\!|2_{\mathbb{N}}|\!)$
}
unfolding *cat-comma-Hom-def cat-cs-simps* **by** *simp*

lemma *cat-comma-components'*:
shows $\mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|Obj|\!)$ = *cat-comma-Obj* $\mathfrak{G}$ $\mathfrak{H}$
and $\mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|Arr|\!)$ = *cat-comma-Arr* $\mathfrak{G}$ $\mathfrak{H}$
and $\mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|Dom|\!)$ = $(\lambda F{\in_\circ}$*cat-comma-Arr* $\mathfrak{G}$ $\mathfrak{H}$. $F(\!|0|\!))$
and $\mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|Cod|\!)$ = $(\lambda F{\in_\circ}$*cat-comma-Arr* $\mathfrak{G}$ $\mathfrak{H}$. $F(\!|1_{\mathbb{N}}|\!))$
and $\mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|Comp|\!)$ =
(
$\lambda GF{\in_\circ}$*cat-comma-composable* $\mathfrak{G}$ $\mathfrak{H}$.
[
$GF(\!|1_{\mathbb{N}}|\!)(\!|0|\!)$,
$GF(\!|0|\!)(\!|1_{\mathbb{N}}|\!)$,
[
$GF(\!|0|\!)(\!|2_{\mathbb{N}}|\!)(\!|0|\!) \circ_{A\mathfrak{A}} GF(\!|1_{\mathbb{N}}|\!)(\!|2_{\mathbb{N}}|\!)(\!|0|\!)$,
$GF(\!|0|\!)(\!|2_{\mathbb{N}}|\!)(\!|1_{\mathbb{N}}|\!) \circ_{A\mathfrak{B}} GF(\!|1_{\mathbb{N}}|\!)(\!|2_{\mathbb{N}}|\!)(\!|1_{\mathbb{N}}|\!)$
]$_\circ$
]$_\circ$
)
and $\mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|CId|\!)$ =
$(\lambda A{\in_\circ}$*cat-comma-Obj* $\mathfrak{G}$ $\mathfrak{H}$. $[A, A, [\mathfrak{A}(\!|CId|\!)(\!|A(\!|0|\!)|\!), \mathfrak{B}(\!|CId|\!)(\!|A(\!|1_{\mathbb{N}}|\!)|\!)]_\circ]_\circ)$
unfolding *cat-comma-components cat-cs-simps* **by** *simp-all*

end

14.2.2 Objects

lemma *cat-comma-ObjI*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $A = [a, b, f]_\circ$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
shows $A \in_\circ \mathfrak{G}$ $_{CF}{\downarrow}_{CF}$ $\mathfrak{H}(\!|Obj|\!)$

using *assms*(*4–6*)
unfolding *cat-comma-Obj-def'*[*OF assms*(*1,2*)] *assms*(*3*) *cat-comma-components*
by *simp*

lemma *cat-comma-ObjD*[*dest*]:
assumes $[a,\ b,\ f]_\circ \in_\circ \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!)$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
using *assms*
unfolding
cat-comma-components'[*OF assms*(*2,3*)] *cat-comma-Obj-def'*[*OF assms*(*2,3*)]
by *auto*

lemma *cat-comma-ObjE*[*elim*]:
assumes $A \in_\circ \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!)$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains $a\ b\ f$ **where** $A = [a,\ b,\ f]_\circ$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
using *assms*
unfolding
cat-comma-components'[*OF assms*(*2,3*)] *cat-comma-Obj-def'*[*OF assms*(*2,3*)]
by *auto*

14.2.3 Arrows

lemma *cat-comma-HomI*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $F = [A,\ B,\ [g,\ h]_\circ]_\circ$
and $A = [a,\ b,\ f]_\circ$
and $B = [a',\ b',\ f']_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
shows $F \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
using *assms*(*1,2,6–10*)
unfolding *cat-comma-Hom-def'*[*OF assms*(*1,2*)] *assms*(*3–5*)
by
(
intro in-set-CollectI exI conjI small-cat-comma-Hom,
unfold cat-comma-components'(*1,2*)[*OF assms*(*1,2*), *symmetric*],
(
cs-concl
cs-simp: *cat-comma-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)+
)
(*clarsimp simp*: *nat-omega-simps*)+

lemma *cat-comma-HomE*[*elim*]:

assumes $F \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
obtains $a\ b\ f\ a'\ b'\ f'\ g\ h$
where $F = [A,\ B,\ [g,\ h]_\circ]_\circ$
and $A = [a,\ b,\ f]_\circ$
and $B = [a',\ b',\ f']_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
using *assms*(*1*)
by
(
unfold
cat-comma-components'[*OF assms*(*2,3*)] *cat-comma-Hom-def'*[*OF assms*(*2,3*)],
elim in-small-setE;
(*unfold mem-Collect-eq*, *elim exE conjE cat-comma-ObjE*[*OF* - *assms*(*2,3*)])*?*,
insert that,
all‹
(*unfold cat-comma-components'*(*1,2*)[*OF assms*(*2,3*), *symmetric*],
elim cat-comma-ObjE[*OF* - *assms*(*2,3*)]) | –
›
)
(*auto simp*: *nat-omega-simps*)

lemma *cat-comma-HomD*[*dest*]:
assumes $[[a,\ b,\ f]_\circ,\ [a',\ b',\ f']_\circ,\ [g,\ h]_\circ]_\circ \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $g : a \mapsto_{\mathfrak{A}} a'$
and $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
using *assms*(*1*) **by** (*force elim*!: *cat-comma-HomE*[*OF* - *assms*(*2,3*)])+

lemma *cat-comma-ArrI*[*cat-comma-cs-intros*]:
assumes $F \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and $A \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
and $B \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
shows $F \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Arr|\!)$
using *assms*
unfolding *cat-comma-components cat-comma-Arr-def*
by (*intro vifunionI*)

lemma *cat-comma-ArrE*[*elim*]:
assumes $F \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Arr|\!)$
obtains $A\ B$
where $F \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and $A \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
and $B \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
using *assms* **unfolding** *cat-comma-components cat-comma-Arr-def* **by** *auto*

lemma *cat-comma-ArrD*[*dest*]:
assumes $[A,\ B,\ F]_\circ \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Arr|\!)$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $[A,\ B,\ F]_\circ \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and $A \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
and $B \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
proof–
from *assms* **obtain** $C\ D$
where $[A,\ B,\ F]_\circ \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ C\ D$
and $C \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
and $D \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
by (*elim cat-comma-ArrE*)
moreover from *cat-comma-HomE*[*OF this*(*1*) *assms*(*2,3*)] **have** $A = C$ **and** $B = D$
by *auto*
ultimately show $[A,\ B,\ F]_\circ \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and $A \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
and $B \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
by *auto*
qed

14.2.4 Domain

lemma *cat-comma-Dom-vsv*[*cat-comma-cs-intros*]: *vsv* $(\mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Dom|\!))$
unfolding *cat-comma-components* **by** *simp*

lemma *cat-comma-Dom-vdomain*[*cat-comma-cs-simps*]:
$\mathcal{D}_\circ\ (\mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Dom|\!)) = \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
unfolding *cat-comma-components* **by** *simp*

lemma *cat-comma-Dom-app*[*cat-comma-cs-simps*]:
assumes $ABF = [A,\ B,\ F]_\circ$ **and** $ABF \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
shows $\mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Dom|\!)(\!|ABF|\!) = A$
using *assms*(*2*) **unfolding** *assms*(*1*) *cat-comma-components* **by** *simp*

lemma *cat-comma-Dom-vrange*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ\ (\mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Dom|\!)) \subseteq_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*)
fix ABF **assume** $ABF \in_\circ \mathcal{D}_\circ\ (\mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Dom|\!))$
then have $ABF \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
by (*cs-prems* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
then obtain $A\ B$
where ABF: $ABF \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and A: $A \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
and B: $B \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
by *auto*
from *this*(*1*) **obtain** $a\ b\ f\ a'\ b'\ f'\ g\ h$
where $ABF = [A,\ B,\ [g,\ h]_\circ]_\circ$
and $A = [a,\ b,\ f]_\circ$
and $B = [a',\ b',\ f']_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by (*elim cat-comma-HomE*[*OF* - *assms*(*1,2*)])
from *ABF this A B* **show** $\mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Dom|\!)(\!|ABF|\!) \in_\circ \mathfrak{G}\ _{CF\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
by
(
cs-concl **cs-shallow**

cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)
qed (*auto intro*: *cat-comma-cs-intros*)

14.2.5 Codomain

lemma *cat-comma-Cod-vsv*[*cat-comma-cs-intros*]: *vsv* ($\mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Cod|\!)$)
unfolding *cat-comma-components* **by** *simp*

lemma *cat-comma-Cod-vdomain*[*cat-comma-cs-simps*]:
$\mathcal{D}_\circ(\mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Cod|\!)) = \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Arr|\!)$
unfolding *cat-comma-components* **by** *simp*

lemma *cat-comma-Cod-app*[*cat-comma-cs-simps*]:
assumes $ABF = [A, B, F]_\circ$ **and** $ABF \in_\circ \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Arr|\!)$
shows $\mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Cod|\!)(\!|ABF|\!) = B$
using *assms*(*2*)
unfolding *assms*(*1*) *cat-comma-components*
by (*simp add*: *nat-omega-simps*)

lemma *cat-comma-Cod-vrange*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ(\mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Cod|\!)) \subseteq_\circ \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*)
fix *ABF* **assume** $ABF \in_\circ \mathcal{D}_\circ(\mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Cod|\!))$
then have $ABF \in_\circ \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Arr|\!)$
by (*cs-prems* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
then obtain *A B*
where *F*: $ABF \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
and *A*: $A \in_\circ \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Obj|\!)$
and *B*: $B \in_\circ \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Obj|\!)$
by *auto*
from *this*(*1*) **obtain** *a b f a′ b′ f′ g h*
where $ABF = [A, B, [g, h]_\circ]_\circ$
and $A = [a, b, f]_\circ$
and $B = [a', b', f']_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by (*elim cat-comma-HomE*[*OF - assms*(*1,2*)])
from *F this A B* **show** $\mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Cod|\!)(\!|ABF|\!) \in_\circ \mathfrak{G}_{CF\downarrow_{CF}}\mathfrak{H}(\!|Obj|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)
qed (*auto intro*: *cat-comma-cs-intros*)

14.2.6 Arrow with a domain and a codomain

lemma *cat-comma-is-arrI*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $ABF = [A, B, F]_\circ$
and $A = [a, b, f]_\circ$
and $B = [a', b', f']_\circ$

```
    and F = [g, h]_∘
    and g : a ↦_𝔄 a'
    and h : b ↦_𝔅 b'
    and f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
    and f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
    and f' ∘_Aℭ 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_Aℭ f
  shows ABF : A ↦_𝔊 CF↓CF ℌ B
proof(intro is-arrI)
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  from assms(7–11) show ABF ∈_∘ 𝔊 CF↓CF ℌ(|Arr|)
    unfolding assms(3–6)
    by
      (
        cs-concl
          cs-simp: cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  with assms(7–11) show 𝔊 CF↓CF ℌ(|Dom|)(|ABF|) = A 𝔊 CF↓CF ℌ(|Cod|)(|ABF|) = B
    unfolding assms(3–6) by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps)+
qed

lemma cat-comma-is-arrD[dest]:
  assumes [[a, b, f]_∘, [a', b', f']_∘, [g, h]_∘]_∘ :
    [a, b, f]_∘ ↦_𝔊 CF↓CF ℌ [a', b', f']_∘
    and 𝔊 : 𝔄 ↦↦_Cα ℭ
    and ℌ : 𝔅 ↦↦_Cα ℭ
  shows g : a ↦_𝔄 a'
    and h : b ↦_𝔅 b'
    and f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
    and f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
    and f' ∘_Aℭ 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_Aℭ f
proof–
  note F-is-arrD = is-arrD[OF assms(1)]
  note F-cat-comma-ArrD = cat-comma-ArrD[OF F-is-arrD(1) assms(2,3)]
  show g : a ↦_𝔄 a'
    and h : b ↦_𝔅 b'
    and f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
    and f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
    and f' ∘_Aℭ 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_Aℭ f
    by (intro cat-comma-HomD[OF F-cat-comma-ArrD(1) assms(2,3)])+
qed

lemma cat-comma-is-arrE[elim]:
  assumes ABF : A ↦_𝔊 CF↓CF ℌ B
    and 𝔊 : 𝔄 ↦↦_Cα ℭ
    and ℌ : 𝔅 ↦↦_Cα ℭ
  obtains a b f a' b' f' g h
    where ABF = [[a, b, f]_∘, [a', b', f']_∘, [g, h]_∘]_∘
      and A = [a, b, f]_∘
      and B = [a', b', f']_∘
      and g : a ↦_𝔄 a'
      and h : b ↦_𝔅 b'
      and f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
      and f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
      and f' ∘_Aℭ 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_Aℭ f
proof–
  note F-is-arrD = is-arrD[OF assms(1)]
```

from *F-is-arrD*(*1*) **obtain** *C D*
where *ABF* $\in_\circ$ *cat-comma-Hom* $\mathfrak{G}$ $\mathfrak{H}$ *C D*
and *C* $\in_\circ$ $\mathfrak{G}$ $_{CF\downarrow_{CF}}$ $\mathfrak{H}(\!|Obj|\!)$
and *D* $\in_\circ$ $\mathfrak{G}$ $_{CF\downarrow_{CF}}$ $\mathfrak{H}(\!|Obj|\!)$
by *auto*
from *this*(*1*) **obtain** *a b f a′ b′ f′ g h*
where *F-def*: *ABF* = [*C*, *D*, [*g*, *h*]$_\circ$]$_\circ$
and *C* = [*a*, *b*, *f*]$_\circ$
and *D* = [*a′*, *b′*, *f′*]$_\circ$
and *g* : *a* $\mapsto_{\mathfrak{A}}$ *a′*
and *h* : *b* $\mapsto_{\mathfrak{B}}$ *b′*
and *f* : $\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$ $\mapsto_{\mathfrak{C}}$ $\mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and *f′* : $\mathfrak{G}(\!|ObjMap|\!)(\!|a′|\!)$ $\mapsto_{\mathfrak{C}}$ $\mathfrak{H}(\!|ObjMap|\!)(\!|b′|\!)$
and *f′* $\circ_{A\mathfrak{C}}$ $\mathfrak{G}(\!|ArrMap|\!)(\!|g|\!)$ = $\mathfrak{H}(\!|ArrMap|\!)(\!|h|\!)$ $\circ_{A\mathfrak{C}}$ *f*
by (*elim cat-comma-HomE*[*OF - assms*(*2*,*3*)])
with *that* **show** *?thesis*
by (*metis F-is-arrD*(*1*,*2*,*3*) *cat-comma-Cod-app cat-comma-Dom-app*)
qed

14.2.7 Composition

lemma *cat-comma-composableI*:
assumes $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
and $\mathfrak{H}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
and *ABCGF* = [*BCG*, *ABF*]$_\circ$
and *BCG* : *B* $\mapsto_{\mathfrak{G}}$ $_{CF\downarrow_{CF}}$ $\mathfrak{H}$ *C*
and *ABF* : *A* $\mapsto_{\mathfrak{G}}$ $_{CF\downarrow_{CF}}$ $\mathfrak{H}$ *B*
shows *ABCGF* $\in_\circ$ *cat-comma-composable* $\mathfrak{G}$ $\mathfrak{H}$
proof–
from *assms*(*1*,*2*,*5*) **obtain** *a b f a′ b′ f′ gh*
where *ABF-def*: *ABF* = [[*a*, *b*, *f*]$_\circ$, [*a′*, *b′*, *f′*]$_\circ$, *gh*]$_\circ$
and *A* = [*a*, *b*, *f*]$_\circ$
and *B* = [*a′*, *b′*, *f′*]$_\circ$
by *auto*
with *assms*(*1*,*2*,*4*) **obtain** *a″ b″ f″ g′h′*
where *BCG-def*: *BCG* = [[*a′*, *b′*, *f′*]$_\circ$, [*a″*, *b″*, *f″*]$_\circ$, *g′h′*]$_\circ$
and *B* = [*a′*, *b′*, *f′*]$_\circ$
and *C* = [*a″*, *b″*, *f″*]$_\circ$
by *auto*
from *is-arrD*(*1*)[*OF assms*(*4*)] **have** *BCG* $\in_\circ$ *cat-comma-Arr* $\mathfrak{G}$ $\mathfrak{H}$
unfolding *cat-comma-components′*(*2*)[*OF assms*(*1*,*2*)].
moreover from *is-arrD*(*1*)[*OF assms*(*5*)] **have** *ABF* $\in_\circ$ *cat-comma-Arr* $\mathfrak{G}$ $\mathfrak{H}$
unfolding *cat-comma-components′*(*2*)[*OF assms*(*1*,*2*)].
ultimately show *?thesis*
unfolding *assms*(*3*) *ABF-def BCG-def cat-comma-composable-def*
by *simp*
qed

lemma *cat-comma-composableE*[*elim*]:
assumes *ABCGF* $\in_\circ$ *cat-comma-composable* $\mathfrak{G}$ $\mathfrak{H}$
and $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
and $\mathfrak{H}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{C}$
obtains *BCG ABF A B C*
where *ABCGF* = [*BCG*, *ABF*]$_\circ$
and *BCG* : *B* $\mapsto_{\mathfrak{G}}$ $_{CF\downarrow_{CF}}$ $\mathfrak{H}$ *C*
and *ABF* : *A* $\mapsto_{\mathfrak{G}}$ $_{CF\downarrow_{CF}}$ $\mathfrak{H}$ *B*
proof–
from *assms*(*1*) **obtain** *A B C G F*

where *ABCGF-def*: $ABCGF = [[B,\ C,\ G]_\circ,\ [A,\ B,\ F]_\circ]_\circ$
and *BCG*: $[B,\ C,\ G]_\circ \in_\circ \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
and *ABF*: $[A,\ B,\ F]_\circ \in_\circ \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
unfolding *cat-comma-composable-def*
by (*auto simp*: *cat-comma-components'*[*OF assms*(*2*,*3*)])
note *BCG* = *cat-comma-ArrD*[*OF BCG assms*(*2*,*3*)]
and *ABF* = *cat-comma-ArrD*[*OF ABF assms*(*2*,*3*)]
from *ABF*(*1*) *assms*(*2*,*3*) **obtain** $a\ b\ f\ a'\ b'\ f'\ g\ h$
where $[A,\ B,\ F]_\circ = [A,\ B,\ [g,\ h]_\circ]_\circ$
and *A-def*: $A = [a,\ b,\ f]_\circ$
and *B-def*: $B = [a',\ b',\ f']_\circ$
and *F-def*: $F = [g,\ h]_\circ$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
and *h*: $h : b \mapsto_{\mathfrak{B}} b'$
and *f*: $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and *f'*: $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and [*cat-comma-cs-simps*]:
$f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by *auto*
with *BCG*(*1*) *assms*(*2*,*3*) **obtain** $a''\ b''\ f''\ g'\ h'$
where *g'h'-def*: $[B,\ C,\ G]_\circ = [B,\ C,\ [g',\ h']_\circ]_\circ$
and *C-def*: $C = [a'',\ b'',\ f'']_\circ$
and *G-def*: $G = [g',\ h']_\circ$
and *g'*: $g' : a' \mapsto_{\mathfrak{A}} a''$
and *h'*: $h' : b' \mapsto_{\mathfrak{B}} b''$
and *f''*: $f'' : \mathfrak{G}(\!|ObjMap|\!)(\!|a''|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b''|\!)$
and [*cat-comma-cs-simps*]:
$f'' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g'|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{C}} f'$
by *auto*
from *F-def* **have** $F = [g,\ h]_\circ$ **by** *simp*
from *assms*(*2*,*3*) $g\ h\ f\ f'\ g'\ h'\ f''$ **have**
$[B,\ C,\ G]_\circ : B \mapsto_{\mathfrak{G}\ CF\downarrow_{CF}\ \mathfrak{H}} C$
unfolding *ABCGF-def F-def G-def A-def B-def C-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-is-arrI*
)+
moreover from *assms*(*2*,*3*) $g\ h\ f\ f'\ g'\ h'\ f''$ **have**
$[A,\ B,\ F]_\circ : A \mapsto_{\mathfrak{G}\ CF\downarrow_{CF}\ \mathfrak{H}} B$
unfolding *ABCGF-def F-def G-def A-def B-def C-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-is-arrI*
)+
ultimately show *?thesis* **using** *that ABCGF-def* **by** *auto*
qed

lemma *cat-comma-Comp-vsv*[*cat-comma-cs-intros*]: *vsv* $(\mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Comp|\!))$
unfolding *cat-comma-components* **by** *auto*

lemma *cat-comma-Comp-vdomain*[*cat-comma-cs-simps*]:
$\mathcal{D}_\circ\ (\mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Comp|\!)) =$ *cat-comma-composable* $\mathfrak{G}\ \mathfrak{H}$
unfolding *cat-comma-components* **by** *auto*

lemma *cat-comma-Comp-app*[*cat-comma-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$

```
    and ℌ : 𝔅 ↦↦_{Cα} ℭ
    and G = [B, C, [g', h']_∘]_∘
    and F = [A, B, [g, h]_∘]_∘
    and G : B ↦_{𝔊 CF↓CF ℌ} C
    and F : A ↦_{𝔊 CF↓CF ℌ} B
  shows G ∘_{A 𝔊 CF↓CF ℌ} F = [A, C, [g' ∘_{A 𝔄} g, h' ∘_{A 𝔅} h]_∘]_∘
  using assms(1,2,5,6)
  unfolding cat-comma-components'[OF assms(1,2)] assms(3,4)
  by
    (
      cs-concl
        cs-simp: omega-of-set V-cs-simps vfsequence-simps
        cs-intro: nat-omega-intros V-cs-intros cat-comma-composableI TrueI
    )

lemma cat-comma-Comp-is-arr[cat-comma-cs-intros]:
  assumes 𝔊 : 𝔄 ↦↦_{Cα} ℭ
    and ℌ : 𝔅 ↦↦_{Cα} ℭ
    and BCG : B ↦_{𝔊 CF↓CF ℌ} C
    and ABF : A ↦_{𝔊 CF↓CF ℌ} B
  shows BCG ∘_{A 𝔊 CF↓CF ℌ} ABF : A ↦_{𝔊 CF↓CF ℌ} C
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  from assms(1,2,4) obtain a b f a' b' f' g h
    where ABF-def: ABF = [[a, b, f]_∘, [a', b', f']_∘, [g, h]_∘]_∘
      and A-def: A = [a, b, f]_∘
      and B-def: B = [a', b', f']_∘
      and g: g : a ↦_𝔄 a'
      and h: h : b ↦_𝔅 b'
      and f: f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
      and f': f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
      and [symmetric, cat-cs-simps]:
        f' ∘_{Aℭ} 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_{Aℭ} f
    by auto
  with assms(1,2,3) obtain a'' b'' f'' g' h'
    where BCG-def: BCG = [[a', b', f']_∘, [a'', b'', f'']_∘, [g', h']_∘]_∘
      and C-def: C = [a'', b'', f'']_∘
      and g': g' : a' ↦_𝔄 a''
      and h': h' : b' ↦_𝔅 b''
      and f'': f'' : 𝔊(|ObjMap|)(|a''|) ↦_ℭ ℌ(|ObjMap|)(|b''|)
      and [cat-cs-simps]: f'' ∘_{Aℭ} 𝔊(|ArrMap|)(|g'|) = ℌ(|ArrMap|)(|h'|) ∘_{Aℭ} f'
    by auto
  from g' have 𝔊g': 𝔊(|ArrMap|)(|g'|) : 𝔊(|ObjMap|)(|a'|) ↦_ℭ 𝔊(|ObjMap|)(|a''|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  note [cat-cs-simps] =
    category.cat-assoc-helper[
      where ℭ=ℭ and h=f'' and g=‹𝔊(|ArrMap|)(|g'|)› and q=‹ℌ(|ArrMap|)(|h'|) ∘_{Aℭ} f'›
      ]
    category.cat-assoc-helper[
      where ℭ=ℭ and h=f and g=‹ℌ(|ArrMap|)(|h|)› and q=‹f' ∘_{Aℭ} 𝔊(|ArrMap|)(|g|)›
      ]
  from assms(1,2,3,4) g h f f' g' h' f'' show ?thesis
    unfolding ABF-def BCG-def A-def B-def C-def
    by (intro cat-comma-is-arrI[OF assms(1,2)])
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-comma-cs-simps cs-intro: cat-cs-intros
```

)+
qed

14.2.8 Identity

lemma *cat-comma-CId-vsv*[*cat-comma-cs-intros*]: *vsv* ($\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|CId|\!)$)
unfolding *cat-comma-components* **by** *simp*

lemma *cat-comma-CId-vdomain*[*cat-comma-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ\ (\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|CId|\!)) = \mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
unfolding *cat-comma-components'*[*OF assms*(*1,2*)] **by** *simp*

lemma *cat-comma-CId-app*[*cat-comma-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $A = [a,\ b\ ,f]_\circ$
and $A \in_\circ \mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
shows $\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|CId|\!)(\!|A|\!) = [A,\ A,\ [\mathfrak{A}(\!|CId|\!)(\!|a|\!),\ \mathfrak{B}(\!|CId|\!)(\!|b|\!)]_\circ]_\circ$
proof–
from *assms*(*4*)[*unfolded assms*(*3*), *unfolded cat-comma-components'*[*OF assms*(*1,2*)]]
have $[a,\ b,\ f]_\circ \in_\circ$ *cat-comma-Obj* $\mathfrak{G}\ \mathfrak{H}$**.**
then show *?thesis*
unfolding *cat-comma-components'*(*6*)[*OF assms*(*1,2*)] *assms*(*3*)
by (*simp add*: *nat-omega-simps*)
qed

14.2.9 *Hom*-set

lemma *cat-comma-Hom*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $A \in_\circ \mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
and $B \in_\circ \mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
shows *Hom* ($\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}$) $A\ B$ = *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
proof(*intro vsubset-antisym vsubsetI*, *unfold in-Hom-iff*)
fix *ABF* **assume** $ABF : A \mapsto_{\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}} B$
with *assms*(*1,2*) **show** $ABF \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
by (*elim cat-comma-is-arrE*[*OF - assms*(*1,2*)], *intro cat-comma-HomI*) *force+*
next
fix *ABF* **assume** $ABF \in_\circ$ *cat-comma-Hom* $\mathfrak{G}\ \mathfrak{H}\ A\ B$
with *assms*(*1,2*) **show** $ABF : A \mapsto_{\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}} B$
by (*elim cat-comma-HomE*[*OF - assms*(*1,2*)], *intro cat-comma-is-arrI*) *force+*
qed

14.2.10 Comma category is a category

lemma *category-cat-comma*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *category* α ($\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}$)
proof–

interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{H}$ **by** (*rule assms*(*2*))

show *?thesis*
proof(*rule categoryI'*)

show *vfsequence* ($\mathfrak{G}_{\ CF{\downarrow}_{CF}}\ \mathfrak{H}$) **unfolding** *cat-comma-def* **by** *auto*

show *vcard* $(\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}) = 6_{\mathbb{N}}$
unfolding *cat-comma-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}\ (\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Dom|\!)) \subseteq_{\circ} \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Obj|\!)$
by (*rule cat-comma-Dom-vrange*[*OF assms*])
show $\mathcal{R}_{\circ}\ (\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Cod|\!)) \subseteq_{\circ} \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Obj|\!)$
by (*rule cat-comma-Cod-vrange*[*OF assms*])
show $(GF \in_{\circ} \mathcal{D}_{\circ}\ (\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Comp|\!))) \longleftrightarrow$
$(\exists\, g\ f\ b\ c\ a.\ GF = [g, f]_{\circ} \wedge g : b \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} c \wedge f : a \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} b)$
for *GF*
proof(*intro iffI*; (*elim exE conjE*)*?*; (*simp only*: *cat-comma-Comp-vdomain*)*?*)
assume *prems*: $GF \in_{\circ} \mathfrak{G}$ *cat-comma-composable* $\mathfrak{H}$
with *assms* **obtain** $G\ F\ abf\ a'b'f'\ a''b''f''$
where $GF = [G, F]_{\circ}$
and $G : a'b'f' \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} a''b''f''$
and $F : abf \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} a'b'f'$
by *auto*
with *assms* **show** $\exists\, g\ f\ b\ c\ a.$
$GF = [g, f]_{\circ} \wedge g : b \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} c \wedge f : a \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} b$
by *auto*
qed (*use assms* **in** ‹*cs-concl cs-shallow cs-intro*: *cat-comma-composableI*›)
from *assms* **show** $\mathcal{D}_{\circ}\ (\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|CId|\!)) = \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Obj|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
from *assms* **show** $G \circ_{A\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} F : A \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} C$
if $G : B \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} C$ **and** $F : A \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} B$
for *B C G A F*
using *that* **by** (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
from *assms* **show**
$H \circ_{A\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} G \circ_{A\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} F =$
$H \circ_{A\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} (G \circ_{A\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} F)$
if $H : C \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} D$
and $G : B \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} C$
and $F : A \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} B$
for *C D H B G A F*
using *assms that*
proof–
from *that*(*3*) *assms* **obtain** $a\ b\ f\ a'\ b'\ f'\ g\ h$
where *F-def*: $F = [[a, b, f]_{\circ}, [a', b', f']_{\circ}, [g, h]_{\circ}]_{\circ}$
and *A-def*: $A = [a, b, f]_{\circ}$
and *B-def*: $B = [a', b', f']_{\circ}$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
and *h*: $h : b \mapsto_{\mathfrak{B}} b'$
and *f*: $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and *f'*: $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and [*cat-cs-simps*]: $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by *auto*
with *that*(*2*) *assms* **obtain** $a''\ b''\ f''\ g'\ h'$
where *G-def*: $G = [[a', b', f']_{\circ}, [a'', b'', f'']_{\circ}, [g', h']_{\circ}]_{\circ}$
and *C-def*: $C = [a'', b'', f'']_{\circ}$
and *g'*: $g' : a' \mapsto_{\mathfrak{A}} a''$
and *h'*: $h' : b' \mapsto_{\mathfrak{B}} b''$
and *f''*: $f'' : \mathfrak{G}(\!|ObjMap|\!)(\!|a''|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b''|\!)$
and [*cat-cs-simps*]:
$f'' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g'|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{C}} f'$
by *auto*
with *that*(*1*) *assms* **obtain** $a'''\ b'''\ f'''\ g''\ h''$
where *H-def*: $H = [[a'', b'', f'']_{\circ}, [a''', b''', f''']_{\circ}, [g'', h'']_{\circ}]_{\circ}$
and *D-def*: $D = [a''', b''', f''']_{\circ}$
and *g''*: $g'' : a'' \mapsto_{\mathfrak{A}} a'''$

```
      and h'': h'' : b'' ↦𝔅 b'''
      and f''': f''' : 𝔊(⦇ObjMap⦈)(⦇a'''⦈) ↦ℭ 𝔎(⦇ObjMap⦈)(⦇b'''⦈)
      and [cat-cs-simps]:
        f''' ∘_Aℭ 𝔊(⦇ArrMap⦈)(⦇g''⦈) = 𝔎(⦇ArrMap⦈)(⦇h''⦈) ∘_Aℭ f''
    by auto
  note [cat-cs-simps] =
    category.cat-assoc-helper[
      where ℭ=ℭ
        and h=f''
        and g=‹𝔊(⦇ArrMap⦈)(⦇g'⦈)›
        and q=‹𝔎(⦇ArrMap⦈)(⦇h'⦈) ∘_Aℭ f'›
      ]
    category.cat-assoc-helper[
      where ℭ=ℭ
        and h=f''
        and g=‹𝔊(⦇ArrMap⦈)(⦇g'⦈)›
        and q=‹𝔎(⦇ArrMap⦈)(⦇h'⦈) ∘_Aℭ f'›
      ]
    category.cat-assoc-helper[
      where ℭ=ℭ
        and h=f'''
        and g=‹𝔊(⦇ArrMap⦈)(⦇g''⦈)›
        and q=‹𝔎(⦇ArrMap⦈)(⦇h''⦈) ∘_Aℭ f''›
      ]
  from assms that g h f f' g' h' f'' g'' h''  f''' show ?thesis
    unfolding F-def G-def H-def A-def B-def C-def D-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed

show 𝔊 _CF↓CF 𝔎(⦇CId⦈)(⦇A⦈) : A ↦𝔊 _CF↓CF 𝔎 A
  if A ∈∘ 𝔊 _CF↓CF 𝔎(⦇Obj⦈) for A
  using that
  by (elim cat-comma-ObjE[OF - assms(1)]; (simp only:)?)
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )+

show 𝔊 _CF↓CF 𝔎(⦇CId⦈)(⦇B⦈) ∘_A𝔊 _CF↓CF 𝔎 F = F
  if F : A ↦𝔊 _CF↓CF 𝔎 B for A B F
  using that
  by (elim cat-comma-is-arrE[OF - assms]; (simp only:)?)
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )+

show F ∘_A𝔊 _CF↓CF 𝔎 𝔊 _CF↓CF 𝔎(⦇CId⦈)(⦇B⦈) = F
  if F : B ↦𝔊 _CF↓CF 𝔎 C for B C F
  using that
  by (elim cat-comma-is-arrE[OF - assms]; (simp only:)?)
```

```
    (
      cs-concl
        cs-simp: cat-cs-simps cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )+

show 𝔊 CF↓CF ℌ(|Obj|) ⊆∘ Vset α
proof(intro vsubsetI, elim cat-comma-ObjE[OF - assms])
  fix F a b f assume prems:
    F = [a, b, f]∘
    a ∈∘ 𝔄(|Obj|)
    b ∈∘ 𝔅(|Obj|)
    f : 𝔊(|ObjMap|)(|a|) ↦ℭ ℌ(|ObjMap|)(|b|)
  from prems(2–4) show F ∈∘ Vset α
    unfolding prems(1) by (cs-concl cs-intro: cat-cs-intros V-cs-intros)
qed

show (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (𝔊 CF↓CF ℌ) a b) ∈∘ Vset α
  if A ⊆∘ 𝔊 CF↓CF ℌ(|Obj|)
    and B ⊆∘ 𝔊 CF↓CF ℌ(|Obj|)
    and A ∈∘ Vset α
    and B ∈∘ Vset α
  for A B
proof–

  define A0 where A0 = ℛ∘ (λF∈∘A. F(|0|))
  define A1 where A1 = ℛ∘ (λF∈∘A. F(|1ℕ|))
  define B0 where B0 = ℛ∘ (λF∈∘B. F(|0|))
  define B1 where B1 = ℛ∘ (λF∈∘B. F(|1ℕ|))

  define A0B0 where A0B0 = (⋃∘a∈∘A0. ⋃∘b∈∘B0. Hom 𝔄 a b)
  define A1B1 where A1B1 = (⋃∘a∈∘A1. ⋃∘b∈∘B1. Hom 𝔅 a b)

  have A0B0: A0B0 ∈∘ Vset α
    unfolding A0B0-def
  proof(rule 𝔊.HomDom.cat-Hom-vifunion-in-Vset; (intro vsubsetI)?)
    show A0 ∈∘ Vset α
      unfolding A0-def
    proof(intro vrange-vprojection-in-VsetI that(3))
      fix F assume F ∈∘ A
      with that(1) have F ∈∘ 𝔊 CF↓CF ℌ(|Obj|) by auto
      with assms obtain a b f where F-def: F = [a, b, f]∘ by auto
      show vsv F unfolding F-def by auto
      show 0 ∈∘ 𝒟∘ F unfolding F-def by simp
    qed auto
    show B0 ∈∘ Vset α
      unfolding B0-def
    proof(intro vrange-vprojection-in-VsetI that(4))
      fix F assume F ∈∘ B
      with that(2) have F ∈∘ 𝔊 CF↓CF ℌ(|Obj|) by auto
      with assms obtain a b f where F-def: F = [a, b, f]∘ by auto
      show vsv F unfolding F-def by auto
      show 0 ∈∘ 𝒟∘ F unfolding F-def by simp
    qed auto
  next
    fix a assume a ∈∘ A0
    with that(1) obtain F
      where a-def: a = F(|0|) and F ∈∘ 𝔊 CF↓CF ℌ(|Obj|)
```

```
    unfolding A0-def by force
  with assms obtain b f where F = [a, b, f]∘ and a ∈∘ 𝔄(|Obj|) by auto
  then show a ∈∘ 𝔄(|Obj|) unfolding a-def by simp
next
  fix a assume a ∈∘ B0
  with that(2) obtain F
    where a-def: a = F(|0|) and F ∈∘ 𝔊 CF↓CF ℌ(|Obj|)
    unfolding B0-def by force
  with assms obtain b f where F = [a, b, f]∘ and a ∈∘ 𝔄(|Obj|) by auto
  then show a ∈∘ 𝔄(|Obj|) unfolding a-def by simp
qed

have A1B1: A1B1 ∈∘ Vset α
  unfolding A1B1-def
proof(rule 𝔉.HomDom.cat-Hom-vifunion-in-Vset; (intro vsubsetI)?)
  show A1 ∈∘ Vset α
    unfolding A1-def
  proof(intro vrange-vprojection-in-VsetI that(3))
    fix F assume F ∈∘ A
    with that(1) have F ∈∘ 𝔊 CF↓CF ℌ(|Obj|) by auto
    with assms obtain a b f where F-def: F = [a, b, f]∘ by auto
    show vsv F unfolding F-def by auto
    show 1ℕ ∈∘ 𝒟∘ F unfolding F-def by (simp add: nat-omega-simps)
  qed auto
  show B1 ∈∘ Vset α
    unfolding B1-def
  proof(intro vrange-vprojection-in-VsetI that(4))
    fix F assume F ∈∘ B
    with that(2) have F ∈∘ 𝔊 CF↓CF ℌ(|Obj|) by auto
    with assms obtain a b f where F-def: F = [a, b, f]∘ by auto
    show vsv F unfolding F-def by auto
    show 1ℕ ∈∘ 𝒟∘ F unfolding F-def by (simp add: nat-omega-simps)
  qed auto
next
  fix b assume b ∈∘ A1
  with that(1) obtain F
    where b-def: b = F(|1ℕ|) and F ∈∘ 𝔊 CF↓CF ℌ(|Obj|)
    unfolding A1-def by force
  with assms obtain a f where F = [a, b, f]∘ and b ∈∘ 𝔅(|Obj|)
    by (auto simp: nat-omega-simps)
  then show b ∈∘ 𝔅(|Obj|) unfolding b-def by simp
next
  fix b assume b ∈∘ B1
  with that(2) obtain F
    where b-def: b = F(|1ℕ|) and F ∈∘ 𝔊 CF↓CF ℌ(|Obj|)
    unfolding B1-def by force
  with assms obtain a f where F = [a, b, f]∘ and b ∈∘ 𝔅(|Obj|)
    by (auto simp: nat-omega-simps)
  then show b ∈∘ 𝔅(|Obj|) unfolding b-def by simp
qed

define Q where
  Q i = (if i = 0 then A else if i = 1ℕ then B else (A0B0 ×• A1B1))
  for i
have
  (⋃∘a∈∘A. ⋃∘b∈∘B.
    Hom (𝔊 CF↓CF ℌ) a b) ⊆∘ (∏∘i∈∘ set {0, 1ℕ, 2ℕ}. Q i)
proof
```

```
    (
      intro vsubsetI,
      elim vifunionE,
      unfold in-Hom-iff,
      intro vproductI ballI
    )
    fix abf a′b′f′ F assume prems:
      abf ∈∘ A a′b′f′ ∈∘ B F : abf ↦⟨𝔊 CF↓CF ℌ⟩ a′b′f′
    from prems(3) assms obtain a b f a′ b′ f′ g h
      where F-def: F = [[a, b, f]∘, [a′, b′, f′]∘, [g, h]∘]∘
        and abf-def: abf = [a, b, f]∘
        and a′b′f′-def: a′b′f′ = [a′, b′, f′]∘
        and g: g : a ↦⟨𝔄⟩ a′
        and h: h : b ↦⟨𝔅⟩ b′
        and f : 𝔊(|ObjMap|)(|a|) ↦⟨ℭ⟩ ℌ(|ObjMap|)(|b|)
        and f′ : 𝔊(|ObjMap|)(|a′|) ↦⟨ℭ⟩ ℌ(|ObjMap|)(|b′|)
        and f′ ∘⟨A ℭ⟩ 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘⟨A ℭ⟩ f
      by auto
    have gh: [g, h]∘ ∈∘ A0B0 ×• A1B1
      unfolding A0B0-def A1B1-def
    proof
      (
        intro ftimesI2 vifunionI,
        unfold in-Hom-iff A0-def B0-def A1-def B1-def
      )
      from prems(1) show a ∈∘ ℛ∘ (λF∈∘A. F(|0|))
        by (intro vsv.vsv-vimageI2′[where a=abf]) (simp-all add: abf-def)
      from prems(2) show a′ ∈∘ ℛ∘ (λF∈∘B. F(|0|))
        by (intro vsv.vsv-vimageI2′[where a=a′b′f′]
          (simp-all add: a′b′f′-def)
      from prems(1) show b ∈∘ ℛ∘ (λF∈∘A. F(|1ℕ|))
        by (intro vsv.vsv-vimageI2′[where a=abf]
          (simp-all add: nat-omega-simps abf-def)
      from prems(2) show b′ ∈∘ ℛ∘ (λF∈∘B. F(|1ℕ|))
        by (intro vsv.vsv-vimageI2′[where a=a′b′f′]
          (simp-all add: nat-omega-simps a′b′f′-def)
    qed (intro g h)+
    show vsv F unfolding F-def by auto
    show 𝒟∘ F = set {0, 1ℕ, 2ℕ}
      by (simp add: F-def three nat-omega-simps)
    fix i assume i ∈∘ set {0, 1ℕ, 2ℕ}
    then consider ‹i = 0› | ‹i = 1ℕ› | ‹i = 2ℕ› by auto
    from this prems show F(|i|) ∈∘ Q i
      by cases
        (simp-all add: F-def Q-def gh abf-def a′b′f′-def nat-omega-simps)
  qed
  moreover have (∏∘i∈∘ set {0, 1ℕ, 2ℕ}. Q i) ∈∘ Vset α
  proof(rule Limit-vproduct-in-VsetI)
    show set {0, 1ℕ, 2ℕ} ∈∘ Vset α
      by (cs-concl cs-shallow cs-intro: V-cs-intros)
    from A0B0 A1B1 assms(1,2) that(3,4) show
      Q i ∈∘ Vset α if i ∈∘ set {0, 1ℕ, 2ℕ}
      for i
      by (simp-all add: Q-def Limit-ftimes-in-VsetI nat-omega-simps)
  qed auto
  ultimately show (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (𝔊 CF↓CF ℌ) a b) ∈∘ Vset α by auto
 qed
qed (auto simp: cat-comma-cs-simps intro: cat-comma-cs-intros)
```

qed

14.2.11 Tiny comma category

lemma *tiny-category-cat-comma*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
shows *tiny-category* α $(\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H})$
proof–

interpret $\mathfrak{G}$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-tm-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*2*))
note $\mathfrak{G}$ = $\mathfrak{G}$.*is-functor-axioms*
and $\mathfrak{H}$ = $\mathfrak{H}$.*is-functor-axioms*
interpret *category* α ‹$\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-comma-cs-intros*)

show *?thesis*
proof(*intro tiny-categoryI' category-cat-comma*)
have *vrange-*$\mathfrak{G}$: $\mathcal{R}_\circ$ ($\mathfrak{G}(\!|ObjMap|\!)$) $\in_\circ$ *Vset* α
by (*simp add*: *vrange-in-VsetI* $\mathfrak{G}$.*tm-cf-ObjMap-in-Vset*)
moreover have *vrange-*$\mathfrak{H}$: $\mathcal{R}_\circ$ ($\mathfrak{H}(\!|ObjMap|\!)$) $\in_\circ$ *Vset* α
by (*simp add*: *vrange-in-VsetI* $\mathfrak{H}$.*tm-cf-ObjMap-in-Vset*)
ultimately have *UU-Hom-in-Vset*:
($\bigcup_\circ a \in_\circ \mathcal{R}_\circ$ ($\mathfrak{G}(\!|ObjMap|\!)$). $\bigcup_\circ b \in_\circ \mathcal{R}_\circ$ ($\mathfrak{H}(\!|ObjMap|\!)$). *Hom* $\mathfrak{C}$ *a b*) $\in_\circ$ *Vset* α
using $\mathfrak{G}$.*cf-ObjMap-vrange* $\mathfrak{H}$.*cf-ObjMap-vrange*
by (*auto intro*: $\mathfrak{G}$.*HomCod.cat-Hom-vifunion-in-Vset*)
define *Q* **where**
Q i =
(
if i = *0*
then $\mathfrak{A}(\!|Obj|\!)$
else
if i = $1_{\mathbb{N}}$
then $\mathfrak{B}(\!|Obj|\!)$
else ($\bigcup_\circ a \in_\circ \mathcal{R}_\circ$ ($\mathfrak{G}(\!|ObjMap|\!)$). $\bigcup_\circ b \in_\circ \mathcal{R}_\circ$ ($\mathfrak{H}(\!|ObjMap|\!)$). *Hom* $\mathfrak{C}$ *a b*)
)
for *i*
have $\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!) \subseteq_\circ (\prod_\circ i \in_\circ$ *set* $\{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}$. *Q i*)
proof(*intro vsubsetI*)
fix *A* **assume** *A* $\in_\circ$ $\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!)$
then obtain *a b f*
where *A-def*: *A* = $[a, b, f]_\circ$
and *a*: *a* $\in_\circ$ $\mathfrak{A}(\!|Obj|\!)$
and *b*: *b* $\in_\circ$ $\mathfrak{B}(\!|Obj|\!)$
and *f*: *f* : $\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
by (*elim cat-comma-ObjE*[*OF* - $\mathfrak{G}$ $\mathfrak{H}$])
from *f* **have** *f*:
f $\in_\circ$ ($\bigcup_\circ a \in_\circ \mathcal{R}_\circ$ ($\mathfrak{G}(\!|ObjMap|\!)$). $\bigcup_\circ b \in_\circ \mathcal{R}_\circ$ ($\mathfrak{H}(\!|ObjMap|\!)$). *Hom* $\mathfrak{C}$ *a b*)
by (*intro vifunionI*, *unfold in-Hom-iff*)
(
simp-all add:
a b
$\mathfrak{H}$.*ObjMap.vsv-vimageI2*
$\mathfrak{H}$.*cf-ObjMap-vdomain*
$\mathfrak{G}$.*ObjMap.vsv-vimageI2*
$\mathfrak{G}$.*cf-ObjMap-vdomain*
)

show $A \in_\circ (\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i)$
proof(*intro vproductI*, *unfold Ball-def*; (*intro allI impI*)*?*)
show $\mathcal{D}_\circ\ A = set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$
unfolding *A-def* **by** (*simp add*: *three nat-omega-simps*)
fix *i* **assume** $i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$
then consider $\langle i = 0\rangle \mid \langle i = 1_\mathbb{N}\rangle \mid \langle i = 2_\mathbb{N}\rangle$ **by** *auto*
from *this a b f* **show** $A(\!|i|\!) \in_\circ Q\ i$
unfolding *A-def Q-def* **by** *cases* (*simp-all add*: *nat-omega-simps*)
qed (*auto simp*: *A-def*)
qed
moreover have $(\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i) \in_\circ Vset\ \alpha$
proof(*rule Limit-vproduct-in-VsetI*)
show $set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\} \in_\circ Vset\ \alpha$
unfolding *three*[*symmetric*] **by** *simp*
from *this* **show** $Q\ i \in_\circ Vset\ \alpha$ **if** $i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$ **for** *i*
using *that assms*(*1*,*2*) *UU-Hom-in-Vset*
by
(
simp-all add:
Q-def
$\mathfrak{G}$.*HomDom.tiny-cat-Obj-in-Vset*
$\mathfrak{H}$.*HomDom.tiny-cat-Obj-in-Vset*
nat-omega-simps
)
qed *auto*
ultimately show [*simp*]: $\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Obj|\!) \in_\circ Vset\ \alpha$ **by** *auto*
define *Q* **where**
Q i =
(
if i = *0*
then $\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Obj|\!)$
else
if i = $1_\mathbb{N}$
then $\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Obj|\!)$
else $\mathfrak{A}(\!|Arr|\!) \times_\bullet \mathfrak{B}(\!|Arr|\!)$
)
for *i*
have $\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Arr|\!) \subseteq_\circ (\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i)$
proof(*intro vsubsetI*)
fix F **assume** $F \in_\circ \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Arr|\!)$
then obtain *abf a′b′f′* **where** *F*: $F : abf \mapsto_{\mathfrak{G}\ CF{\downarrow}CF\ \mathfrak{H}} a'b'f'$
by (*auto intro*: *is-arrI*)
with *assms* **obtain** *a b f a′ b′ f′ g h*
where *F-def*: $F = [[a, b, f]_\circ, [a', b', f']_\circ, [g, h]_\circ]_\circ$
and *abf-def*: $abf = [a, b, f]_\circ$
and *a′b′f′-def*: $a'b'f' = [a', b', f']_\circ$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
and *h*: $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by *auto*
from *g h* **have** $[g, h]_\circ \in_\circ \mathfrak{A}(\!|Arr|\!) \times_\bullet \mathfrak{B}(\!|Arr|\!)$ **by** *auto*
show $F \in_\circ (\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i)$
proof(*intro vproductI*, *unfold Ball-def*; (*intro allI impI*)*?*)
show $\mathcal{D}_\circ\ F = set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$
by (*simp add*: *F-def three nat-omega-simps*)
fix *i* **assume** $i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$

```
      then consider ‹i = 0› | ‹i = 1_ℕ› | ‹i = 2_ℕ› by auto
      from this F g h show F(|i|) ∈∘ Q i
        unfolding Q-def F-def abf-def[symmetric] a'b'f'-def[symmetric]
        by cases (auto simp: nat-omega-simps)
    qed (auto simp: F-def)
  qed
  moreover have (∏∘i∈∘ set {0, 1_ℕ, 2_ℕ}. Q i) ∈∘ Vset α
  proof(rule Limit-vproduct-in-VsetI)
    show set {0, 1_ℕ, 2_ℕ} ∈∘ Vset α
      by (simp add: three[symmetric] nat-omega-simps)
    moreover have 𝔄(|Arr|) ×• 𝔅(|Arr|) ∈∘ Vset α
      by
        (
          auto intro!:
            Limit-ftimes-in-VsetI
            𝔊.𝒵-β 𝒵-def
            𝔊.HomDom.tiny-cat-Arr-in-Vset
            ℌ.HomDom.tiny-cat-Arr-in-Vset
        )
    ultimately show Q i ∈∘ Vset α if i ∈∘ set {0, 1_ℕ, 2_ℕ} for i
      using that assms(1,2) UU-Hom-in-Vset
      by
        (
          simp-all add:
            Q-def
            𝔊.HomDom.tiny-cat-Obj-in-Vset
            ℌ.HomDom.tiny-cat-Obj-in-Vset
            nat-omega-simps
        )
  qed auto
  ultimately show 𝔊 _CF↓CF ℌ(|Arr|) ∈∘ Vset α by auto
qed (rule 𝔊, rule ℌ)

qed
```

14.3 Opposite comma category functor

14.3.1 Background

See [2][7] for background information.

14.3.2 Object flip

```
definition op-cf-commma-obj-flip :: V ⇒ V ⇒ V
  where op-cf-commma-obj-flip 𝔊 ℌ =
    (λA∈∘(𝔊 _CF↓CF ℌ)(|Obj|). [A(|1_ℕ|), A(|0|), A(|2_ℕ|)]∘)
```

Elementary properties.

```
mk-VLambda op-cf-commma-obj-flip-def
  |vsv op-cf-commma-obj-flip-vsv[cat-comma-cs-intros]|
  |vdomain op-cf-commma-obj-flip-vdomain[cat-comma-cs-simps]|
  |app op-cf-commma-obj-flip-app'|

lemma op-cf-commma-obj-flip-app[cat-comma-cs-simps]:
  assumes A = [a, b, f]∘ and A ∈∘ (𝔊 _CF↓CF ℌ)(|Obj|)
  shows op-cf-commma-obj-flip 𝔊 ℌ(|A|) = [b, a, f]∘
```

[7] https://en.wikipedia.org/wiki/Opposite_category

using *assms* **unfolding** *op-cf-commma-obj-flip-def* **by** (*simp add*: *nat-omega-simps*)

lemma *op-cf-commma-obj-flip-v11*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *v11* (*op-cf-commma-obj-flip* $\mathfrak{G}$ $\mathfrak{H}$)
proof(*rule vsv.vsv-valeq-v11I*, *unfold op-cf-commma-obj-flip-vdomain*)
fix *A B* **assume** *prems*:
$A \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
$B \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
op-cf-commma-obj-flip $\mathfrak{G}$ $\mathfrak{H}(\!|A|\!)$ = *op-cf-commma-obj-flip* $\mathfrak{G}$ $\mathfrak{H}(\!|B|\!)$
from *prems*(*1*,*2*) *assms* **obtain** *a b f a′ b′ f′*
where *A-def*: $A = [a, b, f]_\circ$
and *B-def*: $B = [a', b', f']_\circ$
by (*elim cat-comma-ObjE*[*OF - assms*])
from *prems*(*3*,*1*,*2*) **show** *A* = *B*
by (*simp-all add*: *A-def B-def op-cf-commma-obj-flip-app nat-omega-simps*)
qed (*auto intro*: *op-cf-commma-obj-flip-vsv*)

lemma *op-cf-commma-obj-flip-vrange*[*cat-comma-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ$ (*op-cf-commma-obj-flip* $\mathfrak{G}$ $\mathfrak{H}$) = (*op-cf* $\mathfrak{H}$) $_{CF\downarrow CF}$ (*op-cf* $\mathfrak{G}$)$(\!|Obj|\!)$
proof(*intro vsubset-antisym*)
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*2*))
show $\mathcal{R}_\circ$ (*op-cf-commma-obj-flip* $\mathfrak{G}$ $\mathfrak{H}$) $\subseteq_\circ$ (*op-cf* $\mathfrak{H}$) $_{CF\downarrow CF}$ (*op-cf* $\mathfrak{G}$)$(\!|Obj|\!)$
proof
(
intro vsv.vsv-vrange-vsubset op-cf-commma-obj-flip-vsv,
unfold cat-comma-cs-simps
)
fix *A* **assume** $A \in_\circ \mathfrak{G}_{\ CF\downarrow CF}\ \mathfrak{H}(\!|Obj|\!)$
then obtain *a b f*
where *A-def*: $A = [a, b, f]_\circ$
and *a*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *b*: $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and *f*: $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
by (*elim cat-comma-ObjE*[*OF - assms*])
from *a b f* **show**
op-cf-commma-obj-flip $\mathfrak{G}$ $\mathfrak{H}(\!|A|\!)$ $\in_\circ$ (*op-cf* $\mathfrak{H}$) $_{CF\downarrow CF}$ (*op-cf* $\mathfrak{G}$)$(\!|Obj|\!)$
unfolding *A-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
qed
show (*op-cf* $\mathfrak{H}$) $_{CF\downarrow CF}$ (*op-cf* $\mathfrak{G}$)$(\!|Obj|\!)$ $\subseteq_\circ$ $\mathcal{R}_\circ$ (*op-cf-commma-obj-flip* $\mathfrak{G}$ $\mathfrak{H}$)
proof(*intro vsubsetI*)
fix *B* **assume** $B \in_\circ$ (*op-cf* $\mathfrak{H}$) $_{CF\downarrow CF}$ (*op-cf* $\mathfrak{G}$)$(\!|Obj|\!)$
then obtain *a b f*
where *B-def*: $B = [b, a, f]_\circ$
and *b*: $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and *a*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *f*: $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
by
(
elim cat-comma-ObjE[

```
        OF - ℌ.is-functor-op 𝔊.is-functor-op, unfolded cat-op-simps
        ]
    )
  from a b f have B-def: B = op-cf-commma-obj-flip 𝔊 ℌ⦇a, b, f⦈•
    by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps B-def
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )
  from a b f have [a, b, f]∘ ∈∘ 𝒟∘ (op-cf-commma-obj-flip 𝔊 ℌ)
    by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros
    )
  with op-cf-commma-obj-flip-vsv show B ∈∘ ℛ∘ (op-cf-commma-obj-flip 𝔊 ℌ)
    unfolding B-def by auto
 qed
qed
```

14.3.3 Definition and elementary properties

```
definition op-cf-comma :: V ⇒ V ⇒ V
  where op-cf-comma 𝔊 ℌ =
   [
     op-cf-commma-obj-flip 𝔊 ℌ,
     (
       λABF∈∘(𝔊 CF↓CF ℌ)⦇Arr⦈.
         [
           op-cf-commma-obj-flip 𝔊 ℌ⦇ABF⦇1ℕ⦈⦈,
           op-cf-commma-obj-flip 𝔊 ℌ⦇ABF⦇0⦈⦈,
           [ABF⦇2ℕ⦈⦇1ℕ⦈, ABF⦇2ℕ⦈⦇0ℕ⦈]∘
         ]∘
     ),
     op-cat (𝔊 CF↓CF ℌ),
     (op-cf ℌ) CF↓CF (op-cf 𝔊)
   ]∘
```

Components.

```
lemma op-cf-comma-components:
  shows [cat-comma-cs-simps]:
      op-cf-comma 𝔊 ℌ⦇ObjMap⦈ = op-cf-commma-obj-flip 𝔊 ℌ
    and op-cf-comma 𝔊 ℌ⦇ArrMap⦈ =
     (
       λABF∈∘(𝔊 CF↓CF ℌ)⦇Arr⦈.
         [
           op-cf-commma-obj-flip 𝔊 ℌ⦇ABF⦇1ℕ⦈⦈,
           op-cf-commma-obj-flip 𝔊 ℌ⦇ABF⦇0⦈⦈,
           [ABF⦇2ℕ⦈⦇1ℕ⦈, ABF⦇2ℕ⦈⦇0ℕ⦈]∘
         ]∘
     )
    and [cat-comma-cs-simps]:
      op-cf-comma 𝔊 ℌ⦇HomDom⦈ = op-cat (𝔊 CF↓CF ℌ)
    and [cat-comma-cs-simps]:
      op-cf-comma 𝔊 ℌ⦇HomCod⦈ = (op-cf ℌ) CF↓CF (op-cf 𝔊)
  unfolding op-cf-comma-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

14.3.4 Arrow map

mk-VLambda *op-cf-comma-components*(*2*)
|*vsv op-cf-comma-ArrMap-vsv*[*cat-comma-cs-intros*]|
|*vdomain op-cf-comma-ArrMap-vdomain*[*cat-comma-cs-simps*]|
|*app op-cf-comma-ArrMap-app′*|

lemma *op-cf-comma-ArrMap-app*[*cat-comma-cs-simps*]:
assumes $ABF = [[a, b, f]_\circ, [a', b', f']_\circ, [g, h]_\circ]_\circ$
and $ABF \in_\circ \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Arr|\!)$
shows *op-cf-comma* $\mathfrak{G}\ \mathfrak{H}(\!|ArrMap|\!)(\!|ABF|\!) =$
[
op-cf-commma-obj-flip $\mathfrak{G}\ \mathfrak{H}(\!|a', b', f'|\!)_\bullet$,
op-cf-commma-obj-flip $\mathfrak{G}\ \mathfrak{H}(\!|a, b, f|\!)_\bullet$,
$[h, g]_\circ$
]$_\circ$
using *assms*(*2*) **by** (*simp add*: *assms*(*1*) *op-cf-comma-ArrMap-app′ nat-omega-simps*)

lemma *op-cf-comma-ArrMap-v11*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *v11* (*op-cf-comma* $\mathfrak{G}\ \mathfrak{H}(\!|ArrMap|\!)$)
proof
(
rule vsv.vsv-valeq-v11I,
unfold op-cf-comma-ArrMap-vdomain,
intro op-cf-comma-ArrMap-vsv
)
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{H}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}\mathfrak{H}$: *category* $\alpha\ \langle\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}\rangle$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
fix *ABF ABF′* **assume** *prems*:
$ABF \in_\circ \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Arr|\!)$
$ABF' \in_\circ \mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}(\!|Arr|\!)$
op-cf-comma $\mathfrak{G}\ \mathfrak{H}(\!|ArrMap|\!)(\!|ABF|\!) =$ *op-cf-comma* $\mathfrak{G}\ \mathfrak{H}(\!|ArrMap|\!)(\!|ABF'|\!)$
from *prems*(*1*) **obtain** *A B* **where** *ABF*: $ABF : A \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} B$ **by** *auto*
from *prems*(*2*) **obtain** *A′ B′* **where** *ABF′*: $ABF' : A' \mapsto_{\mathfrak{G}_{\ CF{\downarrow}CF}\ \mathfrak{H}} B'$ **by** *auto*
from *ABF* **obtain** *a b f a′ b′ f′ g h*
where *ABF-def*: $ABF = [[a, b, f]_\circ, [a', b', f']_\circ, [g, h]_\circ]_\circ$
and *A-def*: $A = [a, b, f]_\circ$
and *B-def*: $B = [a', b', f']_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $h : b \mapsto_{\mathfrak{B}} b'$
and $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by (*elim cat-comma-is-arrE*[*OF - assms*])
from *ABF′* **obtain** $a'' b'' f'' a''' b''' f''' g' h'$
where *ABF′-def*: $ABF' = [[a'', b'', f'']_\circ, [a''', b''', f''']_\circ, [g', h']_\circ]_\circ$
and *A′-def*: $A' = [a'', b'', f'']_\circ$
and *B′-def*: $B' = [a''', b''', f''']_\circ$
and $g' : a'' \mapsto_{\mathfrak{A}} a'''$
and $h' : b'' \mapsto_{\mathfrak{B}} b'''$
and $f'' : \mathfrak{G}(\!|ObjMap|\!)(\!|a''|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b''|\!)$
and $f''' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'''|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'''|\!)$
and $f''' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g'|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h'|\!) \circ_{A\mathfrak{C}} f''$
by (*elim cat-comma-is-arrE*[*OF - assms*])
from *ABF ABF′* **have** *abf*:

```
    [a, b, f]∘ ∈∘ (𝔊 _CF↓CF ℌ)(|Obj|)
    [a', b', f']∘ ∈∘ (𝔊 _CF↓CF ℌ)(|Obj|)
    [a'', b'', f'']∘ ∈∘ (𝔊 _CF↓CF ℌ)(|Obj|)
    [a''', b''', f''']∘ ∈∘ (𝔊 _CF↓CF ℌ)(|Obj|)
    unfolding ABF-def ABF'-def A-def B-def A'-def B'-def by auto
  note v11-injective = v11.v11-injective[
      OF op-cf-commma-obj-flip-v11, OF assms, unfolded cat-comma-cs-simps
      ]
  from
    prems(3,1,2) assms
    op-cf-commma-obj-flip-v11
    v11-injective[OF abf(1,3)]
    v11-injective[OF abf(2,4)]
  show ABF = ABF'
    by
      (
        simp-all add:
          ABF-def ABF'-def op-cf-comma-ArrMap-app' nat-omega-simps
      )
qed

lemma op-cf-comma-ArrMap-is-arr:
  assumes 𝔊 : 𝔄 ↦↦_Cα ℭ
    and ℌ : 𝔅 ↦↦_Cα ℭ
    and ABF : A ↦_𝔊 _CF↓CF ℌ B
  shows op-cf-comma 𝔊 ℌ(|ArrMap|)(|ABF|) :
    op-cf-commma-obj-flip 𝔊 ℌ(|B|) ↦(op-cf ℌ) _CF↓CF (op-cf 𝔊)
    op-cf-commma-obj-flip 𝔊 ℌ(|A|)
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  interpret 𝔊ℌ: category α ‹𝔊 _CF↓CF ℌ›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
  from assms(3) obtain a b f a' b' f' g h
    where ABF-def: ABF = [[a, b, f]∘, [a', b', f']∘, [g, h]∘]∘
      and A-def: A = [a, b, f]∘
      and B-def: B = [a', b', f']∘
      and g: g : a ↦_𝔄 a'
      and h: h : b ↦_𝔅 b'
      and f: f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
      and f': f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
      and f'g-hf: f' ∘_Aℭ 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_Aℭ f
    by (elim cat-comma-is-arrE[OF - assms(1,2)])
  from g h f f' f'g-hf show ?thesis
    unfolding ABF-def A-def B-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
      )
qed

lemma op-cf-comma-ArrMap-is-arr':
  assumes 𝔊 : 𝔄 ↦↦_Cα ℭ
    and ℌ : 𝔅 ↦↦_Cα ℭ
    and ABF : A ↦_𝔊 _CF↓CF ℌ B
    and A' = op-cf-commma-obj-flip 𝔊 ℌ(|B|)
```

```
    and B' = op-cf-commma-obj-flip 𝔊 ℌ(|A|)
  shows op-cf-comma 𝔊 ℌ(|ArrMap|)(|ABF|) : A' ↦_(op-cf ℌ) CF↓CF (op-cf 𝔊) B'
  using assms(1–3) unfolding assms(4,5) by (intro op-cf-comma-ArrMap-is-arr)

lemma op-cf-comma-ArrMap-vrange[cat-comma-cs-simps]:
  assumes 𝔊 : 𝔄 ↦↦_Cα ℭ and ℌ : 𝔅 ↦↦_Cα ℭ
  shows ℛ∘ (op-cf-comma 𝔊 ℌ(|ArrMap|)) = (op-cf ℌ) CF↓CF (op-cf 𝔊)(|Arr|)
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  interpret 𝔊ℌ: category α ‹𝔊 CF↓CF ℌ›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
  interpret op-𝔊ℌ: category α ‹(op-cf ℌ) CF↓CF (op-cf 𝔊)›
    by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros cat-op-intros)
  show ?thesis
  proof(intro vsubset-antisym)
    show ℛ∘ (op-cf-comma 𝔊 ℌ(|ArrMap|)) ⊆∘ (op-cf ℌ) CF↓CF (op-cf 𝔊)(|Arr|)
    proof
      (
        intro vsv.vsv-vrange-vsubset op-cf-comma-ArrMap-vsv,
        unfold cat-comma-cs-simps
      )
      fix ABF assume prems: ABF ∈∘ 𝔊 CF↓CF ℌ(|Arr|)
      then obtain A B where ABF: ABF : A ↦_𝔊 CF↓CF ℌ B by auto
      from op-cf-comma-ArrMap-is-arr[OF assms this] show
        op-cf-comma 𝔊 ℌ(|ArrMap|)(|ABF|) ∈∘ (op-cf ℌ) CF↓CF (op-cf 𝔊)(|Arr|)
        by auto
    qed
    show (op-cf ℌ) CF↓CF (op-cf 𝔊)(|Arr|) ⊆∘ ℛ∘ (op-cf-comma 𝔊 ℌ(|ArrMap|))
    proof(intro vsubsetI)
      fix ABF assume prems: ABF ∈∘ (op-cf ℌ) CF↓CF (op-cf 𝔊)(|Arr|)
      then obtain A B where ABF: ABF : A ↦_(op-cf ℌ) CF↓CF (op-cf 𝔊) B
        by auto
      then obtain a b f a' b' f' g h
        where ABF-def: ABF = [[a, b, f]∘, [a', b', f']∘, [g, h]∘]∘
          and A-def: A = [a, b, f]∘
          and B-def: B = [a', b', f']∘
          and g: g : a' ↦_𝔅 a
          and h: h : b' ↦_𝔄 b
          and f: f : 𝔊(|ObjMap|)(|b|) ↦_ℭ ℌ(|ObjMap|)(|a|)
          and f': f' : 𝔊(|ObjMap|)(|b'|) ↦_ℭ ℌ(|ObjMap|)(|a'|)
          and f'g-hf: f' ∘_A op-cat ℭ ℌ(|ArrMap|)(|g|) = 𝔊(|ArrMap|)(|h|) ∘_A op-cat ℭ f
        by
          (
            elim cat-comma-is-arrE[
              OF - ℌ.is-functor-op 𝔊.is-functor-op, unfolded cat-op-simps
              ]
          )
      from f'g-hf g h f f' have gf'-fh:
        ℌ(|ArrMap|)(|g|) ∘_Aℭ f' = f ∘_Aℭ 𝔊(|ArrMap|)(|h|)
        by
          (
            cs-prems cs-shallow
              cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-op-intros
          )
      with g h f f' have
        [[b', a', f']∘, [b, a, f]∘, [h, g]∘]∘ ∈∘ 𝒟∘ (op-cf-comma 𝔊 ℌ(|ArrMap|))
        ABF = op-cf-comma 𝔊 ℌ(|ArrMap|)(|[[b', a', f']∘, [b, a, f]∘, [h, g]∘]∘|)•
```

```
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-comma-cs-simps ABF-def
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )+
      with op-cf-comma-ArrMap-vsv show ABF ∈∘ ℛ∘ (op-cf-comma 𝔊 ℌ(|ArrMap|))
        by auto
    qed
  qed
qed
```

14.3.5 Opposite comma category functor is an isomorphism of categories

```
lemma op-cf-comma-is-iso-functor:
  assumes 𝔊 : 𝔄 ↦↦_Cα ℭ and ℌ : 𝔅 ↦↦_Cα ℭ
  shows op-cf-comma 𝔊 ℌ :
    op-cat (𝔊 _CF↓CF ℌ) ↦↦_C.isoα (op-cf ℌ) _CF↓CF (op-cf 𝔊)
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  show ?thesis
  proof(intro is-iso-functorI' is-functorI')
    show vfsequence (op-cf-comma 𝔊 ℌ)
      unfolding op-cf-comma-def by simp
    show vcard (op-cf-comma 𝔊 ℌ) = ℶ_ℕ
      unfolding op-cf-comma-def by (simp add: nat-omega-simps)
    from assms show op-cf-comma 𝔊 ℌ(|ArrMap|)(|ABF|) :
      op-cf-comma 𝔊 ℌ(|ObjMap|)(|A|) ↦_(op-cf ℌ) _CF↓CF (op-cf 𝔊)
      op-cf-comma 𝔊 ℌ(|ObjMap|)(|B|)
      if ABF : A ↦_op-cat (𝔊 _CF↓CF ℌ) B for A B ABF
      using that
      unfolding cat-op-simps
      by
        (
          cs-concl cs-shallow
            cs-intro: op-cf-comma-ArrMap-is-arr' cs-simp: cat-comma-cs-simps
        )
    show
      op-cf-comma 𝔊 ℌ(|ArrMap|)(|G ∘_A op-cat (𝔊 _CF↓CF ℌ) F|) =
        op-cf-comma 𝔊 ℌ(|ArrMap|)(|G|) ∘_A (op-cf ℌ) _CF↓CF (op-cf 𝔊)
          op-cf-comma 𝔊 ℌ(|ArrMap|)(|F|)
      if G : B ↦_op-cat (𝔊 _CF↓CF ℌ) C
        and F : A ↦_op-cat (𝔊 _CF↓CF ℌ) B
      for B C G A F
    proof–
      note G = that(1)[unfolded cat-op-simps]
      note F = that(2)[unfolded cat-op-simps]
      from assms G obtain a b f a' b' f' g h
        where G-def: G = [[a, b, f]∘, [a', b', f']∘, [g, h]∘]∘
          and C-def: C = [a, b, f]∘
          and B-def: B = [a', b', f']∘
          and g: g : a ↦_𝔄 a'
          and h: h : b ↦_𝔅 b'
          and f: f : 𝔊(|ObjMap|)(|a|) ↦_ℭ ℌ(|ObjMap|)(|b|)
          and f': f' : 𝔊(|ObjMap|)(|a'|) ↦_ℭ ℌ(|ObjMap|)(|b'|)
          and [symmetric, cat-comma-cs-simps]:
```

```
        f' ∘_A𝔈 𝔊(|ArrMap|)(|g|) = ℌ(|ArrMap|)(|h|) ∘_A𝔈 f
      by auto
    with assms F obtain a'' b'' f'' g' h'
      where F-def: F = [[a', b', f']∘, [a'', b'', f'']∘, [g', h']∘]∘
        and A-def: A = [a'', b'', f'']∘
        and g': g' : a' ↦_𝔄 a''
        and h': h' : b' ↦_𝔅 b''
        and f'': f'' : 𝔊(|ObjMap|)(|a''|) ↦_𝔈 ℌ(|ObjMap|)(|b''|)
        and [cat-comma-cs-simps]:
          f'' ∘_A𝔈 𝔊(|ArrMap|)(|g'|) = ℌ(|ArrMap|)(|h'|) ∘_A𝔈 f'
      by auto
    note [cat-comma-cs-simps] =
      category.cat-assoc-helper[
        where 𝔈=𝔈 and h=f'' and g=‹𝔊(|ArrMap|)(|g'|)› and q=‹ℌ(|ArrMap|)(|h'|) ∘_A𝔈 f'›
        ]
    from assms that g h f f' g' h' f' f'' show ?thesis
      unfolding cat-op-simps G-def C-def B-def F-def A-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
        )
  qed
  show
    op-cf-comma 𝔊 ℌ(|ArrMap|)(|op-cat (𝔊 _CF↓CF ℌ)(|CId|)(|C|)) =
      (op-cf ℌ) _CF↓CF (op-cf 𝔊)(|CId|)(|op-cf-comma 𝔊 ℌ(|ObjMap|)(|C|))
    if C ∈∘ op-cat (𝔊 _CF↓CF ℌ)(|Obj|) for C
  proof–
    from that[unfolded cat-op-simps] assms obtain a b f
      where C-def: C = [a, b, f]∘
        and a: a ∈∘ 𝔄(|Obj|)
        and b: b ∈∘ 𝔅(|Obj|)
        and f: f : 𝔊(|ObjMap|)(|a|) ↦_𝔈 ℌ(|ObjMap|)(|b|)
      by auto
    from a b f that show ?thesis
      unfolding cat-op-simps C-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
        )
  qed
qed
  (
    cs-concl cs-shallow
      cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
      cs-intro: V-cs-intros cat-cs-intros cat-comma-cs-intros cat-op-intros
  )+
qed

lemma op-cf-comma-is-iso-functor'[cat-comma-cs-intros]:
  assumes 𝔊 : 𝔄 ↦↦_Cα 𝔈
    and ℌ : 𝔅 ↦↦_Cα 𝔈
    and 𝔄' = op-cat (𝔊 _CF↓CF ℌ)
    and 𝔅' = (op-cf ℌ) _CF↓CF (op-cf 𝔊)
  shows op-cf-comma 𝔊 ℌ : 𝔄' ↦↦_C.isoα 𝔅'
```

using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*rule op-cf-comma-is-iso-functor*)

lemma *op-cf-comma-is-functor*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *op-cf-comma* $\mathfrak{G}$ $\mathfrak{H}$:
$op\text{-}cat\ (\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}) \mapsto\mapsto_{C\alpha} (op\text{-}cf\ \mathfrak{H})\ _{CF}{\downarrow}_{CF}\ (op\text{-}cf\ \mathfrak{G})$
by (*rule is-iso-functorD*(*1*)[*OF op-cf-comma-is-iso-functor*[*OF assms*]])

lemma *op-cf-comma-is-functor'*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{A}' = op\text{-}cat\ (\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H})$
and $\mathfrak{B}' = (op\text{-}cf\ \mathfrak{H})\ _{CF}{\downarrow}_{CF}\ (op\text{-}cf\ \mathfrak{G})$
shows *op-cf-comma* $\mathfrak{G}$ $\mathfrak{H} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*rule op-cf-comma-is-functor*)

14.4 Projections for a comma category

14.4.1 Definitions and elementary properties

See Chapter II-6 in [7].

definition *cf-comma-proj-left* :: $V \Rightarrow V \Rightarrow V$ ($\langle(\text{-}\ _{CF}\sqcap\ \text{-})\rangle$ [*1000*, *1000*] *999*)
where $\mathfrak{G}\ _{CF}\sqcap\ \mathfrak{H} =$
[
$(\lambda a \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!).\ a(\!|0|\!))$,
$(\lambda f \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Arr|\!).\ f(\!|2_{\mathbb{N}}|\!)(\!|0|\!))$,
$\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}$,
$\mathfrak{G}(\!|HomDom|\!)$
]$_{\circ}$

definition *cf-comma-proj-right* :: $V \Rightarrow V \Rightarrow V$ ($\langle(\text{-}\sqcap_{CF}\ \text{-})\rangle$ [*1000*, *1000*] *999*)
where $\mathfrak{G} \sqcap_{CF} \mathfrak{H} =$
[
$(\lambda a \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!).\ a(\!|1_{\mathbb{N}}|\!))$,
$(\lambda f \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Arr|\!).\ f(\!|2_{\mathbb{N}}|\!)(\!|1_{\mathbb{N}}|\!))$,
$\mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}$,
$\mathfrak{H}(\!|HomDom|\!)$
]$_{\circ}$

Components.

lemma *cf-comma-proj-left-components*:
shows $\mathfrak{G}\ _{CF}\sqcap\ \mathfrak{H}(\!|ObjMap|\!) = (\lambda a \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!).\ a(\!|0|\!))$
and $\mathfrak{G}\ _{CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!) = (\lambda f \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Arr|\!).\ f(\!|2_{\mathbb{N}}|\!)(\!|0|\!))$
and $\mathfrak{G}\ _{CF}\sqcap\ \mathfrak{H}(\!|HomDom|\!) = \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}$
and $\mathfrak{G}\ _{CF}\sqcap\ \mathfrak{H}(\!|HomCod|\!) = \mathfrak{G}(\!|HomDom|\!)$
unfolding *cf-comma-proj-left-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

lemma *cf-comma-proj-right-components*:
shows $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|ObjMap|\!) = (\lambda a \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Obj|\!).\ a(\!|1_{\mathbb{N}}|\!))$
and $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|ArrMap|\!) = (\lambda f \in_{\circ} \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|Arr|\!).\ f(\!|2_{\mathbb{N}}|\!)(\!|1_{\mathbb{N}}|\!))$
and $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|HomDom|\!) = \mathfrak{G}\ _{CF}{\downarrow}_{CF}\ \mathfrak{H}$
and $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|HomCod|\!) = \mathfrak{H}(\!|HomDom|\!)$
unfolding *cf-comma-proj-right-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ $\mathfrak{H}$

assumes $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H}$: $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)
interpretation $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule* $\mathfrak{H}$)

lemmas *cf-comma-proj-left-components′* =
cf-comma-proj-left-components[*of* $\mathfrak{G}$ $\mathfrak{H}$, *unfolded* $\mathfrak{G}$.*cf-HomDom*]

lemmas *cf-comma-proj-right-components′* =
cf-comma-proj-right-components[*of* $\mathfrak{G}$ $\mathfrak{H}$, *unfolded* $\mathfrak{H}$.*cf-HomDom*]

lemmas [*cat-comma-cs-simps*] =
cf-comma-proj-left-components′(*3*,*4*)
cf-comma-proj-right-components′(*3*,*4*)

end

14.4.2 Object map

mk-VLambda *cf-comma-proj-left-components*(*1*)
|*vsv cf-comma-proj-left-ObjMap-vsv*[*cat-comma-cs-intros*]|
|*vdomain cf-comma-proj-left-ObjMap-vdomain*[*cat-comma-cs-simps*]|

mk-VLambda *cf-comma-proj-right-components*(*1*)
|*vsv cf-comma-proj-right-ObjMap-vsv*[*cat-comma-cs-intros*]|
|*vdomain cf-comma-proj-right-ObjMap-vdomain*[*cat-comma-cs-simps*]|

lemma *cf-comma-proj-left-ObjMap-app*[*cat-comma-cs-simps*]:
assumes $A = [a, b, f]_\circ$ **and** $[a, b, f]_\circ \in_\circ \mathfrak{G}\ _{CF}{\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
shows $\mathfrak{G}\ _{CF}{\prod}\ \mathfrak{H}(\!|ObjMap|\!)(\!|A|\!) = a$
using *assms*(*2*) **unfolding** *assms*(*1*) *cf-comma-proj-left-components* **by** *simp*

lemma *cf-comma-proj-right-ObjMap-app*[*cat-comma-cs-simps*]:
assumes $A = [a, b, f]_\circ$ **and** $[a, b, f]_\circ \in_\circ \mathfrak{G}\ _{CF}{\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
shows $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!)(\!|A|\!) = b$
using *assms*(*2*)
unfolding *assms*(*1*) *cf-comma-proj-right-components*
by (*simp add*: *nat-omega-simps*)

lemma *cf-comma-proj-left-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ\ (\mathfrak{G}\ _{CF}{\prod}\ \mathfrak{H}(\!|ObjMap|\!)) \subseteq_\circ \mathfrak{A}(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-comma-cs-simps*)
fix A **assume** *prems*: $A \in_\circ \mathfrak{G}\ _{CF}{\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
with *assms* **obtain** $a\ b\ f$ **where** *A-def*: $A = [a, b, f]_\circ$ **and** a: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
by *auto*
from *assms prems a* **show** $\mathfrak{G}\ _{CF}{\prod}\ \mathfrak{H}(\!|ObjMap|\!)(\!|A|\!) \in_\circ \mathfrak{A}(\!|Obj|\!)$
unfolding *A-def* **by** (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
qed (*auto intro*: *cat-comma-cs-intros*)

lemma *cf-comma-proj-right-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!)) \subseteq_\circ \mathfrak{B}(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-comma-cs-simps*)
fix A **assume** *prems*: $A \in_\circ \mathfrak{G}\ _{CF}{\downarrow_{CF}}\ \mathfrak{H}(\!|Obj|\!)$
with *assms* **obtain** $a\ b\ f$ **where** *A-def*: $A = [a, b, f]_\circ$ **and** b: $b \in_\circ \mathfrak{B}(\!|Obj|\!)$

by *auto*
from *assms prems b* **show** $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|ObjMap|\!)(\!|A|\!) \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
unfolding *A-def* **by** (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
qed (*auto intro*: *cat-comma-cs-intros*)

14.4.3 Arrow map

mk-VLambda *cf-comma-proj-left-components*(*2*)
|*vsv cf-comma-proj-left-ArrMap-vsv*[*cat-comma-cs-intros*]|
|*vdomain cf-comma-proj-left-ArrMap-vdomain*[*cat-comma-cs-simps*]|

mk-VLambda *cf-comma-proj-right-components*(*2*)
|*vsv cf-comma-proj-right-ArrMap-vsv*[*cat-comma-cs-intros*]|
|*vdomain cf-comma-proj-right-ArrMap-vdomain*[*cat-comma-cs-simps*]|

lemma *cf-comma-proj-left-ArrMap-app*[*cat-comma-cs-simps*]:
assumes $ABF = [A, B, [g, h]_{\circ}]_{\circ}$ **and** $[A, B, [g, h]_{\circ}]_{\circ} \in_{\circ} \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
shows $\mathfrak{G}_{\ CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|ABF|\!) = g$
using *assms*(*2*)
unfolding *assms*(*1*) *cf-comma-proj-left-components*
by (*simp add*: *nat-omega-simps*)

lemma *cf-comma-proj-right-ArrMap-app*[*cat-comma-cs-simps*]:
assumes $ABF = [A, B, [g, h]_{\circ}]_{\circ}$
and $[A, B, [g, h]_{\circ}]_{\circ} \in_{\circ} \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
shows $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|ArrMap|\!)(\!|ABF|\!) = h$
using *assms*(*2*)
unfolding *assms*(*1*) *cf-comma-proj-right-components*
by (*simp add*: *nat-omega-simps*)

lemma *cf-comma-proj-left-ArrMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}\ (\mathfrak{G}_{\ CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)) \subseteq_{\circ} \mathfrak{A}(\!|Arr|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-comma-cs-simps*)
from *assms* **interpret** *category* α $\langle \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H} \rangle$
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
fix *F* **assume** *prems*: $F \in_{\circ} \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
then obtain *A B* **where** $F : A \mapsto_{\mathfrak{G}\ CF\downarrow_{CF}\ \mathfrak{H}} B$ **by** *auto*
with *assms* **obtain** *a b f a′ b′ f′ g h*
where *F-def*: $F = [[a, b, f]_{\circ}, [a', b', f']_{\circ}, [g, h]_{\circ}]_{\circ}$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
by *auto*
from *assms prems g* **show** $\mathfrak{G}_{\ CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|F|\!) \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
unfolding *F-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto intro*: *cat-comma-cs-intros*)

lemma *cf-comma-proj-right-ArrMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}\ (\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|ArrMap|\!)) \subseteq_{\circ} \mathfrak{B}(\!|Arr|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-comma-cs-simps*)
from *assms* **interpret** *category* α $\langle \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H} \rangle$
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
fix *F* **assume** *prems*: $F \in_{\circ} \mathfrak{G}_{\ CF\downarrow_{CF}}\ \mathfrak{H}(\!|Arr|\!)$
then obtain *A B* **where** *F*: $F : A \mapsto_{\mathfrak{G}\ CF\downarrow_{CF}\ \mathfrak{H}} B$ **by** *auto*
with *assms* **obtain** *a b f a′ b′ f′ g h*
where *F-def*: $F = [[a, b, f]_{\circ}, [a', b', f']_{\circ}, [g, h]_{\circ}]_{\circ}$
and *h*: $h : b \mapsto_{\mathfrak{B}} b'$

by *auto*
from *assms prems h* **show** $\mathfrak{G} \sqcap_{CF} \mathfrak{H}(\!|ArrMap|\!)(\!|F|\!) \in_\circ \mathfrak{B}(\!|Arr|\!)$
unfolding *F-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto intro*: *cat-comma-cs-intros*)

14.4.4 Projections for a comma category are functors

lemma *cf-comma-proj-left-is-functor*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H} : \mathfrak{G}\ {}_{CF}\!\downarrow_{CF}\ \mathfrak{H} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
proof–
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*2*))
from *assms* **interpret** $\mathfrak{G}\mathfrak{H}$: *category* α $\langle \mathfrak{G}\ {}_{CF}\!\downarrow_{CF}\ \mathfrak{H} \rangle$
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
show *?thesis*
proof(*rule is-functorI'*)
show *vfsequence* $(\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H})$
unfolding *cf-comma-proj-left-def* **by** *auto*
show *vcard* $(\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}) = \aleph_{\mathbb{N}}$
unfolding *cf-comma-proj-left-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** $\mathcal{R}_\circ\ (\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ObjMap|\!)) \subseteq_\circ \mathfrak{A}(\!|Obj|\!)$
by (*rule cf-comma-proj-left-ObjMap-vrange*)
show $\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|F|\!) : \mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ObjMap|\!)(\!|A|\!) \mapsto_{\mathfrak{A}} \mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ObjMap|\!)(\!|B|\!)$
if $F : A \mapsto_{\mathfrak{G}\ {}_{CF}\!\downarrow_{CF}\ \mathfrak{H}} B$ **for** $A\ B\ F$
proof–
from *assms that* **obtain** $a\ b\ f\ a'\ b'\ f'\ g\ h$
where *F-def*: $F = [[a,\ b,\ f]_\circ,\ [a',\ b',\ f']_\circ,\ [g,\ h]_\circ]_\circ$
and *A-def*: $A = [a,\ b,\ f]_\circ$
and *B-def*: $B = [a',\ b',\ f']_\circ$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
by *auto*
from *that g* **show**
$\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|F|\!) : \mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ObjMap|\!)(\!|A|\!) \mapsto_{\mathfrak{A}} \mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ObjMap|\!)(\!|B|\!)$
unfolding *F-def A-def B-def*
by (*cs-concl* **cs-simp**: *cat-comma-cs-simps* **cs-intro**: *cat-cs-intros*)
qed
show
$\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|G \circ_{A\mathfrak{G}\ {}_{CF}\!\downarrow_{CF}\ \mathfrak{H}} F|\!) =$
$\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|G|\!) \circ_{A\mathfrak{A}} \mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}(\!|ArrMap|\!)(\!|F|\!)$
if $G : B \mapsto_{\mathfrak{G}\ {}_{CF}\!\downarrow_{CF}\ \mathfrak{H}} C$ **and** $F : A \mapsto_{\mathfrak{G}\ {}_{CF}\!\downarrow_{CF}\ \mathfrak{H}} B$ **for** $B\ C\ G\ A\ F$
proof–
from *assms that*(*2*) **obtain** $a\ b\ f\ a'\ b'\ f'\ g\ h$
where *F-def*: $F = [[a,\ b,\ f]_\circ,\ [a',\ b',\ f']_\circ,\ [g,\ h]_\circ]_\circ$
and *A-def*: $A = [a,\ b,\ f]_\circ$
and *B-def*: $B = [a',\ b',\ f']_\circ$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
and *h*: $h : b \mapsto_{\mathfrak{B}} b'$
and *f*: $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b|\!)$
and *f'*: $f' : \mathfrak{G}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|ObjMap|\!)(\!|b'|\!)$
and [*cat-cs-simps*]: $f' \circ_{A\mathfrak{C}} \mathfrak{G}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{H}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{C}} f$
by *auto*
with *that*(*1*) *assms* **obtain** $a''\ b''\ f''\ g'\ h'$
where *G-def*: $G = [[a',\ b',\ f']_\circ,\ [a'',\ b'',\ f'']_\circ,\ [g',\ h']_\circ]_\circ$
and *C-def*: $C = [a'',\ b'',\ f'']_\circ$
and *g'*: $g' : a' \mapsto_{\mathfrak{A}} a''$
and *h'*: $h' : b' \mapsto_{\mathfrak{B}} b''$

```
      and f'': f'' : 𝔊(⌊ObjMap⌋)(⌊a''⌋) ↦_ℭ 𝔎(⌊ObjMap⌋)(⌊b''⌋)
      and [cat-cs-simps]: f'' ∘_Aℭ 𝔊(⌊ArrMap⌋)(⌊g'⌋) = 𝔎(⌊ArrMap⌋)(⌊h'⌋) ∘_Aℭ f'
    by auto
  note [cat-cs-simps] =
    category.cat-assoc-helper
      [
        where ℭ=ℭ
          and h=f''
          and g=‹𝔊(⌊ArrMap⌋)(⌊g'⌋)›
          and q=‹𝔎(⌊ArrMap⌋)(⌊h'⌋) ∘_Aℭ f'›
      ]
    category.cat-assoc-helper
      [
        where ℭ=ℭ
          and h=f
          and g=‹𝔎(⌊ArrMap⌋)(⌊h⌋)›
          and q=‹f' ∘_Aℭ 𝔊(⌊ArrMap⌋)(⌊g⌋)›
      ]
  from assms that g g' h h' f f' f'' show ?thesis
    unfolding F-def G-def A-def B-def C-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: cat-comma-cs-intros cat-cs-intros
      )
 qed
 show 𝔊 _CF↑ 𝔎(⌊ArrMap⌋)(⌊𝔊 _CF↓_CF 𝔎(⌊CId⌋)(⌊A⌋)⌋) = 𝔄(⌊CId⌋)(⌊𝔊 _CF↑ 𝔎(⌊ObjMap⌋)(⌊A⌋)⌋)
   if A ∈∘ 𝔊 _CF↓_CF 𝔎(⌊Obj⌋) for A
 proof–
   from assms that obtain a b f
     where A-def: A = [a, b, f]∘
       and a ∈∘ 𝔄(⌊Obj⌋)
       and b ∈∘ 𝔅(⌊Obj⌋)
       and f : 𝔊(⌊ObjMap⌋)(⌊a⌋) ↦_ℭ 𝔎(⌊ObjMap⌋)(⌊b⌋)
     by auto
   from assms that this(2–4) show ?thesis
     unfolding A-def
     by
       (
         cs-concl cs-shallow
           cs-simp: cat-cs-simps cat-comma-cs-simps
           cs-intro: cat-comma-cs-intros cat-cs-intros
       )
 qed
qed
  (
    use assms in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      ›
  )+
qed

lemma cf-comma-proj-left-is-functor'[cat-comma-cs-intros]:
  assumes 𝔊 : 𝔄 ↦↦_Cα ℭ
```

```
    and ℌ : 𝔅 ↦↦_{Cα} ℭ
    and 𝔄' = 𝔊 _{CF↓CF} ℌ
  shows 𝔊 _{CF}⨅ ℌ : 𝔄' ↦↦_{Cα} 𝔄
  using assms(1,2) unfolding assms(3) by (rule cf-comma-proj-left-is-functor)

lemma cf-comma-proj-right-is-functor:
  assumes 𝔊 : 𝔄 ↦↦_{Cα} ℭ and ℌ : 𝔅 ↦↦_{Cα} ℭ
  shows 𝔊 ⨅_{CF} ℌ : 𝔊 _{CF↓CF} ℌ ↦↦_{Cα} 𝔅
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  from assms interpret 𝔊ℌ: category α ‹𝔊 _{CF↓CF} ℌ›
    by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)
  show ?thesis
  proof(rule is-functorI')
    show vfsequence (𝔊 ⨅_{CF} ℌ)
      unfolding cf-comma-proj-right-def by auto
    show vcard (𝔊 ⨅_{CF} ℌ) = ℵ₀
      unfolding cf-comma-proj-right-def by (simp add: nat-omega-simps)
    from assms show ℛ∘ (𝔊 ⨅_{CF} ℌ⦇ObjMap⦈) ⊆∘ 𝔅⦇Obj⦈
      by (rule cf-comma-proj-right-ObjMap-vrange)
    show 𝔊 ⨅_{CF} ℌ⦇ArrMap⦈⦇F⦈ : 𝔊 ⨅_{CF} ℌ⦇ObjMap⦈⦇A⦈ ↦_𝔅 𝔊 ⨅_{CF} ℌ⦇ObjMap⦈⦇B⦈
      if F : A ↦_{𝔊 _{CF↓CF} ℌ} B for A B F
    proof–
      from assms that obtain a b f a' b' f' g h
        where F-def: F = [[a, b, f]∘, [a', b', f']∘, [g, h]∘]∘
          and A-def: A = [a, b, f]∘
          and B-def: B = [a', b', f']∘
          and h: h : b ↦_𝔅 b'
        by auto
      from that h show
        𝔊 ⨅_{CF} ℌ⦇ArrMap⦈⦇F⦈ : 𝔊 ⨅_{CF} ℌ⦇ObjMap⦈⦇A⦈ ↦_𝔅 𝔊 ⨅_{CF} ℌ⦇ObjMap⦈⦇B⦈
        unfolding F-def A-def B-def
        by (cs-concl cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros)
    qed
    show
      𝔊 ⨅_{CF} ℌ⦇ArrMap⦈⦇G ∘_{A𝔊 _{CF↓CF} ℌ} F⦈ =
        𝔊 ⨅_{CF} ℌ⦇ArrMap⦈⦇G⦈ ∘_{A𝔅} 𝔊 ⨅_{CF} ℌ⦇ArrMap⦈⦇F⦈
      if G : B ↦_{𝔊 _{CF↓CF} ℌ} C and F : A ↦_{𝔊 _{CF↓CF} ℌ} B for B C G A F
    proof–
      from assms that(2) obtain a b f a' b' f' g h
        where F-def: F = [[a, b, f]∘, [a', b', f']∘, [g, h]∘]∘
          and A-def: A = [a, b, f]∘
          and B-def: B = [a', b', f']∘
          and g: g : a ↦_𝔄 a'
          and h: h : b ↦_𝔅 b'
          and f: f : 𝔊⦇ObjMap⦈⦇a⦈ ↦_ℭ ℌ⦇ObjMap⦈⦇b⦈
          and f': f' : 𝔊⦇ObjMap⦈⦇a'⦈ ↦_ℭ ℌ⦇ObjMap⦈⦇b'⦈
          and [cat-cs-simps]: f' ∘_{Aℭ} 𝔊⦇ArrMap⦈⦇g⦈ = ℌ⦇ArrMap⦈⦇h⦈ ∘_{Aℭ} f
        by auto
      with that(1) assms obtain a'' b'' f'' g' h'
        where G-def: G = [[a', b', f']∘, [a'', b'', f'']∘, [g', h']∘]∘
          and C-def: C = [a'', b'', f'']∘
          and g': g' : a' ↦_𝔄 a''
          and h': h' : b' ↦_𝔅 b''
          and f'': f'' : 𝔊⦇ObjMap⦈⦇a''⦈ ↦_ℭ ℌ⦇ObjMap⦈⦇b''⦈
          and [cat-cs-simps]: f'' ∘_{Aℭ} 𝔊⦇ArrMap⦈⦇g'⦈ = ℌ⦇ArrMap⦈⦇h'⦈ ∘_{Aℭ} f'
        by auto
```

```
    note [cat-cs-simps] =
      category.cat-assoc-helper
        [
          where 𝔈=𝔈
            and h=f''
            and g=‹𝔊(|ArrMap|)(|g'|)›
            and q=‹ℌ(|ArrMap|)(|h'|) ∘_A𝔈 f'›
        ]
      category.cat-assoc-helper
        [
          where 𝔈=𝔈
            and h=f
            and g=‹ℌ(|ArrMap|)(|h|)›
            and q=‹f' ∘_A𝔈 𝔊(|ArrMap|)(|g|)›
        ]
      from assms that g g' h h' f f' f'' show ?thesis
        unfolding F-def G-def A-def B-def C-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cat-comma-cs-simps
              cs-intro: cat-comma-cs-intros cat-cs-intros
          )
    qed
    show 𝔊 ⨅_CF ℌ(|ArrMap|)(|𝔊 _CF↓_CF ℌ(|CId|)(|A|)|) = 𝔅(|CId|)(|𝔊 ⨅_CF ℌ(|ObjMap|)(|A|)|)
      if A ∈₀ 𝔊 _CF↓_CF ℌ(|Obj|) for A
    proof–
      from assms that obtain a b f
        where A-def: A = [a, b, f]₀
          and a ∈₀ 𝔄(|Obj|)
          and b ∈₀ 𝔅(|Obj|)
          and f : 𝔊(|ObjMap|)(|a|) ↦_𝔈 ℌ(|ObjMap|)(|b|)
        by auto
      from assms that this(2–4) show ?thesis
        unfolding A-def
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps cat-comma-cs-simps
              cs-intro: cat-comma-cs-intros cat-cs-intros
          )
    qed
  qed
    (
      use assms in
        ‹
          cs-concl cs-shallow
            cs-simp: cat-comma-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
        ›
    )+
qed

lemma cf-comma-proj-right-is-functor'[cat-comma-cs-intros]:
  assumes 𝔊 : 𝔄 ↦↦_Cα 𝔈
    and ℌ : 𝔅 ↦↦_Cα 𝔈
    and 𝔄' = 𝔊 _CF↓_CF ℌ
  shows 𝔊 ⨅_CF ℌ : 𝔄' ↦↦_Cα 𝔅
```

using *assms*(*1*,*2*) **unfolding** *assms*(*3*) **by** (*rule cf-comma-proj-right-is-functor*)

14.4.5 Opposite projections for a comma category

lemma *op-cf-comma-proj-left*:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *op-cf* $(\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}) = (\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H}$
proof–
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{H}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}\mathfrak{H}$: *category* $\alpha\ \langle\mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}\rangle$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
show *op-cf* $(\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}) = (\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H}$
proof(*rule cf-eqI*)
show *op-cf* $(\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H}) : \textit{op-cat}\ (\mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}) \mapsto\mapsto_{C\alpha} \textit{op-cat}\ \mathfrak{A}$
by
(
cs-concl **cs-shallow**
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
then have *ObjMap-dom-lhs*: $\mathcal{D}_\circ\ (\textit{op-cf}\ (\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H})(\!|\textit{ObjMap}|\!)) = \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|\textit{Obj}|\!)$
and *ArrMap-dom-lhs*: $\mathcal{D}_\circ\ (\textit{op-cf}\ (\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H})(\!|\textit{ArrMap}|\!)) = \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|\textit{Arr}|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps cat-op-simps*)+
show $(\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H}$:
$\textit{op-cat}\ (\mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}) \mapsto\mapsto_{C\alpha} \textit{op-cat}\ \mathfrak{A}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-comma-cs-intros cat-op-intros*)
then have *ObjMap-dom-rhs*:
$\mathcal{D}_\circ\ (((\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H})(\!|\textit{ObjMap}|\!)) =$
$\mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|\textit{Obj}|\!)$
and *ArrMap-dom-rhs*:
$\mathcal{D}_\circ\ (((\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H})(\!|\textit{ArrMap}|\!)) =$
$\mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|\textit{Arr}|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps cat-op-simps*)+
show
$\textit{op-cf}\ (\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H})(\!|\textit{ObjMap}|\!) =$
$((\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H})(\!|\textit{ObjMap}|\!)$
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
fix A **assume** $A \in_\circ \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}(\!|\textit{Obj}|\!)$
with *assms* **obtain** $a\ b\ f$
where *A-def*: $A = [a, b, f]_\circ$
and *a*: $a \in_\circ \mathfrak{A}(\!|\textit{Obj}|\!)$
and *b*: $b \in_\circ \mathfrak{B}(\!|\textit{Obj}|\!)$
and *f*: $f : \mathfrak{G}(\!|\textit{ObjMap}|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{H}(\!|\textit{ObjMap}|\!)(\!|b|\!)$
by *auto*
from $a\ b\ f$ **show**
$\textit{op-cf}\ (\mathfrak{G}\ {}_{CF}\sqcap\ \mathfrak{H})(\!|\textit{ObjMap}|\!)(\!|A|\!) =$
$((\textit{op-cf}\ \mathfrak{H}) \sqcap_{CF} (\textit{op-cf}\ \mathfrak{G}) \circ_{CF} \textit{op-cf-comma}\ \mathfrak{G}\ \mathfrak{H})(\!|\textit{ObjMap}|\!)(\!|A|\!)$
unfolding *A-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-comma-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
qed
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps*

```
      cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
    )+
  show
    op-cf (𝔊 CF⨅ 𝔎)(⦇ArrMap⦈) =
      ((op-cf 𝔎) ⨅CF (op-cf 𝔊) ∘CF op-cf-comma 𝔊 𝔎)(⦇ArrMap⦈)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix ABF assume ABF ∈∘ 𝔊 CF↓CF 𝔎(⦇Arr⦈)
    then obtain A B where ABF : A ↦𝔊 CF↓CF 𝔎 B by auto
    with assms obtain a b f a′ b′ f′ g h
      where ABF-def: ABF = [[a, b, f]∘, [a′, b′, f′]∘, [g, h]∘]∘
        and A-def: A = [a, b, f]∘
        and B-def: B = [a′, b′, f′]∘
        and g: g : a ↦𝔄 a′
        and h: h : b ↦𝔅 b′
        and f: f : 𝔊(⦇ObjMap⦈)(⦇a⦈) ↦ℭ 𝔎(⦇ObjMap⦈)(⦇b⦈)
        and f′: f′ : 𝔊(⦇ObjMap⦈)(⦇a′⦈) ↦ℭ 𝔎(⦇ObjMap⦈)(⦇b′⦈)
        and [symmetric, cat-cs-simps]:
          f′ ∘A ℭ 𝔊(⦇ArrMap⦈)(⦇g⦈) = 𝔎(⦇ArrMap⦈)(⦇h⦈) ∘A ℭ f
      by auto
    from g h f f′ show op-cf (𝔊 CF⨅ 𝔎)(⦇ArrMap⦈)(⦇ABF⦈) =
      ((op-cf 𝔎) ⨅CF (op-cf 𝔊) ∘CF op-cf-comma 𝔊 𝔎)(⦇ArrMap⦈)(⦇ABF⦈)
      unfolding ABF-def A-def B-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
        )
  qed
    (
      cs-concl cs-shallow
        cs-simp: cat-op-simps
        cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
    )+
qed simp-all
qed

lemma op-cf-comma-proj-right:
  assumes 𝔊 : 𝔄 ↦↦Cα ℭ and 𝔎 : 𝔅 ↦↦Cα ℭ
  shows op-cf (𝔊 ⨅CF 𝔎) = (op-cf 𝔎) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 𝔎
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(2))
  interpret 𝔊𝔎: category α ‹𝔊 CF↓CF 𝔎›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
  show op-cf (𝔊 ⨅CF 𝔎) = (op-cf 𝔎) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 𝔎
  proof(rule cf-eqI)
    show op-cf (𝔊 ⨅CF 𝔎) : op-cat (𝔊 CF↓CF 𝔎) ↦↦Cα op-cat 𝔅
      by
        (
          cs-concl cs-shallow
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
        )
    then have ObjMap-dom-lhs: 𝒟∘ (op-cf (𝔊 ⨅CF 𝔎)(⦇ObjMap⦈)) = 𝔊 CF↓CF 𝔎(⦇Obj⦈)
      and ArrMap-dom-lhs: 𝒟∘ (op-cf (𝔊 ⨅CF 𝔎)(⦇ArrMap⦈)) = 𝔊 CF↓CF 𝔎(⦇Arr⦈)
      by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cat-op-simps)+
    show (op-cf 𝔎) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 𝔎 :
      op-cat (𝔊 CF↓CF 𝔎) ↦↦Cα op-cat 𝔅
```

```
  by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros)
then have ObjMap-dom-rhs:
  𝒟ₒ (((op-cf ℌ) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 ℌ)(⌊ObjMap⌋)) =
    𝔊 CF↓CF ℌ(⌊Obj⌋)
  and ArrMap-dom-rhs:
    𝒟ₒ (((op-cf ℌ) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 ℌ)(⌊ArrMap⌋)) =
      𝔊 CF↓CF ℌ(⌊Arr⌋)
  by (cs-concl cs-simp: cat-cs-simps cat-op-simps)+
show
  op-cf (𝔊 ⨅CF ℌ)(⌊ObjMap⌋) =
    ((op-cf ℌ) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 ℌ)(⌊ObjMap⌋)
proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
  fix A assume prems: A ∈ₒ 𝔊 CF↓CF ℌ(⌊Obj⌋)
  with assms obtain a b f
    where A-def: A = [a, b, f]ₒ
      and a: a ∈ₒ 𝔄(⌊Obj⌋)
      and b: b ∈ₒ 𝔅(⌊Obj⌋)
      and f: f : 𝔊(⌊ObjMap⌋)(⌊a⌋) ↦ℭ ℌ(⌊ObjMap⌋)(⌊b⌋)
    by auto
  from a b f show
    op-cf (𝔊 ⨅CF ℌ)(⌊ObjMap⌋)(⌊A⌋) =
      ((op-cf ℌ) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 ℌ)(⌊ObjMap⌋)(⌊A⌋)
    unfolding A-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
      )
qed
  (
    cs-concl cs-shallow
      cs-simp: cat-op-simps
      cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
  )+
show
  op-cf (𝔊 ⨅CF ℌ)(⌊ArrMap⌋) =
    ((op-cf ℌ) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 ℌ)(⌊ArrMap⌋)
proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
  fix ABF assume prems: ABF ∈ₒ 𝔊 CF↓CF ℌ(⌊Arr⌋)
  then obtain A B where ABF: ABF : A ↦𝔊 CF↓CF ℌ B by auto
  with assms obtain a b f a′ b′ f′ g h
    where ABF-def: ABF = [[a, b, f]ₒ, [a′, b′, f′]ₒ, [g, h]ₒ]ₒ
      and A-def: A = [a, b, f]ₒ
      and B-def: B = [a′, b′, f′]ₒ
      and g: g : a ↦𝔄 a′
      and h: h : b ↦𝔅 b′
      and f: f : 𝔊(⌊ObjMap⌋)(⌊a⌋) ↦ℭ ℌ(⌊ObjMap⌋)(⌊b⌋)
      and f′: f′ : 𝔊(⌊ObjMap⌋)(⌊a′⌋) ↦ℭ ℌ(⌊ObjMap⌋)(⌊b′⌋)
      and [symmetric, cat-cs-simps]:
        f′ ∘A ℭ 𝔊(⌊ArrMap⌋)(⌊g⌋) = ℌ(⌊ArrMap⌋)(⌊h⌋) ∘A ℭ f
    by auto
  from g h f f′ show op-cf (𝔊 ⨅CF ℌ)(⌊ArrMap⌋)(⌊ABF⌋) =
    ((op-cf ℌ) CF⨅ (op-cf 𝔊) ∘CF op-cf-comma 𝔊 ℌ)(⌊ArrMap⌋)(⌊ABF⌋)
    unfolding ABF-def A-def B-def
    by
      (
        cs-concl
```

```
            cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
        )
    qed
      (
        cs-concl cs-shallow
          cs-simp: cat-op-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
      )+
  qed simp-all
qed
```

14.4.6 Projections for a tiny comma category

```
lemma cf-comma-proj-left-is-tm-functor:
  assumes 𝔊 : 𝔄 ↦↦_{C.tm α} ℭ and ℌ : 𝔅 ↦↦_{C.tm α} ℭ
  shows 𝔊 _{CF}∏ ℌ : 𝔊 _{CF}↓_{CF} ℌ ↦↦_{C.tm α} 𝔄
proof(intro is-tm-functorI′)

  interpret 𝔊: is-tm-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  interpret ℌ: is-tm-functor α 𝔅 ℭ ℌ by (rule assms(2))

  show Π-𝔊ℌ: 𝔊 _{CF}∏ ℌ : 𝔊 _{CF}↓_{CF} ℌ ↦↦_{Cα} 𝔄
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)

  interpret Π-𝔊ℌ: is-functor α ‹𝔊 _{CF}↓_{CF} ℌ› 𝔄 ‹𝔊 _{CF}∏ ℌ›
    by (rule Π-𝔊ℌ)
  interpret 𝔊ℌ: tiny-category α ‹𝔊 _{CF}↓_{CF} ℌ›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-comma-cs-intros)

  show 𝔊 _{CF}∏ ℌ(|ObjMap|) ∈∘ Vset α
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    show ℛ∘ (𝔊 _{CF}∏ ℌ(|ObjMap|)) ∈∘ Vset α
    proof–
      note Π-𝔊ℌ.cf-ObjMap-vrange
      moreover have 𝔄(|Obj|) ∈∘ Vset α by (intro cat-small-cs-intros)
      ultimately show ?thesis by auto
    qed
  qed (auto simp: cf-comma-proj-left-components intro: cat-small-cs-intros)

  show 𝔊 _{CF}∏ ℌ(|ArrMap|) ∈∘ Vset α
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    show ℛ∘ (𝔊 _{CF}∏ ℌ(|ArrMap|)) ∈∘ Vset α
    proof–
      note Π-𝔊ℌ.cf-ArrMap-vrange
      moreover have 𝔄(|Arr|) ∈∘ Vset α by (intro cat-small-cs-intros)
      ultimately show ?thesis by auto
    qed
  qed (auto simp: cf-comma-proj-left-components intro: cat-small-cs-intros)

qed

lemma cf-comma-proj-left-is-tm-functor′[cat-comma-cs-intros]:
  assumes 𝔊 : 𝔄 ↦↦_{C.tm α} ℭ
    and ℌ : 𝔅 ↦↦_{C.tm α} ℭ
    and 𝔊ℌ = 𝔊 _{CF}↓_{CF} ℌ
  shows 𝔊 _{CF}∏ ℌ : 𝔊ℌ ↦↦_{C.tm α} 𝔄
  using assms(1,2) unfolding assms(3) by (rule cf-comma-proj-left-is-tm-functor)
```

lemma *cf-comma-proj-right-is-tm-functor*:
  assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
  shows $\mathfrak{G} \prod_{CF} \mathfrak{H} : \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof(*intro is-tm-functorI'*)

  interpret $\mathfrak{G}$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{H}$: *is-tm-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*2*))

  show Π-$\mathfrak{G}\mathfrak{H}$: $\mathfrak{G} \prod_{CF} \mathfrak{H} : \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-comma-cs-intros*)

  interpret Π-$\mathfrak{G}\mathfrak{H}$: *is-functor* α $\langle \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H} \rangle$ $\mathfrak{B}$ $\langle \mathfrak{G} \prod_{CF} \mathfrak{H} \rangle$
    by (*rule* Π-$\mathfrak{G}\mathfrak{H}$)
  interpret $\mathfrak{G}\mathfrak{H}$: *tiny-category* α $\langle \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H} \rangle$
    by (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-comma-cs-intros*)

  show $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha$
  proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
    show $\mathcal{R}_{\circ}\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!)) \in_{\circ} Vset\ \alpha$
    proof–
      note Π-$\mathfrak{G}\mathfrak{H}$.*cf-ObjMap-vrange*
      moreover have $\mathfrak{B}(\!|Obj|\!) \in_{\circ} Vset\ \alpha$ **by** (*intro cat-small-cs-intros*)
      ultimately show *?thesis* **by** *auto*
    qed
  qed (*auto simp*: *cf-comma-proj-right-components intro*: *cat-small-cs-intros*)

  show $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!) \in_{\circ} Vset\ \alpha$
  proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
    show $\mathcal{R}_{\circ}\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!)) \in_{\circ} Vset\ \alpha$
    proof–
      note Π-$\mathfrak{G}\mathfrak{H}$.*cf-ArrMap-vrange*
      moreover have $\mathfrak{B}(\!|Arr|\!) \in_{\circ} Vset\ \alpha$ **by** (*intro cat-small-cs-intros*)
      ultimately show *?thesis* **by** *auto*
    qed
  qed (*auto simp*: *cf-comma-proj-right-components intro*: *cat-small-cs-intros*)

qed

lemma *cf-comma-proj-right-is-tm-functor'*[*cat-comma-cs-intros*]:
  assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
    and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
    and $\mathfrak{G}\mathfrak{H} = \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}$
  shows $\mathfrak{G} \prod_{CF} \mathfrak{H} : \mathfrak{G}\mathfrak{H} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
  using *assms*(*1,2*) **unfolding** *assms*(*3*) **by** (*rule cf-comma-proj-right-is-tm-functor*)

lemma *cf-comp-cf-comma-proj-left-is-tm-functor*[*cat-comma-cs-intros*]:
  assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H}$
  shows $\mathfrak{G}\ {}_{CF}{\prod}\ \mathfrak{H} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{A}$
proof(*intro is-tm-functorI'*)

  interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*2*))
  interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\langle \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H} \rangle$ $\mathfrak{F}$ **by** (*rule assms*(*3*))
  interpret $\mathfrak{G}\mathfrak{H}$: *is-functor* α $\langle \mathfrak{G}\ {}_{CF}{\downarrow}_{CF}\ \mathfrak{H} \rangle$ $\mathfrak{A}$ $\langle \mathfrak{G}\ {}_{CF}{\prod}\ \mathfrak{H} \rangle$

```
  by (rule cf-comma-proj-left-is-functor[OF assms(1–2)])

show 𝔊 _CF⨅ ℌ ∘_CF 𝔉 : 𝔍 ↦↦_Cα 𝔄
  by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)

show (𝔊 _CF⨅ ℌ ∘_CF 𝔉)⦇ObjMap⦈ ∈∘ Vset α
  unfolding dghm-comp-components
proof(rule vbrelation.vbrelation-Limit-in-VsetI)
  show vbrelation (𝔊 _CF⨅ ℌ⦇ObjMap⦈ ∘∘ 𝔉⦇ObjMap⦈) by auto
  show Limit α by auto
  show 𝒟∘ (𝔊 _CF⨅ ℌ⦇ObjMap⦈ ∘∘ 𝔉⦇ObjMap⦈) ∈∘ Vset α
    by
      (
        cs-concl
          cs-simp: V-cs-simps cat-cs-simps
          cs-intro: 𝔉.cf-ObjMap-vrange cat-small-cs-intros
      )
  show ℛ∘ (𝔊 _CF⨅ ℌ⦇ObjMap⦈ ∘∘ 𝔉⦇ObjMap⦈) ∈∘ Vset α
    unfolding vrange-vcomp
  proof–
    have 𝔊 _CF⨅ ℌ⦇ObjMap⦈ ‘∘ ℛ∘ (𝔉⦇ObjMap⦈) ⊆∘ ⋃∘(⋃∘(⋃∘(ℛ∘ (𝔉⦇ObjMap⦈))))
    proof(intro vsubsetI)
      fix A assume prems: A ∈∘ 𝔊 _CF⨅ ℌ⦇ObjMap⦈ ‘∘ ℛ∘ (𝔉⦇ObjMap⦈)
      then obtain abf
        where abf-in-𝔉: abf ∈∘ ℛ∘ (𝔉⦇ObjMap⦈)
          and 𝔊ℌ-abf: 𝔊 _CF⨅ ℌ⦇ObjMap⦈⦇abf⦈ = A
        by auto
      with 𝔉.ObjMap.vrange-atD obtain j
        where j ∈∘ 𝔍⦇Obj⦈ and 𝔉j: 𝔉⦇ObjMap⦈⦇j⦈ = abf
        by (force simp: 𝔉.cf-ObjMap-vdomain)
      from abf-in-𝔉 𝔉.cf-ObjMap-vrange have abf-in-𝔊ℌ:
        abf ∈∘ 𝔊 _CF↓_CF ℌ⦇Obj⦈
        by auto
      then obtain a b f where abf-def: abf = [a, b, f]∘
        by (elim cat-comma-ObjE[OF - assms(1,2)])
      have a ∈∘ ⋃∘(⋃∘(⋃∘(ℛ∘ (𝔉⦇ObjMap⦈))))
      proof(intro VUnionI)
        from abf-in-𝔉 show [a, b, f]∘ ∈∘ ℛ∘ (𝔉⦇ObjMap⦈)
          unfolding abf-def by auto
        show ⟨0, a⟩ ∈∘ [a, b, f]∘ by auto
        show set {0, a} ∈∘ ⟨0, a⟩ unfolding vpair-def by simp
      qed auto
      with abf-in-𝔊ℌ show A ∈∘ ⋃∘(⋃∘(⋃∘(ℛ∘(𝔉⦇ObjMap⦈))))
        unfolding 𝔊ℌ-abf[symmetric] abf-def
        by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps)
    qed
    moreover have ⋃∘(⋃∘(⋃∘(ℛ∘ (𝔉⦇ObjMap⦈)))) ∈∘ Vset α
      by (intro VUnion-in-VsetI vrange-in-VsetI[OF 𝔉.tm-cf-ObjMap-in-Vset])
    ultimately show 𝔊 _CF⨅ ℌ⦇ObjMap⦈ ‘∘ ℛ∘ (𝔉⦇ObjMap⦈) ∈∘ Vset α by auto
  qed
qed

show (𝔊 _CF⨅ ℌ ∘_CF 𝔉)⦇ArrMap⦈ ∈∘ Vset α
  unfolding dghm-comp-components
proof(rule vbrelation.vbrelation-Limit-in-VsetI)
  show vbrelation (𝔊 _CF⨅ ℌ⦇ArrMap⦈ ∘∘ 𝔉⦇ArrMap⦈) by auto
  show Limit α by auto
  show 𝒟∘ (𝔊 _CF⨅ ℌ⦇ArrMap⦈ ∘∘ 𝔉⦇ArrMap⦈) ∈∘ Vset α
```

by
(
cs-concl
cs-simp: *V-cs-simps cat-cs-simps*
cs-intro: $\mathfrak{F}$.*cf-ArrMap-vrange cat-small-cs-intros*
)
show $\mathcal{R}_\circ$ ($\mathfrak{G}$ $_{CF}\sqcap$ $\mathfrak{H}(\!|ArrMap|\!)$ $\circ_\circ$ $\mathfrak{F}(\!|ArrMap|\!)$) $\in_\circ$ *Vset* α
unfolding *vrange-vcomp*
proof–
have
$\mathfrak{G}$ $_{CF}\sqcap$ $\mathfrak{H}(\!|ArrMap|\!)$ `$_\circ$ $\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$) $\subseteq_\circ$
$\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$))))))
proof(*intro vsubsetI*)
fix F **assume** *prems*: $F \in_\circ$ $\mathfrak{G}$ $_{CF}\sqcap$ $\mathfrak{H}(\!|ArrMap|\!)$ `$_\circ$ $\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$)
then obtain *ABF*
where *ABF-in-*$\mathfrak{F}$: *ABF* $\in_\circ$ $\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$)
and $\mathfrak{GH}$-*ABF*: $\mathfrak{G}$ $_{CF}\sqcap$ $\mathfrak{H}(\!|ArrMap|\!)(\!|ABF|\!) = F$
by *auto*
with $\mathfrak{F}$.*ArrMap.vrange-atD* **obtain** k
where $k \in_\circ \mathfrak{J}(\!|Arr|\!)$ **and** $\mathfrak{F}j$: $\mathfrak{F}(\!|ArrMap|\!)(\!|k|\!) = ABF$
by (*force simp*: $\mathfrak{F}$.*cf-ArrMap-vdomain*)
then obtain i j **where** $k : i \mapsto_{\mathfrak{J}} j$ **by** *auto*
from *ABF-in-*$\mathfrak{F}$ $\mathfrak{F}$.*cf-ArrMap-vrange* **have** *ABF-in-*$\mathfrak{GH}$:
ABF $\in_\circ$ $\mathfrak{G}$ $_{CF}\downarrow_{CF}$ $\mathfrak{H}(\!|Arr|\!)$
by *auto*
then obtain A B **where** $ABF : A \mapsto_{\mathfrak{G}}$ $_{CF}\downarrow_{CF}$ $\mathfrak{H}$ B **by** *auto*
with *assms* **obtain** a b f a' b' f' g h
where *ABF-def*: $ABF = [[a, b, f]_\circ, [a', b', f']_\circ, [g, h]_\circ]_\circ$
by (*elim cat-comma-is-arrE*[*OF - assms*(*1,2*)])
have $g \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$))))))
proof(*intro VUnionI*)
from *ABF-in-*$\mathfrak{F}$ **show**
$[[a, b, f]_\circ, [a', b', f']_\circ, [g, h]_\circ]_\circ \in_\circ \mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$)
unfolding *ABF-def* **by** *auto*
show $\langle 2_{\mathbb{N}}, [g, h]_\circ\rangle \in_\circ [[a, b, f]_\circ, [a', b', f']_\circ, [g, h]_\circ]_\circ$
by (*auto simp*: *nat-omega-simps*)
show *set* $\{2_{\mathbb{N}}, [g, h]_\circ\} \in_\circ \langle 2_{\mathbb{N}}, [g, h]_\circ\rangle$
unfolding *vpair-def* **by** *auto*
show $[g, h]_\circ \in_\circ$ *set* $\{2_{\mathbb{N}}, [g, h]_\circ\}$ **by** *simp*
show $\langle 0, g\rangle \in_\circ [g, h]_\circ$ **by** *auto*
show *set* $\{0, g\} \in_\circ \langle 0, g\rangle$ **unfolding** *vpair-def* **by** *auto*
qed *auto*
with *ABF-in-*$\mathfrak{GH}$ **show** $F \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$))))))
unfolding $\mathfrak{GH}$-*ABF*[*symmetric*] *ABF-def*
by (*cs-concl* **cs-simp**: *cat-cs-simps cat-comma-cs-simps*)
qed
moreover have $\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$))))))) $\in_\circ$ *Vset* α
by (*intro VUnion-in-VsetI vrange-in-VsetI*[*OF* $\mathfrak{F}$.*tm-cf-ArrMap-in-Vset*])
ultimately show $\mathfrak{G}$ $_{CF}\sqcap$ $\mathfrak{H}(\!|ArrMap|\!)$ `$_\circ$ $\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$) $\in_\circ$ *Vset* α **by** *auto*
qed
qed

qed

lemma *cf-comp-cf-comma-proj-right-is-tm-functor*[*cat-comma-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha}$ $\mathfrak{G}$ $_{CF}\downarrow_{CF}$ $\mathfrak{H}$

shows $\mathfrak{G} \prod_{CF} \mathfrak{H} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof(*intro is-tm-functorI'*)

interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{H}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ ‹$\mathfrak{G}$ $_{CF}\downarrow_{CF}$ $\mathfrak{H}$› $\mathfrak{F}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{G}\mathfrak{H}$: *is-functor* α ‹$\mathfrak{G}$ $_{CF}\downarrow_{CF}$ $\mathfrak{H}$› $\mathfrak{B}$ ‹$\mathfrak{G} \prod_{CF} \mathfrak{H}$›
by (*rule cf-comma-proj-right-is-functor*[*OF assms*(*1–2*)])

show $\mathfrak{G} \prod_{CF} \mathfrak{H} \circ_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*)

show $(\mathfrak{G} \prod_{CF} \mathfrak{H} \circ_{CF} \mathfrak{F})(\!|ObjMap|\!) \in_{\circ} Vset\ \alpha$
unfolding *dghm-comp-components*
proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
show *vbrelation* $(\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!) \circ_{\circ} \mathfrak{F}(\!|ObjMap|\!))$ **by** *auto*
show *Limit* α **by** *auto*
show $\mathcal{D}_{\circ}\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!) \circ_{\circ} \mathfrak{F}(\!|ObjMap|\!)) \in_{\circ} Vset\ \alpha$
by
(
cs-concl
cs-simp: *V-cs-simps cat-cs-simps*
cs-intro: $\mathfrak{F}$.*cf-ObjMap-vrange cat-small-cs-intros*
)
show $\mathcal{R}_{\circ}\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!) \circ_{\circ} \mathfrak{F}(\!|ObjMap|\!)) \in_{\circ} Vset\ \alpha$
unfolding *vrange-vcomp*
proof–
have $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!)$ ‘$_{\circ}$ $\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!)) \subseteq_{\circ} \bigcup_{\circ}(\bigcup_{\circ}(\bigcup_{\circ}(\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!)))))$
proof(*intro vsubsetI*)
fix *A* **assume** *prems*: $A \in_{\circ} (\mathfrak{G} \prod_{CF} \mathfrak{H})(\!|ObjMap|\!)$ ‘$_{\circ}$ $\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!))$
then obtain *abf*
where *abf-in-*$\mathfrak{F}$: $abf \in_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!))$
and $\mathfrak{G}\mathfrak{H}$*-abf*: $(\mathfrak{G} \prod_{CF} \mathfrak{H})(\!|ObjMap|\!)(\!|abf|\!) = A$
by (*auto simp*: *cf-comma-proj-right-ObjMap-vsv*)
with $\mathfrak{F}$.*ObjMap.vrange-atD* **obtain** *j*
where $j \in_{\circ} \mathfrak{J}(\!|Obj|\!)$ **and** $\mathfrak{F}j$: $\mathfrak{F}(\!|ObjMap|\!)(\!|j|\!) = abf$
by (*force simp*: $\mathfrak{F}$.*cf-ObjMap-vdomain*)
from *abf-in-*$\mathfrak{F}$ $\mathfrak{F}$.*cf-ObjMap-vrange* **have** *abf-in-*$\mathfrak{G}\mathfrak{H}$:
$abf \in_{\circ} \mathfrak{G}\ _{CF}\downarrow_{CF} \mathfrak{H}(\!|Obj|\!)$
by *auto*
then obtain *a b f* **where** *abf-def*: $abf = [a, b, f]_{\circ}$
by (*elim cat-comma-ObjE*[*OF - assms*(*1,2*)])
have $b \in_{\circ} \bigcup_{\circ}(\bigcup_{\circ}(\bigcup_{\circ}(\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!)))))$
proof(*intro VUnionI*)
from *abf-in-*$\mathfrak{F}$ **show** $[a, b, f]_{\circ} \in_{\circ} \mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!))$
unfolding *abf-def* **by** *auto*
show $\langle 1_{\mathbb{N}}, b\rangle \in_{\circ} [a, b, f]_{\circ}$ **by** (*auto simp*: *nat-omega-simps*)
show *set* $\{1_{\mathbb{N}}, b\} \in_{\circ} \langle 1_{\mathbb{N}}, b\rangle$ **unfolding** *vpair-def* **by** *simp*
qed *auto*
with *abf-in-*$\mathfrak{G}\mathfrak{H}$ **show** $A \in_{\circ} \bigcup_{\circ}(\bigcup_{\circ}(\bigcup_{\circ}(\mathcal{R}_{\circ}(\mathfrak{F}(\!|ObjMap|\!)))))$
unfolding $\mathfrak{G}\mathfrak{H}$*-abf*[*symmetric*] *abf-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
qed
moreover have $\bigcup_{\circ}(\bigcup_{\circ}(\bigcup_{\circ}(\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!))))) \in_{\circ} Vset\ \alpha$
by (*intro VUnion-in-VsetI vrange-in-VsetI*[*OF* $\mathfrak{F}$.*tm-cf-ObjMap-in-Vset*])
ultimately show $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ObjMap|\!)$ ‘$_{\circ}$ $\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!)) \in_{\circ} Vset\ \alpha$ **by** *auto*
qed
qed

show $(\mathfrak{G} \prod_{CF} \mathfrak{H} \circ_{CF} \mathfrak{F})(\!|ArrMap|\!) \in_\circ Vset\ \alpha$
unfolding *dghm-comp-components*
proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
show *vbrelation* $(\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!) \circ_\circ \mathfrak{F}(\!|ArrMap|\!))$ **by** *auto*
show *Limit* α **by** *auto*
show $\mathcal{D}_\circ\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!) \circ_\circ \mathfrak{F}(\!|ArrMap|\!)) \in_\circ Vset\ \alpha$
by
(
cs-concl
cs-simp: *V-cs-simps cat-cs-simps*
cs-intro: $\mathfrak{F}$*.cf-ArrMap-vrange cat-small-cs-intros*
)
show $\mathcal{R}_\circ\ (\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!) \circ_\circ \mathfrak{F}(\!|ArrMap|\!)) \in_\circ Vset\ \alpha$
unfolding *vrange-vcomp*
proof–
have
$\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!)$ '$_\circ\ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!)) \subseteq_\circ$
$\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))))))))$
proof(*intro vsubsetI*)
fix F **assume** *prems*: $F \in_\circ \mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!)$ '$_\circ\ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))$
then obtain *ABF*
where *ABF-in-*$\mathfrak{F}$: $ABF \in_\circ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))$
and $\mathfrak{GH}$*-ABF*: $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!)(\!|ABF|\!) = F$
by (*auto simp*: *cf-comma-proj-right-ArrMap-vsv*)
with $\mathfrak{F}$*.ArrMap.vrange-atD* **obtain** k
where $k \in_\circ \mathfrak{J}(\!|Arr|\!)$ **and** $\mathfrak{F}j$: $\mathfrak{F}(\!|ArrMap|\!)(\!|k|\!) = ABF$
by (*force simp*: $\mathfrak{F}$*.cf-ArrMap-vdomain*)
then obtain $i\ j$ **where** $k : i \mapsto_{\mathfrak{J}} j$ **by** *auto*
from *ABF-in-*$\mathfrak{F}$ $\mathfrak{F}$*.cf-ArrMap-vrange* **have** *ABF-in-*$\mathfrak{GH}$:
$ABF \in_\circ \mathfrak{G}\ _{CF\downarrow CF}\ \mathfrak{H}(\!|Arr|\!)$
by *auto*
then obtain $A\ B$ **where** $ABF : A \mapsto_{\mathfrak{G}\ CF\downarrow CF\ \mathfrak{H}} B$ **by** *auto*
with *assms* **obtain** $a\ b\ f\ a'\ b'\ f'\ g\ h$
where *ABF-def*: $ABF = [[a,\ b,\ f]_\circ,\ [a',\ b',\ f']_\circ,\ [g,\ h]_\circ]_\circ$
by (*elim cat-comma-is-arrE*[*OF - assms*(*1,2*)])
have $h \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))))))))$
proof(*intro VUnionI*)
from *ABF-in-*$\mathfrak{F}$ **show**
$[[a,\ b,\ f]_\circ,\ [a',\ b',\ f']_\circ,\ [g,\ h]_\circ]_\circ \in_\circ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))$
unfolding *ABF-def* **by** *auto*
show $\langle 2_{\mathbb{N}},\ [g,\ h]_\circ\rangle \in_\circ [[a,\ b,\ f]_\circ,\ [a',\ b',\ f']_\circ,\ [g,\ h]_\circ]_\circ$
by (*auto simp*: *nat-omega-simps*)
show *set* $\{2_{\mathbb{N}},\ [g,\ h]_\circ\} \in_\circ \langle 2_{\mathbb{N}},\ [g,\ h]_\circ\rangle$
unfolding *vpair-def* **by** *auto*
show $[g,\ h]_\circ \in_\circ$ *set* $\{2_{\mathbb{N}},\ [g,\ h]_\circ\}$ **by** *simp*
show $\langle 1_{\mathbb{N}},\ h\rangle \in_\circ [g,\ h]_\circ$ **by** (*auto simp*: *nat-omega-simps*)
show *set* $\{1_{\mathbb{N}},\ h\} \in_\circ \langle 1_{\mathbb{N}},\ h\rangle$ **unfolding** *vpair-def* **by** *auto*
qed *auto*
with *ABF-in-*$\mathfrak{GH}$ **show** $F \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))))))))$
unfolding $\mathfrak{GH}$*-ABF*[*symmetric*] *ABF-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps cat-comma-cs-simps*)
qed
moreover have $\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!))))))))) \in_\circ Vset\ \alpha$
by (*intro VUnion-in-VsetI vrange-in-VsetI*[*OF* $\mathfrak{F}$*.tm-cf-ArrMap-in-Vset*])
ultimately show $\mathfrak{G} \prod_{CF} \mathfrak{H}(\!|ArrMap|\!)$ '$_\circ\ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!)) \in_\circ Vset\ \alpha$ **by** *auto*
qed
qed

qed

14.5 Comma categories constructed from a functor and an object

14.5.1 Definitions and elementary properties

See Chapter II-6 in [7].

definition *cat-cf-obj-comma* :: $V \Rightarrow V \Rightarrow V$ ($\langle(-\ _{CF}\!\downarrow\ -)\rangle$ [*1000*, *1000*] *999*)
  where $\mathfrak{F}\ _{CF}\!\downarrow\ b \equiv \mathfrak{F}\ _{CF}\!\downarrow_{CF}$ (*cf-const* (*cat-1 0 0*) ($\mathfrak{F}(\!|$*HomCod*$|\!)$) *b*)

definition *cat-obj-cf-comma* :: $V \Rightarrow V \Rightarrow V$ ($\langle(-\ \downarrow_{CF}\ -)\rangle$ [*1000*, *1000*] *999*)
  where $b \downarrow_{CF} \mathfrak{F} \equiv$ (*cf-const* (*cat-1 0 0*) ($\mathfrak{F}(\!|$*HomCod*$|\!)$) *b*) $_{CF}\!\downarrow_{CF}\ \mathfrak{F}$

Alternative forms of the definitions.

lemma (**in** *is-functor*) *cat-cf-obj-comma-def*:
  $\mathfrak{F}\ _{CF}\!\downarrow\ b = \mathfrak{F}\ _{CF}\!\downarrow_{CF}$ (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ *b*)
  unfolding *cat-cf-obj-comma-def cf-HomCod* **..**

lemma (**in** *is-functor*) *cat-obj-cf-comma-def*:
  $b \downarrow_{CF} \mathfrak{F}$ = (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ *b*) $_{CF}\!\downarrow_{CF}\ \mathfrak{F}$
  unfolding *cat-obj-cf-comma-def cf-HomCod* **..**

Size.

lemma *small-cat-cf-obj-comma-Obj*[*simp*]:
  small $\{[a, 0, f]_\circ \mid a\ f.\ a \in_\circ \mathfrak{A}(\!|Obj|\!) \wedge f : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)\}$
  (**is** ‹*small ?afs*›)
proof–
  define *Q* **where**
    Q i = (*if i = 0 then* $\mathfrak{A}(\!|Obj|\!)$ *else if i =* $1_{\mathbb{N}}$ *then set* $\{0\}$ *else* $\mathfrak{C}(\!|Arr|\!)$)
    for *i*
  have *?afs* $\subseteq$ *elts* ($\prod_\circ i \in_\circ$ *set* $\{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}$. *Q i*)
    unfolding *Q-def*
  proof
    (
      intro subsetI,
      unfold mem-Collect-eq,
      elim exE conjE,
      intro vproductI;
      simp only:
)
    fix *a f* **show** $\mathcal{D}_\circ\ [a, 0, f]_\circ$ = *set* $\{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}$
      by (*simp add*: *three nat-omega-simps*)
  qed (*force simp*: *nat-omega-simps*)+
  then show *small ?afs* **by** (*rule down*)
qed

lemma *small-cat-obj-cf-comma-Obj*[*simp*]:
  small $\{[0, b, f]_\circ \mid b\ f.\ b \in_\circ \mathfrak{B}(\!|Obj|\!) \wedge f : x \mapsto_{\mathfrak{C}} \mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)\}$
  (**is** ‹*small ?bfs*›)
proof–
  define *Q* **where**
    Q i = (*if i = 0 then set* $\{0\}$ *else if i =* $1_{\mathbb{N}}$ *then* $\mathfrak{B}(\!|Obj|\!)$ *else* $\mathfrak{C}(\!|Arr|\!)$)
    for *i*
  have *?bfs* $\subseteq$ *elts* ($\prod_\circ i \in_\circ$ *set* $\{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}$. *Q i*)
    unfolding *Q-def*
  proof

(
intro subsetI,
unfold mem-Collect-eq,
elim exE conjE,
intro vproductI;
simp only:
)
fix *a b f* **show** $\mathcal{D}_\circ\ [0,\ b,\ f]_\circ = set\ \{0,\ 1_{\mathbb{N}},\ 2_{\mathbb{N}}\}$
by (*simp add*: *three nat-omega-simps*)
qed (*force simp*: *nat-omega-simps*)+
then show *small ?bfs* **by** (*rule down*)
qed

14.5.2 Objects

lemma (**in** *is-functor*) *cat-cf-obj-comma-ObjI*[*cat-comma-cs-intros*]:
assumes $A = [a,\ 0,\ f]_\circ$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} b$
shows $A \in_\circ \mathfrak{F}_{\ CF}{\downarrow}\ b(\!|Obj|\!)$
using *assms*(*2*,*3*)
unfolding *assms*(*1*)
by
(
cs-concl
cs-simp: *cat-cs-simps cat-cf-obj-comma-def*
cs-intro: *cat-cs-intros vempty-is-zet cat-comma-ObjI*
)

lemmas [*cat-comma-cs-intros*] = *is-functor.cat-cf-obj-comma-ObjI*

lemma (**in** *is-functor*) *cat-obj-cf-comma-ObjI*[*cat-comma-cs-intros*]:
assumes $A = [0,\ a,\ f]_\circ$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
shows $A \in_\circ b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
using *assms*(*2*,*3*)
unfolding *assms*(*1*)
by
(
cs-concl
cs-simp: *cat-cs-simps cat-obj-cf-comma-def*
cs-intro: *vempty-is-zet cat-cs-intros cat-comma-ObjI*
)

lemmas [*cat-comma-cs-intros*] = *is-functor.cat-obj-cf-comma-ObjI*

lemma (**in** *is-functor*) *cat-cf-obj-comma-ObjD*[*dest*]:
assumes $[a,\ b',\ f]_\circ \in_\circ \mathfrak{F}_{\ CF}{\downarrow}\ b(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $b' = 0$ **and** $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} b$
proof–
from *assms*(*2*) **have** *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
note *obj* = *cat-comma-ObjD*[
OF assms(*1*)[*unfolded cat-cf-obj-comma-def*] *is-functor-axioms this*
]
from *obj*[*unfolded cat-1-components*] **have** [*cat-cs-simps*]: $b' = 0$ **by** *simp*
moreover have *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\!|ObjMap|\!)(\!|b'|\!) = b$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
ultimately show $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ $b' = 0$ $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} b$
using *obj* **by** *auto*
qed

lemmas [*dest*] = *is-functor.cat-cf-obj-comma-ObjD*[*rotated 1*]

lemma (**in** *is-functor*) *cat-obj-cf-comma-ObjD*[*dest*]:
  assumes [b', a, f]$_\circ$ $\in_\circ$ b $\downarrow_{CF}$ $\mathfrak{F}(\lfloor Obj \rfloor)$ **and** $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
  shows $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$ **and** $b' = 0$ **and** $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
proof–
  from *assms*(*2*) **have** *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
  note *obj* = *cat-comma-ObjD*[
      OF assms(*1*)[*unfolded cat-obj-cf-comma-def*] *this is-functor-axioms*
      ]
  from *obj*[*unfolded cat-1-components*] **have** [*cat-cs-simps*]: $b' = 0$ **by** *simp*
  moreover have *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) = b$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  ultimately show $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$ $b' = 0$ $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
    using *obj* **by** *auto*
qed

lemmas [*dest*] = *is-functor.cat-obj-cf-comma-ObjD*[*rotated 1*]

lemma (**in** *is-functor*) *cat-cf-obj-comma-ObjE*[*elim*]:
  assumes $A \in_\circ \mathfrak{F}\ {}_{CF}{\downarrow}\ b(\lfloor Obj \rfloor)$ **and** $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
  obtains a f
    where $A = [a, 0, f]_\circ$
      and $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
      and $f : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} b$
proof–
  from *assms*(*2*) **have** *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
  from *assms*(*1*)[*unfolded cat-cf-obj-comma-def*] *is-functor-axioms this*
  obtain a b' f
    where $A = [a, b', f]_\circ$
      and *a*: $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
      and b': $b' \in_\circ$ *cat-1 0 0*$(\lfloor Obj \rfloor)$
      and *f*: $f : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}}$ *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor)$
    by *auto*
  moreover from b' **have** [*cat-cs-simps*]: $b' = 0$
    unfolding *cat-1-components* **by** *auto*
  moreover have *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) = b$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemmas [*elim*] = *is-functor.cat-cf-obj-comma-ObjE*[*rotated 1*]

lemma (**in** *is-functor*) *cat-obj-cf-comma-ObjE*[*elim*]:
  assumes $A \in_\circ b \downarrow_{CF} \mathfrak{F}(\lfloor Obj \rfloor)$ **and** $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
  obtains a f
    where $A = [0, a, f]_\circ$
      and $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
      and $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
proof–
  from *assms*(*2*) **have** *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
  from *assms*(*1*)[*unfolded cat-obj-cf-comma-def*] *is-functor-axioms this*
  obtain a b' f
    where *A-def*: $A = [b', a, f]_\circ$

and a: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and b': $b' \in_{\circ} cat\text{-}1\ 0\ 0(\!|Obj|\!)$
and f: $f : cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b(\!|ObjMap|\!)(\!|b'|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
by *auto*
moreover from b' **have** [*cat-cs-simps*]: $b' = 0$
unfolding *cat-1-components* **by** *auto*
moreover have $cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b(\!|ObjMap|\!)(\!|b'|\!) = b$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemmas [*elim*] = *is-functor.cat-obj-cf-comma-ObjE*[*rotated 1*]

14.5.3 Arrows

lemma (**in** *is-functor*) *cat-cf-obj-comma-ArrI*[*cat-comma-cs-intros*]:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
and $F = [A, B, [g, 0]_{\circ}]_{\circ}$
and $A = [a, 0, f]_{\circ}$
and $B = [a', 0, f']_{\circ}$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} b$
and $f' : \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{B}} b$
and $f' \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = f$
shows $F \in_{\circ} \mathfrak{F}\ _{CF}{\downarrow}\ b(\!|Arr|\!)$
unfolding *cat-cf-obj-comma-def*
proof(*intro cat-comma-ArrI cat-comma-HomI*)
show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms*(*1*) **show** *const*: $cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b : cat\text{-}1\ 0\ 0 \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
from *vempty-is-zet* **show** *0*: $0 : 0 \mapsto_{cat\text{-}1\ 0\ 0} 0$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-1-CId-app* **cs-intro**: *cat-cs-intros*
)
from *assms*(*6*) **show**
$f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b(\!|ObjMap|\!)(\!|0|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms*(*7*) **show**
$f' : \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{B}} cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b(\!|ObjMap|\!)(\!|0|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *0 assms*(*6*) **show**
$f' \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b(\!|ArrMap|\!)(\!|0|\!) \circ_{A\mathfrak{B}} f$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps assms*(*8*) **cs-intro**: *cat-cs-intros*
)
from *const assms*(*5*,*6*) **show** $A \in_{\circ} \mathfrak{F}\ _{CF}{\downarrow}_{CF}\ (cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b)(\!|Obj|\!)$
by (*fold cat-cf-obj-comma-def*)
(*cs-concl* **cs-simp**: *assms*(*3*) **cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
from *const assms*(*5*,*7*) **show** $B \in_{\circ} \mathfrak{F}\ _{CF}{\downarrow}_{CF}\ (cf\text{-}const\ (cat\text{-}1\ 0\ 0)\ \mathfrak{B}\ b)(\!|Obj|\!)$
by (*fold cat-cf-obj-comma-def*)
(
cs-concl **cs-shallow**
cs-simp: *assms*(*4*) **cs-intro**: *cat-cs-intros cat-comma-cs-intros*
)

qed (*intro assms*)+

lemmas [*cat-comma-cs-intros*] = *is-functor.cat-cf-obj-comma-ArrI*

lemma (**in** *is-functor*) *cat-obj-cf-comma-ArrI*[*cat-comma-cs-intros*]:
  assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
    and $F = [A,\ B,\ [0,\ g]_{\circ}]_{\circ}$
    and $A = [0,\ a,\ f]_{\circ}$
    and $B = [0,\ a',\ f']_{\circ}$
    and $g : a \mapsto_{\mathfrak{A}} a'$
    and $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
    and $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
    and $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} f = f'$
  shows $F \in_{\circ} b \downarrow_{CF} \mathfrak{F}(\!|Arr|\!)$
  unfolding *cat-obj-cf-comma-def*
proof(*intro cat-comma-ArrI cat-comma-HomI*)
  show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
  from *assms*(*1*) **show** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
  from *vempty-is-zet* **show** *0*: $0 : 0 \mapsto_{cat\text{-}1\ 0\ 0} 0$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-1-CId-app* **cs-intro**: *cat-cs-intros*)
  from *assms*(*6*) **show**
    f : *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\!|ObjMap|\!)(\!|0|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *assms*(*7*) **show**
    f' : *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\!|ObjMap|\!)(\!|0|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *0 assms*(*7*) **show**
    $f' \circ_{A\mathfrak{B}}$ *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\!|ArrMap|\!)(\!|0|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} f$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps assms*(*8*) **cs-intro**: *cat-cs-intros*
)
  from *const assms*(*5,6*) **show** $A \in_{\circ}$ (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b) $_{CF}\downarrow_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
    by (*fold cat-obj-cf-comma-def*)
      (*cs-concl* **cs-simp**: *assms*(*3*) **cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
  from *const assms*(*5,7*) **show** $B \in_{\circ}$ (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b) $_{CF}\downarrow_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
    by (*fold cat-obj-cf-comma-def*)
      (
        cs-concl **cs-shallow**
          cs-simp: *assms*(*4*) **cs-intro**: *cat-cs-intros cat-comma-cs-intros*
)
qed (*intro assms*)+

lemmas [*cat-comma-cs-intros*] = *is-functor.cat-obj-cf-comma-ArrI*

lemma (**in** *is-functor*) *cat-cf-obj-comma-ArrE*[*elim*]:
  assumes $F \in_{\circ} \mathfrak{F}\ _{CF}\downarrow b(\!|Arr|\!)$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
  obtains $A\ B\ a\ f\ a'\ f'\ g$
    where $F = [A,\ B,\ [g,\ 0]_{\circ}]_{\circ}$
      and $A = [a,\ 0,\ f]_{\circ}$
      and $B = [a',\ 0,\ f']_{\circ}$
      and $g : a \mapsto_{\mathfrak{A}} a'$
      and $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} b$
      and $f' : \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{B}} b$
      and $f' \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = f$
      and $A \in_{\circ} \mathfrak{F}\ _{CF}\downarrow b(\!|Obj|\!)$

```
    and B ∈∘ 𝔉 CF↓ b(|Obj|)
proof–
  from cat-comma-ArrE[OF assms(1)[unfolded cat-cf-obj-comma-def]]
  obtain A B
    where F: F ∈∘ cat-comma-Hom 𝔉 (cf-const (cat-1 0 0) 𝔅 b) A B
      and A: A ∈∘ 𝔉 CF↓CF (cf-const (cat-1 0 0) 𝔅 b)(|Obj|)
      and B: B ∈∘ 𝔉 CF↓CF (cf-const (cat-1 0 0) 𝔅 b)(|Obj|)
    by auto
  from assms(2) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  from F obtain a b'' f a' b' f' g h
    where F-def: F = [A, B, [g, h]∘]∘
      and A-def: A = [a, b'', f]∘
      and B-def: B = [a', b', f']∘
      and g: g : a ↦𝔄 a'
      and h: h : b'' ↦cat-1 0 0 b'
      and f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 cf-const (cat-1 0 0) 𝔅 b(|ObjMap|)(|b''|)
      and f': f' : 𝔉(|ObjMap|)(|a'|) ↦𝔅 cf-const (cat-1 0 0) 𝔅 b(|ObjMap|)(|b'|)
      and f-def:
        f' ∘A𝔅 𝔉(|ArrMap|)(|g|) = cf-const (cat-1 0 0) 𝔅 b(|ArrMap|)(|h|) ∘A𝔅 f
    by (elim cat-comma-HomE[OF - is-functor-axioms const]) blast
  note hb'b'' = cat-1-is-arrD[OF h]
  from F-def have F-def: F = [A, B, [g, 0]∘]∘
    unfolding hb'b'' by simp
  from A-def have A-def: A = [a, 0, f]∘
    unfolding hb'b'' by simp
  from B-def have B-def: B = [a', 0, f']∘
    unfolding hb'b'' by simp
  from f have f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 b
    unfolding hb'b''
    by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from f' have f': f' : 𝔉(|ObjMap|)(|a'|) ↦𝔅 b
    unfolding hb'b''
    by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from f-def f f' g h have f-def: f' ∘A𝔅 𝔉(|ArrMap|)(|g|) = f
    unfolding hb'b''
    by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from
    that F-def A-def B-def g f f' f-def
    B[folded cat-cf-obj-comma-def] A[folded cat-cf-obj-comma-def]
  show ?thesis
    by blast
qed

lemmas [elim] = is-functor.cat-cf-obj-comma-ArrE[rotated 1]

lemma (in is-functor) cat-obj-cf-comma-ArrE[elim]:
  assumes F ∈∘ b ↓CF 𝔉(|Arr|) and b ∈∘ 𝔅(|Obj|)
  obtains A B a f a' f' g
    where F = [A, B, [0, g]∘]∘
      and A = [0, a, f]∘
      and B = [0, a', f']∘
      and g : a ↦𝔄 a'
      and f : b ↦𝔅 𝔉(|ObjMap|)(|a|)
      and f' : b ↦𝔅 𝔉(|ObjMap|)(|a'|)
      and 𝔉(|ArrMap|)(|g|) ∘A𝔅 f = f'
      and A ∈∘ b ↓CF 𝔉(|Obj|)
      and B ∈∘ b ↓CF 𝔉(|Obj|)
```

proof–
from *cat-comma-ArrE*[*OF assms*(*1*)[*unfolded cat-obj-cf-comma-def*]]
obtain *A B*
where *F*: $F \in_\circ$ *cat-comma-Hom* (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ *b*) $\mathfrak{F}$ *A B*
and *A*: $A \in_\circ$ (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ *b*) $_{CF}{\downarrow}_{CF}$ $\mathfrak{F}(\lfloor Obj \rfloor)$
and *B*: $B \in_\circ$ (*cf-const* (*cat-1 0 0*) $\mathfrak{B}$ *b*) $_{CF}{\downarrow}_{CF}$ $\mathfrak{F}(\lfloor Obj \rfloor)$
by *auto*
from *assms*(*2*) **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ *b* : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
from *F* **obtain** *a b″ f a′ b′ f′ h g*
where *F-def*: $F = [A, B, [h, g]_\circ]_\circ$
and *A-def*: $A = [b', a, f]_\circ$
and *B-def*: $B = [b'', a', f']_\circ$
and *h*: $h : b' \mapsto_{cat\text{-}1\ 0\ 0} b''$
and *g*: $g : a \mapsto_{\mathfrak{A}} a'$
and *f*: f : *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
and *f′*: f' : *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\lfloor ObjMap \rfloor)(\lfloor b'' \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor)$
and *f′-def*:
$f' \circ_{A\mathfrak{B}}$ *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ $b(\lfloor ArrMap \rfloor)(\lfloor h \rfloor) = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \circ_{A\mathfrak{B}} f$
by (*elim cat-comma-HomE*[*OF - const is-functor-axioms*]) *blast*
note *hb′b″* = *cat-1-is-arrD*[*OF h*]
from *F-def* **have** *F-def*: $F = [A, B, [0, g]_\circ]_\circ$
unfolding *hb′b″* **by** *simp*
from *A-def* **have** *A-def*: $A = [0, a, f]_\circ$ **unfolding** *hb′b″* **by** *simp*
from *B-def* **have** *B-def*: $B = [0, a', f']_\circ$ **unfolding** *hb′b″* **by** *simp*
from *f* **have** *f*: $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
unfolding *hb′b″*
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *f′* **have** *f′*: $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor)$
unfolding *hb′b″*
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *f′-def f f′ g h* **have** *f′-def*[*symmetric*]: $f' = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \circ_{A\mathfrak{B}} f$
unfolding *hb′b″*
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from
that F-def A-def B-def g f f′ f′-def
A[*folded cat-obj-cf-comma-def*] *B*[*folded cat-obj-cf-comma-def*]
show *?thesis*
by *blast*
qed

lemmas [*elim*] = *is-functor.cat-obj-cf-comma-ArrE*

lemma (**in** *is-functor*) *cat-cf-obj-comma-ArrD*[*dest*]:
assumes $[[a, b', f]_\circ, [a', b'', f']_\circ, [g, h]_\circ]_\circ \in_\circ \mathfrak{F}\ _{CF}{\downarrow}\ b(\lfloor Arr \rfloor)$
and $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
shows $b' = 0$
and $b'' = 0$
and $h = 0$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $f : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} b$
and $f' : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor) \mapsto_{\mathfrak{B}} b$
and $f' \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) = f$
and $[a, b', f]_\circ \in_\circ \mathfrak{F}\ _{CF}{\downarrow}\ b(\lfloor Obj \rfloor)$
and $[a', b'', f']_\circ \in_\circ \mathfrak{F}\ _{CF}{\downarrow}\ b(\lfloor Obj \rfloor)$
using *cat-cf-obj-comma-ArrE*[*OF assms*] **by** *auto*

lemmas [*dest*] = *is-functor.cat-cf-obj-comma-ArrD*[*rotated 1*]

lemma (in *is-functor*) *cat-obj-cf-comma-ArrD*[*dest*]:
  assumes [[b', a, f]$_\circ$, [b'', a', f']$_\circ$, [h, g]$_\circ$]$_\circ$ $\in_\circ$ b $\downarrow_{CF}$ $\mathfrak{F}(\!|Arr|\!)$
    and b $\in_\circ$ $\mathfrak{B}(\!|Obj|\!)$
  shows $b' = 0$
    and $b'' = 0$
    and $h = 0$
    and $g : a \mapsto_{\mathfrak{A}} a'$
    and $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
    and $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
    and $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!)$ $\circ_{A\mathfrak{B}}$ $f = f'$
    and [b', a, f]$_\circ$ $\in_\circ$ b $\downarrow_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
    and [b'', a', f']$_\circ$ $\in_\circ$ b $\downarrow_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
  using *cat-obj-cf-comma-ArrE*[*OF assms*] **by** *auto*

lemmas [*dest*] = *is-functor.cat-obj-cf-comma-ArrD*

14.5.4 Domain

lemma *cat-cf-obj-comma-Dom-vsv*[*cat-comma-cs-intros*]: *vsv* ($\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Dom|\!)$)
  unfolding *cat-cf-obj-comma-def cat-comma-components* **by** *simp*

lemma *cat-cf-obj-comma-Dom-vdomain*[*cat-comma-cs-simps*]:
  $\mathcal{D}_\circ$ ($\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Dom|\!)$) = $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Arr|\!)$
  unfolding *cat-cf-obj-comma-def cat-comma-components* **by** *simp*

lemma *cat-cf-obj-comma-Dom-app*[*cat-comma-cs-simps*]:
  assumes ABF = [A, B, F]$_\circ$ **and** ABF $\in_\circ$ $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Arr|\!)$
  shows $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Dom|\!)(\!|ABF|\!)$ = A
  using *assms*(*2*)
  unfolding *assms*(*1*) *cat-cf-obj-comma-def cat-comma-components*
  by *simp*

lemma (in *is-functor*) *cat-cf-obj-comma-Dom-vrange*:
  assumes b $\in_\circ$ $\mathfrak{B}(\!|Obj|\!)$
  shows $\mathcal{R}_\circ$ ($\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Dom|\!)$) $\subseteq_\circ$ $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Obj|\!)$
proof–
  from *assms* **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
    by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
  show *?thesis*
    by
      (
        rule cat-comma-Dom-vrange[
          OF is-functor-axioms const, *folded cat-cf-obj-comma-def*
          ]
)
qed

lemma *cat-obj-cf-comma-Dom-vsv*[*cat-comma-cs-intros*]: *vsv* (b $\downarrow_{CF}$ $\mathfrak{F}(\!|Dom|\!)$)
  unfolding *cat-obj-cf-comma-def cat-comma-components* **by** *simp*

lemma *cat-obj-cf-comma-Dom-vdomain*[*cat-comma-cs-simps*]:
  $\mathcal{D}_\circ$ (b $\downarrow_{CF}$ $\mathfrak{F}(\!|Dom|\!)$) = b $\downarrow_{CF}$ $\mathfrak{F}(\!|Arr|\!)$
  unfolding *cat-obj-cf-comma-def cat-comma-components* **by** *simp*

lemma *cat-obj-cf-comma-Dom-app*[*cat-comma-cs-simps*]:
  assumes ABF = [A, B, F]$_\circ$ **and** ABF $\in_\circ$ b $\downarrow_{CF}$ $\mathfrak{F}(\!|Arr|\!)$
  shows b $\downarrow_{CF}$ $\mathfrak{F}(\!|Dom|\!)(\!|ABF|\!)$ = A

using *assms*(*2*)
unfolding *assms*(*1*) *cat-obj-cf-comma-def cat-comma-components*
by *simp*

lemma (**in** *is-functor*) *cat-obj-cf-comma-Dom-vrange*:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{R}_{\circ}\ (b \downarrow_{CF} \mathfrak{F}(\!|Dom|\!)) \subseteq_{\circ} b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
proof–
from *assms* **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\!\!\!\rightarrow_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
show *?thesis*
by
(
rule cat-comma-Dom-vrange[
OF const is-functor-axioms, *folded cat-obj-cf-comma-def*
]
)
qed

14.5.5 Codomain

lemma *cat-cf-obj-comma-Cod-vsv*[*cat-comma-cs-intros*]: *vsv* ($\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Cod|\!)$)
unfolding *cat-cf-obj-comma-def cat-comma-components* **by** *simp*

lemma *cat-cf-obj-comma-Cod-vdomain*[*cat-comma-cs-simps*]:
$\mathcal{D}_{\circ}$ ($\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Cod|\!)$) = $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Arr|\!)$
unfolding *cat-cf-obj-comma-def cat-comma-components* **by** *simp*

lemma *cat-cf-obj-comma-Cod-app*[*cat-comma-cs-simps*]:
assumes $ABF = [A, B, F]_{\circ}$ **and** $ABF \in_{\circ} \mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Arr|\!)$
shows $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Cod|\!)(\!|ABF|\!) = B$
using *assms*(*2*)
unfolding *assms*(*1*) *cat-cf-obj-comma-def cat-comma-components*
by (*simp add*: *nat-omega-simps*)

lemma (**in** *is-functor*) *cat-cf-obj-comma-Cod-vrange*:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{R}_{\circ}$ ($\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Cod|\!)$) $\subseteq_{\circ}$ $\mathfrak{F}$ $_{CF}\!\downarrow$ $b(\!|Obj|\!)$
proof–
from *assms* **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\!\!\!\rightarrow_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
show *?thesis*
by
(
rule cat-comma-Cod-vrange[
OF is-functor-axioms const, *folded cat-cf-obj-comma-def*
]
)
qed

lemma *cat-obj-cf-comma-Cod-vsv*[*cat-comma-cs-intros*]: *vsv* ($b \downarrow_{CF} \mathfrak{F}(\!|Cod|\!)$)
unfolding *cat-obj-cf-comma-def cat-comma-components* **by** *simp*

lemma *cat-obj-cf-comma-Cod-vdomain*[*cat-comma-cs-simps*]:
$\mathcal{D}_{\circ}$ ($b \downarrow_{CF} \mathfrak{F}(\!|Cod|\!)$) = $b \downarrow_{CF} \mathfrak{F}(\!|Arr|\!)$
unfolding *cat-obj-cf-comma-def cat-comma-components* **by** *simp*

lemma *cat-obj-cf-comma-Cod-app*[*cat-comma-cs-simps*]:

assumes $ABF = [A,\ B,\ F]_\circ$ **and** $ABF \in_\circ b \downarrow_{CF} \mathfrak{F}(\!|Arr|\!)$
shows $b \downarrow_{CF} \mathfrak{F}(\!|Cod|\!)(\!|ABF|\!) = B$
using *assms*(*2*)
unfolding *assms*(*1*) *cat-obj-cf-comma-def cat-comma-components*
by (*simp add*: *nat-omega-simps*)

lemma (**in** *is-functor*) *cat-obj-cf-comma-Cod-vrange*:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{R}_\circ\ (b \downarrow_{CF} \mathfrak{F}(\!|Dom|\!)) \subseteq_\circ b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
proof–
from *assms* **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
show *?thesis*
by
(
rule cat-comma-Dom-vrange[
OF const is-functor-axioms, *folded cat-obj-cf-comma-def*
]
)
qed

14.5.6 Arrow with a domain and a codomain

lemma (**in** *is-functor*) *cat-cf-obj-comma-is-arrI*[*cat-comma-cs-intros*]:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $ABF = [A,\ B,\ F]_\circ$
and $A = [a,\ 0,\ f]_\circ$
and $B = [a',\ 0,\ f']_\circ$
and $F = [g,\ 0]_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} b$
and $f' : \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{B}} b$
and $f' \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = f$
shows $ABF : A \mapsto_{\mathfrak{F}\ CF\downarrow\ b} B$
proof(*intro is-arrI*)
from *assms*(*1,6,7,8*) **show** $ABF \in_\circ \mathfrak{F}\ _{CF}{\downarrow}\ b(\!|Arr|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *assms*(*2,3,4,5,9*) **cs-intro**: *cat-comma-cs-intros*
)
with *assms*(*2*) **show** $\mathfrak{F}\ _{CF}{\downarrow}\ b(\!|Dom|\!)(\!|ABF|\!) = A\ \mathfrak{F}\ _{CF}{\downarrow}\ b(\!|Cod|\!)(\!|ABF|\!) = B$
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)+
qed

lemmas [*cat-comma-cs-intros*] = *is-functor.cat-cf-obj-comma-is-arrI*

lemma (**in** *is-functor*) *cat-obj-cf-comma-is-arrI*[*cat-comma-cs-intros*]:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $ABF = [A,\ B,\ F]_\circ$
and $A = [0,\ a,\ f]_\circ$
and $B = [0,\ a',\ f']_\circ$
and $F = [0,\ g]_\circ$
and $g : a \mapsto_{\mathfrak{A}} a'$
and $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
and $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
and $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} f = f'$
shows $ABF : A \mapsto_{b \downarrow_{CF}\ \mathfrak{F}} B$

proof(*intro is-arrI*)
  from *assms*(*1*,*6*,*7*,*8*) **show** $ABF \in_\circ b \downarrow_{CF} \mathfrak{F}(\lfloor Arr \rfloor)$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *assms*(*2*,*3*,*4*,*5*,*9*) **cs-intro**: *cat-comma-cs-intros*
)
  with *assms*(*2*) **show** $b \downarrow_{CF} \mathfrak{F}(\lfloor Dom \rfloor)(\lfloor ABF \rfloor) = A$ $b \downarrow_{CF} \mathfrak{F}(\lfloor Cod \rfloor)(\lfloor ABF \rfloor) = B$
    by (*cs-concl* **cs-shallow** **cs-simp**: *cat-comma-cs-simps*)+
qed

lemmas [*cat-comma-cs-intros*] = *is-functor.cat-obj-cf-comma-is-arrI*

lemma (**in** *is-functor*) *cat-cf-obj-comma-is-arrD*[*dest*]:
  assumes $[[a, b', f]_\circ, [a', b'', f']_\circ, [g, h]_\circ]_\circ :$
    $[a, b', f]_\circ \mapsto_{\mathfrak{F}\ CF\downarrow\ b} [a', b'', f']_\circ$
    and $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
  shows $b' = 0$
    and $b'' = 0$
    and $h = 0$
    and $g : a \mapsto_{\mathfrak{A}} a'$
    and $f : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} b$
    and $f' : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor) \mapsto_{\mathfrak{B}} b$
    and $f' \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) = f$
    and $[a, b', f]_\circ \in_\circ \mathfrak{F}\ _{CF\downarrow}\ b(\lfloor Obj \rfloor)$
    and $[a', b'', f']_\circ \in_\circ \mathfrak{F}\ _{CF\downarrow}\ b(\lfloor Obj \rfloor)$
  by (*intro cat-cf-obj-comma-ArrD*[*OF is-arrD*(*1*)[*OF assms*(*1*)] *assms*(*2*)])+

lemma (**in** *is-functor*) *cat-obj-cf-comma-is-arrD*[*dest*]:
  assumes $[[b', a, f]_\circ, [b'', a', f']_\circ, [h, g]_\circ]_\circ :$
    $[b', a, f]_\circ \mapsto_{b\ \downarrow_{CF}\ \mathfrak{F}} [b'', a', f']_\circ$
    and $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
  shows $b' = 0$
    and $b'' = 0$
    and $h = 0$
    and $g : a \mapsto_{\mathfrak{A}} a'$
    and $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
    and $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor)$
    and $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) \circ_{A\mathfrak{B}} f = f'$
    and $[b', a, f]_\circ \in_\circ b \downarrow_{CF} \mathfrak{F}(\lfloor Obj \rfloor)$
    and $[b'', a', f']_\circ \in_\circ b \downarrow_{CF} \mathfrak{F}(\lfloor Obj \rfloor)$
  by (*intro cat-obj-cf-comma-ArrD*[*OF is-arrD*(*1*)[*OF assms*(*1*)] *assms*(*2*)])+

lemmas [*dest*] = *is-functor.cat-obj-cf-comma-is-arrD*

lemma (**in** *is-functor*) *cat-cf-obj-comma-is-arrE*[*elim*]:
  assumes $ABF : A \mapsto_{\mathfrak{F}\ CF\downarrow\ b} B$ **and** $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
  obtains $a\ f\ a'\ f'\ g$
    where $ABF = [[a, 0, f]_\circ, [a', 0, f']_\circ, [g, 0]_\circ]_\circ$
      and $A = [a, 0, f]_\circ$
      and $B = [a', 0, f']_\circ$
      and $g : a \mapsto_{\mathfrak{A}} a'$
      and $f : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} b$
      and $f' : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor) \mapsto_{\mathfrak{B}} b$
      and $f' \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor) = f$
      and $A \in_\circ \mathfrak{F}\ _{CF\downarrow}\ b(\lfloor Obj \rfloor)$
      and $B \in_\circ \mathfrak{F}\ _{CF\downarrow}\ b(\lfloor Obj \rfloor)$
proof–

note *ABF* = *is-arrD*[*OF assms*(*1*)]
from *ABF*(*1*) **obtain** *C D a f a′ f′ g*
where *ABF-def*: *ABF* = [*C*, *D*, [*g*, *0*]∘]∘
and *C-def*: *C* = [*a*, *0*, *f*]∘
and *D-def*: *D* = [*a′*, *0*, *f′*]∘
and *g*: *g* : *a* ↦$_{\mathfrak{A}}$ *a′*
and *f*: *f* : $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$ ↦$_{\mathfrak{B}}$ *b*
and *f′*: *f′* : $\mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$ ↦$_{\mathfrak{B}}$ *b*
and *f-def*: *f′* ∘$_{A\mathfrak{B}}$ $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!)$ = *f*
and *C*: *C* ∈∘ $\mathfrak{F}$ $_{CF}$↓ *b*(|*Obj*|)
and *D*: *D* ∈∘ $\mathfrak{F}$ $_{CF}$↓ *b*(|*Obj*|)
by (*elim cat-cf-obj-comma-ArrE*[*OF* - *assms*(*2*)])
from *ABF*(*2*) *assms*(*2*) *C-def D-def g f f′ f-def* **have** [*simp*]: *C* = *A*
unfolding *ABF-def*
by
(
cs-prems **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)
from *ABF*(*3*) *assms*(*2*) *C-def D-def g f f′ f-def* **have** [*simp*]: *D* = *B*
unfolding *ABF-def*
by
(
cs-prems **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)
from *that ABF-def C-def D-def g f f′ f-def C D* **show** *?thesis* **by** *auto*
qed

lemmas [*elim*] = *is-functor.cat-cf-obj-comma-is-arrE*

lemma (**in** *is-functor*) *cat-obj-cf-comma-is-arrE*[*elim*]:
assumes *ABF* : *A* ↦$_{b}$ ↓$_{CF}$ $\mathfrak{F}$ *B* **and** *b* ∈∘ $\mathfrak{B}$(|*Obj*|)
obtains *a f a′ f′ g*
where *ABF* = [[*0*, *a*, *f*]∘, [*0*, *a′*, *f′*]∘, [*0*, *g*]∘]∘
and *A* = [*0*, *a*, *f*]∘
and *B* = [*0*, *a′*, *f′*]∘
and *g* : *a* ↦$_{\mathfrak{A}}$ *a′*
and *f* : *b* ↦$_{\mathfrak{B}}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
and *f′* : *b* ↦$_{\mathfrak{B}}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
and $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!)$ ∘$_{A\mathfrak{B}}$ *f* = *f′*
and *A* ∈∘ *b* ↓$_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
and *B* ∈∘ *b* ↓$_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
proof–
note *ABF* = *is-arrD*[*OF assms*(*1*)]
from *ABF*(*1*) **obtain** *C D a f a′ f′ g*
where *ABF-def*: *ABF* = [*C*, *D*, [*0*, *g*]∘]∘
and *C-def*: *C* = [*0*, *a*, *f*]∘
and *D-def*: *D* = [*0*, *a′*, *f′*]∘
and *g*: *g* : *a* ↦$_{\mathfrak{A}}$ *a′*
and *f*: *f* : *b* ↦$_{\mathfrak{B}}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
and *f′*: *f′* : *b* ↦$_{\mathfrak{B}}$ $\mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
and *f′-def*: $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!)$ ∘$_{A\mathfrak{B}}$ *f* = *f′*
and *C*: *C* ∈∘ *b* ↓$_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
and *D*: *D* ∈∘ *b* ↓$_{CF}$ $\mathfrak{F}(\!|Obj|\!)$
by (*elim cat-obj-cf-comma-ArrE*[*OF* - *assms*(*2*)])
from *ABF*(*2*) *assms*(*2*) *C-def D-def g f f′ f′-def* **have** [*simp*]: *C* = *A*
unfolding *ABF-def*

by
(
cs-prems **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)
from *ABF*(*3*) *assms*(*2*) *C-def D-def g f f′ f′-def* **have** [*simp*]: *D* = *B*
unfolding *ABF-def*
by
(
cs-prems **cs-shallow**
cs-simp: *cat-comma-cs-simps* **cs-intro**: *cat-comma-cs-intros*
)
from *that ABF-def C-def D-def g f f′ f′-def C D* **show** *?thesis* **by** *auto*
qed

lemmas [*elim*] = *is-functor.cat-obj-cf-comma-is-arrE*

14.5.7 Composition

lemma *cat-cf-obj-comma-Comp-vsv*[*cat-comma-cs-intros*]: *vsv* (ℱ $_{CF}\!\downarrow$ *b*(‖*Comp*‖))
unfolding *cat-cf-obj-comma-def*
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

lemma *cat-obj-cf-comma-Comp-vsv*[*cat-comma-cs-intros*]: *vsv* (*b* $\downarrow_{CF}$ ℱ(‖*Comp*‖))
unfolding *cat-obj-cf-comma-def*
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

lemma (**in** *is-functor*) *cat-cf-obj-comma-Comp-app*[*cat-comma-cs-simps*]:
assumes *b* ∈$_\circ$ 𝔅(‖*Obj*‖)
and *BCG* = [*B*, *C*, [*g′*, *h′*]$_\circ$]$_\circ$
and *ABF* = [*A*, *B*, [*g*, *h*]$_\circ$]$_\circ$
and *BCG* : *B* ↦$_{\mathfrak{F}\ CF\downarrow\ b}$ *C*
and *ABF* : *A* ↦$_{\mathfrak{F}\ CF\downarrow\ b}$ *B*
shows *BCG* ∘$_{A\mathfrak{F}\ CF\downarrow\ b}$ *ABF* = [*A*, *C*, [*g′* ∘$_{A\mathfrak{A}}$ *g*, *0*]$_\circ$]$_\circ$
proof–
from *assms*(*1*) **have** *const*: *cf-const* (*cat-1 0 0*) 𝔅 *b* : *cat-1 0 0* ↦↦$_{C\alpha}$ 𝔅
by (*cs-concl* **cs-intro**: *vempty-is-zet cat-cs-intros*)
from *assms*(*4*) **obtain** *a f a′ f′ g*
where *BCG-def*: *BCG* = [[*a*, *0*, *f*]$_\circ$, [*a′*, *0*, *f′*]$_\circ$, [*g*, *0*]$_\circ$]$_\circ$
by (*elim cat-cf-obj-comma-is-arrE*[*OF - assms*(*1*)])
from *assms*(*5*) **obtain** *a f a′ f′ g*
where *ABF-def*: *ABF* = [[*a*, *0*, *f*]$_\circ$, [*a′*, *0*, *f′*]$_\circ$, [*g*, *0*]$_\circ$]$_\circ$
by (*elim cat-cf-obj-comma-is-arrE*[*OF - assms*(*1*)])
from *assms*(*2*)[*unfolded BCG-def*] *assms*(*3*)[*unfolded ABF-def*] **have** [*cat-cs-simps*]:
h′ = *0 h* = *0*
by *simp-all*
have *h′* ∘$_{A\ cat\text{-}1\ 0\ 0}$ *h* = *0* **by** (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
show *?thesis*
by
(
rule cat-comma-Comp-app
[
OF
is-functor-axioms
const
assms(*2*,*3*)
assms(*4*)[*unfolded cat-cf-obj-comma-def*]
assms(*5*)[*unfolded cat-cf-obj-comma-def*],

```
      folded cat-cf-obj-comma-def,
      unfolded cat-cs-simps
    ]
  )
qed

lemma (in is-functor) cat-obj-cf-comma-Comp-app[cat-comma-cs-simps]:
  assumes b ∈ₒ 𝔅(|Obj|)
    and BCG = [B, C, [h', g']ₒ]ₒ
    and ABF = [A, B, [h, g]ₒ]ₒ
    and BCG : B ↦_b ↓_CF 𝔉 C
    and ABF : A ↦_b ↓_CF 𝔉 B
  shows BCG ∘_A b ↓_CF 𝔉 ABF = [A, C, [0, g' ∘_A𝔄 g]ₒ]ₒ
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦_Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  from assms(4) obtain a f a' f' g
    where BCG-def: BCG = [[0, a, f]ₒ, [0, a', f']ₒ, [0, g]ₒ]ₒ
    by (elim cat-obj-cf-comma-is-arrE[OF - assms(1)])
  from assms(5) obtain a f a' f' g
    where ABF-def: ABF = [[0, a, f]ₒ, [0, a', f']ₒ, [0, g]ₒ]ₒ
    by (elim cat-obj-cf-comma-is-arrE[OF - assms(1)])
  from assms(2)[unfolded BCG-def] assms(3)[unfolded ABF-def] have [cat-cs-simps]:
    h' = 0 h = 0
    by simp-all
  have h' ∘_A cat-1 0 0 h = 0 by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
    by
      (
        rule cat-comma-Comp-app
          [
            OF
              const
              is-functor-axioms
              assms(2,3)
              assms(4)[unfolded cat-obj-cf-comma-def]
              assms(5)[unfolded cat-obj-cf-comma-def],
            folded cat-obj-cf-comma-def,
            unfolded cat-cs-simps
          ]
      )
qed

lemma (in is-functor) cat-cf-obj-comma-Comp-is-arr[cat-comma-cs-intros]:
  assumes b ∈ₒ 𝔅(|Obj|)
    and BCG : B ↦_𝔉 _CF↓ b C
    and ABF : A ↦_𝔉 _CF↓ b B
  shows BCG ∘_A𝔉 _CF↓ b ABF : A ↦_𝔉 _CF↓ b C
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦_Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  show ?thesis
    by
      (
        rule cat-comma-Comp-is-arr
          [
            OF
              is-functor-axioms
```

```
        const
        assms(2)[unfolded cat-cf-obj-comma-def]
        assms(3)[unfolded cat-cf-obj-comma-def],
      folded cat-cf-obj-comma-def
    ]
  )
qed

lemma (in is-functor) cat-obj-cf-comma-Comp-is-arr[cat-comma-cs-intros]:
  assumes b ∈ₒ 𝔅(|Obj|)
    and BCG : B ↦_b ↓_CF 𝔉 C
    and ABF : A ↦_b ↓_CF 𝔉 B
  shows BCG ∘_A b ↓_CF 𝔉 ABF : A ↦_b ↓_CF 𝔉 C
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦_Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  show ?thesis
    by
      (
        rule cat-comma-Comp-is-arr
          [
            OF
              const
              is-functor-axioms
              assms(2)[unfolded cat-obj-cf-comma-def]
              assms(3)[unfolded cat-obj-cf-comma-def],
            folded cat-obj-cf-comma-def
          ]
      )
qed
```

14.5.8 Identity

```
lemma cat-cf-obj-comma-CId-vsv[cat-comma-cs-intros]: vsv (𝔉 _CF↓ b(|CId|))
  unfolding cat-cf-obj-comma-def
  by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)

lemma cat-obj-cf-comma-CId-vsv[cat-comma-cs-intros]: vsv (b ↓_CF 𝔉(|CId|))
  unfolding cat-obj-cf-comma-def
  by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)

lemma (in is-functor) cat-cf-obj-comma-CId-vdomain[cat-comma-cs-simps]:
  assumes b ∈ₒ 𝔅(|Obj|)
  shows 𝒟ₒ (𝔉 _CF↓ b(|CId|)) = 𝔉 _CF↓ b(|Obj|)
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦_Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  show ?thesis
    by
      (
        rule cat-comma-CId-vdomain[
          OF is-functor-axioms const, folded cat-cf-obj-comma-def
          ]
      )
qed

lemma (in is-functor) cat-obj-cf-comma-CId-vdomain[cat-comma-cs-simps]:
  assumes b ∈ₒ 𝔅(|Obj|)
```

```
  shows 𝒟∘ (b ↓CF 𝔉(⁅CId⁆)) = b ↓CF 𝔉(⁅Obj⁆)
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔙 b : cat-1 0 0 ↦↦Cα 𝔙
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  show 𝒟∘ (b ↓CF 𝔉(⁅CId⁆)) = b ↓CF 𝔉(⁅Obj⁆)
    by
      (
        rule cat-comma-CId-vdomain[
          OF const is-functor-axioms, folded cat-obj-cf-comma-def
          ]
      )
qed

lemma (in is-functor) cat-cf-obj-comma-CId-app[cat-comma-cs-simps]:
  assumes b ∈∘ 𝔙(⁅Obj⁆) and A = [a, b′, f]∘ and A ∈∘ 𝔉 CF↓ b(⁅Obj⁆)
  shows 𝔉 CF↓ b(⁅CId⁆)(⁅A⁆) = [A, A, [𝔄(⁅CId⁆)(⁅a⁆), 0]∘]∘
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔙 b : cat-1 0 0 ↦↦Cα 𝔙
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  from assms(3,2) have b′-def: b′ = 0
    by (auto elim: cat-cf-obj-comma-ObjE[OF - assms(1)])
  have [cat-cs-simps]: cat-1 0 0(⁅CId⁆)(⁅b′⁆) = 0
    unfolding cat-1-components b′-def by simp
  show ?thesis
    by
      (
        rule cat-comma-CId-app
          [
            OF
              is-functor-axioms
              const
              assms(2,3)[unfolded cat-cf-obj-comma-def],
            unfolded cat-cf-obj-comma-def[symmetric] cat-cs-simps
          ]
      )
qed

lemma (in is-functor) cat-obj-cf-comma-CId-app[cat-comma-cs-simps]:
  assumes b ∈∘ 𝔙(⁅Obj⁆) and A = [b′, a, f]∘ and A ∈∘ b ↓CF 𝔉(⁅Obj⁆)
  shows b ↓CF 𝔉(⁅CId⁆)(⁅A⁆) = [A, A, [0, 𝔄(⁅CId⁆)(⁅a⁆)]∘]∘
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔙 b : cat-1 0 0 ↦↦Cα 𝔙
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  from assms(3,2) have b′-def: b′ = 0
    by (auto elim: cat-obj-cf-comma-ObjE[OF - assms(1)])
  have [cat-cs-simps]: cat-1 0 0(⁅CId⁆)(⁅b′⁆) = 0
    unfolding cat-1-components b′-def by simp
  show ?thesis
    by
      (
        rule cat-comma-CId-app
          [
            OF
              const
              is-functor-axioms
              assms(2,3)[unfolded cat-obj-cf-comma-def],
            unfolded cat-obj-cf-comma-def[symmetric] cat-cs-simps
          ]
```

```
    )
qed
```

14.5.9 Comma categories constructed from a functor and an object are categories

```
lemma (in is-functor) category-cat-cf-obj-comma[cat-comma-cs-intros]:
  assumes b ∈∘ 𝔅(|Obj|)
  shows category α (𝔉 CF↓ b)
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  show ?thesis
    by
      (
        rule category-cat-comma[
          OF is-functor-axioms const, folded cat-cf-obj-comma-def
          ]
      )
qed

lemmas [cat-comma-cs-intros] = is-functor.category-cat-cf-obj-comma

lemma (in is-functor) category-cat-obj-cf-comma[cat-comma-cs-intros]:
  assumes b ∈∘ 𝔅(|Obj|)
  shows category α (b ↓CF 𝔉)
proof–
  from assms(1) have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦Cα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-cs-intros)
  show ?thesis
    by
      (
        rule category-cat-comma[
          OF const is-functor-axioms, folded cat-obj-cf-comma-def
          ]
      )
qed

lemmas [cat-comma-cs-intros] = is-functor.category-cat-obj-cf-comma
```

14.5.10 Tiny comma categories constructed from a functor and an object

```
lemma (in is-tm-functor) tiny-category-cat-cf-obj-comma[cat-comma-cs-intros]:
  assumes b ∈∘ 𝔅(|Obj|)
  shows tiny-category α (𝔉 CF↓ b)
proof–
  from assms(1) have const:
    cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦C.tmα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-small-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule tiny-category-cat-comma[
          OF is-tm-functor-axioms const, folded cat-cf-obj-comma-def
          ]
      )
qed

lemma (in is-tm-functor) tiny-category-cat-obj-cf-comma[cat-comma-cs-intros]:
```

```
  assumes b ∈∘ 𝔅(|Obj|)
  shows tiny-category α (b ↓CF 𝔉)
proof–
  from assms(1) have const:
    cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦C.tmα 𝔅
    by (cs-concl cs-intro: vempty-is-zet cat-small-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule tiny-category-cat-comma[
          OF const is-tm-functor-axioms, folded cat-obj-cf-comma-def
          ]
      )
qed
```

14.6 Opposite comma category functors for the comma categories constructed from a functor and an object

14.6.1 Definitions and elementary properties

```
definition op-cf-obj-comma :: V ⇒ V ⇒ V
  where op-cf-obj-comma 𝔉 b =
    op-cf-comma 𝔉 (cf-const (cat-1 0 0) (𝔉(|HomCod|)) b)

definition op-obj-cf-comma :: V ⇒ V ⇒ V
  where op-obj-cf-comma b 𝔉 =
    op-cf-comma (cf-const (cat-1 0 0) (𝔉(|HomCod|)) b) 𝔉
```

Alternative forms of the definitions.

```
lemma (in is-functor) op-cf-obj-comma-def:
  op-cf-obj-comma 𝔉 b = op-cf-comma 𝔉 (cf-const (cat-1 0 0) 𝔅 b)
  unfolding op-cf-obj-comma-def cat-cs-simps by simp

lemma (in is-functor) op-obj-cf-comma-def:
  op-obj-cf-comma b 𝔉 = op-cf-comma (cf-const (cat-1 0 0) 𝔅 b) 𝔉
  unfolding op-obj-cf-comma-def cat-cs-simps by simp
```

14.6.2 Object map

```
lemma op-cf-obj-comma-ObjMap-vsv[cat-comma-cs-intros]:
  vsv (op-cf-obj-comma 𝔉 b(|ObjMap|))
  unfolding op-cf-obj-comma-def
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps cs-intro: cat-comma-cs-intros
    )

lemma op-obj-cf-comma-ObjMap-vsv[cat-comma-cs-intros]:
  vsv (op-obj-cf-comma b 𝔉(|ObjMap|))
  unfolding op-obj-cf-comma-def
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps cs-intro: cat-comma-cs-intros
    )

lemma (in is-functor) op-cf-obj-comma-ObjMap-vdomain[cat-comma-cs-simps]:
```

```
  𝒟∘ (op-cf-obj-comma 𝔉 b(|ObjMap|)) = 𝔉 CF↓ b(|Obj|)
  unfolding op-cf-obj-comma-def
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps cat-cf-obj-comma-def[symmetric]
    )

lemma (in is-functor) op-obj-cf-comma-ObjMap-vdomain[cat-comma-cs-simps]:
  𝒟∘ (op-obj-cf-comma b 𝔉(|ObjMap|)) = b ↓CF 𝔉(|Obj|)
  unfolding op-obj-cf-comma-def
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-comma-cs-simps cat-obj-cf-comma-def[symmetric]
    )

lemma (in is-functor) op-cf-obj-comma-ObjMap-app[cat-comma-cs-simps]:
  assumes A = [a, 0, f]∘ and b ∈∘ 𝔅(|Obj|) and A ∈∘ 𝔉 CF↓ b(|Obj|)
  shows op-cf-obj-comma 𝔉 b(|ObjMap|)(|A|) = [0, a, f]∘
proof–
  have a: a ∈∘ 𝔄(|Obj|) and f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 b
    by (intro cat-cf-obj-comma-ObjD[OF assms(3)[unfolded assms(1)] assms(2)])+
  from assms(2) a f show ?thesis
    using assms(2)
    unfolding assms(1) op-cf-obj-comma-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: V-cs-intros cat-cs-intros cat-comma-cs-intros
      )
qed

lemma (in is-functor) op-obj-cf-comma-ObjMap-app[cat-comma-cs-simps]:
  assumes A = [0, a, f]∘ and b ∈∘ 𝔅(|Obj|) and A ∈∘ b ↓CF 𝔉(|Obj|)
  shows op-obj-cf-comma b 𝔉 (|ObjMap|)(|A|) = [a, 0, f]∘
proof–
  have a: a ∈∘ 𝔄(|Obj|) and f: f : b ↦𝔅 𝔉(|ObjMap|)(|a|)
    by (intro cat-obj-cf-comma-ObjD[OF assms(3)[unfolded assms(1)] assms(2)])+
  from assms(2) a f show ?thesis
    using assms(2)
    unfolding assms(1) op-obj-cf-comma-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: V-cs-intros cat-cs-intros cat-comma-cs-intros
      )
qed
```

14.6.3 Arrow map

```
lemma op-cf-obj-comma-ArrMap-vsv[cat-comma-cs-intros]:
  vsv (op-cf-obj-comma 𝔉 b(|ArrMap|))
  unfolding op-cf-obj-comma-def
  by
    (
```

```
    cs-concl cs-shallow
      cs-simp: cat-comma-cs-simps cs-intro: cat-comma-cs-intros
  )

lemma op-obj-cf-comma-ArrMap-vsv[cat-comma-cs-intros]:
  vsv (op-obj-cf-comma b 𝔉(|ArrMap|))
  unfolding op-obj-cf-comma-def
  by
    (
    cs-concl cs-shallow
      cs-simp: cat-comma-cs-simps cs-intro: cat-comma-cs-intros
    )

lemma (in is-functor) op-cf-obj-comma-ArrMap-vdomain[cat-comma-cs-simps]:
  𝒟∘ (op-cf-obj-comma 𝔉 b(|ArrMap|)) = 𝔉 CF↓ b(|Arr|)
  unfolding op-cf-obj-comma-def
  by
    (
    cs-concl cs-shallow
      cs-simp: cat-comma-cs-simps cat-cf-obj-comma-def[symmetric]
    )

lemmas [cat-comma-cs-simps] = is-functor.op-cf-obj-comma-ArrMap-vdomain

lemma (in is-functor) op-obj-cf-comma-ArrMap-vdomain[cat-comma-cs-simps]:
  𝒟∘ (op-obj-cf-comma b 𝔉(|ArrMap|)) = b ↓CF 𝔉(|Arr|)
  unfolding op-obj-cf-comma-def
  by
    (
    cs-concl cs-shallow
      cs-simp: cat-comma-cs-simps cat-obj-cf-comma-def[symmetric]
    )

lemmas [cat-comma-cs-simps] = is-functor.op-obj-cf-comma-ArrMap-vdomain

lemma (in is-functor) op-cf-obj-comma-ArrMap-app[cat-comma-cs-simps]:
  assumes ABF = [[a, 0, f]∘, [a', 0, f']∘, [g, 0]∘]∘
    and b ∈∘ 𝔅(|Obj|)
    and ABF ∈∘ 𝔉 CF↓ b(|Arr|)
  shows op-cf-obj-comma 𝔉 b(|ArrMap|)(|ABF|) = [[0, a', f']∘, [0, a, f]∘, [0, g]∘]∘
proof–
  from assms(3) have g: g : a ↦𝔄 a'
    and f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 b
    and f': f' : 𝔉(|ObjMap|)(|a'|) ↦𝔅 b
    and [cat-comma-cs-simps]: f' ∘A𝔅 𝔉(|ArrMap|)(|g|) = f
    by (intro cat-cf-obj-comma-ArrD[OF assms(3)[unfolded assms(1)] assms(2)])+
  from assms(2) g f f' show ?thesis
    unfolding assms(1) op-cf-obj-comma-def
    by
      (
      cs-concl
        cs-simp: cat-cs-simps cat-comma-cs-simps cat-1-CId-app
        cs-intro: V-cs-intros cat-cs-intros cat-comma-cs-intros cat-1-is-arrI
      )
qed

lemmas [cat-comma-cs-simps] = is-functor.op-cf-obj-comma-ArrMap-app
```

lemma (in *is-functor*) *op-obj-cf-comma-ArrMap-app*[*cat-comma-cs-simps*]:
assumes $ABF = [[0, a, f]_\circ, [0, a', f']_\circ, [0, h]_\circ]_\circ$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $ABF \in_\circ b \downarrow_{CF} \mathfrak{F}(\!|Arr|\!)$
shows *op-obj-cf-comma* $b\ \mathfrak{F}(\!|ArrMap|\!)(\!|ABF|\!) = [[a', 0, f']_\circ, [a, 0, f]_\circ, [h, 0]_\circ]_\circ$
proof–
from *assms*(*3*) **have** *h*: $h : a \mapsto_{\mathfrak{A}} a'$
and *f*: $f : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
and *f'*: $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$
and [*cat-comma-cs-simps*]: $\mathfrak{F}(\!|ArrMap|\!)(\!|h|\!) \circ_{A\mathfrak{B}} f = f'$
by (*intro cat-obj-cf-comma-ArrD*[*OF assms*(*3*)[*unfolded assms*(*1*)] *assms*(*2*)])+
from *assms*(*2*) *h f f'* **show** *?thesis*
unfolding *assms*(*1*) *op-obj-cf-comma-def*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps cat-1-CId-app*
cs-intro: *V-cs-intros cat-cs-intros cat-comma-cs-intros cat-1-is-arrI*
)
qed

lemmas [*cat-comma-cs-simps*] = *is-functor.op-obj-cf-comma-ArrMap-app*

14.6.4 Opposite comma category functors for the comma categories constructed from a functor and an object are isomorphisms of categories

lemma (in *is-functor*) *op-cf-obj-comma-is-iso-functor*:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows *op-cf-obj-comma* $\mathfrak{F}\ b : op\text{-}cat\ (\mathfrak{F}_{\ CF\downarrow}\ b) \mapsto\mapsto_{C.iso\alpha} b \downarrow_{CF} (op\text{-}cf\ \mathfrak{F})$
proof–
from *assms* **have** *cf-const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}\ b : cat\text{-}1\ 0\ 0 \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *V-cs-intros cat-cs-intros*)
note *cat-obj-cf-comma-def* =
is-functor.cat-obj-cf-comma-def[
OF is-functor-op, *unfolded cat-op-simps*
]
show *?thesis*
by
(
rule op-cf-comma-is-iso-functor
[
OF is-functor-axioms cf-const,
folded cat-cf-obj-comma-def op-cf-obj-comma-def,
unfolded cat-op-simps,
folded cat-obj-cf-comma-def
]
)
qed

lemma (in *is-functor*) *op-cf-obj-comma-is-iso-functor'*[*cat-comma-cs-intros*]:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $\mathfrak{A}' = op\text{-}cat\ (\mathfrak{F}_{\ CF\downarrow}\ b)$
and $\mathfrak{B}' = b \downarrow_{CF} (op\text{-}cf\ \mathfrak{F})$
shows *op-cf-obj-comma* $\mathfrak{F}\ b : \mathfrak{A}' \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}'$
using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule op-cf-obj-comma-is-iso-functor*)

lemmas [*cat-comma-cs-intros*] = *is-functor.op-cf-obj-comma-is-iso-functor'*

```
lemma (in is-functor) op-cf-obj-comma-is-functor:
  assumes b ∈₀ 𝔅(|Obj|)
  shows op-cf-obj-comma 𝔉 b : op-cat (𝔉 CF↓ b) ↦↦Cα b ↓CF (op-cf 𝔉)
  by (rule is-iso-functorD(1)[OF op-cf-obj-comma-is-iso-functor[OF assms]])

lemma (in is-functor) op-cf-obj-comma-is-functor'[cat-comma-cs-intros]:
  assumes b ∈₀ 𝔅(|Obj|)
    and 𝔄' = op-cat (𝔉 CF↓ b)
    and 𝔅' = b ↓CF (op-cf 𝔉)
  shows op-cf-obj-comma 𝔉 b : 𝔄' ↦↦Cα 𝔅'
  using assms(1) unfolding assms(2,3) by (rule op-cf-obj-comma-is-functor)

lemmas [cat-comma-cs-intros] = is-functor.op-cf-obj-comma-is-functor'

lemma (in is-functor) op-obj-cf-comma-is-iso-functor:
  assumes b ∈₀ 𝔅(|Obj|)
  shows op-obj-cf-comma b 𝔉 : op-cat (b ↓CF 𝔉) ↦↦C.isoα (op-cf 𝔉) CF↓ b
proof–
  from assms have cf-const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦Cα 𝔅
    by (cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros)
  note cat-cf-obj-comma-def =
    is-functor.cat-cf-obj-comma-def[
      OF is-functor-op, unfolded cat-op-simps
      ]
  show ?thesis
    by
      (
        rule op-cf-comma-is-iso-functor
          [
            OF cf-const is-functor-axioms,
            folded cat-obj-cf-comma-def op-obj-cf-comma-def,
            unfolded cat-op-simps,
            folded cat-cf-obj-comma-def
          ]
      )
qed

lemma (in is-functor) op-obj-cf-comma-is-iso-functor'[cat-comma-cs-intros]:
  assumes b ∈₀ 𝔅(|Obj|)
    and 𝔄' = op-cat (b ↓CF 𝔉)
    and 𝔅' = (op-cf 𝔉) CF↓ b
  shows op-obj-cf-comma b 𝔉 : 𝔄' ↦↦C.isoα 𝔅'
  using assms(1) unfolding assms(2,3) by (rule op-obj-cf-comma-is-iso-functor)

lemmas [cat-comma-cs-intros] = is-functor.op-obj-cf-comma-is-iso-functor'

lemma (in is-functor) op-obj-cf-comma-is-functor:
  assumes b ∈₀ 𝔅(|Obj|)
  shows op-obj-cf-comma b 𝔉 : op-cat (b ↓CF 𝔉) ↦↦Cα (op-cf 𝔉) CF↓ b
  by (rule is-iso-functorD(1)[OF op-obj-cf-comma-is-iso-functor[OF assms]])

lemma (in is-functor) op-obj-cf-comma-is-functor'[cat-comma-cs-intros]:
  assumes b ∈₀ 𝔅(|Obj|)
    and 𝔄' = op-cat (b ↓CF 𝔉)
    and 𝔅' = (op-cf 𝔉) CF↓ b
  shows op-obj-cf-comma b 𝔉 : 𝔄' ↦↦Cα 𝔅'
  using assms(1) unfolding assms(2,3) by (rule op-obj-cf-comma-is-functor)
```

14.7 Projections for comma categories constructed from a functor and an object

14.7.1 Definitions and elementary properties

definition *cf-cf-obj-comma-proj* :: $V \Rightarrow V \Rightarrow V$ ($\langle$(- $_{CF}\sqcap_{O}$ -)$\rangle$ [*1000*, *1000*] *999*)
  where $\mathfrak{F}\ {}_{CF}\sqcap_{O}\ b \equiv \mathfrak{F}\ {}_{CF}\sqcap\ (\mathit{cf\text{-}const}\ (\mathit{cat\text{-}1\ 0\ 0})\ (\mathfrak{F}(\!|\mathit{HomCod}|\!))\ b)$

definition *cf-obj-cf-comma-proj* :: $V \Rightarrow V \Rightarrow V$ ($\langle$(- $_{O}\sqcap_{CF}$ -)$\rangle$ [*1000*, *1000*] *999*)
  where $b\ {}_{O}\sqcap_{CF}\ \mathfrak{F} \equiv (\mathit{cf\text{-}const}\ (\mathit{cat\text{-}1\ 0\ 0})\ (\mathfrak{F}(\!|\mathit{HomCod}|\!))\ b) \sqcap_{CF} \mathfrak{F}$

Alternative forms of the definitions.

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-def*:
  $\mathfrak{F}\ {}_{CF}\sqcap_{O}\ b = \mathfrak{F}\ {}_{CF}\sqcap\ (\mathit{cf\text{-}const}\ (\mathit{cat\text{-}1\ 0\ 0})\ \mathfrak{B}\ b)$
  unfolding *cf-cf-obj-comma-proj-def cf-HomCod*..

lemma (**in** *is-functor*) *cf-obj-cf-comma-proj-def*:
  $b\ {}_{O}\sqcap_{CF}\ \mathfrak{F} = (\mathit{cf\text{-}const}\ (\mathit{cat\text{-}1\ 0\ 0})\ \mathfrak{B}\ b) \sqcap_{CF} \mathfrak{F}$
  unfolding *cf-obj-cf-comma-proj-def cf-HomCod*..

Components.

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-components*[*cat-comma-cs-simps*]:
  shows $\mathfrak{F}\ {}_{CF}\sqcap_{O}\ b(\!|\mathit{HomDom}|\!) = \mathfrak{F}\ {}_{CF}\!\downarrow\ b$
    and $\mathfrak{F}\ {}_{CF}\sqcap_{O}\ b(\!|\mathit{HomCod}|\!) = \mathfrak{A}$
  unfolding
    cf-cf-obj-comma-proj-def
    cf-comma-proj-left-components
    cat-cf-obj-comma-def[*symmetric*]
    cat-cs-simps
  by *simp-all*

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-obj-comma-proj-components*

lemma (**in** *is-functor*) *cf-obj-cf-comma-proj-components*[*cat-comma-cs-simps*]:
  shows $b\ {}_{O}\sqcap_{CF}\ \mathfrak{F}(\!|\mathit{HomDom}|\!) = b \downarrow_{CF} \mathfrak{F}$
    and $b\ {}_{O}\sqcap_{CF}\ \mathfrak{F}(\!|\mathit{HomCod}|\!) = \mathfrak{A}$
  unfolding
    cf-obj-cf-comma-proj-def
    cf-comma-proj-right-components
    cat-obj-cf-comma-def[*symmetric*]
    cat-cs-simps
  by *simp-all*

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-obj-cf-comma-proj-components*

14.7.2 Object map

lemma *cf-cf-obj-comma-proj-ObjMap-vsv*[*cat-comma-cs-intros*]:
  vsv ($\mathfrak{F}\ {}_{CF}\sqcap_{O}\ b(\!|\mathit{ObjMap}|\!)$)
  unfolding *cf-cf-obj-comma-proj-def*
  by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

lemma *cf-obj-cf-comma-proj-ObjMap-vsv*[*cat-comma-cs-intros*]:
  vsv ($b\ {}_{O}\sqcap_{CF}\ \mathfrak{F}(\!|\mathit{ObjMap}|\!)$)
  unfolding *cf-obj-cf-comma-proj-def*
  by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-ObjMap-vdomain*[*cat-comma-cs-simps*]:

$\mathcal{D}_\circ$ ($\mathfrak{F}_{\ CF}\bigsqcap_O b(\!|ObjMap|\!)$) = $\mathfrak{F}_{\ CF}\!\downarrow b(\!|Obj|\!)$
unfolding *cf-cf-obj-comma-proj-def cf-comma-proj-left-ObjMap-vdomain*
unfolding
cf-cf-obj-comma-proj-def[*symmetric*]
cf-comma-proj-left-components[*symmetric*]
cat-comma-cs-simps
by *simp*

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-obj-comma-proj-ObjMap-vdomain*

lemma (**in** *is-functor*) *cf-obj-cf-comma-proj-ObjMap-vdomain*[*cat-comma-cs-simps*]:
$\mathcal{D}_\circ$ ($b\ {}_O\!\bigsqcap_{CF}\ \mathfrak{F}(\!|ObjMap|\!)$) = $b\downarrow_{CF}\ \mathfrak{F}(\!|Obj|\!)$
unfolding *cf-obj-cf-comma-proj-def cf-comma-proj-right-ObjMap-vdomain*
unfolding
cf-obj-cf-comma-proj-def[*symmetric*]
cf-comma-proj-right-components[*symmetric*]
cat-comma-cs-simps
by *simp*

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-obj-cf-comma-proj-ObjMap-vdomain*

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-ObjMap-app*[*cat-comma-cs-simps*]:
assumes $A = [a, b', f]_\circ$ **and** $[a, b', f]_\circ \in_\circ \mathfrak{F}_{\ CF}\!\downarrow b(\!|Obj|\!)$
shows $\mathfrak{F}_{\ CF}\bigsqcap_O b(\!|ObjMap|\!)(\!|A|\!) = a$
by
(
rule cf-comma-proj-left-ObjMap-app[
OF assms(*1*) *assms*(*2*)[*unfolded cat-cf-obj-comma-def*],
folded cf-cf-obj-comma-proj-def
]
)

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-obj-comma-proj-ObjMap-app*

lemma (**in** *is-functor*) *cf-obj-cf-comma-proj-ObjMap-app*[*cat-comma-cs-simps*]:
assumes $A = [b', a, f]_\circ$ **and** $[b', a, f]_\circ \in_\circ b\downarrow_{CF}\ \mathfrak{F}(\!|Obj|\!)$
shows $b\ {}_O\!\bigsqcap_{CF}\ \mathfrak{F}(\!|ObjMap|\!)(\!|A|\!) = a$
by
(
rule cf-comma-proj-right-ObjMap-app[
OF assms(*1*) *assms*(*2*)[*unfolded cat-obj-cf-comma-def*],
folded cf-obj-cf-comma-proj-def
]
)

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-obj-cf-comma-proj-ObjMap-app*

14.7.3 Arrow map

lemma *cf-cf-obj-comma-proj-ArrMap-vsv*[*cat-comma-cs-intros*]:
vsv ($\mathfrak{F}_{\ CF}\bigsqcap_O b(\!|ArrMap|\!)$)
unfolding *cf-cf-obj-comma-proj-def*
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

lemma *cf-obj-cf-comma-proj-ArrMap-vsv*[*cat-comma-cs-intros*]:
vsv ($b\ {}_O\!\bigsqcap_{CF}\ \mathfrak{F}(\!|ArrMap|\!)$)
unfolding *cf-obj-cf-comma-proj-def*
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-ArrMap-vdomain*[*cat-comma-cs-simps*]:
  $\mathcal{D}_\circ$ ($\mathfrak{F}$ $_{CF}\sqcap_O$ $b(\!|ArrMap|\!)$) = $\mathfrak{F}$ $_{CF}\downarrow$ $b(\!|Arr|\!)$
  unfolding *cf-cf-obj-comma-proj-def cf-comma-proj-left-ArrMap-vdomain*
  unfolding
    cf-cf-obj-comma-proj-def[*symmetric*]
    cf-comma-proj-left-components[*symmetric*]
    cat-comma-cs-simps
  by *simp*

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-obj-comma-proj-ObjMap-vdomain*

lemma (**in** *is-functor*) *cf-obj-cf-comma-proj-ArrMap-vdomain*[*cat-comma-cs-simps*]:
  $\mathcal{D}_\circ$ (b $_O\sqcap_{CF}$ $\mathfrak{F}(\!|ArrMap|\!)$) = b $\downarrow_{CF}$ $\mathfrak{F}(\!|Arr|\!)$
  unfolding *cf-obj-cf-comma-proj-def cf-comma-proj-right-ArrMap-vdomain*
  unfolding
    cf-obj-cf-comma-proj-def[*symmetric*]
    cf-comma-proj-right-components[*symmetric*]
    cat-comma-cs-simps
  by *simp*

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-obj-cf-comma-proj-ArrMap-vdomain*

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-ArrMap-app*[*cat-comma-cs-simps*]:
  assumes ABF = [A, B, [g, h]$_\circ$]$_\circ$
    and [A, B, [g, h]$_\circ$]$_\circ$ $\in_\circ$ $\mathfrak{F}$ $_{CF}\downarrow$ $b(\!|Arr|\!)$
  shows $\mathfrak{F}$ $_{CF}\sqcap_O$ $b(\!|ArrMap|\!)(\!|ABF|\!)$ = g
  by
    (
      rule cf-comma-proj-left-ArrMap-app[
        OF assms(*1*) *assms*(*2*)[*unfolded cat-cf-obj-comma-def*],
        folded cf-cf-obj-comma-proj-def
        ]
)

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-obj-comma-proj-ArrMap-app*

lemma (**in** *is-functor*) *cf-obj-cf-comma-proj-ArrMap-app*[*cat-comma-cs-simps*]:
  assumes ABF = [A, B, [g, h]$_\circ$]$_\circ$
    and [A, B, [g, h]$_\circ$]$_\circ$ $\in_\circ$ b $\downarrow_{CF}$ $\mathfrak{F}(\!|Arr|\!)$
  shows b $_O\sqcap_{CF}$ $\mathfrak{F}(\!|ArrMap|\!)(\!|ABF|\!)$ = h
  by
    (
      rule cf-comma-proj-right-ArrMap-app[
        OF assms(*1*) *assms*(*2*)[*unfolded cat-obj-cf-comma-def*],
        folded cf-obj-cf-comma-proj-def
        ]
)

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-obj-cf-comma-proj-ArrMap-app*

14.7.4 Projections for a comma category are functors

lemma (**in** *is-functor*) *cf-cf-obj-comma-proj-is-functor*:
  assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
  shows $\mathfrak{F}$ $_{CF}\sqcap_O$ b : $\mathfrak{F}$ $_{CF}\downarrow$ b $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{A}$
proof–
  from *assms* **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$

```
    by (cs-concl cs-intro: V-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule cf-comma-proj-left-is-functor[
          OF is-functor-axioms const,
          folded cf-cf-obj-comma-proj-def cat-cf-obj-comma-def
          ]
      )
qed

lemma (in is-functor) cf-cf-obj-comma-proj-is-functor'[cat-comma-cs-intros]:
  assumes b ∈ₒ 𝔅(|Obj|) and 𝔄' = 𝔉 CF↓ b
  shows 𝔉 CF↑ O b : 𝔄' ↦↦Cα 𝔄
  using assms(1) unfolding assms(2) by (rule cf-cf-obj-comma-proj-is-functor)

lemmas [cat-comma-cs-intros] = is-functor.cf-cf-obj-comma-proj-is-functor'

lemma (in is-functor) cf-obj-cf-comma-proj-is-functor:
  assumes b ∈ₒ 𝔅(|Obj|)
  shows b O↑ CF 𝔉 : b ↓CF 𝔉 ↦↦Cα 𝔄
proof–
  from assms have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦Cα 𝔅
    by (cs-concl cs-intro: V-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule cf-comma-proj-right-is-functor[
          OF const is-functor-axioms,
          folded cf-obj-cf-comma-proj-def cat-obj-cf-comma-def
          ]
      )
qed

lemma (in is-functor) cf-obj-cf-comma-proj-is-functor'[cat-comma-cs-intros]:
  assumes b ∈ₒ 𝔅(|Obj|) and 𝔄' = b ↓CF 𝔉
  shows b O↑ CF 𝔉 : 𝔄' ↦↦Cα 𝔄
  using assms(1) unfolding assms(2) by (rule cf-obj-cf-comma-proj-is-functor)

lemmas [cat-comma-cs-intros] = is-functor.cf-obj-cf-comma-proj-is-functor'
```

14.7.5 Opposite projections for comma categories constructed from a functor and an object

```
lemma (in is-functor) op-cf-cf-obj-comma-proj:
  assumes b ∈ₒ 𝔅(|Obj|)
  shows op-cf (𝔉 CF↑ O b) = b O↑ CF (op-cf 𝔉) ∘CF op-cf-obj-comma 𝔉 b
proof–
  from assms have cf-const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦Cα 𝔅
    by (cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule op-cf-comma-proj-left
          [
            OF is-functor-axioms cf-const,
            unfolded cat-op-simps,
            folded
```

cf-cf-obj-comma-proj-def
op-cf-obj-comma-def
is-functor.cf-obj-cf-comma-proj-def[
OF is-functor-op, *unfolded cat-op-simps*
]
]
)
qed

lemma (**in** *is-functor*) *op-cf-obj-cf-comma-proj*:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows *op-cf* $(b\ {}_{O}\!\!\prod_{CF}\ \mathfrak{F}) = (\textit{op-cf}\ \mathfrak{F})\ {}_{CF}\!\!\prod_{O}\ b \circ_{CF}$ *op-obj-cf-comma* $b\ \mathfrak{F}$
proof–
from *assms* **have** *cf-const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *V-cs-intros cat-cs-intros*)
show *?thesis*
by
(
rule op-cf-comma-proj-right
[
OF cf-const is-functor-axioms,
unfolded cat-op-simps,
folded
cf-obj-cf-comma-proj-def
op-obj-cf-comma-def
is-functor.cf-cf-obj-comma-proj-def[
OF is-functor-op, *unfolded cat-op-simps*
]
]
)
qed

14.7.6 Projections for a tiny comma category

lemma (**in** *is-tm-functor*) *cf-cf-obj-comma-proj-is-tm-functor*:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathfrak{F}\ {}_{CF}\!\!\prod_{O}\ b : \mathfrak{F}\ {}_{CF}\!\!\downarrow\ b \mapsto\!\!\mapsto_{C.tm\alpha} \mathfrak{A}$
proof–
from *assms* **have** *const*: *cf-const* (*cat-1 0 0*) $\mathfrak{B}$ b : *cat-1 0 0* $\mapsto\!\!\mapsto_{C.tm\alpha}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *V-cs-intros cat-small-cs-intros cat-cs-intros*)
show *?thesis*
by
(
rule cf-comma-proj-left-is-tm-functor[
OF is-tm-functor-axioms const,
folded cf-cf-obj-comma-proj-def cat-cf-obj-comma-def
]
)
qed

lemma (**in** *is-tm-functor*) *cf-cf-obj-comma-proj-is-tm-functor'*[*cat-comma-cs-intros*]:
assumes $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** $\mathfrak{F}b = \mathfrak{F}\ {}_{CF}\!\!\downarrow\ b$
shows $\mathfrak{F}\ {}_{CF}\!\!\prod_{O}\ b : \mathfrak{F}b \mapsto\!\!\mapsto_{C.tm\alpha} \mathfrak{A}$
using *assms*(*1*) **unfolding** *assms*(*2*) **by** (*rule cf-cf-obj-comma-proj-is-tm-functor*)

lemmas [*cat-comma-cs-intros*] = *is-tm-functor.cf-cf-obj-comma-proj-is-tm-functor'*

lemma (**in** *is-tm-functor*) *cf-obj-cf-comma-proj-is-tm-functor*:

```
  assumes b ∈∘ 𝔅(|Obj|)
  shows b ₀⨅꜀ₓ 𝔉 : b ↓꜀ₓ 𝔉 ↦↦C.tmα 𝔄
proof–
  from assms have const: cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦C.tmα 𝔅
    by (cs-concl cs-intro: V-cs-intros cat-small-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule cf-comma-proj-right-is-tm-functor[
          OF const is-tm-functor-axioms,
          folded cf-obj-cf-comma-proj-def cat-obj-cf-comma-def
          ]
      )
qed

lemma (in is-tm-functor) cf-obj-cf-comma-proj-is-tm-functor'[cat-comma-cs-intros]:
  assumes b ∈∘ 𝔅(|Obj|) and 𝔄' = b ↓꜀ₓ 𝔉
  shows b ₀⨅꜀ₓ 𝔉 : 𝔄' ↦↦C.tmα 𝔄
  using assms(1) unfolding assms(2) by (rule cf-obj-cf-comma-proj-is-tm-functor)

lemmas [cat-comma-cs-intros] = is-tm-functor.cf-obj-cf-comma-proj-is-tm-functor'

lemma cf-comp-cf-cf-obj-comma-proj-is-tm-functor[cat-comma-cs-intros]:
  assumes 𝔊 : 𝔄 ↦↦Cα ℭ
    and 𝔉 : 𝔍 ↦↦C.tmα 𝔊 ₀꜀ₓ↓ c
    and c ∈∘ ℭ(|Obj|)
  shows 𝔊 ₀꜀ₓ⨅₀ c ∘꜀ₓ 𝔉 : 𝔍 ↦↦C.tmα 𝔄
proof–
  interpret 𝔊: is-functor α 𝔄 ℭ 𝔊 by (rule assms(1))
  from assms(3) have cf-const: cf-const (cat-1 0 0) ℭ c : cat-1 0 0 ↦↦Cα ℭ
    by (cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule cf-comp-cf-comma-proj-left-is-tm-functor
          [
            OF assms(1) - assms(2)[unfolded cat-cf-obj-comma-def],
            unfolded cat-cs-simps,
            OF cf-const,
            folded 𝔊.cf-cf-obj-comma-proj-def
          ]
      )
qed

lemma cf-comp-cf-obj-cf-comma-proj-is-tm-functor[cat-comma-cs-intros]:
  assumes ℌ : 𝔅 ↦↦Cα ℭ
    and 𝔉 : 𝔍 ↦↦C.tmα c ↓꜀ₓ ℌ
    and c ∈∘ ℭ(|Obj|)
  shows c ₀⨅꜀ₓ ℌ ∘꜀ₓ 𝔉 : 𝔍 ↦↦C.tmα 𝔅
proof–
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(1))
  from assms(3) have cf-const: cf-const (cat-1 0 0) ℭ c : cat-1 0 0 ↦↦Cα ℭ
    by (cs-concl cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros)
  show ?thesis
    by
      (
        rule cf-comp-cf-comma-proj-right-is-tm-functor
          [
```

OF - assms(1) assms(2)[*unfolded cat-obj-cf-comma-def*],
unfolded cat-cs-simps,
OF cf-const,
folded $\mathfrak{H}$*.cf-obj-cf-comma-proj-def*
]
)
qed

14.8 Comma functors

14.8.1 Definition and elementary properties

See Theorem 1 in Chapter X-3 in [7].

definition *cf-arr-cf-comma* :: $V \Rightarrow V \Rightarrow V$
(‹(- $_{A}\downarrow_{CF}$ -)› [*1000*, *1000*] *999*)
where $g\ {}_{A}\downarrow_{CF}\ \mathfrak{F} =$
[
$(\lambda A \in_{\circ} (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Cod \rfloor)(\lfloor g \rfloor))\ {\downarrow_{CF}}\ \mathfrak{F}(\lfloor Obj \rfloor).\ [0,\ A(\lfloor 1_{\mathbb{N}} \rfloor),\ A(\lfloor 2_{\mathbb{N}} \rfloor) \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ g]_{\circ})$,
(
$\lambda F \in_{\circ} (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Cod \rfloor)(\lfloor g \rfloor))\ {\downarrow_{CF}}\ \mathfrak{F}(\lfloor Arr \rfloor).$
[
$[0,\ F(\lfloor 0 \rfloor)(\lfloor 1_{\mathbb{N}} \rfloor),\ F(\lfloor 0 \rfloor)(\lfloor 2_{\mathbb{N}} \rfloor) \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ g]_{\circ}$,
$[0,\ F(\lfloor 1_{\mathbb{N}} \rfloor)(\lfloor 1_{\mathbb{N}} \rfloor),\ F(\lfloor 1_{\mathbb{N}} \rfloor)(\lfloor 2_{\mathbb{N}} \rfloor) \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ g]_{\circ}$,
$F(\lfloor 2_{\mathbb{N}} \rfloor)$
]$_{\circ}$
),
$(\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Cod \rfloor)(\lfloor g \rfloor))\ {\downarrow_{CF}}\ \mathfrak{F}$,
$(\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Dom \rfloor)(\lfloor g \rfloor))\ {\downarrow_{CF}}\ \mathfrak{F}$
]$_{\circ}$

definition *cf-cf-arr-comma* :: $V \Rightarrow V \Rightarrow V$
(‹(- $_{CF}\downarrow_{A}$ -)› [*1000*, *1000*] *999*)
where $\mathfrak{F}\ {}_{CF}\downarrow_{A}\ g =$
[
$(\lambda A \in_{\circ} \mathfrak{F}\ {}_{CF}\downarrow\ (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Dom \rfloor)(\lfloor g \rfloor))(\lfloor Obj \rfloor).\ [A(\lfloor 0 \rfloor),\ 0,\ g \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ A(\lfloor 2_{\mathbb{N}} \rfloor)]_{\circ})$,
(
$\lambda F \in_{\circ} \mathfrak{F}\ {}_{CF}\downarrow\ (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Dom \rfloor)(\lfloor g \rfloor))(\lfloor Arr \rfloor).$
[
$[F(\lfloor 0 \rfloor)(\lfloor 0 \rfloor),\ 0,\ g \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ F(\lfloor 0 \rfloor)(\lfloor 2_{\mathbb{N}} \rfloor)]_{\circ}$,
$[F(\lfloor 1_{\mathbb{N}} \rfloor)(\lfloor 0 \rfloor),\ 0,\ g \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ F(\lfloor 1_{\mathbb{N}} \rfloor)(\lfloor 2_{\mathbb{N}} \rfloor)]_{\circ}$,
$F(\lfloor 2_{\mathbb{N}} \rfloor)$
]$_{\circ}$
),
$\mathfrak{F}\ {}_{CF}\downarrow\ (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Dom \rfloor)(\lfloor g \rfloor))$,
$\mathfrak{F}\ {}_{CF}\downarrow\ (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Cod \rfloor)(\lfloor g \rfloor))$
]$_{\circ}$

Components.

lemma *cf-arr-cf-comma-components*:
shows $g\ {}_{A}\downarrow_{CF}\ \mathfrak{F}(\lfloor ObjMap \rfloor) =$
$(\lambda A \in_{\circ} (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Cod \rfloor)(\lfloor g \rfloor))\ {\downarrow_{CF}}\ \mathfrak{F}(\lfloor Obj \rfloor).\ [0,\ A(\lfloor 1_{\mathbb{N}} \rfloor),\ A(\lfloor 2_{\mathbb{N}} \rfloor) \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ g]_{\circ})$
and $g\ {}_{A}\downarrow_{CF}\ \mathfrak{F}(\lfloor ArrMap \rfloor) =$
(
$\lambda F \in_{\circ} (\mathfrak{F}(\lfloor HomCod \rfloor)(\lfloor Cod \rfloor)(\lfloor g \rfloor))\ {\downarrow_{CF}}\ \mathfrak{F}(\lfloor Arr \rfloor).$
[
$[0,\ F(\lfloor 0 \rfloor)(\lfloor 1_{\mathbb{N}} \rfloor),\ F(\lfloor 0 \rfloor)(\lfloor 2_{\mathbb{N}} \rfloor) \circ_{A \mathfrak{F}(\lfloor HomCod \rfloor)}\ g]_{\circ}$,

[0, F(1_ℕ)(1_ℕ), F(1_ℕ)(2_ℕ) ∘_{A𝔉(HomCod)} g]∘,
F(2_ℕ)
]∘
)
and g _A↓_{CF} 𝔉(HomDom) = (𝔉(HomCod)(Cod)(g)) ↓_{CF} 𝔉
and g _A↓_{CF} 𝔉(HomCod) = (𝔉(HomCod)(Dom)(g)) ↓_{CF} 𝔉
unfolding cf-arr-cf-comma-def dghm-field-simps
by (simp-all add: nat-omega-simps)

lemma cf-cf-arr-comma-components:
shows 𝔉 _{CF}↓_A g(ObjMap) =
(λA∈∘𝔉 _{CF}↓ (𝔉(HomCod)(Dom)(g))(Obj). [A(0), 0, g ∘_{A𝔉(HomCod)} A(2_ℕ)]∘)
and 𝔉 _{CF}↓_A g(ArrMap) =
(
λF∈∘𝔉 _{CF}↓ (𝔉(HomCod)(Dom)(g))(Arr).
[
[F(0)(0), 0, g ∘_{A𝔉(HomCod)} F(0)(2_ℕ)]∘,
[F(1_ℕ)(0), 0, g ∘_{A𝔉(HomCod)} F(1_ℕ)(2_ℕ)]∘,
F(2_ℕ)
]∘
)
and 𝔉 _{CF}↓_A g(HomDom) = 𝔉 _{CF}↓ (𝔉(HomCod)(Dom)(g))
and 𝔉 _{CF}↓_A g(HomCod) = 𝔉 _{CF}↓ (𝔉(HomCod)(Cod)(g))
unfolding cf-cf-arr-comma-def dghm-field-simps
by (simp-all add: nat-omega-simps)

context is-functor
begin

lemma cf-arr-cf-comma-components′:
assumes g : c ↦_𝔅 c′
shows g _A↓_{CF} 𝔉(ObjMap) = (λA∈∘c′ ↓_{CF} 𝔉(Obj). [0, A(1_ℕ), A(2_ℕ) ∘_{A𝔅} g]∘)
and g _A↓_{CF} 𝔉(ArrMap) =
(
λF∈∘c′ ↓_{CF} 𝔉(Arr).
[
[0, F(0)(1_ℕ), F(0)(2_ℕ) ∘_{A𝔅} g]∘,
[0, F(1_ℕ)(1_ℕ), F(1_ℕ)(2_ℕ) ∘_{A𝔅} g]∘,
F(2_ℕ)
]∘
)
and [cat-comma-cs-simps]: g _A↓_{CF} 𝔉(HomDom) = c′ ↓_{CF} 𝔉
and [cat-comma-cs-simps]: g _A↓_{CF} 𝔉(HomCod) = c ↓_{CF} 𝔉
using assms
unfolding cf-arr-cf-comma-components
by (simp-all add: cat-cs-simps)

lemma cf-cf-arr-comma-components′:
assumes g : c ↦_𝔅 c′
shows 𝔉 _{CF}↓_A g(ObjMap) = (λA∈∘𝔉 _{CF}↓ c(Obj). [A(0), 0, g ∘_{A𝔅} A(2_ℕ)]∘)
and 𝔉 _{CF}↓_A g(ArrMap) =
(
λF∈∘𝔉 _{CF}↓ c(Arr).
[
[F(0)(0), 0, g ∘_{A𝔅} F(0)(2_ℕ)]∘,
[F(1_ℕ)(0), 0, g ∘_{A𝔅} F(1_ℕ)(2_ℕ)]∘,
F(2_ℕ)

]∘
)
and [*cat-comma-cs-simps*]: $\mathfrak{F}_{\ CF\downarrow_A\ g(\!|HomDom|\!)} = \mathfrak{F}_{\ CF\downarrow\ c}$
and [*cat-comma-cs-simps*]: $\mathfrak{F}_{\ CF\downarrow_A\ g(\!|HomCod|\!)} = \mathfrak{F}_{\ CF\downarrow\ c'}$
using *assms*
unfolding *cf-cf-arr-comma-components*
by (*simp-all add*: *cat-cs-simps*)

end

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-arr-cf-comma-components'*(*3*,*4*)
lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-arr-comma-components'*(*3*,*4*)

14.8.2 Object map

mk-VLambda *cf-arr-cf-comma-components*(*1*)[*unfolded VLambda-vid-on*[*symmetric*]]
|*vsv cf-arr-cf-comma-ObjMap-vsv*[*cat-comma-cs-intros*]|

mk-VLambda *cf-cf-arr-comma-components*(*1*)[*unfolded VLambda-vid-on*[*symmetric*]]
|*vsv cf-cf-arr-comma-ObjMap-vsv*[*cat-comma-cs-intros*]|

context *is-functor*
begin

context
fixes *g c c'*
assumes *g*: $g : c \mapsto_{\mathfrak{B}} c'$
begin

mk-VLambda
cf-arr-cf-comma-components'(*1*)[*OF g*, *unfolded VLambda-vid-on*[*symmetric*]]
|*vdomain cf-arr-cf-comma-ObjMap-vdomain*[*cat-comma-cs-simps*]|

mk-VLambda
cf-cf-arr-comma-components'(*1*)[*OF g*, *unfolded VLambda-vid-on*[*symmetric*]]
|*vdomain cf-cf-arr-comma-ObjMap-vdomain*[*cat-comma-cs-simps*]|

end

end

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-arr-cf-comma-ObjMap-vdomain*
lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-arr-comma-ObjMap-vdomain*

lemma (**in** *is-functor*) *cf-arr-cf-comma-ObjMap-app*[*cat-comma-cs-simps*]:
assumes $A = [a', b', f']_\circ$ **and** $A \in_\circ c' \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$ **and** $g : c \mapsto_{\mathfrak{B}} c'$
shows $g_{\ A\downarrow_{CF}\ \mathfrak{F}(\!|ObjMap|\!)(\!|A|\!)} = [a', b', f' \circ_{A\mathfrak{B}} g]_\circ$
proof–
from *assms* **have** *b'*: $b' \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *f*: $f' : c' \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b'|\!)$
and *a'-def*: $a' = 0$
by *auto*
from *assms*(*2*) **show** *?thesis*
unfolding *cf-arr-cf-comma-components'*[*OF assms*(*3*)] *assms*(*1*)
by (*simp add*: *nat-omega-simps a'-def*)
qed

lemma (**in** *is-functor*) *cf-cf-arr-comma-ObjMap-app*[*cat-comma-cs-simps*]:

assumes $A = [a', b', f']_\circ$ **and** $A \in_\circ \mathfrak{F}_{CF}\downarrow c(\!|Obj|\!)$ **and** $g : c \mapsto_{\mathfrak{B}} c'$
shows $\mathfrak{F}_{CF}\downarrow_A g(\!|ObjMap|\!)(\!|A|\!) = [a', b', g \circ_{A\mathfrak{B}} f']_\circ$
proof–
from *assms* **have** *b'-def*: $b' = 0$
and *f*: $f' : \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{B}} c$
and *a'*: $a' \in_\circ \mathfrak{A}(\!|Obj|\!)$
by *auto*
from *assms*(*2*) **show** *?thesis*
unfolding *cf-cf-arr-comma-components'*[*OF assms*(*3*)] *assms*(*1*)
by (*simp add*: *nat-omega-simps b'-def*)
qed

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-arr-cf-comma-ObjMap-app*
lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-arr-comma-ObjMap-app*

lemma (**in** *is-functor*) *cf-arr-cf-comma-ObjMap-vrange*:
assumes $g : c \mapsto_{\mathfrak{B}} c'$
shows $\mathcal{R}_\circ\,(g\ {}_A\!\downarrow_{CF} \mathfrak{F}(\!|ObjMap|\!)) \subseteq_\circ c \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
proof
(
rule vsv.vsv-vrange-vsubset,
unfold cf-arr-cf-comma-ObjMap-vdomain[*OF assms*]
)
fix *A* **assume** $A \in_\circ c' \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
with *assms is-functor-axioms* **obtain** *a f*
where *A-def*: $A = [0, a, f]_\circ$
and *a*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *f*: $f : c' \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
by *auto*
from *assms a f* **show** $g\ {}_A\!\downarrow_{CF} \mathfrak{F}(\!|ObjMap|\!)(\!|A|\!) \in_\circ c \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps A-def*
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
qed (*cs-concl* **cs-shallow** **cs-intro**: *cat-comma-cs-intros*)

lemma (**in** *is-functor*) *cf-cf-arr-comma-ObjMap-vrange*:
assumes $g : c \mapsto_{\mathfrak{B}} c'$
shows $\mathcal{R}_\circ\,(\mathfrak{F}_{CF}\downarrow_A g(\!|ObjMap|\!)) \subseteq_\circ \mathfrak{F}_{CF}\downarrow c'(\!|Obj|\!)$
proof
(
rule vsv.vsv-vrange-vsubset,
unfold cf-cf-arr-comma-ObjMap-vdomain[*OF assms*]
)
fix *A* **assume** $A \in_\circ \mathfrak{F}_{CF}\downarrow c(\!|Obj|\!)$
with *assms is-functor-axioms* **obtain** *a f*
where *A-def*: $A = [a, 0, f]_\circ$
and *a*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and *f*: $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} c$
by *auto*
from *assms a f* **show** $\mathfrak{F}_{CF}\downarrow_A g(\!|ObjMap|\!)(\!|A|\!) \in_\circ \mathfrak{F}_{CF}\downarrow c'(\!|Obj|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-comma-cs-simps A-def*
cs-intro: *cat-cs-intros cat-comma-cs-intros*

)
qed (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)

14.8.3 Arrow map

mk-VLambda *cf-arr-cf-comma-components*(*2*)
|*vsv cf-arr-cf-comma-ArrMap-vsv*[*cat-comma-cs-intros*]|

mk-VLambda *cf-cf-arr-comma-components*(*2*)
|*vsv cf-cf-arr-comma-ArrMap-vsv*[*cat-comma-cs-intros*]|

context *is-functor*
begin

context
fixes g c c'
assumes *g*: $g : c \mapsto_{\mathfrak{B}} c'$
begin

mk-VLambda
cf-arr-cf-comma-components'(*2*)[*OF g*, *unfolded VLambda-vid-on*[*symmetric*]]
|*vdomain cf-arr-cf-comma-ArrMap-vdomain*[*cat-comma-cs-simps*]|

mk-VLambda
cf-cf-arr-comma-components'(*2*)[*OF g*, *unfolded VLambda-vid-on*[*symmetric*]]
|*vdomain cf-cf-arr-comma-ArrMap-vdomain*[*cat-comma-cs-simps*]|

end

end

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-arr-cf-comma-ArrMap-vdomain*
lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-arr-comma-ArrMap-vdomain*

lemma (**in** *is-functor*) *cf-arr-cf-comma-ArrMap-app*[*cat-comma-cs-simps*]:
assumes $A = [[a, b, f]_\circ, [a', b', f']_\circ, [h, k]_\circ]_\circ$
and $[[a, b, f]_\circ, [a', b', f']_\circ, [h, k]_\circ]_\circ :$
$[a, b, f]_\circ \mapsto_{c' \downarrow_{CF}} \mathfrak{F} [a', b', f']_\circ$
and $g : c \mapsto_{\mathfrak{B}} c'$
shows $g_{A\downarrow_{CF}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor A \rfloor) =$
$[[a, b, f \circ_{A\mathfrak{B}} g]_\circ, [a', b', f' \circ_{A\mathfrak{B}} g]_\circ, [h, k]_\circ]_\circ$
proof–
from *assms*(*3*) **have** *c'*: $c' \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$ **by** *auto*
from
cat-obj-cf-comma-is-arrD(*1,2*)[*OF assms*(*2*)[*unfolded cat-comma-cs-simps*] *c'*]
is-arrD(*1*)[*OF assms*(*2*)]
show *?thesis*
unfolding *assms*(*1*) *cf-arr-cf-comma-components'*[*OF assms*(*3*)]
by (*simp-all add*: *nat-omega-simps*)
qed

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-arr-cf-comma-ArrMap-app*

lemma (**in** *is-functor*) *cf-cf-arr-comma-ArrMap-app*[*cat-comma-cs-simps*]:
assumes $A = [[a, b, f]_\circ, [a', b', f']_\circ, [h, k]_\circ]_\circ$
and $[[a, b, f]_\circ, [a', b', f']_\circ, [h, k]_\circ]_\circ :$
$[a, b, f]_\circ \mapsto_{\mathfrak{F}\ _{CF}\downarrow c} [a', b', f']_\circ$
and $g : c \mapsto_{\mathfrak{B}} c'$

shows $\mathfrak{F}_{CF\downarrow A}\ g(\!|ArrMap|\!)(\!|A|\!) =$
$[[a,\ b,\ g \circ_{A\mathfrak{B}} f]_\circ, [a',\ b',\ g \circ_{A\mathfrak{B}} f']_\circ, [h,\ k]_\circ]_\circ$
proof–
from *assms*(*3*) **have** *c*: $c \in_\circ \mathfrak{B}(\!|Obj|\!)$ **by** *auto*
from
cat-cf-obj-comma-is-arrD(*1,2*)[*OF assms*(*2*)[*unfolded cat-comma-cs-simps*] *c*]
is-arrD(*1*)[*OF assms*(*2*)]
show *?thesis*
unfolding *assms*(*1*) *cf-cf-arr-comma-components'*[*OF assms*(*3*)]
by (*simp-all add*: *nat-omega-simps*)
qed

lemmas [*cat-comma-cs-simps*] = *is-functor.cf-cf-arr-comma-ArrMap-app*

14.8.4 Comma functors are functors

lemma (**in** *is-functor*) *cf-arr-cf-comma-is-functor*:
assumes $g : c \mapsto_{\mathfrak{B}} c'$
shows $g_{\ A\downarrow CF}\ \mathfrak{F} : c' \downarrow_{CF} \mathfrak{F} \mapsto\mapsto_{C\alpha} c \downarrow_{CF} \mathfrak{F}$
proof(*rule is-functorI'*)
show *vfsequence* $(g_{\ A\downarrow CF}\ \mathfrak{F})$ **unfolding** *cf-arr-cf-comma-def* **by** *simp*
from *assms* **show** *category* α $(c' \downarrow_{CF} \mathfrak{F})$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
from *assms* **show** *category* α $(c \downarrow_{CF} \mathfrak{F})$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
show *vcard* $(g_{\ A\downarrow CF}\ \mathfrak{F}) = \aleph_0$
unfolding *cf-arr-cf-comma-def* **by** (*simp-all add*: *nat-omega-simps*)
from *assms* **show** $\mathcal{R}_\circ\ (g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ObjMap|\!)) \subseteq_\circ c \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by (*intro cf-arr-cf-comma-ObjMap-vrange*)
show $g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ArrMap|\!)(\!|F|\!) :$
$g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ObjMap|\!)(\!|A|\!) \mapsto_{c\ \downarrow_{CF}\ \mathfrak{F}} g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ObjMap|\!)(\!|B|\!)$
if $F : A \mapsto_{c'\ \downarrow_{CF}\ \mathfrak{F}} B$ **for** *A B F*
proof–
from *assms that* **obtain** *b f b' f' k*
where *F-def*: $F = [[0,\ b,\ f]_\circ, [0,\ b',\ f']_\circ, [0,\ k]_\circ]_\circ$
and *A-def*: $A = [0,\ b,\ f]_\circ$
and *B-def*: $B = [0,\ b',\ f']_\circ$
and *k*: $k : b \mapsto_{\mathfrak{A}} b'$
and *f*: $f : c' \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
and *f'*: $f' : c' \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b'|\!)$
and *f'-def*: $\mathfrak{F}(\!|ArrMap|\!)(\!|k|\!) \circ_{A\mathfrak{B}} f = f'$
by *auto*
from *assms that k f f'* **show** *?thesis*
unfolding *F-def A-def B-def*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps f'-def*[*symmetric*]
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
qed
show $g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ArrMap|\!)(\!|G \circ_{A\ c'\ \downarrow_{CF}\ \mathfrak{F}} F|\!) =$
$g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ArrMap|\!)(\!|G|\!) \circ_{A\ c\ \downarrow_{CF}\ \mathfrak{F}} g_{\ A\downarrow CF}\ \mathfrak{F}(\!|ArrMap|\!)(\!|F|\!)$
if $G : B \mapsto_{c'\ \downarrow_{CF}\ \mathfrak{F}} C$ **and** $F : A \mapsto_{c'\ \downarrow_{CF}\ \mathfrak{F}} B$ **for** *B C G A F*
proof–
from *that*(*2*) *assms* **obtain** *b f b' f' k*
where *F-def*: $F = [[0,\ b,\ f]_\circ, [0,\ b',\ f']_\circ, [0,\ k]_\circ]_\circ$
and *A-def*: $A = [0,\ b,\ f]_\circ$

```
      and B-def: B = [0, b', f']∘
      and k: k : b ↦₍𝔄₎ b'
      and f: f : c' ↦₍𝔅₎ 𝔉(|ObjMap|)(|b|)
      and f': f' : c' ↦₍𝔅₎ 𝔉(|ObjMap|)(|b'|)
      and f'-def: 𝔉(|ArrMap|)(|k|) ∘₍A𝔅₎ f = f'
    by auto
  with that(1) assms obtain b'' f'' k'
    where G-def: G = [[0, b', f']∘, [0, b'', f'']∘, [0, k']∘]∘
      and C-def: C = [0, b'', f'']∘
      and k': k' : b' ↦₍𝔄₎ b''
      and f'': f'' : c' ↦₍𝔅₎ 𝔉(|ObjMap|)(|b''|)
      and f''-def: 𝔉(|ArrMap|)(|k'|) ∘₍A𝔅₎ f' = f''
    by auto
  from assms that k f f' f'' k' show ?thesis
    unfolding F-def G-def A-def B-def C-def
    by
      (
        cs-concl
          cs-simp:
            cat-cs-simps cat-comma-cs-simps
            f''-def[symmetric] f'-def[symmetric]
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed
show g ₍A↓CF₎ 𝔉(|ArrMap|)(|c' ↓₍CF₎ 𝔉(|CId|)(|C|)|) = c ↓₍CF₎ 𝔉(|CId|)(|g ₍A↓CF₎ 𝔉(|ObjMap|)(|C|)|)
  if C ∈∘ c' ↓₍CF₎ 𝔉(|Obj|) for C
proof–
  from that assms obtain a f
    where C-def: C = [0, a, f]∘
      and a: a ∈∘ 𝔄(|Obj|)
      and f: f : c' ↦₍𝔅₎ 𝔉(|ObjMap|)(|a|)
    by auto
  from a assms f show
    g ₍A↓CF₎ 𝔉(|ArrMap|)(|c' ↓₍CF₎ 𝔉(|CId|)(|C|)|) = c ↓₍CF₎ 𝔉(|CId|)(|g ₍A↓CF₎ 𝔉(|ObjMap|)(|C|)|)
    unfolding C-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed
qed
  (
    use assms in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      ›
  )+

lemma (in is-functor) cf-cf-arr-comma-is-functor:
  assumes g : c ↦₍𝔅₎ c'
  shows 𝔉 ₍CF↓A₎ g : 𝔉 ₍CF↓₎ c ↦↦₍Cα₎ 𝔉 ₍CF↓₎ c'
proof(rule is-functorI')
  from assms have c: c ∈∘ 𝔅(|Obj|) by auto
  show vfsequence (𝔉 ₍CF↓A₎ g) unfolding cf-cf-arr-comma-def by simp
```

```
from assms show category α (𝔉 CF↓ c′)
  by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
from assms show category α (𝔉 CF↓ c)
  by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)
show vcard (𝔉 CF↓A g) = ℶ₀
  unfolding cf-cf-arr-comma-def by (simp-all add: nat-omega-simps)
from assms show ℛ∘ (𝔉 CF↓A g(|ObjMap|)) ⊆∘ 𝔉 CF↓ c′(|Obj|)
  by (intro cf-cf-arr-comma-ObjMap-vrange)
show 𝔉 CF↓A g(|ArrMap|)(|F|) :
  𝔉 CF↓A g(|ObjMap|)(|A|) ↦𝔉 CF↓ c′ 𝔉 CF↓A g(|ObjMap|)(|B|)
  if F : A ↦𝔉 CF↓ c B for A B F
proof–
  from assms that obtain a f a′ f′ h
    where F-def: F = [[a, 0, f]∘, [a′, 0, f′]∘, [h, 0]∘]∘
      and A-def: A = [a, 0, f]∘
      and B-def: B = [a′, 0, f′]∘
      and h: h : a ↦𝔄 a′
      and f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 c
      and f′: f′ : 𝔉(|ObjMap|)(|a′|) ↦𝔅 c
      and f′-def: f′ ∘A𝔅 𝔉(|ArrMap|)(|h|) = f
    by auto
  from assms that h f f′ show ?thesis
    unfolding F-def A-def B-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-comma-cs-simps f′-def
          cs-intro: cat-cs-intros cat-comma-cs-intros
      )
qed
show 𝔉 CF↓A g(|ArrMap|)(|G ∘A𝔉 CF↓ c F|) =
  𝔉 CF↓A g(|ArrMap|)(|G|) ∘A𝔉 CF↓ c′ 𝔉 CF↓A g(|ArrMap|)(|F|)
  if G : B ↦𝔉 CF↓ c C and F : A ↦𝔉 CF↓ c B for B C G A F
proof–
  from that(2) assms obtain a f a′ f′ h
    where F-def: F = [[a, 0, f]∘, [a′, 0, f′]∘, [h, 0]∘]∘
      and A-def: A = [a, 0, f]∘
      and B-def: B = [a′, 0, f′]∘
      and h: h : a ↦𝔄 a′
      and f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 c
      and f′: f′ : 𝔉(|ObjMap|)(|a′|) ↦𝔅 c
      and [cat-cs-simps]: f′ ∘A𝔅 𝔉(|ArrMap|)(|h|) = f
    by auto
  with that(1) assms obtain a″ f″ h′
    where G-def: G = [[a′, 0, f′]∘, [a″, 0, f″]∘, [h′, 0]∘]∘
      and C-def: C = [a″, 0, f″]∘
      and h′: h′ : a′ ↦𝔄 a″
      and f″: f″ : 𝔉(|ObjMap|)(|a″|) ↦𝔅 c
      and [cat-cs-simps]: f″ ∘A𝔅 𝔉(|ArrMap|)(|h′|) = f′
    by auto
  note [cat-cs-simps] = category.cat-assoc-helper[
      where 𝔄=𝔅, where h=f″ and g=‹𝔉(|ArrMap|)(|h′|)› and q=f′
      ]
  from assms that c h f f′ f″ h′ show ?thesis
    unfolding F-def G-def A-def B-def C-def
    by
      (
        cs-concl cs-shallow
```

```
            cs-simp: cat-cs-simps cat-comma-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed
  show 𝔉 CF↓A g(|ArrMap|)(|𝔉 CF↓ c(|CId|)(|C|)|) = 𝔉 CF↓ c'(|CId|)(|𝔉 CF↓A g(|ObjMap|)(|C|)|)
    if C ∈∘ 𝔉 CF↓ c(|Obj|) for C
  proof–
    from that assms obtain a f
      where C-def: C = [a, 0, f]∘
        and a: a ∈∘ 𝔄(|Obj|)
        and f: f : 𝔉(|ObjMap|)(|a|) ↦𝔅 c
      by auto
    from a c assms f show ?thesis
      unfolding C-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed
qed
  (
    use assms in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-comma-cs-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros
      ›
  )+

lemma (in is-functor) cf-arr-cf-comma-is-functor'[cat-comma-cs-intros]:
  assumes g : c ↦𝔅 c' and 𝔄' = c' ↓CF 𝔉 and 𝔅' = c ↓CF 𝔉
  shows g A↓CF 𝔉 : 𝔄' ↦↦Cα 𝔅'
  using assms(1) unfolding assms(2,3) by (rule cf-arr-cf-comma-is-functor(1))

lemmas [cat-comma-cs-intros] = is-functor.cf-arr-cf-comma-is-functor'

lemma (in is-functor) cf-cf-arr-comma-is-functor'[cat-comma-cs-intros]:
  assumes g : c ↦𝔅 c' and 𝔄' = 𝔉 CF↓ c and 𝔅' = 𝔉 CF↓ c'
  shows 𝔉 CF↓A g : 𝔄' ↦↦Cα 𝔅'
  using assms(1) unfolding assms(2,3) by (rule cf-cf-arr-comma-is-functor(1))

lemmas [cat-comma-cs-intros] = is-functor.cf-cf-arr-comma-is-functor'

lemma (in is-functor) cf-arr-cf-comma-CId:
  assumes b ∈∘ 𝔅(|Obj|)
  shows (𝔅(|CId|)(|b|)) A↓CF 𝔉 = cf-id (b ↓CF 𝔉)
proof(rule cf-eqI)
  from vempty-is-zet assms show cf-id (b ↓CF 𝔉) : b ↓CF 𝔉 ↦↦Cα b ↓CF 𝔉
    by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)
  from vempty-is-zet assms show (𝔅(|CId|)(|b|)) A↓CF 𝔉 : b ↓CF 𝔉 ↦↦Cα b ↓CF 𝔉
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
  from assms have ObjMap-dom-lhs:
    𝒟∘ ((𝔅(|CId|)(|b|)) A↓CF 𝔉(|ObjMap|)) = b ↓CF 𝔉(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros)
  from assms have ObjMap-dom-rhs:
    𝒟∘ (cf-id (b ↓CF 𝔉)(|ObjMap|)) = b ↓CF 𝔉(|Obj|)
```

```
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show (𝔅(‖CId‖)(‖b‖)) ₐ↓꜀ꜰ 𝔉(‖ObjMap‖) = cf-id (b ↓꜀ꜰ 𝔉)(‖ObjMap‖)
  proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
    fix A assume prems: A ∈∘ b ↓꜀ꜰ 𝔉(‖Obj‖)
    with assms obtain a' f'
      where A-def: A = [0, a', f']∘
        and a': a' ∈∘ 𝔄(‖Obj‖)
        and f': f' : b ↦𝔅 𝔉(‖ObjMap‖)(‖a'‖)
      by auto
    from prems assms vempty-is-zet a' f' show
      (𝔅(‖CId‖)(‖b‖)) ₐ↓꜀ꜰ 𝔉(‖ObjMap‖)(‖A‖) = cf-id (b ↓꜀ꜰ 𝔉)(‖ObjMap‖)(‖A‖)
      unfolding A-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps
            cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)+
  from assms have ArrMap-dom-lhs:
    𝒟∘ ((𝔅(‖CId‖)(‖b‖)) ₐ↓꜀ꜰ 𝔉(‖ArrMap‖)) = b ↓꜀ꜰ 𝔉(‖Arr‖)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros)
  from assms have ArrMap-dom-rhs:
    𝒟∘ (cf-id (b ↓꜀ꜰ 𝔉)(‖ArrMap‖)) = b ↓꜀ꜰ 𝔉(‖Arr‖)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show (𝔅(‖CId‖)(‖b‖)) ₐ↓꜀ꜰ 𝔉(‖ArrMap‖) = cf-id (b ↓꜀ꜰ 𝔉)(‖ArrMap‖)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix F assume prems: F ∈∘ b ↓꜀ꜰ 𝔉(‖Arr‖)
    then obtain A B where F: F : A ↦b ↓꜀ꜰ 𝔉 B by (auto dest: is-arrI)
    from assms F obtain b' f' b'' f'' h
      where F-def: F = [[0, b', f']∘, [0, b'', f'']∘, [0, h]∘]∘
        and A-def: A = [0, b', f']∘
        and B-def: B = [0, b'', f'']∘
        and h: h : b' ↦𝔄 b''
        and f': f' : b ↦𝔅 𝔉(‖ObjMap‖)(‖b'‖)
        and f'': f'' : b ↦𝔅 𝔉(‖ObjMap‖)(‖b''‖)
        and 𝔉(‖ArrMap‖)(‖h‖) ∘A𝔅 f' = f''
      by auto
    from assms prems F h f' f'' show
      (𝔅(‖CId‖)(‖b‖)) ₐ↓꜀ꜰ 𝔉(‖ArrMap‖)(‖F‖) = cf-id (b ↓꜀ꜰ 𝔉)(‖ArrMap‖)(‖F‖)
      unfolding F-def A-def B-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-comma-cs-simps cat-cs-simps cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-shallow cs-intro: cat-comma-cs-intros cat-cs-intros)+
qed simp-all

lemma (in is-functor) cf-cf-arr-comma-CId:
  assumes b ∈∘ 𝔅(‖Obj‖)
  shows 𝔉 ꜀ꜰ↓ₐ (𝔅(‖CId‖)(‖b‖)) = cf-id (𝔉 ꜀ꜰ↓ b)
proof(rule cf-eqI)
  from vempty-is-zet assms show cf-id (𝔉 ꜀ꜰ↓ b) : 𝔉 ꜀ꜰ↓ b ↦↦꜀α 𝔉 ꜀ꜰ↓ b
    by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)
  from vempty-is-zet assms show 𝔉 ꜀ꜰ↓ₐ (𝔅(‖CId‖)(‖b‖)) : 𝔉 ꜀ꜰ↓ b ↦↦꜀α 𝔉 ꜀ꜰ↓ b
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)
  from assms have ObjMap-dom-lhs:
```

```
    𝒟∘ (𝔉 _CF↓A (𝔅(|CId|)(|b|))(|ObjMap|)) = 𝔉 _CF↓ b(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros)
  from assms have ObjMap-dom-rhs:
    𝒟∘ (cf-id (𝔉 _CF↓ b)(|ObjMap|)) = 𝔉 _CF↓ b(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show 𝔉 _CF↓A (𝔅(|CId|)(|b|))(|ObjMap|) = cf-id (𝔉 _CF↓ b)(|ObjMap|)
  proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
    fix A assume prems: A ∈∘ 𝔉 _CF↓ b(|Obj|)
    with assms obtain a' f'
      where A-def: A = [a', 0, f']∘
        and a': a' ∈∘ 𝔄(|Obj|)
        and f': f' : 𝔉(|ObjMap|)(|a'|) ↦𝔅 b
      by auto
    from prems assms vempty-is-zet a' f' show
      𝔉 _CF↓A (𝔅(|CId|)(|b|))(|ObjMap|)(|A|) = cf-id (𝔉 _CF↓ b)(|ObjMap|)(|A|)
      unfolding A-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps cs-intro: cat-cs-intros
        )
  qed (cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros)+
  from assms have ArrMap-dom-lhs:
    𝒟∘ (𝔉 _CF↓A (𝔅(|CId|)(|b|))(|ArrMap|)) = 𝔉 _CF↓ b(|Arr|)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps cs-intro: cat-cs-intros)
  from assms have ArrMap-dom-rhs:
    𝒟∘ (cf-id (𝔉 ↓CF b)(|ArrMap|)) = 𝔉 ↓CF b(|Arr|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show 𝔉 _CF↓A (𝔅(|CId|)(|b|))(|ArrMap|) = cf-id (𝔉 _CF↓ b)(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix F assume prems: F ∈∘ 𝔉 _CF↓ b(|Arr|)
    then obtain A B where F: F : A ↦𝔉 _CF↓ b B by (auto dest: is-arrI)
    from assms F obtain a' f' a'' f'' k
      where F-def: F = [[a', 0, f']∘, [a'', 0, f'']∘, [k, 0]∘]∘
        and A-def: A = [a', 0, f']∘
        and B-def: B = [a'', 0, f'']∘
        and k: k : a' ↦𝔄 a''
        and f': f' : 𝔉(|ObjMap|)(|a'|) ↦𝔅 b
        and f'': f'' : 𝔉(|ObjMap|)(|a''|) ↦𝔅 b
        and [cat-cs-simps]: f'' ∘A𝔅 𝔉(|ArrMap|)(|k|) = f'
      by auto
    from assms prems F k f' f'' show
      𝔉 _CF↓A (𝔅(|CId|)(|b|))(|ArrMap|)(|F|) = cf-id (𝔉 _CF↓ b)(|ArrMap|)(|F|)
      unfolding F-def A-def B-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-comma-cs-simps cat-cs-simps cs-intro: cat-cs-intros
        )
  qed
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-comma-cs-intros cat-cs-intros
    )+
qed simp-all
```

14.8.5 Comma functors and projections

lemma (**in** *is-functor*)
cf-cf-comp-cf-obj-cf-comma-proj-cf-arr-cf-comma[*cat-comma-cs-simps*]:
assumes $f : a \mapsto_{\mathfrak{B}} b$
shows $a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F} = b \ {}_{O}\bigsqcap_{CF} \mathfrak{F}$
proof(*rule cf-eqI*)
from *assms vempty-is-zet* **show** $b \ {}_{O}\bigsqcap_{CF} \mathfrak{F} : b \downarrow_{CF} \mathfrak{F} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
from *assms* **show** $a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F} : b \downarrow_{CF} \mathfrak{F} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*)
from *assms* **have** *ObjMap-dom-lhs*:
$\mathcal{D}_{\circ} ((a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F})(\!|ObjMap|\!)) = b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*
)
from *assms* **have** *ObjMap-dom-rhs*: $\mathcal{D}_{\circ} (b \ {}_{O}\bigsqcap_{CF} \mathfrak{F}(\!|ObjMap|\!)) = b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
show $(a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F})(\!|ObjMap|\!) = b \ {}_{O}\bigsqcap_{CF} \mathfrak{F}(\!|ObjMap|\!)$
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs*)
from *assms* **show** *vsv* $(b \ {}_{O}\bigsqcap_{CF} \mathfrak{F}(\!|ObjMap|\!))$
by (*cs-concl* **cs-shallow cs-intro**: *cat-comma-cs-intros*)
fix A **assume** *prems*: $A \in_{\circ} b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
with *assms* **obtain** $b' f'$
where *A-def*: $A = [0, b', f']_{\circ}$
and b': $b' \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and f': $f' : b \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b'|\!)$
by *auto*
from *prems assms* $b' f'$ **show**
$(a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F})(\!|ObjMap|\!)(\!|A|\!) = b \ {}_{O}\bigsqcap_{CF} \mathfrak{F}(\!|ObjMap|\!)(\!|A|\!)$
unfolding *A-def*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-comma-cs-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros*
)
qed
(
use assms vempty-is-zet **in**
‹*cs-concl cs-intro*: *cat-cs-intros cat-comma-cs-intros*›
)
from *assms* **have** *ArrMap-dom-lhs*:
$\mathcal{D}_{\circ} ((a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F})(\!|ObjMap|\!)) = b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*
)
from *assms vempty-is-zet* **have** *ArrMap-dom-rhs*:
$\mathcal{D}_{\circ} (b \ {}_{O}\bigsqcap_{CF} \mathfrak{F}(\!|ObjMap|\!)) = b \downarrow_{CF} \mathfrak{F}(\!|Obj|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-comma-cs-simps*)
from *assms vempty-is-zet* **have** *ArrMap-dom-lhs*:
$\mathcal{D}_{\circ} ((a \ {}_{O}\bigsqcap_{CF} \mathfrak{F} \circ_{CF} f \ {}_{A}\downarrow_{CF} \mathfrak{F})(\!|ArrMap|\!)) = b \downarrow_{CF} \mathfrak{F}(\!|Arr|\!)$
by
(

```
      cs-concl
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-comma-cs-intros
    )
  from assms have ArrMap-dom-rhs: 𝒟∘ (b ₒ⨅ CF 𝔉(|ArrMap|)) = b ↓CF 𝔉(|Arr|)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps)
  show (a ₒ⨅ CF 𝔉 ∘CF f ₐ↓CF 𝔉)(|ArrMap|) = b ₒ⨅ CF 𝔉(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix F assume F ∈∘ b ↓CF 𝔉(|Arr|)
    then obtain A B where F: F : A ↦b ↓CF 𝔉 B
      by (auto dest: is-arrI)
    with assms obtain b′ f′ b″ f″ h
      where F-def: F = [[0, b′, f′]∘, [0, b″, f″]∘, [0, h]∘]∘
        and A-def: A = [0, b′, f′]∘
        and B-def: B = [0, b″, f″]∘
        and h: h : b′ ↦𝔄 b″
        and f′: f′ : b ↦𝔅 𝔉(|ObjMap|)(|b′|)
        and f″: f″ : b ↦𝔅 𝔉(|ObjMap|)(|b″|)
        and f″-def: 𝔉(|ArrMap|)(|h|) ∘A𝔅 f′ = f″
      by auto
    from vempty-is-zet h assms f′ f″ F show
      (a ₒ⨅ CF 𝔉 ∘CF f ₐ↓CF 𝔉)(|ArrMap|)(|F|) = b ₒ⨅ CF 𝔉(|ArrMap|)(|F|)
      unfolding F-def A-def B-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-comma-cs-simps f″-def[symmetric]
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )+
  qed
    (
      use assms vempty-is-zet in
        ‹cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros›
    )
qed simp-all

lemma (in is-functor)
  cf-cf-comp-cf-cf-obj-comma-proj-cf-cf-arr-comma[cat-comma-cs-simps]:
  assumes f : a ↦𝔅 b
  shows 𝔉 CF⨅ O b ∘CF 𝔉 CF↓A f = 𝔉 CF⨅ O a
proof(rule cf-eqI)
  from assms vempty-is-zet show 𝔉 CF⨅ O a : 𝔉 CF↓ a ↦↦Cα 𝔄
    by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)
  from assms show 𝔉 CF⨅ O b ∘CF 𝔉 CF↓A f : 𝔉 CF↓ a ↦↦Cα 𝔄
    by (cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros)
  from assms have ObjMap-dom-lhs:
    𝒟∘ ((𝔉 CF⨅ O b ∘CF 𝔉 CF↓A f)(|ObjMap|)) = 𝔉 CF↓ a(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  from assms have ObjMap-dom-rhs: 𝒟∘ (𝔉 CF⨅ O a(|ObjMap|)) = 𝔉 CF↓ a(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps)
  show (𝔉 CF⨅ O b ∘CF 𝔉 CF↓A f)(|ObjMap|) = 𝔉 CF⨅ O a(|ObjMap|)
  proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
    from assms show vsv (𝔉 CF⨅ O a(|ObjMap|))
      by (cs-concl cs-shallow cs-intro: cat-comma-cs-intros)
    fix A assume prems: A ∈∘ 𝔉 CF↓ a(|Obj|)
```

```
    with assms obtain a' f'
      where A-def: A = [a', 0, f']∘
        and b': a' ∈∘ 𝔄(|Obj|)
        and f': f' : ℱ(|ObjMap|)(|a'|) ↦𝔅 a
      by auto
    from prems assms b' f' show
      (ℱ CF⨅O b ∘CF ℱ CF↓A f)(|ObjMap|)(|A|) = ℱ CF⨅O a(|ObjMap|)(|A|)
      unfolding A-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )
  qed
    (
      use assms vempty-is-zet in
        ‹cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros›
    )
  from assms vempty-is-zet have ArrMap-dom-lhs:
    𝒟∘ ((ℱ CF⨅O b ∘CF ℱ CF↓A f)(|ArrMap|)) = ℱ CF↓ a(|Arr|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-comma-cs-intros
      )
  from assms have ArrMap-dom-rhs: 𝒟∘ (ℱ CF⨅O a(|ArrMap|)) = ℱ CF↓ a(|Arr|)
    by (cs-concl cs-shallow cs-simp: cat-comma-cs-simps)
  show (ℱ CF⨅O b ∘CF ℱ CF↓A f)(|ArrMap|) = ℱ CF⨅O a(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix F assume F ∈∘ ℱ CF↓ a(|Arr|)
    then obtain A B where F: F : A ↦ℱ CF↓ a B by (auto dest: is-arrI)
    with assms obtain a' f' a'' f'' k
      where F-def: F = [[a', 0, f']∘, [a'', 0, f'']∘, [k, 0]∘]∘
        and A-def: A = [a', 0, f']∘
        and B-def: B = [a'', 0, f'']∘
        and k: k : a' ↦𝔄 a''
        and f': f' : ℱ(|ObjMap|)(|a'|) ↦𝔅 a
        and f'': f'' : ℱ(|ObjMap|)(|a''|) ↦𝔅 a
        and f'-def: f'' ∘A𝔅 ℱ(|ArrMap|)(|k|) = f'
      by auto
    from vempty-is-zet k assms f' f'' F show
      (ℱ CF⨅O b ∘CF ℱ CF↓A f)(|ArrMap|)(|F|) = ℱ CF⨅O a(|ArrMap|)(|F|)
      unfolding F-def A-def B-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps f'-def
            cs-intro: cat-cs-intros cat-comma-cs-intros
        )+
  qed
    (
      use assms vempty-is-zet in
        ‹cs-concl cs-shallow cs-intro: cat-cs-intros cat-comma-cs-intros›
    )
qed simp-all
```

14.8.6 Opposite comma functors

lemma (**in** *is-functor*) *cf-op-cf-obj-comma-cf-arr-cf-comma*:
assumes $g : c \mapsto_{\mathfrak{B}} c'$
shows
$op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c' \circ_{CF} op\text{-}cf\ (\mathfrak{F}_{CF}{\downarrow_A}\ g) =$
$g_{A}{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F}) \circ_{CF} op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c$
proof(*rule cf-eqI*)
from *assms* **interpret** $\mathfrak{F}c$: *category* α $\langle \mathfrak{F}_{CF}{\downarrow}\ c \rangle$
by
(
cs-concl
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-comma-cs-intros*
)
from *assms* **have** *c*: $c \in_{\circ} \mathfrak{B}(\!|Obj|\!)$ **by** *auto*
from *assms* **show** $op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c' \circ_{CF} op\text{-}cf\ (\mathfrak{F}_{CF}{\downarrow_A}\ g) :$
$op\text{-}cat\ (\mathfrak{F}_{CF}{\downarrow}\ c) \mapsto\mapsto_{C\alpha} c'{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F})$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
then have *ObjMap-dom-lhs*:
$\mathcal{D}_{\circ}\ ((op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c' \circ_{CF} op\text{-}cf\ (\mathfrak{F}_{CF}{\downarrow_A}\ g))(\!|ObjMap|\!)) =$
$(op\text{-}cat\ (\mathfrak{F}_{CF}{\downarrow}\ c))(\!|Obj|\!)$
and *ArrMap-dom-lhs*:
$\mathcal{D}_{\circ}\ ((op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c' \circ_{CF} op\text{-}cf\ (\mathfrak{F}_{CF}{\downarrow_A}\ g))(\!|ArrMap|\!)) =$
$(op\text{-}cat\ (\mathfrak{F}_{CF}{\downarrow}\ c))(\!|Arr|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)+
from *assms* **show**
$g_{A}{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F}) \circ_{CF} op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c :$
$op\text{-}cat\ (\mathfrak{F}_{CF}{\downarrow}\ c) \mapsto\mapsto_{C\alpha} c'{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F})$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-comma-cs-intros cat-op-intros*
)
then have *ObjMap-dom-rhs*:
$\mathcal{D}_{\circ}\ ((g_{A}{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F}) \circ_{CF} op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c)(\!|ObjMap|\!)) =$
$(op\text{-}cat\ (\mathfrak{F}_{CF}{\downarrow}\ c))(\!|Obj|\!)$
and *ArrMap-dom-rhs*:
$\mathcal{D}_{\circ}\ ((g_{A}{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F}) \circ_{CF} op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c)(\!|ArrMap|\!)) =$
$(op\text{-}cat\ (\mathfrak{F}_{CF}{\downarrow}\ c))(\!|Arr|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)+
show
$(op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c' \circ_{CF} op\text{-}cf\ (\mathfrak{F}_{CF}{\downarrow_A}\ g))(\!|ObjMap|\!) =$
$(g_{A}{\downarrow_{CF}}\ (op\text{-}cf\ \mathfrak{F}) \circ_{CF} op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c)(\!|ObjMap|\!)$
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs cat-op-simps*)
fix *A* **assume** $A \in_{\circ} \mathfrak{F}_{CF}{\downarrow}\ c(\!|Obj|\!)$
with *assms* **obtain** *a f*
where *A-def*: $A = [a, 0, f]_{\circ}$
and *a*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and *f*: $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} c$
by *auto*
from *assms a f* **show**
$(op\text{-}cf\text{-}obj\text{-}comma\ \mathfrak{F}\ c' \circ_{CF} op\text{-}cf\ (\mathfrak{F}_{CF}{\downarrow_A}\ g))(\!|ObjMap|\!)(\!|A|\!) =$

```
      (g _{A↓CF} (op-cf 𝔉) ∘_{CF} op-cf-obj-comma 𝔉 c)(⦇ObjMap⦈)(⦇A⦈)
    unfolding A-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
      )
  qed
    (
      use assms in
        ‹cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros›
    )+
  show
    (op-cf-obj-comma 𝔉 c' ∘_{CF} op-cf (𝔉 _{CF↓A} g))(⦇ArrMap⦈) =
      (g _{A↓CF} (op-cf 𝔉) ∘_{CF} op-cf-obj-comma 𝔉 c)(⦇ArrMap⦈)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs cat-op-simps)
    fix F assume F ∈_∘ 𝔉 _{CF↓} c(⦇Arr⦈)
    then obtain A B where F: F : A ↦_{𝔉 _{CF↓} c} B by auto
    with assms c obtain a f a' f' h
      where F-def: F = [[a, 0, f]_∘, [a', 0, f']_∘, [h, 0]_∘]_∘
        and A-def: A = [a, 0, f]_∘
        and B-def: B = [a', 0, f']_∘
        and h: h : a ↦_𝔄 a'
        and f: f : 𝔉(⦇ObjMap⦈)(⦇a⦈) ↦_𝔅 c
        and f': f' : 𝔉(⦇ObjMap⦈)(⦇a'⦈) ↦_𝔅 c
        and [cat-comma-cs-simps]: f' ∘_{A𝔅} 𝔉(⦇ArrMap⦈)(⦇h⦈) = f
      by auto
    from F assms h f f' c show
      (op-cf-obj-comma 𝔉 c' ∘_{CF} op-cf (𝔉 _{CF↓A} g))(⦇ArrMap⦈)(⦇F⦈) =
        (g _{A↓CF} (op-cf 𝔉) ∘_{CF} op-cf-obj-comma 𝔉 c)(⦇ArrMap⦈)(⦇F⦈)
      unfolding F-def A-def B-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-comma-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros
        )
  qed
    (
      use assms in
        ‹cs-concl cs-intro: cat-cs-intros cat-comma-cs-intros cat-op-intros›
    )+
qed simp-all
```

15 *Rel*

15.1 Background

The methodology chosen for the exposition of *Rel* as a category is analogous to the one used in [8] for the exposition of *Rel* as a semicategory. The general references for this section are Chapter I-7 in [7] and nLab [1][8].

named-theorems *cat-Rel-cs-simps*
named-theorems *cat-Rel-cs-intros*

lemmas (**in** *arr-Rel*) [*cat-Rel-cs-simps*] =
dg-Rel-shared-cs-simps

lemmas (**in** *arr-Rel*) [*cat-cs-intros*, *cat-Rel-cs-intros*] =
arr-Rel-axioms'

lemmas [*cat-Rel-cs-simps*] =
dg-Rel-shared-cs-simps
arr-Rel.arr-Rel-length
arr-Rel-comp-Rel-id-Rel-left
arr-Rel-comp-Rel-id-Rel-right
arr-Rel.arr-Rel-converse-Rel-converse-Rel
arr-Rel-converse-Rel-eq-iff
arr-Rel-converse-Rel-comp-Rel
arr-Rel-comp-Rel-converse-Rel-left-if-v11
arr-Rel-comp-Rel-converse-Rel-right-if-v11

lemmas [*cat-Rel-cs-intros*] =
dg-Rel-shared-cs-intros
arr-Rel-comp-Rel
arr-Rel.arr-Rel-converse-Rel

lemmas [*cat-cs-simps*] = *incl-Rel-ArrVal-app*

15.2 *Rel* as a category

15.2.1 Definition and elementary properties

definition *cat-Rel* :: $V \Rightarrow V$
where *cat-Rel* α =
[
Vset α,
set $\{T.\ \textit{arr-Rel}\ \alpha\ T\}$,
$(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$,
$(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$,
$(\lambda ST \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Rel}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$,
VLambda (*Vset* α) *id-Rel*
]$_{\circ}$

Components.

lemma *cat-Rel-components*:
shows *cat-Rel* $\alpha(\!|\textit{Obj}|\!)$ = *Vset* α
and *cat-Rel* $\alpha(\!|\textit{Arr}|\!) = \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}$
and *cat-Rel* $\alpha(\!|\textit{Dom}|\!) = (\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$
and *cat-Rel* $\alpha(\!|\textit{Cod}|\!) = (\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$
and *cat-Rel* $\alpha(\!|\textit{Comp}|\!) = (\lambda ST \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Rel}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$

[8] https://ncatlab.org/nlab/show/Rel

and *cat-Rel* $\alpha(\!|CId|\!)$ = *VLambda* (*Vset* α) *id-Rel*
unfolding *cat-Rel-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-cat-Rel*: *cat-smc* (*cat-Rel* α) = *smc-Rel* α
proof(*rule vsv-eqI*)
show *vsv* (*cat-smc* (*cat-Rel* α)) **unfolding** *cat-smc-def* **by** *auto*
show *vsv* (*smc-Rel* α) **unfolding** *smc-Rel-def* **by** *auto*
have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-Rel* α)) = $5_{\mathbb{N}}$
unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-Rel* α) = $5_{\mathbb{N}}$
unfolding *smc-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*cat-smc* (*cat-Rel* α)) = $\mathcal{D}_\circ$ (*smc-Rel* α)
unfolding *dom-lhs dom-rhs* **by** *simp*
show
$a \in_\circ \mathcal{D}_\circ$ (*cat-smc* (*cat-Rel* α)) $\Longrightarrow$ *cat-smc* (*cat-Rel* α)$(\!|a|\!)$ = *smc-Rel* $\alpha(\!|a|\!)$
for *a*
by
(
unfold dom-lhs,
elim-in-numeral,
unfold cat-smc-def dg-field-simps cat-Rel-def smc-Rel-def
)
(*auto simp*: *nat-omega-simps*)
qed

lemmas-with [*folded cat-smc-cat-Rel*, *unfolded slicing-simps*]:
cat-Rel-Obj-iff = *smc-Rel-Obj-iff*
and *cat-Rel-Arr-iff*[*cat-Rel-cs-simps*] = *smc-Rel-Arr-iff*
and *cat-Rel-Dom-vsv*[*cat-Rel-cs-intros*] = *smc-Rel-Dom-vsv*
and *cat-Rel-Dom-vdomain*[*cat-Rel-cs-simps*] = *smc-Rel-Dom-vdomain*
and *cat-Rel-Dom-app*[*cat-Rel-cs-simps*] = *smc-Rel-Dom-app*
and *cat-Rel-Dom-vrange* = *smc-Rel-Dom-vrange*
and *cat-Rel-Cod-vsv*[*cat-Rel-cs-intros*] = *smc-Rel-Cod-vsv*
and *cat-Rel-Cod-vdomain*[*cat-Rel-cs-simps*] = *smc-Rel-Cod-vdomain*
and *cat-Rel-Cod-app*[*cat-Rel-cs-simps*] = *smc-Rel-Cod-app*
and *cat-Rel-Cod-vrange* = *smc-Rel-Cod-vrange*
and *cat-Rel-is-arrI*[*cat-Rel-cs-intros*] = *smc-Rel-is-arrI*
and *cat-Rel-is-arrD* = *smc-Rel-is-arrD*
and *cat-Rel-is-arrE* = *smc-Rel-is-arrE*
and *cat-Rel-is-arr-ArrValE* = *smc-Rel-is-arr-ArrValE*

lemmas-with [*folded cat-smc-cat-Rel*, *unfolded slicing-simps*, *unfolded cat-smc-cat-Rel*]:
cat-Rel-composable-arrs-dg-Rel = *smc-Rel-composable-arrs-dg-Rel*
and *cat-Rel-Comp* = *smc-Rel-Comp*
and *cat-Rel-Comp-app*[*cat-Rel-cs-simps*] = *smc-Rel-Comp-app*
and *cat-Rel-Comp-vdomain*[*simp*] = *smc-Rel-Comp-vdomain*
and *cat-Rel-is-monic-arrI* = *smc-Rel-is-monic-arrI*
and *cat-Rel-is-monic-arrD* = *smc-Rel-is-monic-arrD*
and *cat-Rel-is-monic-arr* = *smc-Rel-is-monic-arr*
and *cat-Rel-is-monic-arr-is-epic-arr* = *smc-Rel-is-monic-arr-is-epic-arr*
and *cat-Rel-is-epic-arr-is-monic-arr* = *smc-Rel-is-epic-arr-is-monic-arr*
and *cat-Rel-is-epic-arrI* = *smc-Rel-is-epic-arrI*
and *cat-Rel-is-epic-arrD* = *smc-Rel-is-epic-arrD*
and *cat-Rel-is-epic-arr* = *smc-Rel-is-epic-arr*

lemmas [*cat-cs-simps*] = *cat-Rel-is-arrD*(*2*,*3*)

lemmas [*cat-Rel-cs-intros*] = *cat-Rel-is-arrI*

lemmas-with (**in** $\mathcal{Z}$) [*folded cat-smc-cat-Rel*, *unfolded slicing-simps*]:
cat-Rel-Hom-vifunion-in-Vset = *smc-Rel-Hom-vifunion-in-Vset*
and *cat-Rel-incl-Rel-is-arr* = *smc-Rel-incl-Rel-is-arr*
and *cat-Rel-incl-Rel-is-arr'*[*cat-Rel-cs-intros*] = *smc-Rel-incl-Rel-is-arr'*
and *cat-Rel-Comp-vrange* = *smc-Rel-Comp-vrange*
and *cat-Rel-obj-terminal* = *smc-Rel-obj-terminal*
and *cat-Rel-obj-initial* = *smc-Rel-obj-initial*
and *cat-Rel-obj-terminal-obj-initial* = *smc-Rel-obj-terminal-obj-initial*
and *cat-Rel-obj-null* = *smc-Rel-obj-null*
and *cat-Rel-is-zero-arr* = *smc-Rel-is-zero-arr*

lemmas [*cat-Rel-cs-intros*] = $\mathcal{Z}$.*cat-Rel-incl-Rel-is-arr'*

15.2.2 Identity

lemma (**in** $\mathcal{Z}$) *cat-Rel-CId-app*[*cat-Rel-cs-simps*]:
assumes $T \in_{\circ} \mathit{Vset}\ \alpha$
shows *cat-Rel* $\alpha(\!|\mathit{CId}|\!)(\!|T|\!) = \mathit{id\text{-}Rel}\ T$
using *assms* **unfolding** *cat-Rel-components* **by** *simp*

lemmas [*cat-Rel-cs-simps*] = $\mathcal{Z}$.*cat-Rel-CId-app*

15.2.3 *Rel* is a category

lemma (**in** $\mathcal{Z}$) *category-cat-Rel*: *category* α (*cat-Rel* α)
proof(*rule categoryI*, *unfold cat-smc-cat-Rel*)

interpret *Rel*: *semicategory* α ‹*cat-smc* (*cat-Rel* α)›
unfolding *cat-smc-cat-Rel* **by** (*simp add*: *semicategory-smc-Rel*)

show *vfsequence* (*cat-Rel* α) **unfolding** *cat-Rel-def* **by** *simp*
show *vcard* (*cat-Rel* α) = $6_{\mathbb{N}}$
unfolding *cat-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show *cat-Rel* $\alpha(\!|\mathit{CId}|\!)(\!|A|\!) : A \mapsto_{\mathit{cat\text{-}Rel}\ \alpha} A$
if $A \in_{\circ}$ *cat-Rel* $\alpha(\!|\mathit{Obj}|\!)$ **for** A
using *that*
unfolding *cat-Rel-Obj-iff*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Rel-cs-simps* **cs-intro**: *cat-Rel-cs-intros arr-Rel-id-RelI*
)
show *cat-Rel* $\alpha(\!|\mathit{CId}|\!)(\!|B|\!) \circ_{A\,\mathit{cat\text{-}Rel}\ \alpha} F = F$
if $F : A \mapsto_{\mathit{cat\text{-}Rel}\ \alpha} B$ **for** $F\ A\ B$
proof–
from *that* **have** *arr-Rel* $\alpha\ F\ A \in_{\circ} \mathit{Vset}\ \alpha\ B \in_{\circ} \mathit{Vset}\ \alpha$
by (*auto elim*: *cat-Rel-is-arrE simp*: *cat-Rel-cs-simps*)
with *that* **show** *?thesis*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-Rel-cs-simps*
cs-intro: *cat-Rel-cs-intros arr-Rel-id-RelI*
)
qed

```
  show F ∘_A cat-Rel α cat-Rel α(|CId|)(|B|) = F
    if F : B ↦_cat-Rel α C for F B C
  proof–
    from that have arr-Rel α F B ∈∘ Vset α C ∈∘ Vset α
      by (auto elim: cat-Rel-is-arrE simp: cat-Rel-cs-simps)
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-Rel-cs-simps
            cs-intro: cat-Rel-cs-intros arr-Rel-id-RelI
        )
  qed

qed (auto simp: semicategory-smc-Rel cat-Rel-components)

lemma (in Z) category-cat-Rel'[cat-Rel-cs-intros]:
  assumes α' = α and α'' = α
  shows category α' (cat-Rel α'')
  unfolding assms by (rule category-cat-Rel)

lemmas [cat-Rel-cs-intros] = Z.category-cat-Rel'
```

15.3 Canonical dagger for *Rel*

15.3.1 Definition and elementary properties

```
definition cf-dag-Rel :: V ⇒ V (‹†_C.Rel›)
  where †_C.Rel α =
    [
      vid-on (cat-Rel α(|Obj|)),
      VLambda (cat-Rel α(|Arr|)) converse-Rel,
      op-cat (cat-Rel α),
      cat-Rel α
    ]∘
```

Components.

```
lemma cf-dag-Rel-components:
  shows †_C.Rel α(|ObjMap|) = vid-on (cat-Rel α(|Obj|))
    and †_C.Rel α(|ArrMap|) = VLambda (cat-Rel α(|Arr|)) converse-Rel
    and †_C.Rel α(|HomDom|) = op-cat (cat-Rel α)
    and †_C.Rel α(|HomCod|) = cat-Rel α
  unfolding cf-dag-Rel-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

Slicing.

```
lemma cf-smcf-cf-dag-Rel: cf-smcf (†_C.Rel α) = †_SMC.Rel α
proof(rule vsv-eqI)
  have dom-lhs: 𝒟∘ (cf-smcf (†_C.Rel α)) = 4ℕ
    unfolding cf-smcf-def by (simp add: nat-omega-simps)
  have dom-rhs: 𝒟∘ (†_SMC.Rel α) = 4ℕ
    unfolding smcf-dag-Rel-def by (simp add: nat-omega-simps)
  show 𝒟∘ (cf-smcf (†_C.Rel α)) = 𝒟∘ (†_SMC.Rel α)
    unfolding dom-lhs dom-rhs by simp
  show A ∈∘ 𝒟∘ (cf-smcf (†_C.Rel α)) ⟹ cf-smcf (†_C.Rel α)(|A|) = †_SMC.Rel α(|A|)
    for A
    by
      (
        unfold dom-lhs,
```

elim-in-numeral,
unfold dghm-field-simps[*symmetric*],
unfold
cat-smc-cat-Rel
slicing-commute[*symmetric*]
cf-smcf-components
smcf-dag-Rel-components
cf-dag-Rel-components
smc-Rel-components
cat-Rel-components
)
simp-all
qed (*auto simp*: *cf-smcf-def smcf-dag-Rel-def*)

lemmas-with [*folded cat-smc-cat-Rel cf-smcf-cf-dag-Rel*, *unfolded slicing-simps*]:
cf-dag-Rel-ObjMap-vsv[*cat-Rel-cs-intros*] = *smcf-dag-Rel-ObjMap-vsv*
and *cf-dag-Rel-ObjMap-vdomain*[*cat-Rel-cs-simps*] = *smcf-dag-Rel-ObjMap-vdomain*
and *cf-dag-Rel-ObjMap-app*[*cat-Rel-cs-simps*] = *smcf-dag-Rel-ObjMap-app*
and *cf-dag-Rel-ObjMap-vrange*[*cat-Rel-cs-simps*] = *smcf-dag-Rel-ObjMap-vrange*
and *cf-dag-Rel-ArrMap-vsv*[*cat-Rel-cs-intros*] = *smcf-dag-Rel-ArrMap-vsv*
and *cf-dag-Rel-ArrMap-vdomain*[*cat-Rel-cs-simps*] = *smcf-dag-Rel-ArrMap-vdomain*
and *cf-dag-Rel-ArrMap-app*[*cat-Rel-cs-simps*] = *smcf-dag-Rel-ArrMap-app*
and *cf-dag-Rel-ArrMap-vrange*[*cat-Rel-cs-simps*] = *smcf-dag-Rel-ArrMap-vrange*
and *cf-dag-Rel-app-is-arr*[*cat-Rel-cs-intros*] = *smcf-dag-Rel-app-is-arr*
and *cf-dag-Rel-ArrMap-app-vdomain*[*cat-cs-simps*] =
smcf-dag-Rel-ArrMap-app-vdomain
and *cf-dag-Rel-ArrMap-app-vrange*[*cat-cs-simps*] =
smcf-dag-Rel-ArrMap-app-vrange
and *cf-dag-Rel-ArrMap-app-iff*[*cat-cs-simps*] = *smcf-dag-Rel-ArrMap-app-iff*
and *cf-dag-Rel-ArrMap-smc-Rel-Comp*[*cat-Rel-cs-simps*] =
smcf-dag-Rel-ArrMap-smc-Rel-Comp

15.3.2 Canonical dagger is a contravariant isomorphism of *Rel*

lemma (**in** $\mathcal{Z}$) *cf-dag-Rel-is-iso-functor*:
$\dagger_{C.Rel}\ \alpha : op\text{-}cat\ (cat\text{-}Rel\ \alpha) \mapsto\!\!\mapsto_{C.iso\alpha} cat\text{-}Rel\ \alpha$
proof
(
rule is-iso-functorI,
unfold
cat-smc-cat-Rel
cf-smcf-cf-dag-Rel
cat-Rel-components
cat-op-simps
slicing-commute[*symmetric*]
)

interpret *is-iso-semifunctor* α ‹*op-smc* (*smc-Rel* α)› ‹*smc-Rel* α› ‹$\dagger_{SMC.Rel}\ \alpha$›
by (*rule smcf-dag-Rel-is-iso-semifunctor*)
interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*rule category-cat-Rel*)

show $\dagger_{C.Rel}\ \alpha : op\text{-}cat\ (cat\text{-}Rel\ \alpha) \mapsto\!\!\mapsto_{C\alpha} cat\text{-}Rel\ \alpha$
proof
(
rule is-functorI,
unfold
cat-smc-cat-Rel
cf-smcf-cf-dag-Rel

```
      cat-op-simps
      slicing-commute[symmetric]
      cf-dag-Rel-components(3,4)
  )
  show vfsequence (†C.Rel α)
    unfolding cf-dag-Rel-def by (simp add: nat-omega-simps)
  show vcard (†C.Rel α) = 4ℕ
    unfolding cf-dag-Rel-def by (simp add: nat-omega-simps)
  show †C.Rel α(|ArrMap|)(|cat-Rel α(|CId|)(|C|)|) = cat-Rel α(|CId|)(|†C.Rel α(|ObjMap|)(|C|)|)
    if C ∈∘ cat-Rel α(|Obj|) for C
  proof–
    from that have C ∈∘ Vset α
      by (auto elim: cat-Rel-is-arrE simp: cat-Rel-Obj-iff)
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-Rel-cs-simps cs-intro: cat-cs-intros arr-Rel-id-RelI
        )
  qed
qed (auto simp: cat-cs-intros intro: smc-cs-intros)

show †SMC.Rel α : op-smc (smc-Rel α) ↦↦SMC.isoα smc-Rel α
  by (rule smcf-dag-Rel-is-iso-semifunctor)

qed

lemma (in Z) cf-dag-Rel-is-iso-functor'[cat-cs-intros]:
  assumes 𝔄' = op-cat (cat-Rel α)
    and 𝔅' = cat-Rel α
    and α' = α
  shows †C.Rel α : 𝔄' ↦↦C.isoα' 𝔅'
  unfolding assms by (rule cf-dag-Rel-is-iso-functor)

lemmas [cat-cs-intros] = Z.cf-dag-Rel-is-iso-functor'
```

15.3.3 Further properties of the canonical dagger

```
lemma (in Z) cf-cn-comp-cf-dag-Rel-cf-dag-Rel:
  †C.Rel α CF∘ †C.Rel α = cf-id (cat-Rel α)
proof(rule cf-smcf-eqI)
  interpret category α ‹cat-Rel α› by (rule category-cat-Rel)
  from cf-dag-Rel-is-iso-functor have dag:
    †C.Rel α : op-cat (cat-Rel α) ↦↦Cα cat-Rel α
    by (simp add: is-iso-functor.axioms(1))
  from cf-cn-comp-is-functorI[OF category-axioms dag dag] show
    †C.Rel α CF∘ †C.Rel α : cat-Rel α ↦↦Cα cat-Rel α .
  show cf-id (cat-Rel α) : cat-Rel α ↦↦Cα cat-Rel α
    by (auto simp: category.cat-cf-id-is-functor category-axioms)
  show cf-smcf (†C.Rel α CF∘ †C.Rel α) = cf-smcf (smcf-id (cat-Rel α))
    unfolding slicing-commute[symmetric] cat-smc-cat-Rel cf-smcf-cf-dag-Rel
    by (simp add: smcf-cn-comp-smcf-dag-Rel-smcf-dag-Rel)
qed simp-all
```

15.4 Isomorphism

```
context
begin
```

private lemma *cat-Rel-is-iso-arr-right-vsubset*:
  assumes $S : B \mapsto_{\textit{cat-Rel}\ \alpha} A$
    and $T : A \mapsto_{\textit{cat-Rel}\ \alpha} B$
    and $S \circ_{A\,\textit{cat-Rel}\ \alpha} T = \textit{cat-Rel}\ \alpha(\!|\textit{CId}|\!)(\!|A|\!)$
    and $T \circ_{A\,\textit{cat-Rel}\ \alpha} S = \textit{cat-Rel}\ \alpha(\!|\textit{CId}|\!)(\!|B|\!)$
  shows $S(\!|\textit{ArrVal}|\!) \subseteq_\circ (T(\!|\textit{ArrVal}|\!))^{-1}{}_\circ$
proof(*rule vsubset-antisym vsubsetI*)

  interpret S: *arr-Rel* α S
    rewrites $S(\!|\textit{ArrDom}|\!) = B$ **and** $S(\!|\textit{ArrCod}|\!) = A$
    by (*intro cat-Rel-is-arrD*[*OF assms*(*1*)])+
  interpret T: *arr-Rel* α T
    rewrites $T(\!|\textit{ArrDom}|\!) = A$ **and** $T(\!|\textit{ArrCod}|\!) = B$
    by (*intro cat-Rel-is-arrD*[*OF assms*(*2*)])+

  interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*simp add*: *S.category-cat-Rel*)

  interpret *dag*: *is-iso-functor* α ‹*op-cat* (*cat-Rel* α)› ‹*cat-Rel* α› ‹$\dagger_{C.Rel}$ α›
    by (*auto simp*: *S.cf-dag-Rel-is-iso-functor*)

  from *assms*(*2*) **have** *A*: $A \in_\circ \textit{cat-Rel}\ \alpha(\!|\textit{Obj}|\!)$ **by** *auto*
  from *assms*(*3*) **have** $(S \circ_{A\,\textit{cat-Rel}\ \alpha} T)(\!|\textit{ArrVal}|\!) = \textit{cat-Rel}\ \alpha(\!|\textit{CId}|\!)(\!|A|\!)(\!|\textit{ArrVal}|\!)$
    by *simp*
  with *A* **have** [*simp*]: $S(\!|\textit{ArrVal}|\!) \circ_\circ T(\!|\textit{ArrVal}|\!) = \textit{vid-on}\ A$
    unfolding *cat-Rel-Comp-app*[*OF assms*(*1*,*2*)]
    by (*simp add*: *id-Rel-components comp-Rel-components cat-Rel-components*)

  from *assms*(*2*) **have** *B*: $B \in_\circ \textit{cat-Rel}\ \alpha(\!|\textit{Obj}|\!)$ **by** *auto*
  from *assms*(*4*) **have** $(T \circ_{A\,\textit{cat-Rel}\ \alpha} S)(\!|\textit{ArrVal}|\!) = \textit{cat-Rel}\ \alpha(\!|\textit{CId}|\!)(\!|B|\!)(\!|\textit{ArrVal}|\!)$
    by *simp*
  with *B* **have** [*simp*]: $T(\!|\textit{ArrVal}|\!) \circ_\circ S(\!|\textit{ArrVal}|\!) = \textit{vid-on}\ B$
    unfolding *cat-Rel-Comp-app*[*OF assms*(*2*,*1*)]
    by (*simp add*: *id-Rel-components comp-Rel-components cat-Rel-components*)

  fix *ab* **assume** *ab*: $ab \in_\circ S(\!|\textit{ArrVal}|\!)$
  with *S.vbrelation* **obtain** *a b* **where** *ab-def*: $ab = \langle a,\ b\rangle$ **and** $a \in_\circ B$
    by (*metis S.arr-Rel-ArrVal-vdomain S.ArrVal.vbrelation-vinE vsubsetE*)
  then have $\langle a,\ a\rangle \in_\circ T(\!|\textit{ArrVal}|\!) \circ_\circ S(\!|\textit{ArrVal}|\!)$ **by** *auto*
  then obtain *c* **where** $\langle a,\ c\rangle \in_\circ S(\!|\textit{ArrVal}|\!)$ **and** *ca*[*intro*]: $\langle c,\ a\rangle \in_\circ T(\!|\textit{ArrVal}|\!)$
    by *blast*
  have $\langle b,\ a\rangle \in_\circ T(\!|\textit{ArrVal}|\!)$
  proof(*rule ccontr*)
    assume $\langle b,\ a\rangle \notin_\circ T(\!|\textit{ArrVal}|\!)$
    with *ca* **have** $c \neq b$ **by** *clarsimp*
    moreover from *ab* **have** $\langle c,\ b\rangle \in_\circ S(\!|\textit{ArrVal}|\!) \circ_\circ T(\!|\textit{ArrVal}|\!)$
      unfolding *ab-def* **by** *blast*
    ultimately show *False* **by** (*simp add*: *vid-on-iff*)
  qed
  then show $ab \in_\circ (T(\!|\textit{ArrVal}|\!))^{-1}{}_\circ$ **unfolding** *ab-def* **by** *clarsimp*

qed

private lemma *cat-Rel-is-iso-arr-left-vsubset*:
  assumes $S : B \mapsto_{\textit{cat-Rel}\ \alpha} A$
    and $T : A \mapsto_{\textit{cat-Rel}\ \alpha} B$
    and $S \circ_{A\,\textit{cat-Rel}\ \alpha} T = \textit{cat-Rel}\ \alpha(\!|\textit{CId}|\!)(\!|A|\!)$
    and $T \circ_{A\,\textit{cat-Rel}\ \alpha} S = \textit{cat-Rel}\ \alpha(\!|\textit{CId}|\!)(\!|B|\!)$

shows $(T(\!|ArrVal|\!))^{-1}_{\circ} \subseteq_{\circ} S(\!|ArrVal|\!)$
using *assms*(*2*,*3*,*4*) *cat-Rel-is-iso-arr-right-vsubset*[*OF assms*(*2*,*1*)]
by *auto*

private lemma *is-iso-arr-dag*:
assumes $S : B \mapsto_{cat\text{-}Rel\ \alpha} A$
and $T : A \mapsto_{cat\text{-}Rel\ \alpha} B$
and $S \circ_{A\, cat\text{-}Rel\ \alpha} T = cat\text{-}Rel\ \alpha(\!|CId|\!)(\!|A|\!)$
and $T \circ_{A\, cat\text{-}Rel\ \alpha} S = cat\text{-}Rel\ \alpha(\!|CId|\!)(\!|B|\!)$
shows $S = \dagger_{C.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)$
proof(*rule arr-Rel-eqI*[*of* α])

interpret *S*: *arr-Rel* α *S* **by** (*intro cat-Rel-is-arrD*(*1*)[*OF assms*(*1*)])
interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*rule S.category-cat-Rel*)
interpret *dag*: *is-iso-functor* α ‹*op-cat* (*cat-Rel* α)› ‹*cat-Rel* α› ‹$\dagger_{C.Rel}\ \alpha$›
by (*auto simp*: *S.cf-dag-Rel-is-iso-functor*)

from *assms*(*1*) **show** *S*: *arr-Rel* α *S* **by** (*fastforce simp*: *cat-Rel-components*(*2*))
from *cf-dag-Rel-app-is-arr*[*OF assms*(*2*)] **show** *arr-Rel* α ($\dagger_{C.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)$)
by (*auto elim*!: *cat-Rel-is-arrE*)
from *assms*(*2*) **have** *T*: *arr-Rel* α *T* **by** (*auto simp*: *cat-Rel-is-arrD*(*1*))
from *S T assms* **show** $S(\!|ArrVal|\!) = \dagger_{C.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)(\!|ArrVal|\!)$
unfolding *cf-dag-Rel-ArrMap-app*[*OF T*] *converse-Rel-components*
by (*intro vsubset-antisym*)
(
simp-all add:
cat-Rel-is-iso-arr-left-vsubset
cat-Rel-is-iso-arr-right-vsubset
)
from *T assms* **show** $S(\!|ArrDom|\!) = \dagger_{C.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)(\!|ArrDom|\!)$
unfolding *cf-dag-Rel-components*
by (*auto simp*: *cat-cs-simps cat-Rel-cs-simps converse-Rel-components*(*1*))
from *S assms* **show** $S(\!|ArrCod|\!) = \dagger_{C.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)(\!|ArrCod|\!)$
by
(
cs-concl
cs-intro: *cat-op-intros cat-cs-intros*
cs-simp: *cat-Rel-cs-simps cat-cs-simps*
)

qed

lemma *cat-Rel-is-iso-arrI*[*intro*]:
assumes $T : A \mapsto_{cat\text{-}Rel\ \alpha} B$
and *v11* $(T(\!|ArrVal|\!))$
and $\mathcal{D}_{\circ}\ (T(\!|ArrVal|\!)) = A$
and $\mathcal{R}_{\circ}\ (T(\!|ArrVal|\!)) = B$
shows $T : A \mapsto_{iso\, cat\text{-}Rel\ \alpha} B$
proof(*rule is-iso-arrI*[**where** *?g* = ‹$\dagger_{C.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)$›])

interpret *T*: *arr-Rel* α *T* **by** (*intro cat-Rel-is-arrD*[*OF assms*(*1*)])+
interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*rule T.category-cat-Rel*)
interpret *v11*: *v11* ‹$T(\!|ArrVal|\!)$› **by** (*rule assms*(*2*))

interpret *is-iso-functor* α ‹*op-cat* (*cat-Rel* α)› ‹*cat-Rel* α› ‹$\dagger_{C.Rel}\ \alpha$›
by (*simp add*: *T.cf-dag-Rel-is-iso-functor*)

show $T : A \mapsto_{cat\text{-}Rel\ \alpha} B$ **by** (*rule assms*(*1*))

```
  show is-inverse (cat-Rel α) (†C.Rel α(|ArrMap|)(|T|)) T
  proof(intro is-inverseI)
    from assms(1) show dag-T: †C.Rel α(|ArrMap|)(|T|) : B ↦cat-Rel α A
      by
        (
          cs-concl
            cs-simp: cat-op-simps cat-Rel-cs-simps
            cs-intro: cat-cs-intros
        )
    show T: T : A ↦cat-Rel α B by (rule assms(1))
    from T T.arr-Rel-axioms v11.v11-axioms assms(3) show
      †C.Rel α(|ArrMap|)(|T|) ∘A cat-Rel α T = cat-Rel α(|CId|)(|A|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-Rel-cs-simps
            cs-intro: cat-cs-intros cat-Rel-cs-intros
        )
    from T T.arr-Rel-axioms v11.v11-axioms assms(4) show
      T ∘A cat-Rel α †C.Rel α(|ArrMap|)(|T|) = cat-Rel α(|CId|)(|B|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-Rel-cs-simps
            cs-intro: cat-cs-intros cat-Rel-cs-intros
        )
  qed

qed

lemma cat-Rel-is-iso-arrD[dest]:
  assumes T : A ↦iso cat-Rel α B
  shows T : A ↦cat-Rel α B
    and v11 (T(|ArrVal|))
    and 𝒟∘ (T(|ArrVal|)) = A
    and ℛ∘ (T(|ArrVal|)) = B
proof–

  from assms show T: T : A ↦cat-Rel α B
    by (simp add: is-iso-arr-def)

  interpret T: arr-Rel α T
    rewrites [simp]: T(|ArrDom|) = A and [simp]: T(|ArrCod|) = B
    by (intro cat-Rel-is-arrD[OF T])+
  interpret is-iso-functor α ‹op-cat (cat-Rel α)› ‹cat-Rel α› ‹†C.Rel α›
    by (simp add: T.cf-dag-Rel-is-iso-functor)

  from is-iso-arrD[OF assms(1)] obtain S where
    is-inverse (cat-Rel α) S T
    by clarsimp
  from is-inverseD[OF this] obtain A′ B′ where S : B′ ↦cat-Rel α A′
    and T : A′ ↦cat-Rel α B′
    and S ∘A cat-Rel α T = cat-Rel α(|CId|)(|A′|)
    and T ∘A cat-Rel α S = cat-Rel α(|CId|)(|B′|)
    by auto
  moreover with T have A′ = A B′ = B by auto
  ultimately have S: S : B ↦cat-Rel α A
```

```
  and ST: S ∘_A cat-Rel α T = cat-Rel α(|CId|)(|A|)
  and TS: T ∘_A cat-Rel α S = cat-Rel α(|CId|)(|B|)
  by auto

from S T ST TS have S-def: S = †_C.Rel α(|ArrMap|)(|T|)
  by (rule is-iso-arr-dag)

interpret S: arr-Rel α ‹†_C.Rel α(|ArrMap|)(|T|)›
  rewrites (†_C.Rel α(|ArrMap|)(|T|))(|ArrDom|) = B
    and (†_C.Rel α(|ArrMap|)(|T|))(|ArrCod|) = A
  by (fold S-def, insert S, all‹elim cat-Rel-is-arrE›)
    (simp-all add: cat-Rel-components)

from T.arr-Rel-axioms S-def have S-T: S(|ArrVal|) = (T(|ArrVal|))⁻¹_∘
  by (simp add: cf-dag-Rel-ArrMap-app converse-Rel-components(1))

from S have A: A ∈_∘ cat-Rel α(|Obj|) and B: B ∈_∘ cat-Rel α(|Obj|) by auto

from B TS A ST have
  (T ∘_Rel S)(|ArrVal|) = id-Rel B(|ArrVal|)
  (S ∘_Rel T)(|ArrVal|) = id-Rel A(|ArrVal|)
  unfolding cat-Rel-Comp-app[OF S T] cat-Rel-Comp-app[OF T S]
  unfolding cat-Rel-components
  by simp-all

then have val-ST: S(|ArrVal|) ∘_∘ T(|ArrVal|) = vid-on A
  and val-TS: T(|ArrVal|) ∘_∘ S(|ArrVal|) = vid-on B
  unfolding comp-Rel-components id-Rel-components by simp-all

show v11 (T(|ArrVal|))
proof(rule v11I)

  show vsv (T(|ArrVal|))
  proof(rule vsvI)
    fix a b c assume prems: ⟨a, b⟩ ∈_∘ T(|ArrVal|) ⟨a, c⟩ ∈_∘ T(|ArrVal|)
    from prems(1) S-T have ⟨b, a⟩ ∈_∘ S(|ArrVal|) by auto
    with prems(2) val-TS have ⟨b, c⟩ ∈_∘ vid-on B by auto
    then show b = c by clarsimp
  qed (auto simp: T.ArrVal.vbrelation-axioms)

  show vsv ((T(|ArrVal|))⁻¹_∘)
  proof(rule vsvI)
    fix a b c
    assume prems: ⟨a, b⟩ ∈_∘ (T(|ArrVal|))⁻¹_∘ ⟨a, c⟩ ∈_∘ (T(|ArrVal|))⁻¹_∘
    with S-T have ⟨a, b⟩ ∈_∘ S(|ArrVal|) and ⟨a, c⟩ ∈_∘ S(|ArrVal|) by auto
    moreover from prems have ⟨b, a⟩ ∈_∘ T(|ArrVal|) and ⟨c, a⟩ ∈_∘ T(|ArrVal|)
      by auto
    ultimately have ⟨b, c⟩ ∈_∘ vid-on A using val-ST by auto
    then show b = c by clarsimp
  qed auto

qed

show D_∘ (T(|ArrVal|)) = A
proof(intro vsubset-antisym vsubsetI)
  fix a assume a ∈_∘ A
  with val-ST have ⟨a, a⟩ ∈_∘ S(|ArrVal|) ∘_∘ T(|ArrVal|) by auto
  then show a ∈_∘ D_∘ (T(|ArrVal|)) by auto
```

qed (*use T.arr-Rel-ArrVal-vdomain* **in** *auto*)

show $\mathcal{R}_{\circ}\ (T(\!|ArrVal|\!)) = B$
proof(*intro vsubset-antisym vsubsetI*)
fix b **assume** $b \in_{\circ} B$
with *val-TS* **have** $\langle b, b\rangle \in_{\circ} T(\!|ArrVal|\!) \circ_{\circ} S(\!|ArrVal|\!)$ **by** *auto*
then show $b \in_{\circ} \mathcal{R}_{\circ}\ (T(\!|ArrVal|\!))$ **by** *auto*
qed (*use T.arr-Rel-ArrVal-vrange* **in** *auto*)

qed

end

lemmas [*cat-Rel-cs-simps*] = *cat-Rel-is-iso-arrD*(*3*,*4*)

lemma *cat-Rel-is-iso-arr*:
$T : A \mapsto_{iso} \textit{cat-Rel}\ \alpha\ B \longleftrightarrow$
$T : A \mapsto_{\textit{cat-Rel}\ \alpha} B \wedge$
v11 $(T(\!|ArrVal|\!)) \wedge$
$\mathcal{D}_{\circ}\ (T(\!|ArrVal|\!)) = A \wedge$
$\mathcal{R}_{\circ}\ (T(\!|ArrVal|\!)) = B$
by *auto*

15.5 The inverse arrow

lemma *cat-Rel-the-inverse*[*cat-Rel-cs-simps*]:
assumes $T : A \mapsto_{iso} \textit{cat-Rel}\ \alpha\ B$
shows ${T^{-1}}_{C\,\textit{cat-Rel}\ \alpha} = {T^{-1}}_{Rel}$
unfolding *the-inverse-def*
proof(*rule the-equality*)
from *assms* **have** *T*: $T : A \mapsto_{\textit{cat-Rel}\ \alpha} B$ **by** *auto*
interpret *T*: *arr-Rel* α T **by** (*intro cat-Rel-is-arrD*[*OF T*])+
interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*rule T.category-cat-Rel*)
from *assms T T.arr-Rel-axioms cat-Rel-is-iso-arrD*(*2*)[*OF assms*]
show *inv-T-T*: *is-inverse* (*cat-Rel* α) $({T^{-1}}_{Rel})$ T
by (*intro is-inverseI*[**where** *a*=*A* **and** *b*=*B*])
(
cs-concl
cs-simp: *cat-cs-simps cat-Rel-cs-simps*
cs-intro: *cat-Rel-cs-intros cat-cs-intros*
)+
fix S **assume** *is-inverse* (*cat-Rel* α) S T
then show $S = {T^{-1}}_{Rel}$
by (*rule category.cat-is-inverse-eq*[*OF Rel.category-axioms - inv-T-T*])
qed

16 *Par*

16.1 Background

The methodology chosen for the exposition of *Par* as a category is analogous to the one used in [8] for the exposition of *Par* as a semicategory.

named-theorems *cat-Par-cs-simps*
named-theorems *cat-Par-cs-intros*

lemmas (**in** *arr-Par*) [*cat-Par-cs-simps*] =
dg-Rel-shared-cs-simps

lemmas (**in** *arr-Par*) [*cat-cs-intros*, *cat-Par-cs-intros*] =
arr-Par-axioms'

lemmas [*cat-Par-cs-simps*] =
dg-Rel-shared-cs-simps
arr-Par.arr-Par-length
arr-Par-comp-Par-id-Par-left
arr-Par-comp-Par-id-Par-right

lemmas [*cat-Par-cs-intros*] =
arr-Par-comp-Par

16.2 *Par* as a category

16.2.1 Definition and elementary properties

definition *cat-Par* :: $V \Rightarrow V$
where *cat-Par* α =
[
Vset α,
set $\{T.\ \textit{arr-Par}\ \alpha\ T\}$,
$(\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Par}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$,
$(\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Par}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$,
$(\lambda ST \in_\circ \textit{composable-arrs}\ (\textit{dg-Par}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$,
VLambda (*Vset* α) *id-Par*
]$_\circ$

Components.

lemma *cat-Par-components*:
shows *cat-Par* $\alpha(\!|\textit{Obj}|\!)$ = *Vset* α
and *cat-Par* $\alpha(\!|\textit{Arr}|\!)$ = *set* $\{T.\ \textit{arr-Par}\ \alpha\ T\}$
and *cat-Par* $\alpha(\!|\textit{Dom}|\!) = (\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Par}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$
and *cat-Par* $\alpha(\!|\textit{Cod}|\!) = (\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Par}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$
and *cat-Par* $\alpha(\!|\textit{Comp}|\!) = (\lambda ST \in_\circ \textit{composable-arrs}\ (\textit{dg-Par}\ \alpha).\ ST(\!|0|\!) \circ_{Par} ST(\!|1_{\mathbb{N}}|\!))$
and *cat-Par* $\alpha(\!|\textit{CId}|\!)$ = *VLambda* (*Vset* α) *id-Par*
unfolding *cat-Par-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-cat-Par*: *cat-smc* (*cat-Par* α) = *smc-Par* α
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-Par* α)) = $5_{\mathbb{N}}$
unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-Par* α) = $5_{\mathbb{N}}$
unfolding *smc-Par-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*cat-smc* (*cat-Par* α)) = $\mathcal{D}_\circ$ (*smc-Par* α)
unfolding *dom-lhs dom-rhs* **by** *simp*

show $a \in_{\circ} \mathcal{D}_{\circ}\ (\mathit{cat\text{-}smc}\ (\mathit{cat\text{-}Par}\ \alpha)) \Longrightarrow \mathit{cat\text{-}smc}\ (\mathit{cat\text{-}Par}\ \alpha)(\!|a|\!) = \mathit{smc\text{-}Par}\ \alpha(\!|a|\!)$
for a
by
(
unfold dom-lhs,
elim-in-numeral,
unfold cat-smc-def dg-field-simps cat-Par-def smc-Par-def
)
(*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *cat-smc-def smc-Par-def*)

lemmas-with [*folded cat-smc-cat-Par*, *unfolded slicing-simps*]:
cat-Par-Obj-iff = *smc-Par-Obj-iff*
and *cat-Par-Arr-iff*[*cat-Par-cs-simps*] = *smc-Par-Arr-iff*
and *cat-Par-Dom-vsv*[*cat-Par-cs-intros*] = *smc-Par-Dom-vsv*
and *cat-Par-Dom-vdomain*[*cat-Par-cs-simps*] = *smc-Par-Dom-vdomain*
and *cat-Par-Dom-vrange* = *smc-Par-Dom-vrange*
and *cat-Par-Dom-app*[*cat-Par-cs-simps*] = *smc-Par-Dom-app*
and *cat-Par-Cod-vsv*[*cat-Par-cs-intros*] = *smc-Par-Cod-vsv*
and *cat-Par-Cod-vdomain*[*cat-Par-cs-simps*] = *smc-Par-Cod-vdomain*
and *cat-Par-Cod-vrange* = *smc-Par-Cod-vrange*
and *cat-Par-Cod-app*[*cat-Par-cs-simps*] = *smc-Par-Cod-app*
and *cat-Par-is-arrI* = *smc-Par-is-arrI*
and *cat-Par-is-arrD* = *smc-Par-is-arrD*
and *cat-Par-is-arrE* = *smc-Par-is-arrE*

lemmas-with [*folded cat-smc-cat-Par*, *unfolded slicing-simps*]:
cat-Par-composable-arrs-dg-Par = *smc-Par-composable-arrs-dg-Par*
and *cat-Par-Comp* = *smc-Par-Comp*
and *cat-Par-Comp-app*[*cat-Par-cs-simps*] = *smc-Par-Comp-app*
and *cat-Par-Comp-vdomain*[*cat-Par-cs-simps*] = *smc-Par-Comp-vdomain*
and *cat-Par-is-monic-arrI* = *smc-Par-is-monic-arrI*
and *cat-Par-is-monic-arrD* = *smc-Par-is-monic-arrD*
and *cat-Par-is-monic-arr* = *smc-Par-is-monic-arr*
and *cat-Par-is-epic-arrI* = *smc-Par-is-epic-arrI*
and *cat-Par-is-epic-arrD* = *smc-Par-is-epic-arrD*
and *cat-Par-is-epic-arr* = *smc-Par-is-epic-arr*

lemmas [*cat-cs-simps*] = *cat-Par-is-arrD*(*2*,*3*)

lemmas [*cat-Par-cs-intros*] = *cat-Par-is-arrI*

lemmas-with (**in** $\mathcal{Z}$) [*folded cat-smc-cat-Par*, *unfolded slicing-simps*]:
cat-Par-Hom-vifunion-in-Vset = *smc-Par-Hom-vifunion-in-Vset*
and *cat-Par-incl-Par-is-arr* = *smc-Par-incl-Par-is-arr*
and *cat-Par-incl-Par-is-arr'*[*cat-Par-cs-intros*] = *smc-Par-incl-Par-is-arr'*
and *cat-Par-Comp-vrange* = *smc-Par-Comp-vrange*
and *cat-Par-obj-terminal* = *smc-Par-obj-terminal*
and *cat-Par-obj-initial* = *smc-Par-obj-initial*
and *cat-Par-obj-terminal-obj-initial* = *smc-Par-obj-terminal-obj-initial*
and *cat-Par-obj-null* = *smc-Par-obj-null*
and *cat-Par-is-zero-arr* = *smc-Par-is-zero-arr*

lemmas [*cat-Par-cs-intros*] = $\mathcal{Z}$.*cat-Par-incl-Par-is-arr'*

16.2.2 Identity

lemma *cat-Par-CId-app*[*cat-Par-cs-simps*]:

assumes $A \in_{\circ} Vset\ \alpha$
shows $cat\text{-}Par\ \alpha(\!|CId|\!)(\!|A|\!) = id\text{-}Par\ A$
using *assms* **unfolding** *cat-Par-components* **by** *simp*

lemma *id-Par-CId-app-app*[*cat-cs-simps*]:
assumes $A \in_{\circ} Vset\ \alpha$ **and** $a \in_{\circ} A$
shows $cat\text{-}Par\ \alpha(\!|CId|\!)(\!|A|\!)(\!|ArrVal|\!)(\!|a|\!) = a$
unfolding *cat-Par-CId-app*[*OF assms*(*1*)] *id-Rel-ArrVal-app*[*OF assms*(*2*)] **by** *simp*

16.2.3 *Par* is a category

lemma (**in** $\mathcal{Z}$) *category-cat-Par*: *category* α (*cat-Par* α)
proof(*intro categoryI*, *unfold cat-smc-cat-Rel cat-smc-cat-Par cat-op-simps*)

interpret *Par*: *semicategory* α ‹*cat-smc* (*cat-Par* α)›
unfolding *cat-smc-cat-Par* **by** (*simp add*: *semicategory-smc-Par*)

show *vfsequence* (*cat-Par* α) **unfolding** *cat-Par-def* **by** *simp*
show *vcard* (*cat-Par* α) = $6_{\mathbb{N}}$
unfolding *cat-Par-def* **by** (*simp add*: *nat-omega-simps*)
show $cat\text{-}Par\ \alpha(\!|CId|\!)(\!|A|\!) : A \mapsto_{cat\text{-}Par\ \alpha} A$ **if** $A \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$ **for** A
using *that*
unfolding *cat-Par-Obj-iff*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Par-cs-simps* **cs-intro**: *cat-Par-cs-intros arr-Par-id-ParI*
)

show $cat\text{-}Par\ \alpha(\!|CId|\!)(\!|B|\!) \circ_{A\ cat\text{-}Par\ \alpha} F = F$
if $F : A \mapsto_{cat\text{-}Par\ \alpha} B$ **for** $F\ A\ B$
proof–
from *that* **have** $arr\text{-}Par\ \alpha\ F\ B \in_{\circ} Vset\ \alpha$
by (*auto elim*: *cat-Par-is-arrE simp*: *cat-Par-cs-simps*)
with *that* $\mathcal{Z}$*-axioms* **show** *?thesis*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-Par-cs-simps*
cs-intro: *cat-Par-cs-intros arr-Par-id-ParI*
)
qed

show $F \circ_{A\ cat\text{-}Par\ \alpha} cat\text{-}Par\ \alpha(\!|CId|\!)(\!|B|\!) = F$
if $F : B \mapsto_{cat\text{-}Par\ \alpha} C$ **for** $F\ B\ C$
proof–
from *that* **have** $arr\text{-}Par\ \alpha\ F\ B \in_{\circ} Vset\ \alpha$
by (*auto elim*: *cat-Par-is-arrE simp*: *cat-Par-cs-simps*)
with *that* **show** *?thesis*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-Par-cs-simps*
cs-intro: *cat-Par-cs-intros arr-Par-id-ParI*
)
qed

qed (*auto simp*: *semicategory-smc-Par cat-Par-components*)

16.2.4 *Par* is a wide replete subcategory of *Rel*

lemma (in $\mathcal{Z}$**)** *wide-replete-subcategory-cat-Par-cat-Rel*:
cat-Par $\alpha \subseteq_{C.wr\alpha}$ *cat-Rel* α
proof(*intro wide-replete-subcategoryI*)
show *wide-subcategory-cat-Par-cat-Rel*: *cat-Par* $\alpha \subseteq_{C.wide\alpha}$ *cat-Rel* α
proof(*intro wide-subcategoryI*, *unfold cat-smc-cat-Rel cat-smc-cat-Par*)
interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*rule category-cat-Rel*)
interpret *Par*: *category* α ‹*cat-Par* α› **by** (*rule category-cat-Par*)
interpret *wide-subsemicategory* α ‹*smc-Par* α› ‹*smc-Rel* α›
by (*simp add*: *wide-subsemicategory-smc-Par-smc-Rel*)
show *cat-Par* $\alpha \subseteq_{C\alpha}$ *cat-Rel* α
proof(*intro subcategoryI*, *unfold cat-smc-cat-Rel cat-smc-cat-Par*)
show *smc-Par* $\alpha \subseteq_{SMC\alpha}$ *smc-Rel* α **by** (*simp add*: *subsemicategory-axioms*)
fix *A* **assume** $A \in_\circ$ *cat-Par* $\alpha(\!|Obj|\!)$
then show *cat-Par* $\alpha(\!|CId|\!)(\!|A|\!)$ = *cat-Rel* $\alpha(\!|CId|\!)(\!|A|\!)$
unfolding *cat-Par-components cat-Rel-components* **by** *simp*
qed
(
auto simp:
subsemicategory-axioms Rel.category-axioms Par.category-axioms
)
qed (*rule wide-subsemicategory-smc-Par-smc-Rel*)
show *cat-Par* $\alpha \subseteq_{C.rep\alpha}$ *cat-Rel* α
proof(*intro replete-subcategoryI*)
interpret *wide-subcategory* α ‹*cat-Par* α› ‹*cat-Rel* α›
by (*rule wide-subcategory-cat-Par-cat-Rel*)
show *cat-Par* $\alpha \subseteq_{C\alpha}$ *cat-Rel* α **by** (*rule subcategory-axioms*)
fix *A B F* **assume** *prems*: $A \in_\circ$ *cat-Par* $\alpha(\!|Obj|\!)$ $F : A \mapsto_{iso cat\text{-}Rel\ \alpha} B$
note *arr-Rel* = *cat-Rel-is-iso-arrD*[*OF prems*(*2*)]
from *arr-Rel*(*2*) **show** $F : A \mapsto_{cat\text{-}Par\ \alpha} B$
by (*intro cat-Par-is-arrI arr-Par-arr-RelI cat-Rel-is-arrD*[*OF arr-Rel*(*1*)])
auto
qed
qed

16.3 Isomorphism

lemma *cat-Par-is-iso-arrI*[*intro*]:
assumes $T : A \mapsto_{cat\text{-}Par\ \alpha} B$
and *v11* ($T(\!|ArrVal|\!)$)
and $\mathcal{D}_\circ$ ($T(\!|ArrVal|\!)$) = *A*
and $\mathcal{R}_\circ$ ($T(\!|ArrVal|\!)$) = *B*
shows $T : A \mapsto_{iso cat\text{-}Par\ \alpha} B$
proof–
interpret *T*: *arr-Par* α *T* **by** (*intro cat-Par-is-arrD*(*1*)[*OF assms*(*1*)])
note [*cat-cs-intros*] = *cat-Rel-is-iso-arrI*
from *T.wide-replete-subcategory-cat-Par-cat-Rel assms* **have**
$T : A \mapsto_{iso cat\text{-}Rel\ \alpha} B$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-sub-cs-intros cat-sub-fw-cs-intros*)
with *T.wide-replete-subcategory-cat-Par-cat-Rel assms* **show**
$T : A \mapsto_{iso cat\text{-}Par\ \alpha} B$
by (*cs-concl* **cs-shallow cs-simp**: *cat-sub-bw-cs-simps*)
qed

lemma *cat-Par-is-iso-arrD*[*dest*]:
assumes $T : A \mapsto_{iso cat\text{-}Par\ \alpha} B$
shows $T : A \mapsto_{cat\text{-}Par\ \alpha} B$
and *v11* ($T(\!|ArrVal|\!)$)

```
    and 𝒟∘ (T(|ArrVal|)) = A
    and ℛ∘ (T(|ArrVal|)) = B
proof–
  interpret T: arr-Par α T
    by (intro cat-Par-is-arrD(1)[OF is-iso-arrD(1)[OF assms(1)]])
  from T.wide-replete-subcategory-cat-Par-cat-Rel assms have T:
    T : A ↦iso cat-Rel α B
    by (cs-concl cs-shallow cs-intro: cat-sub-cs-intros cat-sub-fw-cs-intros)
  show v11 (T(|ArrVal|)) 𝒟∘ (T(|ArrVal|)) = A ℛ∘ (T(|ArrVal|)) = B
    by (intro cat-Rel-is-iso-arrD[OF T])+
qed (rule is-iso-arrD(1)[OF assms])

lemma cat-Par-is-iso-arr:
  T : A ↦iso cat-Par α B ⟷
    T : A ↦cat-Par α B ∧
    v11 (T(|ArrVal|)) ∧
    𝒟∘ (T(|ArrVal|)) = A ∧
    ℛ∘ (T(|ArrVal|)) = B
  by auto
```

16.4 The inverse arrow

```
abbreviation (input) converse-Par :: V ⇒ V (‹(-⁻¹Par)› [1000] 999)
  where a⁻¹Par ≡ a⁻¹Rel

lemma cat-Par-the-inverse[cat-Par-cs-simps]:
  assumes T : A ↦iso cat-Par α B
  shows T⁻¹C cat-Par α = T⁻¹Par
proof–
  interpret T: arr-Par α T
    by (intro cat-Par-is-arrD(1)[OF is-iso-arrD(1)[OF assms(1)]])
  from T.wide-replete-subcategory-cat-Par-cat-Rel assms have T:
    T : A ↦iso cat-Rel α B
    by (cs-concl cs-shallow cs-intro: cat-sub-cs-intros cat-sub-fw-cs-intros)
  from T.wide-replete-subcategory-cat-Par-cat-Rel assms
  have [symmetric, cat-cs-simps]: T⁻¹C cat-Rel α = T⁻¹C cat-Par α
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-sub-bw-cs-simps cs-intro: cat-sub-cs-intros
      )
  from T show T⁻¹C cat-Par α = T⁻¹Par
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Rel-cs-simps cat-cs-simps cs-intro: cat-cs-intros
      )
qed
```

17 *Set*

17.1 Background

The methodology chosen for the exposition of *Set* as a category is analogous to the one used in [8] for the exposition of *Set* as a semicategory.

named-theorems *cat-Set-cs-simps*
named-theorems *cat-Set-cs-intros*

lemmas (**in** *arr-Set*) [*cat-Set-cs-simps*] =
dg-Rel-shared-cs-simps

lemmas (**in** *arr-Set*) [*cat-cs-intros*, *cat-Set-cs-intros*] =
arr-Set-axioms'

lemmas [*cat-Set-cs-simps*] =
dg-Rel-shared-cs-simps
arr-Set.arr-Set-ArrVal-vdomain
arr-Set-comp-Set-id-Set-left
arr-Set-comp-Set-id-Set-right

lemmas [*cat-Set-cs-intros*] =
dg-Rel-shared-cs-intros
arr-Set-comp-Set

named-theorems *cat-rel-par-Set-cs-intros*
named-theorems *cat-rel-par-Set-cs-simps*
named-theorems *cat-rel-Par-set-cs-intros*
named-theorems *cat-rel-Par-set-cs-simps*
named-theorems *cat-Rel-par-set-cs-intros*
named-theorems *cat-Rel-par-set-cs-simps*

17.2 *Set* as a category

17.2.1 Definition and elementary properties

definition *cat-Set* :: $V \Rightarrow V$
where *cat-Set* α =
[
Vset α,
set $\{T.\ \textit{arr-Set}\ \alpha\ T\}$,
$(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$,
$(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$,
$(\lambda ST \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Set}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$,
VLambda (*Vset* α) *id-Set*
]$_{\circ}$

Components.

lemma *cat-Set-components*:
shows *cat-Set* $\alpha(\!|\textit{Obj}|\!)$ = *Vset* α
and *cat-Set* $\alpha(\!|\textit{Arr}|\!)$ = *set* $\{T.\ \textit{arr-Set}\ \alpha\ T\}$
and *cat-Set* $\alpha(\!|\textit{Dom}|\!) = (\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$
and *cat-Set* $\alpha(\!|\textit{Cod}|\!) = (\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$
and *cat-Set* $\alpha(\!|\textit{Comp}|\!)$ =
$(\lambda ST \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Set}\ \alpha).\ ST(\!|0|\!) \circ_{Par} ST(\!|1_{\mathbb{N}}|\!))$
and *cat-Set* $\alpha(\!|\textit{CId}|\!)$ = *VLambda* (*Vset* α) *id-Set*
unfolding *cat-Set-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-cat-Set*: *cat-smc* (*cat-Set* α) = *smc-Set* α
proof(*rule vsv-eqI*)
  have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-Set* α)) = $5_{\mathbb{N}}$
    unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-Set* α) = $5_{\mathbb{N}}$
    unfolding *smc-Set-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*cat-smc* (*cat-Set* α)) = $\mathcal{D}_\circ$ (*smc-Set* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show $a \in_\circ \mathcal{D}_\circ$ (*cat-smc* (*cat-Set* α)) $\Longrightarrow$ *cat-smc* (*cat-Set* α)(|a|) = *smc-Set* α(|a|)
    for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold cat-smc-def dg-field-simps cat-Set-def smc-Set-def
)
      (*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *cat-smc-def smc-Set-def*)

lemmas-with [*folded cat-smc-cat-Set*, *unfolded slicing-simps*]:
  cat-Set-Obj-iff = *smc-Set-Obj-iff*
  and *cat-Set-Arr-iff*[*cat-Set-cs-simps*] = *smc-Set-Arr-iff*
  and *cat-Set-Dom-vsv*[*intro*] = *smc-Set-Dom-vsv*
  and *cat-Set-Dom-vdomain*[*simp*] = *smc-Set-Dom-vdomain*
  and *cat-Set-Dom-vrange* = *smc-Set-Dom-vrange*
  and *cat-Set-Dom-app* = *smc-Set-Dom-app*
  and *cat-Set-Cod-vsv*[*intro*] = *smc-Set-Cod-vsv*
  and *cat-Set-Cod-vdomain*[*simp*] = *smc-Set-Cod-vdomain*
  and *cat-Set-Cod-vrange* = *smc-Set-Cod-vrange*
  and *cat-Set-Cod-app*[*cat-Set-cs-simps*] = *smc-Set-Cod-app*
  and *cat-Set-is-arrI* = *smc-Set-is-arrI*
  and *cat-Set-is-arrD* = *smc-Set-is-arrD*
  and *cat-Set-is-arrE* = *smc-Set-is-arrE*
  and *cat-Set-ArrVal-vdomain*[*cat-cs-simps*] = *smc-Set-ArrVal-vdomain*
  and *cat-Set-ArrVal-app-vrange*[*cat-Set-cs-intros*] = *smc-Set-ArrVal-app-vrange*

lemmas [*cat-cs-simps*] = *cat-Set-is-arrD*(*2*,*3*)

lemmas [*cat-Set-cs-intros*] =
  cat-Set-is-arrI

lemmas-with [*folded cat-smc-cat-Set*, *unfolded slicing-simps*]:
  cat-Set-composable-arrs-dg-Set = *smc-Set-composable-arrs-dg-Set*
  and *cat-Set-Comp* = *smc-Set-Comp*
  and *cat-Set-Comp-app*[*cat-Set-cs-simps*] = *smc-Set-Comp-app*
  and *cat-Set-Comp-vdomain*[*cat-Set-cs-simps*] = *smc-Set-Comp-vdomain*
  and *cat-Set-is-monic-arrI* = *smc-Set-is-monic-arrI*
  and *cat-Set-is-monic-arrD* = *smc-Set-is-monic-arrD*
  and *cat-Set-is-monic-arr* = *smc-Set-is-monic-arr*
  and *cat-Set-is-epic-arrI* = *smc-Set-is-epic-arrI*
  and *cat-Set-is-epic-arrD* = *smc-Set-is-epic-arrD*
  and *cat-Set-is-epic-arr* = *smc-Set-is-epic-arr*

lemmas-with (**in** $\mathcal{Z}$) [*folded cat-smc-cat-Set*, *unfolded slicing-simps*]:
  cat-Set-Hom-vifunion-in-Vset = *smc-Set-Hom-vifunion-in-Vset*
  and *cat-Set-incl-Set-is-arr* = *smc-Set-incl-Set-is-arr*
  and *cat-Set-Comp-ArrVal* = *smc-Set-Comp-ArrVal*

and *cat-Set-Comp-vrange* = *smc-Set-Comp-vrange*
and *cat-Set-obj-terminal* = *smc-Set-obj-terminal*
and *cat-Set-obj-initial* = *smc-Set-obj-initial*
and *cat-Set-obj-null* = *smc-Set-obj-null*
and *cat-Set-is-zero-arr* = *smc-Set-is-zero-arr*

lemmas [*cat-cs-simps*] =
Z.*cat-Set-Comp-ArrVal*

lemma (**in** *Z*) *cat-Set-incl-Set-is-arr'*[*cat-cs-intros*, *cat-Set-cs-intros*]:
assumes $A \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
and $B \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
and $A \subseteq_{\circ} B$
and $A' = A$
and $B' = B$
and $\mathfrak{C}' =$ *cat-Set* α
shows *incl-Set* $A\ B : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule cat-Set-incl-Set-is-arr*)

lemmas [*cat-Set-cs-intros*] = *Z*.*cat-Set-incl-Set-is-arr'*

17.2.2 Identity

lemma *cat-Set-CId-app*[*cat-Set-cs-simps*]:
assumes $A \in_{\circ}$ *Vset* α
shows *cat-Set* $\alpha(\!|CId|\!)(\!|A|\!) =$ *id-Set* A
using *assms* **unfolding** *cat-Set-components* **by** *simp*

lemma *cat-Set-CId-app-app*[*cat-cs-simps*]:
assumes $A \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **and** $a \in_{\circ} A$
shows *cat-Set* $\alpha(\!|CId|\!)(\!|A|\!)(\!|ArrVal|\!)(\!|a|\!) = a$
unfolding
cat-Set-CId-app[*OF assms*(*1*)[*unfolded cat-Set-components*(*1*)]]
id-Rel-ArrVal-app[*OF assms*(*2*)]
by *simp*

17.2.3 *Set* is a category

lemma (**in** *Z*) *category-cat-Set*: *category* α (*cat-Set* α)
proof(*rule categoryI*, *unfold cat-smc-cat-Par cat-smc-cat-Set*)

interpret *Set*: *semicategory* α ‹*cat-smc* (*cat-Set* α)›
unfolding *cat-smc-cat-Set* **by** (*simp add*: *semicategory-smc-Set*)

show *vfsequence* (*cat-Set* α) **unfolding** *cat-Set-def* **by** *simp*
show *vcard* (*cat-Set* α) = $6_{\mathbb{N}}$
unfolding *cat-Set-def* **by** (*simp add*: *nat-omega-simps*)
show *semicategory* α (*smc-Set* α) **by** (*simp add*: *semicategory-smc-Set*)
show *cat-Set* $\alpha(\!|CId|\!)(\!|A|\!) : A \mapsto_{cat\text{-}Set\ \alpha} A$
if $A \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$ **for** A
using *that*
unfolding *cat-Set-Obj-iff*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Set-cs-simps* **cs-intro**: *cat-Set-cs-intros arr-Set-id-SetI*
)

```
  show cat-Set α(|CId|)(|B|) ∘_A cat-Set α F = F
    if F : A ↦_cat-Set α B for F A B
  proof–
    from that have arr-Set α F B ∈∘ Vset α by (auto elim: cat-Set-is-arrE)
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-Set-cs-simps
            cs-intro: cat-Set-cs-intros arr-Set-id-SetI
        )
  qed

  show F ∘_A cat-Set α cat-Set α(|CId|)(|B|) = F
    if F : B ↦_cat-Set α C for F B C
  proof–
    from that have arr-Set α F B ∈∘ Vset α by (auto elim: cat-Set-is-arrE)
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-Set-cs-simps
            cs-intro: cat-Set-cs-intros arr-Set-id-SetI
        )
  qed

qed (auto simp: cat-Set-components)

lemma (in Z) category-cat-Set':
  assumes β = α
  shows category β (cat-Set α)
  unfolding assms by (rule category-cat-Set)

lemmas [cat-cs-intros] = Z.category-cat-Set'
```

17.2.4 *Set* is a wide replete subcategory of *Par*

```
lemma (in Z) wide-replete-subcategory-cat-Set-cat-Par:
  cat-Set α ⊆_{C.wr α} cat-Par α
proof(intro wide-replete-subcategoryI)
  show wide-subcategory-cat-Set-cat-Par: cat-Set α ⊆_{C.wide α} cat-Par α
  proof(intro wide-subcategoryI, unfold cat-smc-cat-Par cat-smc-cat-Set)
    interpret Par: category α ‹cat-Par α› by (rule category-cat-Par)
    interpret Set: category α ‹cat-Set α› by (rule category-cat-Set)
    interpret wide-subsemicategory α ‹smc-Set α› ‹smc-Par α›
      by (simp add: wide-subsemicategory-smc-Set-smc-Par)
    show cat-Set α ⊆_C α cat-Par α
    proof(intro subcategoryI, unfold cat-smc-cat-Par cat-smc-cat-Set)
      show smc-Set α ⊆_{SMC α} smc-Par α by (simp add: subsemicategory-axioms)
      fix A assume A ∈∘ cat-Set α(|Obj|)
      then show cat-Set α(|CId|)(|A|) = cat-Par α(|CId|)(|A|)
        unfolding cat-Set-components cat-Par-components by simp
    qed
      (
        auto simp:
          subsemicategory-axioms Par.category-axioms Set.category-axioms
      )
  qed (rule wide-subsemicategory-smc-Set-smc-Par)
```

show *cat-Set* α $\subseteq_{C.rep\alpha}$ *cat-Par* α
proof(*intro replete-subcategoryI*)
interpret *wide-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-subcategory-cat-Set-cat-Par*)
show *cat-Set* α $\subseteq_{C\alpha}$ *cat-Par* α **by** (*rule subcategory-axioms*)
fix *A B F* **assume** $F : A \mapsto_{iso cat\text{-}Par\ \alpha} B$
note *arr-Par* = *cat-Par-is-iso-arrD*[*OF this*]
from *arr-Par* **show** $F : A \mapsto_{cat\text{-}Set\ \alpha} B$
by (*intro cat-Set-is-arrI arr-Set-arr-ParI cat-Par-is-arrD*[*OF arr-Par*(*1*)])
(*auto simp*: *cat-Par-is-arrD*(*2*))
qed
qed

17.2.5 *Set* is a subcategory of *Set*

lemma (in $\mathcal{Z}$**)** *subcategory-cat-Set-cat-Set*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *cat-Set* α $\subseteq_{C\beta}$ *cat-Set* β
proof–
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro subcategoryI'*)
show *category* β (*cat-Set* α)
by (*rule category.cat-category-if-ge-Limit*, *insert assms*(*2*))
(*cs-concl* **cs-intro**: *cat-cs-intros cat-Rel-cs-intros*)+
show $A \in_\circ$ *cat-Set* $\beta(\!|Obj|\!)$ **if** $A \in_\circ$ *cat-Set* $\alpha(\!|Obj|\!)$ **for** *A*
using *that*
unfolding *cat-Set-components*(*1*)
by (*meson assms*(*2*) *Vset-in-mono* β.*Axiom-of-Extensionality*(*3*))
show *is-arr-if-is-arr*:
$F : A \mapsto_{cat\text{-}Set\ \beta} B$ **if** $F : A \mapsto_{cat\text{-}Set\ \alpha} B$ **for** *A B F*
proof–
note *f* = *cat-Set-is-arrD*[*OF that*]
interpret *f*: *arr-Set* α *F* **by** (*rule f*(*1*))
show *?thesis*
proof(*intro cat-Set-is-arrI arr-SetI*)
show $\mathcal{R}_\circ$ (*F*(|*ArrVal*|)) $\subseteq_\circ$ *F*(|*ArrCod*|)
by (*auto simp*: *f.arr-Set-ArrVal-vrange*)
show *F*(|*ArrDom*|) $\in_\circ$ *Vset* β
by (*auto intro*!: *f.arr-Set-ArrDom-in-Vset Vset-in-mono assms*(*2*))
show *F*(|*ArrCod*|) $\in_\circ$ *Vset* β
by (*auto intro*!: *f.arr-Set-ArrCod-in-Vset Vset-in-mono assms*(*2*))
qed
(
auto simp:
f f.arr-Set-ArrVal-vdomain f.vfsequence-axioms f.arr-Set-length
)
qed
show $G \circ_{A\, cat\text{-}Set\ \alpha} F = G \circ_{A\, cat\text{-}Set\ \beta} F$
if $G : B \mapsto_{cat\text{-}Set\ \alpha} C$ **and** $F : A \mapsto_{cat\text{-}Set\ \alpha} B$ **for** *B C G A F*
proof–
note *g* = *cat-Set-is-arrD*[*OF that*(*1*)] **and** *f* = *cat-Set-is-arrD*[*OF that*(*2*)]
from *that* **have** α*-gf-is-arr*: $G \circ_{A\, cat\text{-}Set\ \alpha} F : A \mapsto_{cat\text{-}Set\ \beta} C$
by (*cs-concl* **cs-intro**: *cat-cs-intros is-arr-if-is-arr*)
from *that* **have** β*-gf-is-arr*: $G \circ_{A\, cat\text{-}Set\ \beta} F : A \mapsto_{cat\text{-}Set\ \beta} C$
by (*cs-concl* **cs-intro**: *cat-cs-intros is-arr-if-is-arr*)
note α*-gf* = *cat-Set-is-arrD*[*OF* α*-gf-is-arr*]
and β*-gf* = *cat-Set-is-arrD*[*OF* β*-gf-is-arr*]

```
      show ?thesis
      proof(rule arr-Set-eqI)
        show arr-Set β (G ∘_A cat-Set α F) by (rule α-gf(1))
        then interpret arr-Set-α-gf: arr-Set β ‹(G ∘_A cat-Set α F)› by simp
        from α-gf-is-arr have dom-lhs: 𝒟∘ ((G ∘_A cat-Set α F)⦇ArrVal⦈) = A
          by (cs-concl cs-shallow cs-simp: cat-cs-simps)
        show arr-Set β (G ∘_A cat-Set β F) by (rule β-gf(1))
        then interpret arr-Set-β-gf: arr-Set β ‹(G ∘_A cat-Set β F)› by simp
        from β-gf-is-arr have dom-rhs: 𝒟∘ ((G ∘_A cat-Set β F)⦇ArrVal⦈) = A
          by (cs-concl cs-shallow cs-simp: cat-cs-simps)
        show (G ∘_A cat-Set α F)⦇ArrVal⦈ = (G ∘_A cat-Set β F)⦇ArrVal⦈
        proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
          fix a assume a ∈∘ A
          from that this show
            (G ∘_A cat-Set α F)⦇ArrVal⦈⦇a⦈ = (G ∘_A cat-Set β F)⦇ArrVal⦈⦇a⦈
            by
              (
                cs-concl cs-shallow
                  cs-simp: cat-cs-simps cs-intro: cat-cs-intros is-arr-if-is-arr
              )
        qed auto
      qed
        (
          use α-gf-is-arr β-gf-is-arr in
            ‹cs-concl cs-shallow cs-simp: cat-cs-simps›
        )+
    qed
  qed
    (
      auto simp:
        assms(2) cat-Set-components Vset-trans Vset-in-mono cat-cs-intros
    )
qed
```

17.2.6 Further properties

```
lemma cat-Set-Comp-ArrVal-vrange:
  assumes S : B ↦_cat-Set α C and T : A ↦_cat-Set α B
  shows ℛ∘ ((S ∘_A cat-Set α T)⦇ArrVal⦈) ⊆∘ ℛ∘ (S⦇ArrVal⦈)
proof(intro vsubsetI)
  note SD = cat-Set-is-arrD[OF assms(1)]
  interpret S: arr-Set α S
    rewrites S⦇ArrDom⦈ = B and S⦇ArrCod⦈ = C
    by (intro SD)+
  from assms(1,2) have S ∘_A cat-Set α T : A ↦_cat-Set α C
    by (cs-concl cs-intro: cat-cs-intros)
  note ST = cat-Set-is-arrD[OF this]
  interpret ST: arr-Set α ‹S ∘_A cat-Set α T›
    rewrites (S ∘_A cat-Set α T)⦇ArrDom⦈ = A
      and (S ∘_A cat-Set α T)⦇ArrCod⦈ = C
    by (intro ST)+
  fix y assume prems: y ∈∘ ℛ∘ ((S ∘_A cat-Set α T)⦇ArrVal⦈)
  with ST.arr-Set-ArrVal-vdomain obtain x
    where x: x ∈∘ A and y-def: y = (S ∘_A cat-Set α T)⦇ArrVal⦈⦇x⦈
    by force
  show y ∈∘ ℛ∘ (S⦇ArrVal⦈)
  proof(intro S.ArrVal.vsv-vimageI2', unfold cat-Set-cs-simps)
    from assms(1,2) x show y = S⦇ArrVal⦈⦇T⦇ArrVal⦈⦇x⦈⦈
```

```
    unfolding y-def
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms(2) x show T(⁅ArrVal⁆)(⁅x⁆) ∈∘ B
    by (cs-concl cs-intro: cat-Set-cs-intros)
 qed
qed
```

17.3 Isomorphism

```
lemma cat-Set-is-iso-arrI[intro]:
 — See [1]⁹).
 assumes T : A ↦_{cat-Set α} B
   and v11 (T(⁅ArrVal⁆))
   and 𝒟∘ (T(⁅ArrVal⁆)) = A
   and ℛ∘ (T(⁅ArrVal⁆)) = B
 shows T : A ↦_{iso cat-Set α} B
proof–
 interpret T: arr-Set α T by (rule cat-Set-is-arrD(1)[OF assms(1)])
 note [cat-cs-intros] = cat-Par-is-iso-arrI
 from T.wide-replete-subcategory-cat-Set-cat-Par assms have
   T : A ↦_{iso cat-Par α} B
   by (cs-concl cs-intro: cat-cs-intros cat-sub-cs-intros cat-sub-fw-cs-intros)
 with T.wide-replete-subcategory-cat-Set-cat-Par assms show
   T : A ↦_{iso cat-Set α} B
   by (cs-concl cs-shallow cs-simp: cat-sub-bw-cs-simps)
qed

lemma cat-Set-is-iso-arrD[dest]:
 assumes T : A ↦_{iso cat-Set α} B
 shows T : A ↦_{cat-Set α} B
   and v11 (T(⁅ArrVal⁆))
   and 𝒟∘ (T(⁅ArrVal⁆)) = A
   and ℛ∘ (T(⁅ArrVal⁆)) = B
proof–
 from assms have T: T : A ↦_{cat-Set α} B by auto
 interpret T: arr-Set α T by (rule cat-Set-is-arrD(1)[OF T])
 from T.wide-replete-subcategory-cat-Set-cat-Par assms have T:
   T : A ↦_{iso cat-Par α} B
   by (cs-concl cs-shallow cs-intro: cat-sub-cs-intros cat-sub-fw-cs-intros)
 show v11 (T(⁅ArrVal⁆)) 𝒟∘ (T(⁅ArrVal⁆)) = A ℛ∘ (T(⁅ArrVal⁆)) = B
   by (intro cat-Par-is-iso-arrD[OF T])+
qed (rule is-iso-arrD(1)[OF assms])

lemma cat-Set-is-iso-arr:
 T : A ↦_{iso cat-Set α} B ⟷
   T : A ↦_{cat-Set α} B ∧
   v11 (T(⁅ArrVal⁆)) ∧
   𝒟∘ (T(⁅ArrVal⁆)) = A ∧
   ℛ∘ (T(⁅ArrVal⁆)) = B
 by auto

lemma (in 𝒵) cat-Set-is-iso-arr-if-monic-and-epic:
 assumes F : A ↦_{mon cat-Set α} B and F : A ↦_{epi cat-Set α} B
 shows F : A ↦_{iso cat-Set α} B
proof–
 note cat-Set-is-monic-arrD[OF assms(1)] cat-Set-is-epic-arrD[OF assms(2)]
 note FD = this(1,2,3,5) cat-Set-is-arrD[OF this(1)]
```

[9] https://ncatlab.org/nlab/show/isomorphism

show *?thesis* **by** (*intro cat-Set-is-iso-arrI FD*)
qed

17.4 The inverse arrow

lemma *cat-Set-ArrVal-app-is-arr*[*cat-cs-intros*]:
assumes $f : a \mapsto_{\mathfrak{A}} b$
and *category* α $\mathfrak{A}$
and $F : Hom\ \mathfrak{A}\ a\ b \mapsto_{cat\text{-}Set\ \alpha} Hom\ \mathfrak{B}\ c\ d$
shows $F(\lfloor ArrVal \rfloor)(\lfloor f \rfloor) : c \mapsto_{\mathfrak{B}} d$
proof–
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*2*))
interpret F: *arr-Set* α F **by** (*rule cat-Set-is-arrD*[*OF assms*(*3*)])
from *assms* **have** $F(\lfloor ArrVal \rfloor)(\lfloor f \rfloor) \in_{\circ} Hom\ \mathfrak{B}\ c\ d$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Set-cs-intros*)
then show *?thesis* **unfolding** *in-Hom-iff* **by** *simp*
qed

abbreviation (*input*) *converse-Set* :: $V \Rightarrow V$ ($\langle(\text{-}^{-1}{}_{Set})\rangle$ [*1000*] *999*)
where $a^{-1}{}_{Set} \equiv a^{-1}{}_{Rel}$

lemma *cat-Set-the-inverse*[*cat-Set-cs-simps*]:
assumes $T : A \mapsto_{iso\ cat\text{-}Set\ \alpha} B$
shows $T^{-1}{}_{C\ cat\text{-}Set\ \alpha} = T^{-1}{}_{Set}$
proof–
from *assms* **have** *T*: $T : A \mapsto_{cat\text{-}Set\ \alpha} B$ **by** *auto*
interpret *arr-Set* α T **by** (*rule cat-Set-is-arrD*(*1*)[*OF T*])
from *wide-replete-subcategory-cat-Set-cat-Par assms* **have** *T*:
$T : A \mapsto_{iso\ cat\text{-}Par\ \alpha} B$
by (*cs-concl* **cs-shallow cs-intro**: *cat-sub-cs-intros cat-sub-fw-cs-intros*)
from *wide-replete-subcategory-cat-Set-cat-Par assms*
have [*symmetric*, *cat-cs-simps*]: $T^{-1}{}_{C\ cat\text{-}Par\ \alpha} = T^{-1}{}_{C\ cat\text{-}Set\ \alpha}$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-sub-bw-cs-simps* **cs-intro**: *cat-sub-cs-intros*
)
from *T* **show** $T^{-1}{}_{C\ cat\text{-}Set\ \alpha} = T^{-1}{}_{Set}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-Par-cs-simps cat-cs-simps* **cs-intro**: $\mathcal{Z}$-β)
qed

lemma *cat-Set-the-inverse-app*[*cat-cs-intros*]:
assumes $T : A \mapsto_{iso\ cat\text{-}Set\ \alpha} B$
and $a \in_{\circ} A$
and [*cat-cs-simps*]: $T(\lfloor ArrVal \rfloor)(\lfloor a \rfloor) = b$
shows $(T^{-1}{}_{C\ cat\text{-}Set\ \alpha})(\lfloor ArrVal \rfloor)(\lfloor b \rfloor) = a$
proof–
from *assms* **have** *T*: $T : A \mapsto_{cat\text{-}Set\ \alpha} B$ **by** *auto*
interpret *arr-Set* α T **by** (*rule cat-Set-is-arrD*(*1*)[*OF T*])
note T = *cat-Set-is-iso-arrD*[*OF assms*(*1*)]
interpret *T*: *v11* $\langle T(\lfloor ArrVal \rfloor)\rangle$ **by** (*rule T*(*2*))
from *T.v11-axioms assms*(*1*,*2*) **show** $T^{-1}{}_{C\ cat\text{-}Set\ \alpha}(\lfloor ArrVal \rfloor)(\lfloor b \rfloor) = a$
by
(
cs-concl **cs-shallow**
cs-simp:
converse-Rel-components V-cs-simps cat-Set-cs-simps cat-cs-simps
cs-intro: *cat-arrow-cs-intros cat-cs-intros*

```
    )
qed

lemma cat-Set-ArrVal-app-the-inverse-is-arr[cat-cs-intros]:
  assumes f : c ↦_𝔅 d
    and category α 𝔅
    and F : Hom 𝔄 a b ↦_iso cat-Set α Hom 𝔅 c d
  shows F^-1_C cat-Set α(|ArrVal|)(|f|) : a ↦_𝔄 b
proof–
  interpret 𝔅: category α 𝔅 by (rule assms(2))
  from cat-Set-is-iso-arrD[OF assms(3)] interpret F: arr-Set α F
    by (simp add: cat-Set-is-arrD)
  from assms have F^-1_C cat-Set α(|ArrVal|)(|f|) ∈_∘ Hom 𝔄 a b
    by (cs-concl cs-intro: cat-cs-intros cat-arrow-cs-intros)
  then show ?thesis unfolding in-Hom-iff by simp
qed

lemma cat-Set-app-the-inverse-app[cat-cs-simps]:
  assumes F : A ↦_iso cat-Set α B and b ∈_∘ B
  shows F(|ArrVal|)(|F^-1_C cat-Set α(|ArrVal|)(|b|)|) = b
proof–
  note F = cat-Set-is-iso-arrD[OF assms(1)]
  note F = F cat-Set-is-arrD[OF F(1)]
  interpret F: arr-Set α F by (rule cat-Set-is-arrD[OF F(1)])
  from assms have [cat-cs-simps]:
    F ∘_A cat-Set α F^-1_C cat-Set α = cat-Set α(|CId|)(|B|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms have [cat-cs-simps]:
    F(|ArrVal|)(|F^-1_C cat-Set α(|ArrVal|)(|b|)|) =
      (F ∘_A cat-Set α F^-1_C cat-Set α)(|ArrVal|)(|b|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-arrow-cs-intros cat-cs-intros
      )
  from assms F(1) F.arr-Par-ArrCod-in-Vset[unfolded F] show ?thesis
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
qed

lemma cat-Set-the-inverse-app-app[cat-cs-simps]:
  assumes F : A ↦_iso cat-Set α B and a ∈_∘ A
  shows F^-1_C cat-Set α(|ArrVal|)(|F(|ArrVal|)(|a|)|) = a
proof–
  note F = cat-Set-is-iso-arrD[OF assms(1)]
  note F = F cat-Set-is-arrD[OF F(1)]
  interpret F: arr-Set α F by (rule cat-Set-is-arrD[OF F(1)])
  from assms have [cat-cs-simps]:
    F^-1_C cat-Set α ∘_A cat-Set α F = cat-Set α(|CId|)(|A|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms have [cat-cs-simps]:
    F^-1_C cat-Set α(|ArrVal|)(|F(|ArrVal|)(|a|)|) =
      (F^-1_C cat-Set α ∘_A cat-Set α F)(|ArrVal|)(|a|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-arrow-cs-intros cat-cs-intros
      )
  from assms F(1) F.arr-Par-ArrDom-in-Vset[unfolded F] show ?thesis
```

by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

17.5 Conversion of a single-valued relation to an arrow in *Set*

17.5.1 Definition and elementary properties

definition *cat-Set-arr-of-vsv* :: $V \Rightarrow V \Rightarrow V$
where *cat-Set-arr-of-vsv f B* = $[f, \mathcal{D}_\circ\ f, B]_\circ$

Components.

lemma *cat-Set-arr-of-vsv-components*:
shows [*cat-Set-cs-simps*]: *cat-Set-arr-of-vsv f B*(|*ArrVal*|) = *f*
and [*cat-Set-cs-simps*]: *cat-Set-arr-of-vsv f B*(|*ArrDom*|) = $\mathcal{D}_\circ\ f$
and [*cat-cs-simps*, *cat-Set-cs-simps*]: *cat-Set-arr-of-vsv f B*(|*ArrCod*|) = *B*
unfolding *cat-Set-arr-of-vsv-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

17.5.2 Conversion of a single-valued relation to an arrow in *Set* is an arrow in *Set*

lemma (**in** $\mathcal{Z}$) *cat-Set-arr-of-vsv-is-arr*:
assumes *vsv r*
and $\mathcal{R}_\circ\ r \subseteq_\circ B$
and $\mathcal{D}_\circ\ r \in_\circ$ *cat-Set* α(|*Obj*|)
and $B \in_\circ$ *cat-Set* α(|*Obj*|)
shows *cat-Set-arr-of-vsv r B* : $\mathcal{D}_\circ\ r \mapsto_{\textit{cat-Set}\ \alpha} B$
proof–
interpret *r*: *vsv r* **by** (*rule assms*)
show *?thesis*
proof(*intro cat-Set-is-arrI arr-SetI*, *unfold cat-Set-arr-of-vsv-components*)
show *vfsequence* (*cat-Set-arr-of-vsv r B*)
unfolding *cat-Set-arr-of-vsv-def* **by** *auto*
show *vcard* (*cat-Set-arr-of-vsv r B*) = $3_{\mathbf{N}}$
unfolding *cat-Set-arr-of-vsv-def* **by** (*auto simp*: *nat-omega-simps*)
qed (*use assms* **in** ‹*auto simp*: *cat-Set-components*›)
qed

17.6 Left restriction for *Set*

17.6.1 Definition and elementary properties

definition *vlrestriction-Set* :: $V \Rightarrow V \Rightarrow V$ (**infixr** ‹$\restriction^l_{Set}$› *80*)
where $T \restriction^l_{Set} C = [T(|ArrVal|) \restriction^l_\circ C, C, T(|ArrCod|)]_\circ$

Components.

lemma *vlrestriction-Set-components*:
shows [*cat-Set-cs-simps*]: $(T \restriction^l_{Set} C)(|ArrVal|) = T(|ArrVal|) \restriction^l_\circ C$
and [*cat-cs-simps*, *cat-Set-cs-simps*]: $(T \restriction^l_{Set} C)(|ArrDom|) = C$
and [*cat-cs-simps*, *cat-Set-cs-simps*]: $(T \restriction^l_{Set} C)(|ArrCod|) = T(|ArrCod|)$
unfolding *vlrestriction-Set-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

17.6.2 Arrow value

lemma *vlrestriction-Set-ArrVal-vdomain*[*cat-cs-simps*]:
assumes $T : A \mapsto_{\textit{cat-Set}\ \alpha} B$ **and** $C \subseteq_\circ A$
shows $\mathcal{D}_\circ\ ((T \restriction^l_{Set} C)(|ArrVal|)) = C$
proof–
note *TD* = *cat-Set-is-arrD*[*OF assms*(*1*)]

```
  interpret T: arr-Set α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro TD)+
  from assms show ?thesis
    unfolding vlrestriction-Set-components
    by (cs-concl cs-simp: V-cs-simps cat-cs-simps cs-intro: V-cs-intros)
qed

lemma vlrestriction-Set-ArrVal-app[cat-cs-simps]:
  assumes T : A ↦_{cat-Set α} B and C ⊆∘ A and x ∈∘ C
  shows (T ↾^l_{Set} C)(|ArrVal|)(|x|) = T(|ArrVal|)(|x|)
proof–
  interpret T: arr-Set α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro cat-Set-is-arrD[OF assms(1)])+
  from assms have x: x ∈∘ A by auto
  with assms show ?thesis
    unfolding vlrestriction-Set-components
    by (cs-concl cs-simp: V-cs-simps cat-cs-simps cs-intro: V-cs-intros)
qed
```

17.6.3 Left restriction for *Set* is an arrow in *Set*

```
lemma vlrestriction-Set-is-arr:
  assumes T : A ↦_{cat-Set α} B and C ⊆∘ A
  shows T ↾^l_{Set} C : C ↦_{cat-Set α} B
proof–
  note TD = cat-Set-is-arrD[OF assms(1)]
  interpret T: arr-Set α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro TD)+
  show ?thesis
  proof(intro cat-Set-is-arrI arr-SetI, unfold cat-Set-cs-simps TD(2,3))
    show vfsequence (T ↾^l_{Set} C)
      unfolding vlrestriction-Set-def by auto
    show vcard (T ↾^l_{Set} C) = 3_ℕ
      unfolding vlrestriction-Set-def by (simp add: nat-omega-simps)
    from assms show 𝒟∘ (T(|ArrVal|) ↾^l∘ C) = C
      by (cs-concl cs-simp: V-cs-simps cat-cs-simps cs-intro: cat-cs-intros)
    show ℛ∘ (T(|ArrVal|) ↾^l∘ C) ⊆∘ B
      unfolding app-vimage-def[symmetric]
    proof(intro vsubsetI)
      fix x assume prems: x ∈∘ T(|ArrVal|) `∘ C
      then obtain c where c ∈∘ C and x-def: x = T(|ArrVal|)(|c|) by auto
      with assms(2) have c: c ∈∘ A by auto
      from c assms show x ∈∘ B
        unfolding x-def by (cs-concl cs-intro: cat-Set-cs-intros)
    qed
    from assms(2) show C ∈∘ Vset α
      using vsubset-in-VsetI by (auto simp: T.arr-Set-ArrDom-in-Vset)
  qed (auto simp: T.arr-Set-ArrCod-in-Vset)
qed

lemma (in 𝒵) vlrestriction-Set-is-monic-arr:
  assumes T : A ↦_{mon cat-Set α} B and C ⊆∘ A
  shows T ↾^l_{Set} C : C ↦_{mon cat-Set α} B
proof–
  note cat-Set-is-monic-arrD[OF assms(1)]
```

```
  note TD = this cat-Set-is-arrD[OF this(1)]
  interpret F: arr-Set α T by (intro TD)+
  interpret ArrVal: v11 ‹T(|ArrVal|)› by (rule TD(2))
  show ?thesis
  proof
    (
      intro
        cat-Set-is-monic-arrI
        vlrestriction-Set-is-arr[OF TD(1) assms(2)],
      unfold cat-Set-cs-simps
    )
    from TD(1) assms(2) show 𝒟∘ (T(|ArrVal|) ↾ˡ∘ C) = C
      by (cs-concl cs-simp: V-cs-simps cat-cs-simps)
  qed auto
qed
```

17.7 Right restriction for *Set*

17.7.1 Definition and elementary properties

```
definition vrrestriction-Set :: V ⇒ V ⇒ V (infixr ‹↾ʳ_Set› 80)
  where T ↾ʳ_Set C = [T(|ArrVal|) ↾ʳ∘ C, T(|ArrDom|), C]∘
```

Components.

```
lemma vrrestriction-Set-components:
  shows [cat-Set-cs-simps]: (T ↾ʳ_Set C)(|ArrVal|) = T(|ArrVal|) ↾ʳ∘ C
    and [cat-cs-simps, cat-Set-cs-simps]: (T ↾ʳ_Set C)(|ArrDom|) = T(|ArrDom|)
    and [cat-cs-simps, cat-Set-cs-simps]: (T ↾ʳ_Set C)(|ArrCod|) = C
  unfolding vrrestriction-Set-def arr-field-simps
  by (simp-all add: nat-omega-simps)
```

17.7.2 Arrow value

```
lemma vrrestriction-Set-ArrVal-app[cat-cs-simps]:
  assumes T : A ↦_cat-Set α B and ℛ∘ (T(|ArrVal|)) ⊆∘ C
  shows (T ↾ʳ_Set C)(|ArrVal|) = T(|ArrVal|)
proof–
  interpret T: arr-Set α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro cat-Set-is-arrD[OF assms(1)])+
  from assms show ?thesis unfolding cat-Set-cs-simps by simp
qed
```

17.7.3 Right restriction for *Set* is an arrow in *Set*

```
lemma vrrestriction-Set-is-arr:
  assumes T : A ↦_cat-Set α B
    and ℛ∘ (T(|ArrVal|)) ⊆∘ C
    and C ∈∘ cat-Set α(|Obj|)
  shows T ↾ʳ_Set C : A ↦_cat-Set α C
proof–
  note TD = cat-Set-is-arrD[OF assms(1)]
  interpret T: arr-Set α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro TD)+
  show ?thesis
  proof(intro cat-Set-is-arrI arr-SetI, unfold cat-Set-cs-simps)
    show vfsequence (T ↾ʳ_Set C) unfolding vrrestriction-Set-def by auto
    show vcard (T ↾ʳ_Set C) = 3_ℕ
```

```
    unfolding vrrestriction-Set-def by (simp add: nat-omega-simps)
  qed
    (
      use assms(2,3) in
        ‹
          auto simp:
            TD(2)
            cat-Set-components
            T.arr-Set-ArrVal-vdomain
            T.arr-Set-ArrDom-in-Vset
        ›
    )
qed

lemma vrrestriction-Set-is-arr'[cat-cs-intros]:
  assumes T : A ↦_{cat-Set α} B
    and ℛ∘ (T(|ArrVal|)) ⊆∘ C
    and C ∈∘ cat-Set α(|Obj|)
    and C' = C
    and 𝔄' = cat-Set α
  shows T ↾^r_{Set} C : A ↦_{𝔄'} C'
  using assms(1–3) unfolding assms(4,5) by (rule vrrestriction-Set-is-arr)
```

17.7.4 Further properties

```
lemma
  assumes T : A ↦_{cat-Set α} B
  shows vrrestriction-Set-vrange-is-arr:
      T ↾^r_{Set} ℛ∘ (T(|ArrVal|)) : A ↦_{cat-Set α} ℛ∘ (T(|ArrVal|))
    and vrrestriction-Set-vrange-ArrVal-app[cat-cs-simps, cat-Set-cs-simps]:
      (T ↾^r_{Set} ℛ∘ (T(|ArrVal|)))(|ArrVal|) = T(|ArrVal|)
proof(intro vrrestriction-Set-is-arr, rule assms)
  note TD = cat-Set-is-arrD[OF assms(1)]
  interpret T: arr-Set α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro TD)+
  show ℛ∘ (T(|ArrVal|)) ∈∘ cat-Set α(|Obj|)
    by (auto simp: cat-Set-components T.arr-Rel-ArrVal-in-Vset vrange-in-VsetI)
qed (auto intro: vrrestriction-Set-ArrVal-app[OF assms])

lemma (in 𝒵) vrrestriction-Set-vrange-is-iso-arr:
  assumes T : A ↦_{mon cat-Set α} B
  shows T ↾^r_{Set} ℛ∘ (T(|ArrVal|)) : A ↦_{iso cat-Set α} ℛ∘ (T(|ArrVal|))
proof–
  note cat-Set-is-monic-arrD[OF assms]
  note TD = this cat-Set-is-arrD[OF this(1)]
  interpret T: arr-Set α T by (intro TD)+
  show ?thesis
    by
      (
        intro cat-Set-is-iso-arrI vrrestriction-Set-vrange-is-arr[OF TD(1)],
        unfold cat-Set-cs-simps
      )
      (simp-all add: TD(2,3))
qed
```

17.7.5 Connections

lemma *cat-Set-Comp-vrrestriction-Set*:
assumes $S : B \mapsto_{cat\text{-}Set\ \alpha} C$
and $T : A \mapsto_{cat\text{-}Set\ \alpha} B$
and $\mathcal{R}_{\circ}\ (S(\!|ArrVal|\!)) \subseteq_{\circ} D$
and $D \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$
shows $S \restriction^{r}_{Set}\ D \circ_{A\, cat\text{-}Set\ \alpha}\ T = (S \circ_{A\, cat\text{-}Set\ \alpha}\ T) \restriction^{r}_{Set}\ D$
proof–

note *SD* = *cat-Set-is-arrD*[*OF assms*(*1*)]
interpret *S*: *arr-Set* α *S*
rewrites [*cat-cs-simps*]: $S(\!|ArrDom|\!) = B$ **and** [*cat-cs-simps*]: $S(\!|ArrCod|\!) = C$
by (*intro SD*)+
note *TD* = *cat-Set-is-arrD*[*OF assms*(*2*)]
interpret *T*: *arr-Set* α *T*
rewrites [*cat-cs-simps*]: $T(\!|ArrDom|\!) = A$ **and** [*cat-cs-simps*]: $T(\!|ArrCod|\!) = B$
by (*intro TD*)+

from *assms*(*3*) *S.arr-Par-ArrVal-vrange* **have** *RS-D*: $\mathcal{R}_{\circ}\ (S(\!|ArrVal|\!)) \subseteq_{\circ} D$ **by** *auto*

from *assms*(*1,2*) **have** $S \circ_{A\, cat\text{-}Set\ \alpha}\ T : A \mapsto_{cat\text{-}Set\ \alpha} C$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)

from *assms*(*1,2*) **have** $\mathcal{R}_{\circ}\ ((S \circ_{A\, cat\text{-}Set\ \alpha}\ T)(\!|ArrVal|\!)) \subseteq_{\circ} \mathcal{R}_{\circ}\ (S(\!|ArrVal|\!))$
by (*intro cat-Set-Comp-ArrVal-vrange*)
with *assms*(*3*) **have** *RST*: $\mathcal{R}_{\circ}\ ((S \circ_{A\, cat\text{-}Set\ \alpha}\ T)(\!|ArrVal|\!)) \subseteq_{\circ} D$ **by** *auto*

from *assms*(*1,2,4*) *RS-D* **have** *SD-T*:
$S \restriction^{r}_{Set}\ D \circ_{A\, cat\text{-}Set\ \alpha}\ T : A \mapsto_{cat\text{-}Set\ \alpha} D$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_{\circ}\ ((S \restriction^{r}_{Set}\ D \circ_{A\, cat\text{-}Set\ \alpha}\ T)(\!|ArrVal|\!)) = A$
by (*simp add*: *cat-cs-simps*)

from *assms*(*1,2,4*) *RST* **have** *ST-D*:
$(S \circ_{A\, cat\text{-}Set\ \alpha}\ T) \restriction^{r}_{Set}\ D : A \mapsto_{cat\text{-}Set\ \alpha} D$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_{\circ}\ (((S \circ_{A\, cat\text{-}Set\ \alpha}\ T) \restriction^{r}_{Set}\ D)(\!|ArrVal|\!)) = A$
by (*simp add*: *cat-cs-simps*)

show $S \restriction^{r}_{Set}\ D \circ_{A\, cat\text{-}Set\ \alpha}\ T = (S \circ_{A\, cat\text{-}Set\ \alpha}\ T) \restriction^{r}_{Set}\ D$
proof(*rule arr-Set-eqI*[*of* α])
show
$(S \restriction^{r}_{Set}\ D \circ_{A\, cat\text{-}Set\ \alpha}\ T)(\!|ArrVal|\!) =$
$((S \circ_{A\, cat\text{-}Set\ \alpha}\ T) \restriction^{r}_{Set}\ D)(\!|ArrVal|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** $a \in_{\circ} A$
with *assms*(*1,2,4*) *RST RS-D* **show**
$(S \restriction^{r}_{Set}\ D \circ_{A\, cat\text{-}Set\ \alpha}\ T)(\!|ArrVal|\!)(\!|a|\!) =$
$((S \circ_{A\, cat\text{-}Set\ \alpha}\ T) \restriction^{r}_{Set}\ D)(\!|ArrVal|\!)(\!|a|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*use SD-T ST-D* **in** ‹*auto dest*: *cat-Set-is-arrD*›)
qed (*use SD-T ST-D* **in** ‹*auto simp*: *cat-Set-is-arrD*›)

qed

lemma (**in** $\mathcal{Z}$) *cat-Set-CId-vrrestriction-Set*[*cat-cs-simps*]:
assumes $A \subseteq_{\circ} B$ **and** $B \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$

shows *cat-Set* $\alpha(\!|CId|\!)(\!|A|\!) \restriction^{r}_{Set} B = incl\text{-}Set\ A\ B$
proof–

from *assms* **have** *A*: $A \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$
unfolding *cat-Set-components* **by** *auto*
from *A* **have** *CId-A*: $cat\text{-}Set\ \alpha(\!|CId|\!)(\!|A|\!) : A \mapsto_{cat\text{-}Set\ \alpha} A$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
with *cat-Set-is-arrD*[*OF CId-A*] *assms*(*1*) **have** *RA-B*:
$\mathcal{R}_{\circ}\ (cat\text{-}Set\ \alpha(\!|CId|\!)(\!|A|\!)(\!|ArrVal|\!)) \subseteq_{\circ} B$
by (*auto intro*: *arr-Set.arr-Set-ArrVal-vrange*)

with *assms A assms*(*1,2*) **have** *lhs-is-arr*:
$cat\text{-}Set\ \alpha(\!|CId|\!)(\!|A|\!) \restriction^{r}_{Set} B : A \mapsto_{cat\text{-}Set\ \alpha} B$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_{\circ}\ ((cat\text{-}Set\ \alpha(\!|CId|\!)(\!|A|\!) \restriction^{r}_{Set} B)(\!|ArrVal|\!)) = A$
by (*simp add*: *cat-cs-simps*)

from *A assms*(*1,2*) **have** *rhs-is-arr*: $incl\text{-}Set\ A\ B : A \mapsto_{cat\text{-}Set\ \alpha} B$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_{\circ}\ ((incl\text{-}Set\ A\ B)(\!|ArrVal|\!)) = A$
by (*simp add*: *cat-cs-simps*)

show *?thesis*
proof(*rule arr-Set-eqI*[*of* α])
show $(cat\text{-}Set\ \alpha(\!|CId|\!)(\!|A|\!) \restriction^{r}_{Set} B)(\!|ArrVal|\!) = incl\text{-}Rel\ A\ B(\!|ArrVal|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** $a \in_{\circ} A$
with *A RA-B* **show**
$(cat\text{-}Set\ \alpha(\!|CId|\!)(\!|A|\!) \restriction^{r}_{Set} B)(\!|ArrVal|\!)(\!|a|\!) = incl\text{-}Rel\ A\ B(\!|ArrVal|\!)(\!|a|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*use lhs-is-arr rhs-is-arr* **in** ‹*auto dest*: *cat-Set-is-arrD*›)
qed (*use lhs-is-arr rhs-is-arr* **in** ‹*auto simp*: *cat-Set-is-arrD*›)

qed

lemma *cat-Set-Comp-incl-Rel-vrrestriction-Set*[*cat-cs-simps*]:
assumes $F : A \mapsto_{cat\text{-}Set\ \alpha} B$ **and** $C \subseteq_{\circ} B$ **and** $\mathcal{R}_{\circ}\ (F(\!|ArrVal|\!)) \subseteq_{\circ} C$
shows $incl\text{-}Rel\ C\ B \circ_{A\, cat\text{-}Set\ \alpha} F \restriction^{r}_{Set} C = F$
proof–
note *FD* = *cat-Set-is-arrD*[*OF assms*(*1*)]
interpret *F*: *arr-Set* α *F*
rewrites [*cat-cs-simps*]: $F(\!|ArrDom|\!) = A$ **and** [*cat-cs-simps*]: $F(\!|ArrCod|\!) = B$
by (*intro FD*)+
from *assms*(*2*) **have** *C*: $C \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$
unfolding *cat-Set-components*(*1*) **by** (*auto intro*: *F.arr-Par-ArrCod-in-Vset*)
from *assms C* **have** *lhs-is-arr*:
$incl\text{-}Rel\ C\ B \circ_{A\, cat\text{-}Set\ \alpha} F \restriction^{r}_{Set} C : A \mapsto_{cat\text{-}Set\ \alpha} B$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_{\circ}\ ((incl\text{-}Rel\ C\ B \circ_{A\, cat\text{-}Set\ \alpha} F \restriction^{r}_{Set} C)(\!|ArrVal|\!)) = A$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
from *assms*(*1*) **have** *dom-rhs*: $\mathcal{D}_{\circ}\ (F(\!|ArrVal|\!)) = A$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
show *?thesis*
proof(*rule arr-Set-eqI*[*of* α])
show $(incl\text{-}Rel\ C\ B \circ_{A\, cat\text{-}Set\ \alpha} F \restriction^{r}_{Set} C)(\!|ArrVal|\!) = F(\!|ArrVal|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** *prems*: $a \in_{\circ} A$
with *assms F.ArrVal.vsv-vimageI2* **have** $F(\!|ArrVal|\!)(\!|a|\!) \in_{\circ} C$

by (*auto simp*: *F.arr-Set-ArrVal-vdomain*)
with *prems assms C* **show**
(*incl-Rel C B* $\circ_{A\,cat\text{-}Set\,\alpha}$ *F* $\restriction^r_{Set}$ *C*)(|*ArrVal*|)(|*a*|) = *F*(|*ArrVal*|)(|*a*|)
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*use assms*(*1*) *lhs-is-arr* **in** ‹*auto dest*: *cat-Set-is-arrD*›)
qed (*use assms*(*1*) *lhs-is-arr* **in** ‹*auto dest*: *cat-Set-is-arrD*›)
qed

17.8 Projection arrows for *vtimes*

17.8.1 Definition and elementary properties

definition *vfst-arrow* :: $V \Rightarrow V \Rightarrow V$
where *vfst-arrow A B* = [(λ*ab*∈$_\circ$*A* ×$_\circ$ *B*. *vfst ab*), *A* ×$_\circ$ *B*, *A*]$_\circ$

definition *vsnd-arrow* :: $V \Rightarrow V \Rightarrow V$
where *vsnd-arrow A B* = [(λ*ab*∈$_\circ$*A* ×$_\circ$ *B*. *vsnd ab*), *A* ×$_\circ$ *B*, *B*]$_\circ$

Components.

lemma *vfst-arrow-components*:
shows *vfst-arrow A B*(|*ArrVal*|) = (λ*ab*∈$_\circ$*A* ×$_\circ$ *B*. *vfst ab*)
and [*cat-cs-simps*]: *vfst-arrow A B*(|*ArrDom*|) = *A* ×$_\circ$ *B*
and [*cat-cs-simps*]: *vfst-arrow A B*(|*ArrCod*|) = *A*
unfolding *vfst-arrow-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *vsnd-arrow-components*:
shows *vsnd-arrow A B*(|*ArrVal*|) = (λ*ab*∈$_\circ$*A* ×$_\circ$ *B*. *vsnd ab*)
and [*cat-cs-simps*]: *vsnd-arrow A B*(|*ArrDom*|) = *A* ×$_\circ$ *B*
and [*cat-cs-simps*]: *vsnd-arrow A B*(|*ArrCod*|) = *B*
unfolding *vsnd-arrow-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

17.8.2 Arrow value

mk-VLambda *vfst-arrow-components*(*1*)
|*vsv vfst-arrow-ArrVal-vsv*[*cat-cs-intros*]|
|*vdomain vfst-arrow-ArrVal-vdomain*[*cat-cs-simps*]|
|*app vfst-arrow-ArrVal-app′*|

mk-VLambda *vsnd-arrow-components*(*1*)
|*vsv vsnd-arrow-ArrVal-vsv*[*cat-cs-intros*]|
|*vdomain vsnd-arrow-ArrVal-vdomain*[*cat-cs-simps*]|
|*app vsnd-arrow-ArrVal-app′*|

lemma *vfst-arrow-ArrVal-app*[*cat-cs-simps*]:
assumes *ab* = ⟨*a*, *b*⟩ **and** *ab* ∈$_\circ$ *A* ×$_\circ$ *B*
shows *vfst-arrow A B*(|*ArrVal*|)(|*ab*|) = *a*
using *assms*(*2*) **unfolding** *assms*(*1*) **by** (*simp add*: *vfst-arrow-ArrVal-app′*)

lemma *vfst-arrow-vrange*: $\mathcal{R}_\circ$ (*vfst-arrow A B*(|*ArrVal*|)) ⊆$_\circ$ *A*
unfolding *vfst-arrow-components*
proof(*intro vrange-VLambda-vsubset*)
fix *ab* **assume** *ab* ∈$_\circ$ *A* ×$_\circ$ *B*
then obtain *a b* **where** *ab-def*: *ab* = ⟨*a*, *b*⟩ **and** *a*: *a* ∈$_\circ$ *A* **by** *clarsimp*
from *a* **show** *vfst ab* ∈$_\circ$ *A* **unfolding** *ab-def* **by** *simp*
qed

lemma *vsnd-arrow-ArrVal-app*[*cat-cs-simps*]:
assumes *ab* = ⟨*a*, *b*⟩ **and** *ab* ∈$_\circ$ *A* ×$_\circ$ *B*
shows *vsnd-arrow A B*(|*ArrVal*|)(|*ab*|) = *b*

using *assms*(*2*) **unfolding** *assms*(*1*) **by** (*simp add*: *vsnd-arrow-ArrVal-app'*)

lemma *vsnd-arrow-vrange*: $\mathcal{R}_{\circ}$ (*vsnd-arrow A B*(|*ArrVal*|)) $\subseteq_{\circ}$ *B*
unfolding *vsnd-arrow-components*
proof(*intro vrange-VLambda-vsubset*)
fix *ab* **assume** *ab* $\in_{\circ}$ *A* $\times_{\circ}$ *B*
then obtain *a b* **where** *ab-def*: *ab* = $\langle a, b \rangle$ **and** *b*: *b* $\in_{\circ}$ *B* **by** *clarsimp*
from *b* **show** *vsnd ab* $\in_{\circ}$ *B* **unfolding** *ab-def* **by** *simp*
qed

17.8.3 Projection arrows are arrows in the category *Set*

lemma (**in** $\mathcal{Z}$) *vfst-arrow-is-cat-Set-arr-Vset*:
assumes *A* $\in_{\circ}$ *Vset* α **and** *B* $\in_{\circ}$ *Vset* α
shows *vfst-arrow A B* : *A* $\times_{\circ}$ *B* $\mapsto_{\textit{cat-Set}\ \alpha}$ *A*
proof(*intro cat-Set-is-arrI arr-SetI*, *unfold cat-cs-simps*)
show *vfsequence* (*vfst-arrow A B*) **unfolding** *vfst-arrow-def* **by** *simp*
show *vcard* (*vfst-arrow A B*) = $3_{\mathbb{N}}$
unfolding *vfst-arrow-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*vfst-arrow A B*(|*ArrVal*|)) $\subseteq_{\circ}$ *A* **by** (*rule vfst-arrow-vrange*)
qed (*use assms* **in** ‹*cs-concl cs-shallow cs-intro*: *V-cs-intros cat-cs-intros*›)+

lemma (**in** $\mathcal{Z}$) *vfst-arrow-is-cat-Set-arr*:
assumes *A* $\in_{\circ}$ *cat-Set* α(|*Obj*|) **and** *B* $\in_{\circ}$ *cat-Set* α(|*Obj*|)
shows *vfst-arrow A B* : *A* $\times_{\circ}$ *B* $\mapsto_{\textit{cat-Set}\ \alpha}$ *A*
using *assms*
unfolding *cat-Set-components*
by (*rule vfst-arrow-is-cat-Set-arr-Vset*)

lemma (**in** $\mathcal{Z}$) *vfst-arrow-is-cat-Set-arr'*[*cat-rel-par-Set-cs-intros*]:
assumes *A* $\in_{\circ}$ *cat-Set* α(|*Obj*|)
and *B* $\in_{\circ}$ *cat-Set* α(|*Obj*|)
and *AB* = *A* $\times_{\circ}$ *B*
and *A'* = *A*
and $\mathfrak{C}'$ = *cat-Set* α
shows *vfst-arrow A B* : *AB* $\mapsto_{\mathfrak{C}'}$ *A'*
using *assms*(*1–2*) **unfolding** *assms*(*3–5*) **by** (*rule vfst-arrow-is-cat-Set-arr*)

lemmas [*cat-rel-par-Set-cs-intros*] = $\mathcal{Z}$.*vfst-arrow-is-cat-Set-arr'*

lemma (**in** $\mathcal{Z}$) *vsnd-arrow-is-cat-Set-arr-Vset*:
assumes *A* $\in_{\circ}$ *Vset* α **and** *B* $\in_{\circ}$ *Vset* α
shows *vsnd-arrow A B* : *A* $\times_{\circ}$ *B* $\mapsto_{\textit{cat-Set}\ \alpha}$ *B*
proof(*intro cat-Set-is-arrI arr-SetI* , *unfold cat-cs-simps*)
show *vfsequence* (*vsnd-arrow A B*) **unfolding** *vsnd-arrow-def* **by** *simp*
show *vcard* (*vsnd-arrow A B*) = $3_{\mathbb{N}}$
unfolding *vsnd-arrow-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*vsnd-arrow A B*(|*ArrVal*|)) $\subseteq_{\circ}$ *B* **by** (*rule vsnd-arrow-vrange*)
qed (*use assms* **in** ‹*cs-concl cs-shallow cs-intro*: *V-cs-intros cat-cs-intros*›)+

lemma (**in** $\mathcal{Z}$) *vsnd-arrow-is-cat-Set-arr*:
assumes *A* $\in_{\circ}$ *cat-Set* α(|*Obj*|) **and** *B* $\in_{\circ}$ *cat-Set* α(|*Obj*|)
shows *vsnd-arrow A B* : *A* $\times_{\circ}$ *B* $\mapsto_{\textit{cat-Set}\ \alpha}$ *B*
using *assms*
unfolding *cat-Set-components*
by (*rule vsnd-arrow-is-cat-Set-arr-Vset*)

lemma (**in** $\mathcal{Z}$) *vsnd-arrow-is-cat-Set-arr'*[*cat-rel-par-Set-cs-intros*]:

assumes $A \in_{\circ} \mathit{cat\text{-}Set}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Set}\ \alpha(\!|\mathit{Obj}|\!)$
and $AB = A \times_{\circ} B$
and $B' = B$
and $\mathfrak{C}' = \mathit{cat\text{-}Set}\ \alpha$
shows $\mathit{vsnd\text{-}arrow}\ A\ B : AB \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–2*) **unfolding** *assms*(*3–5*) **by** (*rule vsnd-arrow-is-cat-Set-arr*)

lemmas [*cat-rel-par-Set-cs-intros*] = $\mathcal{Z}$.*vsnd-arrow-is-cat-Set-arr'*

17.8.4 Projection arrows are arrows in the category *Par*

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Par-arr*:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$ **and** $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
shows $\mathit{vfst\text{-}arrow}\ A\ B : A \times_{\circ} B \mapsto_{\mathit{cat\text{-}Par}\ \alpha} A$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
from *assms* **show** *?thesis*
unfolding *cat-Par-components*(*1*)
by (*intro Set-Par.subcat-is-arrD vfst-arrow-is-cat-Set-arr-Vset*) *auto*
qed

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Par-arr'*[*cat-rel-Par-set-cs-intros*]:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $AB = A \times_{\circ} B$
and $A' = A$
and $\mathfrak{C}' = \mathit{cat\text{-}Par}\ \alpha$
shows $\mathit{vfst\text{-}arrow}\ A\ B : AB \mapsto_{\mathfrak{C}'} A'$
using *assms*(*1–2*) **unfolding** *assms*(*3–5*) **by** (*rule vfst-arrow-is-cat-Par-arr*)

lemmas [*cat-rel-Par-set-cs-intros*] = $\mathcal{Z}$.*vfst-arrow-is-cat-Par-arr'*

lemma (in $\mathcal{Z}$**)** *vsnd-arrow-is-cat-Par-arr*:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$ **and** $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
shows $\mathit{vsnd\text{-}arrow}\ A\ B : A \times_{\circ} B \mapsto_{\mathit{cat\text{-}Par}\ \alpha} B$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
from *assms* **show** *?thesis*
unfolding *cat-Par-components*(*1*)
by (*intro Set-Par.subcat-is-arrD vsnd-arrow-is-cat-Set-arr-Vset*) *auto*
qed

lemma (in $\mathcal{Z}$**)** *vsnd-arrow-is-cat-Par-arr'*[*cat-rel-Par-set-cs-intros*]:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $AB = A \times_{\circ} B$
and $B' = B$
and $\mathfrak{C}' = \mathit{cat\text{-}Par}\ \alpha$
shows $\mathit{vsnd\text{-}arrow}\ A\ B : AB \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–2*) **unfolding** *assms*(*3–5*) **by** (*rule vsnd-arrow-is-cat-Par-arr*)

lemmas [*cat-rel-Par-set-cs-intros*] = $\mathcal{Z}$.*vsnd-arrow-is-cat-Par-arr'*

17.8.5 Projection arrows are arrows in the category *Rel*

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Rel-arr*:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$ **and** $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
shows *vfst-arrow* $A\ B : A \times_\circ B \mapsto_{\textit{cat-Rel}\ \alpha} A$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
interpret *Par-Rel*: *wide-replete-subcategory* α ‹*cat-Par* α› ‹*cat-Rel* α›
by (*rule wide-replete-subcategory-cat-Par-cat-Rel*)
interpret *Set-Rel*: *subcategory* α ‹*cat-Set* α› ‹*cat-Rel* α›
by
(
rule subcat-trans[
OF Set-Par.subcategory-axioms Par-Rel.subcategory-axioms
]
)
from *assms* **show** *?thesis*
unfolding *cat-Rel-components*(*1*)
by (*intro Set-Rel.subcat-is-arrD vfst-arrow-is-cat-Set-arr-Vset*) *auto*
qed

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Rel-arr'*[*cat-Rel-par-set-cs-intros*]:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $AB = A \times_\circ B$
and $A' = A$
and $\mathfrak{C}' =$ *cat-Rel* α
shows *vfst-arrow* $A\ B : AB \mapsto_{\mathfrak{C}'} A'$
using *assms*(*1–2*) **unfolding** *assms*(*3–5*) **by** (*rule vfst-arrow-is-cat-Rel-arr*)

lemmas [*cat-Rel-par-set-cs-intros*] = $\mathcal{Z}$*.vfst-arrow-is-cat-Rel-arr'*

lemma (in $\mathcal{Z}$**)** *vsnd-arrow-is-cat-Rel-arr*:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$ **and** $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
shows *vsnd-arrow* $A\ B : A \times_\circ B \mapsto_{\textit{cat-Rel}\ \alpha} B$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
interpret *Par-Rel*: *wide-replete-subcategory* α ‹*cat-Par* α› ‹*cat-Rel* α›
by (*rule wide-replete-subcategory-cat-Par-cat-Rel*)
interpret *Set-Rel*: *subcategory* α ‹*cat-Set* α› ‹*cat-Rel* α›
by
(
rule subcat-trans[
OF Set-Par.subcategory-axioms Par-Rel.subcategory-axioms
]
)
from *assms* **show** *?thesis*
unfolding *cat-Rel-components*(*1*)
by (*intro Set-Rel.subcat-is-arrD vsnd-arrow-is-cat-Set-arr-Vset*) *auto*
qed

lemma (in $\mathcal{Z}$**)** *vsnd-arrow-is-cat-Rel-arr'*[*cat-Rel-par-set-cs-intros*]:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $AB = A \times_\circ B$
and $B' = B$

and $\mathfrak{C}' = \textit{cat-Rel}\ \alpha$
shows $\textit{vsnd-arrow}\ A\ B : AB \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–2*) **unfolding** *assms*(*3–5*) **by** (*rule vsnd-arrow-is-cat-Rel-arr*)

lemmas [*cat-Rel-par-set-cs-intros*] = $\mathcal{Z}$.*vsnd-arrow-is-cat-Rel-arr'*

17.8.6 Projection arrows are isomorphisms in the category *Set*

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Set-iso-arr-Vset*:
assumes $A \in_\circ \textit{Vset}\ \alpha$ **and** $b \in_\circ \textit{Vset}\ \alpha$
shows $\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\}) : A \times_\circ \textit{set}\ \{b\} \mapsto_{iso \textit{cat-Set}\ \alpha} A$
proof
(
intro
cat-Set-is-iso-arrI
arr-SetI
vfst-arrow-is-cat-Set-arr-Vset
assms,
unfold cat-cs-simps
)
show *v11* $(\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!))$
proof(*rule vsv.vsv-valeq-v11I*, *unfold cat-cs-simps*)
fix *ab ab'* **assume** *prems*:
$ab \in_\circ A \times_\circ \textit{set}\ \{b\}$
$ab' \in_\circ A \times_\circ \textit{set}\ \{b\}$
$\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!)(\!|ab|\!) = \textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!)(\!|ab'|\!)$
from *prems* **obtain** *a* **where** *ab-def*: $ab = \langle a, b\rangle$ **and** *a*: $a \in_\circ A$
by *clarsimp*
from *prems* **obtain** a' **where** *ab'-def*: $ab' = \langle a', b\rangle$ **and** a': $a' \in_\circ A$
by *clarsimp*
from *prems*(*3*) *a a'* **have** $a = a'$
unfolding *ab-def ab'-def*
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *V-cs-intros*)
then show $ab = ab'$ **unfolding** *ab-def ab'-def* **by** *simp*
qed (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show $\mathcal{R}_\circ\ (\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!)) = A$
proof(*intro vsubset-antisym*)
show $A \subseteq_\circ \mathcal{R}_\circ\ (\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!))$
proof(*intro vsubsetI*)
fix *a* **assume** *a*: $a \in_\circ A$
then have *a-def*: $a = \textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!)(\!|\langle a, b\rangle|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *V-cs-intros*)
from *a assms* **show** $a \in_\circ \mathcal{R}_\circ\ (\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\})(\!|\textit{ArrVal}|\!))$
by (*subst a-def*, *use nothing* **in** ‹*intro vsv.vsv-vimageI2*›)
(*auto simp*: *cat-cs-simps cat-cs-intros*)
qed
qed (*rule vfst-arrow-vrange*)
qed (*use assms* **in** *auto*)

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Set-iso-arr*:
assumes $A \in_\circ \textit{cat-Set}\ \alpha(\!|\textit{Obj}|\!)$ **and** $b \in_\circ \textit{cat-Set}\ \alpha(\!|\textit{Obj}|\!)$
shows $\textit{vfst-arrow}\ A\ (\textit{set}\ \{b\}) : A \times_\circ \textit{set}\ \{b\} \mapsto_{iso \textit{cat-Set}\ \alpha} A$
using *assms*
unfolding *cat-Set-components*
by (*rule vfst-arrow-is-cat-Set-iso-arr-Vset*)

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Set-iso-arr'*[*cat-rel-par-Set-cs-intros*]:
assumes $A \in_\circ \textit{cat-Set}\ \alpha(\!|\textit{Obj}|\!)$

and $b \in_{\circ} cat\text{-}Set\ \alpha(\lfloor Obj \rfloor)$
and $AB = A \times_{\circ} set\ \{b\}$
and $A' = A$
and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
shows $vfst\text{-}arrow\ A\ (set\ \{b\}) : AB \mapsto_{iso\mathfrak{C}'} A$
using *assms*(*1–2*)
unfolding *assms*(*3–5*)
by (*rule vfst-arrow-is-cat-Set-iso-arr*)

lemmas [*cat-rel-par-Set-cs-intros*] = $\mathcal{Z}$.*vfst-arrow-is-cat-Set-iso-arr'*

lemma (**in** $\mathcal{Z}$) *vsnd-arrow-is-cat-Set-iso-arr-Vset*:
assumes $a \in_{\circ} Vset\ \alpha$ **and** $B \in_{\circ} Vset\ \alpha$
shows $vsnd\text{-}arrow\ (set\ \{a\})\ B : set\ \{a\} \times_{\circ} B \mapsto_{iso cat\text{-}Set\ \alpha} B$
proof
(
intro
cat-Set-is-iso-arrI
arr-SetI
vsnd-arrow-is-cat-Set-arr-Vset
assms,
unfold cat-cs-simps
)
show *v11* $(vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor))$
proof(*rule vsv.vsv-valeq-v11I*, *unfold cat-cs-simps*)
fix *ab ab'* **assume** *prems*:
$ab \in_{\circ} set\ \{a\} \times_{\circ} B$
$ab' \in_{\circ} set\ \{a\} \times_{\circ} B$
$vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor)(\lfloor ab \rfloor) = vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor)(\lfloor ab' \rfloor)$
from *prems* **obtain** *b* **where** *ab-def*: $ab = \langle a, b \rangle$ **and** *b*: $b \in_{\circ} B$
by *clarsimp*
from *prems* **obtain** *b'* **where** *ab'-def*: $ab' = \langle a, b' \rangle$ **and** *b'*: $b' \in_{\circ} B$
by *clarsimp*
from *prems*(*3*) *b b'* **have** $b = b'$
unfolding *ab-def ab'-def*
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *V-cs-intros*)
then show $ab = ab'$ **unfolding** *ab-def ab'-def* **by** *simp*
qed (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show $\mathcal{R}_{\circ}\ (vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor)) = B$
proof(*intro vsubset-antisym*)
show $B \subseteq_{\circ} \mathcal{R}_{\circ}\ (vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor))$
proof(*intro vsubsetI*)
fix *b* **assume** *b*: $b \in_{\circ} B$
then have *b-def*: $b = vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor)(\lfloor \langle a, b \rangle \rfloor)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *V-cs-intros*)
from *b assms* **show** $b \in_{\circ} \mathcal{R}_{\circ}\ (vsnd\text{-}arrow\ (set\ \{a\})\ B(\lfloor ArrVal \rfloor))$
by (*subst b-def*, *use nothing* **in** ‹*intro vsv.vsv-vimageI2*›)
(*auto simp*: *cat-cs-simps cat-cs-intros*)
qed
qed (*rule vsnd-arrow-vrange*)
qed (*use assms* **in** *auto*)

lemma (**in** $\mathcal{Z}$) *vsnd-arrow-is-cat-Set-iso-arr*:
assumes $a \in_{\circ} cat\text{-}Set\ \alpha(\lfloor Obj \rfloor)$ **and** $B \in_{\circ} cat\text{-}Set\ \alpha(\lfloor Obj \rfloor)$
shows $vsnd\text{-}arrow\ (set\ \{a\})\ B : set\ \{a\} \times_{\circ} B \mapsto_{iso cat\text{-}Set\ \alpha} B$
using *assms*
unfolding *cat-Set-components*
by (*rule vsnd-arrow-is-cat-Set-iso-arr-Vset*)

lemma (in $\mathcal{Z}$**)** *vsnd-arrow-is-cat-Set-iso-arr'*[*cat-rel-par-Set-cs-intros*]:
assumes $a \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$
and $B \in_{\circ} cat\text{-}Set\ \alpha(\!|Obj|\!)$
and $AB = set\ \{a\} \times_{\circ} B$
and $A' = A$
and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
shows $vsnd\text{-}arrow\ (set\ \{a\})\ B : AB \mapsto_{iso\mathfrak{C}'} B$
using *assms*(*1–2*)
unfolding *assms*(*3–5*)
by (*rule vsnd-arrow-is-cat-Set-iso-arr*)

lemmas [*cat-rel-par-Set-cs-intros*] = $\mathcal{Z}$*.vsnd-arrow-is-cat-Set-iso-arr'*

17.8.7 Projection arrows are isomorphisms in the category *Par*

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Par-iso-arr*:
assumes $A \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$ **and** $b \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$
shows $vfst\text{-}arrow\ A\ (set\ \{b\}) : A \times_{\circ} set\ \{b\} \mapsto_{iso cat\text{-}Par\ \alpha} A$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
show $vfst\text{-}arrow\ A\ (set\ \{b\}) : A \times_{\circ} set\ \{b\} \mapsto_{iso cat\text{-}Par\ \alpha} A$
by
(
rule Set-Par.wr-subcat-is-iso-arr-is-iso-arr
[
THEN iffD1,
OF vfst-arrow-is-cat-Set-iso-arr-Vset[
OF assms[*unfolded cat-Par-components*]
]
]
)
qed

lemma (in $\mathcal{Z}$**)** *vfst-arrow-is-cat-Par-iso-arr'*[*cat-rel-Par-set-cs-intros*]:
assumes $A \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$
and $b \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$
and $AB = A \times_{\circ} set\ \{b\}$
and $A' = A$
and $\mathfrak{C}' = cat\text{-}Par\ \alpha$
shows $vfst\text{-}arrow\ A\ (set\ \{b\}) : AB \mapsto_{iso\mathfrak{C}'} A$
using *assms*(*1–2*)
unfolding *assms*(*3–5*)
by (*rule vfst-arrow-is-cat-Par-iso-arr*)

lemmas [*cat-rel-Par-set-cs-intros*] = $\mathcal{Z}$*.vfst-arrow-is-cat-Par-iso-arr'*

lemma (in $\mathcal{Z}$**)** *vsnd-arrow-is-cat-Par-iso-arr*:
assumes $a \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$ **and** $B \in_{\circ} cat\text{-}Par\ \alpha(\!|Obj|\!)$
shows $vsnd\text{-}arrow\ (set\ \{a\})\ B : set\ \{a\} \times_{\circ} B \mapsto_{iso cat\text{-}Par\ \alpha} B$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
show $vsnd\text{-}arrow\ (set\ \{a\})\ B : set\ \{a\} \times_{\circ} B \mapsto_{iso cat\text{-}Par\ \alpha} B$
by
(
rule Set-Par.wr-subcat-is-iso-arr-is-iso-arr

```
      [
        THEN iffD1,
        OF vsnd-arrow-is-cat-Set-iso-arr-Vset[
          OF assms[unfolded cat-Par-components]
          ]
      ]
    )
qed

lemma (in 𝒵) vsnd-arrow-is-cat-Par-iso-arr'[cat-rel-Par-set-cs-intros]:
  assumes a ∈∘ cat-Par α(|Obj|)
    and B ∈∘ cat-Par α(|Obj|)
    and AB = set {a} ×∘ B
    and A' = A
    and 𝔈' = cat-Par α
  shows vsnd-arrow (set {a}) B : AB ↦iso 𝔈' B
  using assms(1–2)
  unfolding assms(3–5)
  by (rule vsnd-arrow-is-cat-Par-iso-arr)

lemmas [cat-rel-Par-set-cs-intros] = 𝒵.vsnd-arrow-is-cat-Par-iso-arr'
```

17.8.8 Projection arrows are isomorphisms in the category *Rel*

```
lemma (in 𝒵) vfst-arrow-is-cat-Rel-iso-arr:
  assumes A ∈∘ cat-Rel α(|Obj|) and b ∈∘ cat-Rel α(|Obj|)
  shows vfst-arrow A (set {b}) : A ×∘ set {b} ↦iso cat-Rel α A
proof–
  interpret Set-Par: wide-replete-subcategory α ‹cat-Set α› ‹cat-Par α›
    by (rule wide-replete-subcategory-cat-Set-cat-Par)
  interpret Par-Rel: wide-replete-subcategory α ‹cat-Par α› ‹cat-Rel α›
    by (rule wide-replete-subcategory-cat-Par-cat-Rel)
  interpret Set-Rel: wide-replete-subcategory α ‹cat-Set α› ‹cat-Rel α›
    by
      (
        rule wr-subcat-trans
          [
            OF
              Set-Par.wide-replete-subcategory-axioms
              Par-Rel.wide-replete-subcategory-axioms
          ]
      )
  show ?thesis
    by
      (
        rule Set-Rel.wr-subcat-is-iso-arr-is-iso-arr
          [
            THEN iffD1,
            OF vfst-arrow-is-cat-Set-iso-arr-Vset[
              OF assms[unfolded cat-Rel-components]
              ]
          ]
      )
qed

lemma (in 𝒵) vfst-arrow-is-cat-Rel-iso-arr'[cat-Rel-par-set-cs-intros]:
  assumes A ∈∘ cat-Rel α(|Obj|)
    and b ∈∘ cat-Rel α(|Obj|)
```

```
    and AB = A ×∘ set {b}
    and A′ = A
    and 𝔈′ = cat-Rel α
  shows vfst-arrow A (set {b}) : AB ↦iso𝔈′ A
  using assms(1–2)
  unfolding assms(3–5)
  by (rule vfst-arrow-is-cat-Rel-iso-arr)

lemmas [cat-Rel-par-set-cs-intros] = 𝒵.vfst-arrow-is-cat-Rel-iso-arr′

lemma (in 𝒵) vsnd-arrow-is-cat-Rel-iso-arr:
  assumes a ∈∘ cat-Rel α(|Obj|) and B ∈∘ cat-Rel α(|Obj|)
  shows vsnd-arrow (set {a}) B : set {a} ×∘ B ↦iso cat-Rel α B
proof–
  interpret Set-Par: wide-replete-subcategory α ‹cat-Set α› ‹cat-Par α›
    by (rule wide-replete-subcategory-cat-Set-cat-Par)
  interpret Par-Rel: wide-replete-subcategory α ‹cat-Par α› ‹cat-Rel α›
    by (rule wide-replete-subcategory-cat-Par-cat-Rel)
  interpret Set-Rel: wide-replete-subcategory α ‹cat-Set α› ‹cat-Rel α›
    by
      (
        rule wr-subcat-trans
          [
            OF
              Set-Par.wide-replete-subcategory-axioms
              Par-Rel.wide-replete-subcategory-axioms
          ]
      )
  show ?thesis
    by
      (
        rule Set-Rel.wr-subcat-is-iso-arr-is-iso-arr
          [
            THEN iffD1,
            OF vsnd-arrow-is-cat-Set-iso-arr-Vset[
              OF assms[unfolded cat-Rel-components]
              ]
          ]
      )
qed

lemma (in 𝒵) vsnd-arrow-is-cat-Rel-iso-arr′[cat-Rel-par-set-cs-intros]:
  assumes a ∈∘ cat-Rel α(|Obj|)
    and B ∈∘ cat-Rel α(|Obj|)
    and AB = set {a} ×∘ B
    and A′ = A
    and 𝔈′ = cat-Rel α
  shows vsnd-arrow (set {a}) B : AB ↦iso𝔈′ B
  using assms(1–2)
  unfolding assms(3–5)
  by (rule vsnd-arrow-is-cat-Rel-iso-arr)

lemmas [cat-Rel-par-set-cs-intros] = 𝒵.vsnd-arrow-is-cat-Rel-iso-arr′
```

17.9 Projection arrow for *vproduct*

```
definition vprojection-arrow :: V ⇒ (V ⇒ V) ⇒ V ⇒ V
  where vprojection-arrow I A i = [vprojection I A i, (∏∘i∈∘I. A i), A i]∘
```

Components.

lemma *vprojection-arrow-components*:
shows *vprojection-arrow I A i*(|*ArrVal*|) = *vprojection I A i*
and *vprojection-arrow I A i*(|*ArrDom*|) = (∏ₒ *i*∈ₒ*I*. *A i*)
and *vprojection-arrow I A i*(|*ArrCod*|) = *A i*
unfolding *vprojection-arrow-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

17.9.1 Projection arrow value

mk-VLambda *vprojection-arrow-components*(*1*)[*unfolded vprojection-def*]
|*vsv vprojection-arrow-ArrVal-vsv*[*cat-Set-cs-intros*]|
|*vdomain vprojection-arrow-ArrVal-vdomain*[*cat-Set-cs-simps*]|
|*app vprojection-arrow-ArrVal-app*[*cat-Set-cs-simps*]|

17.9.2 Projection arrow is an arrow in the category *Set*

lemma (**in** 𝒵) *arr-Set-vprojection-arrow*:
assumes *i* ∈ₒ *I* **and** *VLambda I A* ∈ₒ *Vset α*
shows *arr-Set α* (*vprojection-arrow I A i*)
proof(*intro arr-SetI*)
show *vfsequence* (*vprojection-arrow I A i*)
unfolding *vprojection-arrow-def* **by** *auto*
show *vcard* (*vprojection-arrow I A i*) = $3_{\mathbb{N}}$
unfolding *vprojection-arrow-def* **by** (*simp add*: *nat-omega-simps*)
show *vprojection-arrow I A i*(|*ArrCod*|) ∈ₒ *Vset α*
unfolding *vprojection-arrow-components*
proof–
from *assms*(*1*) **have** *i* ∈ₒ *I* **by** *simp*
then have *A i* ∈ₒ ℛₒ (*VLambda I A*) **by** *auto*
moreover from *assms*(*2*) **have** ℛₒ (*VLambda I A*) ∈ₒ *Vset α*
by (*meson vrange-in-VsetI*)
ultimately show *A i* ∈ₒ *Vset α* **by** *auto*
qed
qed
(
auto
simp: *vprojection-arrow-components*
intro!:
assms
vprojection-vrange-vsubset
Limit-vproduct-in-Vset-if-VLambda-in-VsetI
)

lemma (**in** 𝒵) *vprojection-arrow-is-arr*:
assumes *i* ∈ₒ *I* **and** *VLambda I A* ∈ₒ *Vset α*
shows *vprojection-arrow I A i* : (∏ₒ *i*∈ₒ*I*. *A i*) ↦$_{\textit{cat-Set}\ \alpha}$ *A i*
proof(*intro cat-Set-is-arrI*)
from *assms* **show** *arr-Set α* (*vprojection-arrow I A i*)
by (*rule arr-Set-vprojection-arrow*)
qed (*simp-all add*: *vprojection-arrow-components*)

17.10 Canonical injection arrow for *vdunion*

definition *vcinjection-arrow* :: *V* ⇒ (*V* ⇒ *V*) ⇒ *V* ⇒ *V*
where *vcinjection-arrow I A i* = [*vcinjection A i*, *A i*, (∐ₒ *i*∈ₒ*I*. *A i*)]ₒ

Components.

lemma *vcinjection-arrow-components*:
shows *vcinjection-arrow I A i*(|*ArrVal*|) = *vcinjection A i*
and *vcinjection-arrow I A i*(|*ArrDom*|) = *A i*
and *vcinjection-arrow I A i*(|*ArrCod*|) = (∐∘ *i*∈∘ *I*. *A i*)
unfolding *vcinjection-arrow-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

17.10.1 Canonical injection arrow value

mk-VLambda *vcinjection-arrow-components*(*1*)[*unfolded vcinjection-def*]
|*vsv vcinjection-arrow-ArrVal-vsv*[*cat-Set-cs-intros*]|
|*vdomain vcinjection-arrow-ArrVal-vdomain*[*cat-Set-cs-simps*]|
|*app vcinjection-arrow-ArrVal-app*[*cat-Set-cs-simps*]|

17.10.2 Canonical injection arrow is an arrow in the category *Set*

lemma (**in** $\mathcal{Z}$) *arr-Set-vcinjection-arrow*:
assumes *i* ∈∘ *I* **and** *VLambda I A* ∈∘ *Vset α*
shows *arr-Set α* (*vcinjection-arrow I A i*)
proof(*intro arr-SetI*)
show *vfsequence* (*vcinjection-arrow I A i*)
unfolding *vcinjection-arrow-def* **by** *auto*
show *vcard* (*vcinjection-arrow I A i*) = $\mathfrak{Z}_{\mathbb{N}}$
unfolding *vcinjection-arrow-def* **by** (*simp add*: *nat-omega-simps*)
show *vcinjection-arrow I A i*(|*ArrDom*|) ∈∘ *Vset α*
unfolding *vcinjection-arrow-components*
proof–
from *assms*(*1*) **have** *Ai-def*: *A i* = *VLambda I A*(|*i*|) **by** *simp*
with *assms*(*1*) **have** *A i* ∈∘ $\mathcal{R}_\circ$ (*VLambda I A*) **by** *auto*
with *assms*(*2*) *Limit-α* **show** *A i* ∈∘ *Vset α*
unfolding *Ai-def* **by** (*auto intro*: *vrange-in-VsetI*)
qed
show *vcinjection-arrow I A i*(|*ArrCod*|) ∈∘ *Vset α*
unfolding *vcinjection-arrow-components*
by (*intro Limit-vdunion-in-Vset-if-VLambda-in-VsetI Limit-α assms*)
qed
(
auto
simp: *vcinjection-arrow-components*
intro!: *assms vcinjection-vrange-vsubset*
)

lemma (**in** $\mathcal{Z}$) *vcinjection-arrow-is-arr*:
assumes *i* ∈∘ *I* **and** *VLambda I A* ∈∘ *Vset α*
shows *vcinjection-arrow I A i* : *A i* ↦*cat-Set α* (∐∘ *i*∈∘ *I*. *A i*)
proof(*intro cat-Set-is-arrI*)
from *assms* **show** *arr-Set α* (*vcinjection-arrow I A i*)
by (*rule arr-Set-vcinjection-arrow*)
qed (*simp-all add*: *vcinjection-arrow-components*)

lemma (**in** $\mathcal{Z}$) *vcinjection-arrow-is-arr'*[*cat-cs-intros*]:
assumes *i* ∈∘ *I*
and *VLambda I A* ∈∘ *Vset α*
and *A'* = *A i*
and $\mathfrak{C}'$ = *cat-Set α*
and *P'* = (∐∘ *i*∈∘ *I*. *A i*)
shows *vcinjection-arrow I A i* : *A'* ↦$\mathfrak{C}'$ *P'*
using *assms*(*1,2*) **unfolding** *assms*(*3–5*) **by** (*rule vcinjection-arrow-is-arr*)

17.11 Product arrow value for *Rel*

17.11.1 Definition and elementary properties

definition *prod-2-Rel-ArrVal* :: $V \Rightarrow V \Rightarrow V$
  where *prod-2-Rel-ArrVal S T* =
    set $\{\langle\langle a,\ b\rangle,\ \langle c,\ d\rangle\rangle \mid a\ b\ c\ d.\ \langle a,\ c\rangle \in_\circ S \wedge \langle b,\ d\rangle \in_\circ T\}$

lemma *small-prod-2-Rel-ArrVal*[*simp*]:
  small $\{\langle\langle a,\ b\rangle,\ \langle c,\ d\rangle\rangle \mid a\ b\ c\ d.\ \langle a,\ c\rangle \in_\circ S \wedge \langle b,\ d\rangle \in_\circ T\}$
  (**is** ‹*small ?S*›)
proof(*rule down*)
  show *?S* $\subseteq$ *elts* $((\mathcal{D}_\circ\ S \times_\circ \mathcal{D}_\circ\ T) \times_\circ (\mathcal{R}_\circ\ S \times_\circ \mathcal{R}_\circ\ T))$ **by** *auto*
qed

Rules.

lemma *prod-2-Rel-ArrValI*:
  assumes *ab-cd* = $\langle\langle a,\ b\rangle,\ \langle c,\ d\rangle\rangle$
    and $\langle a,\ c\rangle \in_\circ S$
    and $\langle b,\ d\rangle \in_\circ T$
  shows *ab-cd* $\in_\circ$ *prod-2-Rel-ArrVal S T*
  using *assms* **unfolding** *prod-2-Rel-ArrVal-def* **by** *simp*

lemma *prod-2-Rel-ArrValD*[*dest*]:
  assumes $\langle\langle a,\ b\rangle,\ \langle c,\ d\rangle\rangle \in_\circ$ *prod-2-Rel-ArrVal S T*
  shows $\langle a,\ c\rangle \in_\circ S$ **and** $\langle b,\ d\rangle \in_\circ T$
  using *assms* **unfolding** *prod-2-Rel-ArrVal-def* **by** *auto*

lemma *prod-2-Rel-ArrValE*[*elim*!]:
  assumes *ab-cd* $\in_\circ$ *prod-2-Rel-ArrVal S T*
  obtains *a b c d* **where** *ab-cd* = $\langle\langle a,\ b\rangle,\ \langle c,\ d\rangle\rangle$
    and $\langle a,\ c\rangle \in_\circ S$
    and $\langle b,\ d\rangle \in_\circ T$
  using *assms* **unfolding** *prod-2-Rel-ArrVal-def* **by** *auto*

Elementary properties

lemma *prod-2-Rel-ArrVal-vsubset-vprod*:
  prod-2-Rel-ArrVal S T $\subseteq_\circ$ $((\mathcal{D}_\circ\ S \times_\circ \mathcal{D}_\circ\ T) \times_\circ (\mathcal{R}_\circ\ S \times_\circ \mathcal{R}_\circ\ T))$
  by (*intro vsubsetI*) *auto*

lemma *prod-2-Rel-ArrVal-vbrelation*: *vbrelation* (*prod-2-Rel-ArrVal S T*)
  using *prod-2-Rel-ArrVal-vsubset-vprod* **by** *auto*

lemma *prod-2-Rel-ArrVal-vdomain*: $\mathcal{D}_\circ$ (*prod-2-Rel-ArrVal S T*) = $\mathcal{D}_\circ\ S \times_\circ \mathcal{D}_\circ\ T$
proof(*intro vsubset-antisym*)
  show $\mathcal{D}_\circ\ S \times_\circ \mathcal{D}_\circ\ T \subseteq_\circ \mathcal{D}_\circ$ (*prod-2-Rel-ArrVal S T*)
  proof(*intro vsubsetI*)
    fix *ab* **assume** *ab* $\in_\circ \mathcal{D}_\circ\ S \times_\circ \mathcal{D}_\circ\ T$
    then obtain *a b*
      where *ab-def*: *ab* = $\langle a,\ b\rangle$
        and $a \in_\circ \mathcal{D}_\circ\ S$
        and $b \in_\circ \mathcal{D}_\circ\ T$
      by *auto*
    then obtain *c d* **where** $\langle a,\ c\rangle \in_\circ S$ **and** $\langle b,\ d\rangle \in_\circ T$ **by** *force*
    then have $\langle\langle a,\ b\rangle,\ \langle c,\ d\rangle\rangle \in_\circ$ *prod-2-Rel-ArrVal S T*
      by (*intro prod-2-Rel-ArrValI*) *auto*
    then show *ab* $\in_\circ \mathcal{D}_\circ$ (*prod-2-Rel-ArrVal S T*)
      unfolding *ab-def* **by** (*simp add*: *app-vdomainI*)
  qed

qed (*use prod-2-Rel-ArrVal-vsubset-vprod* **in** *blast*)

lemma *prod-2-Rel-ArrVal-vrange*: $\mathcal{R}_\circ\ (\mathit{prod\text{-}2\text{-}Rel\text{-}ArrVal}\ S\ T) = \mathcal{R}_\circ\ S \times_\circ \mathcal{R}_\circ\ T$
proof(*intro vsubset-antisym*)
show $\mathcal{R}_\circ\ S \times_\circ \mathcal{R}_\circ\ T \subseteq_\circ \mathcal{R}_\circ\ (\mathit{prod\text{-}2\text{-}Rel\text{-}ArrVal}\ S\ T)$
proof(*intro vsubsetI*)
fix *cd* **assume** $cd \in_\circ \mathcal{R}_\circ\ S \times_\circ \mathcal{R}_\circ\ T$
then obtain *c d*
where *cd-def*: $cd = \langle c, d\rangle$
and $c \in_\circ \mathcal{R}_\circ\ S$
and $d \in_\circ \mathcal{R}_\circ\ T$
by *auto*
then obtain *a b* **where** $\langle a, c\rangle \in_\circ S$ **and** $\langle b, d\rangle \in_\circ T$ **by** *force*
then have $\langle\langle a, b\rangle, \langle c, d\rangle\rangle \in_\circ \mathit{prod\text{-}2\text{-}Rel\text{-}ArrVal}\ S\ T$
by (*intro prod-2-Rel-ArrValI*) *auto*
then show $cd \in_\circ \mathcal{R}_\circ\ (\mathit{prod\text{-}2\text{-}Rel\text{-}ArrVal}\ S\ T)$
unfolding *cd-def* **by** (*simp add*: *app-vrangeI*)
qed
qed (*use prod-2-Rel-ArrVal-vsubset-vprod* **in** *blast*)

17.11.2 Further properties

lemma
assumes *vsv g* **and** *vsv f*
shows *prod-2-Rel-ArrVal-vsv*: *vsv* (*prod-2-Rel-ArrVal g f*)
and *prod-2-Rel-ArrVal-app*:
$\bigwedge a\ b.\ [\![\ a \in_\circ \mathcal{D}_\circ\ g;\ b \in_\circ \mathcal{D}_\circ\ f\ ]\!] \Longrightarrow$
prod-2-Rel-ArrVal g f($\lfloor\langle a,b\rangle\rfloor$) = $\langle g(\lfloor a\rfloor), f(\lfloor b\rfloor)\rangle$
proof–
interpret *g*: *vsv g* **by** (*rule assms*(*1*))
interpret *f*: *vsv f* **by** (*rule assms*(*2*))
show *vsv-gf*: *vsv* (*prod-2-Rel-ArrVal g f*)
by (*intro vsvI*; (*elim prod-2-Rel-ArrValE*)*?*; (*unfold prod-2-Rel-ArrVal-def*)*?*)
(*auto simp*: *g.vsv f.vsv*)
fix *a b* **assume** $a \in_\circ \mathcal{D}_\circ\ g\ b \in_\circ \mathcal{D}_\circ\ f$
then have *a-ga*: $\langle a, g(\lfloor a\rfloor)\rangle \in_\circ g$ **and** *b-fb*: $\langle b, f(\lfloor b\rfloor)\rangle \in_\circ f$ **by** *auto*
from *a-ga b-fb* **show** *prod-2-Rel-ArrVal g f*($\lfloor\langle a, b\rangle\rfloor$) = $\langle g(\lfloor a\rfloor), f(\lfloor b\rfloor)\rangle$
by
(
cs-concl **cs-shallow**
cs-simp: *vsv.vsv-appI*[*OF vsv-gf*] **cs-intro**: *prod-2-Rel-ArrValI*
)
qed

lemma *prod-2-Rel-ArrVal-v11*:
assumes *v11 g* **and** *v11 f*
shows *v11* (*prod-2-Rel-ArrVal g f*)
proof–
interpret *g*: *v11 g* **by** (*rule assms*(*1*))
interpret *f*: *v11 f* **by** (*rule assms*(*2*))
show *?thesis*
proof
(
intro vsv.vsv-valeq-v11I prod-2-Rel-ArrVal-vsv g.vsv-axioms f.vsv-axioms,
unfold prod-2-Rel-ArrVal-vdomain
)
fix *ab cd*
assume *prems*:

```
    ab ∈∘ 𝒟∘ g ×∘ 𝒟∘ f
    cd ∈∘ 𝒟∘ g ×∘ 𝒟∘ f
    prod-2-Rel-ArrVal g f(|ab|) = prod-2-Rel-ArrVal g f(|cd|)
  from prems(1) obtain a b
    where ab-def: ab = ⟨a, b⟩ and a: a ∈∘ 𝒟∘ g and b: b ∈∘ 𝒟∘ f
    by auto
  from prems(2) obtain c d
    where cd-def: cd = ⟨c, d⟩ and c: c ∈∘ 𝒟∘ g and d: d ∈∘ 𝒟∘ f
    by auto
  from prems(3) a b c d have ⟨g(|a|), f(|b|)⟩ = ⟨g(|c|), f(|d|)⟩
    unfolding ab-def cd-def
    by
      (
        cs-prems cs-shallow
          cs-simp: prod-2-Rel-ArrVal-app cs-intro: V-cs-intros
      )
  then have g(|a|) = g(|c|) and f(|b|) = f(|d|) by simp-all
  then show ab = cd
    by (auto simp: ab-def cd-def a b c d f.v11-injective g.v11-injective)
 qed
qed

lemma prod-2-Rel-ArrVal-vcomp:
  prod-2-Rel-ArrVal S' T' ∘∘ prod-2-Rel-ArrVal S T =
    prod-2-Rel-ArrVal (S' ∘∘ S) (T' ∘∘ T)
proof–
  interpret ST': vbrelation ‹prod-2-Rel-ArrVal S' T'›
    by (rule prod-2-Rel-ArrVal-vbrelation)
  interpret ST: vbrelation ‹prod-2-Rel-ArrVal S T›
    by (rule prod-2-Rel-ArrVal-vbrelation)
  show ?thesis
  proof(intro vsubset-antisym vsubsetI)
    fix aa'-cc' assume
      aa'-cc' ∈∘ prod-2-Rel-ArrVal S' T' ∘∘ prod-2-Rel-ArrVal S T
    then obtain aa' bb' cc' where ac-def: aa'-cc' = ⟨aa', cc'⟩
      and bc: ⟨bb', cc'⟩ ∈∘ prod-2-Rel-ArrVal S' T'
      and ab: ⟨aa', bb'⟩ ∈∘ prod-2-Rel-ArrVal S T
      by (elim vcompE)
    from bc obtain b b' c c'
      where bb'-cc'-def: ⟨bb', cc'⟩ = ⟨⟨b, b'⟩, ⟨c, c'⟩⟩
        and bc: ⟨b, c⟩ ∈∘ S'
        and bc': ⟨b', c'⟩ ∈∘ T'
      by auto
    with ab obtain a a'
      where aa'-bb'-def: ⟨aa', bb'⟩ = ⟨⟨a, a'⟩, ⟨b, b'⟩⟩
        and ab: ⟨a, b⟩ ∈∘ S
        and ab': ⟨a', b'⟩ ∈∘ T
      by auto
    from bb'-cc'-def have bb'-def: bb' = ⟨b, b'⟩ and cc'-def: cc' = ⟨c, c'⟩
      by simp-all
    from aa'-bb'-def have aa'-def: aa' = ⟨a, a'⟩ and bb'-def: bb' = ⟨b, b'⟩
      by simp-all
    from bc bc' ab ab' show aa'-cc' ∈∘ prod-2-Rel-ArrVal (S' ∘∘ S) (T' ∘∘ T)
      unfolding ac-def aa'-def cc'-def
      by (intro prod-2-Rel-ArrValI)
        (cs-concl cs-shallow cs-intro: prod-2-Rel-ArrValI vcompI)+
  next
    fix aa'-cc' assume aa'-cc' ∈∘ prod-2-Rel-ArrVal (S' ∘∘ S) (T' ∘∘ T)
```

```
    then obtain a a' c c'
      where aa'-cc'-def: aa'-cc' = ⟨⟨a, a'⟩, ⟨c, c'⟩⟩
        and ac: ⟨a, c⟩ ∈∘ S' ∘∘ S
        and ac': ⟨a', c'⟩ ∈∘ T' ∘∘ T
      by blast
    from ac obtain b where ab: ⟨a, b⟩ ∈∘ S and bc: ⟨b, c⟩ ∈∘ S'
      by auto
    from ac' obtain b' where ab': ⟨a', b'⟩ ∈∘ T and bc': ⟨b', c'⟩ ∈∘ T'
      by auto
    from ab bc ab' bc' show
      aa'-cc' ∈∘ prod-2-Rel-ArrVal S' T' ∘∘ prod-2-Rel-ArrVal S T
      unfolding aa'-cc'-def
      by (cs-concl cs-shallow cs-intro: vcompI prod-2-Rel-ArrValI)
  qed
qed

lemma prod-2-Rel-ArrVal-vid-on[cat-cs-simps]:
  prod-2-Rel-ArrVal (vid-on A) (vid-on B) = vid-on (A ×∘ B)
  unfolding prod-2-Rel-ArrVal-def by auto
```

17.12 Product arrow for *Rel*

17.12.1 Definition and elementary properties

```
definition prod-2-Rel :: V ⇒ V ⇒ V (infixr ‹_A×_Rel› 80)
  where prod-2-Rel S T =
    [
      prod-2-Rel-ArrVal (S(|ArrVal|)) (T(|ArrVal|)),
      S(|ArrDom|) ×∘ T(|ArrDom|),
      S(|ArrCod|) ×∘ T(|ArrCod|)
    ]∘

abbreviation (input) prod-2-Par :: V ⇒ V ⇒ V (infixr ‹_A×_Par› 80)
  where prod-2-Par ≡ prod-2-Rel
abbreviation (input) prod-2-Set :: V ⇒ V ⇒ V (infixr ‹_A×_Set› 80)
  where prod-2-Set ≡ prod-2-Rel
```

Components.

```
lemma prod-2-Rel-components:
  shows (S _A×_Rel T)(|ArrVal|) = prod-2-Rel-ArrVal (S(|ArrVal|)) (T(|ArrVal|))
    and [cat-cs-simps]: (S _A×_Rel T)(|ArrDom|) = S(|ArrDom|) ×∘ T(|ArrDom|)
    and [cat-cs-simps]: (S _A×_Rel T)(|ArrCod|) = S(|ArrCod|) ×∘ T(|ArrCod|)
  unfolding prod-2-Rel-def arr-field-simps by (simp-all add: nat-omega-simps)
```

17.12.2 Product arrow for *Rel* is an arrow in *Rel*

```
lemma prod-2-Rel-is-cat-Rel-arr:
  assumes S : A ↦_cat-Rel α B and T : C ↦_cat-Rel α D
  shows S _A×_Rel T : A ×∘ C ↦_cat-Rel α B ×∘ D
proof–
  note S = cat-Rel-is-arrD[OF assms(1)]
  note T = cat-Rel-is-arrD[OF assms(2)]
  interpret S: arr-Rel α S
    rewrites [simp]: S(|ArrDom|) = A and [simp]: S(|ArrCod|) = B
    by (simp-all add: S)
  interpret T: arr-Rel α T
    rewrites [simp]: T(|ArrDom|) = C and [simp]: T(|ArrCod|) = D
    by (simp-all add: T)
  show ?thesis
```

proof(*intro cat-Rel-is-arrI arr-RelI*)
show *vfsequence* (S $_A\times_{Rel}$ T)
unfolding *prod-2-Rel-def* **by** *simp*
show *vcard* (S $_A\times_{Rel}$ T) = $3_{\mathbb{N}}$
unfolding *prod-2-Rel-def* **by** (*simp add*: *nat-omega-simps*)
from S **have** $\mathcal{D}_\circ$ ($S(\!|ArrVal|\!)$) $\subseteq_\circ$ A **and** $\mathcal{R}_\circ$ ($S(\!|ArrVal|\!)$) $\subseteq_\circ$ B **by** *auto*
moreover from T **have** $\mathcal{D}_\circ$ ($T(\!|ArrVal|\!)$) $\subseteq_\circ$ C **and** $\mathcal{R}_\circ$ ($T(\!|ArrVal|\!)$) $\subseteq_\circ$ D
by *auto*
ultimately have
$\mathcal{D}_\circ$ ($S(\!|ArrVal|\!)$) $\times_\circ$ $\mathcal{D}_\circ$ ($T(\!|ArrVal|\!)$) $\subseteq_\circ$ A $\times_\circ$ C
$\mathcal{R}_\circ$ ($S(\!|ArrVal|\!)$) $\times_\circ$ $\mathcal{R}_\circ$ ($T(\!|ArrVal|\!)$) $\subseteq_\circ$ B $\times_\circ$ D
by *auto*
then show
$\mathcal{D}_\circ$ ((S $_A\times_{Rel}$ T)$(\!|ArrVal|\!)$) $\subseteq_\circ$ (S $_A\times_{Rel}$ T)$(\!|ArrDom|\!)$
$\mathcal{R}_\circ$ ((S $_A\times_{Rel}$ T)$(\!|ArrVal|\!)$) $\subseteq_\circ$ (S $_A\times_{Rel}$ T)$(\!|ArrCod|\!)$
unfolding
prod-2-Rel-components prod-2-Rel-ArrVal-vdomain prod-2-Rel-ArrVal-vrange
by (*force simp*: *prod-2-Rel-components*)+
from
S.arr-Rel-ArrDom-in-Vset T.arr-Rel-ArrDom-in-Vset
S.arr-Rel-ArrCod-in-Vset T.arr-Rel-ArrCod-in-Vset
show (S $_A\times_{Rel}$ T)$(\!|ArrDom|\!)$ $\in_\circ$ *Vset* α (S $_A\times_{Rel}$ T)$(\!|ArrCod|\!)$ $\in_\circ$ *Vset* α
unfolding *prod-2-Rel-components*
by (*all*‹*intro Limit-vtimes-in-VsetI*›) *auto*
qed (*auto simp*: *prod-2-Rel-components intro*: *prod-2-Rel-ArrVal-vbrelation*)
qed

lemma *prod-2-Rel-is-cat-Rel-arr'*[*cat-Rel-par-set-cs-intros*]:
assumes S : A $\mapsto_{cat\text{-}Rel\ \alpha}$ B
and T : C $\mapsto_{cat\text{-}Rel\ \alpha}$ D
and A' = A $\times_\circ$ C
and B' = B $\times_\circ$ D
and $\mathfrak{C}'$ = *cat-Rel* α
shows S $_A\times_{Rel}$ T : A' $\mapsto_{\mathfrak{C}'}$ B'
using *assms*(*1,2*) **unfolding** *assms*(*3–5*) **by** (*rule prod-2-Rel-is-cat-Rel-arr*)

17.12.3 Product arrow for *Rel* is an arrow in *Set*

lemma *prod-2-Rel-app*[*cat-rel-par-Set-cs-simps*]:
assumes S : A $\mapsto_{cat\text{-}Set\ \alpha}$ B
and T : C $\mapsto_{cat\text{-}Set\ \alpha}$ D
and a $\in_\circ$ A
and c $\in_\circ$ C
and ac = $\langle a, c\rangle$
shows (S $_A\times_{Set}$ T)$(\!|ArrVal|\!)(\!|ac|\!)$ = $\langle S(\!|ArrVal|\!)(\!|a|\!),\ T(\!|ArrVal|\!)(\!|c|\!)\rangle$
proof–
note S = *cat-Set-is-arrD*[*OF assms*(*1*)]
note T = *cat-Set-is-arrD*[*OF assms*(*2*)]
interpret S: *arr-Set* α S
rewrites [*simp*]: $S(\!|ArrDom|\!)$ = A **and** [*simp*]: $S(\!|ArrCod|\!)$ = B
by (*simp-all add*: S)
interpret T: *arr-Set* α T
rewrites [*simp*]: $T(\!|ArrDom|\!)$ = C **and** [*simp*]: $T(\!|ArrCod|\!)$ = D
by (*simp-all add*: T)
from *assms*(*3,4*) **show** *?thesis*
unfolding *prod-2-Rel-components*(*1*) *assms*(*5*)
by
(

```
      cs-concl cs-shallow
        cs-simp:
          S.arr-Set-ArrVal-vdomain
          T.arr-Set-ArrVal-vdomain
          prod-2-Rel-ArrVal-app
        cs-intro: V-cs-intros
    )
qed

lemma prod-2-Rel-is-cat-Set-arr:
  assumes S : A ↦_cat-Set α B and T : C ↦_cat-Set α D
  shows S ₐ×_Set T : A ×_∘ C ↦_cat-Set α B ×_∘ D
proof–

  note S = cat-Set-is-arrD[OF assms(1)]
  note T = cat-Set-is-arrD[OF assms(2)]

  interpret S: arr-Set α S
    rewrites [simp]: S(|ArrDom|) = A and [simp]: S(|ArrCod|) = B
    by (simp-all add: S)
  interpret T: arr-Set α T
    rewrites [simp]: T(|ArrDom|) = C and [simp]: T(|ArrCod|) = D
    by (simp-all add: T)

  show ?thesis
  proof(intro cat-Set-is-arrI arr-SetI)
    show vfsequence (S ₐ×_Set T)
      unfolding prod-2-Rel-def by simp
    show vcard (S ₐ×_Set T) = 3_ℕ
      unfolding prod-2-Rel-def by (simp add: nat-omega-simps)
    from S.arr-Set-ArrVal-vrange T.arr-Set-ArrVal-vrange show
      ℛ_∘ ((S ₐ×_Set T)(|ArrVal|)) ⊆_∘ (S ₐ×_Set T)(|ArrCod|)
      unfolding
        prod-2-Rel-components prod-2-Rel-ArrVal-vdomain prod-2-Rel-ArrVal-vrange
      by auto
    from assms S.arr-Par-ArrDom-in-Vset T.arr-Par-ArrDom-in-Vset show
      (S ₐ×_Set T)(|ArrDom|) ∈_∘ Vset α
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: V-cs-intros)
    from assms S.arr-Par-ArrCod-in-Vset T.arr-Par-ArrCod-in-Vset show
      (S ₐ×_Set T)(|ArrCod|) ∈_∘ Vset α
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: V-cs-intros)
    from assms show (S ₐ×_Set T)(|ArrDom|) = A ×_∘ C
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from assms show (S ₐ×_Set T)(|ArrCod|) = B ×_∘ D
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    show vsv ((S ₐ×_Set T)(|ArrVal|))
      unfolding prod-2-Rel-components
      by (intro prod-2-Rel-ArrVal-vsv S.ArrVal.vsv-axioms T.ArrVal.vsv-axioms)
  qed
    (
      auto simp:
        cat-cs-simps cat-Set-cs-simps
        prod-2-Rel-ArrVal-vdomain prod-2-Rel-components(1)
    )

qed

lemma prod-2-Rel-is-cat-Set-arr'[cat-rel-par-Set-cs-intros]:
```

assumes $S : A \mapsto_{cat\text{-}Set\ \alpha} B$
and $T : C \mapsto_{cat\text{-}Set\ \alpha} D$
and $AC = A \times_\circ C$
and $BD = B \times_\circ D$
and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
shows $S\ {}_A\times_{Set}\ T : AC \mapsto_{\mathfrak{C}'} BD$
using *assms*(*1*,*2*) **unfolding** *assms*(*3–5*) **by** (*rule prod-2-Rel-is-cat-Set-arr*)

17.12.4 Product arrow for *Rel* is an isomorphism in *Set*

lemma *prod-2-Rel-is-cat-Set-iso-arr*:
assumes $S : A \mapsto_{iso\,cat\text{-}Set\ \alpha} B$ **and** $T : C \mapsto_{iso\,cat\text{-}Set\ \alpha} D$
shows $S\ {}_A\times_{Set}\ T : A \times_\circ C \mapsto_{iso\,cat\text{-}Set\ \alpha} B \times_\circ D$
proof–
note $S = cat\text{-}Set\text{-}is\text{-}iso\text{-}arrD[OF\ assms(1)]$
note $T = cat\text{-}Set\text{-}is\text{-}iso\text{-}arrD[OF\ assms(2)]$
show *?thesis*
proof
(
intro cat-Set-is-iso-arrI prod-2-Rel-is-cat-Set-arr[*OF S*(*1*) *T*(*1*)],
unfold prod-2-Rel-components
)
show $\mathcal{D}_\circ\ (prod\text{-}2\text{-}Rel\text{-}ArrVal\ (S(\!|ArrVal|\!))\ (T(\!|ArrVal|\!))) = A \times_\circ C$
unfolding *prod-2-Rel-ArrVal-vdomain*
by (*cs-concl* **cs-shallow cs-simp**: *S*(*3*) *T*(*3*) **cs-intro**: *cat-cs-intros*)
show $\mathcal{R}_\circ\ (prod\text{-}2\text{-}Rel\text{-}ArrVal\ (S(\!|ArrVal|\!))\ (T(\!|ArrVal|\!))) = B \times_\circ D$
unfolding *prod-2-Rel-ArrVal-vrange*
by (*cs-concl* **cs-shallow cs-simp**: *S*(*4*) *T*(*4*) **cs-intro**: *cat-cs-intros*)
qed (*use S*(*2*) *T*(*2*) **in** ‹*cs-concl cs-shallow cs-intro*: *prod-2-Rel-ArrVal-v11*›)
qed

lemma *prod-2-Rel-is-cat-Set-iso-arr'*[*cat-rel-par-Set-cs-intros*]:
assumes $S : A \mapsto_{iso\,cat\text{-}Set\ \alpha} B$
and $T : C \mapsto_{iso\,cat\text{-}Set\ \alpha} D$
and $AC = A \times_\circ C$
and $BD = B \times_\circ D$
and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
shows $S\ {}_A\times_{Set}\ T : AC \mapsto_{iso\,\mathfrak{C}'} BD$
using *assms*(*1*,*2*)
unfolding *assms*(*3–5*)
by (*rule prod-2-Rel-is-cat-Set-iso-arr*)

17.12.5 Further elementary properties

lemma *prod-2-Rel-Comp*:
assumes $G' : B' \mapsto_{cat\text{-}Rel\ \alpha} B''$
and $F' : A' \mapsto_{cat\text{-}Rel\ \alpha} A''$
and $G : B \mapsto_{cat\text{-}Rel\ \alpha} B'$
and $F : A \mapsto_{cat\text{-}Rel\ \alpha} A'$
shows
$G'\ {}_A\times_{Rel}\ F' \circ_{A\,cat\text{-}Rel\ \alpha}\ G\ {}_A\times_{Rel}\ F =$
$(G' \circ_{A\,cat\text{-}Rel\ \alpha}\ G)\ {}_A\times_{Rel}\ (F' \circ_{A\,cat\text{-}Rel\ \alpha}\ F)$
proof–

from *cat-Rel-is-arrD*(*1*)[*OF assms*(*1*)] **interpret** $\mathcal{Z}\ \alpha$ **by** *auto*

interpret *Rel*: *category* α ‹*cat-Rel* α› **by** (*rule category-cat-Rel*)
note [*cat-cs-simps*] = *cat-Rel-is-arrD*(*2*,*3*)

```
from assms have GF'-GF:
  G' _{A×Rel} F' ∘_{A cat-Rel α} G _{A×Rel} F :
    B ×_∘ A ↦_{cat-Rel α} B'' ×_∘ A''
  by (cs-concl cs-shallow cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros)
from assms Rel.category-axioms have GG'-FF':
  (G' ∘_{A cat-Rel α} G) _{A×Rel} (F' ∘_{A cat-Rel α} F) :
    B ×_∘ A ↦_{cat-Rel α} B'' ×_∘ A''
  by (cs-concl cs-shallow cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros)

show ?thesis
proof(rule arr-Rel-eqI[of α])
  from GF'-GF show arr-Rel-GF'-GF:
    arr-Rel α (G' _{A×Rel} F' ∘_{A cat-Rel α} G _{A×Rel} F)
    by (auto dest: cat-Rel-is-arrD(1))
  from GG'-FF' show arr-Rel-GG'-FF':
    arr-Rel α ((G' ∘_{A cat-Rel α} G) _{A×Rel} (F' ∘_{A cat-Rel α} F))
    by (auto dest: cat-Rel-is-arrD(1))
  show (G' _{A×Rel} F' ∘_{A cat-Rel α} G _{A×Rel} F)(|ArrVal|) =
    ((G' ∘_{A cat-Rel α} G) _{A×Rel} (F' ∘_{A cat-Rel α} F))(|ArrVal|)
  proof(intro vsubset-antisym vsubsetI)
    fix R assume
      R ∈_∘ (G' _{A×Rel} F' ∘_{A cat-Rel α} G _{A×Rel} F)(|ArrVal|)
    from this assms have R ∈_∘
      prod-2-Rel-ArrVal (G'(|ArrVal|)) (F'(|ArrVal|)) ∘_∘
      prod-2-Rel-ArrVal (G(|ArrVal|)) (F(|ArrVal|))
      by
        (
          cs-prems cs-shallow
            cs-simp:
              prod-2-Rel-components(1)
              comp-Rel-components(1)
              cat-Rel-cs-simps
            cs-intro: cat-Rel-par-set-cs-intros
        )
    from this[unfolded prod-2-Rel-ArrVal-vcomp] assms show
      R ∈_∘ ((G' ∘_{A cat-Rel α} G) _{A×Rel} (F' ∘_{A cat-Rel α} F))(|ArrVal|)
      by
        (
          cs-concl cs-shallow cs-simp:
            prod-2-Rel-components comp-Rel-components(1) cat-Rel-cs-simps
        )
  next
    fix R assume
      R ∈_∘ ((G' ∘_{A cat-Rel α} G) _{A×Rel} (F' ∘_{A cat-Rel α} F))(|ArrVal|)
    from this assms have
      R ∈_∘ prod-2-Rel-ArrVal (G'(|ArrVal|) ∘_∘ G(|ArrVal|)) (F'(|ArrVal|) ∘_∘ F(|ArrVal|))
      by
        (
          cs-prems cs-shallow cs-simp:
            comp-Rel-components prod-2-Rel-components cat-Rel-cs-simps
        )
    from this[folded prod-2-Rel-ArrVal-vcomp] assms show
      R ∈_∘ ((G' _{A×Rel} F') ∘_{A cat-Rel α} (G _{A×Rel} F))(|ArrVal|)
      by
        (
          cs-concl cs-shallow
            cs-simp:
```

```
        prod-2-Rel-components comp-Rel-components(1) cat-Rel-cs-simps
      cs-intro: cat-Rel-par-set-cs-intros
    )
  qed

 qed
  (
   use GF'-GF assms in
    ‹
     cs-concl
      cs-simp: cat-cs-simps
      cs-intro: cat-cs-intros cat-Rel-cs-intros
    ›
  )+

qed

lemma (in 𝒵) prod-2-Rel-CId[cat-cs-simps]:
 assumes A ∈₀ cat-Rel α(|Obj|) and B ∈₀ cat-Rel α(|Obj|)
 shows
  (cat-Rel α(|CId|)(|A|)) ₐ×ᵣₑₗ (cat-Rel α(|CId|)(|B|)) = cat-Rel α(|CId|)(|A ×₀ B|)
proof–
 interpret Rel: category α ‹cat-Rel α› by (rule category-cat-Rel)
 from assms have A-B:
  (cat-Rel α(|CId|)(|A|)) ₐ×ᵣₑₗ (cat-Rel α(|CId|)(|B|)) :
   A ×₀ B ↦₍cat-Rel α₎ A ×₀ B
  by (cs-concl cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros)
 from assms Rel.category-axioms have AB:
  cat-Rel α(|CId|)(|A ×₀ B|) : A ×₀ B ↦₍cat-Rel α₎ A ×₀ B
  by
   (
    cs-concl
     cs-simp: cat-Rel-components(1) cs-intro: V-cs-intros cat-cs-intros
   )
 show ?thesis
 proof(rule arr-Rel-eqI)
  from A-B show arr-Rel-GF'-GF:
   arr-Rel α ((cat-Rel α(|CId|)(|A|)) ₐ×ᵣₑₗ (cat-Rel α(|CId|)(|B|)))
   by (auto dest: cat-Rel-is-arrD(1))
  from AB show arr-Rel-GG'-FF': arr-Rel α (cat-Rel α(|CId|)(|A ×₀ B|))
   by (auto dest: cat-Rel-is-arrD(1))
  from assms show
   ((cat-Rel α(|CId|)(|A|)) ₐ×ᵣₑₗ (cat-Rel α(|CId|)(|B|)))(|ArrVal|) =
    cat-Rel α(|CId|)(|A ×₀ B|)(|ArrVal|)
   by
    (
     cs-concl
      cs-simp:
       id-Rel-components prod-2-Rel-components
       cat-cs-simps cat-Rel-cs-simps
      cs-intro: V-cs-intros  cat-cs-intros
    )
 qed
  (
   use A-B assms in
    ‹
     cs-concl
      cs-simp: prod-2-Rel-components cat-Rel-cs-simps
```

```
        cs-intro: cat-cs-intros
      ›
  )+
qed

lemma cf-dag-Rel-ArrMap-app-prod-2-Rel:
  assumes S : A ↦_{cat-Rel α} B and T : C ↦_{cat-Rel α} D
  shows
    †_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖) =
      (†_{C.Rel} α(‖ArrMap‖)(‖S‖)) ×_{A Rel} (†_{C.Rel} α(‖ArrMap‖)(‖T‖))
proof–

  interpret S: arr-Rel α S by (intro cat-Rel-is-arrD[OF assms(1)])
  interpret Rel: category α ‹cat-Rel α› by (rule S.category-cat-Rel)
  interpret dag-Rel: is-iso-functor α ‹op-cat (cat-Rel α)› ‹cat-Rel α› ‹†_{C.Rel} α›
    by (rule S.cf-dag-Rel-is-iso-functor)

  note ST = prod-2-Rel-is-cat-Rel-arr[OF assms]

  from assms have dag-S: †_{C.Rel} α(‖ArrMap‖)(‖S‖) : B ↦_{cat-Rel α} A
    and dag-T: †_{C.Rel} α(‖ArrMap‖)(‖T‖) : D ↦_{cat-Rel α} C
    by
      (
        cs-concl
          cs-simp: cat-Rel-cs-simps cat-op-simps cs-intro: cat-cs-intros
      )+
  from assms have dag-prod:
    †_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖) : B ×∘ D ↦_{cat-Rel α} A ×∘ C
    by
      (
        cs-concl
          cs-simp: cat-Rel-cs-simps cat-op-simps
          cs-intro: V-cs-intros cat-cs-intros cat-Rel-par-set-cs-intros
      )
  from dag-S dag-T have prod-dag:
    (†_{C.Rel} α(‖ArrMap‖)(‖S‖)) ×_{A Rel} (†_{C.Rel} α(‖ArrMap‖)(‖T‖)) :
      B ×∘ D ↦_{cat-Rel α} A ×∘ C
    by (cs-concl cs-shallow cs-intro: cat-Rel-par-set-cs-intros)

  note [cat-cs-simps] =
    prod-2-Rel-ArrVal-vdomain prod-2-Rel-ArrVal-vrange prod-2-Rel-components
  from dag-prod ST have [cat-cs-simps]:
    𝒟∘ (†_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖)(‖ArrVal‖)) = ℛ∘ (S(‖ArrVal‖)) ×∘ ℛ∘ (T(‖ArrVal‖))
    ℛ∘ (†_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖)(‖ArrVal‖)) = 𝒟∘ (S(‖ArrVal‖)) ×∘ 𝒟∘ (T(‖ArrVal‖))
    by (cs-concl cs-simp: cat-cs-simps)+

 show
    †_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖) =
      (†_{C.Rel} α(‖ArrMap‖)(‖S‖)) ×_{A Rel} (†_{C.Rel} α(‖ArrMap‖)(‖T‖))
  proof(rule arr-Rel-eqI)
    from dag-prod show arr-Rel-dag-prod:
      arr-Rel α (†_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖))
      by (auto dest: cat-Rel-is-arrD)
    then interpret dag-prod: arr-Rel α ‹†_{C.Rel} α(‖ArrMap‖)(‖S ×_{A Rel} T‖)› by simp
    from prod-dag show arr-Rel-prod-dag:
      arr-Rel α ((†_{C.Rel} α(‖ArrMap‖)(‖S‖)) ×_{A Rel} (†_{C.Rel} α(‖ArrMap‖)(‖T‖)))
      by (auto dest: cat-Rel-is-arrD)
    then interpret prod-dag:
```

```
  arr-Rel α ‹(†C.Rel α(⦇ArrMap⦈)(⦇S⦈)) A×Rel (†C.Rel α(⦇ArrMap⦈)(⦇T⦈))›
  by simp
from ST have arr-Rel-ST: arr-Rel α (S A×Rel T)
  by (auto dest: cat-Rel-is-arrD)
show
  †C.Rel α(⦇ArrMap⦈)(⦇S A×Rel T⦈)(⦇ArrVal⦈) =
    ((†C.Rel α(⦇ArrMap⦈)(⦇S⦈)) A×Rel (†C.Rel α(⦇ArrMap⦈)(⦇T⦈)))(⦇ArrVal⦈)
proof(intro vsubset-antisym vsubsetI)
  fix bd-ac assume prems: bd-ac ∈∘ †C.Rel α(⦇ArrMap⦈)(⦇S A×Rel T⦈)(⦇ArrVal⦈)
  then obtain bd ac
    where bd-ac-def: bd-ac = ⟨bd, ac⟩
      and bd: bd ∈∘ ℛ∘ (S(⦇ArrVal⦈)) ×∘ ℛ∘ (T(⦇ArrVal⦈))
      and ac: ac ∈∘ 𝒟∘ (S(⦇ArrVal⦈)) ×∘ 𝒟∘ (T(⦇ArrVal⦈))
    by (elim cat-Rel-is-arr-ArrValE[OF dag-prod prems, unfolded cat-cs-simps])
  have ⟨ac, bd⟩ ∈∘ prod-2-Rel-ArrVal (S(⦇ArrVal⦈)) (T(⦇ArrVal⦈))
    by
      (
        rule prems[
          unfolded
            bd-ac-def
            cf-dag-Rel-ArrMap-app-iff[OF ST]
            prod-2-Rel-components
        ]
      )
  then obtain a b c d
    where ab: ⟨a, b⟩ ∈∘ S(⦇ArrVal⦈)
      and cd: ⟨c, d⟩ ∈∘ T(⦇ArrVal⦈)
      and bd-def: bd = ⟨b, d⟩
      and ac-def: ac = ⟨a, c⟩
    by auto
  show bd-ac ∈∘ ((†C.Rel α(⦇ArrMap⦈)(⦇S⦈)) A×Rel (†C.Rel α(⦇ArrMap⦈)(⦇T⦈)))(⦇ArrVal⦈)
    unfolding prod-2-Rel-components
  proof(intro prod-2-Rel-ArrValI)
    show bd-ac = ⟨⟨b, d⟩, ⟨a, c⟩⟩ unfolding bd-ac-def bd-def ac-def by simp
    from assms ab cd show
      ⟨b, a⟩ ∈∘ †C.Rel α(⦇ArrMap⦈)(⦇S⦈)(⦇ArrVal⦈)
      ⟨d, c⟩ ∈∘ †C.Rel α(⦇ArrMap⦈)(⦇T⦈)(⦇ArrVal⦈)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)+
  qed
next
  fix bd-ac assume prems:
    bd-ac ∈∘ ((†C.Rel α(⦇ArrMap⦈)(⦇S⦈)) A×Rel (†C.Rel α(⦇ArrMap⦈)(⦇T⦈)))(⦇ArrVal⦈)
  then obtain a b c d
    where bd-ac-def: bd-ac = ⟨⟨b, d⟩, a, c⟩
      and ba: ⟨b, a⟩ ∈∘ †C.Rel α(⦇ArrMap⦈)(⦇S⦈)(⦇ArrVal⦈)
      and dc: ⟨d, c⟩ ∈∘ †C.Rel α(⦇ArrMap⦈)(⦇T⦈)(⦇ArrVal⦈)
    by (elim prod-2-Rel-ArrValE[OF prems[unfolded prod-2-Rel-components]])
  then have ab: ⟨a, b⟩ ∈∘ S(⦇ArrVal⦈) and cd: ⟨c, d⟩ ∈∘ T(⦇ArrVal⦈)
    unfolding assms[THEN cf-dag-Rel-ArrMap-app-iff] by simp-all
  from ST ab cd show bd-ac ∈∘ †C.Rel α(⦇ArrMap⦈)(⦇S A×Rel T⦈)(⦇ArrVal⦈)
    unfolding bd-ac-def
    by
      (
        cs-concl cs-shallow
          cs-simp: prod-2-Rel-components cat-cs-simps
          cs-intro: prod-2-Rel-ArrValI cat-cs-intros
      )
qed
```

qed (*use dag-prod prod-dag* **in** ‹*cs-concl cs-simp*: *cat-cs-simps*›)+

qed

17.13 Product functor for *Rel*

definition *cf-prod-2-Rel* :: $V \Rightarrow V$
where *cf-prod-2-Rel* $\mathfrak{A}$ =
[
($\lambda AB \in_\circ (\mathfrak{A} \times_C \mathfrak{A})(\lfloor Obj \rfloor).\ AB(\lfloor 0 \rfloor) \times_\circ AB(\lfloor 1_{\mathbb{N}} \rfloor)$),
($\lambda ST \in_\circ (\mathfrak{A} \times_C \mathfrak{A})(\lfloor Arr \rfloor).\ (ST(\lfloor 0 \rfloor))\ _A\times_{Rel}\ (ST(\lfloor 1_{\mathbb{N}} \rfloor))$),
$\mathfrak{A} \times_C \mathfrak{A}$,
$\mathfrak{A}$
]$_\circ$

Components.

lemma *cf-prod-2-Rel-components*:
shows *cf-prod-2-Rel* $\mathfrak{A}(\lfloor ObjMap \rfloor)$ = ($\lambda AB \in_\circ (\mathfrak{A} \times_C \mathfrak{A})(\lfloor Obj \rfloor).\ AB(\lfloor 0 \rfloor) \times_\circ AB(\lfloor 1_{\mathbb{N}} \rfloor)$)
and *cf-prod-2-Rel* $\mathfrak{A}(\lfloor ArrMap \rfloor)$ =
($\lambda ST \in_\circ (\mathfrak{A} \times_C \mathfrak{A})(\lfloor Arr \rfloor).\ (ST(\lfloor 0 \rfloor))\ _A\times_{Rel}\ (ST(\lfloor 1_{\mathbb{N}} \rfloor))$)
and [*cat-cs-simps*]: *cf-prod-2-Rel* $\mathfrak{A}(\lfloor HomDom \rfloor) = \mathfrak{A} \times_C \mathfrak{A}$
and [*cat-cs-simps*]: *cf-prod-2-Rel* $\mathfrak{A}(\lfloor HomCod \rfloor) = \mathfrak{A}$
unfolding *cf-prod-2-Rel-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

17.13.1 Object map

mk-VLambda *cf-prod-2-Rel-components*(*1*)
|*vsv cf-prod-2-Rel-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-prod-2-Rel-ObjMap-vdomain*[*cat-cs-simps*]|

lemma *cf-prod-2-Rel-ObjMap-app*[*cat-cs-simps*]:
assumes $AB = [A, B]_\circ$ **and** $AB \in_\circ (\mathfrak{A} \times_C \mathfrak{A})(\lfloor Obj \rfloor)$
shows $A \otimes_{HM.O} \textit{cf-prod-2-Rel}\ \mathfrak{A}\ B = A \times_\circ B$
using *assms*(*2*)
unfolding *assms*(*1*) *cf-prod-2-Rel-components*
by (*simp add*: *nat-omega-simps*)

lemma (**in** $\mathcal{Z}$) *cf-prod-2-Rel-ObjMap-vrange*:
$\mathcal{R}_\circ$ (*cf-prod-2-Rel* (*cat-Rel* α)($\lfloor ObjMap \rfloor$)) $\subseteq_\circ$ *cat-Rel* $\alpha(\lfloor Obj \rfloor)$
proof–
interpret *Rel*: *category* α ‹*cat-Rel* α›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Rel-cs-intros*)
show *?thesis*
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-cs-simps*)
fix *AB* **assume** *prems*: $AB \in_\circ$ (*cat-Rel* $\alpha \times_C$ *cat-Rel* α)($\lfloor Obj \rfloor$)
with *Rel.category-axioms* **obtain** *A B* **where** *AB-def*: $AB = [A, B]_\circ$
and *A*: $A \in_\circ$ *cat-Rel* $\alpha(\lfloor Obj \rfloor)$
and *B*: $B \in_\circ$ *cat-Rel* $\alpha(\lfloor Obj \rfloor)$
by (*elim cat-prod-2-ObjE*[*rotated 2*])
from *prems A B* **show** *cf-prod-2-Rel* (*cat-Rel* α)($\lfloor ObjMap \rfloor$)($\lfloor AB \rfloor$) $\in_\circ$ *cat-Rel* $\alpha(\lfloor Obj \rfloor)$
unfolding *AB-def cat-Rel-components*(*1*)
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-Rel-cs-simps* **cs-intro**: *V-cs-intros*
)
qed (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
qed

17.13.2 Arrow map

mk-VLambda *cf-prod-2-Rel-components*(*2*)
|*vsv cf-prod-2-Rel-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-prod-2-Rel-ArrMap-vdomain*[*cat-cs-simps*]|

lemma *cf-prod-2-Rel-ArrMap-app*[*cat-cs-simps*]:
assumes $GF = [G, F]_\circ$ **and** $GF \in_\circ (\mathfrak{A} \times_C \mathfrak{A})(\!|Arr|\!)$
shows $G \otimes_{HM.A} cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{A}\ F = G\ _{A\times_{Rel}}\ F$
using *assms*(*2*)
unfolding *assms*(*1*) *cf-prod-2-Rel-components*
by (*simp add*: *nat-omega-simps*)

17.13.3 Product functor for *Rel* is a functor

lemma (**in** $\mathcal{Z}$) *cf-prod-2-Rel-is-functor*:
cf-prod-2-Rel (*cat-Rel* α) : *cat-Rel* $\alpha \times_C$ *cat-Rel* $\alpha \mapsto\mapsto_{C\alpha}$ *cat-Rel* α
proof–

interpret *Rel*: *category* α ‹*cat-Rel* α›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Rel-cs-intros*)

show *?thesis*
proof(*rule is-functorI'*)
show *vfsequence* (*cf-prod-2-Rel* (*cat-Rel* α))
unfolding *cf-prod-2-Rel-def* **by** *auto*
show *vcard* (*cf-prod-2-Rel* (*cat-Rel* α)) = $4_{\mathbb{N}}$
unfolding *cf-prod-2-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*cf-prod-2-Rel* (*cat-Rel* α)(|*ObjMap*|)) $\subseteq_\circ$ *cat-Rel* α(|*Obj*|)
by (*rule cf-prod-2-Rel-ObjMap-vrange*)
show *cf-prod-2-Rel* (*cat-Rel* α)(|*ArrMap*|)(|*GF*|) :
cf-prod-2-Rel (*cat-Rel* α)(|*ObjMap*|)(|*AB*|) $\mapsto_{cat\text{-}Rel\ \alpha}$
cf-prod-2-Rel (*cat-Rel* α)(|*ObjMap*|)(|*CD*|)
if *GF* : *AB* $\mapsto_{cat\text{-}Rel\ \alpha \times_C cat\text{-}Rel\ \alpha}$ *CD* **for** *AB CD GF*
proof–
from *that* **obtain** *G F A B C D*
where *GF-def*: $GF = [G, F]_\circ$
and *AB-def*: $AB = [A, B]_\circ$
and *CD-def*: $CD = [C, D]_\circ$
and *G*: $G : A \mapsto_{cat\text{-}Rel\ \alpha} C$
and *F*: $F : B \mapsto_{cat\text{-}Rel\ \alpha} D$
by (*elim cat-prod-2-is-arrE*[*OF Rel.category-axioms Rel.category-axioms*])
from *that G F* **show** *?thesis*
unfolding *GF-def AB-def CD-def*
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro:
cat-Rel-par-set-cs-intros cat-cs-intros cat-prod-cs-intros
)
qed

show
cf-prod-2-Rel (*cat-Rel* α)(|*ArrMap*|)(|$GF' \circ_{A\ cat\text{-}Rel\ \alpha \times_C cat\text{-}Rel\ \alpha}\ GF$|) =
cf-prod-2-Rel (*cat-Rel* α)(|*ArrMap*|)(|*GF'*|) $\circ_{A\ cat\text{-}Rel\ \alpha}$
cf-prod-2-Rel (*cat-Rel* α)(|*ArrMap*|)(|*GF*|)
if *GF'* : *AB'* $\mapsto_{cat\text{-}Rel\ \alpha \times_C cat\text{-}Rel\ \alpha}$ *AB''*
and *GF* : *AB* $\mapsto_{cat\text{-}Rel\ \alpha \times_C cat\text{-}Rel\ \alpha}$ *AB'*

```
    for AB′ AB″ GF′ AB GF
  proof–
    from that(2) obtain G F A A′ B B′
      where GF-def: GF = [G, F]∘
        and AB-def: AB = [A, B]∘
        and AB′-def: AB′ = [A′, B′]∘
        and G: G : A ↦_cat-Rel α A′
        and F: F : B ↦_cat-Rel α B′
      by (elim cat-prod-2-is-arrE[OF Rel.category-axioms Rel.category-axioms])
    with that(1) obtain G′ F′ A″ B″
      where GF′-def: GF′ = [G′, F′]∘
        and AB″-def: AB″ = [A″, B″]∘
        and G′: G′ : A′ ↦_cat-Rel α A″
        and F′: F′ : B′ ↦_cat-Rel α B″
      by
        (
          auto elim:
            cat-prod-2-is-arrE[OF Rel.category-axioms Rel.category-axioms]
        )
    from that G F G′ F′ show ?thesis
      unfolding GF-def AB-def AB′-def GF′-def AB″-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-prod-cs-simps prod-2-Rel-Comp
            cs-intro: cat-cs-intros cat-prod-cs-intros
        )
  qed

  show
    cf-prod-2-Rel (cat-Rel α)(|ArrMap|)((cat-Rel α ×_C cat-Rel α)(|CId|)(|AB|)) =
      cat-Rel α(|CId|)(|cf-prod-2-Rel (cat-Rel α)(|ObjMap|)(|AB|)|)
    if AB ∈∘ (cat-Rel α ×_C cat-Rel α)(|Obj|) for AB
  proof–
    from that obtain A B
      where AB-def: AB = [A, B]∘
        and A: A ∈∘ cat-Rel α(|Obj|)
        and B: B ∈∘ cat-Rel α(|Obj|)
      by (elim cat-prod-2-ObjE[OF Rel.category-axioms Rel.category-axioms])
    from A B show ?thesis
      unfolding AB-def
      by
        (
          cs-concl
            cs-simp:
              cf-prod-2-Rel-ObjMap-app cf-prod-2-Rel-ArrMap-app
              cat-cs-simps cat-prod-cs-simps
            cs-intro:
              V-cs-intros cat-cs-intros cat-Rel-cs-intros cat-prod-cs-intros
        )
  qed

qed
  (
    cs-concl cs-shallow
      cs-simp: cat-cs-simps
      cs-intro: cat-cs-intros cat-cs-intros cat-Rel-cs-intros
  )+
```

qed

lemma (in $\mathcal{Z}$**)** *cf-prod-2-Rel-is-functor'*[*cat-cs-intros*]:
assumes $\mathfrak{A}'$ = *cat-Rel* α $\times_C$ *cat-Rel* α
and $\mathfrak{B}'$ = *cat-Rel* α
and α' = α
shows *cf-prod-2-Rel* (*cat-Rel* α) : $\mathfrak{A}' \mapsto\mapsto_{C\,\alpha'} \mathfrak{B}'$
unfolding *assms* **by** (*rule cf-prod-2-Rel-is-functor*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$.*cf-prod-2-Rel-is-functor'*

17.14 Product universal property arrow for *Set*

17.14.1 Definition and elementary properties

definition *cat-Set-obj-prod-up* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *cat-Set-obj-prod-up I F A* φ =
$[(\lambda a \in_\circ A.\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ArrVal|\!)(\!|a|\!))),\ A,\ (\prod_\circ i \in_\circ I.\ F\ i)]_\circ$

Components.

lemma *cat-Set-obj-prod-up-components*:
shows *cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)$ =
$(\lambda a \in_\circ A.\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ArrVal|\!)(\!|a|\!)))$
and [*cat-Set-cs-simps*]:
cat-Set-obj-prod-up I F A $\varphi(\!|ArrDom|\!)$ = *A*
and [*cat-Set-cs-simps*]:
cat-Set-obj-prod-up I F A $\varphi(\!|ArrCod|\!)$ = $(\prod_\circ i \in_\circ I.\ F\ i)$
unfolding *cat-Set-obj-prod-up-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

17.14.2 Arrow value

mk-VLambda *cat-Set-obj-prod-up-components*(*1*)
|*vsv cat-Set-obj-prod-up-ArrVal-vsv*[*cat-Set-cs-intros*]|
|*vdomain cat-Set-obj-prod-up-ArrVal-vdomain*[*cat-Set-cs-simps*]|
|*app cat-Set-obj-prod-up-ArrVal-app*|

lemma *cat-Set-obj-prod-up-ArrVal-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : A \mapsto_{cat\text{-}Set\ \alpha} F\ i$
shows $\mathcal{R}_\circ$ (*cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)$) $\subseteq_\circ$ $(\prod_\circ i \in_\circ I.\ F\ i)$
unfolding *cat-Set-obj-prod-up-components*
proof(*intro vrange-VLambda-vsubset vproductI*)
fix *a* **assume** *prems*: $a \in_\circ A$
show $\forall\, i \in_\circ I.\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ArrVal|\!)(\!|a|\!))(\!|i|\!) \in_\circ F\ i$
proof(*intro ballI*)
fix *i* **assume** $i \in_\circ I$
with *assms prems* **show** $(\lambda i \in_\circ I.\ \varphi\ i(\!|ArrVal|\!)(\!|a|\!))(\!|i|\!) \in_\circ F\ i$
by (*cs-concl* **cs-shallow cs-simp**: *V-cs-simps* **cs-intro**: *cat-Set-cs-intros*)
qed
qed *auto*

lemma *cat-Set-obj-prod-up-ArrVal-app-vdomain*[*cat-Set-cs-simps*]:
assumes $a \in_\circ A$
shows $\mathcal{D}_\circ$ (*cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)(\!|a|\!)$) = *I*
unfolding *cat-Set-obj-prod-up-ArrVal-app*[*OF assms*] **by** *simp*

lemma *cat-Set-obj-prod-up-ArrVal-app-component*[*cat-Set-cs-simps*]:
assumes $a \in_\circ A$ **and** $i \in_\circ I$

shows *cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)(\!|a|\!)(\!|i|\!) = \varphi\ i(\!|ArrVal|\!)(\!|a|\!)$
using *assms*
by (*cs-concl* **cs-shallow cs-simp**: *cat-Set-obj-prod-up-ArrVal-app V-cs-simps*)

lemma *cat-Set-obj-prod-up-ArrVal-app-vrange*:
assumes $a \in_\circ A$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : A \mapsto_{cat\text{-}Set\ \alpha} F\ i$
shows $\mathcal{R}_\circ$ (*cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)(\!|a|\!)) \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ F\ i)$
proof(*intro vsubsetI*)
fix *b* **assume** *prems*: $b \in_\circ \mathcal{R}_\circ$ (*cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)(\!|a|\!)$)
from *assms*(*1*) **have** *vsv* (*cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)(\!|a|\!)$)
by (*auto simp*: *cat-Set-obj-prod-up-components*)
with *prems* **obtain** *i*
where *b-def*: b = *cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)(\!|a|\!)(\!|i|\!)$
and *i*: $i \in_\circ I$
by
(
auto
elim: *vsv.vrange-atE*
simp: *cat-Set-obj-prod-up-ArrVal-app*[*OF assms*(*1*)]
)
from *cat-Set-obj-prod-up-ArrVal-app-component*[*OF assms*(*1*) *i*] *b-def* **have** *b-def'*:
$b = \varphi\ i(\!|ArrVal|\!)(\!|a|\!)$
by *simp*
from *assms*(*1*) *assms*(*2*)[*OF i*] **have** $b \in_\circ F\ i$
unfolding *b-def'* **by** (*cs-concl* **cs-shallow cs-intro**: *cat-Set-cs-intros*)
with *i* **show** $b \in_\circ (\bigcup_\circ i \in_\circ I.\ F\ i)$ **by** *force*
qed

17.14.3 Product universal property arrow for *Set* is an arrow in *Set*

lemma (**in** $\mathcal{Z}$) *cat-Set-obj-prod-up-cat-Set-is-arr*:
assumes $A \in_\circ$ *cat-Set* $\alpha(\!|Obj|\!)$
and *VLambda I F* $\in_\circ$ *Vset* α
and $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : A \mapsto_{cat\text{-}Set\ \alpha} F\ i$
shows *cat-Set-obj-prod-up I F A* $\varphi : A \mapsto_{cat\text{-}Set\ \alpha} (\prod_\circ i \in_\circ I.\ F\ i)$
proof(*intro cat-Set-is-arrI arr-SetI*)
show *vfsequence* (*cat-Set-obj-prod-up I F A* φ)
unfolding *cat-Set-obj-prod-up-def* **by** *auto*
show *vcard* (*cat-Set-obj-prod-up I F A* φ) = $3_\mathbb{N}$
unfolding *cat-Set-obj-prod-up-def* **by** (*auto simp*: *nat-omega-simps*)
show
$\mathcal{R}_\circ$ (*cat-Set-obj-prod-up I F A* $\varphi(\!|ArrVal|\!)$) $\subseteq_\circ$
cat-Set-obj-prod-up I F A $\varphi(\!|ArrCod|\!)$
unfolding *cat-Set-obj-prod-up-components*(*3*)
by (*rule cat-Set-obj-prod-up-ArrVal-vrange*[*OF assms*(*3*)])
show *cat-Set-obj-prod-up I F A* $\varphi(\!|ArrCod|\!) \in_\circ$ *Vset* α
unfolding *cat-Set-cs-simps*
by (*rule Limit-vproduct-in-Vset-if-VLambda-in-VsetI*)
(*simp-all add*: *cat-Set-cs-simps assms*)
qed
(
auto
simp: *assms*[*unfolded cat-Set-components*(*1*)] *cat-Set-cs-simps*
intro: *cat-Set-cs-intros*
)

17.14.4 Further properties

lemma (in $\mathcal{Z}$**)** *cat-Set-cf-comp-proj-obj-prod-up*:
assumes $A \in_{\circ}$ *cat-Set* $\alpha(\!|Obj|\!)$
and *VLambda I F* $\in_{\circ}$ *Vset* α
and $\bigwedge i.\ i \in_{\circ} I \Longrightarrow \varphi\ i : A \mapsto_{cat\text{-}Set\ \alpha} F\ i$
and $i \in_{\circ} I$
shows
$\varphi\ i =$ *vprojection-arrow I F i* $\circ_{A\,cat\text{-}Set\ \alpha}$ *cat-Set-obj-prod-up I F A* φ
(**is** $\langle \varphi\ i =$ *?Fi* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?*$\varphi\rangle$)
proof(*rule arr-Set-eqI*[*of* α])
note $\varphi i =$ *assms*(*3*)[*OF assms*(*4*)]
note $\varphi i =$ *cat-Set-is-arrD*[*OF* φi] φi
have *Fi*: *?Fi* : $(\prod_{\circ} i \in_{\circ} I.\ F\ i) \mapsto_{cat\text{-}Set\ \alpha} F\ i$
by (*rule vprojection-arrow-is-arr*[*OF assms*(*4,2*)])
from *cat-Set-obj-prod-up-cat-Set-is-arr*[*OF assms*(*1,2,3*)] **have** φ:
cat-Set-obj-prod-up I F A $\varphi : A \mapsto_{cat\text{-}Set\ \alpha} (\prod_{\circ} i \in_{\circ} I.\ F\ i)$
by *simp*
show *arr-Set* α $(\varphi\ i)$ **by** (*rule* φi(*1*))
interpret φi: *arr-Set* α $\langle \varphi\ i \rangle$ **by** (*rule* φi(*1*))
from *Fi* φ **have** *Fi-*φ: *?Fi* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?*$\varphi : A \mapsto_{cat\text{-}Set\ \alpha} F\ i$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
then show *arr-Set-Fi-*φ: *arr-Set* α (*?Fi* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?*φ)
by (*auto simp*: *cat-Set-is-arrD*(*1*))
interpret *arr-Set* α $\langle$*?Fi* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?*$\varphi\rangle$ **by** (*rule arr-Set-Fi-*φ)
from φi **have** *dom-lhs*: $\mathcal{D}_{\circ}$ $(\varphi\ i(\!|ArrVal|\!)) = A$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *Fi-*φ **have** *dom-rhs*: $\mathcal{D}_{\circ}$ $((?Fi \circ_{A\,cat\text{-}Set\ \alpha} ?\varphi)(\!|ArrVal|\!)) = A$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $\varphi\ i(\!|ArrVal|\!) = (?Fi \circ_{A\,cat\text{-}Set\ \alpha} ?\varphi)(\!|ArrVal|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** *prems*: $a \in_{\circ} A$
from *assms*(*4*) *prems* φi(*4*) φ *Fi* **show**
$\varphi\ i(\!|ArrVal|\!)(\!|a|\!) = (?Fi \circ_{A\,cat\text{-}Set\ \alpha} ?\varphi)(\!|ArrVal|\!)(\!|a|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Set-cs-simps cat-cs-simps*
cs-intro: *cat-Set-cs-intros cat-cs-intros*
)
qed *auto*
from *Fi* φ **show**
$\varphi\ i(\!|ArrDom|\!) = (?Fi \circ_{A\,cat\text{-}Set\ \alpha} ?\varphi)(\!|ArrDom|\!)$
$\varphi\ i(\!|ArrCod|\!) = (?Fi \circ_{A\,cat\text{-}Set\ \alpha} ?\varphi)(\!|ArrCod|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps cat-Set-cs-simps* φi(*2,3*))+
qed

17.15 Coproduct universal property arrow for *Set*

17.15.1 Definition and elementary properties

definition *cat-Set-obj-coprod-up* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *cat-Set-obj-coprod-up I F A* φ =
$[(\lambda ix \in_{\circ} (\coprod_{\circ} i \in_{\circ} I.\ F\ i).\ \varphi\ (vfst\ ix)(\!|ArrVal|\!)(\!|vsnd\ ix|\!)),\ (\coprod_{\circ} i \in_{\circ} I.\ F\ i),\ A]_{\circ}$

Components.

lemma *cat-Set-obj-coprod-up-components*:
shows *cat-Set-obj-coprod-up I F A* $\varphi(\!|ArrVal|\!) =$
$(\lambda ix \in_{\circ} (\coprod_{\circ} i \in_{\circ} I.\ F\ i).\ \varphi\ (vfst\ ix)(\!|ArrVal|\!)(\!|vsnd\ ix|\!))$

and [*cat-Set-cs-simps*]:
cat-Set-obj-coprod-up I F A φ(|*ArrDom*|) = (∐∘ *i*∈∘ *I*. *F i*)
and [*cat-Set-cs-simps*]:
cat-Set-obj-coprod-up I F A φ(|*ArrCod*|) = *A*
unfolding *cat-Set-obj-coprod-up-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

17.15.2 Arrow value

mk-VLambda *cat-Set-obj-coprod-up-components*(*1*)
|*vsv cat-Set-obj-coprod-up-ArrVal-vsv*[*cat-Set-cs-intros*]|
|*vdomain cat-Set-obj-coprod-up-ArrVal-vdomain*[*cat-Set-cs-simps*]|
|*app cat-Set-obj-coprod-up-ArrVal-app′*|

lemma *cat-Set-obj-coprod-up-ArrVal-app*[*cat-cs-simps*]:
assumes *ix* = ⟨*i*, *x*⟩ **and** ⟨*i*, *x*⟩ ∈∘ (∐∘ *i*∈∘ *I*. *F i*)
shows *cat-Set-obj-coprod-up I F A φ*(|*ArrVal*|)(|*ix*|) = *φ i*(|*ArrVal*|)(|*x*|)
using *assms* **by** (*auto simp*: *cat-Set-obj-coprod-up-ArrVal-app′*)

lemma *cat-Set-obj-coprod-up-ArrVal-vrange*:
assumes ⋀*i*. *i* ∈∘ *I* ⟹ *φ i* : *F i* ↦*cat-Set α A*
shows ℛ∘ (*cat-Set-obj-coprod-up I F A φ*(|*ArrVal*|)) ⊆∘ *A*
proof
(
intro vsv.vsv-vrange-vsubset cat-Set-obj-coprod-up-ArrVal-vsv,
unfold cat-Set-cs-simps
)
fix *ix* **assume** *ix* ∈∘ (∐∘ *i*∈∘ *I*. *F i*)
then obtain *i x* **where** *ix-def*: *ix* = ⟨*i*, *x*⟩ **and** *i*: *i* ∈∘ *I* **and** *x*: *x* ∈∘ *F i*
by *auto*
show *cat-Set-obj-coprod-up I F A φ*(|*ArrVal*|)(|*ix*|) ∈∘ *A*
proof(*cs-concl-step cat-Set-obj-coprod-up-ArrVal-app*)
show *ix* = ⟨*i*, *x*⟩ **by** (*rule ix-def*)
from *i x* **show** ⟨*i*, *x*⟩ ∈∘ (∐∘ *i*∈∘ *I*. *F i*) **by** *auto*
from *i x assms*[*OF i*] **show** *φ i*(|*ArrVal*|)(|*x*|) ∈∘ *A*
by (*auto intro*: *cat-Set-ArrVal-app-vrange*)
qed
qed

17.15.3 Coproduct universal property arrow for *Set* is an arrow in *Set*

lemma (**in** *Z*) *cat-Set-obj-coprod-up-cat-Set-is-arr*:
assumes *A* ∈∘ *cat-Set α*(|*Obj*|)
and *VLambda I F* ∈∘ *Vset α*
and ⋀*i*. *i* ∈∘ *I* ⟹ *φ i* : *F i* ↦*cat-Set α A*
shows *cat-Set-obj-coprod-up I F A φ* : (∐∘ *i*∈∘ *I*. *F i*) ↦*cat-Set α A*
proof(*intro cat-Set-is-arrI arr-SetI*)
show *vfsequence* (*cat-Set-obj-coprod-up I F A φ*)
unfolding *cat-Set-obj-coprod-up-def* **by** *auto*
show *vcard* (*cat-Set-obj-coprod-up I F A φ*) = *3*ℕ
unfolding *cat-Set-obj-coprod-up-def* **by** (*auto simp*: *nat-omega-simps*)
show
ℛ∘ (*cat-Set-obj-coprod-up I F A φ*(|*ArrVal*|)) ⊆∘
cat-Set-obj-coprod-up I F A φ(|*ArrCod*|)
unfolding *cat-Set-obj-coprod-up-components*(*3*)
by (*rule cat-Set-obj-coprod-up-ArrVal-vrange*[*OF assms*(*3*)])
show *cat-Set-obj-coprod-up I F A φ*(|*ArrCod*|) ∈∘ *Vset α*
by (*simp-all add*: *cat-Set-cs-simps assms*[*unfolded cat-Set-components*(*1*)])

qed
(
auto simp:
assms
cat-Set-obj-coprod-up-components
Limit-vdunion-in-Vset-if-VLambda-in-VsetI
)

17.15.4 Further properties

lemma (in $\mathcal{Z}$**)** *cat-Set-cf-comp-coprod-up-vcia*:
assumes $A \in_{\circ} \mathit{cat\text{-}Set}\ \alpha(\!|\mathit{Obj}|\!)$
and $\mathit{VLambda}\ I\ F \in_{\circ} \mathit{Vset}\ \alpha$
and $\bigwedge i.\ i \in_{\circ} I \Longrightarrow \varphi\ i : F\ i \mapsto_{\mathit{cat\text{-}Set}\ \alpha} A$
and $i \in_{\circ} I$
shows
$\varphi\ i = \mathit{cat\text{-}Set\text{-}obj\text{-}coprod\text{-}up}\ I\ F\ A\ \varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha} \mathit{vcinjection\text{-}arrow}\ I\ F\ i$
(**is** $\langle \varphi\ i = ?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi \rangle$)
proof(*rule arr-Set-eqI*[*of* α])
note $\varphi i = \mathit{assms}(3)[\mathit{OF}\ \mathit{assms}(4)]$
note $\varphi i = \mathit{cat\text{-}Set\text{-}is\text{-}arrD}[\mathit{OF}\ \varphi i]\ \varphi i$
have *Fi*: $?Fi : F\ i \mapsto_{\mathit{cat\text{-}Set}\ \alpha} (\coprod_{\circ} i \in_{\circ} I.\ F\ i)$
by (*rule vcinjection-arrow-is-arr*[*OF assms*(4,2)])
from *cat-Set-obj-coprod-up-cat-Set-is-arr*[*OF assms*(1,2,3)] **have** φ:
$\mathit{cat\text{-}Set\text{-}obj\text{-}coprod\text{-}up}\ I\ F\ A\ \varphi : (\coprod_{\circ} i \in_{\circ} I.\ F\ i) \mapsto_{\mathit{cat\text{-}Set}\ \alpha} A$
by *simp*
show $\mathit{arr\text{-}Set}\ \alpha\ (\varphi\ i)$ **by** (*rule* $\varphi i(1)$)
then interpret φi: $\mathit{arr\text{-}Set}\ \alpha\ \langle \varphi\ i \rangle$ **.**
from *Fi* φ **have** *Fi-φ*: $?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi : F\ i \mapsto_{\mathit{cat\text{-}Set}\ \alpha} A$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
then show *arr-Set-Fi-φ*: $\mathit{arr\text{-}Set}\ \alpha\ (?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi)$
by (*auto simp*: *cat-Set-is-arrD*(1))
interpret $\mathit{arr\text{-}Set}\ \alpha\ \langle ?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi \rangle$ **by** (*rule arr-Set-Fi-φ*)
from φi **have** *dom-lhs*: $\mathcal{D}_{\circ}\ (\varphi\ i(\!|\mathit{ArrVal}|\!)) = F\ i$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *Fi-φ* **have** *dom-rhs*: $\mathcal{D}_{\circ}\ ((?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi)(\!|\mathit{ArrVal}|\!)) = F\ i$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $\varphi\ i(\!|\mathit{ArrVal}|\!) = (?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi)(\!|\mathit{ArrVal}|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** $a \in_{\circ} F\ i$
from *assms*(4) *this* $\varphi i(4)$ φ *Fi* **show**
$\varphi\ i(\!|\mathit{ArrVal}|\!)(\!|a|\!) = (?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi)(\!|\mathit{ArrVal}|\!)(\!|a|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Set-cs-simps cat-cs-simps*
cs-intro: *vdunionI cat-Set-cs-intros cat-cs-intros*
)
qed *auto*
from *Fi* φ **show**
$\varphi\ i(\!|\mathit{ArrDom}|\!) = (?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi)(\!|\mathit{ArrDom}|\!)$
$\varphi\ i(\!|\mathit{ArrCod}|\!) = (?\varphi \circ_{A\,\mathit{cat\text{-}Set}\ \alpha}\ ?Fi)(\!|\mathit{ArrCod}|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps cat-Set-cs-simps* $\varphi i(2,3)$)+
qed

17.16 Equalizer object for the category *Set*

The definition of the (non-categorical concept of an) equalizer can be found in [2][10]

definition *vequalizer* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *vequalizer X f g* = *set* $\{x.\ x \in_\circ X \wedge f(\lfloor ArrVal \rfloor)(\lfloor x \rfloor) = g(\lfloor ArrVal \rfloor)(\lfloor x \rfloor)\}$

lemma *small-vequalizer*[*simp*]:
  small $\{x.\ x \in_\circ X \wedge f(\lfloor ArrVal \rfloor)(\lfloor x \rfloor) = g(\lfloor ArrVal \rfloor)(\lfloor x \rfloor)\}$
  by *auto*

Rules.

lemma *vequalizerI*:
  assumes $x \in_\circ X$ **and** $f(\lfloor ArrVal \rfloor)(\lfloor x \rfloor) = g(\lfloor ArrVal \rfloor)(\lfloor x \rfloor)$
  shows $x \in_\circ$ *vequalizer X f g*
  using *assms* **unfolding** *vequalizer-def* **by** *auto*

lemma *vequalizerD*[*dest*]:
  assumes $x \in_\circ$ *vequalizer X f g*
  shows $x \in_\circ X$ **and** $f(\lfloor ArrVal \rfloor)(\lfloor x \rfloor) = g(\lfloor ArrVal \rfloor)(\lfloor x \rfloor)$
  using *assms* **unfolding** *vequalizer-def* **by** *auto*

lemma *vequalizerE*[*elim*]:
  assumes $x \in_\circ$ *vequalizer X f g*
  obtains $x \in_\circ X$ **and** $f(\lfloor ArrVal \rfloor)(\lfloor x \rfloor) = g(\lfloor ArrVal \rfloor)(\lfloor x \rfloor)$
  using *assms* **unfolding** *vequalizer-def* **by** *auto*

Elementary results.

lemma *vequalizer-vsubset-vdomain*[*cat-Set-cs-intros*]: *vequalizer a g f* $\subseteq_\circ$ *a*
  by *auto*

lemma *Limit-vequalizer-in-Vset*[*cat-Set-cs-intros*]:
  assumes *Limit* α **and** $a \in_\circ$ *cat-Set* $\alpha(\lfloor Obj \rfloor)$
  shows *vequalizer a g f* $\in_\circ$ *cat-Set* $\alpha(\lfloor Obj \rfloor)$
  using *assms* **unfolding** *cat-Set-components*(*1*) **by** *auto*

lemma *vequalizer-flip*: *vequalizer a f g* = *vequalizer a g f*
  unfolding *vequalizer-def* **by** *auto*

lemma *cat-Set-incl-Set-commute*:
  assumes $\mathfrak{g} : \mathfrak{a} \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{b}$ **and** $\mathfrak{f} : \mathfrak{a} \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{b}$
  shows
    $\mathfrak{g} \circ_{A\, cat\text{-}Set\ \alpha}$ *incl-Set* (*vequalizer* $\mathfrak{a}\ \mathfrak{f}\ \mathfrak{g}$) $\mathfrak{a}$ =
    $\mathfrak{f} \circ_{A\, cat\text{-}Set\ \alpha}$ *incl-Set* (*vequalizer* $\mathfrak{a}\ \mathfrak{f}\ \mathfrak{g}$) $\mathfrak{a}$
  (**is** $\langle \mathfrak{g} \circ_{A\, cat\text{-}Set\ \alpha}$ *?incl* = $\mathfrak{f} \circ_{A\, cat\text{-}Set\ \alpha}$ *?incl*$\rangle$)
proof–

  interpret $\mathfrak{g}$: *arr-Set* α $\mathfrak{g}$
    rewrites $\mathfrak{g}(\lfloor ArrDom \rfloor) = \mathfrak{a}$ **and** $\mathfrak{g}(\lfloor ArrCod \rfloor) = \mathfrak{b}$
    by (*intro cat-Set-is-arrD*[*OF assms*(*1*)])+
  interpret $\mathfrak{f}$: *arr-Set* α $\mathfrak{f}$
    rewrites $\mathfrak{f}(\lfloor ArrDom \rfloor) = \mathfrak{a}$ **and** $\mathfrak{f}(\lfloor ArrCod \rfloor) = \mathfrak{b}$
    by (*intro cat-Set-is-arrD*[*OF assms*(*2*)])+

  note [*cat-Set-cs-intros*] = $\mathfrak{g}$.*arr-Set-ArrDom-in-Vset* $\mathfrak{f}$.*arr-Set-ArrCod-in-Vset*

  from *assms* **have** $\mathfrak{g}$-*incl*:

[10] https://en.wikipedia.org/wiki/Equaliser_(mathematics)

```
    𝔤 ∘_A cat-Set α ?incl : vequalizer 𝔞 𝔣 𝔤 ↦_cat-Set α 𝔟
    by (cs-concl cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros)
  then have dom-lhs: 𝒟∘ ((𝔤 ∘_A cat-Set α ?incl)(|ArrVal|)) = vequalizer 𝔞 𝔣 𝔤
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)+
  from assms have 𝔣-incl:
    𝔣 ∘_A cat-Set α ?incl : vequalizer 𝔞 𝔣 𝔤 ↦_cat-Set α 𝔟
    by (cs-concl cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros)
  then have dom-rhs: 𝒟∘ ((𝔣 ∘_A cat-Set α ?incl)(|ArrVal|)) = vequalizer 𝔞 𝔣 𝔤
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)+

  show ?thesis
  proof(rule arr-Set-eqI)
    from 𝔤-incl show arr-Set-𝔤-incl: arr-Set α (𝔤 ∘_A cat-Set α ?incl)
      by (auto dest: cat-Set-is-arrD(1))
    interpret arr-Set-𝔤-incl: arr-Set α ‹𝔤 ∘_A cat-Set α ?incl›
      by (rule arr-Set-𝔤-incl)
    from 𝔣-incl show arr-Set-𝔣-incl: arr-Set α (𝔣 ∘_A cat-Set α ?incl)
      by (auto dest: cat-Set-is-arrD(1))
    interpret arr-Set-𝔣-incl: arr-Set α ‹𝔣 ∘_A cat-Set α ?incl›
      by (rule arr-Set-𝔣-incl)
    show (𝔤 ∘_A cat-Set α ?incl)(|ArrVal|) = (𝔣 ∘_A cat-Set α ?incl)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈∘ vequalizer 𝔞 𝔣 𝔤
      with assms show
        (𝔤 ∘_A cat-Set α ?incl)(|ArrVal|)(|a|) = (𝔣 ∘_A cat-Set α ?incl)(|ArrVal|)(|a|)
        by
          (
            cs-concl
              cs-simp: vequalizerD(2) cat-Set-cs-simps cat-cs-simps
              cs-intro: V-cs-intros cat-Set-cs-intros cat-cs-intros
          )
    qed auto
  qed (use 𝔤-incl 𝔣-incl in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed
```

17.17 Application of a function to a finite sequence as an arrow in *Set*

```
definition vfsequence-map :: V ⇒ V
  where vfsequence-map F =
    [
      (λxs∈∘vfsequences-on (F(|ArrDom|)). F(|ArrVal|) ∘∘ xs),
      vfsequences-on (F(|ArrDom|)),
      vfsequences-on (F(|ArrCod|))
    ]∘
```

Components.

```
lemma vfsequence-map-components:
  shows vfsequence-map F(|ArrVal|) =
    (λxs∈∘vfsequences-on (F(|ArrDom|)). F(|ArrVal|) ∘∘ xs)
    and [cat-cs-simps]: vfsequence-map F(|ArrDom|) = vfsequences-on (F(|ArrDom|))
    and [cat-cs-simps]: vfsequence-map F(|ArrCod|) = vfsequences-on (F(|ArrCod|))
  unfolding vfsequence-map-def arr-field-simps
  by (simp-all add: nat-omega-simps)
```

17.17.1 Arrow value

```
mk-VLambda vfsequence-map-components(1)
```

```
  |vsv vfsequence-map-ArrVal-vsv[cat-cs-intros, cat-Set-cs-intros]|
  |vdomain vfsequence-map-ArrVal-vdomain[cat-cs-simps, cat-Set-cs-simps]|
  |app vfsequence-map-ArrVal-app|

lemma vfsequence-map-ArrVal-app-app:
  assumes F : A ↦_{cat-Set α} B
    and xs ∈∘ vfsequences-on A
    and i ∈∘ 𝒟∘ xs
  shows vfsequence-map F(|ArrVal|)(|xs|)(|i|) = F(|ArrVal|)(|xs(|i|)|)
proof–
  note FD = cat-Set-is-arrD[OF assms(1)]
  interpret arr-Set α F
    rewrites F(|ArrDom|) = A and F(|ArrCod|) = B
    by (intro FD)+
  note xsD = vfsequences-onD[OF assms(2)]
  interpret xs: vfsequence xs by (rule xsD(1))
  from assms xsD(2)[OF assms(3)] show ?thesis
    by
      (
        cs-concl
          cs-simp: V-cs-simps cat-cs-simps vfsequence-map-ArrVal-app
          cs-intro: V-cs-intros
      )
qed
```

17.17.2 Application of a function to a finite sequence is an arrow in *Set*

```
lemma vfsequence-map-is-arr:
  assumes F : A ↦_{cat-Set α} B
  shows vfsequence-map F : vfsequences-on A ↦_{cat-Set α} vfsequences-on B
proof–

  note FD = cat-Set-is-arrD[OF assms(1)]
  interpret arr-Set α F
    rewrites [cat-cs-simps]: F(|ArrDom|) = A and [cat-cs-simps]: F(|ArrCod|) = B
    by (intro FD)+

  show ?thesis
  proof(intro cat-Set-is-arrI arr-SetI , unfold cat-cs-simps)
    show vfsequence (vfsequence-map F)
      unfolding vfsequence-map-def by auto
    show vcard (vfsequence-map F) = 3_ℕ
      unfolding vfsequence-map-def by (simp-all add: nat-omega-simps)
    show ℛ∘ (vfsequence-map F(|ArrVal|)) ⊆∘ vfsequences-on B
      unfolding vfsequence-map-components
    proof
      (
        intro vrange-VLambda-vsubset vfsequences-onI;
        elim vfsequences-onE;
        unfold cat-cs-simps
      )
      fix xs assume prems: vfsequence xs i ∈∘ 𝒟∘ xs ⟹ xs(|i|) ∈∘ A for i
      interpret xs: vfsequence xs by (rule prems(1))
      have [intro]: x ∈∘ 𝒟∘ (F(|ArrVal|)) if x ∈∘ ℛ∘ xs for x
      proof–
        from that obtain i where i: i ∈∘ 𝒟∘ xs and x-def: x = xs(|i|)
          by (auto dest: xs.vrange-atD)
        from prems(2)[OF i] show x ∈∘ 𝒟∘ (F(|ArrVal|))
```

```
      unfolding x-def arr-Set-ArrVal-vdomain .
    qed
    show vfsequence (F(|ArrVal|) ∘ₒ xs)
      by (intro vfsequence-vcomp-vsv-vfsequence vsubsetI)
        (auto intro: prems(1))
    fix i assume prems': i ∈ₒ 𝒟ₒ (F(|ArrVal|) ∘ₒ xs)
    moreover have 𝒟ₒ (F(|ArrVal|) ∘ₒ xs) = 𝒟ₒ xs
      by (intro vdomain-vcomp-vsubset vsubsetI) (auto intro: prems(1))
    ultimately have i: i ∈ₒ 𝒟ₒ xs by simp
    with assms(1) prems(2)[OF i] show (F(|ArrVal|) ∘ₒ xs)(|i|) ∈ₒ B
      by
        (
          cs-concl
            cs-simp: V-cs-simps cat-cs-simps
            cs-intro: V-cs-intros cat-Set-cs-intros
        )
  qed

  qed
    (
      auto intro:
        vfsequences-on-in-VsetI
        arr-Set-ArrDom-in-Vset
        arr-Set-ArrCod-in-Vset
        cat-cs-intros
    )

qed

lemma (in 𝒵) vfsequence-map-is-monic-arr:
  assumes F : A ↦_{mon cat-Set α} B
  shows vfsequence-map F : vfsequences-on A ↦_{mon cat-Set α} vfsequences-on B
proof–

  note cat-Set-is-monic-arrD[OF assms]
  note FD = this cat-Set-is-arrD[OF this(1)]
  interpret F: arr-Set α F
    rewrites [cat-cs-simps]: F(|ArrDom|) = A and [cat-cs-simps]: F(|ArrCod|) = B
    by (intro FD)+

  show ?thesis
  proof
    (
      intro cat-Set-is-monic-arrI vfsequence-map-is-arr FD(1) vsv.vsv-valeq-v11I,
      unfold cat-cs-simps;
      (elim vfsequences-onE)?
    )

    fix xs ys assume prems:
      vfsequence-map F(|ArrVal|)(|xs|) = vfsequence-map F(|ArrVal|)(|ys|)
      vfsequence xs
      ⋀i. i ∈ₒ 𝒟ₒ xs ⟹ xs(|i|) ∈ₒ A
      vfsequence ys
      ⋀i. i ∈ₒ 𝒟ₒ ys ⟹ ys(|i|) ∈ₒ A

    interpret xs: vfsequence xs by (rule prems(2))
    interpret ys: vfsequence ys by (rule prems(4))
```

have $xs \in_\circ$ *vfsequences-on* $(F(\!|ArrDom|\!))$
  unfolding *cat-cs-simps* **by** (*intro vfsequences-onI prems*(*2*,*3*))
from *vfsequence-map-ArrVal-app*[*OF this*] **have** *F-xs*:
  vfsequence-map $F(\!|ArrVal|\!)(\!|xs|\!) = F(\!|ArrVal|\!) \circ_\circ xs$
  by *simp*
from *prems*(*3*) **have** *rxs*: $\mathcal{R}_\circ\ xs \subseteq_\circ A$
  by (*intro vsubsetI*) (*auto dest*: *xs.vrange-atD*)
from *xs.vfsequence-vdomain-in-omega* **have** *dxs*: $\mathcal{D}_\circ\ xs \in_\circ Vset\ \alpha$
  by (*auto intro*!: *Axiom-of-Infinity*)
note *xs-is-arr* = *cat-Set-arr-of-vsv-is-arr*
  [
    OF xs.vsv-axioms rxs,
    unfolded cat-Set-components(*1*),
    OF dxs F.arr-Par-ArrDom-in-Vset
  ]

have *ys*: $ys \in_\circ$ *vfsequences-on* $(F(\!|ArrDom|\!))$
  unfolding *cat-cs-simps* **by** (*intro vfsequences-onI prems*(*4*,*5*))
from *vfsequence-map-ArrVal-app*[*OF this*] **have** *F-ys*:
  vfsequence-map $F(\!|ArrVal|\!)(\!|ys|\!) = F(\!|ArrVal|\!) \circ_\circ ys$
  by *simp*
from *prems*(*5*) **have** *rys*: $\mathcal{R}_\circ\ ys \subseteq_\circ A$
  by (*intro vsubsetI*) (*auto dest*: *ys.vrange-atD*)
from *ys.vfsequence-vdomain-in-omega* **have** *dys*: $\mathcal{D}_\circ\ ys \in_\circ Vset\ \alpha$
  by (*auto intro*!: *Axiom-of-Infinity*)
note *ys-is-arr* = *cat-Set-arr-of-vsv-is-arr*
  [
    OF ys.vsv-axioms rys,
    unfolded cat-Set-components(*1*),
    OF dys F.arr-Par-ArrDom-in-Vset
  ]

note *Fxs-Fys* = *prems*(*1*)[*unfolded F-xs F-ys*]

from *rxs* **have** *dom-rxs*: $\mathcal{D}_\circ\ (F(\!|ArrVal|\!) \circ_\circ xs) = \mathcal{D}_\circ\ xs$
  by (*intro vdomain-vcomp-vsubset vsubsetI*, *unfold F.arr-Set-ArrVal-vdomain*)
    auto
moreover from *rys* **have** *dom-rys*: $\mathcal{D}_\circ\ (F(\!|ArrVal|\!) \circ_\circ ys) = \mathcal{D}_\circ\ ys$
  by (*intro vdomain-vcomp-vsubset vsubsetI*, *unfold F.arr-Set-ArrVal-vdomain*)
    auto
ultimately have *dxs-dys*: $\mathcal{D}_\circ\ xs = \mathcal{D}_\circ\ ys$
  by (*simp add*: *prems*(*1*)[*unfolded F-xs F-ys*])

from *FD*(*1*) *xs-is-arr* **have** *lhs-is-arr*:
  $F \circ_{A\,cat\text{-}Set\ \alpha}\ cat\text{-}Set\text{-}arr\text{-}of\text{-}vsv\ xs\ A : \mathcal{D}_\circ\ xs \mapsto_{cat\text{-}Set\ \alpha} B$
  by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then have *dom-lhs*:
  $\mathcal{D}_\circ\ ((F \circ_{A\,cat\text{-}Set\ \alpha}\ cat\text{-}Set\text{-}arr\text{-}of\text{-}vsv\ xs\ A)(\!|ArrVal|\!)) = \mathcal{D}_\circ\ xs$
  by (*simp add*: *cat-cs-simps*)

from *FD*(*1*) *ys-is-arr* **have** *rhs-is-arr*:
  $F \circ_{A\,cat\text{-}Set\ \alpha}\ cat\text{-}Set\text{-}arr\text{-}of\text{-}vsv\ ys\ A : \mathcal{D}_\circ\ xs \mapsto_{cat\text{-}Set\ \alpha} B$
  by (*cs-concl* **cs-simp**: *dxs-dys* **cs-intro**: *cat-cs-intros*)
then have *dom-rhs*:
  $\mathcal{D}_\circ\ ((F \circ_{A\,cat\text{-}Set\ \alpha}\ cat\text{-}Set\text{-}arr\text{-}of\text{-}vsv\ ys\ A)(\!|ArrVal|\!)) = \mathcal{D}_\circ\ xs$
  by (*simp add*: *cat-cs-simps*)

have *F-xs-F-ys*:

```
    F ∘_A cat-Set α cat-Set-arr-of-vsv xs A =
      F ∘_A cat-Set α cat-Set-arr-of-vsv ys A
  proof(rule arr-Set-eqI[of α])
    show
      (F ∘_A cat-Set α cat-Set-arr-of-vsv xs A)(⌊ArrVal⌋) =
        (F ∘_A cat-Set α cat-Set-arr-of-vsv ys A)(⌊ArrVal⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix i assume prems: i ∈_∘ 𝒟_∘ xs
      from prems rxs have xsi: xs(⌊i⌋) ∈_∘ A
        by (auto dest: xs.vdomain-atD)
      from prems rys have ysi: ys(⌊i⌋) ∈_∘ A
        by (auto simp: dxs-dys dest: ys.vdomain-atD)
      from arg-cong[OF Fxs-Fys, where f=‹λx. x(⌊i⌋)›] prems FD(1) xsi ysi
      have F(⌊ArrVal⌋)(⌊xs(⌊i⌋)⌋) = F(⌊ArrVal⌋)(⌊ys(⌊i⌋)⌋)
        by
          (
            cs-prems
              cs-simp: V-cs-simps cat-cs-simps dxs-dys[symmetric]
              cs-intro: V-cs-intros cat-cs-intros
          )
      with prems FD(1) xs-is-arr ys-is-arr show
        (F ∘_A cat-Set α cat-Set-arr-of-vsv xs A)(⌊ArrVal⌋)(⌊i⌋) =
          (F ∘_A cat-Set α cat-Set-arr-of-vsv ys A)(⌊ArrVal⌋)(⌊i⌋)
        by
          (
            cs-concl
              cs-simp: cat-Set-cs-simps cat-cs-simps dxs-dys[symmetric]
              cs-intro: cat-cs-intros
          )
    qed (use lhs-is-arr rhs-is-arr in ‹auto dest: cat-Set-is-arrD›)
  qed
    (
      use lhs-is-arr rhs-is-arr in
        ‹auto simp: cat-cs-simps dest: cat-Set-is-arrD(1)›
    )+
  have cat-Set-arr-of-vsv xs A = cat-Set-arr-of-vsv ys A
    by
      (
        rule is-monic-arrD(2)[
          OF assms(1) xs-is-arr, unfolded dxs-dys, OF ys-is-arr, OF F-xs-F-ys
          ]
      )
  from arg-cong [OF this, where f=‹λx. x(⌊ArrVal⌋)›, unfolded cat-Set-cs-simps]
  show xs = ys .

qed (auto intro: cat-cs-intros)

qed

lemma (in 𝒵) vfsequence-map-is-epic-arr:
  assumes F : A ↦_epi cat-Set α B
  shows vfsequence-map F : vfsequences-on A ↦_epi cat-Set α vfsequences-on B
proof–

  note cat-Set-is-epic-arrD[OF assms]
  note FD = this cat-Set-is-arrD[OF this(1)]

  interpret F: arr-Set α F
```

```
  rewrites [cat-cs-simps]: F(|ArrDom|) = A and [cat-cs-simps]: F(|ArrCod|) = B
  by (intro FD)+
interpret SF: arr-Set α ‹vfsequence-map F›
  rewrites vfsequence-map F(|ArrDom|) = vfsequences-on A
    and vfsequence-map F(|ArrCod|) = vfsequences-on B
  by (intro cat-Set-is-arrD[OF vfsequence-map-is-arr[OF FD(1)]])+

show ?thesis
proof
  (
    intro cat-Set-is-epic-arrI,
    rule vfsequence-map-is-arr[OF FD(1)],
    rule vsubset-antisym,
    rule SF.arr-Par-ArrVal-vrange,
    rule vsubsetI
  )
  fix xs assume prems: xs ∈∘ vfsequences-on B
  note xsD = vfsequences-onD[OF prems]
  interpret vfsequence xs by (rule xsD(1))
  define ys where ys = (λi∈∘𝒟∘ xs. SOME x. x ∈∘ A ∧ xs(|i|) = F(|ArrVal|)(|x|))
  have ys-vdomain: 𝒟∘ ys = 𝒟∘ xs unfolding ys-def by simp
  interpret ys: vfsequence ys
    by (rule vfsequenceI)
      (auto intro: vfsequence-vdomain-in-omega simp: ys-def)
  have ysi: ys(|i|) = (SOME x. x ∈∘ A ∧ xs(|i|) = F(|ArrVal|)(|x|))
    if i ∈∘ 𝒟∘ xs for i
    using that unfolding ys-def by simp
  have ysi: ys(|i|) ∈∘ A
    and xsi-def: xs(|i|) = F(|ArrVal|)(|ys(|i|)|)
    if i ∈∘ 𝒟∘ xs for i
  proof–
    have xs(|i|) ∈∘ ℛ∘ (F(|ArrVal|)) by (rule xsD(2)[OF that, folded FD(2)])
    then obtain x where x: x ∈∘ A and xsi-def: xs(|i|) = F(|ArrVal|)(|x|)
      by (auto elim: F.ArrVal.vrange-atE simp: F.arr-Set-ArrVal-vdomain)
    show ys(|i|) ∈∘ A and xs(|i|) = F(|ArrVal|)(|ys(|i|)|)
      unfolding ysi[OF that]
      by
        (
          all‹rule someI2-ex, intro exI conjI; (elim conjE)?›,
          tactic‹distinct-subgoals-tac›
        )
        (auto simp: x xsi-def)
  qed
  show xs ∈∘ ℛ∘ (vfsequence-map F(|ArrVal|))
  proof
    (
      intro vsv.vsv-vimageI2′ cat-cs-intros,
      cs-concl-step vfsequence-map-ArrVal-app,
      unfold cat-cs-simps,
      tactic‹distinct-subgoals-tac›
    )
    show ys ∈∘ vfsequences-on A
      by (intro vfsequences-onI ys.vfsequence-axioms)
        (auto intro: ysi simp: ys-vdomain)
    show xs = F(|ArrVal|) ∘∘ ys
    proof(rule vsv-eqI)
      show 𝒟∘ xs = 𝒟∘ (F(|ArrVal|) ∘∘ ys)
        unfolding ys-vdomain[symmetric]
```

```
      proof(intro vdomain-vcomp-vsubset[symmetric] vsubsetI)
        fix y assume y ∈∘ ℛ∘ ys
        then obtain i where i: i ∈∘ 𝒟∘ ys and y-def: y = ys⦇i⦈
          by (auto dest: ys.vrange-atD)
        from i show y ∈∘ 𝒟∘ (F⦇ArrVal⦈)
          unfolding y-def F.arr-Set-ArrVal-vdomain ys-vdomain by (rule ysi)
      qed
      show xs⦇i⦈ = (F⦇ArrVal⦈ ∘∘ ys)⦇i⦈
        if i ∈∘ 𝒟∘ xs for i
        using FD(1) that
        by
          (
            cs-concl
              cs-simp: V-cs-simps cat-cs-simps xsi-def ys-vdomain
              cs-intro: V-cs-intros ysi
          )
    qed (auto intro: vsv-vcomp)
  qed
qed

lemma vfsequence-map-is-iso-arr:
  assumes F : A ↦iso cat-Set α B
  shows vfsequence-map F : vfsequences-on A ↦iso cat-Set α vfsequences-on B
proof–
  note cat-Set-is-iso-arrD[OF assms]
  note FD = this cat-Set-is-arrD[OF this(1)]
  interpret F: arr-Set α F
    rewrites [cat-cs-simps]: F⦇ArrDom⦈ = A and [cat-cs-simps]: F⦇ArrCod⦈ = B
    by (intro FD)+
  interpret Set: category α ‹cat-Set α› by (cs-concl cs-intro: cat-cs-intros)
  show ?thesis
    by
      (
        intro
        F.cat-Set-is-iso-arr-if-monic-and-epic
        F.vfsequence-map-is-monic-arr[
          OF Set.cat-is-iso-arr-is-monic-arr[OF assms]
          ]
        F.vfsequence-map-is-epic-arr[
          OF Set.cat-is-iso-arr-is-epic-arr[OF assms]
          ]
      )
qed
```

17.18 An injection from the range of an arrow in *Set* into its domain

17.18.1 Definition and elementary properties

```
definition vrange-iso :: V ⇒ V
  where vrange-iso F =
    [
      (λy∈∘ℛ∘ (F⦇ArrVal⦈). (SOME x. x ∈∘ F⦇ArrDom⦈ ∧ y = F⦇ArrVal⦈⦇x⦈)),
      ℛ∘ (F⦇ArrVal⦈),
      F⦇ArrDom⦈
    ]∘
```

Components.

lemma *vrange-iso-components*:
shows *vrange-iso* $F(\!|ArrVal|\!) =$
$(\lambda y \in_\circ \mathcal{R}_\circ\ (F(\!|ArrVal|\!)).\ (SOME\ x.\ x \in_\circ F(\!|ArrDom|\!) \wedge y = F(\!|ArrVal|\!)(\!|x|\!)))$
and [*cat-cs-simps*]: *vrange-iso* $F(\!|ArrDom|\!) = \mathcal{R}_\circ\ (F(\!|ArrVal|\!))$
and [*cat-cs-simps*]: *vrange-iso* $F(\!|ArrCod|\!) = F(\!|ArrDom|\!)$
unfolding *vrange-iso-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

17.18.2 Arrow value

mk-VLambda *vrange-iso-components*(*1*)
|*vsv vrange-iso-ArrVal-vsv*[*cat-cs-intros*]|
|*vdomain vrange-iso-ArrVal-vdomain*[*cat-cs-simps*]|
|*app vrange-iso-ArrVal-app*|

lemma *vrange-iso-ArrVal-rules*:
assumes $F : A \mapsto_{cat\text{-}Set\ \alpha} B$ **and** $y \in_\circ \mathcal{R}_\circ\ (F(\!|ArrVal|\!))$
shows *vrange-iso* $F(\!|ArrVal|\!)(\!|y|\!) \in_\circ A$
and $y = F(\!|ArrVal|\!)(\!|$*vrange-iso* $F(\!|ArrVal|\!)(\!|y|\!)|\!)$
proof–
note *FD* = *cat-Set-is-arrD*[*OF assms*(*1*)]
interpret *F*: *arr-Set* α *F*
rewrites [*cat-cs-simps*]: $F(\!|ArrDom|\!) = A$ **and** [*cat-cs-simps*]: $F(\!|ArrCod|\!) = B$
by (*intro FD*)+
from *assms*(*2*) **have** *vri-Fy-def*:
vrange-iso $F(\!|ArrVal|\!)(\!|y|\!) = (SOME\ x.\ x \in_\circ F(\!|ArrDom|\!) \wedge y = F(\!|ArrVal|\!)(\!|x|\!))$
by (*cs-concl* **cs-simp**: *vrange-iso-ArrVal-app*)
from *assms*(*2*) *F.arr-Set-ArrVal-vdomain* **obtain** *x*
where *x*: $x \in_\circ A$ **and** *y-def*: $y = F(\!|ArrVal|\!)(\!|x|\!)$
by (*auto elim*: *F.ArrVal.vrange-atE*)
show *vrange-iso* $F(\!|ArrVal|\!)(\!|y|\!) \in_\circ A$
and $y = F(\!|ArrVal|\!)(\!|$*vrange-iso* $F(\!|ArrVal|\!)(\!|y|\!)|\!)$
unfolding *vri-Fy-def cat-cs-simps*
by (*all*‹*rule someI2-ex*; (*intro exI conjI*)*?*; (*elim conjE*)*?*›)
(*simp-all add*: *x y-def*)
qed

17.18.3 An injection from the range of a function into its domain is a monic in *Set*

lemma *vrange-iso-is-arr*:
assumes $F : A \mapsto_{cat\text{-}Set\ \alpha} B$
shows *vrange-iso* $F : \mathcal{R}_\circ\ (F(\!|ArrVal|\!)) \mapsto_{cat\text{-}Set\ \alpha} A$
proof–

note *FD* = *cat-Set-is-arrD*[*OF assms*(*1*)]
interpret *F*: *arr-Set* α *F*
rewrites [*cat-cs-simps*]: $F(\!|ArrDom|\!) = A$ **and** [*cat-cs-simps*]: $F(\!|ArrCod|\!) = B$
by (*intro FD*)+

show *vrange-iso* $F : \mathcal{R}_\circ\ (F(\!|ArrVal|\!)) \mapsto_{cat\text{-}Set\ \alpha} A$
proof(*intro cat-Set-is-arrI arr-SetI*, *unfold cat-cs-simps*)
show *vfsequence* (*vrange-iso F*)
unfolding *vrange-iso-def* **by** (*simp-all add*: *nat-omega-simps*)
show *vsv* (*vrange-iso* $F(\!|ArrVal|\!)$)
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
then interpret *vsv* ‹*vrange-iso* $F(\!|ArrVal|\!)$›
rewrites $\mathcal{D}_\circ$ (*vrange-iso* $F(\!|ArrVal|\!)$) = $\mathcal{R}_\circ\ (F(\!|ArrVal|\!))$
unfolding *cat-cs-simps* **by** *simp-all*
show *vcard* (*vrange-iso F*) = $3_{\mathbb{N}}$

```
    unfolding vrange-iso-def by (simp-all add: nat-omega-simps)
  show ℛ∘ (vrange-iso F(|ArrVal|)) ⊆∘ A
  proof(intro vsubsetI)
    fix x assume x ∈∘ ℛ∘ (vrange-iso F(|ArrVal|))
    then obtain y where y: y ∈∘ ℛ∘ (F(|ArrVal|))
      and x-def: x = vrange-iso F(|ArrVal|)(|y|)
      by (auto dest: vrange-atD)
    show x ∈∘ A
      unfolding x-def
      by (rule vrange-iso-ArrVal-rules(1)[OF assms y, unfolded cat-cs-simps])
  qed
qed
(
  auto
    simp: F.arr-Set-ArrDom-in-Vset
    intro: vrange-in-VsetI F.arr-Rel-ArrVal-in-Vset
)

qed

lemma vrange-iso-is-arr':
  assumes F : A ↦_{cat-Set α} B
    and B' = ℛ∘ (F(|ArrVal|))
    and 𝔈' = cat-Set α
  shows vrange-iso F : B' ↦_{𝔈'} A
  using assms(1) unfolding assms(2,3) by (rule vrange-iso-is-arr)

lemma vrange-iso-is-monic-arr:
  assumes F : A ↦_{cat-Set α} B
  shows vrange-iso F : ℛ∘ (F(|ArrVal|)) ↦_{mon cat-Set α} A
proof–
  note FD = cat-Set-is-arrD[OF assms(1)]
  interpret F: arr-Set α F
    rewrites [cat-cs-simps]: F(|ArrDom|) = A and [cat-cs-simps]: F(|ArrCod|) = B
    by (intro FD)+
  show ?thesis
  proof
    (
      intro cat-Set-is-monic-arrI vrange-iso-is-arr,
      rule assms,
      rule vsv.vsv-valeq-v11I[OF vrange-iso-ArrVal-vsv],
      unfold cat-cs-simps
    )
    fix x y assume prems:
      x ∈∘ ℛ∘ (F(|ArrVal|))
      y ∈∘ ℛ∘ (F(|ArrVal|))
      vrange-iso F(|ArrVal|)(|x|) = vrange-iso F(|ArrVal|)(|y|)
    show x = y
      by
        (
          rule vrange-iso-ArrVal-rules(2)
            [
              OF assms prems(1),
              unfolded prems(3),
              folded vrange-iso-ArrVal-rules(2)[OF assms prems(2)]
            ]
        )
  qed simp
```

qed

lemma *vrange-iso-is-monic-arr'*:
assumes $F : A \mapsto_{cat\text{-}Set\ \alpha} B$
and $B' = \mathcal{R}_{\circ}\ (F(\!|ArrVal|\!))$
and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
shows *vrange-iso* $F : B' \mapsto_{mon_{\mathfrak{C}'}} A$
using *assms*(*1*) **unfolding** *assms*(*2,3*) **by** (*rule vrange-iso-is-monic-arr*)

17.19 Auxiliary

This subsection is reserved for insignificant helper lemmas and rules that are used in applied formalization elsewhere.

lemma (**in** $\mathcal{Z}$) *cat-Rel-CId-is-cat-Set-arr*:
assumes $A \in_{\circ} cat\text{-}Rel\ \alpha(\!|Obj|\!)$
shows $cat\text{-}Rel\ \alpha(\!|CId|\!)(\!|A|\!) : A \mapsto_{cat\text{-}Set\ \alpha} A$
proof–
from *assms* **show** *?thesis*
unfolding *cat-Rel-components cat-Set-components*(*6*)[*symmetric*]
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Set-components*(*1*) **cs-intro**: *cat-cs-intros*
)
qed

lemma (**in** $\mathcal{Z}$) *cat-Rel-CId-is-cat-Set-arr'*[*cat-rel-par-Set-cs-intros*]:
assumes $A \in_{\circ} cat\text{-}Rel\ \alpha(\!|Obj|\!)$
and $B' = A$
and $C' = A$
and $\mathfrak{C}' = cat\text{-}Set\ \alpha$
shows $cat\text{-}Rel\ \alpha(\!|CId|\!)(\!|A|\!) : B' \mapsto_{\mathfrak{C}'} C'$
using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cat-Rel-CId-is-cat-Set-arr*)

18 *GRPH*

18.1 Background

The methodology for the exposition of *GRPH* as a category is analogous to the one used in [8] for the exposition of *GRPH* as a semicategory.

named-theorems *cat-GRPH-simps*
named-theorems *cat-GRPH-intros*

18.2 Definition and elementary properties

definition *cat-GRPH* :: $V \Rightarrow V$
  where *cat-GRPH* α =
    [
      set $\{\mathfrak{C}.$ *digraph* α $\mathfrak{C}\}$,
      all-dghms α,
      $(\lambda\mathfrak{F}\in_\circ$*all-dghms* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$,
      $(\lambda\mathfrak{F}\in_\circ$*all-dghms* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$,
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-GRPH* α)$.$ $\mathfrak{GF}(\!|0|\!) \circ_{DGHM} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$,
      $(\lambda\mathfrak{C}\in_\circ$*set* $\{\mathfrak{C}.$ *digraph* α $\mathfrak{C}\}.$ *dghm-id* $\mathfrak{C})$
    ]$_\circ$

Components.

lemma *cat-GRPH-components*:
  shows *cat-GRPH* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *digraph* α $\mathfrak{C}\}$
    and *cat-GRPH* $\alpha(\!|$*Arr*$|\!)$ = *all-dghms* α
    and *cat-GRPH* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-dghms* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$
    and *cat-GRPH* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-dghms* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
    and *cat-GRPH* $\alpha(\!|$*Comp*$|\!)$ =
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-GRPH* α)$.$ $\mathfrak{GF}(\!|0|\!) \circ_{DGHM} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
    and *cat-GRPH* $\alpha(\!|$*CId*$|\!)$ = $(\lambda\mathfrak{C}\in_\circ$*set* $\{\mathfrak{C}.$ *digraph* α $\mathfrak{C}\}.$ *dghm-id* $\mathfrak{C})$
  unfolding *cat-GRPH-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-GRPH*: *cat-smc* (*cat-GRPH* α) = *smc-GRPH* α
proof(*rule vsv-eqI*)
  have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-GRPH* α)) = $5_{\mathbb{N}}$
    unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-GRPH* α) = $5_{\mathbb{N}}$
    unfolding *smc-GRPH-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*cat-smc* (*cat-GRPH* α)) = $\mathcal{D}_\circ$ (*smc-GRPH* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show
    $a \in_\circ \mathcal{D}_\circ$ (*cat-smc* (*cat-GRPH* α)) $\Longrightarrow$ *cat-smc* (*cat-GRPH* α)$(\!|a|\!)$ = *smc-GRPH* $\alpha(\!|a|\!)$
    for *a*
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold cat-smc-def dg-field-simps cat-GRPH-def smc-GRPH-def
)
      (*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *cat-smc-def smc-GRPH-def*)

lemmas-with [*folded cat-smc-GRPH*, *unfolded slicing-simps*]:
  — Digraph
  cat-GRPH-ObjI = *smc-GRPH-ObjI*

and *cat-GRPH-ObjD* = *smc-GRPH-ObjD*
and *cat-GRPH-ObjE* = *smc-GRPH-ObjE*
and *cat-GRPH-Obj-iff*[*cat-GRPH-simps*] = *smc-GRPH-Obj-iff*
and *cat-GRPH-Dom-app*[*cat-GRPH-simps*] = *smc-GRPH-Dom-app*
and *cat-GRPH-Cod-app*[*cat-GRPH-simps*] = *smc-GRPH-Cod-app*
and *cat-GRPH-is-arrI* = *smc-GRPH-is-arrI*
and *cat-GRPH-is-arrD* = *smc-GRPH-is-arrD*
and *cat-GRPH-is-arrE* = *smc-GRPH-is-arrE*
and *cat-GRPH-is-arr-iff*[*cat-GRPH-simps*] = *smc-GRPH-is-arr-iff*

lemmas-with [*folded cat-smc-GRPH*, *unfolded slicing-simps*, *unfolded cat-smc-GRPH*]:
— Semicategory
cat-GRPH-Comp-vdomain = *smc-GRPH-Comp-vdomain*
and *cat-GRPH-composable-arrs-dg-GRPH* = *smc-GRPH-composable-arrs-dg-GRPH*
and *cat-GRPH-Comp* = *smc-GRPH-Comp*
and *cat-GRPH-Comp-app*[*cat-GRPH-simps*] = *smc-GRPH-Comp-app*

lemmas-with (**in** $\mathcal{Z}$) [*folded cat-smc-GRPH*, *unfolded slicing-simps*]:
— Semicategory
cat-GRPH-obj-initialI = *smc-GRPH-obj-initialI*
and *cat-GRPH-obj-initialD* = *smc-GRPH-obj-initialD*
and *cat-GRPH-obj-initialE* = *smc-GRPH-obj-initialE*
and *cat-GRPH-obj-initial-iff*[*cat-GRPH-simps*] = *smc-GRPH-obj-initial-iff*
and *cat-GRPH-obj-terminalI* = *smc-GRPH-obj-terminalI*
and *cat-GRPH-obj-terminalE* = *smc-GRPH-obj-terminalE*

18.3 Identity

lemma *cat-GRPH-CId-app*[*cat-GRPH-simps*]:
assumes *digraph* α $\mathfrak{C}$
shows *cat-GRPH* $\alpha(\!|CId|\!)(\!|\mathfrak{C}|\!)$ = *dghm-id* $\mathfrak{C}$
using *assms* **unfolding** *cat-GRPH-components* **by** *simp*

lemma *cat-GRPH-CId-vdomain*: $\mathcal{D}_\circ$ (*cat-GRPH* $\alpha(\!|CId|\!)$) = *set* $\{\mathfrak{C}.\ digraph\ \alpha\ \mathfrak{C}\}$
unfolding *cat-GRPH-components* **by** *auto*

lemma *cat-GRPH-CId-vrange*: $\mathcal{R}_\circ$ (*cat-GRPH* $\alpha(\!|CId|\!)$) $\subseteq_\circ$ *all-dghms* α
proof(*rule vsubsetI*)
fix $\mathfrak{H}$ **assume** $\mathfrak{H} \in_\circ \mathcal{R}_\circ$ (*cat-GRPH* $\alpha(\!|CId|\!)$)
then obtain $\mathfrak{A}$
where $\mathfrak{H}$-*def*: $\mathfrak{H}$ = *cat-GRPH* $\alpha(\!|CId|\!)(\!|\mathfrak{A}|\!)$ **and** $\mathfrak{A}$: $\mathfrak{A} \in_\circ \mathcal{D}_\circ$ (*cat-GRPH* $\alpha(\!|CId|\!)$)
unfolding *cat-GRPH-components* **by** *auto*
from $\mathfrak{A}$ **have** $\mathfrak{H}$-*def'*: $\mathfrak{H}$ = *dghm-id* $\mathfrak{A}$
unfolding $\mathfrak{H}$-*def cat-GRPH-CId-vdomain* **by** (*auto simp*: *cat-GRPH-CId-app*)
from $\mathfrak{A}$ *digraph.dg-dghm-id-is-dghm* **show** $\mathfrak{H} \in_\circ$ *all-dghms* α
unfolding $\mathfrak{H}$-*def'* *cat-GRPH-CId-vdomain* **by** *force*
qed

18.4 *GRPH* is a category

lemma (**in** $\mathcal{Z}$) *tiny-category-cat-GRPH*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *tiny-category* β (*cat-GRPH* α)
proof(*intro tiny-categoryI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *vfsequence* (*cat-GRPH* α) **unfolding** *cat-GRPH-def* **by** *simp*
show *vcard* (*cat-GRPH* α) = $6_{\mathbb{N}}$
unfolding *cat-GRPH-def* **by** (*simp add*: *nat-omega-simps*)

```
  show cat-GRPH α(|CId|)(|𝔅|) ∘_A cat-GRPH α 𝔉 = 𝔉
    if 𝔉 : 𝔄 ↦_cat-GRPH α 𝔅 for 𝔉 𝔄 𝔅
    using that
    unfolding cat-GRPH-is-arr-iff
    by (cs-concl cs-simp: dg-cs-simps cat-GRPH-simps cs-intro: dg-cs-intros)
  show 𝔉 ∘_A cat-GRPH α cat-GRPH α(|CId|)(|𝔅|) = 𝔉
    if 𝔉 : 𝔅 ↦_cat-GRPH α ℭ for 𝔉 𝔅 ℭ
    using that
    unfolding cat-GRPH-is-arr-iff
    by (cs-concl cs-simp: dg-cs-simps cat-GRPH-simps cs-intro: dg-cs-intros)
qed
  (
    simp-all add:
      assms
      cat-smc-GRPH
      cat-GRPH-components
      digraph.dg-dghm-id-is-dghm
      cat-GRPH-is-arr-iff
      tiny-semicategory-smc-GRPH
  )
```

18.5 Isomorphism

```
lemma cat-GRPH-is-iso-arrI:
  assumes 𝔉 : 𝔄 ↦↦_DG.iso α 𝔅
  shows 𝔉 : 𝔄 ↦_iso cat-GRPH α 𝔅
proof(intro is-iso-arrI is-inverseI)
  from assms show 𝔉: 𝔉 : 𝔄 ↦_cat-GRPH α 𝔅
    unfolding cat-GRPH-is-arr-iff by auto
  note iso-thms = is-iso-dghm-is-iso-arr[OF assms]
  from iso-thms(1) show inv-𝔉: inv-dghm 𝔉 : 𝔅 ↦_cat-GRPH α 𝔄
    unfolding cat-GRPH-is-arr-iff by auto
  from assms show 𝔉 : 𝔄 ↦_cat-GRPH α 𝔅
    unfolding cat-GRPH-is-arr-iff by auto
  from assms have 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅 by auto
  show inv-dghm 𝔉 ∘_A cat-GRPH α 𝔉 = cat-GRPH α(|CId|)(|𝔄|)
    unfolding cat-GRPH-CId-app[OF 𝔄] cat-GRPH-Comp-app[OF inv-𝔉 𝔉]
    by (rule iso-thms(2))
  show 𝔉 ∘_A cat-GRPH α inv-dghm 𝔉 = cat-GRPH α(|CId|)(|𝔅|)
    unfolding cat-GRPH-CId-app[OF 𝔅] cat-GRPH-Comp-app[OF 𝔉 inv-𝔉]
    by (rule iso-thms(3))
qed

lemma cat-GRPH-is-iso-arrD:
  assumes 𝔉 : 𝔄 ↦_iso cat-GRPH α 𝔅
  shows 𝔉 : 𝔄 ↦↦_DG.iso α 𝔅
proof–
  from is-iso-arrD[OF assms] have 𝔉: 𝔉 : 𝔄 ↦_cat-GRPH α 𝔅
    and (∃𝔊. is-inverse (cat-GRPH α) 𝔊 𝔉)
    by simp-all
  then obtain 𝔊 where 𝔊𝔉: is-inverse (cat-GRPH α) 𝔊 𝔉 by clarsimp
  then obtain 𝔄′ 𝔅′ where 𝔊′: 𝔊 : 𝔅′ ↦_cat-GRPH α 𝔄′
    and 𝔉′: 𝔉 : 𝔄′ ↦_cat-GRPH α 𝔅′
    and 𝔊𝔉: 𝔊 ∘_A cat-GRPH α 𝔉 = cat-GRPH α(|CId|)(|𝔄′|)
    and 𝔉𝔊: 𝔉 ∘_A cat-GRPH α 𝔊 = cat-GRPH α(|CId|)(|𝔅′|)
    by auto
  from 𝔉 𝔉′ have 𝔄′: 𝔄′ = 𝔄 and 𝔅′: 𝔅′ = 𝔅 by auto
  from 𝔉 have 𝔉: 𝔉 : 𝔄 ↦↦_DG α 𝔅 unfolding cat-GRPH-is-arr-iff by simp
```

then have $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$ **by** *auto*
from $\mathfrak{G}'$ **have** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{A}$
unfolding $\mathfrak{A}'$ $\mathfrak{B}'$ *cat-GRPH-is-arr-iff* **by** *simp*
moreover from $\mathfrak{G}\mathfrak{F}$ **have** $\mathfrak{G} \circ_{DGHM} \mathfrak{F}$ = *dghm-id* $\mathfrak{A}$
unfolding $\mathfrak{A}'$ *cat-GRPH-Comp-app*[*OF* $\mathfrak{G}'$ $\mathfrak{F}'$] *cat-GRPH-CId-app*[*OF* $\mathfrak{A}$] **by** *simp*
moreover from $\mathfrak{F}\mathfrak{G}$ **have** $\mathfrak{F} \circ_{DGHM} \mathfrak{G}$ = *dghm-id* $\mathfrak{B}$
unfolding $\mathfrak{B}'$ *cat-GRPH-Comp-app*[*OF* $\mathfrak{F}'$ $\mathfrak{G}'$] *cat-GRPH-CId-app*[*OF* $\mathfrak{B}$] **by** *simp*
ultimately show *?thesis* **using** $\mathfrak{F}$ **by** (*elim is-iso-arr-is-iso-dghm*)
qed

lemma *cat-GRPH-is-iso-arrE*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}GRPH\ \alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{B}$
using *assms* **by** (*auto dest*: *cat-GRPH-is-iso-arrD*)

lemma *cat-GRPH-is-iso-arr-iff*[*cat-GRPH-simps*]:
$\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}GRPH\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{B}$
using *cat-GRPH-is-iso-arrI cat-GRPH-is-iso-arrD* **by** *auto*

18.6 Isomorphic objects

lemma *cat-GRPH-obj-isoI*:
assumes $\mathfrak{A} \approx_{DG\alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{obj cat\text{-}GRPH\ \alpha} \mathfrak{B}$
proof–
from *iso-digraphD*[*OF assms*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{B}$
by *clarsimp*
from *cat-GRPH-is-iso-arrI*[*OF this*] **show** *?thesis* **by** (*rule obj-isoI*)
qed

lemma *cat-GRPH-obj-isoD*:
assumes $\mathfrak{A} \approx_{obj cat\text{-}GRPH\ \alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{DG\alpha} \mathfrak{B}$
proof–
from *obj-isoD*[*OF assms*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}GRPH\ \alpha} \mathfrak{B}$
by *clarsimp*
from *cat-GRPH-is-iso-arrD*[*OF this*] **show** *?thesis*
by (*rule iso-digraphI*)
qed

lemma *cat-GRPH-obj-isoE*:
assumes $\mathfrak{A} \approx_{obj cat\text{-}GRPH\ \alpha} \mathfrak{B}$
obtains $\mathfrak{A} \approx_{DG\alpha} \mathfrak{B}$
using *assms* **by** (*auto simp*: *cat-GRPH-obj-isoD*)

lemma *cat-GRPH-obj-iso-iff*: $\mathfrak{A} \approx_{obj cat\text{-}GRPH\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{A} \approx_{DG\alpha} \mathfrak{B}$
using *cat-GRPH-obj-isoI cat-GRPH-obj-isoD* **by** (*intro iffI*) *auto*

19 *SemiCAT*

19.1 Background

The methodology for the exposition of *SemiCAT* as a category is analogous to the one used in [8] for the exposition of *SemiCAT* as a semicategory.

named-theorems *cat-SemiCAT-simps*
named-theorems *cat-SemiCAT-intros*

19.2 Definition and elementary properties

definition *cat-SemiCAT* :: $V \Rightarrow V$
  where *cat-SemiCAT* α =
    [
      set $\{\mathfrak{C}.\ \textit{semicategory}\ \alpha\ \mathfrak{C}\}$,
      all-smcfs α,
      $(\lambda\mathfrak{F}\in_\circ \textit{all-smcfs}\ \alpha.\ \mathfrak{F}(\!|\textit{HomDom}|\!))$,
      $(\lambda\mathfrak{F}\in_\circ \textit{all-smcfs}\ \alpha.\ \mathfrak{F}(\!|\textit{HomCod}|\!))$,
      $(\lambda\mathfrak{GF}\in_\circ \textit{composable-arrs}\ (\textit{dg-SemiCAT}\ \alpha).\ \mathfrak{GF}(\!|0|\!) \circ_{SMCF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$,
      $(\lambda\mathfrak{C}\in_\circ \textit{set}\ \{\mathfrak{C}.\ \textit{semicategory}\ \alpha\ \mathfrak{C}\}.\ \textit{smcf-id}\ \mathfrak{C})$
    ]$_\circ$

Components.

lemma *cat-SemiCAT-components*:
  shows *cat-SemiCAT* $\alpha(\!|\textit{Obj}|\!)$ = *set* $\{\mathfrak{C}.\ \textit{semicategory}\ \alpha\ \mathfrak{C}\}$
    and *cat-SemiCAT* $\alpha(\!|\textit{Arr}|\!)$ = *all-smcfs* α
    and *cat-SemiCAT* $\alpha(\!|\textit{Dom}|\!)$ = $(\lambda\mathfrak{F}\in_\circ \textit{all-smcfs}\ \alpha.\ \mathfrak{F}(\!|\textit{HomDom}|\!))$
    and *cat-SemiCAT* $\alpha(\!|\textit{Cod}|\!)$ = $(\lambda\mathfrak{F}\in_\circ \textit{all-smcfs}\ \alpha.\ \mathfrak{F}(\!|\textit{HomCod}|\!))$
    and *cat-SemiCAT* $\alpha(\!|\textit{Comp}|\!)$ =
      $(\lambda\mathfrak{GF}\in_\circ \textit{composable-arrs}\ (\textit{dg-SemiCAT}\ \alpha).\ \mathfrak{GF}(\!|0|\!) \circ_{SMCF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
    and *cat-SemiCAT* $\alpha(\!|\textit{CId}|\!)$ = $(\lambda\mathfrak{C}\in_\circ \textit{set}\ \{\mathfrak{C}.\ \textit{semicategory}\ \alpha\ \mathfrak{C}\}.\ \textit{smcf-id}\ \mathfrak{C})$
  unfolding *cat-SemiCAT-def dg-field-simps*
  by (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-SemiCAT*: *cat-smc* (*cat-SemiCAT* α) = *smc-SemiCAT* α
proof(*rule vsv-eqI*)
  have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-SemiCAT* α)) = $5_{\mathbb{N}}$
    unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-SemiCAT* α) = $5_{\mathbb{N}}$
    unfolding *smc-SemiCAT-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*cat-smc* (*cat-SemiCAT* α)) = $\mathcal{D}_\circ$ (*smc-SemiCAT* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show $a \in_\circ \mathcal{D}_\circ$ (*cat-smc* (*cat-SemiCAT* α)) $\Longrightarrow$
    cat-smc (*cat-SemiCAT* α)$(\!|a|\!)$ = *smc-SemiCAT* $\alpha(\!|a|\!)$
    for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold cat-smc-def dg-field-simps cat-SemiCAT-def smc-SemiCAT-def
)
      (*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *cat-smc-def smc-SemiCAT-def*)

lemmas-with [*folded cat-smc-SemiCAT*, *unfolded slicing-simps*]:
  — Digraph

cat-SemiCAT-ObjI = *smc-SemiCAT-ObjI*
and *cat-SemiCAT-ObjD* = *smc-SemiCAT-ObjD*
and *cat-SemiCAT-ObjE* = *smc-SemiCAT-ObjE*
and *cat-SemiCAT-Obj-iff*[*cat-SemiCAT-simps*] = *smc-SemiCAT-Obj-iff*
and *cat-SemiCAT-Dom-app*[*cat-SemiCAT-simps*] = *smc-SemiCAT-Dom-app*
and *cat-SemiCAT-Cod-app*[*cat-SemiCAT-simps*] = *smc-SemiCAT-Cod-app*
and *cat-SemiCAT-is-arrI* = *smc-SemiCAT-is-arrI*
and *cat-SemiCAT-is-arrD* = *smc-SemiCAT-is-arrD*
and *cat-SemiCAT-is-arrE* = *smc-SemiCAT-is-arrE*
and *cat-SemiCAT-is-arr-iff*[*cat-SemiCAT-simps*] = *smc-SemiCAT-is-arr-iff*

lemmas-with [
folded cat-smc-SemiCAT, *unfolded slicing-simps*, *unfolded cat-smc-SemiCAT*
]:
— Semicategory
cat-SemiCAT-Comp-vdomain = *smc-SemiCAT-Comp-vdomain*
and *cat-SemiCAT-composable-arrs-dg-SemiCAT* =
smc-SemiCAT-composable-arrs-dg-SemiCAT
and *cat-SemiCAT-Comp* = *smc-SemiCAT-Comp*
and *cat-SemiCAT-Comp-app*[*cat-SemiCAT-simps*] = *smc-SemiCAT-Comp-app*
and *cat-SemiCAT-Comp-vrange* = *smc-SemiCAT-Comp-vrange*

lemmas-with (in $\mathcal{Z}$**)** [*folded cat-smc-SemiCAT*, *unfolded slicing-simps*]:
— Semicategory
cat-SemiCAT-obj-initialI = *smc-SemiCAT-obj-initialI*
and *cat-SemiCAT-obj-initialD* = *smc-SemiCAT-obj-initialD*
and *cat-SemiCAT-obj-initialE* = *smc-SemiCAT-obj-initialE*
and *cat-SemiCAT-obj-initial-iff*[*cat-SemiCAT-simps*] =
smc-SemiCAT-obj-initial-iff
and *cat-SemiCAT-obj-terminalI* = *smc-SemiCAT-obj-terminalI*
and *cat-SemiCAT-obj-terminalE* = *smc-SemiCAT-obj-terminalE*

19.3 Identity

lemma *cat-SemiCAT-CId-app*[*cat-SemiCAT-simps*]:
assumes *semicategory* α $\mathfrak{C}$
shows *cat-SemiCAT* $\alpha(\!|CId|\!)(\!|\mathfrak{C}|\!)$ = *smcf-id* $\mathfrak{C}$
using *assms* **unfolding** *cat-SemiCAT-components* **by** *simp*

lemma *cat-SemiCAT-CId-vdomain*[*cat-SemiCAT-simps*]:
$\mathcal{D}_\circ$ (*cat-SemiCAT* $\alpha(\!|CId|\!)$) = *set* $\{\mathfrak{C}.$ *semicategory* α $\mathfrak{C}\}$
unfolding *cat-SemiCAT-components* **by** *auto*

lemma *cat-SemiCAT-CId-vrange*: $\mathcal{R}_\circ$ (*cat-SemiCAT* $\alpha(\!|CId|\!)$) $\subseteq_\circ$ *all-smcfs* α
proof(*rule vsubsetI*)
fix $\mathfrak{H}$ **assume** $\mathfrak{H}$ $\in_\circ$ $\mathcal{R}_\circ$ (*cat-SemiCAT* $\alpha(\!|CId|\!)$)
then obtain $\mathfrak{A}$
where $\mathfrak{H}$-*def*: $\mathfrak{H}$ = *cat-SemiCAT* $\alpha(\!|CId|\!)(\!|\mathfrak{A}|\!)$
and $\mathfrak{A}$: $\mathfrak{A}$ $\in_\circ$ $\mathcal{D}_\circ$ (*cat-SemiCAT* $\alpha(\!|CId|\!)$)
unfolding *cat-SemiCAT-components* **by** *auto*
from $\mathfrak{A}$ **have** $\mathfrak{H}$-*def'*: $\mathfrak{H}$ = *smcf-id* $\mathfrak{A}$
unfolding $\mathfrak{H}$-*def cat-SemiCAT-CId-vdomain* **by** (*auto simp*: *cat-SemiCAT-CId-app*)
from $\mathfrak{A}$ *semicategory.smc-smcf-id-is-semifunctor* **show** $\mathfrak{H}$ $\in_\circ$ *all-smcfs* α
unfolding $\mathfrak{H}$-*def'* *cat-SemiCAT-CId-vdomain* **by** *force*
qed

19.4 *SemiCAT* is a category

lemma (in $\mathcal{Z}$**)** *tiny-category-cat-SemiCAT*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_{\circ} \beta$
shows *tiny-category* β (*cat-SemiCAT* α)
proof(*intro tiny-categoryI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *vfsequence* (*cat-SemiCAT* α) **unfolding** *cat-SemiCAT-def* **by** *simp*
show *vcard* (*cat-SemiCAT* α) = $6_{\mathbb{N}}$
unfolding *cat-SemiCAT-def* **by** (*simp add*: *nat-omega-simps*)
show *cat-SemiCAT* $\alpha(\!|CId|\!)(\!|\mathfrak{B}|\!) \circ_{A\,cat\text{-}SemiCAT\ \alpha} \mathfrak{F} = \mathfrak{F}$
if $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}SemiCAT\ \alpha} \mathfrak{B}$ **for** $\mathfrak{F}$ $\mathfrak{A}$ $\mathfrak{B}$
using *that*
unfolding *cat-SemiCAT-is-arr-iff*
by (*cs-concl* **cs-simp**: *smc-cs-simps cat-SemiCAT-simps* **cs-intro**: *smc-cs-intros*)
show $\mathfrak{F} \circ_{A\,cat\text{-}SemiCAT\ \alpha}$ *cat-SemiCAT* $\alpha(\!|CId|\!)(\!|\mathfrak{B}|\!) = \mathfrak{F}$
if $\mathfrak{F} : \mathfrak{B} \mapsto_{cat\text{-}SemiCAT\ \alpha} \mathfrak{C}$ **for** $\mathfrak{F}$ $\mathfrak{B}$ $\mathfrak{C}$
using *that*
unfolding *cat-SemiCAT-is-arr-iff*
by (*cs-concl* **cs-simp**: *smc-cs-simps cat-SemiCAT-simps* **cs-intro**: *smc-cs-intros*)
qed
(
simp-all add:
assms
cat-smc-SemiCAT
cat-SemiCAT-components
cat-SemiCAT-is-arr-iff
tiny-semicategory-smc-SemiCAT
semicategory.smc-smcf-id-is-semifunctor
)

19.5 Isomorphism

lemma *cat-SemiCAT-is-iso-arrI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC.iso\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto_{iso\,cat\text{-}SemiCAT\ \alpha} \mathfrak{B}$
proof(*intro is-iso-arrI is-inverseI*)
interpret *is-iso-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*)
from *assms* **show** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}SemiCAT\ \alpha} \mathfrak{B}$
unfolding *cat-SemiCAT-is-arr-iff* **by** *auto*
note *iso-thms* = *is-iso-semifunctor-is-iso-arr*[*OF assms*]
from *iso-thms*(*1*) **show** *inv-*$\mathfrak{F}$: *inv-smcf* $\mathfrak{F} : \mathfrak{B} \mapsto_{cat\text{-}SemiCAT\ \alpha} \mathfrak{A}$
unfolding *cat-SemiCAT-is-arr-iff* **by** *auto*
from *assms* **show** $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}SemiCAT\ \alpha} \mathfrak{B}$
unfolding *cat-SemiCAT-is-arr-iff* **by** *auto*
from *assms* **have** $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** *auto*
show *inv-smcf* $\mathfrak{F} \circ_{A\,cat\text{-}SemiCAT\ \alpha} \mathfrak{F}$ = *cat-SemiCAT* $\alpha(\!|CId|\!)(\!|\mathfrak{A}|\!)$
unfolding *cat-SemiCAT-CId-app*[*OF* $\mathfrak{A}$] *cat-SemiCAT-Comp-app*[*OF inv-*$\mathfrak{F}$ $\mathfrak{F}$]
by (*rule iso-thms*(*2*))
show $\mathfrak{F} \circ_{A\,cat\text{-}SemiCAT\ \alpha}$ *inv-smcf* $\mathfrak{F}$ = *cat-SemiCAT* $\alpha(\!|CId|\!)(\!|\mathfrak{B}|\!)$
unfolding *cat-SemiCAT-CId-app*[*OF* $\mathfrak{B}$] *cat-SemiCAT-Comp-app*[*OF* $\mathfrak{F}$ *inv-*$\mathfrak{F}$]
by (*rule iso-thms*(*3*))
qed

lemma *cat-SemiCAT-is-iso-arrD*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{iso\,cat\text{-}SemiCAT\ \alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC.iso\alpha} \mathfrak{B}$
proof–
from *is-iso-arrD*[*OF assms*] **have** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}SemiCAT\ \alpha} \mathfrak{B}$

and $(\exists \mathfrak{G}.\ \textit{is-inverse}\ (\textit{cat-SemiCAT}\ \alpha)\ \mathfrak{G}\ \mathfrak{F})$
by *simp-all*
then obtain $\mathfrak{G}$ **where** $\mathfrak{G}\mathfrak{F}$: $\textit{is-inverse}\ (\textit{cat-SemiCAT}\ \alpha)\ \mathfrak{G}\ \mathfrak{F}$ **by** *clarsimp*
then obtain $\mathfrak{A}'\ \mathfrak{B}'$ **where** $\mathfrak{G}'$: $\mathfrak{G} : \mathfrak{B}' \mapsto_{\textit{cat-SemiCAT}\ \alpha} \mathfrak{A}'$
and $\mathfrak{F}'$: $\mathfrak{F} : \mathfrak{A}' \mapsto_{\textit{cat-SemiCAT}\ \alpha} \mathfrak{B}'$
and $\mathfrak{G}\mathfrak{F}$: $\mathfrak{G} \circ_{A\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{F} = \textit{cat-SemiCAT}\ \alpha(\!|\textit{CId}|\!)(|\mathfrak{A}'|)$
and $\mathfrak{F}\mathfrak{G}$: $\mathfrak{F} \circ_{A\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{G} = \textit{cat-SemiCAT}\ \alpha(\!|\textit{CId}|\!)(|\mathfrak{B}'|)$
by *auto*
from $\mathfrak{F}$ $\mathfrak{F}'$ **have** $\mathfrak{A}'$: $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}'$: $\mathfrak{B}' = \mathfrak{B}$ **by** *auto*
from $\mathfrak{F}$ **have** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$ **unfolding** *cat-SemiCAT-is-arr-iff* **by** *simp*
interpret *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
have $\mathfrak{A}$: *semicategory* $\alpha\ \mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* $\alpha\ \mathfrak{B}$
by (*cs-concl* **cs-intro**: *smc-cs-intros*)+
from $\mathfrak{G}'$ **have** $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{A}$
unfolding $\mathfrak{A}'\ \mathfrak{B}'$ *cat-SemiCAT-is-arr-iff* **by** *simp*
moreover from $\mathfrak{G}\mathfrak{F}$ **have** $\mathfrak{G} \circ_{SMCF} \mathfrak{F} = \textit{smcf-id}\ \mathfrak{A}$
unfolding $\mathfrak{A}'$ *cat-SemiCAT-Comp-app*[*OF* $\mathfrak{G}'\ \mathfrak{F}'$] *cat-SemiCAT-CId-app*[*OF* $\mathfrak{A}$]
by *simp*
moreover from $\mathfrak{F}\mathfrak{G}$ **have** $\mathfrak{F} \circ_{SMCF} \mathfrak{G} = \textit{smcf-id}\ \mathfrak{B}$
unfolding $\mathfrak{B}'$ *cat-SemiCAT-Comp-app*[*OF* $\mathfrak{F}'\ \mathfrak{G}'$] *cat-SemiCAT-CId-app*[*OF* $\mathfrak{B}$]
by *simp*
ultimately show *?thesis*
using $\mathfrak{F}$ **by** (*elim is-iso-arr-is-iso-semifunctor*)
qed

lemma *cat-SemiCAT-is-iso-arrE*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{iso\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
using *assms* **by** (*auto dest*: *cat-SemiCAT-is-iso-arrD*)

lemma *cat-SemiCAT-is-iso-arr-iff*[*cat-SemiCAT-simps*]:
$\mathfrak{F} : \mathfrak{A} \mapsto_{iso\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
using *cat-SemiCAT-is-iso-arrI cat-SemiCAT-is-iso-arrD* **by** *auto*

19.6 Isomorphic objects

lemma *cat-SemiCAT-obj-isoI*:
assumes $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{obj\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{B}$
proof–
from *iso-semicategoryD*[*OF assms*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
by *clarsimp*
from *cat-SemiCAT-is-iso-arrI*[*OF this*] **show** *?thesis* **by** (*rule obj-isoI*)
qed

lemma *cat-SemiCAT-obj-isoD*:
assumes $\mathfrak{A} \approx_{obj\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$
proof–
from *obj-isoD*[*OF assms*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto_{iso\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{B}$
by *clarsimp*
from *cat-SemiCAT-is-iso-arrD*[*OF this*] **show** *?thesis*
by (*rule iso-semicategoryI*)
qed

lemma *cat-SemiCAT-obj-isoE*:
assumes $\mathfrak{A} \approx_{obj\,\textit{cat-SemiCAT}\ \alpha} \mathfrak{B}$
obtains $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$

using *assms* **by** (*auto simp*: *cat-SemiCAT-obj-isoD*)

lemma *cat-SemiCAT-obj-iso-iff*[*cat-SemiCAT-simps*]:
$\mathfrak{A} \approx_{obj} \textit{cat-SemiCAT}\ \alpha\ \mathfrak{B} \longleftrightarrow \mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$
using *cat-SemiCAT-obj-isoI cat-SemiCAT-obj-isoD* **by** (*intro iffI*) *auto*

20 *CAT* as a digraph

20.1 Background

CAT is usually defined as a category of categories and functors (e.g., see Chapter I-2 in [7]). However, there is little that can prevent one from exposing *CAT* as a digraph and provide additional structure gradually in subsequent theories. Thus, in this section, α-*CAT* is defined as a digraph of categories and functors in the set V_α, and α-*Cat* is defined as a digraph of tiny categories and tiny functors in V_α.

named-theorems *dg-CAT-simps*
named-theorems *dg-CAT-intros*

20.2 Definition and elementary properties

definition *dg-CAT* :: $V \Rightarrow V$
  where *dg-CAT* α =
    [
      set $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}$,
      all-cfs α,
      $(\lambda\mathfrak{F}{\in_\circ}$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$,
      $(\lambda\mathfrak{F}{\in_\circ}$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
    ]$_\circ$

Components.

lemma *dg-CAT-components*:
  shows *dg-CAT* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}$
    and *dg-CAT* $\alpha(\!|$*Arr*$|\!)$ = *all-cfs* α
    and *dg-CAT* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}{\in_\circ}$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$
    and *dg-CAT* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}{\in_\circ}$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
  unfolding *dg-CAT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

20.3 Object

lemma *dg-CAT-ObjI*:
  assumes *category* α $\mathfrak{A}$
  shows $\mathfrak{A}$ $\in_\circ$ *dg-CAT* $\alpha(\!|$*Obj*$|\!)$
  using *assms* **unfolding** *dg-CAT-components* **by** *auto*

lemma *dg-CAT-ObjD*:
  assumes $\mathfrak{A}$ $\in_\circ$ *dg-CAT* $\alpha(\!|$*Obj*$|\!)$
  shows *category* α $\mathfrak{A}$
  using *assms* **unfolding** *dg-CAT-components* **by** *auto*

lemma *dg-CAT-ObjE*:
  assumes $\mathfrak{A}$ $\in_\circ$ *dg-CAT* $\alpha(\!|$*Obj*$|\!)$
  obtains *category* α $\mathfrak{A}$
  using *assms* **unfolding** *dg-CAT-components* **by** *auto*

lemma *dg-CAT-Obj-iff*[*dg-CAT-simps*]: $\mathfrak{A}$ $\in_\circ$ *dg-CAT* $\alpha(\!|$*Obj*$|\!)$ $\longleftrightarrow$ *category* α $\mathfrak{A}$
  unfolding *dg-CAT-components* **by** *auto*

20.4 Domain and codomain

lemma [*dg-CAT-simps*]:
  assumes $\mathfrak{F}$ $\in_\circ$ *all-cfs* α
  shows *dg-CAT-Dom-app*: *dg-CAT* $\alpha(\!|$*Dom*$|\!)(\!|\mathfrak{F}|\!)$ = $\mathfrak{F}(\!|$*HomDom*$|\!)$
    and *dg-CAT-Cod-app*: *dg-CAT* $\alpha(\!|$*Cod*$|\!)(\!|\mathfrak{F}|\!)$ = $\mathfrak{F}(\!|$*HomCod*$|\!)$
  using *assms* **unfolding** *dg-CAT-components* **by** *auto*

20.5 *CAT* is a digraph

lemma (**in** $\mathcal{Z}$) *tiny-category-dg-CAT*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows *tiny-digraph* β (*dg-CAT* α)
proof(*intro tiny-digraphI*)
interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(*1*))
show *vfsequence* (*dg-CAT* α) **unfolding** *dg-CAT-def* **by** *simp*
show *vcard* (*dg-CAT* α) = $\mathcal{4}_{\mathbb{N}}$
unfolding *dg-CAT-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*dg-CAT* α(|*Dom*|)) $\subseteq_{\circ}$ *dg-CAT* α(|*Obj*|)
proof(*intro vsubsetI*)
fix $\mathfrak{A}$ **assume** $\mathfrak{A} \in_{\circ} \mathcal{R}_{\circ}$ (*dg-CAT* α(|*Dom*|))
then obtain $\mathfrak{F}$ **where** $\mathfrak{F} \in_{\circ}$ *all-cfs* α **and** $\mathfrak{A}$ = $\mathfrak{F}$(|*HomDom*|)
unfolding *dg-CAT-components* **by** *auto*
then obtain $\mathfrak{B}$ $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
unfolding *dg-CAT-components* **by** *auto*
then interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** *simp*
show $\mathfrak{A} \in_{\circ}$ *dg-CAT* α(|*Obj*|)
by (*simp add*: *dg-CAT-components HomDom.category-axioms*)
qed
show $\mathcal{R}_{\circ}$ (*dg-CAT* α(|*Cod*|)) $\subseteq_{\circ}$ *dg-CAT* α(|*Obj*|)
proof(*intro vsubsetI*)
fix $\mathfrak{B}$ **assume** $\mathfrak{B} \in_{\circ} \mathcal{R}_{\circ}$ (*dg-CAT* α(|*Cod*|))
then obtain $\mathfrak{F}$ **where** $\mathfrak{F} \in_{\circ} \mathcal{D}_{\circ}$ (*dg-CAT* α(|*Cod*|)) **and** $\mathfrak{B}$ = $\mathfrak{F}$(|*HomCod*|)
unfolding *dg-CAT-components* **by** *auto*
then obtain $\mathfrak{A}$ $\mathfrak{F}$
where $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{B}$-*def*: $\mathfrak{B}$ = $\mathfrak{F}$(|*HomCod*|)
unfolding *dg-CAT-components* **by** *auto*
have $\mathfrak{B}$ = $\mathfrak{F}$(|*HomCod*|) **unfolding** $\mathfrak{B}$-*def* **by** *simp*
interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
show $\mathfrak{B} \in_{\circ}$ *dg-CAT* α(|*Obj*|)
by (*simp add*: *HomCod.category-axioms dg-CAT-components*)
qed
show *dg-CAT* α(|*Obj*|) $\in_{\circ}$ *Vset* β
unfolding *dg-CAT-components* **by** (*rule categories-in-Vset*[*OF assms*])
show *dg-CAT* α(|*Arr*|) $\in_{\circ}$ *Vset* β
unfolding *dg-CAT-components* **by** (*rule all-cfs-in-Vset*[*OF assms*])
qed (*simp-all add*: *assms dg-CAT-components*)

20.6 Arrow with a domain and a codomain

lemma *dg-CAT-is-arrI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}CAT\ \alpha} \mathfrak{B}$
proof(*intro is-arrI*, *unfold dg-CAT-components*(*2*))
interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*)
from *assms* **show** $\mathfrak{F} \in_{\circ}$ *all-cfs* α **by** *auto*
with *assms* **show** *dg-CAT* α(|*Dom*|)(|$\mathfrak{F}$|) = $\mathfrak{A}$ *dg-CAT* α(|*Cod*|)(|$\mathfrak{F}$|) = $\mathfrak{B}$
by (*simp-all add*: *dg-CAT-components cat-cs-simps*)
qed

lemma *dg-CAT-is-arrD*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}CAT\ \alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
using *assms* **by** (*elim is-arrE*) (*auto simp*: *dg-CAT-components*)

lemma *dg-CAT-is-arrE*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}CAT\ \alpha} \mathfrak{B}$

obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
using *assms* **by** (*simp add*: *dg-CAT-is-arrD*)

lemma *dg-CAT-is-arr-iff*[*dg-CAT-simps*]:
$\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}CAT\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*auto intro*: *dg-CAT-is-arrI dest*: *dg-CAT-is-arrD*)

21 *CAT* as a semicategory

21.1 Background

The subsection presents the theory of the semicategories of α-categories. It continues the development that was initiated in section 20.

named-theorems *smc-CAT-simps*
named-theorems *smc-CAT-intros*

21.2 Definition and elementary properties

definition *smc-CAT* :: $V \Rightarrow V$
  where *smc-CAT* α =
    [
      set $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}$,
      all-cfs α,
      $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!|))$,
      $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!|))$,
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-CAT* α)$.$ $\mathfrak{GF}(\!|0|\!|) \circ_{CF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
    ]$_\circ$

Components.

lemma *smc-CAT-components*:
  shows *smc-CAT* $\alpha(\!|$*Obj*$|\!|)$ = *set* $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}$
    and *smc-CAT* $\alpha(\!|$*Arr*$|\!|)$ = *all-cfs* α
    and *smc-CAT* $\alpha(\!|$*Dom*$|\!|)$ = $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!|))$
    and *smc-CAT* $\alpha(\!|$*Cod*$|\!|)$ = $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!|))$
    and *smc-CAT* $\alpha(\!|$*Comp*$|\!|)$ = $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-CAT* α)$.$ $\mathfrak{GF}(\!|0|\!|) \circ_{CF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
  unfolding *smc-CAT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-CAT*: *smc-dg* (*smc-CAT* α) = *dg-CAT* α
proof(*rule vsv-eqI*)
  show *vsv* (*smc-dg* (*smc-CAT* α)) **unfolding** *smc-dg-def* **by** *auto*
  show *vsv* (*dg-CAT* α) **unfolding** *dg-CAT-def* **by** *auto*
  have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-CAT* α)) = $4_{\mathbb{N}}$
    unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-CAT* α) = $4_{\mathbb{N}}$
    unfolding *dg-CAT-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*smc-dg* (*smc-CAT* α)) = $\mathcal{D}_\circ$ (*dg-CAT* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show $\mathfrak{A} \in_\circ \mathcal{D}_\circ$ (*smc-dg* (*smc-CAT* α)) $\Longrightarrow$ *smc-dg* (*smc-CAT* α)$(\!|\mathfrak{A}|\!)$ = *dg-CAT* $\alpha(\!|\mathfrak{A}|\!)$
    for $\mathfrak{A}$
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold smc-dg-def dg-field-simps smc-CAT-def dg-CAT-def
)
      (*auto simp*: *nat-omega-simps*)
qed

lemmas-with [*folded smc-dg-CAT*, *unfolded slicing-simps*]:
  smc-CAT-ObjI = *dg-CAT-ObjI*
  and *smc-CAT-ObjD* = *dg-CAT-ObjD*
  and *smc-CAT-ObjE* = *dg-CAT-ObjE*
  and *smc-CAT-Obj-iff*[*smc-CAT-simps*] = *dg-CAT-Obj-iff*

and *smc-CAT-Dom-app*[*smc-CAT-simps*] = *dg-CAT-Dom-app*
and *smc-CAT-Cod-app*[*smc-CAT-simps*] = *dg-CAT-Cod-app*
and *smc-CAT-is-arrI* = *dg-CAT-is-arrI*
and *smc-CAT-is-arrD* = *dg-CAT-is-arrD*
and *smc-CAT-is-arrE* = *dg-CAT-is-arrE*
and *smc-CAT-is-arr-iff*[*smc-CAT-simps*] = *dg-CAT-is-arr-iff*

21.3 Composable arrows

lemma *smc-CAT-composable-arrs-dg-CAT*:
composable-arrs (*dg-CAT* α) = *composable-arrs* (*smc-CAT* α)
unfolding *composable-arrs-def smc-dg-CAT*[*symmetric*] *slicing-simps* **by** *auto*

lemma *smc-CAT-Comp*:
smc-CAT $\alpha(\!|Comp|\!)$ = $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*smc-CAT* α). $\mathfrak{GF}(\!|0|\!) \circ_{SMCF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
unfolding *smc-CAT-components smc-CAT-composable-arrs-dg-CAT* **..**

21.4 Composition

lemma *smc-CAT-Comp-app*[*smc-CAT-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto_{smc\text{-}CAT\ \alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto_{smc\text{-}CAT\ \alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{A\,smc\text{-}CAT\ \alpha} \mathfrak{F} = \mathfrak{G} \circ_{SMCF} \mathfrak{F}$
proof–
from *assms* **have** $[\mathfrak{G}, \mathfrak{F}]_\circ \in_\circ$ *composable-arrs* (*smc-CAT* α)
by (*auto simp*: *smc-cs-intros*)
then show $\mathfrak{G} \circ_{A\,smc\text{-}CAT\ \alpha} \mathfrak{F} = \mathfrak{G} \circ_{SMCF} \mathfrak{F}$
unfolding *smc-CAT-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-CAT-Comp-vdomain*: $\mathcal{D}_\circ$ (*smc-CAT* $\alpha(\!|Comp|\!)$) = *composable-arrs* (*smc-CAT* α)
unfolding *smc-CAT-Comp* **by** *auto*

lemma *smc-CAT-Comp-vrange*: $\mathcal{R}_\circ$ (*smc-CAT* $\alpha(\!|Comp|\!)$) $\subseteq_\circ$ *all-cfs* α
proof(*rule vsubsetI*)
fix $\mathfrak{H}$ **assume** $\mathfrak{H} \in_\circ \mathcal{R}_\circ$ (*smc-CAT* $\alpha(\!|Comp|\!)$)
then obtain $\mathfrak{GF}$
where $\mathfrak{H}$-*def*: $\mathfrak{H}$ = *smc-CAT* $\alpha(\!|Comp|\!)(\!|\mathfrak{GF}|\!)$
and $\mathfrak{GF} \in_\circ \mathcal{D}_\circ$ (*smc-CAT* $\alpha(\!|Comp|\!)$)
by (*auto simp*: *smc-CAT-components intro*: *smc-cs-intros*)
then obtain $\mathfrak{G}$ $\mathfrak{F}$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
where $\mathfrak{GF} = [\mathfrak{G}, \mathfrak{F}]_\circ$
and $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{B} \mapsto_{smc\text{-}CAT\ \alpha} \mathfrak{C}$
and $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto_{smc\text{-}CAT\ \alpha} \mathfrak{B}$
by (*auto simp*: *smc-CAT-Comp-vdomain*)
with $\mathfrak{H}$-*def* **have** $\mathfrak{H}$-*def*′: $\mathfrak{H} = \mathfrak{G} \circ_{A\,smc\text{-}CAT\ \alpha} \mathfrak{F}$ **by** *simp*
from $\mathfrak{G}$ $\mathfrak{F}$ **show** $\mathfrak{H} \in_\circ$ *all-cfs* α
unfolding $\mathfrak{H}$-*def*′ **by** (*auto simp*: *smc-CAT-simps intro*: *cat-cs-intros*)
qed

21.5 *CAT* is a category

lemma (**in** $\mathcal{Z}$) *tiny-semicategory-smc-CAT*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *tiny-semicategory* β (*smc-CAT* α)
proof(*intro tiny-semicategoryI*, *unfold smc-CAT-is-arr-iff*)
show *vfsequence* (*smc-CAT* α) **unfolding** *smc-CAT-def* **by** *auto*
show *vcard* (*smc-CAT* α) = $5_{\mathbb{N}}$
unfolding *smc-CAT-def* **by** (*simp add*: *nat-omega-simps*)

show $(\mathfrak{GF} \in_\circ \mathcal{D}_\circ\ (\textit{smc-CAT}\ \alpha(\!|\textit{Comp}|\!)))\ \longleftrightarrow$
$(\exists \mathfrak{G}\ \mathfrak{F}\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{A}.\ \mathfrak{GF} = [\mathfrak{G}, \mathfrak{F}]_\circ \wedge \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C} \wedge \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B})$
for $\mathfrak{GF}$
unfolding *smc-CAT-Comp-vdomain*
proof
show $\mathfrak{GF} \in_\circ$ *composable-arrs* (*smc-CAT* α) $\Longrightarrow$
$\exists \mathfrak{G}\ \mathfrak{F}\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{A}.\ \mathfrak{GF} = [\mathfrak{G}, \mathfrak{F}]_\circ \wedge \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C} \wedge \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by (*elim composable-arrsE*) (*auto simp*: *smc-CAT-is-arr-iff*)
next
assume $\exists \mathfrak{G}\ \mathfrak{F}\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{A}.\ \mathfrak{GF} = [\mathfrak{G}, \mathfrak{F}]_\circ \wedge \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C} \wedge \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
with *smc-CAT-is-arr-iff* **show** $\mathfrak{GF} \in_\circ$ *composable-arrs* (*smc-CAT* α)
unfolding *smc-CAT-Comp-vdomain* **by** (*auto intro*: *smc-cs-intros*)
qed
show $[\![\ \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C};\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\ ]\!] \Longrightarrow$
$\mathfrak{G} \circ_{A\,\textit{smc-CAT}\ \alpha} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
for $\mathfrak{G}\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{A}$
by (*cs-concl* **cs-simp**: *smc-CAT-simps* **cs-intro**: *cat-cs-intros*)

fix $\mathfrak{H}\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{G}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{A}$
assume $\mathfrak{H} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{D}\ \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
moreover then have $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}\ \mathfrak{H} \circ_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{D}$
by (*cs-concl* **cs-simp**: *smc-CAT-simps* **cs-intro**: *cat-cs-intros*)+
ultimately show
$\mathfrak{H} \circ_{A\,\textit{smc-CAT}\ \alpha} \mathfrak{G} \circ_{A\,\textit{smc-CAT}\ \alpha} \mathfrak{F} = \mathfrak{H} \circ_{A\,\textit{smc-CAT}\ \alpha} (\mathfrak{G} \circ_{A\,\textit{smc-CAT}\ \alpha} \mathfrak{F})$
by (*simp add*: *smc-CAT-is-arr-iff smc-CAT-Comp-app cf-comp-assoc*)
qed (*auto simp*: *assms smc-dg-CAT tiny-category-dg-CAT smc-CAT-components*)

21.6 Initial object

lemma (**in** $\mathcal{Z}$) *smc-CAT-obj-initialI*: *obj-initial* (*smc-CAT* α) *cat-0*
— See [1][11]).
unfolding *obj-initial-def*
proof(*intro obj-terminalI*, *unfold smc-op-simps smc-CAT-is-arr-iff smc-CAT-Obj-iff*)
show *category* α *cat-0* **by** (*intro category-cat-0*)
fix $\mathfrak{A}$ **assume** *category* α $\mathfrak{A}$
then interpret *category* α $\mathfrak{A}$ **.**
show $\exists! f.\ f$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
proof
show *cf-0*: *cf-0* $\mathfrak{A}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
by (*simp add*: *cf-0-is-ft-functor category-axioms is-ft-functor.axioms*(*1*))
fix $\mathfrak{F}$ **assume** *prems*: $\mathfrak{F}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
interpret $\mathfrak{F}$: *is-functor* α *cat-0* $\mathfrak{A}$ $\mathfrak{F}$ **using** *prems* **.**
show $\mathfrak{F}$ = *cf-0* $\mathfrak{A}$
proof(*rule cf-eqI*)
show $\mathfrak{F}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$ **by** (*simp add*: *prems*)
from *cf-0* **show** *cf-0* $\mathfrak{A}$: *cat-0* $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
unfolding *smc-CAT-is-arr-iff* **by** *simp*
have $\mathcal{D}_\circ$ ($\mathfrak{F}$(|*ObjMap*|)) = *0* **by** (*auto simp*: *cat-0-components cat-cs-simps*)
then show $\mathfrak{F}$(|*ObjMap*|) = *cf-0* $\mathfrak{A}$(|*ObjMap*|)
using $\mathfrak{F}$.*ObjMap.vbrelation-vintersection-vdomain*
by (*auto simp*: *cf-0-components*)
have $\mathcal{D}_\circ$ ($\mathfrak{F}$(|*ArrMap*|)) = *0* **by** (*auto simp*: *cat-0-components cat-cs-simps*)
with $\mathfrak{F}$.*ArrMap.vbrelation-vintersection-vdomain* **show**
$\mathfrak{F}$(|*ArrMap*|) = *cf-0* $\mathfrak{A}$(|*ArrMap*|)
by (*auto simp*: *cf-0-components*)
qed (*simp-all add*: *cf-0-components*)
qed

[11] https://ncatlab.org/nlab/show/initial+object

qed

lemma (**in** $\mathcal{Z}$) *smc-CAT-obj-initialD*:
assumes *obj-initial* (*smc-CAT* α) $\mathfrak{A}$
shows $\mathfrak{A}$ = *cat-0*
using *assms* **unfolding** *obj-initial-def*
proof(*elim obj-terminalE*, *unfold smc-op-simps smc-CAT-is-arr-iff smc-CAT-Obj-iff*)
assume *prems*:
category α $\mathfrak{A}$
category α $\mathfrak{B}$ $\Longrightarrow$ $\exists!\mathfrak{F}.$ $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
for $\mathfrak{B}$
from *prems*(*2*)[*OF category-cat-0*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha}$ *cat-0*
by *meson*
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{A}$ *cat-0* $\mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
have $\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ObjMap|\!)$) $\subseteq_\circ$ *0*
unfolding *cat-0-components*(*1*)[*symmetric*] **by** (*simp add*: $\mathfrak{F}$.*cf-ObjMap-vrange*)
then have $\mathfrak{F}(\!|ObjMap|\!)$ = *0* **by** (*auto intro*: $\mathfrak{F}$.*ObjMap.vsv-vrange-vempty*)
with $\mathfrak{F}$.*cf-ObjMap-vdomain* **have** *Obj*[*simp*]: $\mathfrak{A}(\!|Obj|\!)$ = *0* **by** *auto*
have $\mathcal{R}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$) $\subseteq_\circ$ *0*
unfolding *cat-0-components*(*2*)[*symmetric*] **by** (*simp add*: $\mathfrak{F}$.*cf-ArrMap-vrange*)
then have $\mathfrak{F}(\!|ArrMap|\!)$ = *0* **by** (*auto intro*: $\mathfrak{F}$.*ArrMap.vsv-vrange-vempty*)
with $\mathfrak{F}$.*cf-ArrMap-vdomain* **have** *Arr*[*simp*]: $\mathfrak{A}(\!|Arr|\!)$ = *0* **by** *auto*
from $\mathfrak{F}$.*HomDom.Dom.vdomain-vrange-is-vempty* **have** [*simp*]: $\mathfrak{A}(\!|Dom|\!)$ = *0*
by (*fastforce simp*: $\mathfrak{F}$.*HomDom.cat-Dom-vempty-if-Arr-vempty*)
from $\mathfrak{F}$.*HomDom.Cod.vdomain-vrange-is-vempty* **have** [*simp*]: $\mathfrak{A}(\!|Cod|\!)$ = *0*
by (*fastforce simp*: $\mathfrak{F}$.*HomDom.cat-Cod-vempty-if-Arr-vempty*)
from *Arr* **have** $\mathfrak{A}(\!|Arr|\!)$ $\widehat{}_\times$ $2_\mathbb{N}$ = *0* **by** (*simp add*: *vcpower-of-vempty*)
with $\mathfrak{F}$.*HomDom.Comp.pnop-vdomain* **have** $\mathcal{D}_\circ$ ($\mathfrak{A}(\!|Comp|\!)$) = *0* **by** *simp*
with $\mathfrak{F}$.*HomDom.Comp.vdomain-vrange-is-vempty* **have** [*simp*]: $\mathfrak{A}(\!|Comp|\!)$ = *0*
by (*auto intro*: $\mathfrak{F}$.*HomDom.Comp.vsv-vrange-vempty*)
have $\mathcal{D}_\circ$ ($\mathfrak{A}(\!|CId|\!)$) = *0*
by (*simp add*: $\mathfrak{F}$.*HomDom.cat-CId-vdomain*)
with $\mathfrak{F}$.*HomDom.CId.vdomain-vrange-is-vempty* $\mathfrak{F}$.*HomDom.CId.vsv-vrange-vempty*
have [*simp*]: $\mathfrak{A}(\!|CId|\!)$ = *0*
by *simp*
show $\mathfrak{A}$ = *cat-0*
by (*rule cat-eqI*[*of* α])
(*simp-all add*: *prems*(*1*) *cat-0-components category-cat-0*)
qed

lemma (**in** $\mathcal{Z}$) *smc-CAT-obj-initialE*:
assumes *obj-initial* (*smc-CAT* α) $\mathfrak{A}$
obtains $\mathfrak{A}$ = *cat-0*
using *assms* **by** (*auto dest*: *smc-CAT-obj-initialD*)

lemma (**in** $\mathcal{Z}$) *smc-CAT-obj-initial-iff*[*smc-CAT-simps*]:
obj-initial (*smc-CAT* α) $\mathfrak{A}$ $\longleftrightarrow$ $\mathfrak{A}$ = *cat-0*
using *smc-CAT-obj-initialI smc-CAT-obj-initialD* **by** *auto*

21.7 Terminal object

lemma (**in** $\mathcal{Z}$) *smc-CAT-obj-terminalI*:
— See [1][12].
assumes $a \in_\circ$ *Vset* α **and** $f \in_\circ$ *Vset* α
shows *obj-terminal* (*smc-CAT* α) (*cat-1 a f*)
proof(*intro obj-terminalI*, *unfold smc-op-simps smc-CAT-is-arr-iff smc-CAT-Obj-iff*)
fix $\mathfrak{A}$ **assume** *prems*: *category* α $\mathfrak{A}$

[12] https://ncatlab.org/nlab/show/terminal+object

```
  then interpret category α 𝔄 .
  show ∃!𝔉'. 𝔉' : 𝔄 ↦↦_Cα cat-1 a f
  proof
    show cf-1: cf-const 𝔄 (cat-1 a f) a : 𝔄 ↦↦_Cα cat-1 a f
      by (rule cf-const-is-functor)
        (auto simp: assms prems category-cat-1 cat-1-components)
    fix 𝔉' assume 𝔉' : 𝔄 ↦↦_Cα cat-1 a f
    then interpret 𝔉': is-functor α 𝔄 ‹cat-1 a f› 𝔉' .
    show 𝔉' = cf-const 𝔄 (cat-1 a f) a
    proof(rule cf-eqI, unfold dghm-const-components)
      from cf-1 show cf-const 𝔄 (cat-1 a f) a : 𝔄 ↦↦_Cα cat-1 a f by simp
      show 𝔉'(|ObjMap|) = vconst-on (𝔄(|Obj|)) a
      proof(cases‹𝔄(|Obj|) = 0›)
        case True
        with 𝔉'.ObjMap.vbrelation-vintersection-vdomain have 𝔉'(|ObjMap|) = 0
          by (auto simp: cat-cs-simps)
        with True show ?thesis by simp
      next
        case False
        then have 𝒟ₒ (𝔉'(|ObjMap|)) ≠ 0 by (auto simp: cat-cs-simps)
        then have ℛₒ (𝔉'(|ObjMap|)) ≠ 0
          by (simp add: 𝔉'.ObjMap.vsv-vdomain-vempty-vrange-vempty)
        moreover from 𝔉'.cf-ObjMap-vrange have ℛₒ (𝔉'(|ObjMap|)) ⊆ₒ set {a}
          by (simp add: cat-1-components)
        ultimately have ℛₒ (𝔉'(|ObjMap|)) = set {a} by auto
        then show ?thesis
          by (intro vsv.vsv-is-vconst-onI) (auto simp: cat-cs-simps)
      qed
      show 𝔉'(|ArrMap|) = vconst-on (𝔄(|Arr|)) (cat-1 a f(|CId|)(|a|))
      proof(cases‹𝔄(|Arr|) = 0›)
        case True
        with
          𝔉'.ArrMap.vdomain-vrange-is-vempty
          vsv.vsv-vrange-vempty[OF 𝔉'.cf-ArrMap-vsv]
        have 𝔉'(|ArrMap|) = 0
          by (auto simp: cat-cs-simps)
        with True show ?thesis by simp
      next
        case False
        then have 𝒟ₒ (𝔉'(|ArrMap|)) ≠ 0 by (auto simp: cat-cs-simps)
        then have ℛₒ (𝔉'(|ArrMap|)) ≠ 0
          by (simp add: 𝔉'.ArrMap.vsv-vdomain-vempty-vrange-vempty)
        moreover from 𝔉'.cf-ArrMap-vrange have ℛₒ (𝔉'(|ArrMap|)) ⊆ₒ set {f}
          by (simp add: cat-1-components)
        ultimately have ℛₒ (𝔉'(|ArrMap|)) = set {f} by auto
        then show ?thesis
          by
            (
              cs-concl cs-shallow
                cs-simp: V-cs-simps cat-cs-simps cat-1-components
                cs-intro: V-cs-intros vsv.vsv-is-vconst-onI
            )+
      qed
    qed (auto intro: cat-cs-intros)
  qed
qed (simp add: assms category-cat-1)

lemma (in 𝒵) smc-CAT-obj-terminalE:
```

```
  assumes obj-terminal (smc-CAT α) 𝔅
  obtains a f where a ∈∘ Vset α and f ∈∘ Vset α and 𝔅 = cat-1 a f
  using assms
proof(elim obj-terminalE, unfold cat-op-simps smc-CAT-is-arr-iff smc-CAT-Obj-iff)

  assume prems: category α 𝔅 category α 𝔄 ⟹ ∃!𝔉. 𝔉 : 𝔄 ↦↦_Cα 𝔅 for 𝔄
  interpret 𝔅: category α 𝔅 by (rule prems(1))

  obtain a where 𝔅-Obj: 𝔅(|Obj|) = set {a} and a: a ∈∘ Vset α
  proof–
    have cat-1: category α (cat-1 0 0) by (rule category-cat-1) auto
    from prems(2)[OF cat-1] obtain 𝔉
      where 𝔉: 𝔉 : cat-1 0 0 ↦↦_Cα 𝔅
        and 𝔊𝔉: 𝔊 : cat-1 0 0 ↦↦_Cα 𝔅 ⟹ 𝔊 = 𝔉 for 𝔊
      by fastforce
    interpret 𝔉: is-functor α ‹cat-1 0 0› 𝔅 𝔉 by (rule 𝔉)
    have 𝒟∘ (𝔉(|ObjMap|)) = set {0} by (simp add: cat-1-components cat-cs-simps)
    then obtain a where vrange-𝔉[simp]: ℛ∘ (𝔉(|ObjMap|)) = set {a}
      by (auto intro: 𝔉.ObjMap.vsv-vdomain-vsingleton-vrange-vsingleton)
    with 𝔅.cat-Obj-vsubset-Vset 𝔉.cf-ObjMap-vrange have [simp]: a ∈∘ Vset α
      by auto
    from 𝔉.cf-ObjMap-vrange have set {a} ⊆∘ 𝔅(|Obj|) by simp
    moreover have 𝔅(|Obj|) ⊆∘ set {a}
    proof(rule ccontr)
      assume ¬𝔅(|Obj|) ⊆∘ set {a}
      then obtain b where ba: b ≠ a and b: b ∈∘ 𝔅(|Obj|) by force
      have cf-const (cat-1 0 0) 𝔅 b : cat-1 0 0 ↦↦_Cα 𝔅
        by (rule cf-const-is-functor)
          (simp-all add: 𝔅.category-axioms category-cat-1 b)
      then have 𝔊-def: cf-const (cat-1 0 0) 𝔅 b = 𝔉 by (rule 𝔊𝔉)
      have ℛ∘ (cf-const (cat-1 0 0) 𝔅 b(|ObjMap|)) = set {b}
        unfolding dghm-const-components cat-1-components by simp
      with vrange-𝔉 ba show False unfolding 𝔊-def by simp
    qed
    ultimately have 𝔅(|Obj|) = set {a} by simp
    with that show ?thesis by simp
  qed

  have 𝔅-Arr: 𝔅(|Arr|) = set {𝔅(|CId|)(|a|)}
  proof(rule vsubset-antisym)
    from 𝔅-Obj show set {𝔅(|CId|)(|a|)} ⊆∘ 𝔅(|Arr|)
      by (blast intro: 𝔅.cat-is-arrD(1) cat-cs-intros)
    show 𝔅(|Arr|) ⊆∘ set {𝔅(|CId|)(|a|)}
    proof(intro vsubsetI)
      fix f assume f ∈∘ 𝔅(|Arr|)
      with 𝔅-Obj have f: f : a ↦_𝔅 a
        by (metis 𝔅.cat-is-arrD(2,3) is-arrI vsingleton-iff)
      from f have cf-const 𝔅 𝔅 a : 𝔅 ↦↦_Cα 𝔅
        by (intro cf-const-is-functor) (auto simp: 𝔅.category-axioms)
      moreover from f have cf-id 𝔅 : 𝔅 ↦↦_Cα 𝔅
        by (intro category.cat-cf-id-is-functor)
          (auto simp: 𝔅.category-axioms)
      ultimately have cf-const 𝔅 𝔅 a = cf-id 𝔅
        by (metis prems(2) 𝔅.category-axioms)
      moreover from f have cf-const 𝔅 𝔅 a(|ArrMap|)(|f|) = 𝔅(|CId|)(|a|)
        by (simp add: ‹f ∈∘ 𝔅(|Arr|)› dghm-const-ArrMap-app)
      moreover from f have cf-id 𝔅(|ArrMap|)(|f|) = f
        unfolding dghm-id-components by (simp add: cat-cs-intros)
```

```
    ultimately show f ∈∘ set {𝔅(|CId|)(|a|)} by simp
  qed
qed

have 𝔅 = cat-1 a (𝔅(|CId|)(|a|))
proof(rule cat-eqI[of α], unfold cat-1-components)
  from 𝔅.cat-Arr-vsubset-Vset 𝔅-Arr show category α (cat-1 a (𝔅(|CId|)(|a|)))
    by (intro category-cat-1) (auto simp: a)
  show 𝔅(|Arr|) = set {𝔅(|CId|)(|a|)} by (simp add: 𝔅-Arr)
  then have 𝒟∘ (𝔅(|Dom|)) = set {𝔅(|CId|)(|a|)}
    by (simp add: cat-cs-simps cat-cs-intros)
  moreover have ℛ∘ (𝔅(|Dom|)) = set {a}
    using 𝔅.cat-Dom-vrange 𝔅.cat-CId-is-arr 𝔅.cat-Dom-vdomain
    by (auto simp: 𝔅-Obj elim: 𝔅.Dom.vbrelation-vinE)
  ultimately show 𝔅(|Dom|) = set {⟨𝔅(|CId|)(|a|), a⟩}
    using 𝔅.Dom.vsv-vdomain-vrange-vsingleton by simp
  have 𝒟∘ (𝔅(|Cod|)) = set {𝔅(|CId|)(|a|)}
    by (simp add: 𝔅-Arr cat-cs-simps)
  moreover have ℛ∘ (𝔅(|Cod|)) = set {a}
    using 𝔅.cat-Cod-vrange 𝔅.cat-CId-is-arr 𝔅.cat-Cod-vdomain
    by (auto simp: 𝔅-Obj elim: 𝔅.Cod.vbrelation-vinE)
  ultimately show 𝔅(|Cod|) = set {⟨𝔅(|CId|)(|a|), a⟩}
    by (auto intro: 𝔅.Cod.vsv-vdomain-vrange-vsingleton)
  show 𝔅(|Comp|) = set {⟨[𝔅(|CId|)(|a|), 𝔅(|CId|)(|a|)]∘, 𝔅(|CId|)(|a|)⟩}
  proof(rule vsv-eqI)
    show dom:
      𝒟∘ (𝔅(|Comp|)) = 𝒟∘ (set {⟨[𝔅(|CId|)(|a|), 𝔅(|CId|)(|a|)]∘, 𝔅(|CId|)(|a|)⟩})
      unfolding vdomain-vsingleton
    proof(rule vsubset-antisym)
      show 𝒟∘ (𝔅(|Comp|)) ⊆∘ set {[𝔅(|CId|)(|a|), 𝔅(|CId|)(|a|)]∘}
        by (intro vsubsetI)
          (metis 𝔅.cat-Comp-vdomain 𝔅-Arr vsingleton-iff 𝔅.cat-is-arrD(1))
      show set {[𝔅(|CId|)(|a|), 𝔅(|CId|)(|a|)]∘} ⊆∘ 𝒟∘ (𝔅(|Comp|))
        by
          (
            metis
              𝔅-Obj vsingleton-iff
              𝔅.cat-CId-is-arr
              𝔅.cat-Comp-vdomainI
              vsubset-vsingleton-left
          )
    qed
    have 𝔅(|CId|)(|a|) ∘A𝔅 𝔅(|CId|)(|a|) = 𝔅(|CId|)(|a|)
      by (metis 𝔅-Obj 𝔅.cat-CId-is-arr 𝔅.cat-CId-left-left vsingletonI)
    then show a′ ∈∘ 𝒟∘ (𝔅(|Comp|)) ⟹
      𝔅(|Comp|)(|a′|) = set {⟨[𝔅(|CId|)(|a|), 𝔅(|CId|)(|a|)]∘, 𝔅(|CId|)(|a|)⟩}(|a′|)
      for a′
      unfolding dom by simp
  qed (auto simp: 𝔅-Obj 𝔅-Arr)
  have 𝒟∘ (𝔅(|CId|)) = set {a} by (simp add: 𝔅-Obj 𝔅.cat-CId-vdomain)
  moreover then have ℛ∘ (𝔅(|CId|)) = set {𝔅(|CId|)(|a|)}
    by
      (
        metis
          𝔅.CId.vdomain-atE
          𝔅.CId.vsv-vdomain-vsingleton-vrange-vsingleton
          vsingleton-iff
      )
```

ultimately show $\mathfrak{B}(\lfloor CId \rfloor) = set\ \{\langle a, \mathfrak{B}(\lfloor CId \rfloor)(\lfloor a \rfloor)\rangle\}$
by (*blast intro*: $\mathfrak{B}$.*CId.vsv-vdomain-vrange-vsingleton*)
qed (*auto simp*: $\mathfrak{B}$-*Obj cat-cs-intros*)

with *a that* $\mathfrak{B}$.*cat-Arr-vsubset-Vset* $\mathfrak{B}$-*Arr* **show** *?thesis* **by** *auto*

qed

22 *CAT*

22.1 Background

The subsection presents the theory of the categories of α-categories. It continues the development that was initiated in sections 20-21.

named-theorems *cat-CAT-simps*
named-theorems *cat-CAT-intros*

22.2 Definition and elementary properties

definition *cat-CAT* :: $V \Rightarrow V$
  where *cat-CAT* α =
    [
      set $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}$,
      all-cfs α,
      $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$,
      $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$,
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-CAT* α). $\mathfrak{GF}(\!|0|\!) \circ_{CF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$,
      $(\lambda\mathfrak{C}\in_\circ$*set* $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}.$ *cf-id* $\mathfrak{C})$
    ]$_\circ$

Components.

lemma *cat-CAT-components*:
  shows *cat-CAT* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}$
    and *cat-CAT* $\alpha(\!|$*Arr*$|\!)$ = *all-cfs* α
    and *cat-CAT* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$
    and *cat-CAT* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-cfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
    and *cat-CAT* $\alpha(\!|$*Comp*$|\!)$ =
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-CAT* α). $\mathfrak{GF}(\!|0|\!) \circ_{CF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
    and *cat-CAT* $\alpha(\!|$*CId*$|\!)$ = $(\lambda\mathfrak{C}\in_\circ$*set* $\{\mathfrak{C}.$ *category* α $\mathfrak{C}\}.$ *cf-id* $\mathfrak{C})$
  unfolding *cat-CAT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-CAT*: *cat-smc* (*cat-CAT* α) = *smc-CAT* α
proof(*rule vsv-eqI*)
  have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-CAT* α)) = $5_{\mathbb{N}}$
    unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-CAT* α) = $5_{\mathbb{N}}$
    unfolding *smc-CAT-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*cat-smc* (*cat-CAT* α)) = $\mathcal{D}_\circ$ (*smc-CAT* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show *a* $\in_\circ$ $\mathcal{D}_\circ$ (*cat-smc* (*cat-CAT* α)) $\Longrightarrow$ *cat-smc* (*cat-CAT* α)$(\!|a|\!)$ = *smc-CAT* $\alpha(\!|a|\!)$
    for *a*
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold cat-smc-def dg-field-simps cat-CAT-def smc-CAT-def
)
      (*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *cat-smc-def smc-CAT-def*)

lemmas-with [*folded cat-smc-CAT*, *unfolded slicing-simps*]:
  — Digraph
  cat-CAT-ObjI = *smc-CAT-ObjI*
  and *cat-CAT-ObjD* = *smc-CAT-ObjD*

and *cat-CAT-ObjE* = *smc-CAT-ObjE*
and *cat-CAT-Obj-iff*[*cat-CAT-simps*] = *smc-CAT-Obj-iff*
and *cat-CAT-Dom-app*[*cat-CAT-simps*] = *smc-CAT-Dom-app*
and *cat-CAT-Cod-app*[*cat-CAT-simps*] = *smc-CAT-Cod-app*
and *cat-CAT-is-arrI* = *smc-CAT-is-arrI*
and *cat-CAT-is-arrD* = *smc-CAT-is-arrD*
and *cat-CAT-is-arrE* = *smc-CAT-is-arrE*
and *cat-CAT-is-arr-iff*[*cat-CAT-simps*] = *smc-CAT-is-arr-iff*

lemmas-with [*folded cat-smc-CAT*, *unfolded slicing-simps*, *unfolded cat-smc-CAT*]:
— Semicategory
cat-CAT-Comp-vdomain = *smc-CAT-Comp-vdomain*
and *cat-CAT-composable-arrs-dg-CAT* = *smc-CAT-composable-arrs-dg-CAT*
and *cat-CAT-Comp* = *smc-CAT-Comp*
and *cat-CAT-Comp-app*[*cat-CAT-simps*] = *smc-CAT-Comp-app*
and *cat-CAT-Comp-vrange* = *smc-CAT-Comp-vrange*

lemmas-with (**in** $\mathcal{Z}$) [*folded cat-smc-CAT*, *unfolded slicing-simps*]:
— Semicategory
cat-CAT-obj-initialI = *smc-CAT-obj-initialI*
and *cat-CAT-obj-initialD* = *smc-CAT-obj-initialD*
and *cat-CAT-obj-initialE* = *smc-CAT-obj-initialE*
and *cat-CAT-obj-initial-iff*[*cat-CAT-simps*] = *smc-CAT-obj-initial-iff*
and *cat-CAT-obj-terminalI* = *smc-CAT-obj-terminalI*
and *cat-CAT-obj-terminalE* = *smc-CAT-obj-terminalE*

22.3 Identity

lemma *cat-CAT-CId-app*[*cat-CAT-simps*]:
assumes *category* α $\mathfrak{C}$
shows *cat-CAT* $\alpha(\!|CId|\!)(\!|\mathfrak{C}|\!)$ = *cf-id* $\mathfrak{C}$
using *assms* **unfolding** *cat-CAT-components* **by** *simp*

lemma *cat-CAT-CId-vdomain*: $\mathcal{D}_\circ$ (*cat-CAT* $\alpha(\!|CId|\!)$) = *set* $\{\mathfrak{C}.\ category\ \alpha\ \mathfrak{C}\}$
unfolding *cat-CAT-components* **by** *auto*

lemma *cat-CAT-CId-vrange*: $\mathcal{R}_\circ$ (*cat-CAT* $\alpha(\!|CId|\!)$) $\subseteq_\circ$ *all-cfs* α
proof(*rule vsubsetI*)
fix $\mathfrak{H}$ **assume** $\mathfrak{H} \in_\circ \mathcal{R}_\circ$ (*cat-CAT* $\alpha(\!|CId|\!)$)
then obtain $\mathfrak{A}$
where $\mathfrak{H}$-*def*: $\mathfrak{H}$ = *cat-CAT* $\alpha(\!|CId|\!)(\!|\mathfrak{A}|\!)$
and $\mathfrak{A}$: $\mathfrak{A} \in_\circ \mathcal{D}_\circ$ (*cat-CAT* $\alpha(\!|CId|\!)$)
unfolding *cat-CAT-components* **by** *auto*
from $\mathfrak{A}$ **have** $\mathfrak{H}$-*def'*: $\mathfrak{H}$ = *cf-id* $\mathfrak{A}$
unfolding $\mathfrak{H}$-*def cat-CAT-CId-vdomain* **by** (*auto simp*: *cat-CAT-CId-app*)
from $\mathfrak{A}$ *category.cat-cf-id-is-functor* **show** $\mathfrak{H} \in_\circ$ *all-cfs* α
unfolding $\mathfrak{H}$-*def'* *cat-CAT-CId-vdomain* **by** *force*
qed

22.4 *CAT* is a category

lemma (**in** $\mathcal{Z}$) *tiny-category-cat-CAT*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *tiny-category* β (*cat-CAT* α)
proof(*intro tiny-categoryI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *vfsequence* (*cat-CAT* α) **unfolding** *cat-CAT-def* **by** *simp*
show *vcard* (*cat-CAT* α) = $6_\mathbb{N}$

```
    unfolding cat-CAT-def by (simp add: nat-omega-simps)
  show 𝔉 : 𝔄 ↦_cat-CAT α 𝔅 ⟹ cat-CAT α(|CId|)(|𝔅|) ∘_A cat-CAT α 𝔉 = 𝔉
    for 𝔉 𝔄 𝔅
  proof–
    assume prems: 𝔉 : 𝔄 ↦_cat-CAT α 𝔅
    then have b: category α 𝔅 unfolding cat-CAT-is-arr-iff by auto
    with digraph.dg-dghm-id-is-dghm have
      cat-CAT α(|CId|)(|𝔅|) : 𝔅 ↦_cat-CAT α 𝔅
      by
        (
          simp add:
            cat-CAT-CId-app cat-CAT-is-arrI category.cat-cf-id-is-functor
        )
    with prems b show cat-CAT α(|CId|)(|𝔅|) ∘_A cat-CAT α 𝔉 = 𝔉
      by
        (
          simp add:
            cat-CAT-CId-app
            cat-CAT-Comp-app
            cat-CAT-is-arr-iff
            is-functor.cf-cf-comp-cf-id-left
        )
  qed
  show 𝔉 : 𝔅 ↦_cat-CAT α ℭ ⟹ 𝔉 ∘_A cat-CAT α cat-CAT α(|CId|)(|𝔅|) = 𝔉
    for 𝔉 𝔅 ℭ
  proof–
    assume prems: 𝔉 : 𝔅 ↦_cat-CAT α ℭ
    then have b: category α 𝔅 unfolding cat-CAT-is-arr-iff by auto
    then have cat-CAT α(|CId|)(|𝔅|) : 𝔅 ↦_cat-CAT α 𝔅
      by
        (
          simp add:
            cat-CAT-CId-app cat-CAT-is-arrI category.cat-cf-id-is-functor
        )
    with prems b show 𝔉 ∘_A cat-CAT α cat-CAT α(|CId|)(|𝔅|) = 𝔉
      by
        (
          auto
            simp: cat-CAT-CId-app cat-CAT-Comp-app cat-CAT-is-arr-iff
            intro: is-functor.cf-cf-comp-cf-id-right
        )
  qed
qed
  (
    simp-all add:
      assms
      cat-smc-CAT
      cat-CAT-components
      𝒵.intro
      𝒵-Limit-αω
      𝒵-ω-αω
      cat-CAT-is-arr-iff
      tiny-semicategory-smc-CAT
      category.cat-cf-id-is-functor
  )

lemmas [cat-cs-intros] = 𝒵.tiny-category-cat-CAT
```

22.5 Isomorphism

lemma *cat-CAT-is-iso-arrI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}CAT\ \alpha} \mathfrak{B}$
proof(*intro is-iso-arrI is-inverseI*)
from *assms* **show** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}CAT\ \alpha} \mathfrak{B}$
unfolding *cat-CAT-is-arr-iff* **by** *auto*
note *iso-thms* = *is-iso-functor-is-iso-arr*[*OF assms*]
from *iso-thms*(*1*) **show** *inv-*$\mathfrak{F}$: *inv-cf* $\mathfrak{F} : \mathfrak{B} \mapsto_{cat\text{-}CAT\ \alpha} \mathfrak{A}$
unfolding *cat-CAT-is-arr-iff* **by** *auto*
from *assms* **show** $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}CAT\ \alpha} \mathfrak{B}$
unfolding *cat-CAT-is-arr-iff* **by** *auto*
from *assms* **have** $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** *auto*
show *inv-cf* $\mathfrak{F} \circ_{A cat\text{-}CAT\ \alpha} \mathfrak{F}$ = *cat-CAT* α(|*CId*|)(|$\mathfrak{A}$|)
unfolding *cat-CAT-CId-app*[*OF* $\mathfrak{A}$] *cat-CAT-Comp-app*[*OF inv-*$\mathfrak{F}$ $\mathfrak{F}$]
by (*rule iso-thms*(*2*))
show $\mathfrak{F} \circ_{A cat\text{-}CAT\ \alpha}$ *inv-cf* $\mathfrak{F}$ = *cat-CAT* α(|*CId*|)(|$\mathfrak{B}$|)
unfolding *cat-CAT-CId-app*[*OF* $\mathfrak{B}$] *cat-CAT-Comp-app*[*OF* $\mathfrak{F}$ *inv-*$\mathfrak{F}$]
by (*rule iso-thms*(*3*))
qed

lemma *cat-CAT-is-iso-arrD*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}CAT\ \alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
proof–
from *is-iso-arrD*[*OF assms*] **have** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto_{cat\text{-}CAT\ \alpha} \mathfrak{B}$
and ($\exists \mathfrak{G}$. *is-inverse* (*cat-CAT* α) $\mathfrak{G}$ $\mathfrak{F}$)
by *simp-all*
then obtain $\mathfrak{G}$ **where** *is-inverse* (*cat-CAT* α) $\mathfrak{G}$ $\mathfrak{F}$ **by** *clarsimp*
then obtain $\mathfrak{A}'$ $\mathfrak{B}'$ **where** $\mathfrak{G}'$: $\mathfrak{G} : \mathfrak{B}' \mapsto_{cat\text{-}CAT\ \alpha} \mathfrak{A}'$
and $\mathfrak{F}'$: $\mathfrak{F} : \mathfrak{A}' \mapsto_{cat\text{-}CAT\ \alpha} \mathfrak{B}'$
and $\mathfrak{G}\mathfrak{F}$: $\mathfrak{G} \circ_{A cat\text{-}CAT\ \alpha} \mathfrak{F}$ = *cat-CAT* α(|*CId*|)(|$\mathfrak{A}'$|)
and $\mathfrak{F}\mathfrak{G}$: $\mathfrak{F} \circ_{A cat\text{-}CAT\ \alpha} \mathfrak{G}$ = *cat-CAT* α(|*CId*|)(|$\mathfrak{B}'$|)
by *auto*
from $\mathfrak{F}$ $\mathfrak{F}'$ **have** $\mathfrak{A}'$: $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$: $\mathfrak{B}'$ = $\mathfrak{B}$ **by** *auto*
from $\mathfrak{F}$ **have** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **unfolding** *cat-CAT-is-arr-iff* **by** *simp*
then have $\mathfrak{A}$: *category* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** *auto*
from $\mathfrak{G}'$ **have** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{A}$ **unfolding** $\mathfrak{A}'$ $\mathfrak{B}'$ *cat-CAT-is-arr-iff* **by** *simp*
moreover from $\mathfrak{G}\mathfrak{F}$ **have** $\mathfrak{G} \circ_{CF} \mathfrak{F}$ = *cf-id* $\mathfrak{A}$
unfolding $\mathfrak{A}'$ *cat-CAT-Comp-app*[*OF* $\mathfrak{G}'$ $\mathfrak{F}'$] *cat-CAT-CId-app*[*OF* $\mathfrak{A}$] **by** *simp*
moreover from $\mathfrak{F}\mathfrak{G}$ **have** $\mathfrak{F} \circ_{CF} \mathfrak{G}$ = *cf-id* $\mathfrak{B}$
unfolding $\mathfrak{B}'$ *cat-CAT-Comp-app*[*OF* $\mathfrak{F}'$ $\mathfrak{G}'$] *cat-CAT-CId-app*[*OF* $\mathfrak{B}$] **by** *simp*
ultimately show *?thesis* **using** $\mathfrak{F}$ **by** (*elim is-iso-arr-is-iso-functor*)
qed

lemma *cat-CAT-is-iso-arrE*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}CAT\ \alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
using *assms* **by** (*auto dest*: *cat-CAT-is-iso-arrD*)

lemma *cat-CAT-is-iso-arr-iff*[*cat-CAT-simps*]:
$\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}CAT\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.iso\alpha} \mathfrak{B}$
using *cat-CAT-is-iso-arrI cat-CAT-is-iso-arrD* **by** *auto*

22.6 Isomorphic objects

lemma *cat-CAT-obj-isoI*:
assumes $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$

shows $\mathfrak{A} \approx_{obj cat\text{-}CAT\ \alpha} \mathfrak{B}$
proof–
from *iso-categoryD*[*OF assms*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\!\mapsto_{C.iso\alpha} \mathfrak{B}$
by *clarsimp*
from *cat-CAT-is-iso-arrI*[*OF this*] **show** *?thesis* **by** (*rule obj-isoI*)
qed

lemma *cat-CAT-obj-isoD*:
assumes $\mathfrak{A} \approx_{obj cat\text{-}CAT\ \alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
proof–
from *obj-isoD*[*OF assms*] **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto_{iso cat\text{-}CAT\ \alpha} \mathfrak{B}$
by *clarsimp*
from *cat-CAT-is-iso-arrD*[*OF this*] **show** *?thesis* **by** (*rule iso-categoryI*)
qed

lemma *cat-CAT-obj-isoE*:
assumes $\mathfrak{A} \approx_{obj cat\text{-}CAT\ \alpha} \mathfrak{B}$
obtains $\mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
using *assms* **by** (*auto simp*: *cat-CAT-obj-isoD*)

lemma *cat-CAT-obj-iso-iff*[*cat-CAT-simps*]:
$\mathfrak{A} \approx_{obj cat\text{-}CAT\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{A} \approx_{C\alpha} \mathfrak{B}$
using *cat-CAT-obj-isoI cat-CAT-obj-isoD* **by** (*intro iffI*) *auto*

23 *FUNCT* and *Funct* as digraphs

23.1 Background

A general reference for this section is Chapter II-4 in [7].

named-theorems *dg-FUNCT-cs-simps*
named-theorems *dg-FUNCT-cs-intros*
named-theorems *cat-map-cs-simps*
named-theorems *cat-map-cs-intros*
named-theorems *cat-map-extra-cs-simps*

23.2 Functor map

23.2.1 Definition and elementary properties

definition *cf-map* :: $V \Rightarrow V$
  where *cf-map* $\mathfrak{F}$ = $[\mathfrak{F}(\!|ObjMap|\!), \mathfrak{F}(\!|ArrMap|\!)]_\circ$

abbreviation *cf-maps* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$ ≡ *set* $\{$*cf-map* $\mathfrak{F}$ | $\mathfrak{F}$. $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$

abbreviation *tm-cf-maps* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$ ≡ *set* $\{$*cf-map* $\mathfrak{F}$ | $\mathfrak{F}$. $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

lemma *tm-cf-maps-subset-cf-maps*:
  $\{$*cf-map* $\mathfrak{F}$ | $\mathfrak{F}$. $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\} \subseteq \{$*cf-map* $\mathfrak{F}$ | $\mathfrak{F}$. $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$
  by *auto*

Components.

lemma *cf-map-components*[*cat-map-cs-simps*]:
  shows *cf-map* $\mathfrak{F}(\!|ObjMap|\!)$ = $\mathfrak{F}(\!|ObjMap|\!)$
    and *cf-map* $\mathfrak{F}(\!|ArrMap|\!)$ = $\mathfrak{F}(\!|ArrMap|\!)$
  unfolding *cf-map-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Sequence characterization.

lemma *dg-FUNCT-Obj-components*:
  shows $[FOM, FAM]_\circ(\!|ObjMap|\!)$ = *FOM*
    and $[FOM, FAM]_\circ(\!|ArrMap|\!)$ = *FAM*
  unfolding *dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *cf-map-vfsequence*[*cat-map-cs-intros*]: *vfsequence* (*cf-map* $\mathfrak{F}$)
  unfolding *cf-map-def* **by** *auto*

lemma *cf-map-vdomain*[*cat-map-cs-simps*]: $\mathcal{D}_\circ$ (*cf-map* $\mathfrak{F}$) = $2_\mathbb{N}$
  unfolding *cf-map-def* **by** (*simp add*: *nat-omega-simps*)

lemma (**in** *is-functor*) *cf-map-vsubset-cf*: *cf-map* $\mathfrak{F} \subseteq_\circ \mathfrak{F}$
  by (*unfold cf-map-def*, *subst* (*3*) *cf-def*)
    (*cs-concl* **cs-shallow cs-intro**: *vcons-vsubset′ V-cs-intros*)

Size.

lemma (**in** *is-functor*) *cf-map-ObjMap-in-Vset*:
  assumes $\alpha \in_\circ \beta$
  shows *cf-map* $\mathfrak{F}(\!|ObjMap|\!) \in_\circ$ *Vset* β
  using *assms* **unfolding** *cf-map-components* **by** (*intro cf-ObjMap-in-Vset*)

lemma (**in** *is-tm-functor*) *tm-cf-map-ObjMap-in-Vset*: *cf-map* $\mathfrak{F}(\!|ObjMap|\!) \in_\circ$ *Vset* α

unfolding *cf-map-components* **by** (*rule tm-cf-ObjMap-in-Vset*)

lemma (**in** *is-functor*) *cf-map-ArrMap-in-Vset*:
assumes $\alpha \in_\circ \beta$
shows *cf-map* $\mathfrak{F}(\!|ArrMap|\!) \in_\circ$ *Vset* β
using *assms* **unfolding** *cf-map-components* **by** (*intro cf-ArrMap-in-Vset*)

lemma (**in** *is-tm-functor*) *tm-cf-map-ArrMap-in-Vset*: *cf-map* $\mathfrak{F}(\!|ArrMap|\!) \in_\circ$ *Vset* α
unfolding *cf-map-components* **by** (*rule tm-cf-ArrMap-in-Vset*)

lemma (**in** *is-functor*) *cf-map-in-Vset-4*: *cf-map* $\mathfrak{F} \in_\circ$ *Vset* $(\alpha + 4_{\mathbb{N}})$
proof–
note [*folded VPow-iff*, *folded Vset-succ*[*OF Ord-α*], *cat-cs-intros*] =
cf-ObjMap-vsubset-Vset
cf-ArrMap-vsubset-Vset
show *?thesis*
by (*subst cf-map-def*, *succ-of-numeral*)
(
cs-concl
cs-simp: *plus-V-succ-right V-cs-simps*
cs-intro: *cat-cs-intros V-cs-intros*
)
qed

lemma (**in** *is-tm-functor*) *tm-cf-map-in-Vset*: *cf-map* $\mathfrak{F} \in_\circ$ *Vset* α
using *tm-cf-ObjMap-in-Vset tm-cf-ArrMap-in-Vset* **unfolding** *cf-map-def*
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros*)

lemma (**in** *is-functor*) *cf-map-in-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *cf-map* $\mathfrak{F} \in_\circ$ *Vset* β
using *assms cf-map-in-Vset-4 cf-map-vsubset-cf*
by (*auto intro*!: *cf-in-Vset*)

lemma *cf-maps-subset-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows $\{$*cf-map* $\mathfrak{F} \mid \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\} \subseteq$ *elts* (*Vset* β)
proof(*intro subsetI*, *unfold mem-Collect-eq*, *elim exE conjE*)
fix x $\mathfrak{F}$ **assume** *x-def*: $x =$ *cf-map* $\mathfrak{F}$ **and** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
show $x \in_\circ$ *Vset* β **unfolding** *x-def* **by** (*rule cf-map-in-Vset*[*OF assms*])
qed

lemma *small-cf-maps*[*simp*]: *small* $\{$*cf-map* $\mathfrak{F} \mid \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$
proof(*cases* ‹$\mathcal{Z}$ α›)
case *True*
from *is-functor.cf-map-in-Vset* **show** *?thesis*
by (*intro down*[*of* - ‹*Vset* $(\alpha + \omega)$›])
(*auto simp*: *True* $\mathcal{Z}$.*Limit-αω* $\mathcal{Z}$.$\mathcal{Z}$-*ω-αω* $\mathcal{Z}$.*intro* $\mathcal{Z}$.$\mathcal{Z}$-*α-αω*)
next
case *False*
then have $\{$*cf-map* $\mathfrak{F} \mid \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\} = \{\}$ **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma *small-tm-cf-maps*[*simp*]: *small* $\{$*cf-map* $\mathfrak{F} \mid \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$
by (*rule smaller-than-small*[*OF small-cf-maps tm-cf-maps-subset-cf-maps*])

lemma (**in** $\mathcal{Z}$) *cf-maps-in-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$ $\in_\circ$ *Vset* β
proof(*rule vsubset-in-VsetI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$ $\subseteq_\circ$ *Vset* (α + $\mathcal{4}_{\mathbb{N}}$)
proof(*intro vsubsetI*)
fix $\mathfrak{F}$ **assume** $\mathfrak{F} \in_\circ$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
then obtain $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}'$ **where** $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-map* $\mathfrak{F}'$ **and** $\mathfrak{F}$: $\mathfrak{F}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by *auto*
interpret *is-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}'$ **using** $\mathfrak{F}$ **by** *simp*
show $\mathfrak{F} \in_\circ$ *Vset* (α + $\mathcal{4}_{\mathbb{N}}$) **unfolding** $\mathfrak{F}$-*def* **by** (*rule cf-map-in-Vset-4*)
qed
from *assms*(*2*) **show** *Vset* (α + $\mathcal{4}_{\mathbb{N}}$) $\in_\circ$ *Vset* β
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros Ord-cs-intros*)
qed

lemma (**in** $\mathcal{Z}$) *tm-cf-maps-vsubset-Vset*: *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$ $\subseteq_\circ$ *Vset* α
proof(*intro vsubsetI*)
fix $\mathfrak{F}$ **assume** $\mathfrak{F} \in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
then obtain $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}'$
where $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-map* $\mathfrak{F}'$ **and** $\mathfrak{F}$: $\mathfrak{F}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
by *auto*
then show $\mathfrak{F} \in_\circ$ *Vset* α **by** (*force simp*: *is-tm-functor.tm-cf-map-in-Vset*)
qed

Rules.

lemma (**in** *is-functor*) *cf-mapsI*: *cf-map* $\mathfrak{F} \in_\circ$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
by (*auto intro*: *cat-cs-intros*)

lemma (**in** *is-tm-functor*) *tm-cf-mapsI*: *cf-map* $\mathfrak{F} \in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
by (*auto intro*: *cat-small-cs-intros*)

lemma (**in** *is-functor*) *cf-mapsI'*:
assumes $\mathfrak{F}'$ = *cf-map* $\mathfrak{F}$
shows $\mathfrak{F}' \in_\circ$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding *assms* **by** (*rule cf-mapsI*)

lemma (**in** *is-tm-functor*) *tm-cf-mapsI'*:
assumes $\mathfrak{F}'$ = *cf-map* $\mathfrak{F}$
shows $\mathfrak{F}' \in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding *assms* **by** (*rule tm-cf-mapsI*)

lemmas [*cat-map-cs-intros*] =
is-functor.cf-mapsI

lemmas *cf-mapsI'*[*cat-map-cs-intros*] =
is-functor.cf-mapsI'[*rotated*]

lemmas [*cat-map-cs-intros*] =
is-tm-functor.tm-cf-mapsI

lemmas *tm-cf-mapsI'*[*cat-map-cs-intros*] =
is-tm-functor.tm-cf-mapsI'[*rotated*]

lemma *cf-mapsE*[*elim*]:
assumes $\mathfrak{F} \in_\circ$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
obtains $\mathfrak{G}$ **where** $\mathfrak{F}$ = *cf-map* $\mathfrak{G}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

using *assms* **by** *force*

lemma *tm-cf-mapsE*[*elim*]:
assumes $\mathfrak{F} \in_{\circ} \textit{tm-cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
obtains $\mathfrak{G}$ **where** $\mathfrak{F} = \textit{cf-map}\ \mathfrak{G}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
using *assms* **by** *force*

The opposite functor map.

lemma (**in** *is-functor*) *cf-map-op-cf*[*cat-op-simps*]: $\textit{cf-map}\ (\textit{op-cf}\ \mathfrak{F}) = \textit{cf-map}\ \mathfrak{F}$
proof(*rule vsv-eqI*, *unfold cat-map-cs-simps*)
show $a \in_{\circ} 2_{\mathbb{N}} \Longrightarrow \textit{cf-map}\ (\textit{op-cf}\ \mathfrak{F})(\!|a|\!) = \textit{cf-map}\ \mathfrak{F}(\!|a|\!)$ **for** *a*
by
(
elim-in-numeral,
unfold dghm-field-simps[*symmetric*] *cf-map-components cat-op-simps*
)
simp-all
qed (*auto intro*: *cat-map-cs-intros*)

lemmas [*cat-op-simps*] = *is-functor.cf-map-op-cf*

Elementary properties.

lemma *tm-cf-maps-vsubset-cf-maps*: $\textit{tm-cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B} \subseteq_{\circ} \textit{cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
using *tm-cf-maps-subset-cf-maps* **by** *simp*

lemma *tm-cf-maps-in-cf-maps*:
assumes $\mathfrak{F} \in_{\circ} \textit{tm-cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
shows $\mathfrak{F} \in_{\circ} \textit{cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
using *assms tm-cf-maps-vsubset-cf-maps*[*of* $\alpha\ \mathfrak{A}\ \mathfrak{B}$] **by** *blast*

lemma *cf-map-inj*:
assumes $\textit{cf-map}\ \mathfrak{F} = \textit{cf-map}\ \mathfrak{G}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{F} = \mathfrak{G}$
proof(*rule cf-eqI*)
from *assms*(*1*) **have** *ObjMap*: $\textit{cf-map}\ \mathfrak{F}(\!|\textit{ObjMap}|\!) = \textit{cf-map}\ \mathfrak{G}(\!|\textit{ObjMap}|\!)$
and *ArrMap*: $\textit{cf-map}\ \mathfrak{F}(\!|\textit{ArrMap}|\!) = \textit{cf-map}\ \mathfrak{G}(\!|\textit{ArrMap}|\!)$
by *auto*
from *ObjMap* **show** $\mathfrak{F}(\!|\textit{ObjMap}|\!) = \mathfrak{G}(\!|\textit{ObjMap}|\!)$ **unfolding** *cf-map-components* **by** *simp*
from *ArrMap* **show** $\mathfrak{F}(\!|\textit{ArrMap}|\!) = \mathfrak{G}(\!|\textit{ArrMap}|\!)$ **unfolding** *cf-map-components* **by** *simp*
qed (*auto intro*: *assms*(*2,3*))

lemma *cf-map-eq-iff*[*cat-map-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\textit{cf-map}\ \mathfrak{F} = \textit{cf-map}\ \mathfrak{G} \longleftrightarrow \mathfrak{F} = \mathfrak{G}$
using *cf-map-inj*[*OF - assms*] **by** *auto*

lemma *cf-map-eqI*:
assumes $\mathfrak{F} \in_{\circ} \textit{cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
and $\mathfrak{G} \in_{\circ} \textit{cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
and $\mathfrak{F}(\!|\textit{ObjMap}|\!) = \mathfrak{G}(\!|\textit{ObjMap}|\!)$
and $\mathfrak{F}(\!|\textit{ArrMap}|\!) = \mathfrak{G}(\!|\textit{ArrMap}|\!)$
shows $\mathfrak{F} = \mathfrak{G}$
proof–
from *assms*(*1*) **obtain** $\mathfrak{F}'$
where $\mathfrak{F}$-*def*: $\mathfrak{F} = \textit{cf-map}\ \mathfrak{F}'$ **and** $\mathfrak{F}'$: $\mathfrak{F}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by *auto*
from *assms*(*2*) **obtain** $\mathfrak{G}'$
where $\mathfrak{G}$-*def*: $\mathfrak{G} = \textit{cf-map}\ \mathfrak{G}'$ **and** $\mathfrak{G}'$: $\mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$

by *auto*
show *?thesis*
proof(*rule vsv-eqI*, *unfold* $\mathfrak{F}$*-def* $\mathfrak{G}$*-def*)
show $a \in_{\circ} \mathcal{D}_{\circ}$ (*cf-map* $\mathfrak{F}'$) $\Longrightarrow$ *cf-map* $\mathfrak{F}'(\lfloor a \rfloor)$ = *cf-map* $\mathfrak{G}'(\lfloor a \rfloor)$ **for** a
by
(
unfold cf-map-vdomain,
elim-in-numeral,
insert assms(*3*,*4*),
unfold $\mathfrak{F}$*-def* $\mathfrak{G}$*-def*
)
(*auto simp*: *dghm-field-simps*)
qed (*auto simp*: *cat-map-cs-simps intro*: *cat-map-cs-intros*)
qed

23.3 Conversion of a functor map to a functor

23.3.1 Definition and elementary properties

definition *cf-of-cf-map* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ = $[\mathfrak{F}(\lfloor ObjMap \rfloor), \mathfrak{F}(\lfloor ArrMap \rfloor), \mathfrak{A}, \mathfrak{B}]_{\circ}$

Components.

lemma *cf-of-cf-map-components*:
shows *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\lfloor ObjMap \rfloor)$ = $\mathfrak{F}(\lfloor ObjMap \rfloor)$
and *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\lfloor ArrMap \rfloor)$ = $\mathfrak{F}(\lfloor ArrMap \rfloor)$
and *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\lfloor HomDom \rfloor)$ = $\mathfrak{A}$
and *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}(\lfloor HomCod \rfloor)$ = $\mathfrak{B}$
unfolding *cf-of-cf-map-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemmas [*cat-map-extra-cs-simps*] = *cf-of-cf-map-components*(*1–2*)
lemmas [*cat-map-cs-simps*] = *cf-of-cf-map-components*(*3–4*)

23.3.2 The conversion of a functor map to a functor is a functor

lemma (**in** *is-functor*) *cf-of-cf-map-is-functor*:
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$) : $\mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof(*rule is-functorI'*)
show *vfsequence* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$))
unfolding *cf-of-cf-map-def* **by** *simp*
show *vcard* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)) = $4_{\mathbb{N}}$
unfolding *cf-of-cf-map-def* **by** (*simp add*: *nat-omega-simps*)
show
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)$(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$:
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)$(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$ $\mapsto_{\mathfrak{B}}$
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)$(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
if $f : a \mapsto_{\mathfrak{A}} b$ **for** a b f
unfolding *cf-of-cf-map-components cf-map-components*
using *is-functor-axioms that*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)$(\lfloor ArrMap \rfloor)(\lfloor g \circ_{A\mathfrak{A}} f \rfloor)$ =
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)$(\lfloor ArrMap \rfloor)(\lfloor g \rfloor)$ $\circ_{A\mathfrak{B}}$
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)$(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$
if $g : b \mapsto_{\mathfrak{A}} c$ **and** $f : a \mapsto_{\mathfrak{A}} b$ **for** b c g a f
using *is-functor-axioms that*
unfolding *cf-of-cf-map-components cf-map-components*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
show

cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ArrMap*|)(|$\mathfrak{A}$(|*CId*|)(|*c*|)|) =
$\mathfrak{B}$(|*CId*|)(|*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ObjMap*|)(|*c*|)|)
if *c* $\in_\circ$ $\mathfrak{A}$(|*Obj*|) **for** *c*
using *is-functor-axioms that*
unfolding *cf-of-cf-map-components cf-map-components*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
qed
(
auto simp:
cat-cs-simps
cf-of-cf-map-components
cf-map-components
cf-ObjMap-vrange
intro: *cat-cs-intros*
)

lemma (**in** *is-functor*) *cf-of-cf-map-is-functor'*:
assumes $\mathfrak{F}'$ = *cf-map* $\mathfrak{F}$
and $\mathfrak{A}'$ = $\mathfrak{A}$
and $\mathfrak{B}'$ = $\mathfrak{B}$
shows *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}'$: $\mathfrak{A}'$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule cf-of-cf-map-is-functor*)

lemmas [*cat-map-cs-intros*] = *is-functor.cf-of-cf-map-is-functor'*

23.3.3 The value of the conversion of a functor map to a functor

lemma (**in** *is-functor*) *cf-of-cf-map-of-cf-map*[*cat-map-cs-simps*]:
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$) = $\mathfrak{F}$
proof(*rule cf-eqI*)
show *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$) : $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
proof(*rule is-functorI'*)
show *vfsequence* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$))
unfolding *cf-of-cf-map-def* **by** *auto*
show *vcard* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)) = $4_{\mathbb{N}}$
unfolding *cf-of-cf-map-def* **by** (*simp add*: *nat-omega-simps*)
show
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ArrMap*|)(|*f*|) :
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ObjMap*|)(|*a*|) $\mapsto_{\mathfrak{B}}$
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ObjMap*|)(|*b*|)
if *f* : *a* $\mapsto_{\mathfrak{A}}$ *b* **for** *a b f*
unfolding *cf-of-cf-map-components cf-map-components*
using *is-functor-axioms that*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ArrMap*|)(|*g* $\circ_{A\mathfrak{A}}$ *f*|) =
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ArrMap*|)(|*g*|) $\circ_{A\mathfrak{B}}$
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ArrMap*|)(|*f*|)
if *g* : *b* $\mapsto_{\mathfrak{A}}$ *c* **and** *f* : *a* $\mapsto_{\mathfrak{A}}$ *b* **for** *b c g a f*
unfolding *cf-of-cf-map-components cf-map-components*
using *is-functor-axioms that*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
show
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ArrMap*|)(|$\mathfrak{A}$(|*CId*|)(|*c*|)|) =
$\mathfrak{B}$(|*CId*|)(|*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$)(|*ObjMap*|)(|*c*|)|)
if *c* $\in_\circ$ $\mathfrak{A}$(|*Obj*|) **for** *c*
unfolding *cf-of-cf-map-components cf-map-components*
using *is-functor-axioms that*

by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
qed
(
auto simp:
cat-cs-simps
cf-of-cf-map-components
cf-map-components
cf-ObjMap-vrange
intro: *cat-cs-intros*
)
qed (*auto simp*: *cf-of-cf-map-components cf-map-components intro*: *cat-cs-intros*)

lemmas [*cat-map-cs-simps*] = *is-functor.cf-of-cf-map-of-cf-map*

23.4 Natural transformation arrow

23.4.1 Definition and elementary properties

definition *ntcf-arrow* :: $V \Rightarrow V$
where *ntcf-arrow* $\mathfrak{N}$ = $[\mathfrak{N}(\lfloor NTMap \rfloor),\ cf\text{-}map\ (\mathfrak{N}(\lfloor NTDom \rfloor)),\ cf\text{-}map\ (\mathfrak{N}(\lfloor NTCod \rfloor))]_\circ$

abbreviation *ntcf-arrows* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$ ≡
set $\{ntcf\text{-}arrow\ \mathfrak{N} \mid \mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$

abbreviation *tm-ntcf-arrows* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *tm-ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$ ≡
set $\{ntcf\text{-}arrow\ \mathfrak{N} \mid \mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\}$

lemma *tm-ntcf-arrows-subset-ntcf-arrows*:
$\{ntcf\text{-}arrow\ \mathfrak{N} \mid \mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}\} \subseteq$
$\{ntcf\text{-}arrow\ \mathfrak{N} \mid \mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}\}$
by *auto*

Components.

lemma *ntcf-arrow-components*:
shows [*cat-map-cs-simps*]: $ntcf\text{-}arrow\ \mathfrak{N}(\lfloor NTMap \rfloor) = \mathfrak{N}(\lfloor NTMap \rfloor)$
and $ntcf\text{-}arrow\ \mathfrak{N}(\lfloor NTDom \rfloor) = cf\text{-}map\ (\mathfrak{N}(\lfloor NTDom \rfloor))$
and $ntcf\text{-}arrow\ \mathfrak{N}(\lfloor NTCod \rfloor) = cf\text{-}map\ (\mathfrak{N}(\lfloor NTCod \rfloor))$
unfolding *ntcf-arrow-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma (**in** *is-ntcf*) *ntcf-arrow-components'*:
shows $ntcf\text{-}arrow\ \mathfrak{N}(\lfloor NTMap \rfloor) = \mathfrak{N}(\lfloor NTMap \rfloor)$
and $ntcf\text{-}arrow\ \mathfrak{N}(\lfloor NTDom \rfloor) = cf\text{-}map\ \mathfrak{F}$
and $ntcf\text{-}arrow\ \mathfrak{N}(\lfloor NTCod \rfloor) = cf\text{-}map\ \mathfrak{G}$
unfolding *ntcf-arrow-components ntcf-NTDom ntcf-NTCod* **by** *simp-all*

lemmas [*cat-map-cs-simps*] = *is-ntcf.ntcf-arrow-components'*(*2*,*3*)

Elementary properties.

lemma *dg-FUNCT-Arr-components*:
shows $[NTM,\ NTD,\ NTC]_\circ(\lfloor NTMap \rfloor) = NTM$
and $[NTM,\ NTD,\ NTC]_\circ(\lfloor NTDom \rfloor) = NTD$
and $[NTM,\ NTD,\ NTC]_\circ(\lfloor NTCod \rfloor) = NTC$
unfolding *nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *ntcf-arrow-vfsequence*[*cat-map-cs-intros*]: *vfsequence* (*ntcf-arrow* $\mathfrak{N}$)
unfolding *ntcf-arrow-def* **by** *simp*

lemma *ntcf-arrow-vdomain*[*cat-map-cs-simps*]: $\mathcal{D}_\circ$ (*ntcf-arrow* $\mathfrak{N}$) = $\mathcal{3}_\mathbf{N}$
unfolding *ntcf-arrow-def* **by** (*simp add*: *nat-omega-simps*)

Size.

lemma (**in** *is-ntcf*) *ntcf-arrow-NTMap-in-Vset*:
assumes $\alpha \in_\circ \beta$
shows *ntcf-arrow* $\mathfrak{N}$(|*NTMap*|) $\in_\circ$ *Vset* β
using *assms* **unfolding** *ntcf-arrow-components* **by** (*intro ntcf-NTMap-in-Vset*)

lemma (**in** *is-tm-ntcf*) *tm-ntcf-arrow-NTMap-in-Vset*:
ntcf-arrow $\mathfrak{N}$(|*NTMap*|) $\in_\circ$ *Vset* α
unfolding *ntcf-arrow-components* **by** (*rule tm-ntcf-NTMap-in-Vset*)

lemma (**in** *is-ntcf*) *ntcf-arrow-NTDom-in-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *ntcf-arrow* $\mathfrak{N}$(|*NTDom*|) $\in_\circ$ *Vset* β
using *assms* **unfolding** *ntcf-arrow-components'* **by** (*rule NTDom.cf-map-in-Vset*)

lemma (**in** *is-tm-ntcf*) *tm-ntcf-arrow-NTDom-in-Vset*:
ntcf-arrow $\mathfrak{N}$(|*NTDom*|) $\in_\circ$ *Vset* α
unfolding *ntcf-arrow-components'* **by** (*rule NTDom.tm-cf-map-in-Vset*)

lemma (**in** *is-ntcf*) *ntcf-arrow-NTCod-in-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *ntcf-arrow* $\mathfrak{N}$(|*NTCod*|) $\in_\circ$ *Vset* β
using *assms* **unfolding** *ntcf-arrow-components'* **by** (*rule NTCod.cf-map-in-Vset*)

lemma (**in** *is-tm-ntcf*) *tm-ntcf-arrow-NTCod-in-Vset*:
ntcf-arrow $\mathfrak{N}$(|*NTCod*|) $\in_\circ$ *Vset* α
unfolding *ntcf-arrow-components'* **by** (*rule NTCod.tm-cf-map-in-Vset*)

lemma (**in** *is-ntcf*) *ntcf-arrow-in-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *ntcf-arrow* $\mathfrak{N} \in_\circ$ *Vset* β
proof–
interpret *ntcf-arrow*: *vfsequence* ‹*ntcf-arrow* $\mathfrak{N}$›
by (*auto intro*: *cat-map-cs-intros*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *?thesis*
proof(*rule vsv.vsv-Limit-vsv-in-VsetI*)
from *assms* **show** $\mathcal{D}_\circ$ (*ntcf-arrow* $\mathfrak{N}$) $\in_\circ$ *Vset* β
by (*auto simp*: *cat-map-cs-simps*)
have $n \in_\circ \mathcal{D}_\circ$ (*ntcf-arrow* $\mathfrak{N}$) $\Longrightarrow$ *ntcf-arrow* $\mathfrak{N}$(|n|) $\in_\circ$ *Vset* β **for** n
by
(
unfold ntcf-arrow-vdomain,
elim-in-numeral,
all‹*rewrite in* ⌑ $\in_\circ$ - *nt-field-simps*[*symmetric*]›,
insert assms,
unfold ntcf-arrow-components'
)
(
auto intro:
ntcf-NTMap-in-Vset NTDom.cf-map-in-Vset NTCod.cf-map-in-Vset
)
with *ntcf-arrow.vsv-vrange-vsubset* **show** $\mathcal{R}_\circ$ (*ntcf-arrow* $\mathfrak{N}$) $\subseteq_\circ$ *Vset* β
by *simp*

```
  qed (auto simp: cat-map-cs-simps)
qed

lemma (in is-tm-ntcf) tm-ntcf-arrow-in-Vset: ntcf-arrow 𝔑 ∈∘ Vset α
proof–
  interpret tm-ntcf-arrow: vfsequence ‹ntcf-arrow 𝔑›
    by (auto intro: cat-map-cs-intros)
  show ?thesis
  proof(rule vsv.vsv-Limit-vsv-in-VsetI)
    have n ∈∘ 𝒟∘ (ntcf-arrow 𝔑) ⟹ ntcf-arrow 𝔑(|n|) ∈∘ Vset α for n
      by
        (
          unfold ntcf-arrow-vdomain,
          elim-in-numeral,
          all‹rewrite in ⌑ ∈∘ - nt-field-simps[symmetric]›
        )
        (
          intro tm-ntcf-arrow-NTMap-in-Vset
          tm-ntcf-arrow-NTDom-in-Vset
          tm-ntcf-arrow-NTCod-in-Vset
        )+
    with tm-ntcf-arrow.vsv-vrange-vsubset show ℛ∘ (ntcf-arrow 𝔑) ⊆∘ Vset α
      by auto
  qed (auto simp: cat-map-cs-simps)
qed

lemma ntcf-arrows-subset-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows
    {ntcf-arrow 𝔑 | 𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅} ⊆ elts (Vset β)
proof(intro subsetI, unfold mem-Collect-eq, elim exE conjE)
  fix x 𝔑 𝔉 𝔊 assume x-def: x = ntcf-arrow 𝔑
    and 𝔑: 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
  interpret is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule 𝔑)
  show x ∈∘ Vset β unfolding x-def by (rule ntcf-arrow-in-Vset[OF assms])
qed

lemma tm-ntcf-arrows-subset-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows
    {ntcf-arrow 𝔑 | 𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦CF.tm 𝔊 : 𝔄 ↦↦C.tmα 𝔅} ⊆
      elts (Vset β)
proof(intro subsetI, unfold mem-Collect-eq, elim exE conjE)
  fix x 𝔑 𝔉 𝔊 assume x-def: x = ntcf-arrow 𝔑
    and 𝔑: 𝔑 : 𝔉 ↦CF.tm 𝔊 : 𝔄 ↦↦C.tmα 𝔅
  interpret is-tm-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule 𝔑)
  show x ∈∘ Vset β unfolding x-def by (rule ntcf-arrow-in-Vset[OF assms])
qed

lemma small-ntcf-arrows[simp]:
  small {ntcf-arrow 𝔑 | 𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-ntcf.ntcf-arrow-in-Vset show ?thesis
    by (intro down[of - ‹Vset (α + ω)›])
      (auto simp: True 𝒵.𝒵-Limit-αω 𝒵.𝒵-ω-αω 𝒵.intro 𝒵.𝒵-α-αω)
next
  case False
```

```
  then have {ntcf-arrow 𝔑 | 𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅} = {}
    by auto
  then show ?thesis by simp
qed

lemma small-tm-ntcf-arrows[simp]:
  small {ntcf-arrow 𝔑 | 𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦_CF.tm 𝔊 : 𝔄 ↦↦_C.tmα 𝔅}
  by
    (
      rule smaller-than-small[
        OF small-ntcf-arrows tm-ntcf-arrows-subset-ntcf-arrows
        ]
    )

lemma (in is-ntcf) ntcf-arrow-in-Vset-𝒻: ntcf-arrow 𝔑 ∈ₒ Vset (α + 𝒻 ℕ)
proof–
  note [folded VPow-iff, folded Vset-succ[OF Ord-α], cat-cs-intros] =
    ntcf-NTMap-vsubset-Vset
  from NTDom.cf-map-in-Vset-4 have [cat-cs-intros]:
    cf-map 𝔉 ∈ₒ Vset (succ (succ (succ (succ α))))
    by succ-of-numeral
      (cs-prems cs-shallow cs-simp: plus-V-succ-right V-cs-simps)
  from NTCod.cf-map-in-Vset-4 have [cat-cs-intros]:
    cf-map 𝔊 ∈ₒ Vset (succ (succ (succ (succ α))))
    by succ-of-numeral
      (cs-prems cs-shallow cs-simp: plus-V-succ-right V-cs-simps)
  show ?thesis
    by (subst ntcf-arrow-def, succ-of-numeral, unfold cat-cs-simps)
      (
        cs-concl
          cs-simp: plus-V-succ-right V-cs-simps
          cs-intro: V-cs-intros cat-cs-intros
      )
qed

lemma (in 𝒵) ntcf-arrows-in-Vset:
  assumes 𝒵 β and α ∈ₒ β
  shows ntcf-arrows α 𝔄 𝔅 ∈ₒ Vset β
proof(rule vsubset-in-VsetI)
  interpret β: 𝒵 β by (rule assms(1))
  show ntcf-arrows α 𝔄 𝔅 ⊆ₒ Vset (α + 𝒻 ℕ)
  proof(intro vsubsetI)
    fix 𝔑 assume 𝔑 ∈ₒ ntcf-arrows α 𝔄 𝔅
    then obtain 𝔑' 𝔉 𝔊
      where 𝔑-def: 𝔑 = ntcf-arrow 𝔑'
        and 𝔑': 𝔑' : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
      by clarsimp
    interpret is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑' using 𝔑' by simp
    show 𝔑 ∈ₒ Vset (α + 𝒻 ℕ) unfolding 𝔑-def by (rule ntcf-arrow-in-Vset-𝒻)
  qed
  from assms(2) show Vset (α + 𝒻 ℕ) ∈ₒ Vset β
    by (cs-concl cs-shallow cs-intro: V-cs-intros Ord-cs-intros)
qed

lemma (in 𝒵) tm-ntcf-arrows-vsubset-Vset: tm-ntcf-arrows α 𝔄 𝔅 ⊆ₒ Vset α
  by (clarsimp simp: is-tm-ntcf.tm-ntcf-arrow-in-Vset)
```

Rules.

lemma (**in** *is-ntcf*) *ntcf-arrowsI*: *ntcf-arrow* $\mathfrak{N} \in_{\circ}$ *ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
using *is-ntcf-axioms* **by** *auto*

lemma (**in** *is-tm-ntcf*) *tm-ntcf-arrowsI*: *ntcf-arrow* $\mathfrak{N} \in_{\circ}$ *tm-ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
using *is-ntcf-axioms* **by** (*auto intro*: *cat-small-cs-intros*)

lemma (**in** *is-ntcf*) *ntcf-arrowsI'*:
assumes $\mathfrak{N}' =$ *ntcf-arrow* $\mathfrak{N}$
shows $\mathfrak{N}' \in_{\circ}$ *ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
unfolding *assms*(*1*) **by** (*rule ntcf-arrowsI*)

lemma (**in** *is-tm-ntcf*) *tm-ntcf-arrowsI'*:
assumes $\mathfrak{N}' =$ *ntcf-arrow* $\mathfrak{N}$
shows $\mathfrak{N}' \in_{\circ}$ *tm-ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
unfolding *assms*(*1*) **by** (*rule tm-ntcf-arrowsI*)

lemmas [*cat-map-cs-intros*] =
is-ntcf.ntcf-arrowsI

lemmas *ntcf-arrowsI'*[*cat-map-cs-intros*] =
is-ntcf.ntcf-arrowsI'[*rotated*]

lemmas [*cat-map-cs-intros*] =
is-tm-ntcf.tm-ntcf-arrowsI

lemmas *tm-ntcf-arrowsI'*[*cat-map-cs-intros*] =
is-tm-ntcf.tm-ntcf-arrowsI'[*rotated*]

lemma *ntcf-arrowsE*[*elim*]:
assumes $\mathfrak{N} \in_{\circ}$ *ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
obtains $\mathfrak{M}\ \mathfrak{F}\ \mathfrak{G}$ **where** $\mathfrak{N} =$ *ntcf-arrow* $\mathfrak{M}$ **and** $\mathfrak{M} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
using *assms* **by** *auto*

lemma *tm-ntcf-arrowsE*[*elim*]:
assumes $\mathfrak{N} \in_{\circ}$ *tm-ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
obtains $\mathfrak{M}\ \mathfrak{F}\ \mathfrak{G}$ **where** $\mathfrak{N} =$ *ntcf-arrow* $\mathfrak{M}$
and $\mathfrak{M} : \mathfrak{F} \mapsto_{CF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
using *assms* **by** *auto*

Elementary properties.

lemma *tm-ntcf-arrows-vsubset-ntcf-arrows*:
tm-ntcf-arrows $\alpha\ \mathfrak{A}\ \mathfrak{B} \subseteq_{\circ}$ *ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
using *tm-ntcf-arrows-subset-ntcf-arrows* **by** *auto*

lemma *tm-ntcf-arrows-in-cf-arrows*[*cat-map-cs-intros*]:
assumes $\mathfrak{N} \in_{\circ}$ *tm-ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
shows $\mathfrak{N} \in_{\circ}$ *ntcf-arrows* $\alpha\ \mathfrak{A}\ \mathfrak{B}$
using *assms tm-ntcf-arrows-vsubset-ntcf-arrows*[*of* $\alpha\ \mathfrak{A}\ \mathfrak{B}$] **by** *blast*

lemma *ntcf-arrow-inj*:
assumes *ntcf-arrow* $\mathfrak{M} =$ *ntcf-arrow* $\mathfrak{N}$
and $\mathfrak{M} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows $\mathfrak{M} = \mathfrak{N}$
proof(*rule ntcf-eqI*)
interpret $\mathfrak{M}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{M}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}'\ \mathfrak{G}'\ \mathfrak{N}$ **by** (*rule assms*(*3*))
from *assms*(*1*) **have** *NTMap*: *ntcf-arrow* $\mathfrak{M}(\lfloor NTMap \rfloor) =$ *ntcf-arrow* $\mathfrak{N}(\lfloor NTMap \rfloor)$

```
    and NTDom: ntcf-arrow 𝔐(‖NTDom‖) = ntcf-arrow 𝔑(‖NTDom‖)
    and NTCod: ntcf-arrow 𝔐(‖NTCod‖) = ntcf-arrow 𝔑(‖NTCod‖)
    by auto
  from NTMap show 𝔐(‖NTMap‖) = 𝔑(‖NTMap‖) unfolding ntcf-arrow-components by simp
  from NTDom NTCod show 𝔐(‖NTDom‖) = 𝔑(‖NTDom‖) 𝔐(‖NTCod‖) = 𝔑(‖NTCod‖)
    unfolding ntcf-arrow-components cf-map-components
    by
      (
        auto simp:
          cat-cs-simps
          cf-map-eq-iff[OF 𝔐.NTDom.is-functor-axioms 𝔑.NTDom.is-functor-axioms]
          cf-map-eq-iff[OF 𝔐.NTCod.is-functor-axioms 𝔑.NTCod.is-functor-axioms]
      )
  from assms(2,3) show
    𝔐 : 𝔐(‖NTDom‖) ↦_CF 𝔐(‖NTCod‖) : 𝔄 ↦↦_Cα 𝔅
    𝔑 : 𝔑(‖NTDom‖) ↦_CF 𝔑(‖NTCod‖) : 𝔄 ↦↦_Cα 𝔅
    by (auto simp: cat-cs-simps)
qed auto

lemma ntcf-arrow-eq-iff[cat-map-cs-simps]:
  assumes 𝔐 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅 and 𝔑 : 𝔉' ↦_CF 𝔊' : 𝔄 ↦↦_Cα 𝔅
  shows ntcf-arrow 𝔐 = ntcf-arrow 𝔑 ⟷ 𝔐 = 𝔑
  using ntcf-arrow-inj[OF - assms] by auto

lemma ntcf-arrow-eqI:
  assumes 𝔐 ∈₀ ntcf-arrows α 𝔄 𝔅
    and 𝔑 ∈₀ ntcf-arrows α 𝔄 𝔅
    and 𝔐(‖NTMap‖) = 𝔑(‖NTMap‖)
    and 𝔐(‖NTDom‖) = 𝔑(‖NTDom‖)
    and 𝔐(‖NTCod‖) = 𝔑(‖NTCod‖)
  shows 𝔐 = 𝔑
proof–
  from assms(1) obtain 𝔐' 𝔉 𝔊
    where 𝔐-def: 𝔐 = ntcf-arrow 𝔐' and 𝔐': 𝔐' : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
    by auto
  from assms(2) obtain 𝔑' 𝔉' 𝔊'
    where 𝔑-def: 𝔑 = ntcf-arrow 𝔑' and 𝔑': 𝔑' : 𝔉' ↦_CF 𝔊' : 𝔄 ↦↦_Cα 𝔅
    by auto
  show ?thesis
  proof(rule vsv-eqI, unfold 𝔐-def 𝔑-def)
    show a ∈₀ 𝒟₀ (ntcf-arrow 𝔐') ⟹ ntcf-arrow 𝔐'(‖a‖) = ntcf-arrow 𝔑'(‖a‖)
      for a
      by
        (
          unfold ntcf-arrow-vdomain,
          elim-in-numeral,
          insert assms(3–5),
          unfold 𝔐-def 𝔑-def,
          fold nt-field-simps
        )
        simp-all
  qed (auto intro: cat-map-cs-intros simp: cat-map-cs-simps)
qed
```

23.5 Conversion of a natural transformation arrow to a natural transformation

23.5.1 Definition and elementary properties

definition *ntcf-of-ntcf-arrow* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$ =
    [
      $\mathfrak{N}(\lfloor NTMap \rfloor)$,
      cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ $(\mathfrak{N}(\lfloor NTDom \rfloor))$,
      cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ $(\mathfrak{N}(\lfloor NTCod \rfloor))$,
      $\mathfrak{A}$,
      $\mathfrak{B}$
    ]$_\circ$

Components.

lemma *ntcf-of-ntcf-arrow-components*:
  shows *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\lfloor NTMap \rfloor) = \mathfrak{N}(\lfloor NTMap \rfloor)$
    and *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\lfloor NTDom \rfloor) =$ *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $(\mathfrak{N}(\lfloor NTDom \rfloor))$
    and *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\lfloor NTCod \rfloor) =$ *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $(\mathfrak{N}(\lfloor NTCod \rfloor))$
    and *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
    and *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}(\lfloor NTDGCod \rfloor) = \mathfrak{B}$
  unfolding *ntcf-of-ntcf-arrow-def nt-field-simps*
  by (*simp-all add*: *nat-omega-simps*)

lemmas [*cat-map-extra-cs-simps*] = *ntcf-of-ntcf-arrow-components*(*1*)
lemmas [*cat-map-cs-simps*] = *ntcf-of-ntcf-arrow-components*(*2–5*)

23.5.2 The conversion of a natural transformation arrow to a natural transformation is a natural transformation

lemma (**in** *is-ntcf*) *ntcf-of-ntcf-arrow-is-ntcf*:
  ntcf-of-ntcf-arrow $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$) : $\mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof(*rule is-ntcfI'*)
  show *vfsequence* (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$))
    unfolding *ntcf-of-ntcf-arrow-def* **by** *auto*
  show *vcard* (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$)) = $5_{\mathbb{N}}$
    unfolding *ntcf-of-ntcf-arrow-def* **by** (*simp add*: *nat-omega-simps*)
  show *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$)$(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$:
    $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
    if $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$ **for** a
    using *is-ntcf-axioms that*
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-map-cs-simps cat-map-extra-cs-simps*
          cs-intro: *cat-cs-intros*
)
  show *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$)$(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) =$
    $\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{B}}$ *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$)$(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
    if $f : a \mapsto_{\mathfrak{A}} b$ **for** a b f
    using *is-ntcf-axioms that*
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *ntcf-Comp-commute cat-map-cs-simps cat-map-extra-cs-simps*
          cs-intro: *cat-cs-intros*
)
qed

(
use *is-ntcf-axioms* **in**
‹*auto* *simp*: *cat-cs-simps* *cat-map-cs-simps* *cat-map-extra-cs-simps*›
)

lemma (**in** *is-ntcf*) *ntcf-of-ntcf-arrow-is-ntcf'*:
assumes $\mathfrak{N}' =$ *ntcf-arrow* $\mathfrak{N}$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}' : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule* *ntcf-of-ntcf-arrow-is-ntcf*)

lemmas [*cat-map-cs-intros*] = *is-ntcf.ntcf-of-ntcf-arrow-is-ntcf'*

23.5.3 The composition of the conversion of a natural transformation arrow to a natural transformation

lemma (**in** *is-ntcf*) *ntcf-of-ntcf-arrow*[*cat-map-cs-simps*]:
ntcf-of-ntcf-arrow $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{N}$) = $\mathfrak{N}$
by (*rule* *ntcf-eqI*)
(
auto
simp: *cat-map-cs-simps* *cat-map-extra-cs-simps*
intro: *cat-cs-intros* *ntcf-of-ntcf-arrow-is-ntcf*
)

lemmas [*cat-map-cs-simps*] = *is-ntcf.ntcf-of-ntcf-arrow*

23.6 Composition of the natural transformation arrows

definition *ntcf-arrow-vcomp* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-arrow-vcomp* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{M}$ $\mathfrak{N}$ =
ntcf-arrow (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{M}$ $\bullet_{NTCF}$ *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$)

syntax -*ntcf-arrow-vcomp* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
(‹(-/ $\bullet_{NTCF}$-,- -)› [*55*, *56*, *57*, *58*] *55*)
syntax-consts -*ntcf-arrow-vcomp* $\rightleftharpoons$ *ntcf-arrow-vcomp*
translations $\mathfrak{M} \bullet_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{N} \rightleftharpoons$ *CONST* *ntcf-arrow-vcomp* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{M}$ $\mathfrak{N}$

Components.

lemma (**in** *is-ntcf*) *ntcf-arrow-vcomp-components*:
(*ntcf-arrow* $\mathfrak{M}$ $\bullet_{NTCF\mathfrak{A},\mathfrak{B}}$ *ntcf-arrow* $\mathfrak{N}$)(|*NTMap*|) = ($\mathfrak{M}$ $\bullet_{NTCF}$ $\mathfrak{N}$)(|*NTMap*|)
(*ntcf-arrow* $\mathfrak{M}$ $\bullet_{NTCF\mathfrak{A},\mathfrak{B}}$ *ntcf-arrow* $\mathfrak{N}$)(|*NTDom*|) = *cf-map* (($\mathfrak{M}$ $\bullet_{NTCF}$ $\mathfrak{N}$)(|*NTDom*|))
(*ntcf-arrow* $\mathfrak{N}$ $\bullet_{NTCF\mathfrak{A},\mathfrak{B}}$ *ntcf-arrow* $\mathfrak{M}$)(|*NTCod*|) = *cf-map* (($\mathfrak{N}$ $\bullet_{NTCF}$ $\mathfrak{M}$)(|*NTCod*|))
unfolding
ntcf-arrow-vcomp-def
ntsmcf-vcomp-components
ntcf-arrow-components
ntcf-of-ntcf-arrow-components
by (*simp-all* *add*: *cat-cs-simps* *cat-map-cs-simps*)

lemmas [*cat-map-cs-simps*] = *is-ntcf.ntcf-arrow-vcomp-components*

Elementary properties.

lemma *ntcf-arrow-vcomp-ntcf-vcomp*[*cat-map-cs-simps*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *ntcf-arrow* $\mathfrak{M}$ $\bullet_{NTCF\mathfrak{A},\mathfrak{B}}$ *ntcf-arrow* $\mathfrak{N}$ = *ntcf-arrow* ($\mathfrak{M}$ $\bullet_{NTCF}$ $\mathfrak{N}$)
by (*rule* *ntcf-arrow-eqI*, *insert* *assms*)
(
cs-concl

cs-simp: *ntcf-arrow-vcomp-def cat-map-cs-simps cat-cs-simps*
cs-intro: *cat-map-cs-intros cat-cs-intros*
)+

23.7 Identity natural transformation arrow

definition *ntcf-arrow-id* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ = *ntcf-arrow* (*ntcf-id* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$))

Components.

lemma (**in** *is-functor*) *ntcf-arrow-id-components*:
(*ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$))(|*NTMap*|) = *ntcf-id* $\mathfrak{F}$(|*NTMap*|)
(*ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$))(|*NTDom*|) = *cf-map* (*ntcf-id* $\mathfrak{F}$(|*NTDom*|))
(*ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$))(|*NTCod*|) = *cf-map* (*ntcf-id* $\mathfrak{F}$(|*NTCod*|))
unfolding *ntcf-arrow-id-def ntcf-arrow-components*
by (*simp-all add*: *cat-map-cs-simps*)

lemmas [*cat-map-cs-simps*] = *is-functor.ntcf-arrow-id-components*

Identity natural transformation arrow is a natural transformation arrow.

lemma *ntcf-arrow-id-ntcf-id*[*cat-map-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
shows *ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ (*cf-map* $\mathfrak{F}$) = *ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)
by (*rule ntcf-arrow-eqI*, *insert assms*, *unfold ntcf-arrow-id-def*)
(
cs-concl
cs-simp: *cat-map-cs-simps cat-cs-simps*
cs-intro: *cat-map-cs-intros cat-cs-intros*
)

23.8 *FUNCT*

23.8.1 Definition and elementary properties

definition *dg-FUNCT* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$ =
[
cf-maps α $\mathfrak{A}$ $\mathfrak{B}$,
ntcf-arrows α $\mathfrak{A}$ $\mathfrak{B}$,
$(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}$(|*NTDom*|)),
$(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}$(|*NTCod*|))
]$_\circ$

lemmas [*dg-FUNCT-cs-simps*] = *cat-map-cs-simps*
lemmas [*dg-FUNCT-cs-intros*] = *cat-map-cs-intros*

Components.

lemma *dg-FUNCT-components*:
shows *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|) = *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
and *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Arr*|) = *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
and *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|) = $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}$(|*NTDom*|))
and *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|) = $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}$(|*NTCod*|))
unfolding *dg-FUNCT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

23.8.2 Objects

lemma (**in** *is-functor*) *dg-FUNCT-ObjI*: *cf-map* $\mathfrak{F}$ $\in_\circ$ *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
unfolding *dg-FUNCT-components* **by** (*auto intro*: *cat-cs-intros*)

23.8.3 Domain and codomain

mk-VLambda *dg-FUNCT-components*(*3*)
|*vsv dg-FUNCT-Dom-vsv*[*dg-FUNCT-cs-intros*]|
|*vdomain dg-FUNCT-Dom-vdomain*[*dg-FUNCT-cs-simps*]|

mk-VLambda *dg-FUNCT-components*(*4*)
|*vsv dg-FUNCT-Cod-vsv*[*dg-FUNCT-cs-intros*]|
|*vdomain dg-FUNCT-Cod-vdomain*[*dg-FUNCT-cs-simps*]|

lemma (**in** *is-ntcf*)
shows *dg-FUNCT-Dom-app*: *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{F}$
and *dg-FUNCT-Cod-app*: *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{G}$
proof–
from *is-ntcf-axioms* **show**
dg-FUNCT α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{F}$
dg-FUNCT α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{G}$
unfolding *dg-FUNCT-components*
by
(
cs-concl
cs-simp: *dg-FUNCT-cs-simps V-cs-simps* **cs-intro**: *dg-FUNCT-cs-intros*
)+
qed

lemma (**in** *is-ntcf*)
assumes $\mathfrak{N}'$ = *ntcf-arrow* $\mathfrak{N}$
shows *dg-FUNCT-Dom-app′*: *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|$\mathfrak{N}'$|) = *cf-map* $\mathfrak{F}$
and *dg-FUNCT-Cod-app′*: *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|$\mathfrak{N}'$|) = *cf-map* $\mathfrak{G}$
unfolding *assms* **by** (*intro dg-FUNCT-Dom-app dg-FUNCT-Cod-app*)+

lemmas [*dg-FUNCT-cs-simps*] =
is-ntcf.dg-FUNCT-Dom-app′
is-ntcf.dg-FUNCT-Cod-app′

lemma
shows *dg-FUNCT-Dom-vrange*: $\mathcal{R}_\circ$ (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)) $\subseteq_\circ$ *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
and *dg-FUNCT-Cod-vrange*: $\mathcal{R}_\circ$ (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)) $\subseteq_\circ$ *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
unfolding *dg-FUNCT-components*
proof(*all*‹*intro vrange-VLambda-vsubset*›)
fix $\mathfrak{N}$ **assume** $\mathfrak{N}$ $\in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
then obtain $\mathfrak{M}$ $\mathfrak{F}$ $\mathfrak{G}$ **where** $\mathfrak{N}$-*def*[*dg-FUNCT-cs-simps*]: $\mathfrak{N}$ = *ntcf-arrow* $\mathfrak{M}$
and $\mathfrak{M}$: $\mathfrak{M}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C\alpha}$ $\mathfrak{B}$
by *auto*
from $\mathfrak{M}$ **show** $\mathfrak{N}$(|*NTDom*|) $\in_\circ$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
by (*cs-concl* **cs-simp**: *dg-FUNCT-cs-simps* **cs-intro**: *dg-FUNCT-cs-intros cat-cs-intros*)
from $\mathfrak{M}$ **show** $\mathfrak{N}$(|*NTCod*|) $\in_\circ$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
by
(
cs-concl **cs-shallow**
cs-simp: *dg-FUNCT-cs-simps* **cs-intro**: *dg-FUNCT-cs-intros cat-cs-intros*
)
qed

23.8.4 *FUNCT* is a tiny digraph

lemma (**in** $\mathcal{Z}$) *tiny-digraph-dg-FUNCT*:
assumes $\mathcal{Z}$ β **and** α $\in_\circ$ β
shows *tiny-digraph* β (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)

proof–
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro tiny-digraphI*)
show *vfsequence* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) **unfolding** *dg-FUNCT-def* **by** *simp*
show *vcard* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = $\mathcal{4}_{\mathbb{N}}$
unfolding *dg-FUNCT-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)) $\subseteq_{\circ}$ *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
by (*simp add*: *dg-FUNCT-Dom-vrange dg-FUNCT-Cod-vrange*)
show $\mathcal{R}_{\circ}$ (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)) $\subseteq_{\circ}$ *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
by (*simp add*: *dg-FUNCT-Dom-vrange dg-FUNCT-Cod-vrange*)
from *assms* **show** *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|) $\in_{\circ}$ *Vset* β
unfolding *dg-FUNCT-components*(*1*) **by** (*rule cf-maps-in-Vset*)
show *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Arr*|) $\in_{\circ}$ *Vset* β
unfolding *dg-FUNCT-components*(*2*) **by** (*rule ntcf-arrows-in-Vset*[*OF assms*])
qed (*auto simp*: *dg-FUNCT-cs-simps dg-FUNCT-components*(*1*,*2*) *intro*: *dg-FUNCT-cs-intros*)
qed

23.8.5 Arrow with a domain and a codomain

lemma *dg-FUNCT-is-arrI*:
assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
shows *ntcf-arrow* $\mathfrak{N}$: *cf-map* $\mathfrak{F}$ $\mapsto_{dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ *cf-map* $\mathfrak{G}$
proof(*intro is-arrI*, *unfold dg-FUNCT-components*(*1*,*2*))
interpret *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*)
from *assms* **show** *ntcf-arrow* $\mathfrak{N}$ $\in_{\circ}$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$ **by** *auto*
from *assms* **show**
dg-FUNCT α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{F}$
dg-FUNCT α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{G}$
by (*cs-concl* **cs-shallow cs-simp**: *dg-FUNCT-cs-simps*)+
qed

lemma *dg-FUNCT-is-arrI'*:
assumes $\mathfrak{N}'$ = *ntcf-arrow* $\mathfrak{N}$
and $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
and $\mathfrak{F}'$ = *cf-map* $\mathfrak{F}$
and $\mathfrak{G}'$ = *cf-map* $\mathfrak{G}$
shows $\mathfrak{N}'$: $\mathfrak{F}'$ $\mapsto_{dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{G}'$
using *assms*(*2*) **unfolding** *assms*(*1*,*3*,*4*) **by** (*rule dg-FUNCT-is-arrI*)

lemmas [*dg-FUNCT-cs-intros*] = *dg-FUNCT-is-arrI'*

lemma *dg-FUNCT-is-arrD*[*dest*]:
assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{G}$
shows *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$:
cf-of-cf-map $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mapsto_{CF}$ *cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
and $\mathfrak{N}$ = *ntcf-arrow* (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$)
and $\mathfrak{F}$ = *cf-map* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)
and $\mathfrak{G}$ = *cf-map* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$)
proof–
note $\mathfrak{N}$ = *is-arrD*[*OF assms*, *unfolded dg-FUNCT-components*(*2*)]
obtain $\mathfrak{N}'$ $\mathfrak{F}'$ $\mathfrak{G}'$ **where** $\mathfrak{N}$-*def*: $\mathfrak{N}$ = *ntcf-arrow* $\mathfrak{N}'$
and $\mathfrak{N}'$: $\mathfrak{N}'$: $\mathfrak{F}'$ $\mapsto_{CF}$ $\mathfrak{G}'$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
by (*elim ntcf-arrowsE*[*OF* $\mathfrak{N}$(*1*)])
from $\mathfrak{N}$(*2*) $\mathfrak{N}'$ **have** $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-map* $\mathfrak{F}'$
by (*cs-prems* **cs-simp**: $\mathfrak{N}$-*def dg-FUNCT-cs-simps*) *simp*
from $\mathfrak{N}$(*3*) $\mathfrak{N}'$ **have** $\mathfrak{G}$-*def*: $\mathfrak{G}$ = *cf-map* $\mathfrak{G}'$
by (*cs-prems* **cs-simp**: $\mathfrak{N}$-*def dg-FUNCT-cs-simps*) *simp*

from $\mathfrak{N}'$ **have** $\mathfrak{N}'$-*def*: $\mathfrak{N}' = ntcf\text{-}of\text{-}ntcf\text{-}arrow\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}$
unfolding $\mathfrak{N}$-*def* **by** (*cs-concl* **cs-shallow cs-simp**: *dg-FUNCT-cs-simps*)
from $\mathfrak{N}'$ **have** $\mathfrak{F}'$-*def*: $\mathfrak{F}' = cf\text{-}of\text{-}cf\text{-}map\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$
and $\mathfrak{G}'$-*def*: $\mathfrak{G}' = cf\text{-}of\text{-}cf\text{-}map\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$
unfolding $\mathfrak{F}$-*def* $\mathfrak{G}$-*def*
by (*cs-concl* **cs-simp**: *dg-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros*)+
from $\mathfrak{N}'$ **show** $ntcf\text{-}of\text{-}ntcf\text{-}arrow\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}$:
$cf\text{-}of\text{-}cf\text{-}map\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F} \mapsto_{CF} cf\text{-}of\text{-}cf\text{-}map\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N} = ntcf\text{-}arrow\ (ntcf\text{-}of\text{-}ntcf\text{-}arrow\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N})$
and $\mathfrak{F} = cf\text{-}map\ (cf\text{-}of\text{-}cf\text{-}map\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F})$
and $\mathfrak{G} = cf\text{-}map\ (cf\text{-}of\text{-}cf\text{-}map\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G})$
by (*fold* $\mathfrak{F}'$-*def* $\mathfrak{G}'$-*def* $\mathfrak{N}'$-*def* $\mathfrak{F}$-*def* $\mathfrak{G}$-*def* $\mathfrak{N}$-*def*) *simp-all*
qed

lemma *dg-FUNCT-is-arrE*[*elim*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}} \mathfrak{G}$
obtains $\mathfrak{N}'\ \mathfrak{F}'\ \mathfrak{G}'$
where $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
and $\mathfrak{N} = ntcf\text{-}arrow\ \mathfrak{N}'$
and $\mathfrak{F} = cf\text{-}map\ \mathfrak{F}'$
and $\mathfrak{G} = cf\text{-}map\ \mathfrak{G}'$
using *dg-FUNCT-is-arrD*[*OF assms*] **by** *auto*

23.9 *Funct*

23.9.1 Definition and elementary properties

definition *dg-Funct* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *dg-Funct* $\alpha\ \mathfrak{A}\ \mathfrak{B}$ =
[
tm-cf-maps $\alpha\ \mathfrak{A}\ \mathfrak{B}$,
tm-ntcf-arrows $\alpha\ \mathfrak{A}\ \mathfrak{B}$,
$(\lambda\mathfrak{N}\in_\circ tm\text{-}ntcf\text{-}arrows\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N}(\!|NTDom|\!))$,
$(\lambda\mathfrak{N}\in_\circ tm\text{-}ntcf\text{-}arrows\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N}(\!|NTCod|\!))$
]$_\circ$

Components.

lemma *dg-Funct-components*:
shows $dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Obj|\!) = tm\text{-}cf\text{-}maps\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
and $dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Arr|\!) = tm\text{-}ntcf\text{-}arrows\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
and $dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Dom|\!) = (\lambda\mathfrak{N}\in_\circ tm\text{-}ntcf\text{-}arrows\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N}(\!|NTDom|\!))$
and $dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Cod|\!) = (\lambda\mathfrak{N}\in_\circ tm\text{-}ntcf\text{-}arrows\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N}(\!|NTCod|\!))$
unfolding *dg-Funct-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

23.9.2 Objects

lemma (**in** *is-tm-functor*) *dg-Funct-ObjI*: $cf\text{-}map\ \mathfrak{F} \in_\circ dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Obj|\!)$
unfolding *dg-Funct-components* **by** (*auto simp*: *cat-small-cs-intros*)

23.9.3 Domain and codomain

mk-VLambda *dg-Funct-components*(*3*)
|*vsv dg-Funct-Dom-vsv*[*dg-FUNCT-cs-intros*]|
|*vdomain dg-Funct-Dom-vdomain*[*dg-FUNCT-cs-simps*]|

mk-VLambda *dg-Funct-components*(*4*)
|*vsv dg-Funct-Cod-vsv*[*dg-FUNCT-cs-intros*]|
|*vdomain dg-Funct-Cod-vdomain*[*dg-FUNCT-cs-simps*]|

lemma (**in** *is-tm-ntcf*)
  shows *dg-Funct-Dom-app*: *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{F}$
    and *dg-Funct-Cod-app*: *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{G}$
proof–
  from *is-tm-ntcf-axioms* **show**
    dg-Funct α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{F}$
    dg-Funct α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|*ntcf-arrow* $\mathfrak{N}$|) = *cf-map* $\mathfrak{G}$
    unfolding *dg-Funct-components*
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *dg-FUNCT-cs-simps V-cs-simps*
          cs-intro: *dg-FUNCT-cs-intros cat-cs-intros*
)+
qed

lemma (**in** *is-tm-ntcf*)
  assumes $\mathfrak{N}'$ = *ntcf-arrow* $\mathfrak{N}$
  shows *dg-Funct-Dom-app'*: *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)(|$\mathfrak{N}'$|) = *cf-map* $\mathfrak{F}$
    and *dg-Funct-Cod-app'*: *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)(|$\mathfrak{N}'$|) = *cf-map* $\mathfrak{G}$
  unfolding *assms* **by** (*intro dg-Funct-Dom-app dg-Funct-Cod-app*)+

lemmas [*dg-FUNCT-cs-simps*] =
  is-tm-ntcf.dg-Funct-Dom-app'
  is-tm-ntcf.dg-Funct-Cod-app'

lemma
  shows *dg-Funct-Dom-vrange*: $\mathcal{R}_\circ$ (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)) $\subseteq_\circ$ *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
    and *dg-Funct-Cod-vrange*: $\mathcal{R}_\circ$ (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)) $\subseteq_\circ$ *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
  unfolding *dg-Funct-components*
proof(*all*‹*intro vrange-VLambda-vsubset*›)
  fix $\mathfrak{N}$ **assume** $\mathfrak{N}$ $\in_\circ$ *tm-ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
  then obtain $\mathfrak{M}$ $\mathfrak{F}$ $\mathfrak{G}$ **where** $\mathfrak{N}$-*def*[*dg-FUNCT-cs-simps*]: $\mathfrak{N}$ = *ntcf-arrow* $\mathfrak{M}$
    and $\mathfrak{M}$: $\mathfrak{M}$: $\mathfrak{F}$ $\mapsto_{CF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.tm\alpha}$ $\mathfrak{B}$
    by *auto*
  from $\mathfrak{M}$ **show** $\mathfrak{N}$(|*NTDom*|) $\in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
    by
      (
        cs-concl
          cs-simp: *dg-FUNCT-cs-simps*
          cs-intro: *dg-FUNCT-cs-intros cat-small-cs-intros*
)
  from $\mathfrak{M}$ **show** $\mathfrak{N}$(|*NTCod*|) $\in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *dg-FUNCT-cs-simps*
          cs-intro: *dg-FUNCT-cs-intros cat-small-cs-intros*
)
qed

23.9.4 Arrow with a domain and a codomain

lemma *dg-Funct-is-arrI*:
  assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{C.tm\alpha}$ $\mathfrak{B}$
  shows *ntcf-arrow* $\mathfrak{N}$: *cf-map* $\mathfrak{F}$ $\mapsto_{dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ *cf-map* $\mathfrak{G}$
proof(*intro is-arrI*, *unfold dg-Funct-components*(*1*,*2*))
  interpret *is-tm-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*)

```
  from assms show ntcf-arrow 𝔑 ∈∘ tm-ntcf-arrows α 𝔄 𝔅 by auto
  from assms show
    dg-Funct α 𝔄 𝔅(|Dom|)(|ntcf-arrow 𝔑|) = cf-map 𝔉
    dg-Funct α 𝔄 𝔅(|Cod|)(|ntcf-arrow 𝔑|) = cf-map 𝔊
    by (cs-concl cs-shallow cs-simp: dg-FUNCT-cs-simps)+
qed

lemma dg-Funct-is-arrI':
  assumes 𝔑' = ntcf-arrow 𝔑
    and 𝔑 : 𝔉 ↦CF.tm 𝔊 : 𝔄 ↦↦C.tmα 𝔅
    and 𝔉' = cf-map 𝔉
    and 𝔊' = cf-map 𝔊
  shows 𝔑' : 𝔉' ↦dg-Funct α 𝔄 𝔅 𝔊'
  using assms(2) unfolding assms(1,3,4) by (rule dg-Funct-is-arrI)

lemmas [dg-FUNCT-cs-intros] = dg-Funct-is-arrI'

lemma dg-Funct-is-arrD[dest]:
  assumes 𝔑 : 𝔉 ↦dg-Funct α 𝔄 𝔅 𝔊
  shows ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑 :
    cf-of-cf-map 𝔄 𝔅 𝔉 ↦CF.tm cf-of-cf-map 𝔄 𝔅 𝔊 : 𝔄 ↦↦C.tmα 𝔅
    and 𝔑 = ntcf-arrow (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑)
    and 𝔉 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔉)
    and 𝔊 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔊)
proof–
  note 𝔑 = is-arrD[OF assms, unfolded dg-Funct-components(2)]
  obtain 𝔑' 𝔉' 𝔊' where 𝔑-def: 𝔑 = ntcf-arrow 𝔑'
    and 𝔑': 𝔑' : 𝔉' ↦CF.tm 𝔊' : 𝔄 ↦↦C.tmα 𝔅
    by (elim tm-ntcf-arrowsE[OF 𝔑(1)])
  from 𝔑(2) 𝔑' have 𝔉-def: 𝔉 = cf-map 𝔉'
    by (cs-prems cs-simp: 𝔑-def dg-FUNCT-cs-simps) simp
  from 𝔑(3) 𝔑' have 𝔊-def: 𝔊 = cf-map 𝔊'
    by (cs-prems cs-simp: 𝔑-def dg-FUNCT-cs-simps) simp
  from 𝔑' have 𝔑'-def: 𝔑' = ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑
    unfolding 𝔑-def
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-FUNCT-cs-simps cs-intro: cat-small-cs-intros cat-cs-intros
      )
  from 𝔑' have 𝔉'-def: 𝔉' = cf-of-cf-map 𝔄 𝔅 𝔉
    and 𝔊'-def: 𝔊' = cf-of-cf-map 𝔄 𝔅 𝔊
    unfolding 𝔉-def 𝔊-def
    by
      (
        cs-concl
          cs-simp: dg-FUNCT-cs-simps cs-intro: cat-small-cs-intros cat-cs-intros
      )+
  from 𝔑' show ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑 :
    cf-of-cf-map 𝔄 𝔅 𝔉 ↦CF.tm cf-of-cf-map 𝔄 𝔅 𝔊 : 𝔄 ↦↦C.tmα 𝔅
    and 𝔑 = ntcf-arrow (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑)
    and 𝔉 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔉)
    and 𝔊 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔊)
    by (fold 𝔉'-def 𝔊'-def 𝔑'-def 𝔉-def 𝔊-def 𝔑-def) simp-all
qed

lemma dg-Funct-is-arrE[elim]:
  assumes 𝔑 : 𝔉 ↦dg-Funct α 𝔄 𝔅 𝔊
```

obtains $\mathfrak{N}'$ $\mathfrak{F}'$ $\mathfrak{G}'$ **where** $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF.tm} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{N} = ntcf\text{-}arrow\ \mathfrak{N}'$
and $\mathfrak{F} = cf\text{-}map\ \mathfrak{F}'$
and $\mathfrak{G} = cf\text{-}map\ \mathfrak{G}'$
using *dg-Funct-is-arrD*[*OF assms*] **by** *auto*

23.9.5 *Funct* is a digraph

lemma *digraph-dg-Funct*:
assumes *tiny-category* α $\mathfrak{A}$ **and** *category* α $\mathfrak{B}$
shows *digraph* α (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)
proof–

interpret *tiny-category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{B}$: *category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))

show *?thesis*

proof(*intro digraphI*)

show *vfsequence* (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) **unfolding** *dg-Funct-def* **by** *simp*
show *vcard* (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) = $\mathcal{4}_{\mathbb{N}}$
unfolding *dg-Funct-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Dom*|)) $\subseteq_\circ$ *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
by (*simp add*: *dg-Funct-Dom-vrange dg-Funct-Cod-vrange*)
show $\mathcal{R}_\circ$ (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Cod*|)) $\subseteq_\circ$ *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
by (*simp add*: *dg-Funct-Dom-vrange dg-Funct-Cod-vrange*)
show *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|) $\subseteq_\circ$ *Vset* α
unfolding *dg-Funct-components*(*1,2*) **by** (*rule tm-cf-maps-vsubset-Vset*)

have *RA*:
($\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|))) $\in_\circ$ *Vset* α
($\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|))) $\subseteq_\circ$ $\mathfrak{B}$(|*Obj*|)
if $A \subseteq_\circ$ *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|) **and** $A \in_\circ$ *Vset* α **for** A
proof–
have ($\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|))) $\subseteq_\circ$ $\mathfrak{B}$(|*Obj*|)
and ($\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|))) $\subseteq_\circ$ $\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ A))))$
proof(*all*‹*intro vsubsetI*›)
fix f **assume** $f \in_\circ$ ($\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|)))
then obtain $\mathfrak{F}$ **where** $\mathfrak{F}$: $\mathfrak{F} \in_\circ A$ **and** f: $f \in_\circ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|)) **by** *auto*
with *that*(*1*) **have** $\mathfrak{F} \in_\circ$ *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|) **by** (*elim vsubsetE*)
then obtain $\mathfrak{F}'$
where $\mathfrak{F}$-*def*: $\mathfrak{F} = cf\text{-}map\ \mathfrak{F}'$ **and** $\mathfrak{F}'$: $\mathfrak{F}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
unfolding *dg-Funct-components* **by** *auto*
interpret $\mathfrak{F}'$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}'$ **by** (*rule* $\mathfrak{F}'$)
from f **obtain** a **where** $a \in_\circ \mathcal{D}_\circ$ ($\mathfrak{F}'$(|*ObjMap*|)) **and** af: $\langle a, f\rangle \in_\circ \mathfrak{F}'$(|*ObjMap*|)
unfolding $\mathfrak{F}$-*def cf-map-components vdomain-iff* **by** *force*
then show $f \in_\circ \mathfrak{B}$(|*Obj*|)
using $\mathfrak{F}'$.*cf-ObjMap-vrange* $\mathfrak{F}$-*def cf-map-components*(*1*) *f vsubsetE* **by** *auto*
show $f \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ A))))$
proof(*intro VUnionI*)
show $\mathfrak{F} \in_\circ A$ **by** (*rule* $\mathfrak{F}$)
show *set* {*0*, $\mathfrak{F}$(|*ObjMap*|)} $\in_\circ \langle []_\circ, \mathfrak{F}$(|*ObjMap*|)$\rangle$ **unfolding** *vpair-def* **by** *simp*
show $\langle a, f\rangle \in_\circ \mathfrak{F}$(|*ObjMap*|)
unfolding $\mathfrak{F}$-*def cf-map-components* **by** (*intro af*)
show *set* {a, f} $\in_\circ \langle a, f\rangle$ **unfolding** *vpair-def* **by** *clarsimp*
qed (*clarsimp simp*: $\mathfrak{F}$-*def cf-map-def dg-FUNCT-Obj-components*)+
qed

moreover have $\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ(\bigcup_\circ A))))) \in_\circ Vset\ \alpha$
by (*intro VUnion-in-VsetI that*(*2*))
ultimately show
$(\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ObjMap|\!))) \in_\circ Vset\ \alpha$
$(\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ObjMap|\!))) \subseteq_\circ \mathfrak{B}(\!|Obj|\!)$
by *blast+*
qed

fix *A B* **assume** *prems*:
$A \subseteq_\circ$ *dg-Funct* $\alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Obj|\!)$
$B \subseteq_\circ$ *dg-Funct* $\alpha\ \mathfrak{A}\ \mathfrak{B}(\!|Obj|\!)$
$A \in_\circ Vset\ \alpha$
$B \in_\circ Vset\ \alpha$

define *ARs* **where** $ARs = (\bigcup_\circ \mathfrak{F} \in_\circ A.\ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ObjMap|\!)))$
define *BRs* **where** $BRs = (\bigcup_\circ \mathfrak{G} \in_\circ B.\ \mathcal{R}_\circ\ (\mathfrak{G}(\!|ObjMap|\!)))$
define *Hom-AB* **where** $Hom\text{-}AB = (\bigcup_\circ a \in_\circ ARs.\ \bigcup_\circ b \in_\circ BRs.\ Hom\ \mathfrak{B}\ a\ b)$

define *Q* **where**
$Q\ i = (\textit{if } i = 0 \textit{ then } VPow\ (\mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB) \textit{ else if } i = 1_\mathbb{N} \textit{ then } A \textit{ else } B)$
for *i*
have
$\{[\mathfrak{N}, \mathfrak{F}, \mathfrak{G}]_\circ \mid \mathfrak{N}\ \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} \subseteq_\circ \mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB \wedge \mathfrak{F} \in_\circ A \wedge \mathfrak{G} \in_\circ B\} \subseteq$
elts $(\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i)$
proof(*intro subsetI*, *unfold mem-Collect-eq*, *elim exE conjE*)
fix $\mathfrak{N}\mathfrak{F}\mathfrak{G}\ \mathfrak{N}\ \mathfrak{F}\ \mathfrak{G}$ **assume** *prems'*:
$\mathfrak{N}\mathfrak{F}\mathfrak{G} = [\mathfrak{N}, \mathfrak{F}, \mathfrak{G}]_\circ\ \mathfrak{N} \subseteq_\circ \mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB\ \mathfrak{F} \in_\circ A\ \mathfrak{G} \in_\circ B$
show $\mathfrak{N}\mathfrak{F}\mathfrak{G} \in_\circ (\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i)$
proof(*intro vproductI*, *unfold Ball-def*; (*intro allI impI*)*?*)
fix *i* **assume** $i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$
then consider $\langle i = 0 \rangle \mid \langle i = 1_\mathbb{N} \rangle \mid \langle i = 2_\mathbb{N} \rangle$ **by** *auto*
then show $\mathfrak{N}\mathfrak{F}\mathfrak{G}(\!|i|\!) \in_\circ Q\ i$
by *cases* (*auto simp*: *Q-def prems' nat-omega-simps*)
qed (*auto simp*: *prems'*(*1*) *three nat-omega-simps*)
qed
moreover then have *small*[*simp*]:
$small\ \{[r, a, b]_\circ \mid r\ a\ b.\ r \subseteq_\circ \mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB \wedge a \in_\circ A \wedge b \in_\circ B\}$
by (*rule down*)
ultimately have
$set\ \{[r, a, b]_\circ \mid r\ a\ b.\ r \subseteq_\circ \mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB \wedge a \in_\circ A \wedge b \in_\circ B\} \subseteq_\circ$
$(\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i)$
by *auto*
moreover have $(\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}.\ Q\ i) \in_\circ Vset\ \alpha$
proof(*rule Limit-vproduct-in-VsetI*)
show $set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\} \in_\circ Vset\ \alpha$
by (*cs-concl* **cs-intro**: *V-cs-intros cat-cs-intros* **cs-simp**: *V-cs-simps*)
have $Hom\text{-}AB \in_\circ Vset\ \alpha$
unfolding *Hom-AB-def*
by
(
intro $\mathfrak{B}$.*cat-Hom-vifunion-in-Vset prems*(*3,4*),
unfold ARs-def BRs-def;
intro RA prems
)
moreover have $\mathfrak{A}(\!|Obj|\!) \in_\circ Vset\ \alpha$ **by** (*intro tiny-cat-Obj-in-Vset*)
ultimately have $VPow\ (\mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB) \in_\circ Vset\ \alpha$
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros*)
with *prems*(*3,4*) **show** $Q\ i \in_\circ Vset\ \alpha$ **if** $i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}\}$ **for** *i*

```
    unfolding Q-def by (simp-all add: nat-omega-simps)
qed auto
moreover have
  (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (dg-Funct α 𝔄 𝔅) a b) ⊆∘
    set {[r, a, b]∘ | r a b. r ⊆∘ 𝔄(|Obj|) ×∘ Hom-AB ∧ a ∈∘ A ∧ b ∈∘ B}
proof(rule vsubsetI)
  fix 𝔑 assume 𝔑 ∈∘ (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (dg-Funct α 𝔄 𝔅) a b)
  then obtain 𝔉 𝔊
    where 𝔉: 𝔉 ∈∘ A
      and 𝔊: 𝔊 ∈∘ B
      and 𝔑-ab: 𝔑 ∈∘ Hom (dg-Funct α 𝔄 𝔅) 𝔉 𝔊
    by auto
  then have 𝔑 : 𝔉 ↦_dg-Funct α 𝔄 𝔅 𝔊 by simp
  note 𝔑 = dg-Funct-is-arrD[OF this]
  show
    𝔑 ∈∘ set {[r, a, b]∘ | r a b. r ⊆∘ 𝔄(|Obj|) ×∘ Hom-AB ∧ a ∈∘ A ∧ b ∈∘ B}
  proof(intro in-set-CollectI small exI conjI)
    show 𝔑 =
      [
        ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑(|NTMap|),
        cf-map (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑(|NTDom|)),
        cf-map (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑(|NTCod|))
      ]∘
      by (rule 𝔑(2)[unfolded ntcf-arrow-def])
    interpret 𝔑: is-tm-ntcf α
      𝔄 𝔅
      ‹cf-of-cf-map 𝔄 𝔅 𝔉› ‹cf-of-cf-map 𝔄 𝔅 𝔊›
      ‹ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑›
      rewrites ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑(|NTMap|) = 𝔑(|NTMap|)
        and cf-of-cf-map 𝔄 𝔅 𝔉(|ObjMap|) = 𝔉(|ObjMap|)
        and cf-of-cf-map 𝔄 𝔅 𝔊(|ObjMap|) = 𝔊(|ObjMap|)
      by
        (
          rule 𝔑(1),
          unfold ntcf-of-ntcf-arrow-components cf-of-cf-map-components
        )
        simp-all
    show ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑(|NTMap|) ⊆∘ 𝔄(|Obj|) ×∘ Hom-AB
    proof(intro vsubsetI, unfold ntcf-of-ntcf-arrow-components)
      fix af assume prems'': af ∈∘ 𝔑(|NTMap|)
      then obtain a f where af-def: af = ⟨a, f⟩
        and a: a ∈∘ 𝔄(|Obj|)
        and f: f ∈∘ ℛ∘ (𝔑(|NTMap|))
        by (elim 𝔑.NTMap.vbrelation-vinE)
      from prems'' have f-def: f = 𝔑(|NTMap|)(|a|)
        unfolding af-def 𝔑.NTMap.vsv-ex1-app1[OF a] .
      have 𝔑a: 𝔑(|NTMap|)(|a|) : 𝔉(|ObjMap|)(|a|) ↦_𝔅 𝔊(|ObjMap|)(|a|)
        by (rule 𝔑.ntcf-NTMap-is-arr[OF a])
      have f ∈∘ Hom-AB
        unfolding f-def Hom-AB-def ARs-def BRs-def
      proof(intro vifunionI, unfold in-Hom-iff)
        show 𝔉 ∈∘ A by (intro 𝔉)
        from a show 𝔉(|ObjMap|)(|a|) ∈∘ ℛ∘ (𝔉(|ObjMap|))
          by (metis 𝔑.NTDom.ObjMap.vdomain-atD 𝔑.NTDom.cf-ObjMap-vdomain)
        show 𝔊 ∈∘ B by (intro 𝔊)
        from a show 𝔊(|ObjMap|)(|a|) ∈∘ ℛ∘ (𝔊(|ObjMap|))
          by (metis 𝔑.NTCod.ObjMap.vdomain-atD 𝔑.NTCod.cf-ObjMap-vdomain)
        show 𝔑(|NTMap|)(|a|) : 𝔉(|ObjMap|)(|a|) ↦_𝔅 𝔊(|ObjMap|)(|a|) by (intro 𝔑a)
```

qed
with a **show** $af \in_\circ \mathfrak{A}(\!|Obj|\!) \times_\circ Hom\text{-}AB$ **unfolding** $af\text{-}def\ f\text{-}def$ **by** $simp$
qed
show $cf\text{-}map\ (ntcf\text{-}of\text{-}ntcf\text{-}arrow\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}(\!|NTDom|\!)) \in_\circ A$
unfolding $\mathfrak{N}.ntcf\text{-}NTDom\ \mathfrak{N}(3)[symmetric]$ **by** $(rule\ \mathfrak{F})$
show $cf\text{-}map\ (ntcf\text{-}of\text{-}ntcf\text{-}arrow\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}(\!|NTCod|\!)) \in_\circ B$
unfolding $\mathfrak{N}.ntcf\text{-}NTCod\ \mathfrak{N}(4)[symmetric]$ **by** $(rule\ \mathfrak{G})$
qed
qed
ultimately show $(\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ Hom\ (dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B})\ a\ b) \in_\circ Vset\ \alpha$
by $blast$
qed $(auto\ simp\colon dg\text{-}Funct\text{-}components)$

qed

23.9.6 *Funct* is a subdigraph of *FUNCT*

lemma $subdigraph\text{-}dg\text{-}Funct\text{-}dg\text{-}FUNCT$:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$ **and** $tiny\text{-}category\ \alpha\ \mathfrak{A}$ **and** $category\ \alpha\ \mathfrak{B}$
shows $dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B} \subseteq_{DG\beta} dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
proof$(intro\ subdigraphI,\ unfold\ dg\text{-}FUNCT\text{-}components(1)\ dg\text{-}Funct\text{-}components(1))$
interpret $\mathfrak{A}$: $tiny\text{-}category\ \alpha\ \mathfrak{A}$ **by** $(rule\ assms(3))$
interpret β: $\mathcal{Z}\ \beta$ **by** $(rule\ assms(1))$
show $digraph\ \beta\ (dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B})$
by $(intro\ digraph.dg\text{-}digraph\text{-}if\text{-}ge\text{-}Limit[OF\ digraph\text{-}dg\text{-}Funct]\ assms)$
from $assms$ show $digraph\ \beta\ (dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B})$
by $(cs\text{-}concl$ **cs-shallow cs-intro**$\colon dg\text{-}small\text{-}cs\text{-}intros\ \mathfrak{A}.tiny\text{-}digraph\text{-}dg\text{-}FUNCT)$
show $\mathfrak{F} \in_\circ cf\text{-}maps\ \alpha\ \mathfrak{A}\ \mathfrak{B}$ **if** $\mathfrak{F} \in_\circ tm\text{-}cf\text{-}maps\ \alpha\ \mathfrak{A}\ \mathfrak{B}$ **for** $\mathfrak{F}$
using $that$
by $(cs\text{-}concl$ **cs-shallow cs-intro**$\colon dg\text{-}FUNCT\text{-}cs\text{-}intros\ tm\text{-}cf\text{-}maps\text{-}in\text{-}cf\text{-}maps)$
show $\mathfrak{N} : \mathfrak{F} \mapsto_{dg\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}} \mathfrak{G}$ **if** $\mathfrak{N} : \mathfrak{F} \mapsto_{dg\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}} \mathfrak{G}$
for $\mathfrak{N}\ \mathfrak{F}\ \mathfrak{G}$
proof–
note $f = dg\text{-}Funct\text{-}is\text{-}arrD[OF\ that]$
from $f(1)$ show $?thesis$
by $(subst\ f(2),\ use\ nothing$ **in** $\langle subst\ f(3),\ subst\ f(4)\rangle)$
$(cs\text{-}concl$ **cs-shallow cs-intro**$\colon dg\text{-}FUNCT\text{-}cs\text{-}intros\ cat\text{-}small\text{-}cs\text{-}intros)$
qed
qed

24 *FUNCT* and *Funct* as semicategories

24.1 Background

The subsection presents the theory of the semicategories of α-functors between two α-categories. It continues the development that was initiated in section 23. A general reference for this section is Chapter II-4 in [7].

named-theorems *smc-FUNCT-cs-simps*
named-theorems *smc-FUNCT-cs-intros*

lemmas [*smc-FUNCT-cs-simps*] = *cat-map-cs-simps*
lemmas [*smc-FUNCT-cs-intros*] = *cat-map-cs-intros*

24.2 *FUNCT*

24.2.1 Definition and elementary properties

definition *smc-FUNCT* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$ =
    [
      cf-maps α $\mathfrak{A}$ $\mathfrak{B}$,
      ntcf-arrows α $\mathfrak{A}$ $\mathfrak{B}$,
      $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTDom|\!))$,
      $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTCod|\!))$,
      $(\lambda\mathfrak{MN}\in_\circ$*composable-arrs* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$). $\mathfrak{MN}(\!|0|\!) \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{MN}(\!|1_{\mathbb{N}}|\!))$
    ]$_\circ$

Components.

lemma *smc-FUNCT-components*:
  shows *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Obj|\!)$ = *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
    and *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Arr|\!)$ = *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    and *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Dom|\!)$ = $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTDom|\!))$
    and *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Cod|\!)$ = $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTCod|\!))$
    and *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Comp|\!)$ =
      $(\lambda\mathfrak{MN}\in_\circ$*composable-arrs* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$). $\mathfrak{MN}(\!|0|\!) \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{MN}(\!|1_{\mathbb{N}}|\!))$
  unfolding *smc-FUNCT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-FUNCT*: *smc-dg* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$
proof(*rule vsv-eqI*)
  show *vsv* (*smc-dg* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) **unfolding** *smc-dg-def* **by** *auto*
  show *vsv* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) **unfolding** *dg-FUNCT-def* **by** *auto*
  have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) = $4_{\mathbb{N}}$
    unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = $4_{\mathbb{N}}$
    unfolding *dg-FUNCT-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*smc-dg* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) = $\mathcal{D}_\circ$ (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show $a \in_\circ \mathcal{D}_\circ$ (*smc-dg* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) $\Longrightarrow$
    smc-dg (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)$(\!|a|\!)$ = *dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|a|\!)$
    for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold smc-dg-def dg-field-simps smc-FUNCT-def dg-FUNCT-def
)

(*auto simp*: *nat-omega-simps*)
qed

context *is-ntcf*
begin

lemmas-with [*folded smc-dg-FUNCT*, *unfolded slicing-simps*]:
smc-FUNCT-Dom-app = *dg-FUNCT-Dom-app*
and *smc-FUNCT-Cod-app* = *dg-FUNCT-Cod-app*

end

lemmas [*smc-FUNCT-cs-simps*] =
is-ntcf.smc-FUNCT-Dom-app
is-ntcf.smc-FUNCT-Cod-app

lemmas-with [*folded smc-dg-FUNCT*, *unfolded slicing-simps*]:
smc-FUNCT-Dom-vsv[*intro*] = *dg-FUNCT-Dom-vsv*
and *smc-FUNCT-Dom-vdomain*[*smc-FUNCT-cs-simps*] = *dg-FUNCT-Dom-vdomain*
and *smc-FUNCT-Cod-vsv*[*intro*] = *dg-FUNCT-Cod-vsv*
and *smc-FUNCT-Cod-vdomain*[*smc-FUNCT-cs-simps*] = *dg-FUNCT-Cod-vdomain*
and *smc-FUNCT-Dom-vrange* = *dg-FUNCT-Dom-vrange*
and *smc-FUNCT-Cod-vrange* = *dg-FUNCT-Cod-vrange*
and *smc-FUNCT-is-arrI* = *dg-FUNCT-is-arrI*
and *smc-FUNCT-is-arrI'*[*smc-FUNCT-cs-intros*] = *dg-FUNCT-is-arrI'*
and *smc-FUNCT-is-arrD* = *dg-FUNCT-is-arrD*
and *smc-FUNCT-is-arrE*[*elim*] = *dg-FUNCT-is-arrE*

24.2.2 Composable arrows

lemma *smc-FUNCT-composable-arrs-dg-FUNCT*:
composable-arrs (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = *composable-arrs* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)
unfolding *composable-arrs-def smc-dg-FUNCT*[*symmetric*] *slicing-simps* **by** *auto*

lemma *smc-FUNCT-Comp*:
smc-FUNCT α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|) =
($\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$). $\mathfrak{GF}$(|*0*|) $\cdot_{NTCF\mathfrak{A},\mathfrak{B}}$ $\mathfrak{GF}$(|$1_{\mathbb{N}}$|))
unfolding *smc-FUNCT-components smc-FUNCT-composable-arrs-dg-FUNCT* **..**

24.2.3 Composition

lemma *smc-FUNCT-Comp-vsv*[*intro*]: *vsv* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|))
unfolding *smc-FUNCT-Comp* **by** *simp*

lemma *smc-FUNCT-Comp-vdomain*:
$\mathcal{D}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|)) = *composable-arrs* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)
unfolding *smc-FUNCT-Comp* **by** *auto*

lemma *smc-FUNCT-Comp-app*[*smc-FUNCT-cs-simps*]:
assumes $\mathfrak{M}$: $\mathfrak{G}$ $\mapsto_{smc\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{H}$ **and** $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{smc\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{G}$
shows $\mathfrak{M}$ $\circ_{A\,smc\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{N}$ = $\mathfrak{M}$ $\cdot_{NTCF\mathfrak{A},\mathfrak{B}}$ $\mathfrak{N}$
proof–
from *assms* **have** [$\mathfrak{M}$, $\mathfrak{N}$]$_\circ$ $\in_\circ$ *composable-arrs* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)
by (*auto intro*: *smc-cs-intros*)
then show $\mathfrak{M}$ $\circ_{A\,smc\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{N}$ = $\mathfrak{M}$ $\cdot_{NTCF\mathfrak{A},\mathfrak{B}}$ $\mathfrak{N}$
unfolding *smc-FUNCT-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-FUNCT-Comp-vrange*: $\mathcal{R}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|)) $\subseteq_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
proof(*rule vsubsetI*)
fix $\mathfrak{L}$ **assume** *prems*: $\mathfrak{L}$ $\in_\circ$ $\mathcal{R}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|))
then obtain $\mathfrak{MN}$
where $\mathfrak{L}$-*def*: $\mathfrak{L}$ = *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|)(|$\mathfrak{MN}$|)
and $\mathfrak{MN}$ $\in_\circ$ $\mathcal{D}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|))
unfolding *smc-FUNCT-components* **by** (*auto intro*: *smc-cs-intros*)
then obtain $\mathfrak{M}$ $\mathfrak{N}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{H}$
where $\mathfrak{MN}$ = [$\mathfrak{M}$, $\mathfrak{N}$]$_\circ$
and $\mathfrak{M}$: $\mathfrak{M}$: $\mathfrak{G}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{H}$
and $\mathfrak{N}$: $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{G}$
by (*auto simp*: *smc-FUNCT-Comp-vdomain*)
with $\mathfrak{L}$-*def* **have** $\mathfrak{L}$-*def'*: $\mathfrak{L}$ = $\mathfrak{M}$ $\circ_{A\,\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{N}$ **by** *simp*
note $\mathfrak{M}$ = *smc-FUNCT-is-arrD*[*OF* $\mathfrak{M}$]
and $\mathfrak{N}$ = *smc-FUNCT-is-arrD*[*OF* $\mathfrak{N}$]
from $\mathfrak{M}$(*1*) $\mathfrak{N}$(*1*) **show** $\mathfrak{L}$ $\in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding $\mathfrak{L}$-*def'*
by (*subst* $\mathfrak{M}$(*2*), *subst* $\mathfrak{N}$(*2*), *remdups*)
(
cs-concl
cs-simp: *smc-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros smc-FUNCT-cs-intros*
)
qed

24.2.4 *FUNCT* is a semicategory

lemma (**in** $\mathcal{Z}$) *tiny-semicategory-smc-FUNCT*:
assumes $\mathcal{Z}$ β **and** α $\in_\circ$ β
shows *tiny-semicategory* β (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)
proof(*intro tiny-semicategoryI*)
show *vfsequence* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) **by** (*simp add*: *smc-FUNCT-def*)
show *vcard* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = $5_{\mathbb{N}}$
unfolding *smc-FUNCT-def* **by** (*simp add*: *nat-omega-simps*)
show ($\mathfrak{MN}$ $\in_\circ$ $\mathcal{D}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|))) =
(
$\exists \mathfrak{M}\ \mathfrak{N}\ \mathfrak{G}\ \mathfrak{H}\ \mathfrak{F}.$
$\mathfrak{MN}$ = [$\mathfrak{M}$, $\mathfrak{N}$]$_\circ$ $\wedge$
$\mathfrak{M}$: $\mathfrak{G}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{H}$ $\wedge$
$\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{G}$
)
for $\mathfrak{MN}$
unfolding *smc-FUNCT-Comp* **by** (*auto intro*: *smc-cs-intros*)
show *Comp-is-arr*: $\mathfrak{M}$ $\circ_{A\,\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{H}$
if $\mathfrak{M}$: $\mathfrak{G}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{H}$ **and** $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{G}$
for $\mathfrak{M}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{N}$ $\mathfrak{F}$
proof–
note *g* = *smc-FUNCT-is-arrD*[*OF that*(*1*)]
note *f* = *smc-FUNCT-is-arrD*[*OF that*(*2*)]
from *g*(*1*) *f*(*1*) **show** $\mathfrak{M}$ $\circ_{A\,\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{\textit{smc-FUNCT}\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{H}$
by (*subst g*(*2*), *subst g*(*4*), *subst f*(*2*), *subst f*(*3*), *remdups*)
(
cs-concl **cs-shallow**
cs-simp: *smc-FUNCT-cs-simps*
cs-intro: *smc-FUNCT-cs-intros cat-cs-intros*
)
qed
fix $\mathfrak{L}$ $\mathfrak{H}$ $\mathfrak{K}$ $\mathfrak{M}$ $\mathfrak{G}$ $\mathfrak{N}$ $\mathfrak{F}$
assume *prems*:

```
    𝔏 : ℌ ↦_smc-FUNCT α 𝔄 𝔅 𝔎
    𝔐 : 𝔊 ↦_smc-FUNCT α 𝔄 𝔅 ℌ
    𝔑 : 𝔉 ↦_smc-FUNCT α 𝔄 𝔅 𝔊
  note 𝔏 = smc-FUNCT-is-arrD[OF prems(1)]
  note 𝔐 = smc-FUNCT-is-arrD[OF prems(2)]
  note 𝔑 = smc-FUNCT-is-arrD[OF prems(3)]
  from 𝔏(1) 𝔐(1) 𝔑(1) show
    (𝔏 ∘_A smc-FUNCT α 𝔄 𝔅 𝔐) ∘_A smc-FUNCT α 𝔄 𝔅 𝔑 =
      𝔏 ∘_A smc-FUNCT α 𝔄 𝔅 (𝔐 ∘_A smc-FUNCT α 𝔄 𝔅 𝔑)
    by (subst (1 2) 𝔏(2), subst (1 2) 𝔐(2), subst (1 2) 𝔑(2), remdups)
      (
        cs-concl
          cs-simp: smc-FUNCT-cs-simps cat-cs-simps
          cs-intro: smc-FUNCT-cs-intros cat-cs-intros
      )
qed
  (
    simp-all add:
      assms
      smc-dg-FUNCT
      smc-FUNCT-components
      tiny-digraph-dg-FUNCT[OF assms(1,2)]
  )
```

24.3 *Funct*

24.3.1 Definition and elementary properties

```
definition smc-Funct :: V ⇒ V ⇒ V ⇒ V
  where smc-Funct α 𝔄 𝔅 =
    [
      tm-cf-maps α 𝔄 𝔅,
      tm-ntcf-arrows α 𝔄 𝔅,
      (λ𝔑∈₀tm-ntcf-arrows α 𝔄 𝔅. 𝔑(|NTDom|)),
      (λ𝔑∈₀tm-ntcf-arrows α 𝔄 𝔅. 𝔑(|NTCod|)),
      (λ𝔐𝔑∈₀composable-arrs (dg-Funct α 𝔄 𝔅). 𝔐𝔐(|0|) ∙_NTCF𝔄,𝔅 𝔐𝔐(|1_ℕ|))
    ]₀
```

Components.

```
lemma smc-Funct-components:
  shows smc-Funct α 𝔄 𝔅(|Obj|) = tm-cf-maps α 𝔄 𝔅
    and smc-Funct α 𝔄 𝔅(|Arr|) = tm-ntcf-arrows α 𝔄 𝔅
    and smc-Funct α 𝔄 𝔅(|Dom|) = (λ𝔑∈₀tm-ntcf-arrows α 𝔄 𝔅. 𝔑(|NTDom|))
    and smc-Funct α 𝔄 𝔅(|Cod|) = (λ𝔑∈₀tm-ntcf-arrows α 𝔄 𝔅. 𝔑(|NTCod|))
    and smc-Funct α 𝔄 𝔅(|Comp|) =
      (λ𝔐𝔑∈₀composable-arrs (dg-Funct α 𝔄 𝔅). 𝔐𝔐(|0|) ∙_NTCF𝔄,𝔅 𝔐𝔐(|1_ℕ|))
  unfolding smc-Funct-def dg-field-simps by (simp-all add: nat-omega-simps)
```

Slicing.

```
lemma smc-dg-Funct: smc-dg (smc-Funct α 𝔄 𝔅) = dg-Funct α 𝔄 𝔅
proof(rule vsv-eqI)
  show vsv (smc-dg (smc-Funct α 𝔄 𝔅)) unfolding smc-dg-def by auto
  show vsv (dg-Funct α 𝔄 𝔅) unfolding dg-Funct-def by auto
  have dom-lhs: 𝒟₀ (smc-dg (smc-Funct α 𝔄 𝔅)) = 4_ℕ
    unfolding smc-dg-def by (simp add: nat-omega-simps)
  have dom-rhs: 𝒟₀ (dg-Funct α 𝔄 𝔅) = 4_ℕ
    unfolding dg-Funct-def by (simp add: nat-omega-simps)
  show 𝒟₀ (smc-dg (smc-Funct α 𝔄 𝔅)) = 𝒟₀ (dg-Funct α 𝔄 𝔅)
```

unfolding *dom-lhs dom-rhs* **by** *simp*
show $a \in_{\circ} \mathcal{D}_{\circ}$ (*smc-dg* (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)) $\Longrightarrow$
smc-dg (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)(|a|) = *dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|a|)
for a
by
(
unfold dom-lhs,
elim-in-numeral,
unfold smc-dg-def dg-field-simps smc-Funct-def dg-Funct-def
)
(*auto simp*: *nat-omega-simps*)
qed

context *is-tm-ntcf*
begin

lemmas-with [*folded smc-dg-Funct*, *unfolded slicing-simps*]:
smc-Funct-Dom-app = *dg-Funct-Dom-app*
and *smc-Funct-Cod-app* = *dg-Funct-Cod-app*

end

lemmas [*smc-FUNCT-cs-simps*] =
is-tm-ntcf.smc-Funct-Dom-app
is-tm-ntcf.smc-Funct-Cod-app

lemmas-with [*folded smc-dg-Funct*, *unfolded slicing-simps*]:
smc-Funct-Dom-vsv[*intro*] = *dg-Funct-Dom-vsv*
and *smc-Funct-Dom-vdomain*[*smc-FUNCT-cs-simps*] = *dg-Funct-Dom-vdomain*
and *smc-Funct-Cod-vsv*[*intro*] = *dg-Funct-Cod-vsv*
and *smc-Funct-Cod-vdomain*[*smc-FUNCT-cs-simps*] = *dg-Funct-Cod-vdomain*
and *smc-Funct-Dom-vrange* = *dg-Funct-Dom-vrange*
and *smc-Funct-Cod-vrange* = *dg-Funct-Cod-vrange*
and *smc-Funct-is-arrI* = *dg-Funct-is-arrI*
and *smc-Funct-is-arrI'*[*smc-FUNCT-cs-intros*] = *dg-Funct-is-arrI'*
and *smc-Funct-is-arrD* = *dg-Funct-is-arrD*
and *smc-Funct-is-arrE*[*elim*] = *dg-Funct-is-arrE*

24.3.2 Composable arrows

lemma *smc-Funct-composable-arrs-dg-FUNCT*:
composable-arrs (*dg-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) = *composable-arrs* (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)
unfolding *composable-arrs-def smc-dg-Funct*[*symmetric*] *slicing-simps* **by** *auto*

lemma *smc-Funct-Comp*:
smc-Funct α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|) =
($\lambda \mathfrak{GF} \in_{\circ}$ *composable-arrs* (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$). $\mathfrak{GF}$(|0|) $\cdot_{NTCF_{\mathfrak{A},\mathfrak{B}}}$ $\mathfrak{GF}$(|$1_{\mathbb{N}}$|))
unfolding *smc-Funct-components smc-Funct-composable-arrs-dg-FUNCT* **..**

24.3.3 Composition

lemma *smc-Funct-Comp-vsv*[*intro*]: *vsv* (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|))
unfolding *smc-Funct-Comp* **by** *simp*

lemma *smc-Funct-Comp-vdomain*:
$\mathcal{D}_{\circ}$ (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Comp*|)) = *composable-arrs* (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)
unfolding *smc-Funct-Comp* **by** *auto*

lemma *smc-Funct-Comp-app*[*smc-FUNCT-cs-simps*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{H}}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}}$
shows $\mathfrak{M} \circ_{A\ \textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}} = \mathfrak{M} \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{N}$
proof–
from *assms* **have** $[\mathfrak{M}, \mathfrak{N}]_{\circ} \in_{\circ} \textit{composable-arrs}\ (\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B})$
by (*auto intro*: *smc-cs-intros*)
then show $\mathfrak{M} \circ_{A\ \textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}} = \mathfrak{M} \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{N}$
unfolding *smc-Funct-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-Funct-Comp-vrange*:
assumes *category* $\alpha\ \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ (\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|\textit{Comp}|\!)) \subseteq_{\circ} \textit{tm-ntcf-arrows}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
proof(*rule vsubsetI*)
fix $\mathfrak{L}$ **assume** $\mathfrak{L} \in_{\circ} \mathcal{R}_{\circ}\ (\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|\textit{Comp}|\!))$
then obtain $\mathfrak{MN}$
where $\mathfrak{L}$-*def*: $\mathfrak{L} = \textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|\textit{Comp}|\!)(\!|\mathfrak{MN}|\!)$
and $\mathfrak{MN} \in_{\circ} \mathcal{D}_{\circ}\ (\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|\textit{Comp}|\!))$
unfolding *smc-Funct-components*
by (*auto intro*: *smc-cs-intros*)
then obtain $\mathfrak{M}\ \mathfrak{N}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{H}$
where $\mathfrak{MN} = [\mathfrak{M}, \mathfrak{N}]_{\circ}$
and $\mathfrak{M}$: $\mathfrak{M} : \mathfrak{G} \mapsto_{\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{H}}$
and $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{F} \mapsto_{\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}}$
by (*auto simp*: *smc-Funct-Comp-vdomain*)
with $\mathfrak{L}$-*def* **have** $\mathfrak{L}$-*def'*: $\mathfrak{L} = \mathfrak{M} \circ_{A\ \textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{N}}$ **by** *simp*
note $\mathfrak{M} = \textit{smc-Funct-is-arrD}[\textit{OF}\ \mathfrak{M}]$
and $\mathfrak{N} = \textit{smc-Funct-is-arrD}[\textit{OF}\ \mathfrak{N}]$
from *assms* $\mathfrak{M}(1)\ \mathfrak{N}(1)$ **show** $\mathfrak{L} \in_{\circ} \textit{tm-ntcf-arrows}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$
unfolding $\mathfrak{L}$-*def'*
by (*subst* $\mathfrak{M}(2)$, *use nothing* **in** ‹*subst* $\mathfrak{N}(2)$›)
(
cs-concl
cs-simp: *smc-FUNCT-cs-simps*
cs-intro: *smc-FUNCT-cs-intros cat-small-cs-intros*
)
qed

24.3.4 *Funct* is a semicategory

lemma *semicategory-smc-Funct*:
assumes *tiny-category* $\alpha\ \mathfrak{A}$ **and** *category* $\alpha\ \mathfrak{B}$
shows *semicategory* $\alpha\ (\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B})$ (**is** ‹*semicategory* α *?Funct*›)
proof–
interpret *tiny-category* $\alpha\ \mathfrak{A}$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro semicategoryI*)
show *vfsequence ?Funct* **by** (*simp add*: *smc-Funct-def*)
show *vcard ?Funct* = $5_{\mathbf{N}}$
unfolding *smc-Funct-def* **by** (*simp add*: *nat-omega-simps*)
show $(\mathfrak{MN} \in_{\circ} \mathcal{D}_{\circ}\ (\textit{smc-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}(\!|\textit{Comp}|\!))) =$
$(\exists \mathfrak{M}\ \mathfrak{N}\ \mathfrak{G}\ \mathfrak{H}\ \mathfrak{F}.\ \mathfrak{MN} = [\mathfrak{M}, \mathfrak{N}]_{\circ} \wedge \mathfrak{M} : \mathfrak{G} \mapsto_{\textit{?Funct}} \mathfrak{H} \wedge \mathfrak{N} : \mathfrak{F} \mapsto_{\textit{?Funct}} \mathfrak{G})$
for $\mathfrak{MN}$
unfolding *smc-Funct-Comp* **by** (*auto intro*: *smc-cs-intros*)
show *Comp-is-arr*: $\mathfrak{M} \circ_{A\ \textit{?Funct}} \mathfrak{N} : \mathfrak{F} \mapsto_{\textit{?Funct}} \mathfrak{H}$
if $\mathfrak{M} : \mathfrak{G} \mapsto_{\textit{?Funct}} \mathfrak{H}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{\textit{?Funct}} \mathfrak{G}$
for $\mathfrak{M}\ \mathfrak{G}\ \mathfrak{H}\ \mathfrak{N}\ \mathfrak{F}$
proof–

```
    note 𝔐 = smc-Funct-is-arrD[OF that(1)]
    note 𝔑 = smc-Funct-is-arrD[OF that(2)]
    from assms 𝔐(1) 𝔑(1) show
      𝔐 ∘A ?Funct 𝔑 : 𝔉 ↦?Funct ℌ
      by (subst 𝔐(2), use nothing in ‹subst 𝔐(4), subst 𝔑(2), subst 𝔑(3)›)
        (
          cs-concl
            cs-simp: smc-FUNCT-cs-simps
            cs-intro: smc-FUNCT-cs-intros cat-small-cs-intros
        )
  qed
  show 𝔏 ∘A ?Funct 𝔐 ∘A ?Funct 𝔑 = 𝔏 ∘A ?Funct (𝔐 ∘A ?Funct 𝔑)
    if 𝔏 : ℌ ↦?Funct 𝔎 𝔐 : 𝔊 ↦?Funct ℌ 𝔑 : 𝔉 ↦?Funct 𝔊
    for 𝔏 ℌ 𝔎 𝔐 𝔊 𝔑 𝔉
  proof–
    note 𝔏 = smc-Funct-is-arrD[OF that(1)]
    note 𝔐 = smc-Funct-is-arrD[OF that(2)]
    note 𝔑 = smc-Funct-is-arrD[OF that(3)]
    from assms 𝔏(1) 𝔐(1) 𝔑(1) show
      (𝔏 ∘A ?Funct 𝔐) ∘A ?Funct 𝔑 = 𝔏 ∘A ?Funct (𝔐 ∘A ?Funct 𝔑)
      by
        (
          subst (1 2) 𝔏(2),
          use nothing in ‹subst (1 2) 𝔐(2), subst (1 2) 𝔑(2)›
        )
        (
          cs-concl
            cs-simp: smc-FUNCT-cs-simps cat-cs-simps cat-small-cs-simps
            cs-intro: smc-FUNCT-cs-intros cat-cs-intros cat-small-cs-intros
        )
  qed
qed (auto simp: assms smc-dg-Funct smc-Funct-components digraph-dg-Funct)
qed
```

24.3.5 *Funct* is a subsemicategory of *FUNCT*

```
lemma subsemicategory-smc-Funct-smc-FUNCT:
  assumes 𝒵 β and α ∈∘ β and tiny-category α 𝔄 and category α 𝔅
  shows smc-Funct α 𝔄 𝔅 ⊆SMCβ smc-FUNCT α 𝔄 𝔅
proof(intro subsemicategoryI, unfold smc-dg-FUNCT smc-dg-Funct)
  interpret category α 𝔅 by (rule assms(4))
  interpret smc-Funct: semicategory α ‹smc-Funct α 𝔄 𝔅›
    by (rule semicategory-smc-Funct[OF assms(3,4)])
  show semicategory β (smc-Funct α 𝔄 𝔅)
    by (rule semicategory.smc-semicategory-if-ge-Limit[OF - assms(1,2)])
      (auto simp: smc-cs-simps intro: smc-cs-intros)
  from assms show semicategory β (smc-FUNCT α 𝔄 𝔅)
    by
      (
        cs-concl
          cs-intro: smc-small-cs-intros tiny-semicategory-smc-FUNCT
      )
  show dg-Funct α 𝔄 𝔅 ⊆DGβ dg-FUNCT α 𝔄 𝔅
    by (rule subdigraph-dg-Funct-dg-FUNCT[OF assms])
  show 𝔐 ∘A smc-Funct α 𝔄 𝔅 𝔑 = 𝔐 ∘A smc-FUNCT α 𝔄 𝔅 𝔑
    if 𝔐 : 𝔊 ↦smc-Funct α 𝔄 𝔅 ℌ and 𝔑 : 𝔉 ↦smc-Funct α 𝔄 𝔅 𝔊
    for 𝔊 ℌ 𝔐 𝔉 𝔑
  proof–
```

```
    note 𝔐 = smc-Funct-is-arrD[OF that(1)]
    note 𝔑 = smc-Funct-is-arrD[OF that(2)]
    from 𝔐(1) 𝔑(1) show ?thesis
      by (subst (1 2) 𝔐(2), use nothing in ‹subst (1 2) 𝔑(2)›)
        (
          cs-concl cs-shallow
            cs-simp: smc-FUNCT-cs-simps cat-small-cs-simps
            cs-intro: smc-FUNCT-cs-intros cat-small-cs-intros
        )
  qed
qed
```

25 *FUNCT* and *Funct*

25.1 Background

The subsection presents the theory of the categories of α-functors between two α-categories. It continues the development that was initiated in sections 23 and 24. A general reference for this section is Chapter II-4 in [7].

named-theorems *cat-FUNCT-cs-simps*
named-theorems *cat-FUNCT-cs-intros*

lemmas (**in** *is-functor*) [*cat-FUNCT-cs-simps*] = *cat-map-cs-simps*
lemmas (**in** *is-functor*) [*cat-FUNCT-cs-intros*] = *cat-map-cs-intros*

lemmas [*cat-FUNCT-cs-simps*] = *cat-map-cs-simps*
lemmas [*cat-FUNCT-cs-intros*] = *cat-map-cs-intros*

25.2 *FUNCT*

25.2.1 Definition and elementary properties

definition *cat-FUNCT* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$ =
    [
      cf-maps α $\mathfrak{A}$ $\mathfrak{B}$,
      ntcf-arrows α $\mathfrak{A}$ $\mathfrak{B}$,
      $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTDom|\!))$,
      $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTCod|\!))$,
      $(\lambda\mathfrak{MN}\in_\circ$*composable-arrs* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$). $\mathfrak{MN}(\!|0|\!) \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{MN}(\!|1_{\mathbb{N}}|\!))$,
      $(\lambda\mathfrak{F}\in_\circ$*cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$. *ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F})$
    ]$_\circ$

Components.

lemma *cat-FUNCT-components*:
  shows [*cat-FUNCT-cs-simps*]: *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Obj|\!)$ = *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
    and *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Arr|\!)$ = *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    and *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Dom|\!)$ = $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTDom|\!))$
    and *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Cod|\!)$ = $(\lambda\mathfrak{N}\in_\circ$*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}(\!|NTCod|\!))$
    and *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|Comp|\!)$ =
      $(\lambda\mathfrak{MN}\in_\circ$*composable-arrs* (*dg-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$). $\mathfrak{MN}(\!|0|\!) \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{MN}(\!|1_{\mathbb{N}}|\!))$
    and *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|CId|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$. *ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F})$
  unfolding *cat-FUNCT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-FUNCT*: *cat-smc* (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$
proof(*rule vsv-eqI*)
  show *vsv* (*cat-smc* (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) **unfolding** *cat-smc-def* **by** *auto*
  show *vsv* (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) **unfolding** *smc-FUNCT-def* **by** *auto*
  have *dom-lhs*: $\mathcal{D}_\circ$ (*cat-smc* (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) = $5_{\mathbb{N}}$
    unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) = $5_{\mathbb{N}}$
    unfolding *smc-FUNCT-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*cat-smc* (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) = $\mathcal{D}_\circ$ (*smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show *a* $\in_\circ$ $\mathcal{D}_\circ$ (*cat-smc* (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)) $\Longrightarrow$
    cat-smc (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$)$(\!|a|\!)$ = *smc-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}(\!|a|\!)$
    for *a*
    by

(
unfold dom-lhs,
elim-in-numeral,
unfold cat-smc-def dg-field-simps cat-FUNCT-def smc-FUNCT-def
)
(*auto simp*: *nat-omega-simps*)
qed

context *is-ntcf*
begin

lemmas-with [*folded cat-smc-FUNCT*, *unfolded slicing-simps*]:
cat-FUNCT-Dom-app = *smc-FUNCT-Dom-app*
and *cat-FUNCT-Cod-app* = *smc-FUNCT-Cod-app*

end

lemmas [*smc-FUNCT-cs-simps*] =
is-ntcf.cat-FUNCT-Dom-app
is-ntcf.cat-FUNCT-Cod-app

lemmas-with [*folded cat-smc-FUNCT*, *unfolded slicing-simps*]:
cat-FUNCT-Dom-vsv[*intro*] = *smc-FUNCT-Dom-vsv*
and *cat-FUNCT-Dom-vdomain*[*cat-FUNCT-cs-simps*] = *smc-FUNCT-Dom-vdomain*
and *cat-FUNCT-Cod-vsv*[*intro*] = *smc-FUNCT-Cod-vsv*
and *cat-FUNCT-Cod-vdomain*[*cat-FUNCT-cs-simps*] = *smc-FUNCT-Cod-vdomain*
and *cat-FUNCT-Dom-vrange* = *smc-FUNCT-Dom-vrange*
and *cat-FUNCT-Cod-vrange* = *smc-FUNCT-Cod-vrange*
and *cat-FUNCT-is-arrI* = *smc-FUNCT-is-arrI*
and *cat-FUNCT-is-arrI'*[*cat-FUNCT-cs-intros*] = *smc-FUNCT-is-arrI'*
and *cat-FUNCT-is-arrD* = *smc-FUNCT-is-arrD*
and *cat-FUNCT-is-arrE*[*elim*] = *smc-FUNCT-is-arrE*

lemmas-with [*folded cat-smc-FUNCT*, *unfolded slicing-simps*]:
cat-FUNCT-Comp-app[*cat-FUNCT-cs-simps*] = *smc-FUNCT-Comp-app*

25.2.2 Identity

mk-VLambda *cat-FUNCT-components*(*6*)
|*vsv cat-FUNCT-CId-vsv*[*cat-FUNCT-cs-intros*]|
|*vdomain cat-FUNCT-CId-vdomain*[*cat-FUNCT-cs-simps*]|
|*app cat-FUNCT-CId-app*[*cat-FUNCT-cs-simps*]|

lemma *smc-FUNCT-CId-vrange*: $\mathcal{R}_{\circ}$ (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*CId*|)) $\subseteq_{\circ}$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding *cat-FUNCT-components*
proof(*rule vrange-VLambda-vsubset*)
fix x **assume** $x \in_{\circ}$ *cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
then obtain $\mathfrak{F}$ **where** *x-def*: x = *cf-map* $\mathfrak{F}$ **and** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
by *clarsimp*
then show *ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ $x \in_{\circ}$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding *x-def*
by
(
cs-concl
cs-simp: *cat-FUNCT-cs-simps* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*
)
qed

25.2.3 The conversion of a natural transformation arrow to a natural transformation is a bijection

lemma *bij-betw-ntcf-of-ntcf-arrow*:
  bij-betw
    (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$)
    (*elts* (*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$))
    (*elts* (*ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$))
proof(*intro bij-betw-imageI inj-onI subset-antisym subsetI*)
  fix $\mathfrak{M}$ $\mathfrak{N}$ **assume** *prems*:
    $\mathfrak{M}$ $\in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    $\mathfrak{N}$ $\in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    ntcf-of-ntcf-arrow $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{M}$ = *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{N}$
  from *prems*(*1*) **obtain** $\mathfrak{M}'$ $\mathfrak{F}$ $\mathfrak{G}$
    where $\mathfrak{M}$-*def*: $\mathfrak{M}$ = *ntcf-arrow* $\mathfrak{M}'$ **and** $\mathfrak{M}'$: $\mathfrak{M}' : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by *auto*
  from *prems*(*2*) **obtain** $\mathfrak{N}'$ $\mathfrak{F}'$ $\mathfrak{G}'$
    where $\mathfrak{N}$-*def*: $\mathfrak{N}$ = *ntcf-arrow* $\mathfrak{N}'$ **and** $\mathfrak{N}'$: $\mathfrak{N}' : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by *auto*
  from *prems*(*3*) **have** $\mathfrak{M}'$ = $\mathfrak{N}'$
    unfolding
      $\mathfrak{M}$-*def*
      $\mathfrak{N}$-*def*
      is-ntcf.ntcf-of-ntcf-arrow[*OF* $\mathfrak{M}'$]
      is-ntcf.ntcf-of-ntcf-arrow[*OF* $\mathfrak{N}'$]
    by *simp*
  then show $\mathfrak{M}$ = $\mathfrak{N}$ **unfolding** $\mathfrak{M}$-*def* $\mathfrak{N}$-*def* **by** *auto*
next
  fix $\mathfrak{M}$ **assume**
    $\mathfrak{M}$ $\in$ *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ ‘ *elts* (*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$)
  then obtain $\mathfrak{M}'$ **where** $\mathfrak{M}'$: $\mathfrak{M}'$ $\in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    and $\mathfrak{M}$-*def*: $\mathfrak{M}$ = *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{M}'$
    by *auto*
  from $\mathfrak{M}'$ **obtain** $\mathfrak{M}''$ $\mathfrak{F}$ $\mathfrak{G}$
    where $\mathfrak{M}'$-*def*: $\mathfrak{M}'$ = *ntcf-arrow* $\mathfrak{M}''$
      and $\mathfrak{M}''$: $\mathfrak{M}'' : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    by *auto*
  from $\mathfrak{M}''$ **show** $\mathfrak{M}$ $\in_\circ$ *ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$
    unfolding $\mathfrak{M}$-*def* $\mathfrak{M}'$-*def* *is-ntcf.ntcf-of-ntcf-arrow*[*OF* $\mathfrak{M}''$] **by** *auto*
next
  fix $\mathfrak{M}$ **assume** $\mathfrak{M}$ $\in_\circ$ *ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$
  then obtain $\mathfrak{F}$ $\mathfrak{G}$ **where** $\mathfrak{M}$: $\mathfrak{M} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **by** *clarsimp*
  then have $\mathfrak{M}$ = *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ (*ntcf-arrow* $\mathfrak{M}$)
    by (*cs-concl* **cs-shallow cs-simp**: *cat-FUNCT-cs-simps*)
  moreover from $\mathfrak{M}$ **have** *ntcf-arrow* $\mathfrak{M}$ $\in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    by (*cs-concl* **cs-intro**: *cat-FUNCT-cs-intros*)
  ultimately show $\mathfrak{M}$ $\in$ *ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$ ‘ *elts* (*ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$)
    by *simp*
qed

lemma *bij-betw-ntcf-of-ntcf-arrow-Hom*:
  assumes $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$ **and** $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
  shows *bij-betw*
    (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ $\mathfrak{B}$)
    (*elts* (*Hom* (*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$) (*cf-map* $\mathfrak{F}$) (*cf-map* $\mathfrak{G}$)))
    (*elts* (*these-ntcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$))
proof–

```
  interpret 𝔉: is-functor α 𝔄 𝔅 𝔉 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔄 𝔅 𝔊  by (rule assms(2))

  from assms have [cat-cs-simps]:
    cf-of-cf-map 𝔄 𝔅 (cf-map 𝔉) = 𝔉
    cf-of-cf-map 𝔄 𝔅 (cf-map 𝔊) = 𝔊
    by (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)+

  show ?thesis
  proof
    (
      rule bij-betw-subset[OF bij-betw-ntcf-of-ntcf-arrow];
      (intro subset-antisym subsetI)?;
      (unfold in-Hom-iff)?
    )
    fix 𝔑 assume prems: 𝔑 : cf-map 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 cf-map 𝔊
    note 𝔑 = cat-FUNCT-is-arrD[OF prems, unfolded cat-cs-simps]
    from 𝔑(1) show 𝔑 ∈∘ ntcf-arrows α 𝔄 𝔅
      by (subst 𝔑(2)) (cs-concl cs-intro: cat-FUNCT-cs-intros)
  next
    fix 𝔑 assume
      𝔑 ∈ ntcf-of-ntcf-arrow 𝔄 𝔅 ‘
        elts (Hom (cat-FUNCT α 𝔄 𝔅) (cf-map 𝔉) (cf-map 𝔊))
    then obtain 𝔑'
      where 𝔑': 𝔑' ∈∘ Hom (cat-FUNCT α 𝔄 𝔅) (cf-map 𝔉) (cf-map 𝔊)
        and 𝔑-def: 𝔑 = ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑'
      by auto
    note 𝔑' = cat-FUNCT-is-arrD[
        OF 𝔑'[unfolded cat-cs-simps], unfolded cat-cs-simps
        ]
    from 𝔑'(1) show 𝔑 ∈∘ these-ntcfs α 𝔄 𝔅 𝔉 𝔊 unfolding 𝔑-def by simp
  next
    fix 𝔑 assume 𝔑 ∈∘ these-ntcfs α 𝔄 𝔅 𝔉 𝔊
    then have 𝔑: 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅 by simp
    then have 𝔑 = ntcf-of-ntcf-arrow 𝔄 𝔅 (ntcf-arrow 𝔑)
      by (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)
    moreover from 𝔑 have
      ntcf-arrow 𝔑 ∈∘ Hom (cat-FUNCT α 𝔄 𝔅) (cf-map 𝔉) (cf-map 𝔊)
      unfolding in-Hom-iff by (cs-concl cs-shallow cs-intro: cat-FUNCT-cs-intros)
    ultimately show
      𝔑 ∈ ntcf-of-ntcf-arrow 𝔄 𝔅 ‘
        elts (Hom (cat-FUNCT α 𝔄 𝔅) (cf-map 𝔉) (cf-map 𝔊))
      by simp
  qed

qed
```

25.2.4 *FUNCT* is a category

```
lemma (in 𝒵) tiny-category-cat-FUNCT[cat-FUNCT-cs-intros]:
  assumes 𝒵 β and α ∈∘ β
  shows tiny-category β (cat-FUNCT α 𝔄 𝔅) (is ‹tiny-category β ?FUNCT›)
proof(intro tiny-categoryI)
  show vfsequence ?FUNCT unfolding cat-FUNCT-def by auto
  show vcard ?FUNCT = 6_ℕ
    unfolding cat-FUNCT-def by (simp add: nat-omega-simps)
  from assms show tiny-semicategory β (cat-smc ?FUNCT)
    unfolding cat-smc-FUNCT
```

```
    by (auto simp add: tiny-semicategory-smc-FUNCT)
  show CId-a: ?FUNCT(|CId|)(|𝔉'|) : 𝔉' ↦_?FUNCT 𝔉' if 𝔉' ∈_∘ ?FUNCT(|Obj|) for 𝔉'
  proof–
    from that obtain 𝔉 where 𝔉'-def: 𝔉' = cf-map 𝔉 and 𝔉: 𝔉 : 𝔄 ↦↦_Cα 𝔅
      unfolding cat-FUNCT-components by clarsimp
    show ?thesis
      using that 𝔉
      unfolding cat-FUNCT-components(1) 𝔉'-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
  show ?FUNCT(|CId|)(|𝔊|) ∘_A ?FUNCT 𝔑 = 𝔑 if 𝔑 : 𝔉 ↦_?FUNCT 𝔊 for 𝔑 𝔉 𝔊
  proof–
    from that obtain 𝔑' 𝔉' 𝔊'
      where 𝔑': 𝔑' : 𝔉' ↦_CF 𝔊' : 𝔄 ↦↦_Cα 𝔅
        and 𝔑-def[cat-FUNCT-cs-simps]: 𝔑 = ntcf-arrow 𝔑'
        and 𝔉-def[cat-FUNCT-cs-simps]: 𝔉 = cf-map 𝔉'
        and 𝔊-def[cat-FUNCT-cs-simps]: 𝔊 = cf-map 𝔊'
      by auto
    from 𝔑' show cat-FUNCT α 𝔄 𝔅(|CId|)(|𝔊|) ∘_A cat-FUNCT α 𝔄 𝔅 𝔑 = 𝔑
      by
        (
          cs-concl
            cs-simp: cat-FUNCT-cs-simps cat-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
  show 𝔑 ∘_A ?FUNCT ?FUNCT(|CId|)(|𝔊|) = 𝔑 if 𝔑 : 𝔊 ↦_?FUNCT ℌ for 𝔑 𝔊 ℌ
  proof–
    note 𝔑 = cat-FUNCT-is-arrD[OF that]
    from 𝔑(1) show 𝔑 ∘_A cat-FUNCT α 𝔄 𝔅 cat-FUNCT α 𝔄 𝔅(|CId|)(|𝔊|) = 𝔑
      by (subst (1 2) 𝔑(2), subst 𝔑(3), remdups)
        (
          cs-concl
            cs-simp: cat-FUNCT-cs-simps cat-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
qed (simp-all add: assms cat-FUNCT-components)

lemmas (in 𝒵) [cat-FUNCT-cs-intros] = tiny-category-cat-FUNCT
```

25.2.5 Isomorphism

```
lemma cat-FUNCT-is-iso-arrI:
  assumes 𝔑 : 𝔉 ↦_CF.iso 𝔊 : 𝔄 ↦↦_Cα 𝔅
  shows ntcf-arrow 𝔑 : cf-map 𝔉 ↦_iso cat-FUNCT α 𝔄 𝔅 cf-map 𝔊
proof(intro is-iso-arrI is-inverseI)
  interpret 𝔑: is-iso-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms)
  show is-arr-𝔑: ntcf-arrow 𝔑 : cf-map 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 cf-map 𝔊
    by (simp add: assms cat-FUNCT-is-arrI is-iso-ntcf.axioms(1))
  interpret inv-𝔑: is-iso-ntcf α 𝔄 𝔅 𝔊 𝔉 ‹inv-ntcf 𝔑›
    using CZH-ECAT-NTCF.iso-ntcf-is-iso-arr(1)[OF assms] by simp
  from assms show is-arr-inv-𝔑:
```

```
    ntcf-arrow (inv-ntcf 𝔑) : cf-map 𝔊 ↦_cat-FUNCT α 𝔄 𝔅 cf-map 𝔉
    by
      (
        cs-concl cs-shallow cs-intro:
          ntcf-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from assms show ntcf-arrow 𝔑 : cf-map 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 cf-map 𝔊
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
  from assms show
    ntcf-arrow (inv-ntcf 𝔑) ∘_A cat-FUNCT α 𝔄 𝔅 ntcf-arrow 𝔑 =
      cat-FUNCT α 𝔄 𝔅(| CId |)(| cf-map 𝔉 |)
    ntcf-arrow 𝔑 ∘_A cat-FUNCT α 𝔄 𝔅 ntcf-arrow (inv-ntcf 𝔑) =
      cat-FUNCT α 𝔄 𝔅(| CId |)(| cf-map 𝔊 |)
    by
      (
        cs-concl cs-shallow
          cs-simp: iso-ntcf-is-iso-arr(2,3) cat-FUNCT-cs-simps
          cs-intro: ntcf-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )+
qed

lemma cat-FUNCT-is-iso-arrI'[cat-FUNCT-cs-intros]:
  assumes 𝔑 : 𝔉 ↦_CF.iso 𝔊 : 𝔄 ↦↦_Cα 𝔅
    and 𝔑' = ntcf-arrow 𝔑
    and 𝔉' = cf-map 𝔉
    and 𝔊' = cf-map 𝔊
  shows 𝔑' : 𝔉' ↦_iso cat-FUNCT α 𝔄 𝔅 cf-map 𝔊
  using assms(1) unfolding assms(2–4) by (rule cat-FUNCT-is-iso-arrI)

lemma cat-FUNCT-is-iso-arrD:
  assumes 𝔑 : 𝔉 ↦_iso cat-FUNCT α 𝔄 𝔅 𝔊 (is ‹𝔑 : 𝔉 ↦_iso ?FUNCT 𝔊›)
  shows ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑 :
    cf-of-cf-map 𝔄 𝔅 𝔉 ↦_CF.iso cf-of-cf-map 𝔄 𝔅 𝔊 : 𝔄 ↦↦_Cα 𝔅
    and 𝔑 = ntcf-arrow (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑)
    and 𝔉 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔉)
    and 𝔊 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔊)
proof–
  from assms(1) have 𝔑: 𝔑 : 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 𝔊
    unfolding is-iso-arr-def by simp
  interpret 𝒵 α by (rule is-ntcfD[OF cat-FUNCT-is-arrD(1)[OF 𝔑]])
  define β where β = α + ω
  have 𝒵β: 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: 𝒵-α-αω 𝒵.intro 𝒵-Limit-αω 𝒵-ω-αω β-def)
  interpret FUNCT: tiny-category β ?FUNCT
    by (rule tiny-category-cat-FUNCT[OF 𝒵β αβ, of 𝔄 𝔅])
  have inv-𝔑: 𝔑⁻¹_C ?FUNCT : 𝔊 ↦_iso ?FUNCT 𝔉
    and inv-𝔑-𝔑: 𝔑⁻¹_C ?FUNCT ∘_A ?FUNCT 𝔑 = ?FUNCT(| CId |)(| 𝔉 |)
    and 𝔑-inv-𝔑: 𝔑 ∘_A ?FUNCT 𝔑⁻¹_C ?FUNCT = ?FUNCT(| CId |)(| 𝔊 |)
    by
      (
        intro
          FUNCT.cat-the-inverse-is-iso-arr[OF assms]
          FUNCT.cat-the-inverse-Comp-CId[OF assms]
      )+
  from assms is-iso-arrD inv-𝔑
  have 𝔑-is-arr: 𝔑 : 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 𝔊
    and inv-𝔑-is-arr: 𝔑⁻¹_C ?FUNCT : 𝔊 ↦_cat-FUNCT α 𝔄 𝔅 𝔉
    by auto
```

```
  note 𝔑-is-arr = cat-FUNCT-is-arrD[OF 𝔑-is-arr]
  note inv-𝔑-is-arr = cat-FUNCT-is-arrD[OF inv-𝔑-is-arr]
  let ?𝔑 = ‹ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑›
    and ?inv-𝔑 = ‹ntcf-of-ntcf-arrow 𝔄 𝔅 (𝔑⁻¹_C cat-FUNCT α 𝔄 𝔅)›
  from inv-𝔑-𝔑 𝔑-is-arr(1) inv-𝔑-is-arr(1) have inv-𝔑-𝔑:
    ?inv-𝔑 ∘_NTCF ?𝔑 = ntcf-id (cf-of-cf-map 𝔄 𝔅 𝔉)
    by
      (
        subst (asm) inv-𝔑-is-arr(2),
        use nothing in ‹subst (asm) (2) 𝔑-is-arr(2), subst (asm) 𝔑-is-arr(3)›
      )
      (
        cs-prems cs-shallow
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-FUNCT-cs-intros cat-cs-intros
      )
  from 𝔑-inv-𝔑 inv-𝔑-is-arr(1) 𝔑-is-arr(1) have 𝔑-inv-𝔑:
    ?𝔑 ∘_NTCF ?inv-𝔑 = ntcf-id (cf-of-cf-map 𝔄 𝔅 𝔊)
    by
      (
        subst (asm) inv-𝔑-is-arr(2),
        use nothing in ‹subst (asm) 𝔑-is-arr(2), subst (asm) 𝔑-is-arr(4)›
      )
      (
        cs-prems cs-shallow
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-FUNCT-cs-intros cat-cs-intros
      )
  show ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑 :
    cf-of-cf-map 𝔄 𝔅 𝔉 ↦_CF.iso cf-of-cf-map 𝔄 𝔅 𝔊 : 𝔄 ↦↦_Cα 𝔅
    by
      (
        rule CZH-ECAT-NTCF.is-iso-arr-is-iso-ntcf[
          OF 𝔑-is-arr(1) inv-𝔑-is-arr(1) 𝔑-inv-𝔑 inv-𝔑-𝔑
          ]
      )
  show 𝔑 = ntcf-arrow (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑)
    and 𝔉 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔉)
    and 𝔊 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔊)
    by (intro 𝔑-is-arr(2–4))+
qed
```

25.3 *Funct*

25.3.1 Definition and elementary properties

```
definition cat-Funct :: V ⇒ V ⇒ V ⇒ V
  where cat-Funct α 𝔄 𝔅 =
    [
      tm-cf-maps α 𝔄 𝔅,
      tm-ntcf-arrows α 𝔄 𝔅,
      (λ𝔑∈₀tm-ntcf-arrows α 𝔄 𝔅. 𝔑(|NTDom|)),
      (λ𝔑∈₀tm-ntcf-arrows α 𝔄 𝔅. 𝔑(|NTCod|)),
      (λ𝔐𝔑∈₀composable-arrs (dg-Funct α 𝔄 𝔅). 𝔐𝔑(|0|) ∘_NTCF_𝔄,𝔅 𝔐𝔑(|1_ℕ|)),
      (λ𝔉∈₀tm-cf-maps α 𝔄 𝔅. ntcf-arrow-id 𝔄 𝔅 𝔉)
    ]₀
```

Components.

lemma *cat-Funct-components*:
  shows [*cat-FUNCT-cs-simps*]: *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor Obj \rfloor)$ = *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
    and *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor Arr \rfloor)$ = *tm-ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
    and *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor Dom \rfloor)$ = $(\lambda \mathfrak{N} \in_{\circ} \textit{tm-ntcf-arrows}\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N}(\lfloor NTDom \rfloor))$
    and *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor Cod \rfloor)$ = $(\lambda \mathfrak{N} \in_{\circ} \textit{tm-ntcf-arrows}\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N}(\lfloor NTCod \rfloor))$
    and *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor Comp \rfloor)$ =
      $(\lambda \mathfrak{MN} \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Funct}\ \alpha\ \mathfrak{A}\ \mathfrak{B}).\ \mathfrak{MN}(\lfloor 0 \rfloor) \cdot_{NTCF\mathfrak{A},\mathfrak{B}} \mathfrak{MN}(\lfloor 1_{\mathbb{N}} \rfloor))$
    and *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor CId \rfloor)$ = $(\lambda \mathfrak{F} \in_{\circ} \textit{tm-cf-maps}\ \alpha\ \mathfrak{A}\ \mathfrak{B}.\ \textit{ntcf-arrow-id}\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F})$
  unfolding *cat-Funct-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *cat-smc-Funct*: *cat-smc* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) = *smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$
proof(*rule vsv-eqI*)
  show *vsv* (*cat-smc* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)) **unfolding** *cat-smc-def* **by** *auto*
  show *vsv* (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) **unfolding** *smc-Funct-def* **by** *auto*
  have *dom-lhs*: $\mathcal{D}_{\circ}$ (*cat-smc* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)) = $5_{\mathbb{N}}$
    unfolding *cat-smc-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_{\circ}$ (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) = $5_{\mathbb{N}}$
    unfolding *smc-Funct-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_{\circ}$ (*cat-smc* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)) = $\mathcal{D}_{\circ}$ (*smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show *a* $\in_{\circ}$ $\mathcal{D}_{\circ}$ (*cat-smc* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)) $\Longrightarrow$
    cat-smc (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)($\lfloor a \rfloor$) = *smc-Funct* α $\mathfrak{A}$ $\mathfrak{B}(\lfloor a \rfloor)$
    for *a*
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold cat-smc-def dg-field-simps cat-Funct-def smc-Funct-def
)
      (*auto simp*: *nat-omega-simps*)
qed

context *is-tm-ntcf*
begin

lemmas-with [*folded cat-smc-Funct*, *unfolded slicing-simps*]:
  cat-Funct-Dom-app = *smc-Funct-Dom-app*
  and *cat-Funct-Cod-app* = *smc-Funct-Cod-app*

end

lemmas [*cat-FUNCT-cs-simps*] =
  is-tm-ntcf.cat-Funct-Dom-app
  is-tm-ntcf.cat-Funct-Cod-app

lemmas-with [*folded cat-smc-Funct*, *unfolded slicing-simps*]:
  cat-Funct-Dom-vsv[*cat-FUNCT-cs-intros*] = *smc-Funct-Dom-vsv*
  and *cat-Funct-Dom-vdomain*[*cat-FUNCT-cs-simps*] = *smc-Funct-Dom-vdomain*
  and *cat-Funct-Cod-vsv*[*cat-FUNCT-cs-intros*] = *smc-Funct-Cod-vsv*
  and *cat-Funct-Cod-vdomain*[*cat-FUNCT-cs-simps*] = *smc-Funct-Cod-vdomain*
  and *cat-Funct-Dom-vrange* = *smc-Funct-Dom-vrange*
  and *cat-Funct-Cod-vrange* = *smc-Funct-Cod-vrange*
  and *cat-Funct-is-arrI* = *smc-Funct-is-arrI*
  and *cat-Funct-is-arrI'*[*cat-FUNCT-cs-intros*] = *smc-Funct-is-arrI'*
  and *cat-Funct-is-arrD* = *smc-Funct-is-arrD*
  and *cat-Funct-is-arrE*[*elim*] = *smc-Funct-is-arrE*

lemmas-with [*folded cat-smc-Funct*, *unfolded slicing-simps*]:
cat-Funct-Comp-app[*cat-FUNCT-cs-simps*] = *smc-Funct-Comp-app*

25.3.2 Identity

mk-VLambda *cat-Funct-components*(*6*)
|*vsv cat-Funct-CId-vsv*[*intro*]|
|*vdomain cat-Funct-CId-vdomain*[*cat-FUNCT-cs-simps*]|
|*app cat-Funct-CId-app*[*cat-FUNCT-cs-simps*]|

lemma *smc-Funct-CId-vrange*: $\mathcal{R}_\circ$ (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*CId*|)) $\subseteq_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding *cat-Funct-components*
proof(*rule vrange-VLambda-vsubset*)
fix $\mathfrak{F}'$ **assume** $\mathfrak{F}' \in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
then obtain $\mathfrak{F}$ **where** $\mathfrak{F}'$-*def*: $\mathfrak{F}'$ = *cf-map* $\mathfrak{F}$ **and** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
by *clarsimp*
then show *ntcf-arrow-id* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}' \in_\circ$ *ntcf-arrows* α $\mathfrak{A}$ $\mathfrak{B}$
by
(
cs-concl
cs-simp: *cat-FUNCT-cs-simps* $\mathfrak{F}'$-*def*
cs-intro: *cat-FUNCT-cs-intros cat-small-cs-intros*
)
qed

25.3.3 *Funct* is a category

lemma *category-cat-Funct*:
assumes *tiny-category* α $\mathfrak{A}$ **and** *category* α $\mathfrak{B}$
shows *category* α (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) (**is** ‹*category* α *?Funct*›)
proof–
interpret *tiny-category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro categoryI*)
show *vfsequence ?Funct* **by** (*simp add*: *cat-Funct-def*)
show *vcard ?Funct* = $6_{\mathbb{N}}$
unfolding *cat-Funct-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** *semicategory* α (*cat-smc* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$))
unfolding *cat-smc-Funct* **by** (*rule semicategory-smc-Funct*)
show $\mathcal{D}_\circ$ (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*CId*|)) = *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-Funct-components cat-FUNCT-cs-simps*)
show *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*CId*|)(|$\mathfrak{F}$|) : $\mathfrak{F} \mapsto_{cat\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}} \mathfrak{F}$
if $\mathfrak{F} \in_\circ$ *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*Obj*|) **for** $\mathfrak{F}$
proof–
from *that* **have** $\mathfrak{F} \in_\circ$ *tm-cf-maps* α $\mathfrak{A}$ $\mathfrak{B}$
unfolding *cat-Funct-components* **by** *simp*
then obtain $\mathfrak{F}'$
where $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-map* $\mathfrak{F}'$ **and** $\mathfrak{F}'$: $\mathfrak{F}' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
by *auto*
from *assms* $\mathfrak{F}'$ **show** *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*CId*|)(|$\mathfrak{F}$|) : $\mathfrak{F} \mapsto_{cat\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}} \mathfrak{F}$
by
(
cs-concl
cs-simp: *cat-FUNCT-cs-simps* $\mathfrak{F}$-*def*
cs-intro: *cat-FUNCT-cs-intros cat-small-cs-intros*
)
qed
show *cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(|*CId*|)(|$\mathfrak{G}$|) $\circ_{A\,cat\text{-}Funct\ \alpha\ \mathfrak{A}\ \mathfrak{B}}$ $\mathfrak{N}$ = $\mathfrak{N}$

```
    if 𝔑 : 𝔉 ↦_cat-Funct α 𝔄 𝔅 𝔊 for 𝔉 𝔊 𝔑
  proof–
    note 𝔑 = cat-Funct-is-arrD[OF that]
    from assms 𝔑(1) show
      cat-Funct α 𝔄 𝔅(|CId|)(|𝔊|) ∘_A cat-Funct α 𝔄 𝔅 𝔑 = 𝔑
      by (subst (1 2) 𝔑(2), use nothing in ‹subst 𝔑(4)›)
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros
        )
  qed
  show 𝔑 ∘_A cat-Funct α 𝔄 𝔅 cat-Funct α 𝔄 𝔅(|CId|)(|𝔊|) = 𝔑
    if 𝔑 : 𝔊 ↦_cat-Funct α 𝔄 𝔅 ℌ for 𝔊 ℌ 𝔑
  proof–
    note 𝔑 = cat-Funct-is-arrD[OF that]
    from assms 𝔑(1) show
      𝔑 ∘_A cat-Funct α 𝔄 𝔅 cat-Funct α 𝔄 𝔅(|CId|)(|𝔊|) = 𝔑
      by (subst (1 2) 𝔑(2), use nothing in ‹subst 𝔑(3)›)
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros
        )
  qed
 qed auto
qed

lemma category-cat-Funct'[cat-FUNCT-cs-intros]:
 assumes tiny-category α 𝔄
   and category α 𝔅
   and β = α
 shows category α (cat-Funct β 𝔄 𝔅)
 using assms(1,2) unfolding assms(3) by (rule category-cat-Funct)
```

25.3.4 *Funct* is a subcategory of *FUNCT*

```
lemma subcategory-cat-Funct-cat-FUNCT:
 assumes 𝒵 β and α ∈∘ β and tiny-category α 𝔄 and category α 𝔅
 shows cat-Funct α 𝔄 𝔅 ⊆_CO β cat-FUNCT α 𝔄 𝔅
proof
 (
   intro subcategoryI,
   unfold cat-smc-FUNCT cat-smc-Funct cat-Funct-components(1)
 )
 interpret category α 𝔅 by (rule assms(4))
 interpret 𝔄𝔅: category α ‹cat-Funct α 𝔄 𝔅›
   by (rule category-cat-Funct[OF assms(3,4)])
 show category β (cat-Funct α 𝔄 𝔅)
   by (rule category.cat-category-if-ge-Limit[OF - assms(1,2)])
     (auto intro: cat-cs-intros)
 from assms show category β (cat-FUNCT α 𝔄 𝔅)
   by (cs-concl cs-intro: tiny-category-cat-FUNCT cat-small-cs-intros)
 show smc-Funct α 𝔄 𝔅 ⊆_SMC β smc-FUNCT α 𝔄 𝔅
   by (rule subsemicategory-smc-Funct-smc-FUNCT[OF assms])
 show cat-Funct α 𝔄 𝔅(|CId|)(|𝔉|) = cat-FUNCT α 𝔄 𝔅(|CId|)(|𝔉|)
   if ‹𝔉 ∈∘ tm-cf-maps α 𝔄 𝔅› for 𝔉
 proof–
```

```
    from that obtain 𝔉′ where 𝔉-def: 𝔉 = cf-map 𝔉′
      and 𝔉′: 𝔉′ : 𝔄 ↦↦_{C.tm α} 𝔅
      by auto
    from that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-FUNCT-cs-simps
            cs-intro: cat-FUNCT-cs-intros tm-cf-maps-in-cf-maps
        )
  qed
qed
```

25.3.5 Isomorphism

```
lemma (in is-tm-iso-ntcf) cat-Funct-is-iso-arrI:
  assumes category α 𝔅
  shows ntcf-arrow 𝔑 : cf-map 𝔉 ↦_{iso cat-Funct α 𝔄 𝔅} cf-map 𝔊
proof(intro is-iso-arrI is-inverseI)
  from is-tm-iso-ntcf-axioms show
    ntcf-arrow 𝔑 : cf-map 𝔉 ↦_{cat-Funct α 𝔄 𝔅} cf-map 𝔊
    by (cs-concl cs-shallow cs-intro: ntcf-cs-intros cat-FUNCT-cs-intros)
  interpret inv-𝔑: is-tm-iso-ntcf α 𝔄 𝔅 𝔊 𝔉 ‹inv-ntcf 𝔑›
    by (rule iso-tm-ntcf-is-iso-arr(1)[OF assms is-tm-iso-ntcf-axioms])
  from inv-𝔑.is-tm-iso-ntcf-axioms show
    ntcf-arrow (inv-ntcf 𝔑) : cf-map 𝔊 ↦_{cat-Funct α 𝔄 𝔅} cf-map 𝔉
    by (cs-concl cs-shallow cs-intro: ntcf-cs-intros cat-FUNCT-cs-intros)
  from is-tm-iso-ntcf-axioms show
    ntcf-arrow 𝔑 : cf-map 𝔉 ↦_{cat-Funct α 𝔄 𝔅} cf-map 𝔊
    by (cs-concl cs-shallow cs-intro: ntcf-cs-intros cat-FUNCT-cs-intros)
  from assms is-tm-iso-ntcf-axioms show
    ntcf-arrow (inv-ntcf 𝔑) ∘_{A cat-Funct α 𝔄 𝔅} ntcf-arrow 𝔑 =
      cat-Funct α 𝔄 𝔅(|CId|)(|cf-map 𝔉|)
    ntcf-arrow 𝔑 ∘_{A cat-Funct α 𝔄 𝔅} ntcf-arrow (inv-ntcf 𝔑) =
      cat-Funct α 𝔄 𝔅(|CId|)(|cf-map 𝔊|)
    by
      (
        cs-concl
          cs-simp: iso-tm-ntcf-is-iso-arr(2,3) cat-FUNCT-cs-simps
          cs-intro: ntcf-cs-intros cat-FUNCT-cs-intros cat-small-cs-intros
      )+
qed

lemma (in is-tm-iso-ntcf) cat-Funct-is-iso-arrI′:
  assumes category α 𝔅
    and 𝔑′ = ntcf-arrow 𝔑
    and 𝔉′ = cf-map 𝔉
    and 𝔊′ = cf-map 𝔊
  shows 𝔑′ : 𝔉′ ↦_{iso cat-Funct α 𝔄 𝔅} cf-map 𝔊
  using assms(1) unfolding assms(2–4) by (rule cat-Funct-is-iso-arrI)

lemmas [cat-FUNCT-cs-intros] =
  is-tm-iso-ntcf.cat-Funct-is-iso-arrI′[rotated 2]

lemma cat-Funct-is-iso-arrD:
  assumes tiny-category α 𝔄
    and category α 𝔅
    and 𝔑 : 𝔉 ↦_{iso cat-Funct α 𝔄 𝔅} 𝔊 (is ‹𝔑 : 𝔉 ↦_{iso ?Funct} 𝔊›)
```

```
  shows ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑 :
    cf-of-cf-map 𝔄 𝔅 𝔉 ↦_CF.tm.iso cf-of-cf-map 𝔄 𝔅 𝔊 : 𝔄 ↦↦_C.tmα 𝔅
    and 𝔑 = ntcf-arrow (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑)
    and 𝔉 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔉)
    and 𝔊 = cf-map (cf-of-cf-map 𝔄 𝔅 𝔊)
proof–
  interpret Funct: category α ?Funct
    by (rule category-cat-Funct[OF assms(1,2)])
  have inv-𝔑: 𝔑^-1_C ?Funct : 𝔊 ↦_iso ?Funct 𝔉
    and inv-𝔑-𝔑: 𝔑^-1_C ?Funct ∘_A ?Funct 𝔑 = ?Funct(|CId|)(|𝔉|)
    and 𝔑-inv-𝔑: 𝔑 ∘_A ?Funct 𝔑^-1_C ?Funct = ?Funct(|CId|)(|𝔊|)
    by
      (
        intro
          Funct.cat-the-inverse-is-iso-arr[OF assms(3)]
          Funct.cat-the-inverse-Comp-CId[OF assms(3)]
      )+
  from assms is-iso-arrD inv-𝔑
  have 𝔑-is-arr: 𝔑 : 𝔉 ↦_cat-Funct α 𝔄 𝔅 𝔊
    and inv-𝔑-is-arr: 𝔑^-1_C ?Funct : 𝔊 ↦_cat-Funct α 𝔄 𝔅 𝔉
    by auto
  note 𝔑-is-arr = cat-Funct-is-arrD[OF 𝔑-is-arr]
  note inv-𝔑-is-arr = cat-Funct-is-arrD[OF inv-𝔑-is-arr]
  let ?𝔑 = ‹ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑›
    and ?inv-𝔑 = ‹ntcf-of-ntcf-arrow 𝔄 𝔅 (𝔑^-1_C cat-Funct α 𝔄 𝔅)›
  from inv-𝔑-𝔑 𝔑-is-arr(1) inv-𝔑-is-arr(1) have inv-𝔑-𝔑:
    ?inv-𝔑 ∙_NTCF ?𝔑 = ntcf-id (cf-of-cf-map 𝔄 𝔅 𝔉)
    by
      (
        subst (asm) inv-𝔑-is-arr(2),
        use nothing in ‹subst (asm) (2) 𝔑-is-arr(2), subst (asm) 𝔑-is-arr(3)›
      )
      (
        cs-prems
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros
      )
  from 𝔑-inv-𝔑 inv-𝔑-is-arr(1) 𝔑-is-arr(1) have 𝔑-inv-𝔑:
    ?𝔑 ∙_NTCF ?inv-𝔑 = ntcf-id (cf-of-cf-map 𝔄 𝔅 𝔊)
    by
      (
        subst (asm) inv-𝔑-is-arr(2),
        use nothing in ‹subst (asm) 𝔑-is-arr(2), subst (asm) 𝔑-is-arr(4)›
      )
      (
        cs-prems
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros
      )
  show ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑 :
    cf-of-cf-map 𝔄 𝔅 𝔉 ↦_CF.tm.iso cf-of-cf-map 𝔄 𝔅 𝔊 : 𝔄 ↦↦_C.tmα 𝔅
    by
      (
        rule is-iso-arr-is-tm-iso-ntcf[
          OF 𝔑-is-arr(1) inv-𝔑-is-arr(1) 𝔑-inv-𝔑 inv-𝔑-𝔑
          ]
      )
  show 𝔑 = ntcf-arrow (ntcf-of-ntcf-arrow 𝔄 𝔅 𝔑)
```

and $\mathfrak{F}$ = *cf-map* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$)
and $\mathfrak{G}$ = *cf-map* (*cf-of-cf-map* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$)
by (*intro* $\mathfrak{N}$-*is-arr*(*2–4*))+
qed

25.4 Diagonal functor

25.4.1 Definition and elementary properties

See Chapter III-3 in [7].

definition *cf-diagonal* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle\Delta_{CF}\rangle$)
where Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}$ =
[
($\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!)$. *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ a)),
($\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!)$. *ntcf-arrow* (*ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f)),
$\mathfrak{C}$,
cat-FUNCT α $\mathfrak{J}$ $\mathfrak{C}$
]$_\circ$

Components.

lemma *cf-diagonal-components*:
shows Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|ObjMap|\!)$ = ($\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!)$. *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ a))
and Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|ArrMap|\!)$ = ($\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!)$. *ntcf-arrow* (*ntcf-const* $\mathfrak{J}$ $\mathfrak{C}$ f))
and Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|HomDom|\!)$ = $\mathfrak{C}$
and Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|HomCod|\!)$ = *cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}$
unfolding *cf-diagonal-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

25.4.2 Object map

mk-VLambda *cf-diagonal-components*(*1*)
|*vsv cf-diagonal-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-diagonal-ObjMap-vdomain*[*cat-cs-simps*]|
|*app cf-diagonal-ObjMap-app*[*cat-cs-simps*]|

lemma *cf-diagonal-ObjMap-vrange*:
assumes $\mathcal{Z}$ β**and** $\alpha \in_\circ \beta$ **and** *category* α $\mathfrak{J}$ **and** *category* α $\mathfrak{C}$
shows $\mathcal{R}_\circ$ (Δ_{CF} α $\mathfrak{J}$ $\mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_\circ$ *cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}(\!|Obj|\!)$
unfolding *cf-diagonal-components*
proof(*rule vrange-VLambda-vsubset*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
interpret *category* α $\mathfrak{J}$ **by** (*rule assms*(*3*))
interpret *FUNCT*: *tiny-category* β $\langle$(*cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}$)$\rangle$
by (*rule* $\mathcal{Z}$.*tiny-category-cat-FUNCT*[*OF* $\mathcal{Z}$-*axioms assms*(*1,2*)])
fix x **assume** *prems*: $x \in_\circ \mathfrak{C}(\!|Obj|\!)$
from *prems assms* **show** *cf-map* (*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ x) $\in_\circ$ *cat-FUNCT* α $\mathfrak{J}$ $\mathfrak{C}(\!|Obj|\!)$
unfolding *cat-FUNCT-components*(*1*)
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
qed

25.4.3 Arrow map

mk-VLambda *cf-diagonal-components*(*2*)
|*vsv cf-diagonal-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain cf-diagonal-ArrMap-vdomain*[*cat-cs-simps*]|
|*app cf-diagonal-ArrMap-app*[*cat-cs-simps*]|

25.4.4 Diagonal functor is a functor

lemma *cf-diagonal-is-functor*[*cat-cs-intros*]:

```
  assumes 𝒵 β and α ∈∘ β and category α 𝔍 and category α ℭ
  shows Δ_CF α 𝔍 ℭ : ℭ ↦↦_Cβ cat-FUNCT α 𝔍 ℭ (is ‹?Δ : ℭ ↦↦_Cβ ?FUNCT›)
proof–

  interpret β: 𝒵 β by (rule assms(1))
  interpret 𝔍: category α 𝔍 by (rule assms(3))
  interpret ℭ: category α ℭ by (rule assms(4))
  interpret FUNCT: tiny-category β ‹(cat-FUNCT α 𝔍 ℭ)›
    by (rule 𝒵.tiny-category-cat-FUNCT[OF 𝔍.𝒵-axioms assms(1,2)])

  show ?thesis
  proof(intro is-functorI')
    show vfsequence ?Δ
      unfolding cf-diagonal-def by (simp add: nat-omega-simps)
    show category β ℭ by (rule ℭ.cat-category-if-ge-Limit[OF assms(1,2)])
    from assms show category β (cat-FUNCT α 𝔍 ℭ)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show vcard ?Δ = ℶ₀
      unfolding cf-diagonal-def by (simp add: nat-omega-simps)
    show vsv (?Δ(|ObjMap|)) unfolding cf-diagonal-components by simp
    from assms show ℛ∘ (?Δ(|ObjMap|)) ⊆∘ ?FUNCT(|Obj|)
      by (rule cf-diagonal-ObjMap-vrange)
    show ?Δ(|ArrMap|)(|f|) : ?Δ(|ObjMap|)(|a|) ↦_?FUNCT ?Δ(|ObjMap|)(|b|)
      if f : a ↦_ℭ b for f a b
      using that
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-small-cs-intros
        )
    show ?Δ(|ArrMap|)(|g ∘_Aℭ f|) = ?Δ(|ArrMap|)(|g|) ∘_A ?FUNCT ?Δ(|ArrMap|)(|f|)
      if g : b ↦_ℭ c and f : a ↦_ℭ b for g b c f a
      using that 𝔍.category-axioms ℭ.category-axioms
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    fix c assume c ∈∘ ℭ(|Obj|)
    with 𝔍.category-axioms ℭ.category-axioms show
      ?Δ(|ArrMap|)(|ℭ(|CId|)(|c|)|) = ?FUNCT(|CId|)(|?Δ(|ObjMap|)(|c|)|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (auto simp: cf-diagonal-components cat-smc-FUNCT)

qed

lemma cf-diagonal-is-functor'[cat-cs-intros]:
  assumes 𝒵 β
    and α ∈∘ β
    and category α 𝔍
    and category α ℭ
```

and $\mathfrak{A}' = \mathfrak{C}$
and $\mathfrak{B}' = \textit{cat-FUNCT}\ \alpha\ \mathfrak{J}\ \mathfrak{C}$
shows $\Delta_{CF}\ \alpha\ \mathfrak{J}\ \mathfrak{C} : \mathfrak{A}' \mapsto\mapsto_{C\beta} \mathfrak{B}'$
using *assms*(*1–4*) **unfolding** *assms*(*5–6*) **by** (*rule cf-diagonal-is-functor*)

25.5 Diagonal functor for functors with tiny maps

25.5.1 Definition and elementary properties

See Chapter III-3 in [7].

definition *tm-cf-diagonal* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle\Delta_{CF.tm}\rangle$)
where $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C} =$
[
($\lambda a \in_\circ \mathfrak{C}(\!|\textit{Obj}|\!).\ \textit{cf-map}\ (\textit{cf-const}\ \mathfrak{J}\ \mathfrak{C}\ a)$),
($\lambda f \in_\circ \mathfrak{C}(\!|\textit{Arr}|\!).\ \textit{ntcf-arrow}\ (\textit{ntcf-const}\ \mathfrak{J}\ \mathfrak{C}\ f)$),
$\mathfrak{C}$,
cat-Funct $\alpha\ \mathfrak{J}\ \mathfrak{C}$
]$_\circ$

Components.

lemma *tm-cf-diagonal-components*:
shows $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{ObjMap}|\!) = (\lambda a \in_\circ \mathfrak{C}(\!|\textit{Obj}|\!).\ \textit{cf-map}\ (\textit{cf-const}\ \mathfrak{J}\ \mathfrak{C}\ a))$
and $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{ArrMap}|\!) = (\lambda f \in_\circ \mathfrak{C}(\!|\textit{Arr}|\!).\ \textit{ntcf-arrow}\ (\textit{ntcf-const}\ \mathfrak{J}\ \mathfrak{C}\ f))$
and $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{HomDom}|\!) = \mathfrak{C}$
and $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{HomCod}|\!) = \textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}$
unfolding *tm-cf-diagonal-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

25.5.2 Object map

mk-VLambda *tm-cf-diagonal-components*(*1*)
|*vsv tm-cf-diagonal-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain tm-cf-diagonal-ObjMap-vdomain*[*cat-cs-simps*]|
|*app tm-cf-diagonal-ObjMap-app*[*cat-cs-simps*]|

lemma *tm-cf-diagonal-ObjMap-vrange*:
assumes *tiny-category* $\alpha\ \mathfrak{J}$ **and** *category* $\alpha\ \mathfrak{C}$
shows $\mathcal{R}_\circ\ (\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{ObjMap}|\!)) \subseteq_\circ \textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{Obj}|\!)$
unfolding *tm-cf-diagonal-components*
proof(*rule vrange-VLambda-vsubset*)
fix x **assume** $x \in_\circ \mathfrak{C}(\!|\textit{Obj}|\!)$
with *assms category-cat-Funct*[*OF assms*] **show**
$\textit{cf-map}\ (\textit{cf-const}\ \mathfrak{J}\ \mathfrak{C}\ x) \in_\circ \textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}(\!|\textit{Obj}|\!)$
unfolding *cat-Funct-components*(*1*)
by (*cs-concl* **cs-intro**: *cat-small-cs-intros cat-FUNCT-cs-intros*)
qed

25.5.3 Arrow map

mk-VLambda *tm-cf-diagonal-components*(*2*)
|*vsv tm-cf-diagonal-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain tm-cf-diagonal-ArrMap-vdomain*[*cat-cs-simps*]|
|*app tm-cf-diagonal-ArrMap-app*[*cat-cs-simps*]|

25.5.4 Diagonal functor for functors with tiny maps is a functor

lemma *tm-cf-diagonal-is-functor*[*cat-cs-intros*]:
assumes *tiny-category* $\alpha\ \mathfrak{J}$ **and** *category* $\alpha\ \mathfrak{C}$
shows $\Delta_{CF.tm}\ \alpha\ \mathfrak{J}\ \mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \textit{cat-Funct}\ \alpha\ \mathfrak{J}\ \mathfrak{C}$

```
    (is ‹?Δ : 𝔈 ↦↦_Cα ?Funct›)
proof–

  interpret 𝔍: tiny-category α 𝔍 by (rule assms(1))
  interpret 𝔈: category α 𝔈 by (rule assms(2))

  show ?thesis
  proof(intro is-functorI')
    show vfsequence ?Δ
      unfolding tm-cf-diagonal-def by (simp add: nat-omega-simps)
    from assms(2) show category α 𝔈
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from assms show category α ?Funct
      by (cs-concl cs-shallow cs-intro: cat-cs-intros category-cat-Funct)
    show vcard ?Δ = ℵ₄
      unfolding tm-cf-diagonal-def by (simp add: nat-omega-simps)
    show vsv (?Δ(|ObjMap|)) unfolding tm-cf-diagonal-components by simp
    from assms show ℛ∘ (?Δ(|ObjMap|)) ⊆∘ ?Funct(|Obj|)
      by (rule tm-cf-diagonal-ObjMap-vrange)
    show ?Δ(|ArrMap|)(|f|) : ?Δ(|ObjMap|)(|a|) ↦_?Funct ?Δ(|ObjMap|)(|b|)
      if f : a ↦_𝔈 b for f a b
      using that
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-small-cs-intros
        )
    show ?Δ(|ArrMap|)(|g ∘_A𝔈 f|) = ?Δ(|ArrMap|)(|g|) ∘_A ?Funct ?Δ(|ArrMap|)(|f|)
      if g : b ↦_𝔈 c and f : a ↦_𝔈 b for g b c f a
      using that 𝔍.category-axioms 𝔈.category-axioms
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    fix c assume c ∈∘ 𝔈(|Obj|)
    with 𝔍.category-axioms 𝔈.category-axioms show
      ?Δ(|ArrMap|)(|𝔈(|CId|)(|c|)|) = ?Funct(|CId|)(|?Δ(|ObjMap|)(|c|)|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (auto simp: tm-cf-diagonal-components cat-smc-FUNCT)

qed

lemma tm-cf-diagonal-is-functor'[cat-cs-intros]:
  assumes tiny-category α 𝔍
    and category α 𝔈
    and α' = α
    and 𝔄 = 𝔈
    and 𝔅 = cat-Funct α 𝔍 𝔈
  shows Δ_CF.tm α 𝔍 𝔈 : 𝔄 ↦↦_Cα' 𝔅
  using assms(1–2) unfolding assms(3–5) by (rule tm-cf-diagonal-is-functor)
```

25.6 Functor raised to the power of a category

25.6.1 Definition and elementary properties

Most of the definitions and the results presented in this and the remaining subsections can be found in [7] and [12] (e.g., see Chapter X-3 in [7]).

definition *exp-cf-cat* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$ =
[
(
$\lambda \mathfrak{S} \in_{\circ}$ *cat-FUNCT* α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|))(|*Obj*|).
cf-map ($\mathfrak{K}$ $\circ_{CF}$ *cf-of-cf-map* $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|)) $\mathfrak{S}$)
),
(
$\lambda \sigma \in_{\circ}$ *cat-FUNCT* α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|))(|*Arr*|).
ntcf-arrow ($\mathfrak{K}$ $\circ_{CF-NTCF}$ *ntcf-of-ntcf-arrow* $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|)) σ)
),
cat-FUNCT α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|)),
cat-FUNCT α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomCod*|))
]$_{\circ}$

Components.

lemma *exp-cf-cat-components*:
shows *exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$(|*ObjMap*|) =
(
$\lambda \mathfrak{S} \in_{\circ}$ *cat-FUNCT* α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|))(|*Obj*|).
cf-map ($\mathfrak{K}$ $\circ_{CF}$ *cf-of-cf-map* $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|)) $\mathfrak{S}$)
)
and
exp-cf-cat α $\mathfrak{K}$ $\mathfrak{A}$(|*ArrMap*|) =
(
$\lambda \sigma \in_{\circ}$ *cat-FUNCT* α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|))(|*Arr*|).
ntcf-arrow ($\mathfrak{K}$ $\circ_{CF-NTCF}$ (*ntcf-of-ntcf-arrow* $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|)) σ))
)
and *exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$(|*HomDom*|) = *cat-FUNCT* α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomDom*|))
and *exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$(|*HomCod*|) = *cat-FUNCT* α $\mathfrak{A}$ ($\mathfrak{K}$(|*HomCod*|))
unfolding *exp-cf-cat-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

25.6.2 Object map

mk-VLambda *exp-cf-cat-components*(*1*)
|*vsv exp-cf-cat-components-ObjMap-vsv*[*cat-FUNCT-cs-intros*]|

context
fixes α $\mathfrak{K}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)

mk-VLambda *exp-cf-cat-components*(*1*)[**where** $\mathfrak{K}$=$\mathfrak{K}$ **and** α=α, *unfolded cat-cs-simps*]
|*vdomain exp-cf-cat-components-ObjMap-vdomain*[*cat-FUNCT-cs-simps*]|
|*app exp-cf-cat-components-ObjMap-app*[*cat-FUNCT-cs-simps*]|

end

25.6.3 Arrow map

mk-VLambda *exp-cf-cat-components*(*2*)
|*vsv exp-cf-cat-components-ArrMap-vsv*[*cat-FUNCT-cs-intros*]|

context
fixes α $\mathfrak{K}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)

mk-VLambda *exp-cf-cat-components*(*2*)[**where** $\mathfrak{K}$=$\mathfrak{K}$ **and** α=α, *unfolded cat-cs-simps*]
|*vdomain exp-cf-cat-components-ArrMap-vdomain*[*cat-FUNCT-cs-simps*]|
|*app exp-cf-cat-components-ArrMap-app*[*cat-FUNCT-cs-simps*]|

end

25.6.4 Domain and codomain

context
fixes α $\mathfrak{K}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{K}$: $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule* $\mathfrak{K}$)

lemmas *exp-cf-cat-HomDom*[*cat-FUNCT-cs-simps*] =
exp-cf-cat-components(*3*)[**where** $\mathfrak{K}$=$\mathfrak{K}$ **and** α=α, *unfolded cat-cs-simps*]
and *exp-cf-cat-HomCod*[*cat-FUNCT-cs-simps*] =
exp-cf-cat-components(*4*)[**where** $\mathfrak{K}$=$\mathfrak{K}$ **and** α=α, *unfolded cat-cs-simps*]

end

25.6.5 Functor raised to the power of a category is a functor

lemma *exp-cf-cat-is-tiny-functor*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$ **and** *category* α $\mathfrak{A}$ **and** $\mathfrak{K} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$: *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$ $\mapsto\mapsto_{C.tiny\beta}$ *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{C}$
proof–
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{K}$ **by** (*rule assms*(*4*))
from *assms*(*2–4*) **interpret** $\mathfrak{AB}$: *tiny-category* β ‹*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$›
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *assms*(*2–4*) **interpret** $\mathfrak{AC}$: *tiny-category* β ‹*cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{C}$›
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
show *?thesis*
proof(*intro is-tiny-functorI′ is-functorI′*)
show *vfsequence* (*exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$) **unfolding** *exp-cf-cat-def* **by** *simp*
show *vcard* (*exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$) = $\mathfrak{4}_{\mathbb{N}}$
unfolding *exp-cf-cat-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*exp-cf-cat* α $\mathfrak{K}$ $\mathfrak{A}$(|*ObjMap*|)) $\subseteq_\circ$ *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{C}$(|*Obj*|)
proof
(
unfold cat-FUNCT-components exp-cf-cat-components,
intro vrange-VLambda-vsubset,
unfold cat-cs-simps
)

```
    fix 𝔉 assume 𝔉 ∈∘ cf-maps α 𝔄 𝔅
    then obtain 𝔉′ where 𝔉-def: 𝔉 = cf-map 𝔉′ and 𝔉′: 𝔉′ : 𝔄 ↦↦_Cα 𝔅
      by auto
    from assms(2–4) 𝔉′ show
      cf-map (ℜ ∘_CF cf-of-cf-map 𝔄 𝔅 𝔉) ∈∘ cf-maps α 𝔄 ℭ
      by (cs-concl cs-simp: 𝔉-def cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
  qed
  show exp-cf-cat α ℜ 𝔄⦇ArrMap⦈⦇𝔑⦈ :
    exp-cf-cat α ℜ 𝔄⦇ObjMap⦈⦇𝔉⦈ ↦_cat-FUNCT α 𝔄 ℭ
    exp-cf-cat α ℜ 𝔄⦇ObjMap⦈⦇𝔊⦈
    if 𝔑 : 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 𝔊 for 𝔉 𝔊 𝔑
  proof–
    note 𝔑 = cat-FUNCT-is-arrD[OF that]
    from 𝔑(1,3,4) assms(2–4) show ?thesis
    by (subst 𝔑(2), use nothing in ‹subst 𝔑(3), subst 𝔑(4)›)
      (
        cs-concl
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  qed
  show
    exp-cf-cat α ℜ 𝔄⦇ArrMap⦈⦇𝔐 ∘_A cat-FUNCT α 𝔄 𝔅 𝔑⦈ =
      exp-cf-cat α ℜ 𝔄⦇ArrMap⦈⦇𝔐⦈ ∘_A cat-FUNCT α 𝔄 ℭ
      exp-cf-cat α ℜ 𝔄⦇ArrMap⦈⦇𝔑⦈
    if 𝔐 : 𝔊 ↦_cat-FUNCT α 𝔄 𝔅 ℌ and 𝔑 : 𝔉 ↦_cat-FUNCT α 𝔄 𝔅 𝔊
    for 𝔊 ℌ 𝔐 𝔉 𝔑
  proof–
    note 𝔐 = cat-FUNCT-is-arrD[OF that(1)]
      and 𝔑 = cat-FUNCT-is-arrD[OF that(2)]
    from 𝔐(1,3,4) 𝔑(1,3,4) assms(2–4) show ?thesis
      by (subst (1 2) 𝔐(2), use nothing in ‹subst (1 2) 𝔑(2)›)
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps cf-ntcf-comp-ntcf-vcomp
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
  show
    exp-cf-cat α ℜ 𝔄⦇ArrMap⦈⦇cat-FUNCT α 𝔄 𝔅⦇CId⦈⦇𝔉⦈⦈ =
      cat-FUNCT α 𝔄 ℭ⦇CId⦈⦇exp-cf-cat α ℜ 𝔄⦇ObjMap⦈⦇𝔉⦈⦈
    if 𝔉 ∈∘ cat-FUNCT α 𝔄 𝔅⦇Obj⦈ for 𝔉
  proof–
    from that[unfolded cat-FUNCT-components] obtain 𝔊
      where 𝔉-def: 𝔉 = cf-map 𝔊 and 𝔊: 𝔊 : 𝔄 ↦↦_Cα 𝔅
      by auto
    from 𝔊 show
      exp-cf-cat α ℜ 𝔄⦇ArrMap⦈⦇cat-FUNCT α 𝔄 𝔅⦇CId⦈⦇𝔉⦈⦈ =
        cat-FUNCT α 𝔄 ℭ⦇CId⦈⦇exp-cf-cat α ℜ 𝔄⦇ObjMap⦈⦇𝔉⦈⦈
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps 𝔉-def
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
qed
  (
```

```
    use assms(1,2) in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      ›
  )+
qed

lemma exp-cf-cat-is-tiny-functor'[cat-FUNCT-cs-intros]:
  assumes 𝒵 β
    and α ∈∘ β
    and category α 𝔄
    and 𝔎 : 𝔅 ↦↦_{Cα} ℭ
    and 𝔄' = cat-FUNCT α 𝔄 𝔅
    and 𝔅' = cat-FUNCT α 𝔄 ℭ
  shows exp-cf-cat α 𝔎 𝔄 : 𝔄' ↦↦_{C.tiny β} 𝔅'
  using assms(1–4) unfolding assms(5,6) by (rule exp-cf-cat-is-tiny-functor)
```

25.6.6 Further properties

```
lemma exp-cf-cat-cf-comp:
  assumes category α 𝔇 and 𝔊 : 𝔅 ↦↦_{Cα} ℭ and 𝔉 : 𝔄 ↦↦_{Cα} 𝔅
  shows exp-cf-cat α (𝔊 ∘_{CF} 𝔉) 𝔇 = exp-cf-cat α 𝔊 𝔇 ∘_{CF} exp-cf-cat α 𝔉 𝔇
proof(rule cf-eqI)

  interpret 𝔇: category α 𝔇 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  interpret 𝔉: is-functor α 𝔄 𝔅 𝔉 by (rule assms(3))

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔇.𝒵-Limit-αω 𝔇.𝒵-ω-αω 𝒵-def 𝔇.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  from αβ show
    exp-cf-cat α (𝔊 ∘_{CF} 𝔉) 𝔇 : cat-FUNCT α 𝔇 𝔄 ↦↦_{Cβ} cat-FUNCT α 𝔇 ℭ
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from αβ show
    exp-cf-cat α 𝔊 𝔇 ∘_{CF} exp-cf-cat α 𝔉 𝔇 :
      cat-FUNCT α 𝔇 𝔄 ↦↦_{Cβ} cat-FUNCT α 𝔇 ℭ
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from αβ have dom-lhs:
    𝒟∘ (exp-cf-cat α (𝔊 ∘_{CF} 𝔉) 𝔇(|ObjMap|)) = cat-FUNCT α 𝔇 𝔄(|Obj|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
```

```
      cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )
from αβ have dom-rhs:
  𝒟∘ ((exp-cf-cat α 𝔊 𝔇 ∘CF exp-cf-cat α 𝔉 𝔇)(⟦ObjMap⟧)) =
    cat-FUNCT α 𝔇 𝔄(⟦Obj⟧)
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )
show
  exp-cf-cat α (𝔊 ∘CF 𝔉) 𝔇(⟦ObjMap⟧) =
    (exp-cf-cat α 𝔊 𝔇 ∘CF exp-cf-cat α 𝔉 𝔇)(⟦ObjMap⟧)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  show vsv (exp-cf-cat α (𝔊 ∘CF 𝔉) 𝔇(⟦ObjMap⟧))
    by (cs-concl cs-shallow cs-intro: cat-FUNCT-cs-intros)
  from αβ show vsv ((exp-cf-cat α 𝔊 𝔇 ∘CF exp-cf-cat α 𝔉 𝔇)(⟦ObjMap⟧))
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  fix ℌ assume ℌ ∈∘ cat-FUNCT α 𝔇 𝔄(⟦Obj⟧)
  then have ℌ ∈∘ cf-maps α 𝔇 𝔄 unfolding cat-FUNCT-components by simp
  then obtain ℌ′ where ℌ-def: ℌ = cf-map ℌ′ and ℌ′: ℌ′ : 𝔇 ↦↦→Cα 𝔄
    by auto
  from assms αβ ℌ′ show
    exp-cf-cat α (𝔊 ∘CF 𝔉) 𝔇(⟦ObjMap⟧)(⟦ℌ⟧) =
      (exp-cf-cat α 𝔊 𝔇 ∘CF exp-cf-cat α 𝔉 𝔇)(⟦ObjMap⟧)(⟦ℌ⟧)
    by (subst (1 2) ℌ-def)
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
qed simp
from αβ have dom-lhs:
  𝒟∘ (exp-cf-cat α (𝔊 ∘CF 𝔉) 𝔇(⟦ArrMap⟧)) = cat-FUNCT α 𝔇 𝔄(⟦Arr⟧)
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )
from αβ have dom-rhs:
  𝒟∘ ((exp-cf-cat α 𝔊 𝔇 ∘CF exp-cf-cat α 𝔉 𝔇)(⟦ArrMap⟧)) =
    cat-FUNCT α 𝔇 𝔄(⟦Arr⟧)
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )
show
  exp-cf-cat α (𝔊 ∘CF 𝔉) 𝔇(⟦ArrMap⟧) =
    (exp-cf-cat α 𝔊 𝔇 ∘CF exp-cf-cat α 𝔉 𝔇)(⟦ArrMap⟧)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
```

```
    show vsv (exp-cf-cat α (𝔊 ∘_CF 𝔉) 𝔇(|ArrMap|))
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-FUNCT-cs-intros)
    from αβ show vsv ((exp-cf-cat α 𝔊 𝔇 ∘_CF exp-cf-cat α 𝔉 𝔇)(|ArrMap|))
      by
        (
          cs-concl cs-intro:
            cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    fix 𝔑 assume 𝔑 ∈₀ cat-FUNCT α 𝔇 𝔄(|Arr|)
    then obtain ℌ ℌ' where 𝔑: 𝔑 : ℌ ↦_cat-FUNCT α 𝔇 𝔄 ℌ'
      by (auto intro: is-arrI)
    note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
    from αβ assms 𝔑(1,3,4) show
      exp-cf-cat α (𝔊 ∘_CF 𝔉) 𝔇(|ArrMap|)(|𝔑|) =
        (exp-cf-cat α 𝔊 𝔇 ∘_CF exp-cf-cat α 𝔉 𝔇)(|ArrMap|)(|𝔑|)
      by (subst (1 2) 𝔑(2))
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps cf-comp-cf-ntcf-comp-assoc
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed simp
qed simp-all

lemma exp-cf-cat-cf-id-cat:
  assumes category α ℭ and category α 𝔇
  shows exp-cf-cat α (cf-id ℭ) 𝔇 = cf-id (cat-FUNCT α 𝔇 ℭ)
proof(rule cf-eqI)

  interpret ℭ: category α ℭ by (rule assms)
  interpret 𝔇: category α 𝔇 by (rule assms)

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def ℭ.𝒵-Limit-αω ℭ.𝒵-ω-αω 𝒵-def ℭ.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  from αβ show
    cf-id (cat-FUNCT α 𝔇 ℭ) : cat-FUNCT α 𝔇 ℭ ↦↦_Cβ cat-FUNCT α 𝔇 ℭ
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )
  from αβ show
    exp-cf-cat α (cf-id ℭ) 𝔇 : cat-FUNCT α 𝔇 ℭ ↦↦_Cβ cat-FUNCT α 𝔇 ℭ
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )
  from αβ have ObjMap-dom-lhs:
    𝒟₀ (exp-cf-cat α (cf-id ℭ) 𝔇(|ObjMap|)) = cat-FUNCT α 𝔇 ℭ(|Obj|)
    by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)
  from αβ have ObjMap-dom-rhs:
    𝒟₀ (cf-id (cat-FUNCT α 𝔇 ℭ)(|ObjMap|)) = cat-FUNCT α 𝔇 ℭ(|Obj|)
```

```
    by (cs-concl cs-simp: cat-cs-simps)
  show exp-cf-cat α (cf-id 𝔇) 𝔈(⁅ObjMap⁆) = cf-id (cat-FUNCT α 𝔇 𝔈)(⁅ObjMap⁆)
  proof
    (
      rule vsv-eqI,
      unfold ObjMap-dom-lhs ObjMap-dom-rhs cat-FUNCT-components(1)
    )
    fix ℌ assume prems: ℌ ∈₀ cf-maps α 𝔇 𝔈
    then obtain ℌ' where ℌ-def: ℌ = cf-map ℌ' and ℌ': ℌ' : 𝔇 ↦↦_Cα 𝔈
      by clarsimp
    from prems ℌ' show
      exp-cf-cat α (cf-id 𝔈) 𝔇(⁅ObjMap⁆)(⁅ℌ⁆) = cf-id (cat-FUNCT α 𝔇 𝔈)(⁅ObjMap⁆)(⁅ℌ⁆)
      by (subst (1 2) ℌ-def)
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (cs-concl cs-shallow cs-intro: cat-cs-intros cat-FUNCT-cs-intros)+
  from αβ have ArrMap-dom-lhs:
    𝒟₀ (cf-id (cat-FUNCT α 𝔇 𝔈)(⁅ArrMap⁆)) = cat-FUNCT α 𝔇 𝔈(⁅Arr⁆)
    by (cs-concl cs-simp: cat-cs-simps)
  from αβ have ArrMap-dom-rhs:
    𝒟₀ (exp-cf-cat α (cf-id 𝔈) 𝔇(⁅ArrMap⁆)) = cat-FUNCT α 𝔇 𝔈(⁅Arr⁆)
    by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)
  show exp-cf-cat α (cf-id 𝔈) 𝔇(⁅ArrMap⁆) = cf-id (cat-FUNCT α 𝔇 𝔈)(⁅ArrMap⁆)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix 𝔑 assume 𝔑 ∈₀ cat-FUNCT α 𝔇 𝔈(⁅Arr⁆)
    then obtain 𝔉 𝔊 where 𝔑: 𝔑 : 𝔉 ↦_cat-FUNCT α 𝔇 𝔈 𝔊
      by (auto intro: is-arrI)
    note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
    from 𝔑(1,3,4) αβ show
      exp-cf-cat α (cf-id 𝔈) 𝔇(⁅ArrMap⁆)(⁅𝔑⁆) =
        cf-id (cat-FUNCT α 𝔇 𝔈)(⁅ArrMap⁆)(⁅𝔑⁆)
      by (subst (1 2) 𝔑(2))
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (cs-concl cs-shallow cs-intro: cat-cs-intros cat-FUNCT-cs-intros)

qed simp-all


lemma cf-comp-exp-cf-cat-exp-cf-cat-cf-id[cat-FUNCT-cs-simps]:
  assumes category α 𝔄 and 𝔉 : 𝔅 ↦↦_Cα 𝔈
  shows exp-cf-cat α 𝔉 𝔄 ∘_CF exp-cf-cat α (cf-id 𝔅) 𝔄 = exp-cf-cat α 𝔉 𝔄
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔉: is-functor α 𝔅 𝔈 𝔉 by (rule assms(2))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp
```

```
show ?thesis
proof(rule cf-eqI)
  from assms αβ β show 𝔉𝔄:
    exp-cf-cat α 𝔉 𝔄 : cat-FUNCT α 𝔄 𝔅 ↦↦_{Cβ} cat-FUNCT α 𝔄 ℭ
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-FUNCT-cs-intros)
  with assms αβ show
    exp-cf-cat α 𝔉 𝔄 ∘_{CF} exp-cf-cat α (cf-id 𝔅) 𝔄 :
      cat-FUNCT α 𝔄 𝔅 ↦↦_{Cβ} cat-FUNCT α 𝔄 ℭ
    by
      (
        cs-concl cs-intro:
          cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )
  from assms αβ have ObjMap-dom-lhs:
    𝒟∘ ((exp-cf-cat α 𝔉 𝔄 ∘_{CF} exp-cf-cat α (cf-id 𝔅) 𝔄)(⟦ObjMap⟧)) =
      cat-FUNCT α 𝔄 𝔅(⟦Obj⟧)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )
  from assms have ObjMap-dom-rhs:
    𝒟∘ (exp-cf-cat α 𝔉 𝔄(⟦ObjMap⟧)) = cat-FUNCT α 𝔄 𝔅(⟦Obj⟧)
    by (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)
  from assms αβ have ArrMap-dom-lhs:
    𝒟∘ ((exp-cf-cat α 𝔉 𝔄 ∘_{CF} exp-cf-cat α (cf-id 𝔅) 𝔄)(⟦ArrMap⟧)) =
      cat-FUNCT α 𝔄 𝔅(⟦Arr⟧)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )
  from assms have ArrMap-dom-rhs:
    𝒟∘ (exp-cf-cat α 𝔉 𝔄(⟦ArrMap⟧)) = cat-FUNCT α 𝔄 𝔅(⟦Arr⟧)
    by (cs-concl cs-shallow cs-simp: cat-FUNCT-cs-simps)
  show
    (exp-cf-cat α 𝔉 𝔄 ∘_{CF} exp-cf-cat α (cf-id 𝔅) 𝔄)(⟦ObjMap⟧) =
      exp-cf-cat α 𝔉 𝔄(⟦ObjMap⟧)
  proof
    (
      rule vsv-eqI,
      unfold ObjMap-dom-lhs ObjMap-dom-rhs cat-FUNCT-components(1)
    )
    fix ℌ assume prems: ℌ ∈∘ cf-maps α 𝔄 𝔅
    then obtain ℌ' where ℌ-def: ℌ = cf-map ℌ' and ℌ': ℌ' : 𝔄 ↦↦_{Cα} 𝔅
      by clarsimp
    from prems ℌ' assms 𝔉𝔄 αβ show
      (exp-cf-cat α 𝔉 𝔄 ∘_{CF} exp-cf-cat α (cf-id 𝔅) 𝔄)(⟦ObjMap⟧)(⟦ℌ⟧) =
        exp-cf-cat α 𝔉 𝔄(⟦ObjMap⟧)(⟦ℌ⟧)
      unfolding ℌ-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros
        )
```

```
    qed
      (
        use assms 𝔉𝔄 αβ in
          ‹
            cs-concl
              cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros
          ›
      )
    show
      (exp-cf-cat α 𝔉 𝔄 ∘_CF exp-cf-cat α (cf-id 𝔅) 𝔄)(⌊ArrMap⌋) =
        exp-cf-cat α 𝔉 𝔄(⌊ArrMap⌋)
    proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
      fix 𝔐 assume 𝔐 ∈₀ cat-FUNCT α 𝔄 𝔅(⌊Arr⌋)
      then obtain 𝔉' 𝔊' where 𝔐: 𝔐 : 𝔉' ↦_cat-FUNCT α 𝔄 𝔅 𝔊'
        by (auto intro: is-arrI)
      note 𝔐 = cat-FUNCT-is-arrD[OF 𝔐]
      from 𝔐(1) assms 𝔉𝔄 αβ show
        (exp-cf-cat α 𝔉 𝔄 ∘_CF exp-cf-cat α (cf-id 𝔅) 𝔄)(⌊ArrMap⌋)(⌊𝔐⌋) =
          exp-cf-cat α 𝔉 𝔄(⌊ArrMap⌋)(⌊𝔐⌋)
        by (subst (1 2) 𝔐(2))
          (
            cs-concl
              cs-simp: cat-cs-simps cat-FUNCT-cs-simps
              cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros
          )
    qed
      (
        use assms αβ in
          ‹
            cs-concl cs-intro:
              cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros
          ›
      )
  qed simp-all

qed

lemma cf-comp-exp-cf-cat-cf-id-exp-cf-cat[cat-FUNCT-cs-simps]:
  assumes category α 𝔄 and 𝔉 : 𝔅 ↦↦_Cα ℭ
  shows exp-cf-cat α (cf-id ℭ) 𝔄 ∘_CF exp-cf-cat α 𝔉 𝔄 = exp-cf-cat α 𝔉 𝔄
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔉: is-functor α 𝔅 ℭ 𝔉 by (rule assms(2))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule cf-eqI)
    from assms αβ β show 𝔉𝔄:
      exp-cf-cat α 𝔉 𝔄 : cat-FUNCT α 𝔄 𝔅 ↦↦_Cβ cat-FUNCT α 𝔄 ℭ
      by (cs-concl cs-simp: cs-intro: cat-small-cs-intros cat-FUNCT-cs-intros)
    with assms αβ show
      exp-cf-cat α (cf-id ℭ) 𝔄 ∘_CF exp-cf-cat α 𝔉 𝔄 :
        cat-FUNCT α 𝔄 𝔅 ↦↦_Cβ cat-FUNCT α 𝔄 ℭ
```

```
  by
    (
      cs-concl cs-intro:
        cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
from assms αβ have ObjMap-dom-lhs:
  𝒟∘ ((exp-cf-cat α (cf-id ℭ) 𝔄 ∘CF exp-cf-cat α 𝔉 𝔄)(⌊ObjMap⌋)) =
    cat-FUNCT α 𝔄 𝔅(⌊Obj⌋)
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
from assms have ObjMap-dom-rhs:
  𝒟∘ (exp-cf-cat α 𝔉 𝔄(⌊ObjMap⌋)) = cat-FUNCT α 𝔄 𝔅(⌊Obj⌋)
  by (cs-concl cs-simp: cat-FUNCT-cs-simps)
from assms αβ have ArrMap-dom-lhs:
  𝒟∘ ((exp-cf-cat α (cf-id ℭ) 𝔄 ∘CF exp-cf-cat α 𝔉 𝔄)(⌊ArrMap⌋)) =
    cat-FUNCT α 𝔄 𝔅(⌊Arr⌋)
  by
    (
      cs-concl
        cs-simp: cat-cs-simps
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
from assms have ArrMap-dom-rhs:
  𝒟∘ (exp-cf-cat α 𝔉 𝔄(⌊ArrMap⌋)) = cat-FUNCT α 𝔄 𝔅(⌊Arr⌋)
  by (cs-concl cs-simp: cat-FUNCT-cs-simps)
show
  (exp-cf-cat α (cf-id ℭ) 𝔄 ∘CF exp-cf-cat α 𝔉 𝔄)(⌊ObjMap⌋) =
    exp-cf-cat α 𝔉 𝔄(⌊ObjMap⌋)
proof
  (
    rule vsv-eqI,
    unfold ObjMap-dom-lhs ObjMap-dom-rhs cat-FUNCT-components(1)
  )
  fix ℌ assume prems: ℌ ∈∘ cf-maps α 𝔄 𝔅
  then obtain ℌ' where ℌ-def: ℌ = cf-map ℌ' and ℌ': ℌ' : 𝔄 ↦↦Cα 𝔅
    by clarsimp
  from prems ℌ' assms αβ 𝔉𝔄 show
    (exp-cf-cat α (cf-id ℭ) 𝔄 ∘CF exp-cf-cat α 𝔉 𝔄)(⌊ObjMap⌋)(⌊ℌ⌋) =
      exp-cf-cat α 𝔉 𝔄(⌊ObjMap⌋)(⌊ℌ⌋)
    unfolding ℌ-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros
      )
qed
  (
    use assms αβ 𝔉𝔄 in
      ‹
        cs-concl
          cs-intro: cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros
      ›
  )
```

show
(*exp-cf-cat* α (*cf-id* $\mathfrak{C}$) $\mathfrak{A}$ $\circ_{CF}$ *exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{A}$)(|*ArrMap*|) =
exp-cf-cat α $\mathfrak{F}$ $\mathfrak{A}$(|*ArrMap*|)
proof(*rule vsv-eqI*, *unfold ArrMap-dom-lhs ArrMap-dom-rhs*)
fix $\mathfrak{M}$ **assume** $\mathfrak{M}$ $\in_{\circ}$ *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{B}$(|*Arr*|)
then obtain $\mathfrak{F}'$ $\mathfrak{G}'$ **where** $\mathfrak{M}$: $\mathfrak{M} : \mathfrak{F}' \mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}}\ \mathfrak{G}'$
by (*auto intro*: *is-arrI*)
note $\mathfrak{M}$ = *cat-FUNCT-is-arrD*[*OF* $\mathfrak{M}$]
from $\mathfrak{M}$(*1*) *assms* $\alpha\beta$ $\mathfrak{F}\mathfrak{A}$ **show**
(*exp-cf-cat* α (*cf-id* $\mathfrak{C}$) $\mathfrak{A}$ $\circ_{CF}$ *exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{A}$)(|*ArrMap*|)(|$\mathfrak{M}$|) =
exp-cf-cat α $\mathfrak{F}$ $\mathfrak{A}$(|*ArrMap*|)(|$\mathfrak{M}$|)
by (*subst* (*1 2*) $\mathfrak{M}$(*2*))
(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps*
cs-intro: *cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros*
)
qed
(
use assms $\alpha\beta$ **in**
‹
cs-concl
cs-intro: *cat-FUNCT-cs-intros cat-small-cs-intros cat-cs-intros*
›
)
qed *simp-all*

qed

25.7 Category raised to the power of a functor

25.7.1 Definition and elementary properties

definition *exp-cat-cf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{K}$ =
[
(
$\lambda \mathfrak{S} \in_{\circ} cat\text{-}FUNCT\ \alpha$ ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$(|*Obj*|).
cf-map (*cf-of-cf-map* ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$ $\mathfrak{S}$ $\circ_{CF}$ $\mathfrak{K}$)
),
(
$\lambda \sigma \in_{\circ} cat\text{-}FUNCT\ \alpha$ ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$(|*Arr*|).
ntcf-arrow (*ntcf-of-ntcf-arrow* ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$ σ $\circ_{NTCF-CF}$ $\mathfrak{K}$)
),
cat-FUNCT α ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$,
cat-FUNCT α ($\mathfrak{K}$(|*HomDom*|)) $\mathfrak{A}$
]$_{\circ}$

Components.

lemma *exp-cat-cf-components*:
shows *exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{K}$(|*ObjMap*|) =
(
$\lambda \mathfrak{S} \in_{\circ} cat\text{-}FUNCT\ \alpha$ ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$(|*Obj*|).
cf-map (*cf-of-cf-map* ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$ $\mathfrak{S}$ $\circ_{CF}$ $\mathfrak{K}$)
)
and *exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{K}$(|*ArrMap*|) =
(
$\lambda \sigma \in_{\circ} cat\text{-}FUNCT\ \alpha$ ($\mathfrak{K}$(|*HomCod*|)) $\mathfrak{A}$(|*Arr*|).

ntcf-arrow (*ntcf-of-ntcf-arrow* (𝔎(‖*HomCod*‖)) 𝔄 *σ* ∘$_{NTCF-CF}$ 𝔎)
)
and *exp-cat-cf* *α* 𝔄 𝔎(‖*HomDom*‖) = *cat-FUNCT* *α* (𝔎(‖*HomCod*‖)) 𝔄
and *exp-cat-cf* *α* 𝔄 𝔎(‖*HomCod*‖) = *cat-FUNCT* *α* (𝔎(‖*HomDom*‖)) 𝔄
unfolding *exp-cat-cf-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

25.7.2 Object map

context
fixes *α* 𝔎 𝔅 ℭ
assumes 𝔎: 𝔎 : 𝔅 ↦↦$_{C\alpha}$ ℭ
begin

interpretation 𝔎: *is-functor* *α* 𝔅 ℭ 𝔎 **by** (*rule* 𝔎)

mk-VLambda *exp-cat-cf-components*(*1*)[**where** 𝔎=𝔎 **and** *α*=*α*, *unfolded cat-cs-simps*]
|*vsv exp-cat-cf-components-ObjMap-vsv*[*cat-FUNCT-cs-intros*]|
|*vdomain exp-cat-cf-components-ObjMap-vdomain*[*cat-FUNCT-cs-simps*]|
|*app exp-cat-cf-components-ObjMap-app*[*cat-FUNCT-cs-simps*]|

end

25.7.3 Arrow map

context
fixes *α* 𝔎 𝔅 ℭ
assumes 𝔎: 𝔎 : 𝔅 ↦↦$_{C\alpha}$ ℭ
begin

interpretation 𝔎: *is-functor* *α* 𝔅 ℭ 𝔎 **by** (*rule* 𝔎)

mk-VLambda *exp-cat-cf-components*(*2*)[**where** 𝔎=𝔎 **and** *α*=*α*, *unfolded cat-cs-simps*]
|*vsv exp-cat-cf-components-ArrMap-vsv*[*cat-FUNCT-cs-intros*]|
|*vdomain exp-cat-cf-components-ArrMap-vdomain*[*cat-FUNCT-cs-simps*]|
|*app exp-cat-cf-components-ArrMap-app*[*cat-FUNCT-cs-simps*]|

end

25.7.4 Domain and codomain

context
fixes *α* 𝔎 𝔅 ℭ
assumes 𝔎: 𝔎 : 𝔅 ↦↦$_{C\alpha}$ ℭ
begin

interpretation 𝔎: *is-functor* *α* 𝔅 ℭ 𝔎 **by** (*rule* 𝔎)

lemmas *exp-cat-cf-HomDom*[*cat-FUNCT-cs-simps*] =
exp-cat-cf-components(*3*)[**where** 𝔎=𝔎 **and** *α*=*α*, *unfolded cat-cs-simps*]
and *exp-cat-cf-HomCod*[*cat-FUNCT-cs-simps*] =
exp-cat-cf-components(*4*)[**where** 𝔎=𝔎 **and** *α*=*α*, *unfolded cat-cs-simps*]

end

25.7.5 Category raised to the power of a functor is a functor

lemma *exp-cat-cf-is-tiny-functor*:
assumes 𝒵 *β* **and** *α* ∈$_{\circ}$ *β* **and** *category* *α* 𝔄 **and** 𝔎 : 𝔅 ↦↦$_{C\alpha}$ ℭ
shows *exp-cat-cf* *α* 𝔄 𝔎 : *cat-FUNCT* *α* ℭ 𝔄 ↦↦$_{C.tiny\beta}$ *cat-FUNCT* *α* 𝔅 𝔄

```
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret 𝔄: category α 𝔄 by (rule assms(3))
  interpret 𝔎: is-functor α 𝔅 ℭ 𝔎 by (rule assms(4))
  from assms(2–4) interpret ℭ𝔄: tiny-category β ‹cat-FUNCT α ℭ 𝔄›
    by (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
  from assms(2–4) interpret 𝔅𝔄: tiny-category β ‹cat-FUNCT α 𝔅 𝔄›
    by (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
  show ?thesis
  proof(intro is-tiny-functorI′ is-functorI′)
    show vfsequence (exp-cat-cf α 𝔄 𝔎) unfolding exp-cat-cf-def by auto
    show vcard (exp-cat-cf α 𝔄 𝔎) = 4ℕ
      unfolding exp-cat-cf-def by (simp-all add: nat-omega-simps)
    show ℛ∘ (exp-cat-cf α 𝔄 𝔎(|ObjMap|)) ⊆∘ cat-FUNCT α 𝔅 𝔄(|Obj|)
    proof
      (
        unfold cat-FUNCT-components exp-cat-cf-components,
        intro vrange-VLambda-vsubset,
        unfold cat-cs-simps
      )
      fix 𝔉 assume 𝔉 ∈∘ cf-maps α ℭ 𝔄
      then obtain 𝔉′ where 𝔉-def: 𝔉 = cf-map 𝔉′ and 𝔉′: 𝔉′ : ℭ ↦↦CF α 𝔄
        by auto
      from assms(2–4) 𝔉′ show
        cf-map (cf-of-cf-map ℭ 𝔄 𝔉 ∘CF 𝔎) ∈∘ cf-maps α 𝔅 𝔄
        unfolding 𝔉-def
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-FUNCT-cs-simps
              cs-intro: cat-cs-intros cat-FUNCT-cs-intros
          )
    qed
    show exp-cat-cf α 𝔄 𝔎(|ArrMap|)(|𝔑|) :
      exp-cat-cf α 𝔄 𝔎(|ObjMap|)(|𝔉|) ↦cat-FUNCT α 𝔅 𝔄
      exp-cat-cf α 𝔄 𝔎(|ObjMap|)(|𝔊|)
      if 𝔑 : 𝔉 ↦cat-FUNCT α ℭ 𝔄 𝔊 for 𝔉 𝔊 𝔑
    proof–
      note 𝔑 = cat-FUNCT-is-arrD[OF that]
      from 𝔑(1) assms(2–4) show ?thesis
        by (subst 𝔑(2), use nothing in ‹subst 𝔑(3), subst 𝔑(4)›)
          (
            cs-concl
              cs-simp: cat-cs-simps cat-FUNCT-cs-simps
              cs-intro: cat-cs-intros  cat-FUNCT-cs-intros
          )
    qed
    show
      exp-cat-cf α 𝔄 𝔎(|ArrMap|)(|𝔐 ∘A cat-FUNCT α ℭ 𝔄 𝔑|) =
        exp-cat-cf α 𝔄 𝔎(|ArrMap|)(|𝔐|) ∘A cat-FUNCT α 𝔅 𝔄
        exp-cat-cf α 𝔄 𝔎(|ArrMap|)(|𝔑|)
      if 𝔐 : 𝔊 ↦cat-FUNCT α ℭ 𝔄 ℌ and 𝔑 : 𝔉 ↦cat-FUNCT α ℭ 𝔄 𝔊
      for 𝔊 ℌ 𝔐 𝔉 𝔑
    proof–
      note 𝔐 = cat-FUNCT-is-arrD[OF that(1)]
        and 𝔑 = cat-FUNCT-is-arrD[OF that(2)]
      from 𝔐(1) 𝔑(1) assms(2–4) show ?thesis
        by (subst (1 2) 𝔐(2), use nothing in ‹subst (1 2) 𝔑(2)›)
```

```
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
    qed
    show
      exp-cat-cf α 𝔄 ℜ(|ArrMap|)(|cat-FUNCT α ℭ 𝔄(|CId|)(|ℱ|)|) =
        cat-FUNCT α 𝔅 𝔄(|CId|)(|exp-cat-cf α 𝔄 ℜ(|ObjMap|)(|ℱ|)|)
      if ℱ ∈₀ cat-FUNCT α ℭ 𝔄(|Obj|) for ℱ
    proof–
      from that have ℱ: ℱ ∈₀ cf-maps α ℭ 𝔄
        unfolding cat-FUNCT-components by simp
      then obtain ℱ' where ℱ-def: ℱ = cf-map ℱ' and ℱ': ℱ' : ℭ ↦↦_{Cα} 𝔄
        by auto
      from assms(2–4) ℱ ℱ' show ?thesis
        by
          (
        cs-concl
          cs-simp:
            cat-cs-simps cat-FUNCT-cs-simps cat-FUNCT-components(1) ℱ-def
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
          )
    qed
  qed
    (
      cs-concl
        cs-simp: cat-FUNCT-cs-simps
        cs-intro: cat-small-cs-intros cat-FUNCT-cs-intros cat-cs-intros
    )+
qed

lemma exp-cat-cf-is-tiny-functor'[cat-FUNCT-cs-intros]:
  assumes 𝒵 β
    and α ∈₀ β
    and category α 𝔄
    and ℜ : 𝔅 ↦↦_{Cα} ℭ
    and 𝔄' = cat-FUNCT α ℭ 𝔄
    and 𝔅' = cat-FUNCT α 𝔅 𝔄
  shows exp-cat-cf α 𝔄 ℜ : 𝔄' ↦↦_{C.tinyβ} 𝔅'
  using assms(1–4) unfolding assms(5,6) by (rule exp-cat-cf-is-tiny-functor)
```

25.7.6 Further properties

```
lemma exp-cat-cf-cat-cf-id:
  assumes category α 𝔄 and category α ℭ
  shows exp-cat-cf α 𝔄 (cf-id ℭ) = cf-id (cat-FUNCT α ℭ 𝔄)
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret ℭ: category α ℭ by (rule assms(2))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
```

proof(*rule cf-eqI*)

from $\alpha\beta$ **show** *exp-cat-cf* α $\mathfrak{A}$ (*cf-id* $\mathfrak{C}$) :
cat-FUNCT α $\mathfrak{C}$ $\mathfrak{A}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
from $\alpha\beta$ **show**
cf-id (*cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$) : *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$
by
(
cs-concl
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
from $\alpha\beta$ **have** *ObjMap-dom-lhs*:
$\mathcal{D}_\circ$ (*exp-cat-cf* α $\mathfrak{A}$ (*cf-id* $\mathfrak{C}$)(⟦*ObjMap*⟧)) = *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$(⟦*Obj*⟧)
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros*
)
from $\alpha\beta$ **have** *ObjMap-dom-rhs*:
$\mathcal{D}_\circ$ (*cf-id* (*cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$)(⟦*ObjMap*⟧)) = *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$(⟦*Obj*⟧)
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-FUNCT-cs-intros*
)
show *exp-cat-cf* α $\mathfrak{A}$ (*cf-id* $\mathfrak{C}$)(⟦*ObjMap*⟧) = *cf-id* (*cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$)(⟦*ObjMap*⟧)
proof(*rule vsv-eqI*, *unfold ObjMap-dom-lhs ObjMap-dom-rhs cat-FUNCT-components*(*1*))
fix $\mathfrak{F}$ **assume** $\mathfrak{F}$ $\in_\circ$ *cf-maps* α $\mathfrak{C}$ $\mathfrak{A}$
then obtain $\mathfrak{F}'$ **where** $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-map* $\mathfrak{F}'$ **and** $\mathfrak{F}'$: $\mathfrak{F}'$: $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{A}$
by *clarsimp*
from $\mathfrak{F}'$ **show**
exp-cat-cf α $\mathfrak{A}$ (*cf-id* $\mathfrak{C}$)(⟦*ObjMap*⟧)(⟦$\mathfrak{F}$⟧) =
cf-id (*cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$)(⟦*ObjMap*⟧)(⟦$\mathfrak{F}$⟧)
by
(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps* $\mathfrak{F}$-*def*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)
qed (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)+

from $\alpha\beta$ **have** *ArrMap-dom-lhs*:
$\mathcal{D}_\circ$ (*exp-cat-cf* α $\mathfrak{A}$ (*cf-id* $\mathfrak{C}$)(⟦*ArrMap*⟧)) = *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$(⟦*Arr*⟧)
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros*
)
from $\alpha\beta$ **have** *ArrMap-dom-rhs*:

```
    𝒟∘ (cf-id (cat-FUNCT α 𝔈 𝔄)(|ArrMap|)) = cat-FUNCT α 𝔈 𝔄(|Arr|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-FUNCT-cs-intros
      )
  show exp-cat-cf α 𝔄 (cf-id 𝔈)(|ArrMap|) = cf-id (cat-FUNCT α 𝔈 𝔄)(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix 𝔑 assume 𝔑 ∈∘ cat-FUNCT α 𝔈 𝔄(|Arr|)
    then obtain ℌ ℌ' where 𝔑: 𝔑 : ℌ ↦_cat-FUNCT α 𝔈 𝔄 ℌ'
      by (auto intro: is-arrI)
    note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
    from 𝔑(1) show
      exp-cat-cf α 𝔄 (cf-id 𝔈)(|ArrMap|)(|𝔑|) =
        cf-id (cat-FUNCT α 𝔈 𝔄)(|ArrMap|)(|𝔑|)
      by (subst (1 2) 𝔑(2))
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)+

qed simp-all

qed

lemma exp-cat-cf-cf-comp:
  assumes category α 𝔄 and 𝔊 : 𝔈 ↦↦_Cα 𝔇 and 𝔉 : 𝔅 ↦↦_Cα 𝔈
  shows exp-cat-cf α 𝔄 (𝔊 ∘_CF 𝔉) = exp-cat-cf α 𝔄 𝔉 ∘_CF exp-cat-cf α 𝔄 𝔊
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔈 𝔇 𝔊 by (rule assms(2))
  interpret 𝔉: is-functor α 𝔅 𝔈 𝔉 by (rule assms(3))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule cf-eqI)
    from β αβ show exp-cat-cf α 𝔄 (𝔊 ∘_CF 𝔉) :
      cat-FUNCT α 𝔇 𝔄 ↦↦_Cβ cat-FUNCT α 𝔅 𝔄
      by
        (
          cs-concl
            cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
        )
    from β αβ show exp-cat-cf α 𝔄 𝔉 ∘_CF exp-cat-cf α 𝔄 𝔊 :
      cat-FUNCT α 𝔇 𝔄 ↦↦_Cβ cat-FUNCT α 𝔅 𝔄
      by
        (
          cs-concl
            cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
        )
```

```
from β αβ have ObjMap-dom-lhs:
  𝒟∘ (exp-cat-cf α 𝔄 (𝔊 ∘CF 𝔉)⦇ObjMap⦈) = cat-FUNCT α 𝔇 𝔄⦇Obj⦈
  by
    (
      cs-concl
        cs-simp: cat-FUNCT-cs-simps
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
from β αβ have ObjMap-dom-rhs:
  𝒟∘ ((exp-cat-cf α 𝔄 𝔉 ∘CF exp-cat-cf α 𝔄 𝔊)⦇ObjMap⦈) =
    cat-FUNCT α 𝔇 𝔄⦇Obj⦈
  by
    (
      cs-concl
        cs-simp: cat-cs-simps cat-FUNCT-cs-simps
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
from β αβ have ArrMap-dom-lhs:
  𝒟∘ (exp-cat-cf α 𝔄 (𝔊 ∘CF 𝔉)⦇ArrMap⦈) = cat-FUNCT α 𝔇 𝔄⦇Arr⦈
  by
    (
      cs-concl
        cs-simp: cat-FUNCT-cs-simps
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
from β αβ have ArrMap-dom-rhs:
  𝒟∘ ((exp-cat-cf α 𝔄 𝔉 ∘CF exp-cat-cf α 𝔄 𝔊)⦇ArrMap⦈) =
    cat-FUNCT α 𝔇 𝔄⦇Arr⦈
  by
    (
      cs-concl
        cs-simp: cat-cs-simps cat-FUNCT-cs-simps
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
    )
show
  exp-cat-cf α 𝔄 (𝔊 ∘CF 𝔉)⦇ObjMap⦈ =
    (exp-cat-cf α 𝔄 𝔉 ∘CF exp-cat-cf α 𝔄 𝔊)⦇ObjMap⦈
proof
  (
    rule vsv-eqI,
    unfold ObjMap-dom-lhs ObjMap-dom-rhs cat-FUNCT-components(1)
  )
  fix ℌ assume ℌ ∈∘ cf-maps α 𝔇 𝔄
  then obtain ℌ' where ℌ-def: ℌ = cf-map ℌ' and ℌ': ℌ' : 𝔇 ↦↦CF α 𝔄
    by clarsimp
  from β αβ ℌ' assms show
    exp-cat-cf α 𝔄 (𝔊 ∘CF 𝔉)⦇ObjMap⦈⦇ℌ⦈ =
      (exp-cat-cf α 𝔄 𝔉 ∘CF exp-cat-cf α 𝔄 𝔊)⦇ObjMap⦈⦇ℌ⦈
    unfolding ℌ-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
qed
  (
    use β αβ in
```

```
      ‹
        cs-concl
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      ›
    )+
  show exp-cat-cf α 𝔄 (𝔊 ∘_CF 𝔉)(|ArrMap|) =
    (exp-cat-cf α 𝔄 𝔉 ∘_CF exp-cat-cf α 𝔄 𝔊)(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix 𝔑 assume 𝔑 ∈₀ cat-FUNCT α 𝔇 𝔄(|Arr|)
    then obtain ℌ ℌ' where 𝔑: 𝔑 : ℌ ↦_cat-FUNCT α 𝔇 𝔄 ℌ'
      by (auto intro: is-arrI)
    note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
    from assms 𝔑(1) β αβ show
      exp-cat-cf α 𝔄 (𝔊 ∘_CF 𝔉)(|ArrMap|)(|𝔑|) =
        (exp-cat-cf α 𝔄 𝔉 ∘_CF exp-cat-cf α 𝔄 𝔊)(|ArrMap|)(|𝔑|)
      by (subst (1 2) 𝔑(2))
        (
          cs-concl
            cs-simp:
              cat-FUNCT-cs-simps cat-cs-simps ntcf-cf-comp-ntcf-cf-comp-assoc
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
    (
      use β αβ in
        ‹
          cs-concl
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        ›
    )+
qed simp-all

qed
```

25.8 Natural transformation raised to the power of a category

25.8.1 Definition and elementary properties

```
definition exp-ntcf-cat :: V ⇒ V ⇒ V ⇒ V
  where exp-ntcf-cat α 𝔑 𝔇 =
    [
      (
        λ𝔖∈₀cat-FUNCT α 𝔇 (𝔑(|NTDGDom|))(|Obj|).
          ntcf-arrow (𝔑 ∘_NTCF-CF cf-of-cf-map 𝔇 (𝔑(|NTDGDom|)) 𝔖)
      ),
      exp-cf-cat α (𝔑(|NTDom|)) 𝔇,
      exp-cf-cat α (𝔑(|NTCod|)) 𝔇,
      cat-FUNCT α 𝔇 (𝔑(|NTDGDom|)),
      cat-FUNCT α 𝔇 (𝔑(|NTDGCod|))
    ]₀
```

Components.

```
lemma exp-ntcf-cat-components:
  shows exp-ntcf-cat α 𝔑 𝔇(|NTMap|) =
    (
      λ𝔖∈₀cat-FUNCT α 𝔇 (𝔑(|NTDGDom|))(|Obj|).
        ntcf-arrow (𝔑 ∘_NTCF-CF cf-of-cf-map 𝔇 (𝔑(|NTDGDom|)) 𝔖)
```

)
and *exp-ntcf-cat* α $\mathfrak{N}$ $\mathfrak{D}(\lfloor NTDom \rfloor)$ = *exp-cf-cat* α $(\mathfrak{N}(\lfloor NTDom \rfloor))$ $\mathfrak{D}$
and *exp-ntcf-cat* α $\mathfrak{N}$ $\mathfrak{D}(\lfloor NTCod \rfloor)$ = *exp-cf-cat* α $(\mathfrak{N}(\lfloor NTCod \rfloor))$ $\mathfrak{D}$
and *exp-ntcf-cat* α $\mathfrak{N}$ $\mathfrak{D}(\lfloor NTDGDom \rfloor)$ = *cat-FUNCT* α $\mathfrak{D}$ $(\mathfrak{N}(\lfloor NTDGDom \rfloor))$
and *exp-ntcf-cat* α $\mathfrak{N}$ $\mathfrak{D}(\lfloor NTDGCod \rfloor)$ = *cat-FUNCT* α $\mathfrak{D}$ $(\mathfrak{N}(\lfloor NTDGCod \rfloor))$
unfolding *exp-ntcf-cat-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

25.8.2 Natural transformation map

mk-VLambda *exp-ntcf-cat-components*(*1*)
|*vsv exp-ntcf-cat-components-NTMap-vsv*[*cat-FUNCT-cs-intros*]|

context *is-ntcf*
begin

lemmas *exp-ntcf-cat-components′* =
exp-ntcf-cat-components[**where** α=α **and** $\mathfrak{N}$=$\mathfrak{N}$, *unfolded cat-cs-simps*]

lemmas [*cat-FUNCT-cs-simps*] = *exp-ntcf-cat-components′*(*2–5*)

mk-VLambda *exp-ntcf-cat-components*(*1*)[**where** $\mathfrak{N}$=$\mathfrak{N}$, *unfolded cat-cs-simps*]
|*vdomain exp-ntcf-cat-components-NTMap-vdomain*[*cat-FUNCT-cs-simps*]|
|*app exp-ntcf-cat-components-NTMap-app*[*cat-FUNCT-cs-simps*]|

end

lemmas [*cat-FUNCT-cs-simps*] =
is-ntcf.exp-ntcf-cat-components′(*2–5*)
is-ntcf.exp-ntcf-cat-components-NTMap-vdomain
is-ntcf.exp-ntcf-cat-components-NTMap-app

25.8.3 Natural transformation raised to the power of a category is a natural transformation

lemma *exp-ntcf-cat-is-tiny-ntcf*:
assumes $\mathcal{Z}$ β
and α $\in_\circ$ β
and $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
and *category* α $\mathfrak{D}$
shows *exp-ntcf-cat* α $\mathfrak{N}$ $\mathfrak{D}$:
exp-cf-cat α $\mathfrak{F}$ $\mathfrak{D}$ $\mapsto_{CF.tiny}$ *exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{D}$:
cat-FUNCT α $\mathfrak{D}$ $\mathfrak{A}$ $\mapsto\mapsto_{C.tiny\beta}$ *cat-FUNCT* α $\mathfrak{D}$ $\mathfrak{B}$
proof(*rule is-tiny-ntcfI′*)

interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{D}$: *category* α $\mathfrak{D}$ **by** (*rule assms*(*4*))

let *?exp-*$\mathfrak{N}$ = ‹*exp-ntcf-cat* α $\mathfrak{N}$ $\mathfrak{D}$›
let *?exp-*$\mathfrak{F}$ = ‹*exp-cf-cat* α $\mathfrak{F}$ $\mathfrak{D}$›
let *?exp-*$\mathfrak{G}$ = ‹*exp-cf-cat* α $\mathfrak{G}$ $\mathfrak{D}$›

from *assms*(*1*,*2*) **show**
exp-cf-cat α $\mathfrak{F}$ $\mathfrak{D}$: *cat-FUNCT* α $\mathfrak{D}$ $\mathfrak{A}$ $\mapsto\mapsto_{C.tiny\beta}$ *cat-FUNCT* α $\mathfrak{D}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *assms*(*1*,*2*) **show**
exp-cf-cat α $\mathfrak{G}$ $\mathfrak{D}$: *cat-FUNCT* α $\mathfrak{D}$ $\mathfrak{A}$ $\mapsto\mapsto_{C.tiny\beta}$ *cat-FUNCT* α $\mathfrak{D}$ $\mathfrak{B}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)

```
show ?exp-𝔑 :
  ?exp-𝔉 ↦_CF ?exp-𝔊 : cat-FUNCT α 𝔇 𝔄 ↦↦_Cβ cat-FUNCT α 𝔇 𝔅
proof(rule is-ntcfI')

  show vfsequence (?exp-𝔑) unfolding exp-ntcf-cat-def by auto
  show vcard (?exp-𝔑) = 5_ℕ
    unfolding exp-ntcf-cat-def by (simp add: nat-omega-simps)

  from assms(1,2) show ?exp-𝔉 : cat-FUNCT α 𝔇 𝔄 ↦↦_Cβ cat-FUNCT α 𝔇 𝔅
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from assms(1,2) show ?exp-𝔊 : cat-FUNCT α 𝔇 𝔄 ↦↦_Cβ cat-FUNCT α 𝔇 𝔅
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  show ?exp-𝔑(|NTMap|)(|𝔎|) :
    ?exp-𝔉(|ObjMap|)(|𝔎|) ↦_cat-FUNCT α 𝔇 𝔅 ?exp-𝔊(|ObjMap|)(|𝔎|)
    if 𝔎 ∈_∘ cat-FUNCT α 𝔇 𝔄(|Obj|) for 𝔎
  proof–
    from that[unfolded cat-FUNCT-cs-simps] have 𝔎 ∈_∘ cf-maps α 𝔇 𝔄 by simp
    then obtain 𝔎' where 𝔎-def: 𝔎 = cf-map 𝔎' and 𝔎': 𝔎' : 𝔇 ↦↦_Cα 𝔄
      by auto
    from 𝔎' show ?thesis
      by
        (
          cs-concl
            cs-simp: cat-FUNCT-cs-simps 𝔎-def
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed

  show
    ?exp-𝔑(|NTMap|)(|𝔗|) ∘_A cat-FUNCT α 𝔇 𝔅 ?exp-𝔉(|ArrMap|)(|𝔏|) =
      ?exp-𝔊(|ArrMap|)(|𝔏|) ∘_A cat-FUNCT α 𝔇 𝔅 ?exp-𝔑(|NTMap|)(|𝔖|)
    if 𝔏 : 𝔖 ↦_cat-FUNCT α 𝔇 𝔄 𝔗 for 𝔖 𝔗 𝔏
  proof–
    note 𝔏 = cat-FUNCT-is-arrD[OF that]
    let ?𝔖 = ‹cf-of-cf-map 𝔇 𝔄 𝔖›
      and ?𝔗 = ‹cf-of-cf-map 𝔇 𝔄 𝔗›
      and ?𝔏 = ‹ntcf-of-ntcf-arrow 𝔇 𝔄 𝔏›
    have [cat-cs-simps]:
      (𝔑 ∘_NTCF-CF ?𝔗) ·_NTCF (𝔉 ∘_CF-NTCF ?𝔏) =
        (𝔊 ∘_CF-NTCF ?𝔏) ·_NTCF (𝔑 ∘_NTCF-CF ?𝔖)
    proof(rule ntcf-eqI)
      from 𝔏(1) show
        (𝔑 ∘_NTCF-CF ?𝔗) ·_NTCF (𝔉 ∘_CF-NTCF ?𝔏) :
          𝔉 ∘_CF ?𝔖 ↦_CF 𝔊 ∘_CF ?𝔗 : 𝔇 ↦↦_Cα 𝔅
        by (cs-concl cs-intro: cat-cs-intros)
      from 𝔏(1) show
        (𝔊 ∘_CF-NTCF ?𝔏) ·_NTCF (𝔑 ∘_NTCF-CF ?𝔖) :
          𝔉 ∘_CF ?𝔖 ↦_CF 𝔊 ∘_CF ?𝔗 : 𝔇 ↦↦_Cα 𝔅
```

```
      by (cs-concl cs-intro: cat-cs-intros)
    from 𝔏(1) have dom-lhs:
      𝒟∘ (((𝔑 ∘NTCF−CF ?𝔗) ⋅NTCF (𝔉 ∘CF−NTCF ?𝔏))(⦇NTMap⦈)) = 𝔇(⦇Obj⦈)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from 𝔏(1) have dom-rhs:
      𝒟∘ (((𝔊 ∘CF−NTCF ?𝔏) ⋅NTCF (𝔑 ∘NTCF−CF ?𝔖))(⦇NTMap⦈)) = 𝔇(⦇Obj⦈)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show
      ((𝔑 ∘NTCF−CF ?𝔗) ⋅NTCF (𝔉 ∘CF−NTCF ?𝔏))(⦇NTMap⦈) =
        ((𝔊 ∘CF−NTCF ?𝔏) ⋅NTCF (𝔑 ∘NTCF−CF ?𝔖))(⦇NTMap⦈)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix d assume d ∈∘ 𝔇(⦇Obj⦈)
      with 𝔏(1) show
        ((𝔑 ∘NTCF−CF ?𝔗) ⋅NTCF (𝔉 ∘CF−NTCF ?𝔏))(⦇NTMap⦈)(⦇d⦈) =
          ((𝔊 ∘CF−NTCF ?𝔏) ⋅NTCF (𝔑 ∘NTCF−CF ?𝔖))(⦇NTMap⦈)(⦇d⦈)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed (cs-concl cs-intro: cat-cs-intros)
  qed simp-all
  from 𝔏(1,3,4) that show ?thesis
    by (subst (1 2) 𝔏(2), use nothing in ‹subst 𝔏(3), subst 𝔏(4)›)
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
 qed
qed
 (
   cs-concl
     cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
 )+

qed

lemma exp-ntcf-cat-is-tiny-ntcf'[cat-FUNCT-cs-intros]:
 assumes ℤ β
   and α ∈∘ β
   and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
   and category α 𝔇
   and 𝔉' = exp-cf-cat α 𝔉 𝔇
   and 𝔊' = exp-cf-cat α 𝔊 𝔇
   and 𝔄' = cat-FUNCT α 𝔇 𝔄
   and 𝔅' = cat-FUNCT α 𝔇 𝔅
 shows exp-ntcf-cat α 𝔑 𝔇 : 𝔉' ↦CF.tiny 𝔊' : 𝔄' ↦↦C.tinyβ 𝔅'
 using assms(1–4) unfolding assms(5–8) by (rule exp-ntcf-cat-is-tiny-ntcf)
```

25.8.4 Further properties

```
lemma exp-ntcf-cat-cf-ntcf-comp:
 assumes 𝔑 : 𝔉 ↦CF 𝔊 : 𝔄 ↦↦Cα 𝔅
   and ℌ : 𝔅 ↦↦Cα ℭ
   and category α 𝔇
 shows
   exp-ntcf-cat α (ℌ ∘CF−NTCF 𝔑) 𝔇 =
     exp-cf-cat α ℌ 𝔇 ∘CF−NTCF exp-ntcf-cat α 𝔑 𝔇
proof–

 interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
```

```
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  interpret 𝔇: category α 𝔇 by (rule assms(3))

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔑.𝒵-Limit-αω 𝔑.𝒵-ω-αω 𝒵-def 𝔑.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)
    from αβ have dom-lhs:
      𝒟∘ (exp-ntcf-cat α (ℌ ∘_CF-NTCF 𝔑) 𝔇(|NTMap|)) = cat-FUNCT α 𝔇 𝔄(|Obj|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ have dom-rhs:
      𝒟∘ ((exp-cf-cat α ℌ 𝔇 ∘_CF-NTCF exp-ntcf-cat α 𝔑 𝔇)(|NTMap|)) =
        cat-FUNCT α 𝔇 𝔄(|Obj|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    show
      exp-ntcf-cat α (ℌ ∘_CF-NTCF 𝔑) 𝔇(|NTMap|) =
        (exp-cf-cat α ℌ 𝔇 ∘_CF-NTCF exp-ntcf-cat α 𝔑 𝔇)(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
      fix 𝔎 assume prems: 𝔎 ∈∘ cf-maps α 𝔇 𝔄
      then obtain 𝔎' where 𝔎-def: 𝔎 = cf-map 𝔎' and 𝔎': 𝔎' : 𝔇 ↦↦_Cα 𝔄
        by (auto intro: is-arrI)
      from αβ prems 𝔎' show
        exp-ntcf-cat α (ℌ ∘_CF-NTCF 𝔑) 𝔇(|NTMap|)(|𝔎|) =
          (exp-cf-cat α ℌ 𝔇 ∘_CF-NTCF exp-ntcf-cat α 𝔑 𝔇)(|NTMap|)(|𝔎|)
        by
          (
            cs-concl
              cs-simp:
                cf-ntcf-comp-ntcf-cf-comp-assoc
                cat-cs-simps cat-FUNCT-cs-simps
                𝔎-def
              cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )
    qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
  qed
    (
      cs-concl
        cs-simp: exp-cf-cat-cf-comp cat-cs-simps cat-FUNCT-cs-simps
        cs-intro: αβ cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )+

qed

lemma exp-ntcf-cat-ntcf-cf-comp:
  assumes 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔅 ↦↦_Cα ℭ
```

```
    and 𝔥 : 𝔄 ↦↦_Cα 𝔅
    and category α 𝔇
  shows
    exp-ntcf-cat α (𝔑 ∘_NTCF-CF 𝔥) 𝔇 =
      exp-ntcf-cat α 𝔑 𝔇 ∘_NTCF-CF exp-cf-cat α 𝔥 𝔇
proof–

  interpret 𝔑: is-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret 𝔥: is-functor α 𝔄 𝔅 𝔥 by (rule assms(2))
  interpret 𝔇: category α 𝔇 by (rule assms(3))

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝔑.𝒵-Limit-αω 𝔑.𝒵-ω-αω 𝒵-def 𝔑.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)
    from αβ have dom-lhs:
      𝒟₀ (exp-ntcf-cat α (𝔑 ∘_NTCF-CF 𝔥) 𝔇(⌊NTMap⌋)) = cat-FUNCT α 𝔇 𝔄(⌊Obj⌋)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ have dom-rhs:
      𝒟₀ ((exp-ntcf-cat α 𝔑 𝔇 ∘_NTCF-CF exp-cf-cat α 𝔥 𝔇)(⌊NTMap⌋)) =
        cat-FUNCT α 𝔇 𝔄(⌊Obj⌋)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    show
      exp-ntcf-cat α (𝔑 ∘_NTCF-CF 𝔥) 𝔇(⌊NTMap⌋) =
        (exp-ntcf-cat α 𝔑 𝔇 ∘_NTCF-CF exp-cf-cat α 𝔥 𝔇)(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
      fix 𝔎 assume prems: 𝔎 ∈₀ cf-maps α 𝔇 𝔄
      then obtain 𝔎' where 𝔎-def: 𝔎 = cf-map 𝔎' and 𝔎': 𝔎' : 𝔇 ↦↦_Cα 𝔄
        by (auto intro: is-arrI)
      from αβ assms prems 𝔎' show
        exp-ntcf-cat α (𝔑 ∘_NTCF-CF 𝔥) 𝔇(⌊NTMap⌋)(⌊𝔎⌋) =
          (exp-ntcf-cat α 𝔑 𝔇 ∘_NTCF-CF exp-cf-cat α 𝔥 𝔇)(⌊NTMap⌋)(⌊𝔎⌋)
        by
          (
            cs-concl
              cs-simp:
                ntcf-cf-comp-ntcf-cf-comp-assoc
                cat-cs-simps cat-FUNCT-cs-simps
                𝔎-def
              cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )
    qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
  qed
    (
      cs-concl
```

```
      cs-simp: exp-cf-cat-cf-comp cat-cs-simps cat-FUNCT-cs-simps
      cs-intro: αβ cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )+
qed

lemma exp-ntcf-cat-ntcf-vcomp:
  assumes category α 𝔄
    and 𝔐 : 𝔊 ↦CF 𝔎 : 𝔅 ↦↦Cα ℭ
    and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα ℭ
  shows
    exp-ntcf-cat α (𝔐 ∘NTCF 𝔑) 𝔄 =
      exp-ntcf-cat α 𝔐 𝔄 ∘NTCF exp-ntcf-cat α 𝔑 𝔄
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔐: is-ntcf α 𝔅 ℭ 𝔊 𝔎 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(3))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)
    from αβ show
      exp-ntcf-cat α (𝔐 ∘NTCF 𝔑) 𝔄 :
        exp-cf-cat α 𝔉 𝔄 ↦CF exp-cf-cat α 𝔎 𝔄 :
        cat-FUNCT α 𝔄 𝔅 ↦↦Cβ cat-FUNCT α 𝔄 ℭ
      by
        (
          cs-concl
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ show
      exp-ntcf-cat α 𝔐 𝔄 ∘NTCF exp-ntcf-cat α 𝔑 𝔄 :
        exp-cf-cat α 𝔉 𝔄 ↦CF exp-cf-cat α 𝔎 𝔄 :
        cat-FUNCT α 𝔄 𝔅 ↦↦Cβ cat-FUNCT α 𝔄 ℭ
      by
        (
          cs-concl
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ have dom-lhs:
      𝒟∘ ((exp-ntcf-cat α 𝔐 𝔄 ∘NTCF exp-ntcf-cat α 𝔑 𝔄)(⌊NTMap⌋)) =
        cat-FUNCT α 𝔄 𝔅(⌊Obj⌋)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    have dom-rhs:
      𝒟∘ (exp-ntcf-cat α (𝔐 ∘NTCF 𝔑) 𝔄(⌊NTMap⌋)) = cat-FUNCT α 𝔄 𝔅(⌊Obj⌋)
      by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)
    show
      exp-ntcf-cat α (𝔐 ∘NTCF 𝔑) 𝔄(⌊NTMap⌋) =
        (exp-ntcf-cat α 𝔐 𝔄 ∘NTCF exp-ntcf-cat α 𝔑 𝔄)(⌊NTMap⌋)
```

```
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
      fix 𝔉′ assume 𝔉′ ∈∘ cf-maps α 𝔄 𝔅
      then obtain 𝔉″
        where 𝔉′-def: 𝔉′ = cf-map 𝔉″ and 𝔉″: 𝔉″ : 𝔄 ↦↦_{Cα} 𝔅
        by auto
      from 𝔉″ αβ show
        exp-ntcf-cat α (𝔐 ⊙_{NTCF} 𝔑) 𝔄⦇NTMap⦈⦇𝔉′⦈ =
          (exp-ntcf-cat α 𝔐 𝔄 ⊙_{NTCF} exp-ntcf-cat α 𝔑 𝔄)⦇NTMap⦈⦇𝔉′⦈
        unfolding 𝔉′-def
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-FUNCT-cs-simps
              cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )
    qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)+
  qed simp-all

qed

lemma ntcf-id-exp-cf-cat:
  assumes category α 𝔄 and 𝔉 : 𝔅 ↦↦_{Cα} ℭ
  shows ntcf-id (exp-cf-cat α 𝔉 𝔄) = exp-ntcf-cat α (ntcf-id 𝔉) 𝔄
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔉: is-functor α 𝔅 ℭ 𝔉 by (rule assms(2))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)
    from αβ show exp-ntcf-cat α (ntcf-id 𝔉) 𝔄 :
      exp-cf-cat α 𝔉 𝔄 ↦_{CF} exp-cf-cat α 𝔉 𝔄 :
      cat-FUNCT α 𝔄 𝔅 ↦↦_{Cβ} cat-FUNCT α 𝔄 ℭ
      by
        (
          cs-concl
            cs-simp: cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ show ntcf-id (exp-cf-cat α 𝔉 𝔄) :
      exp-cf-cat α 𝔉 𝔄 ↦_{CF} exp-cf-cat α 𝔉 𝔄 :
      cat-FUNCT α 𝔄 𝔅 ↦↦_{Cβ} cat-FUNCT α 𝔄 ℭ
      by
        (
          cs-concl
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ assms have dom-lhs:
      𝒟∘ (ntcf-id (exp-cf-cat α 𝔉 𝔄)⦇NTMap⦈) = cat-FUNCT α 𝔄 𝔅⦇Obj⦈
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
```

```
        cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )
    from αβ assms have dom-rhs:
      𝒟₀ (exp-ntcf-cat α (ntcf-id 𝔉) 𝔄(⁅NTMap⁆)) = cat-FUNCT α 𝔄 𝔅(⁅Obj⁆)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
        )
    show
      ntcf-id (exp-cf-cat α 𝔉 𝔄)(⁅NTMap⁆) = exp-ntcf-cat α (ntcf-id 𝔉) 𝔄(⁅NTMap⁆)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
      fix 𝔊 assume 𝔊 ∈₀ cf-maps α 𝔄 𝔅
      then obtain 𝔊'
        where 𝔊-def: 𝔊 = cf-map 𝔊' and 𝔊': 𝔊' : 𝔄 ↦↦_Cα 𝔅
        by auto
      from 𝔊' αβ show
        ntcf-id (exp-cf-cat α 𝔉 𝔄)(⁅NTMap⁆)(⁅𝔊⁆) =
          exp-ntcf-cat α (ntcf-id 𝔉) 𝔄(⁅NTMap⁆)(⁅𝔊⁆)
        unfolding 𝔊-def
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-FUNCT-cs-simps
              cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
          )
    qed
      (
        cs-concl
          cs-intro: αβ cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )+
  qed simp-all
qed
```

25.9 Category raised to the power of the natural transformation

25.9.1 Definition and elementary properties

```
definition exp-cat-ntcf :: V ⇒ V ⇒ V ⇒ V
  where exp-cat-ntcf α ℭ 𝔑 =
    [
      (
        λ𝔖∈₀cat-FUNCT α (𝔑(⁅NTDGCod⁆)) ℭ(⁅Obj⁆).
          ntcf-arrow (cf-of-cf-map (𝔑(⁅NTDGCod⁆)) ℭ 𝔖 ∘_CF-NTCF 𝔑)
      ),
      exp-cat-cf α ℭ (𝔑(⁅NTDom⁆)),
      exp-cat-cf α ℭ (𝔑(⁅NTCod⁆)),
      cat-FUNCT α (𝔑(⁅NTDGCod⁆)) ℭ,
      cat-FUNCT α (𝔑(⁅NTDGDom⁆)) ℭ
    ]₀
```

Components.

```
lemma exp-cat-ntcf-components:
  shows exp-cat-ntcf α ℭ 𝔑(⁅NTMap⁆) =
    (
      λ𝔖∈₀cat-FUNCT α (𝔑(⁅NTDGCod⁆)) ℭ(⁅Obj⁆).
        ntcf-arrow (cf-of-cf-map (𝔑(⁅NTDGCod⁆)) ℭ 𝔖 ∘_CF-NTCF 𝔑)
```

)
and *exp-cat-ntcf* α $\mathfrak{C}$ $\mathfrak{N}(\!|NTDom|\!)$ = *exp-cat-cf* α $\mathfrak{C}$ ($\mathfrak{N}(\!|NTDom|\!)$)
and *exp-cat-ntcf* α $\mathfrak{C}$ $\mathfrak{N}(\!|NTCod|\!)$ = *exp-cat-cf* α $\mathfrak{C}$ ($\mathfrak{N}(\!|NTCod|\!)$)
and *exp-cat-ntcf* α $\mathfrak{C}$ $\mathfrak{N}(\!|NTDGDom|\!)$ = *cat-FUNCT* α ($\mathfrak{N}(\!|NTDGCod|\!)$) $\mathfrak{C}$
and *exp-cat-ntcf* α $\mathfrak{C}$ $\mathfrak{N}(\!|NTDGCod|\!)$ = *cat-FUNCT* α ($\mathfrak{N}(\!|NTDGDom|\!)$) $\mathfrak{C}$
unfolding *exp-cat-ntcf-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

25.9.2 Natural transformation map

mk-VLambda *exp-cat-ntcf-components*(*1*)
|*vsv exp-cat-ntcf-components-NTMap-vsv*[*cat-FUNCT-cs-intros*]|

context *is-ntcf*
begin

lemmas *exp-cat-ntcf-components′* =
exp-cat-ntcf-components[**where** α=α **and** $\mathfrak{N}$=$\mathfrak{N}$, *unfolded cat-cs-simps*]

lemmas [*cat-FUNCT-cs-simps*] = *exp-cat-ntcf-components′*(*2–5*)

mk-VLambda *exp-cat-ntcf-components*(*1*)[**where** $\mathfrak{N}$=$\mathfrak{N}$, *unfolded cat-cs-simps*]
|*vdomain exp-cat-ntcf-components-NTMap-vdomain*[*cat-FUNCT-cs-simps*]|
|*app exp-cat-ntcf-components-NTMap-app*[*cat-FUNCT-cs-simps*]|

end

lemmas *exp-cat-ntcf-components′* = *is-ntcf.exp-cat-ntcf-components′*

lemmas [*cat-FUNCT-cs-simps*] =
is-ntcf.exp-cat-ntcf-components′(*2–5*)
is-ntcf.exp-cat-ntcf-components-NTMap-vdomain
is-ntcf.exp-cat-ntcf-components-NTMap-app

25.9.3 Category raised to the power of a natural transformation is a natural transformation

lemma *exp-cat-ntcf-is-tiny-ntcf*:
assumes $\mathcal{Z}$ β
and α $\in_\circ$ β
and $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{B}$
and *category* α $\mathfrak{C}$
shows *exp-cat-ntcf* α $\mathfrak{C}$ $\mathfrak{N}$:
exp-cat-cf α $\mathfrak{C}$ $\mathfrak{F}$ $\mapsto_{CF.tiny}$ *exp-cat-cf* α $\mathfrak{C}$ $\mathfrak{G}$:
cat-FUNCT α $\mathfrak{B}$ $\mathfrak{C}$ $\mapsto\mapsto_{C.tiny\beta}$ *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{C}$
proof(*rule is-tiny-ntcfI′*)

interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
interpret $\mathfrak{C}$: *category* α $\mathfrak{C}$ **by** (*rule assms*(*4*))

let *?exp-*$\mathfrak{N}$ = ‹*exp-cat-ntcf* α $\mathfrak{C}$ $\mathfrak{N}$›
let *?exp-*$\mathfrak{F}$ = ‹*exp-cat-cf* α $\mathfrak{C}$ $\mathfrak{F}$›
let *?exp-*$\mathfrak{G}$ = ‹*exp-cat-cf* α $\mathfrak{C}$ $\mathfrak{G}$›

from *assms*(*1,2*) **show**
exp-cat-cf α $\mathfrak{C}$ $\mathfrak{G}$: *cat-FUNCT* α $\mathfrak{B}$ $\mathfrak{C}$ $\mapsto\mapsto_{C.tiny\beta}$ *cat-FUNCT* α $\mathfrak{A}$ $\mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
from *assms*(*1,2*) **show**

```
  exp-cat-cf α 𝔈 𝔉 : cat-FUNCT α 𝔄 𝔈 ↦↦_{C.tiny β} cat-FUNCT α 𝔄 𝔈
  by (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)

show ?exp-𝔑 : ?exp-𝔉 ↦_{CF} ?exp-𝔊 : cat-FUNCT α 𝔄 𝔈 ↦↦_{Cβ} cat-FUNCT α 𝔄 𝔈
proof(rule is-ntcfI')

  show vfsequence (?exp-𝔑) unfolding exp-cat-ntcf-def by auto
  show vcard (?exp-𝔑) = 5_ℕ
    unfolding exp-cat-ntcf-def by (simp add: nat-omega-simps)

  from assms(1,2) show
    exp-cat-cf α 𝔈 𝔊 : cat-FUNCT α 𝔄 𝔈 ↦↦_{Cβ} cat-FUNCT α 𝔄 𝔈
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  from assms(1,2) show
    exp-cat-cf α 𝔈 𝔉 : cat-FUNCT α 𝔄 𝔈 ↦↦_{Cβ} cat-FUNCT α 𝔄 𝔈
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  show exp-cat-ntcf α 𝔈 𝔑(|NTMap|)(|𝔎|) :
    exp-cat-cf α 𝔈 𝔉(|ObjMap|)(|𝔎|) ↦_{cat-FUNCT α 𝔄 𝔈}
    exp-cat-cf α 𝔈 𝔊(|ObjMap|)(|𝔎|)
    if 𝔎 ∈_∘ cat-FUNCT α 𝔄 𝔈(|Obj|) for 𝔎
  proof–
    from that[unfolded cat-FUNCT-cs-simps] have 𝔎 ∈_∘ cf-maps α 𝔄 𝔈 by simp
    then obtain 𝔎' where 𝔎-def: 𝔎 = cf-map 𝔎' and 𝔎': 𝔎' : 𝔄 ↦↦_{Cα} 𝔈
      by auto
    from 𝔎' show ?thesis
      unfolding 𝔎-def
      by
        (
          cs-concl
            cs-simp: cat-FUNCT-cs-simps 𝔎-def
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed

  show
    ?exp-𝔑(|NTMap|)(|𝔗|) ∘_{A cat-FUNCT α 𝔄 𝔈} ?exp-𝔉(|ArrMap|)(|𝔏|) =
      ?exp-𝔊(|ArrMap|)(|𝔏|) ∘_{A cat-FUNCT α 𝔄 𝔈} ?exp-𝔑(|NTMap|)(|𝔖|)
    if 𝔏 : 𝔖 ↦_{cat-FUNCT α 𝔄 𝔈} 𝔗 for 𝔖 𝔗 𝔏
  proof–
    note 𝔏 = cat-FUNCT-is-arrD[OF that]
    let ?𝔖 = ‹cf-of-cf-map 𝔄 𝔈 𝔖›
      and ?𝔗 = ‹cf-of-cf-map 𝔄 𝔈 𝔗›
      and ?𝔏 = ‹ntcf-of-ntcf-arrow 𝔄 𝔈 𝔏›
    have [cat-cs-simps]:
      (?𝔗 ∘_{CF-NTCF} 𝔑) ·_{NTCF} (?𝔏 ∘_{NTCF-CF} 𝔉) =
        (?𝔏 ∘_{NTCF-CF} 𝔊) ·_{NTCF} (?𝔖 ∘_{CF-NTCF} 𝔑)
    proof(rule ntcf-eqI)
      from 𝔏(1) show
```

```
      (?𝔗 ∘_CF−NTCF 𝔑) ·_NTCF (?𝔏 ∘_NTCF−CF 𝔉) :
        ?𝔖 ∘_CF 𝔉 ↦_CF ?𝔗 ∘_CF 𝔊 : 𝔄 ↦↦_Cα ℭ
      by (cs-concl cs-intro: cat-cs-intros)
    from 𝔏(1) show
      (?𝔏 ∘_NTCF−CF 𝔊) ·_NTCF (?𝔖 ∘_CF−NTCF 𝔑) :
        ?𝔖 ∘_CF 𝔉 ↦_CF ?𝔗 ∘_CF 𝔊 : 𝔄 ↦↦_Cα ℭ
      by (cs-concl cs-intro: cat-cs-intros)
    from 𝔏(1) have dom-lhs:
      𝒟∘ (((?𝔗 ∘_CF−NTCF 𝔑) ·_NTCF (?𝔏 ∘_NTCF−CF 𝔉))(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from 𝔏(1) have dom-rhs:
      𝒟∘ (((?𝔏 ∘_NTCF−CF 𝔊) ·_NTCF (?𝔖 ∘_CF−NTCF 𝔑))(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    show
      ((?𝔗 ∘_CF−NTCF 𝔑) ·_NTCF (?𝔏 ∘_NTCF−CF 𝔉))(⌊NTMap⌋) =
        ((?𝔏 ∘_NTCF−CF 𝔊) ·_NTCF (?𝔖 ∘_CF−NTCF 𝔑))(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈∘ 𝔄(⌊Obj⌋)
      with 𝔏(1) show
        ((?𝔗 ∘_CF−NTCF 𝔑) ·_NTCF (?𝔏 ∘_NTCF−CF 𝔉))(⌊NTMap⌋)(⌊a⌋) =
          ((?𝔏 ∘_NTCF−CF 𝔊) ·_NTCF (?𝔖 ∘_CF−NTCF 𝔑))(⌊NTMap⌋)(⌊a⌋)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps is-ntcf.ntcf-Comp-commute
              cs-intro: cat-cs-intros
          )
    qed (cs-concl cs-intro: cat-cs-intros)
  qed simp-all
  from 𝔏(1,3,4) that show ?thesis
    by (subst (1 2) 𝔏(2), use nothing in ‹subst 𝔏(3), subst 𝔏(4)›)
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
qed

qed
  (
    cs-concl
      cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
  )+

qed

lemma exp-cat-ntcf-is-tiny-ntcf'[cat-FUNCT-cs-intros]:
  assumes 𝒵 β
    and α ∈∘ β
    and 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
    and category α ℭ
    and 𝔉' = exp-cat-cf α ℭ 𝔉
    and 𝔊' = exp-cat-cf α ℭ 𝔊
    and 𝔄' = cat-FUNCT α 𝔅 ℭ
    and 𝔅' = cat-FUNCT α 𝔄 ℭ
  shows exp-cat-ntcf α ℭ 𝔑 : 𝔉' ↦_CF.tiny 𝔊' : 𝔄' ↦↦_C.tinyβ 𝔅'
  using assms(1–4) unfolding assms(5–8) by (rule exp-cat-ntcf-is-tiny-ntcf)
```

25.9.4 Further properties

lemma *ntcf-id-exp-cat-cf*:
assumes *category* α $\mathfrak{A}$ **and** $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *ntcf-id* (*exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{F}$) = *exp-cat-ntcf* α $\mathfrak{A}$ (*ntcf-id* $\mathfrak{F}$)
proof–

interpret $\mathfrak{A}$: *category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))

define β **where** $\beta = \alpha + \omega$
have β: $\mathcal{Z}$ β **and** $\alpha\beta$: $\alpha \in_{\circ} \beta$
by (*simp-all add*: β-*def* $\mathfrak{A}$.$\mathcal{Z}$-*Limit-*$\alpha\omega$ $\mathfrak{A}$.$\mathcal{Z}$-ω-$\alpha\omega$ $\mathcal{Z}$-*def* $\mathfrak{A}$.$\mathcal{Z}$-α-$\alpha\omega$)
then interpret β: $\mathcal{Z}$ β **by** *simp*

show *?thesis*
proof(*rule ntcf-eqI*)
from $\alpha\beta$ **show** *exp-cat-ntcf* α $\mathfrak{A}$ (*ntcf-id* $\mathfrak{F}$) :
exp-cat-cf α $\mathfrak{A}$ $\mathfrak{F}$ $\mapsto_{CF}$ *exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{F}$:
cat-FUNCT α $\mathfrak{C}$ $\mathfrak{A}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{B}$ $\mathfrak{A}$
by
(
cs-concl
cs-simp: *cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
from *assms* β $\alpha\beta$ **show** *ntcf-id* (*exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{F}$) :
exp-cat-cf α $\mathfrak{A}$ $\mathfrak{F}$ $\mapsto_{CF}$ *exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{F}$:
cat-FUNCT α $\mathfrak{C}$ $\mathfrak{A}$ $\mapsto\mapsto_{C\beta}$ *cat-FUNCT* α $\mathfrak{B}$ $\mathfrak{A}$
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
from $\alpha\beta$ *assms* **have** *dom-lhs*:
$\mathcal{D}_{\circ}$ (*exp-cat-ntcf* α $\mathfrak{A}$ (*ntcf-id* $\mathfrak{F}$)(|*NTMap*|)) = *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$(|*Obj*|)
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros*
)
from $\alpha\beta$ *assms* **have** *dom-rhs*:
$\mathcal{D}_{\circ}$ (*ntcf-id* (*exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{F}$)(|*NTMap*|)) = *cat-FUNCT* α $\mathfrak{C}$ $\mathfrak{A}$(|*Obj*|)
by
(
cs-concl
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros*
)
show
ntcf-id (*exp-cat-cf* α $\mathfrak{A}$ $\mathfrak{F}$)(|*NTMap*|) = *exp-cat-ntcf* α $\mathfrak{A}$ (*ntcf-id* $\mathfrak{F}$)(|*NTMap*|)
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs cat-FUNCT-components*(*1*))
fix $\mathfrak{G}$ **assume** $\mathfrak{G} \in_{\circ}$ *cf-maps* α $\mathfrak{C}$ $\mathfrak{A}$
then obtain $\mathfrak{G}'$
where $\mathfrak{G}$-*def*: $\mathfrak{G}$ = *cf-map* $\mathfrak{G}'$ **and** $\mathfrak{G}'$: $\mathfrak{G}' : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathfrak{A}$
by *auto*

```
    from 𝔊′ αβ show
      ntcf-id (exp-cat-cf α 𝔄 𝔉)(⌊NTMap⌋)(⌊𝔊⌋) =
        exp-cat-ntcf α 𝔄 (ntcf-id 𝔉)(⌊NTMap⌋)(⌊𝔊⌋)
      unfolding 𝔊-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
    (
      cs-concl
        cs-intro: αβ cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
    )+
qed simp-all

qed

lemma exp-cat-ntcf-ntcf-cf-comp:
  assumes 𝔑 : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα ℭ
    and 𝔎 : 𝔄 ↦↦Cα 𝔅
    and category α 𝔇
  shows
    exp-cat-ntcf α 𝔇 (𝔑 ∘NTCF-CF 𝔎) =
      exp-cat-cf α 𝔇 𝔎 ∘CF-NTCF exp-cat-ntcf α 𝔇 𝔑
proof–

  interpret 𝔑: is-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret 𝔎: is-functor α 𝔄 𝔅 𝔎 by (rule assms(2))
  interpret 𝔇: category α 𝔇 by (rule assms(3))

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔑.𝒵-Limit-αω 𝔑.𝒵-ω-αω 𝒵-def 𝔑.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)
    from αβ have dom-lhs:
      𝒟∘ (exp-cat-ntcf α 𝔇 (𝔑 ∘NTCF-CF 𝔎)(⌊NTMap⌋)) = cat-FUNCT α ℭ 𝔇(⌊Obj⌋)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ have dom-rhs:
      𝒟∘ ((exp-cat-cf α 𝔇 𝔎 ∘CF-NTCF exp-cat-ntcf α 𝔇 𝔑)(⌊NTMap⌋)) =
        cat-FUNCT α ℭ 𝔇(⌊Obj⌋)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )

    show
```

```
    exp-cat-ntcf α 𝔇 (𝔑 ∘_NTCF-CF ℌ)(|NTMap|) =
      (exp-cat-cf α 𝔇 ℌ ∘_CF-NTCF exp-cat-ntcf α 𝔇 𝔑)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
    fix 𝔎 assume prems: 𝔎 ∈₀ cf-maps α ℭ 𝔇
    then obtain 𝔎' where 𝔎-def: 𝔎 = cf-map 𝔎' and 𝔎': 𝔎' : ℭ ↦↦_Cα 𝔇
      by (auto intro: is-arrI)
    from αβ assms prems 𝔎' show
      exp-cat-ntcf α 𝔇 (𝔑 ∘_NTCF-CF ℌ)(|NTMap|)(|𝔎|) =
        (exp-cat-cf α 𝔇 ℌ ∘_CF-NTCF exp-cat-ntcf α 𝔇 𝔑)(|NTMap|)(|𝔎|)
      unfolding 𝔎-def
      by
        (
          cs-concl
            cs-simp:
              cf-ntcf-comp-ntcf-cf-comp-assoc cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)
qed
  (
    cs-concl
      cs-simp: exp-cat-cf-cf-comp cat-cs-simps cat-FUNCT-cs-simps
      cs-intro: αβ cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )+

qed

lemma exp-cat-ntcf-cf-ntcf-comp:
  assumes 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔅
    and ℌ : 𝔅 ↦↦_Cα ℭ
    and category α 𝔇
  shows
    exp-cat-ntcf α 𝔇 (ℌ ∘_CF-NTCF 𝔑) =
      exp-cat-ntcf α 𝔇 𝔑 ∘_NTCF-CF exp-cat-cf α 𝔇 ℌ
proof–

  interpret 𝔑: is-ntcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret ℌ: is-functor α 𝔅 ℭ ℌ by (rule assms(2))
  interpret 𝔇: category α 𝔇 by (rule assms(3))

  define β where β = α + ω
  have 𝒵 β and αβ: α ∈₀ β
    by (simp-all add: β-def 𝔑.𝒵-Limit-αω 𝔑.𝒵-ω-αω 𝒵-def 𝔑.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)

    from αβ have dom-lhs:
      𝒟₀ (exp-cat-ntcf α 𝔇 (ℌ ∘_CF-NTCF 𝔑)(|NTMap|)) = cat-FUNCT α ℭ 𝔇(|Obj|)
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
    from αβ have dom-rhs:
      𝒟₀ ((exp-cat-ntcf α 𝔇 𝔑 ∘_NTCF-CF exp-cat-cf α 𝔇 ℌ)(|NTMap|)) =
```

```
      cat-FUNCT α ℭ 𝔇(|Obj|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  show
    exp-cat-ntcf α 𝔇 (ℌ ∘CF−NTCF 𝔑)(|NTMap|) =
      (exp-cat-ntcf α 𝔇 𝔑 ∘NTCF−CF exp-cat-cf α 𝔇 ℌ)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
    fix 𝔎 assume prems: 𝔎 ∈∘ cf-maps α ℭ 𝔇
    then obtain 𝔎′ where 𝔎-def: 𝔎 = cf-map 𝔎′ and 𝔎′: 𝔎′ : ℭ ↦↦Cα 𝔇
      by (auto intro: is-arrI)
    from assms αβ prems 𝔎′ show
      exp-cat-ntcf α 𝔇 (ℌ ∘CF−NTCF 𝔑)(|NTMap|)(|𝔎|) =
        (exp-cat-ntcf α 𝔇 𝔑 ∘NTCF−CF exp-cat-cf α 𝔇 ℌ)(|NTMap|)(|𝔎|)
      by
        (
          cs-concl
            cs-simp:
              cf-comp-cf-ntcf-comp-assoc cat-cs-simps cat-FUNCT-cs-simps 𝔎-def
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)

qed
  (
    cs-concl
      cs-simp: exp-cat-cf-cf-comp cat-cs-simps cat-FUNCT-cs-simps
      cs-intro: αβ cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
  )+

qed

lemma exp-cat-ntcf-ntcf-vcomp:
  assumes category α 𝔄
    and 𝔐 : 𝔊 ↦CF ℌ : 𝔅 ↦↦Cα ℭ
    and 𝔑 : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα ℭ
  shows
    exp-cat-ntcf α 𝔄 (𝔐 ∙NTCF 𝔑) =
      exp-cat-ntcf α 𝔄 𝔐 ∙NTCF exp-cat-ntcf α 𝔄 𝔑
proof–

  interpret 𝔄: category α 𝔄 by (rule assms(1))
  interpret 𝔐: is-ntcf α 𝔅 ℭ 𝔊 ℌ 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntcf α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(3))

  define β where β = α + ω
  have β: 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝔄.𝒵-Limit-αω 𝔄.𝒵-ω-αω 𝒵-def 𝔄.𝒵-α-αω)
  then interpret β: 𝒵 β by simp

  show ?thesis
  proof(rule ntcf-eqI)
    from β αβ show
      exp-cat-ntcf α 𝔄 (𝔐 ∙NTCF 𝔑) :
```

```
      exp-cat-cf α 𝔄 𝔉 ↦_CF exp-cat-cf α 𝔄 𝔊 :
      cat-FUNCT α ℭ 𝔄 ↦↦_Cβ cat-FUNCT α 𝔅 𝔄
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from αβ show
    exp-cat-ntcf α 𝔄 𝔐 ∘_NTCF exp-cat-ntcf α 𝔄 𝔑 :
      exp-cat-cf α 𝔄 𝔉 ↦_CF exp-cat-cf α 𝔄 𝔊 :
      cat-FUNCT α ℭ 𝔄 ↦↦_Cβ cat-FUNCT α 𝔅 𝔄
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from αβ have dom-lhs:
    𝒟∘ ((exp-cat-ntcf α 𝔄 (𝔐 ∘_NTCF 𝔑))(⌊NTMap⌋)) = cat-FUNCT α ℭ 𝔄(⌊Obj⌋)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from αβ have dom-rhs:
    𝒟∘ ((exp-cat-ntcf α 𝔄 𝔐 ∘_NTCF exp-cat-ntcf α 𝔄 𝔑)(⌊NTMap⌋)) =
      cat-FUNCT α ℭ 𝔄(⌊Obj⌋)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )

  show
    exp-cat-ntcf α 𝔄 (𝔐 ∘_NTCF 𝔑)(⌊NTMap⌋) =
      (exp-cat-ntcf α 𝔄 𝔐 ∘_NTCF exp-cat-ntcf α 𝔄 𝔑)(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs cat-FUNCT-components(1))
    fix 𝔉′ assume 𝔉′ ∈∘ cf-maps α ℭ 𝔄
    then obtain 𝔉″
      where 𝔉′-def: 𝔉′ = cf-map 𝔉″ and 𝔉″: 𝔉″ : ℭ ↦↦_Cα 𝔄
      by clarsimp
    from 𝔉″ αβ show
      exp-cat-ntcf α 𝔄 (𝔐 ∘_NTCF 𝔑)(⌊NTMap⌋)(⌊𝔉′⌋) =
        (exp-cat-ntcf α 𝔄 𝔐 ∘_NTCF exp-cat-ntcf α 𝔄 𝔑)(⌊NTMap⌋)(⌊𝔉′⌋)
      by
        (
          cs-concl
            cs-simp:
              cat-cs-simps cat-FUNCT-cs-simps cf-ntcf-comp-ntcf-vcomp 𝔉′-def
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        )
  qed (cs-concl cs-intro: cat-cs-intros cat-FUNCT-cs-intros)+

qed simp-all

qed
```

26 *Hom*-functor

26.1 *hom*-function

The *hom*-function is a part of the definition of the *Hom*-functor, as presented in [1][13].

definition *cf-hom* :: $V \Rightarrow V \Rightarrow V$
  where *cf-hom* $\mathfrak{C}$ f =
    [
      (
        $\lambda q \in_{\circ} Hom\ \mathfrak{C}\ (\mathfrak{C}(\!|Cod|\!)(\!|vpfst\ f|\!))\ (\mathfrak{C}(\!|Dom|\!)(\!|vpsnd\ f|\!)).$
          $vpsnd\ f \circ_{A\mathfrak{C}} q \circ_{A\mathfrak{C}} vpfst\ f$
      ),
      $Hom\ \mathfrak{C}\ (\mathfrak{C}(\!|Cod|\!)(\!|vpfst\ f|\!))\ (\mathfrak{C}(\!|Dom|\!)(\!|vpsnd\ f|\!))$,
      $Hom\ \mathfrak{C}\ (\mathfrak{C}(\!|Dom|\!)(\!|vpfst\ f|\!))\ (\mathfrak{C}(\!|Cod|\!)(\!|vpsnd\ f|\!))$
    ]$_{\circ}$

Components.

lemma *cf-hom-components*:
  shows *cf-hom* $\mathfrak{C}$ $f(\!|ArrVal|\!)$ =
      (
        $\lambda q \in_{\circ} Hom\ \mathfrak{C}\ (\mathfrak{C}(\!|Cod|\!)(\!|vpfst\ f|\!))\ (\mathfrak{C}(\!|Dom|\!)(\!|vpsnd\ f|\!)).$
          $vpsnd\ f \circ_{A\mathfrak{C}} q \circ_{A\mathfrak{C}} vpfst\ f$
)
    and *cf-hom* $\mathfrak{C}$ $f(\!|ArrDom|\!)$ = $Hom\ \mathfrak{C}\ (\mathfrak{C}(\!|Cod|\!)(\!|vpfst\ f|\!))\ (\mathfrak{C}(\!|Dom|\!)(\!|vpsnd\ f|\!))$
    and *cf-hom* $\mathfrak{C}$ $f(\!|ArrCod|\!)$ = $Hom\ \mathfrak{C}\ (\mathfrak{C}(\!|Dom|\!)(\!|vpfst\ f|\!))\ (\mathfrak{C}(\!|Cod|\!)(\!|vpsnd\ f|\!))$
  unfolding *cf-hom-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

26.1.1 Arrow value

mk-VLambda *cf-hom-components*(*1*)
  |*vsv cf-hom-ArrVal-vsv*[*cat-cs-intros*]|

lemma *cf-hom-ArrVal-vdomain*[*cat-cs-simps*]:
  assumes $g : a \mapsto_{op\text{-}cat\ \mathfrak{C}}\ b$ **and** $f : a' \mapsto_{\mathfrak{C}}\ b'$
  shows $\mathcal{D}_{\circ}$ (*cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)$) = $Hom\ \mathfrak{C}\ a\ a'$
  using *assms*
  unfolding *cf-hom-components*
  by (*simp-all add*: *nat-omega-simps cat-op-simps cat-cs-simps*)

lemma *cf-hom-ArrVal-app*[*cat-cs-simps*]:
  assumes $g : c \mapsto_{op\text{-}cat\ \mathfrak{C}}\ d$ **and** $q : c \mapsto_{\mathfrak{C}}\ c'$ **and** $f : c' \mapsto_{\mathfrak{C}}\ d'$
  shows *cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)(\!|q|\!) = f \circ_{A\mathfrak{C}} q \circ_{A\mathfrak{C}} g$
  using *assms*
  unfolding *cf-hom-components*
  by (*simp-all add*: *nat-omega-simps cat-op-simps cat-cs-simps*)

lemma (**in** *category*) *cf-hom-ArrVal-vrange*:
  assumes $g : a \mapsto_{op\text{-}cat\ \mathfrak{C}}\ b$ **and** $f : a' \mapsto_{\mathfrak{C}}\ b'$
  shows $\mathcal{R}_{\circ}$ (*cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)$) $\subseteq_{\circ}$ $Hom\ \mathfrak{C}\ b\ b'$
proof(*intro vsubsetI*)
  interpret *gf*: *vsv* ‹*cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)$›
    unfolding *cf-hom-components* **by** *auto*
  fix *y* **assume** $y \in_{\circ} \mathcal{R}_{\circ}$ (*cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)$)
  then obtain *q* **where** *y-def*: y = *cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)(\!|q|\!)$
    and $q \in_{\circ} \mathcal{D}_{\circ}$ (*cf-hom* $\mathfrak{C}$ $[g, f]_{\circ}(\!|ArrVal|\!)$)
    by (*metis gf.vrange-atD*)

[13] https://ncatlab.org/nlab/show/hom-functor

then have *q*: $q : a \mapsto_{\mathfrak{C}} a'$
unfolding *cf-hom-ArrVal-vdomain*[*OF assms*] **by** *simp*
from *assms q* **show** $y \in_{\circ} Hom\ \mathfrak{C}\ b\ b'$
unfolding *y-def cf-hom-ArrVal-app*[*OF assms*(*1*) *q assms*(*2*)] *cat-op-simps*
by (*auto intro*: *cat-cs-intros*)
qed

26.1.2 Arrow domain

lemma (**in** *category*) *cf-hom-ArrDom*:
assumes $gf : [c,\ c']_{\circ} \mapsto_{op\text{-}cat} \mathfrak{C} \times_C \mathfrak{C}\ dd'$
shows $cf\text{-}hom\ \mathfrak{C}\ gf(\lfloor ArrDom \rfloor) = Hom\ \mathfrak{C}\ c\ c'$
proof–
from *assms* **obtain** *g f d d'*
where $gf = [g,\ f]_{\circ}$ **and** $g : c \mapsto_{op\text{-}cat} \mathfrak{C}\ d$ **and** $f : c' \mapsto_{\mathfrak{C}} d'$
unfolding *cf-hom-components*
by (*elim cat-prod-2-is-arrE*[*rotated 2*]) (*auto intro*: *cat-cs-intros*)
then show *?thesis*
unfolding *cf-hom-components*
by (*simp-all add*: *nat-omega-simps cat-op-simps cat-cs-simps*)
qed

lemmas [*cat-cs-simps*] = *category.cf-hom-ArrDom*

26.1.3 Arrow codomain

lemma (**in** *category*) *cf-hom-ArrCod*:
assumes $gf : cc' \mapsto_{op\text{-}cat} \mathfrak{C} \times_C \mathfrak{C}\ [d,\ d']_{\circ}$
shows $cf\text{-}hom\ \mathfrak{C}\ gf(\lfloor ArrCod \rfloor) = Hom\ \mathfrak{C}\ d\ d'$
proof–
from *assms* **obtain** *g f c c'*
where $gf = [g,\ f]_{\circ}$ **and** $g : c \mapsto_{op\text{-}cat} \mathfrak{C}\ d$ **and** $f : c' \mapsto_{\mathfrak{C}} d'$
unfolding *cf-hom-components*
by (*elim cat-prod-2-is-arrE*[*rotated 2*]) (*auto intro*: *cat-cs-intros*)
then show *?thesis*
unfolding *cf-hom-components*
by (*simp-all add*: *nat-omega-simps cat-op-simps cat-cs-simps*)
qed

lemmas [*cat-cs-simps*] = *category.cf-hom-ArrCod*

26.1.4 *hom*-function is an arrow in the category *Set*

lemma (**in** *category*) *cat-cf-hom-ArrRel*:
assumes $gf : cc' \mapsto_{op\text{-}cat} \mathfrak{C} \times_C \mathfrak{C}\ dd'$
shows $arr\text{-}Set\ \alpha\ (cf\text{-}hom\ \mathfrak{C}\ gf)$
proof(*intro arr-SetI*)
from *assms* **obtain** *g f c c' d d'*
where *gf-def*: $gf = [g,\ f]_{\circ}$
and *cc'-def*: $cc' = [c,\ c']_{\circ}$
and *dd'-def*: $dd' = [d,\ d']_{\circ}$
and *op-g*: $g : c \mapsto_{op\text{-}cat} \mathfrak{C}\ d$
and *f*: $f : c' \mapsto_{\mathfrak{C}} d'$
unfolding *cf-hom-components*
by (*elim cat-prod-2-is-arrE*[*rotated 2*]) (*auto intro*: *cat-cs-intros*)
from *op-g* **have** *g*: $g : d \mapsto_{\mathfrak{C}} c$ **unfolding** *cat-op-simps* **by** *simp*
then have [*simp*]: $\mathfrak{C}(\lfloor Dom \rfloor)(\lfloor g \rfloor) = d\ \mathfrak{C}(\lfloor Cod \rfloor)(\lfloor g \rfloor) = c$
and *d*: $d \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$ **and** *c*: $c \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$

```
    by auto
  from f have [simp]: ℭ(|Dom|)(|f|) = c' ℭ(|Cod|)(|f|) = d'
    and c': c' ∈∘ ℭ(|Obj|) and d': d' ∈∘ ℭ(|Obj|)
    by auto
  show vfsequence (cf-hom ℭ gf) unfolding cf-hom-def by simp
  show vsv-hom-fg: vsv (cf-hom ℭ gf(|ArrVal|))
    unfolding cf-hom-components by auto
  show vcard (cf-hom ℭ gf) = 3_ℕ
    unfolding cf-hom-def by (simp add: nat-omega-simps)
  show [simp]: 𝒟∘ (cf-hom ℭ gf(|ArrVal|)) = cf-hom ℭ gf(|ArrDom|)
    unfolding cf-hom-components by auto
  show ℛ∘ (cf-hom ℭ gf(|ArrVal|)) ⊆∘ cf-hom ℭ gf(|ArrCod|)
  proof(rule vsubsetI)
    interpret hom-fg: vsv ‹cf-hom ℭ gf(|ArrVal|)› by (simp add: vsv-hom-fg)
    fix y assume y ∈∘ ℛ∘ (cf-hom ℭ gf(|ArrVal|))
    then obtain q where y-def: y = cf-hom ℭ gf(|ArrVal|)(|q|)
      and q: q ∈∘ 𝒟∘ (cf-hom ℭ gf(|ArrVal|))
      by (blast dest: hom-fg.vrange-atD)
    from q have q: q : c ↦_ℭ c'
      by (simp add: cf-hom-ArrDom[OF assms[unfolded cc'-def]])
    with g f have f ∘_{A ℭ} q ∘_{A ℭ} g : d ↦_ℭ d'
      by (auto intro: cat-cs-intros)
    then show y ∈∘ cf-hom ℭ gf(|ArrCod|)
      unfolding cf-hom-ArrCod[OF assms[unfolded dd'-def]]
      unfolding y-def gf-def cf-hom-ArrVal-app[OF op-g q f]
      by auto
  qed
  from c c' show cf-hom ℭ gf(|ArrDom|) ∈∘ Vset α
    unfolding cf-hom-components gf-def
    by (auto simp: nat-omega-simps intro: cat-cs-intros)
  from d d' show cf-hom ℭ gf(|ArrCod|) ∈∘ Vset α
    unfolding cf-hom-components gf-def
    by (auto simp: nat-omega-simps intro: cat-cs-intros)
qed auto

lemmas [cat-cs-intros] = category.cat-cf-hom-ArrRel

lemma (in category) cat-cf-hom-cat-Set-is-arr:
  assumes gf : [a, b]∘ ↦_{op-cat ℭ ×_C ℭ} [c, d]∘
  shows cf-hom ℭ gf : Hom ℭ a b ↦_{cat-Set α} Hom ℭ c d
proof(intro is-arrI)
  from assms cat-cf-hom-ArrRel show cf-hom ℭ gf ∈∘ cat-Set α(|Arr|)
    unfolding cat-Set-components by auto
  with assms show
    cat-Set α(|Dom|)(|cf-hom ℭ gf|) = Hom ℭ a b
    cat-Set α(|Cod|)(|cf-hom ℭ gf|) = Hom ℭ c d
    unfolding cat-Set-components
    by (simp-all add: cf-hom-ArrDom[OF assms] cf-hom-ArrCod[OF assms])
qed

lemma (in category) cat-cf-hom-cat-Set-is-arr':
  assumes gf : [a, b]∘ ↦_{op-cat ℭ ×_C ℭ} [c, d]∘
    and 𝔄' = Hom ℭ a b
    and 𝔅' = Hom ℭ c d
    and ℭ' = cat-Set α
  shows cf-hom ℭ gf : 𝔄' ↦_{ℭ'} 𝔅'
  using assms(1) unfolding assms(2–4) by (rule cat-cf-hom-cat-Set-is-arr)
```

lemmas [*cat-cs-intros*] = *category.cat-cf-hom-cat-Set-is-arr′*

26.1.5 Composition

lemma (**in** *category*) *cat-cf-hom-Comp*:
assumes $g : b \mapsto_{op\text{-}cat\ \mathfrak{C}}\ c$
and $g' : b' \mapsto_{\mathfrak{C}}\ c'$
and $f : a \mapsto_{op\text{-}cat\ \mathfrak{C}}\ b$
and $f' : a' \mapsto_{\mathfrak{C}}\ b'$
shows
$\mathfrak{C}\text{-}hom\ [g,\ g']_{\circ} \circ_{A\ cat\text{-}Set\ \alpha}\ \mathfrak{C}\text{-}hom\ [f,\ f']_{\circ} =$
$\mathfrak{C}\text{-}hom\ [g \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ f,\ g' \circ_{A\ \mathfrak{C}}\ f']_{\circ}$
proof–

interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)

from *assms*(*1*,*3*) **have** *g*: $g : c \mapsto_{\mathfrak{C}}\ b$ **and** *f*: $f : b \mapsto_{\mathfrak{C}}\ a$
unfolding *cat-op-simps* **by** *simp-all*

from *assms*(*2*,*4*) *g f Set.category-axioms category-axioms* **have** *gg′-ff′*:
$\mathfrak{C}\text{-}hom\ [g,\ g']_{\circ} \circ_{A\ cat\text{-}Set\ \alpha}\ \mathfrak{C}\text{-}hom\ [f,\ f']_{\circ} :$
$Hom\ \mathfrak{C}\ a\ a' \mapsto_{cat\text{-}Set\ \alpha}\ Hom\ \mathfrak{C}\ c\ c'$
by
(
cs-concl **cs-shallow**
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
then have *dom-lhs*:
$\mathcal{D}_{\circ}\ ((\mathfrak{C}\text{-}hom\ [g,\ g']_{\circ} \circ_{A\ cat\text{-}Set\ \alpha}\ \mathfrak{C}\text{-}hom\ [f,\ f']_{\circ})(\!|ArrVal|\!)) =$
$Hom\ \mathfrak{C}\ a\ a'$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)+
from *assms*(*2*,*4*) *g f Set.category-axioms category-axioms* **have** *gf-g′f′*:
$\mathfrak{C}\text{-}hom\ [g \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ f,\ g' \circ_{A\ \mathfrak{C}}\ f']_{\circ} :$
$Hom\ \mathfrak{C}\ a\ a' \mapsto_{cat\text{-}Set\ \alpha}\ Hom\ \mathfrak{C}\ c\ c'$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros cat-prod-cs-intros*)
then have *dom-rhs*:
$\mathcal{D}_{\circ}\ (\mathfrak{C}\text{-}hom\ [g \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ f,\ g' \circ_{A\ \mathfrak{C}}\ f']_{\circ}(\!|ArrVal|\!)) = Hom\ \mathfrak{C}\ a\ a'$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)

show *?thesis*
proof(*rule arr-Set-eqI*[*of* α])

from *gg′-ff′* **show** *arr-Set-gg′-ff′*:
arr-Set α ($\mathfrak{C}\text{-}hom\ [g,\ g']_{\circ} \circ_{A\ cat\text{-}Set\ \alpha}\ \mathfrak{C}\text{-}hom\ [f,\ f']_{\circ}$)
by (*auto dest*: *cat-Set-is-arrD*(*1*))
from *gf-g′f′* **show** *arr-Set-gf-g′f′*:
arr-Set α ($\mathfrak{C}\text{-}hom\ [g \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ f,\ g' \circ_{A\ \mathfrak{C}}\ f']_{\circ}$)
by (*auto dest*: *cat-Set-is-arrD*(*1*))

show ($\mathfrak{C}\text{-}hom\ [g,\ g']_{\circ} \circ_{A\ cat\text{-}Set\ \alpha}\ \mathfrak{C}\text{-}hom\ [f,\ f']_{\circ}$)(|*ArrVal*|) =
$\mathfrak{C}\text{-}hom\ [g \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ f,\ g' \circ_{A\ \mathfrak{C}}\ f']_{\circ}$(|*ArrVal*|)
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *q* **assume** $q \in_{\circ} Hom\ \mathfrak{C}\ a\ a'$
then have *q*: $q : a \mapsto_{\mathfrak{C}}\ a'$ **by** *auto*
from *category-axioms g f assms*(*2*,*4*) *q Set.category-axioms* **show**
($\mathfrak{C}\text{-}hom\ [g,\ g']_{\circ} \circ_{A\ cat\text{-}Set\ \alpha}\ \mathfrak{C}\text{-}hom\ [f,\ f']_{\circ}$)(|*ArrVal*|)(|*q*|) =
$\mathfrak{C}\text{-}hom\ [g \circ_{A\ op\text{-}cat\ \mathfrak{C}}\ f,\ g' \circ_{A\ \mathfrak{C}}\ f']_{\circ}$(|*ArrVal*|)(|*q*|)
by

```
        (
          cs-concl
            cs-intro: cat-op-intros cat-cs-intros cat-prod-cs-intros
            cs-simp: cat-op-simps cat-cs-simps
        )
    qed (use arr-Set-gg'-ff' arr-Set-gf-g'f' in auto)

  qed (use gg'-ff' gf-g'f' in ‹cs-concl cs-simp: cat-cs-simps›)+

qed

lemmas [cat-cs-simps] = category.cat-cf-hom-Comp
```

26.1.6 Identity

```
lemma (in category) cat-cf-hom-CId:
  assumes [c, c']∘ ∈∘ (op-cat 𝔄 ×C 𝔄)(|Obj|)
  shows cf-hom 𝔄 [𝔄(|CId|)(|c|), 𝔄(|CId|)(|c'|)]∘ = cat-Set α(|CId|)(|Hom 𝔄 c c'|)
proof–

  interpret Set: category α ‹cat-Set α› by (rule category-cat-Set)
  interpret op-𝔄: category α ‹op-cat 𝔄› by (rule category-op)

  from assms have op-c: c ∈∘ op-cat 𝔄(|Obj|) and c': c' ∈∘ 𝔄(|Obj|)
    by (auto elim: cat-prod-2-ObjE[rotated 2] intro: cat-cs-intros)
  then have c: c ∈∘ 𝔄(|Obj|) unfolding cat-op-simps by simp

  from c c' category-axioms Set.category-axioms have cf-hom-cc':
    cf-hom 𝔄 [𝔄(|CId|)(|c|), 𝔄(|CId|)(|c'|)]∘ : Hom 𝔄 c c' ↦_cat-Set α Hom 𝔄 c c'
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  then have dom-lhs: 𝒟∘ (cf-hom 𝔄 [𝔄(|CId|)(|c|), 𝔄(|CId|)(|c'|)]∘(|ArrVal|)) = Hom 𝔄 c c'
    by (cs-concl cs-simp: cat-cs-simps)
  from c c' category-axioms Set.category-axioms have CId-cc':
    cat-Set α(|CId|)(|Hom 𝔄 c c'|) : Hom 𝔄 c c' ↦_cat-Set α Hom 𝔄 c c'
    by
      (
        cs-concl
          cs-simp: cat-Set-cs-simps cat-Set-components(1)
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  then have dom-rhs: 𝒟∘ (cat-Set α(|CId|)(|Hom 𝔄 c c'|)(|ArrVal|)) = Hom 𝔄 c c'
    by (cs-concl cs-shallow cs-simp: cat-cs-simps )

  show ?thesis
  proof(rule arr-Set-eqI[of α])
    from cf-hom-cc' show arr-Set-CId-cc':
      arr-Set α (cf-hom 𝔄 [𝔄(|CId|)(|c|), 𝔄(|CId|)(|c'|)]∘)
      by (auto dest: cat-Set-is-arrD(1))
    from CId-cc' show arr-Set-Hom-cc':
      arr-Set α (cat-Set α(|CId|)(|Hom 𝔄 c c'|))
      by (auto simp: cat-Set-is-arrD(1))
    show cf-hom 𝔄 [𝔄(|CId|)(|c|), 𝔄(|CId|)(|c'|)]∘(|ArrVal|) =
      cat-Set α(|CId|)(|Hom 𝔄 c c'|)(|ArrVal|)
```

proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs in-Hom-iff*)
fix q **assume** $q : c \mapsto_{\mathfrak{C}} c'$
with *category-axioms* **show**
cf-hom $\mathfrak{C}$ $[\mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor), \mathfrak{C}(\lfloor CId \rfloor)(\lfloor c' \rfloor)]_{\circ}(\lfloor ArrVal \rfloor)(\lfloor q \rfloor)$ =
cat-Set $\alpha(\lfloor CId \rfloor)(\lfloor Hom\ \mathfrak{C}\ c\ c' \rfloor)(\lfloor ArrVal \rfloor)(\lfloor q \rfloor)$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps cat-Set-cs-simps*
cs-intro: *cat-cs-intros*
)
qed (*use arr-Set-CId-cc′ arr-Set-Hom-cc′* **in** *auto*)

qed (*use cf-hom-cc′ CId-cc′* **in** ‹*cs-concl cs-simp*: *cat-cs-simps*›)+

qed

lemmas [*cat-cs-simps*] = *category.cat-cf-hom-CId*

26.1.7 Opposite *hom*-function

lemma (**in** *category*) *cat-op-cat-cf-hom*:
assumes $g : a \mapsto_{\mathfrak{C}} b$ **and** $g' : a' \mapsto_{op\text{-}cat\ \mathfrak{C}} b'$
shows *cf-hom* (*op-cat* $\mathfrak{C}$) $[g, g']_{\circ}$ = *cf-hom* $\mathfrak{C}$ $[g', g]_{\circ}$
proof(*rule arr-Set-eqI*[*of* α])
from *assms* **show** *arr-Set* α (*cf-hom* (*op-cat* $\mathfrak{C}$) $[g, g']_{\circ}$)
by
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-prod-cs-intros*
)
from *assms* **show** *arr-Set* α (*cf-hom* $\mathfrak{C}$ $[g', g]_{\circ}$)
by
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
from *assms* **have** *dom-lhs*:
$\mathcal{D}_{\circ}$ (*cf-hom* (*op-cat* $\mathfrak{C}$) $[g, g']_{\circ}(\lfloor ArrVal \rfloor)$) = *Hom* $\mathfrak{C}$ a' a
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)
from *assms* **have** *dom-rhs*: $\mathcal{D}_{\circ}$ (*cf-hom* $\mathfrak{C}$ $[g', g]_{\circ}(\lfloor ArrVal \rfloor)$) = *Hom* $\mathfrak{C}$ a' a
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)
show *cf-hom* (*op-cat* $\mathfrak{C}$) $[g, g']_{\circ}(\lfloor ArrVal \rfloor)$ = *cf-hom* $\mathfrak{C}$ $[g', g]_{\circ}(\lfloor ArrVal \rfloor)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs in-Hom-iff*)
fix f **assume** $f : a' \mapsto_{\mathfrak{C}} a$
with *assms* **show**
cf-hom (*op-cat* $\mathfrak{C}$) $[g, g']_{\circ}(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$ = *cf-hom* $\mathfrak{C}$ $[g', g]_{\circ}(\lfloor ArrVal \rfloor)(\lfloor f \rfloor)$
unfolding *cat-op-simps*
by

```
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
      )
  qed (simp-all add: cf-hom-components)
  from category-axioms assms show
    cf-hom (op-cat ℭ) [g, g']∘(|ArrDom|) = cf-hom ℭ [g', g]∘(|ArrDom|)
    by
      (
        cs-concl cs-shallow
          cs-simp: category.cf-hom-ArrDom cat-op-simps
          cs-intro: cat-op-intros cat-prod-cs-intros
      )
  from category-axioms assms show
    cf-hom (op-cat ℭ) [g, g']∘(|ArrCod|) = cf-hom ℭ [g', g]∘(|ArrCod|)
    by
      (
        cs-concl cs-shallow
          cs-simp: category.cf-hom-ArrCod cat-op-simps
          cs-intro: cat-op-intros cat-prod-cs-intros
      )
qed

lemmas [cat-cs-simps] = category.cat-op-cat-cf-hom
```

26.2 *Hom*-functor

26.2.1 Definition and elementary properties

See [1][14].

```
definition cf-Hom :: V ⇒ V ⇒ V (‹Hom_O.Cα-'(/-,-/')›)
  where Hom_O.Cα ℭ(-,-) =
    [
      (λa∈∘(op-cat ℭ ×_C ℭ)(|Obj|). Hom ℭ (vpfst a) (vpsnd a)),
      (λf∈∘(op-cat ℭ ×_C ℭ)(|Arr|). cf-hom ℭ f),
      op-cat ℭ ×_C ℭ,
      cat-Set α
    ]∘
```

Components.

```
lemma cf-Hom-components:
  shows Hom_O.Cα ℭ(-,-)(|ObjMap|) =
    (λa∈∘(op-cat ℭ ×_C ℭ)(|Obj|). Hom ℭ (vpfst a) (vpsnd a))
    and Hom_O.Cα ℭ(-,-)(|ArrMap|) = (λf∈∘(op-cat ℭ ×_C ℭ)(|Arr|). cf-hom ℭ f)
    and Hom_O.Cα ℭ(-,-)(|HomDom|) = op-cat ℭ ×_C ℭ
    and Hom_O.Cα ℭ(-,-)(|HomCod|) = cat-Set α
  unfolding cf-Hom-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

26.2.2 Object map

```
mk-VLambda cf-Hom-components(1)
  |vsv cf-Hom-ObjMap-vsv|

lemma cf-Hom-ObjMap-vdomain[cat-cs-simps]:
  𝒟∘ (Hom_O.Cα ℭ(-,-)(|ObjMap|)) = (op-cat ℭ ×_C ℭ)(|Obj|)
  unfolding cf-Hom-components by simp
```

[14] https://ncatlab.org/nlab/show/hom-functor

lemma *cf-Hom-ObjMap-app*[*cat-cs-simps*]:
  assumes [*a*, *b*]$_\circ$ $\in_\circ$ (*op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$)(|*Obj*|)
  shows $Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|)(|*a*, *b*|)$_\bullet$ = *Hom* $\mathfrak{C}$ *a b*
  using *assms* **unfolding** *cf-Hom-components* **by** (*simp add*: *nat-omega-simps*)

lemma (**in** *category*) *cf-Hom-ObjMap-vrange*:
  $\mathcal{R}_\circ$ ($Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|)) $\subseteq_\circ$ *cat-Set* α(|*Obj*|)
proof(*intro vsubsetI*)
  interpret *op-*$\mathfrak{C}$: *category* α ‹*op-cat* $\mathfrak{C}$› **by** (*simp add*: *category-op*)
  fix *y* **assume** *y* $\in_\circ$ $\mathcal{R}_\circ$ ($Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|))
  then obtain *x* **where** *y-def*: *y* = $Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|)(|*x*|)
    and *x*: *x* $\in_\circ$ (*op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$)(|*Obj*|)
    unfolding *cf-Hom-components* **by** *auto*
  then obtain *a b* **where** *x-def*: *x* = [*a*, *b*]$_\circ$
    and *a*: *a* $\in_\circ$ *op-cat* $\mathfrak{C}$(|*Obj*|)
    and *b*: *b* $\in_\circ$ $\mathfrak{C}$(|*Obj*|)
    by (*elim cat-prod-2-ObjE*[*OF op-*$\mathfrak{C}$.*category-axioms category-axioms x*])
  from *a* **have** *a*: *a* $\in_\circ$ $\mathfrak{C}$(|*Obj*|) **unfolding** *cat-op-simps* **by** *simp*
  from *a b* **show** *y* $\in_\circ$ *cat-Set* α(|*Obj*|)
    unfolding
      y-def x-def cf-Hom-ObjMap-app[*OF x*[*unfolded x-def*]] *cat-Set-components*
    by (*auto simp*: *cat-cs-intros*)
qed

26.2.3 Arrow map

mk-VLambda *cf-Hom-components*(*2*)
  |*vsv cf-Hom-ArrMap-vsv*|
  |*vdomain cf-Hom-ArrMap-vdomain*[*cat-cs-simps*]|
  |*app cf-Hom-ArrMap-app*[*cat-cs-simps*]|

26.2.4 *Hom*-functor is a functor

lemma (**in** *category*) *cat-Hom-is-functor*:
  $Hom_{O.C\alpha}\mathfrak{C}(-,-)$: *op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof–

  interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)
  interpret $\mathfrak{C}\mathfrak{C}$: *category* α ‹*op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$›
    by (*simp add*: *category-axioms category-cat-prod-2 category-op*)
  interpret *op-*$\mathfrak{C}$: *category* α ‹*op-cat* $\mathfrak{C}$› **by** (*rule category-op*)

  show *?thesis*
  proof(*intro is-functorI'*)

    show *vfsequence* $Hom_{O.C\alpha}\mathfrak{C}(-,-)$
      unfolding *cf-Hom-def* **by** *simp*
    show *op-*$\mathfrak{C}$*-*$\mathfrak{C}$: *category* α (*op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$) **by** (*auto simp*: *cat-cs-intros*)
    show *vcard* $Hom_{O.C\alpha}\mathfrak{C}(-,-)$ = $4_\mathbb{N}$
      unfolding *cf-Hom-def* **by** (*simp add*: *nat-omega-simps*)

    show $\mathcal{R}_\circ$ ($Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|)) $\subseteq_\circ$ *cat-Set* α(|*Obj*|)
      by (*simp add*: *cf-Hom-ObjMap-vrange*)
    show $Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ArrMap*|)(|*gf*|) :
      $Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|)(|*ab*|) $\mapsto_{cat\text{-}Set\ \alpha}$ $Hom_{O.C\alpha}\mathfrak{C}(-,-)$(|*ObjMap*|)(|*cd*|)
      if *gf*: *gf* : *ab* $\mapsto_{op\text{-}cat\ \mathfrak{C}\ \times_C\ \mathfrak{C}}$ *cd* **for** *gf ab cd*
      unfolding *slicing-simps cat-smc-cat-Set*[*symmetric*]

```
proof–
  obtain g f a b c d where gf-def: gf = [g, f]∘
    and ab-def: ab = [a, b]∘
    and cd-def: cd = [c, d]∘
    and g : a ↦_op-cat 𝔈 c
    and f: f : b ↦_𝔈 d
    by (elim cat-prod-2-is-arrE[OF category-op category-axioms gf])
  then have g: g : c ↦_𝔈 a unfolding cat-op-simps by simp
  from category-axioms that g f show Hom_O.Cα𝔈(−,−)(|ArrMap|)(|gf|) :
    Hom_O.Cα𝔈(−,−)(|ObjMap|)(|ab|) ↦_cat-Set α Hom_O.Cα𝔈(−,−)(|ObjMap|)(|cd|)
    unfolding gf-def ab-def cd-def
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros)
qed

show Hom_O.Cα𝔈(−,−)(|ArrMap|)(|gg′ ∘_A op-cat 𝔈 ×_C 𝔈 ff′|) =
  Hom_O.Cα𝔈(−,−)(|ArrMap|)(|gg′|) ∘_A cat-Set α Hom_O.Cα𝔈(−,−)(|ArrMap|)(|ff′|)
  if gg′: gg′ : bb′ ↦_op-cat 𝔈 ×_C 𝔈 cc′
    and ff′: ff′ : aa′ ↦_op-cat 𝔈 ×_C 𝔈 bb′
  for gg′ bb′ cc′ ff′ aa′
proof–
  obtain g g′ b b′ c c′
    where gg′-def: gg′ = [g, g′]∘
      and bb′-def: bb′ = [b, b′]∘
      and cc′-def: cc′ = [c, c′]∘
      and g : b ↦_op-cat 𝔈 c
      and g′: g′ : b′ ↦_𝔈 c′
    by (elim cat-prod-2-is-arrE[OF category-op category-axioms gg′])
  moreover obtain f f′ a a′ b″ b‴
    where ff′-def: ff′ = [f, f′]∘
      and aa′-def: aa′ = [a, a′]∘
      and bb′ = [b″, b‴]∘
      and f : a ↦_op-cat 𝔈 b″
      and f′ : a′ ↦_𝔈 b‴
    by (elim cat-prod-2-is-arrE[OF category-op category-axioms ff′])
  ultimately have f: f : b ↦_𝔈 a
    and f′: f′ : a′ ↦_𝔈 b′
    and g: g : c ↦_𝔈 b
    by (auto simp: cat-op-simps)
  from category-axioms that g f  g′ f′ show ?thesis
    unfolding
      slicing-simps cat-smc-cat-Set[symmetric]
      gg′-def bb′-def cc′-def ff′-def aa′-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cat-prod-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed

show Hom_O.Cα𝔈(−,−)(|ArrMap|)(|(op-cat 𝔈 ×_C 𝔈)(|CId|)(|cc′|)|) =
  cat-Set α(|CId|)(|Hom_O.Cα𝔈(−,−)(|ObjMap|)(|cc′|)|)
  if cc′ ∈_∘ (op-cat 𝔈 ×_C 𝔈)(|Obj|) for cc′
proof–
  from that obtain c c′
    where cc′-def: cc′ = [c, c′]∘
      and c: c ∈_∘ op-cat 𝔈(|Obj|)
      and c′: c′ ∈_∘ 𝔈(|Obj|)
```

```
    by (elim cat-prod-2-ObjE[rotated 2]) (auto intro: cat-cs-intros)
  then have c: c ∈₀ ℭ⦇Obj⦈ unfolding cat-op-simps by simp
  with c' category-axioms Set.category-axioms that show ?thesis
    unfolding cc'-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cat-prod-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed

qed (auto simp: cf-Hom-components cat-cs-intros)

qed

lemma (in category) cat-Hom-is-functor':
  assumes β = α and 𝔄' = op-cat ℭ ×_C ℭ and 𝔅' = cat-Set α
  shows Hom_{O.Cα}ℭ(−,−) : 𝔄' ↦↦_{Cβ} 𝔅'
  unfolding assms by (rule cat-Hom-is-functor)

lemmas [cat-cs-intros] = category.cat-Hom-is-functor'
```

26.3 Composition of a *Hom*-functor and two functors

26.3.1 Definition and elementary properties

```
definition cf-bcomp-Hom :: V ⇒ V ⇒ V ⇒ V ⇒ V (‹Hom_{O.C}1-'(/-−,-−/')›)
  — The following definition may seem redundant, but it will help to avoid proof duplication later.
  where Hom_{O.Cα}ℭ(𝔉−,𝔊−) = cf-cn-cov-bcomp (Hom_{O.Cα}ℭ(−,−)) 𝔉 𝔊
```

26.3.2 Object map

```
lemma cf-bcomp-Hom-ObjMap-vsv: vsv (Hom_{O.Cα}ℭ(𝔉−,𝔊−)⦇ObjMap⦈)
  unfolding cf-bcomp-Hom-def by (rule cf-cn-cov-bcomp-ObjMap-vsv)

lemma cf-bcomp-Hom-ObjMap-vdomain[cat-cs-simps]:
  assumes 𝔉 : 𝔄 ↦↦_{Cα} ℭ and 𝔊 : 𝔅 ↦↦_{Cα} ℭ
  shows 𝒟₀ (Hom_{O.Cα}ℭ(𝔉−,𝔊−)⦇ObjMap⦈) = (op-cat 𝔄 ×_C 𝔅)⦇Obj⦈
  using assms unfolding cf-bcomp-Hom-def by (rule cf-cn-cov-bcomp-ObjMap-vdomain)

lemma cf-bcomp-Hom-ObjMap-app[cat-cs-simps]:
  assumes 𝔉 : 𝔄 ↦↦_{Cα} ℭ
    and 𝔊 : 𝔅 ↦↦_{Cα} ℭ
    and [a, b]₀ ∈₀ (op-cat 𝔄 ×_C 𝔅)⦇Obj⦈
  shows Hom_{O.Cα}ℭ(𝔉−,𝔊−)⦇ObjMap⦈⦇a, b⦈• =
    Hom_{O.Cα}ℭ(−,−)⦇ObjMap⦈⦇𝔉⦇ObjMap⦈⦇a⦈, 𝔊⦇ObjMap⦈⦇b⦈⦈•
  using assms unfolding cf-bcomp-Hom-def by (rule cf-cn-cov-bcomp-ObjMap-app)

lemma (in category) cf-bcomp-Hom-ObjMap-vrange:
  assumes 𝔉 : 𝔄 ↦↦_{Cα} ℭ
    and 𝔊 : 𝔅 ↦↦_{Cα} ℭ
  shows ℛ₀ (Hom_{O.Cα}ℭ(𝔉−,𝔊−)⦇ObjMap⦈) ⊆₀ cat-Set α⦇Obj⦈
  using category-axioms
  unfolding cf-bcomp-Hom-def
  by (intro cf-cn-cov-bcomp-ObjMap-vrange[OF assms])
    (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
```

26.3.3 Arrow map

lemma *cf-bcomp-Hom-ArrMap-vsv*: *vsv* (*Hom*$_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}-)(\lfloor ArrMap \rfloor)$)
unfolding *cf-bcomp-Hom-def* **by** (*rule cf-cn-cov-bcomp-ArrMap-vsv*)

lemma *cf-bcomp-Hom-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ$ (*Hom*$_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}-)(\lfloor ArrMap \rfloor)$) = (*op-cat* $\mathfrak{A} \times_C \mathfrak{B}$)($\lfloor Arr \rfloor$)
using *assms*
unfolding *cf-bcomp-Hom-def*
by (*rule cf-cn-cov-bcomp-ArrMap-vdomain*)

lemma *cf-bcomp-Hom-ArrMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $[f, g]_\circ \in_\circ$ (*op-cat* $\mathfrak{A} \times_C \mathfrak{B}$)($\lfloor Arr \rfloor$)
shows
Hom$_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}-)(\lfloor ArrMap \rfloor)(\lfloor f, g \rfloor)_\bullet$ =
Hom$_{O.C\alpha}\mathfrak{C}(-,-)(\lfloor ArrMap \rfloor)(\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor), \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor))_\bullet$
using *assms*
unfolding *cf-bcomp-Hom-def*
by (*rule cf-cn-cov-bcomp-ArrMap-app*)

lemma (**in** *category*) *cf-bcomp-Hom-ArrMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ$ (*Hom*$_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}-)(\lfloor ArrMap \rfloor)$) $\subseteq_\circ$ *cat-Set* $\alpha(\lfloor Arr \rfloor)$
using *category-axioms*
unfolding *cf-bcomp-Hom-def*
by (*intro cf-cn-cov-bcomp-ArrMap-vrange*[*OF assms*])
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

26.3.4 Composition of a *Hom*-functor and two functors is a functor

lemma (**in** *category*) *cat-cf-bcomp-Hom-is-functor*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *Hom*$_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}-)$: *op-cat* $\mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
using *assms category-axioms*
unfolding *cf-bcomp-Hom-def*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma (**in** *category*) *cat-cf-bcomp-Hom-is-functor'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\beta = \alpha$
and $\mathfrak{A}' =$ *op-cat* $\mathfrak{A} \times_C \mathfrak{B}$
and $\mathfrak{B}' =$ *cat-Set* α
shows *Hom*$_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}-) : \mathfrak{A}' \mapsto\mapsto_{C\beta} \mathfrak{B}'$
using *assms*(*1,2*) **unfolding** *assms*(*3–5*) **by** (*rule cat-cf-bcomp-Hom-is-functor*)

lemmas [*cat-cs-intros*] = *category.cat-cf-bcomp-Hom-is-functor'*

26.4 Composition of a *Hom*-functor and a functor

26.4.1 Definition and elementary properties

See subsection 1.15 in [3].

definition *cf-lcomp-Hom* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle Hom_{O.C^1}\text{-}'(/--,-/')\rangle$)
  where $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-) = cf\text{-}cn\text{-}cov\text{-}lcomp\ \mathfrak{C}\ (Hom_{O.C\alpha}\mathfrak{C}(-,-))\ \mathfrak{F}$

definition *cf-rcomp-Hom* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle Hom_{O.C^1}\text{-}'(/-,--/')\rangle$)
  where $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-) = cf\text{-}cn\text{-}cov\text{-}rcomp\ \mathfrak{C}\ (Hom_{O.C\alpha}\mathfrak{C}(-,-))\ \mathfrak{G}$

26.4.2 Object map

lemma *cf-lcomp-Hom-ObjMap-vsv*[*cat-cs-intros*]: *vsv* ($Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ObjMap|\!)$)
  unfolding *cf-lcomp-Hom-def* **by** (*rule cf-cn-cov-lcomp-ObjMap-vsv*)

lemma *cf-rcomp-Hom-ObjMap-vsv*[*cat-cs-intros*]: *vsv* ($Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ObjMap|\!)$)
  unfolding *cf-rcomp-Hom-def* **by** (*rule cf-cn-cov-rcomp-ObjMap-vsv*)

lemma *cf-lcomp-Hom-ObjMap-vdomain*[*cat-cs-simps*]:
  assumes *category* $\alpha\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows $\mathcal{D}_\circ\ (Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ObjMap|\!)) = (op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C})(\!|Obj|\!)$
  using *assms*
  by
    (
      cs-concl **cs-shallow**
        cs-simp: *cat-cs-simps cf-lcomp-Hom-def* **cs-intro**: *cat-cs-intros*
)

lemma *cf-rcomp-Hom-ObjMap-vdomain*[*cat-cs-simps*]:
  assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows $\mathcal{D}_\circ\ (Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ObjMap|\!)) = (op\text{-}cat\ \mathfrak{C} \times_C \mathfrak{B})(\!|Obj|\!)$
  using *assms*
  by
    (
      cs-concl **cs-shallow**
        cs-simp: *cat-cs-simps cf-rcomp-Hom-def* **cs-intro**: *cat-cs-intros*
)

lemma *cf-lcomp-Hom-ObjMap-app*[*cat-cs-simps*]:
  assumes *category* $\alpha\ \mathfrak{C}$
    and $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $b \in_\circ op\text{-}cat\ \mathfrak{B}(\!|Obj|\!)$
    and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
  shows $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ObjMap|\!)(\!|b,\ c|\!)_\bullet =$
    $Hom_{O.C\alpha}\mathfrak{C}(-,-)(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),\ c|\!)_\bullet$
  using *assms*
  unfolding *cf-lcomp-Hom-def*
  by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-prod-cs-intros*)

lemma *cf-rcomp-Hom-ObjMap-app*[*cat-cs-simps*]:
  assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $c \in_\circ op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
    and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
  shows $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ObjMap|\!)(\!|c,\ b|\!)_\bullet =$
    $Hom_{O.C\alpha}\mathfrak{C}(-,-)(\!|ObjMap|\!)(\!|c,\ \mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)|\!)_\bullet$
  using *assms*
  by

(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cf-rcomp-Hom-def*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

lemma (**in** *category*) *cat-cf-lcomp-Hom-ObjMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ\ (Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ObjMap|\!)) \subseteq_\circ cat\text{-}Set\ \alpha(\!|Obj|\!)$
using *category-axioms assms*
unfolding *cf-lcomp-Hom-def*
by (*intro cf-cn-cov-lcomp-ObjMap-vrange*)
(*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)

lemma (**in** *category*) *cat-cf-rcomp-Hom-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_\circ\ (Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ObjMap|\!)) \subseteq_\circ cat\text{-}Set\ \alpha(\!|Obj|\!)$
using *category-axioms assms*
unfolding *cf-rcomp-Hom-def*
by (*intro cf-cn-cov-rcomp-ObjMap-vrange*)
(*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)

26.4.3 Arrow map

lemma *cf-lcomp-Hom-ArrMap-vsv*[*cat-cs-intros*]: *vsv* $(Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ArrMap|\!))$
unfolding *cf-lcomp-Hom-def* **by** (*rule cf-cn-cov-lcomp-ArrMap-vsv*)

lemma *cf-rcomp-Hom-ArrMap-vsv*[*cat-cs-intros*]: *vsv* $(Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ArrMap|\!))$
unfolding *cf-rcomp-Hom-def* **by** (*rule cf-cn-cov-rcomp-ArrMap-vsv*)

lemma *cf-lcomp-Hom-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* $\alpha\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ\ (Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ArrMap|\!)) = (op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C})(\!|Arr|\!)$
using *assms*
unfolding *cf-lcomp-Hom-def*
by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma *cf-rcomp-Hom-ArrMap-vdomain*[*cat-cs-simps*]:
assumes *category* $\alpha\ \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_\circ\ (Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ArrMap|\!)) = (op\text{-}cat\ \mathfrak{C} \times_C \mathfrak{B})(\!|Arr|\!)$
using *assms*
unfolding *cf-rcomp-Hom-def*
by (*cs-concl* **cs-shallow** **cs-simp**: *cat-cs-simps*)

lemma *cf-lcomp-Hom-ArrMap-app*[*cat-cs-simps*]:
assumes *category* $\alpha\ \mathfrak{C}$
and $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $g : a \mapsto_{op\text{-}cat\ \mathfrak{B}} b$
and $f : a' \mapsto_{\mathfrak{C}} b'$
shows $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\!|ArrMap|\!)(\!|g, f|\!)_\bullet =$
$Hom_{O.C\alpha}\mathfrak{C}(-,-)(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!), f|\!)_\bullet$
using *assms*
unfolding *cf-lcomp-Hom-def cat-op-simps*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-prod-cs-intros*

)

lemma *cf-rcomp-Hom-ArrMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $g : a \mapsto_{op\text{-}cat\ \mathfrak{C}} b$
and $f : a' \mapsto_{\mathfrak{B}} b'$
shows $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\lfloor ArrMap \rfloor)(\lfloor g,\ f \rfloor)_{\bullet} =$
$Hom_{O.C\alpha}\mathfrak{C}(-,-)(\lfloor ArrMap \rfloor)(\lfloor g,\ \mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \rfloor)_{\bullet}$
using *assms*
by
(
cs-concl
cs-simp: *cat-cs-simps cf-rcomp-Hom-def*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

lemma (**in** *category*) *cf-lcomp-Hom-ArrMap-vrange*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}\ (Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)(\lfloor ArrMap \rfloor)) \subseteq_{\circ} cat\text{-}Set\ \alpha(\lfloor Arr \rfloor)$
using *category-axioms assms*
unfolding *cf-lcomp-Hom-def*
by (*intro cf-cn-cov-lcomp-ArrMap-vrange*)
(*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemma (**in** *category*) *cf-rcomp-Hom-ArrMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}\ (Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\lfloor ArrMap \rfloor)) \subseteq_{\circ} cat\text{-}Set\ \alpha(\lfloor Arr \rfloor)$
using *category-axioms assms*
unfolding *cf-rcomp-Hom-def*
by (*intro cf-cn-cov-rcomp-ArrMap-vrange*)
(*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

26.4.4 Further properties

lemma *cf-bcomp-Hom-cf-lcomp-Hom*[*cat-cs-simps*]:
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,cf\text{-}id\ \mathfrak{C}-) = Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$
unfolding *cf-lcomp-Hom-def cf-cn-cov-lcomp-def cf-bcomp-Hom-def* **..**

lemma *cf-bcomp-Hom-cf-rcomp-Hom*[*cat-cs-simps*]:
$Hom_{O.C\alpha}\mathfrak{C}(cf\text{-}id\ \mathfrak{C}-,\mathfrak{G}-) = Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$
unfolding *cf-rcomp-Hom-def cf-cn-cov-rcomp-def cf-bcomp-Hom-def* **..**

26.4.5 Composition of a *Hom*-functor and a functor is a functor

lemma (**in** *category*) *cat-cf-lcomp-Hom-is-functor*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-) : op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
using *category-axioms assms*
unfolding *cf-lcomp-Hom-def*
by (*intro cf-cn-cov-lcomp-is-functor*)
(*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

lemma (**in** *category*) *cat-cf-lcomp-Hom-is-functor'*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\beta = \alpha$
and $\mathfrak{A}' = op\text{-}cat\ \mathfrak{B} \times_C \mathfrak{C}$
and $\mathfrak{B}' = cat\text{-}Set\ \alpha$
shows $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-) : \mathfrak{A}' \mapsto\mapsto_{C\beta} \mathfrak{B}'$

using *assms*(*1*)
unfolding *assms*(*2–4*)
by (*rule cat-cf-lcomp-Hom-is-functor*)

lemmas [*cat-cs-intros*] = *category.cat-cf-lcomp-Hom-is-functor'*

lemma (**in** *category*) *cat-cf-rcomp-Hom-is-functor*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-) : op\text{-}cat\ \mathfrak{C} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
using *category-axioms assms*
unfolding *cf-rcomp-Hom-def*
by (*intro cf-cn-cov-rcomp-is-functor*)
(*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros cat-op-intros*)

lemma (**in** *category*) *cat-cf-rcomp-Hom-is-functor'*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\beta = \alpha$
and $\mathfrak{A}' = op\text{-}cat\ \mathfrak{C} \times_C \mathfrak{B}$
and $\mathfrak{B}' = cat\text{-}Set\ \alpha$
shows $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-) : \mathfrak{A}' \mapsto\mapsto_{C\beta} \mathfrak{B}'$
using *assms*(*1*)
unfolding *assms*(*2–4*)
by (*rule cat-cf-rcomp-Hom-is-functor*)

lemmas [*cat-cs-intros*] = *category.cat-cf-rcomp-Hom-is-functor'*

26.4.6 Flip of a projections of a *Hom*-functor

lemma (**in** *category*) *cat-bifunctor-flip-cf-rcomp-Hom*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
bifunctor-flip (*op-cat* $\mathfrak{C}$) $\mathfrak{B}$ ($Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$) =
$Hom_{O.C\alpha}op\text{-}cat\ \mathfrak{C}(op\text{-}cf\ \mathfrak{G}-,-)$
proof(*rule cf-eqI*)

interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*)

from *category-axioms assms* **show** *bf-Hom*:
bifunctor-flip (*op-cat* $\mathfrak{C}$) $\mathfrak{B}$ $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)$:
$\mathfrak{B} \times_C op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
from *category-axioms assms* **show** *op-Hom*:
$Hom_{O.C\alpha}op\text{-}cat\ \mathfrak{C}(op\text{-}cf\ \mathfrak{G}-,-) : \mathfrak{B} \times_C op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)
from *bf-Hom* **have** *ObjMap-dom-lhs*:
$\mathcal{D}_\circ$ (*bifunctor-flip* (*op-cat* $\mathfrak{C}$) $\mathfrak{B}$ $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ObjMap|\!)$) =
($\mathfrak{B} \times_C op\text{-}cat\ \mathfrak{C}$)($|Obj|$)
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
from *op-Hom* **have** *ObjMap-dom-rhs*:
$\mathcal{D}_\circ$ ($Hom_{O.C\alpha}op\text{-}cat\ \mathfrak{C}(op\text{-}cf\ \mathfrak{G}-,-)(\!|ObjMap|\!)$) = ($\mathfrak{B} \times_C op\text{-}cat\ \mathfrak{C}$)($|Obj|$)
by (*cs-concl* **cs-simp**: *cat-cs-simps*)
from *bf-Hom* **have** *ArrMap-dom-lhs*:
$\mathcal{D}_\circ$ (*bifunctor-flip* (*op-cat* $\mathfrak{C}$) $\mathfrak{B}$ $Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G}-)(\!|ArrMap|\!)$) =
($\mathfrak{B} \times_C op\text{-}cat\ \mathfrak{C}$)($|Arr|$)
by (*cs-concl* **cs-simp**: *cat-cs-simps*)

```
  from op-Hom have ArrMap-dom-rhs:
    𝒟∘ (Hom_{O.Cα} op-cat 𝔈(op-cf 𝔊−,−)(|ArrMap|)) = (𝔅 ×_C op-cat 𝔈)(|Arr|)
    by (cs-concl cs-simp: cat-cs-simps)

  show
    bifunctor-flip (op-cat 𝔈) 𝔅 Hom_{O.Cα}𝔈(−,𝔊−)(|ObjMap|) =
      Hom_{O.Cα} op-cat 𝔈(op-cf 𝔊−,−)(|ObjMap|)
  proof(rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
    fix bc assume bc ∈∘ (𝔅 ×_C op-cat 𝔈)(|Obj|)
    then obtain b c
      where bc-def: bc = [b, c]∘ and b: b ∈∘ 𝔅(|Obj|) and c: c ∈∘ 𝔈(|Obj|)
      by
        (
          auto
            elim: cat-prod-2-ObjE[OF 𝔊.HomDom.category-axioms category-op]
            simp: cat-op-simps
        )
    from category-axioms assms b c show
      bifunctor-flip (op-cat 𝔈) 𝔅 Hom_{O.Cα}𝔈(−,𝔊−)(|ObjMap|)(|bc|) =
        Hom_{O.Cα} op-cat 𝔈(op-cf 𝔊−,−)(|ObjMap|)(|bc|)
      unfolding bc-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
  qed (auto intro: cat-cs-intros)

  show
    bifunctor-flip (op-cat 𝔈) 𝔅 Hom_{O.Cα}𝔈(−,𝔊−)(|ArrMap|) =
      Hom_{O.Cα} op-cat 𝔈(op-cf 𝔊−,−)(|ArrMap|)
  proof(rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
    fix gf assume gf ∈∘ (𝔅 ×_C op-cat 𝔈)(|Arr|)
    then obtain g f
      where gf-def: gf = [g, f]∘ and g ∈∘ 𝔅(|Arr|) and f ∈∘ 𝔈(|Arr|)
      by
        (
          auto
            elim: cat-prod-2-ArrE[OF 𝔊.HomDom.category-axioms category-op]
            simp: cat-op-simps
        )
    then obtain a b c d where g: g : a ↦_𝔅 b and f: f : c ↦_𝔈 d
      by (auto intro!: is-arrI)
    from category-axioms assms g f show
      bifunctor-flip (op-cat 𝔈) 𝔅 Hom_{O.Cα}𝔈(−,𝔊−)(|ArrMap|)(|gf|) =
        Hom_{O.Cα} op-cat 𝔈(op-cf 𝔊−,−)(|ArrMap|)(|gf|)
      unfolding gf-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
  qed (auto intro: cat-cs-intros)

qed (auto intro: cat-cs-intros simp: cat-op-simps)
```

lemmas [*cat-cs-simps*] = *category.cat-bifunctor-flip-cf-rcomp-Hom*

lemma (**in** *category*) *cat-bifunctor-flip-cf-lcomp-Hom*:
  assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows
    bifunctor-flip (*op-cat* $\mathfrak{B}$) $\mathfrak{C}$ ($Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$) =
      $Hom_{O.C\alpha}$*op-cat* $\mathfrak{C}(-,$*op-cf* $\mathfrak{F}-)$
proof–
  interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
  note *Hom*-$\mathfrak{F}$ =
    category.cat-bifunctor-flip-cf-rcomp-Hom
      [
        OF category-op is-functor-op[*OF assms*],
        unfolded cat-op-simps,
        symmetric
      ]
  from *category-axioms assms* **show** *?thesis*
    by (*subst Hom*-$\mathfrak{F}$)
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-cs-simps cat-op-simps*
          cs-intro: *cat-cs-intros cat-op-intros*
)+
qed

lemmas [*cat-cs-simps*] = *category.cat-bifunctor-flip-cf-lcomp-Hom*

26.5 Projections of the *Hom*-functor

The projections of the *Hom*-functor coincide with the definitions of the *Hom*-functor given in Chapter II-2 in [7]. They are also exposed in the aforementioned article in nLab [1][15].

26.5.1 Definitions and elementary properties

definition *cf-Hom-snd* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle Hom_{O.C^1}$-$'(/$-,-$/')\rangle$)
  where $Hom_{O.C\alpha}\mathfrak{C}(a,-) = Hom_{O.C\alpha}\mathfrak{C}(-,-)_{op\text{-}cat\ \mathfrak{C},\mathfrak{C}}(a,-)_{CF}$
definition *cf-Hom-fst* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle Hom_{O.C^1}$-$'(/$-,-$/')\rangle$)
  where $Hom_{O.C\alpha}\mathfrak{C}(-,b) = Hom_{O.C\alpha}\mathfrak{C}(-,-)_{op\text{-}cat\ \mathfrak{C},\mathfrak{C}}(-,b)_{CF}$

26.5.2 Projections of the *Hom*-functor are functors

lemma (**in** *category*) *cat-cf-Hom-snd-is-functor*:
  assumes $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
  shows $Hom_{O.C\alpha}\mathfrak{C}(a,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof–
  from *assms* **have** *a*: $a \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$ **unfolding** *cat-op-simps* **by** *simp*
  have *op-*$\mathfrak{C}$: *category* α (*op-cat* $\mathfrak{C}$) **by** (*auto intro*: *cat-cs-intros*)
  from *op-*$\mathfrak{C}$ *category-axioms cat-Hom-is-functor a* **show** *?thesis*
    unfolding *cf-Hom-snd-def* **by** (*rule bifunctor-proj-snd-is-functor*)
qed

lemma (**in** *category*) *cat-cf-Hom-snd-is-functor'*:
  assumes $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $\beta = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{D}' =$ *cat-Set* α
  shows $Hom_{O.C\alpha}\mathfrak{C}(a,-) : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}'$
  using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cat-cf-Hom-snd-is-functor*)

[15] https://ncatlab.org/nlab/show/hom-functor

lemmas [*cat-cs-intros*] = *category.cat-cf-Hom-snd-is-functor'*

lemma (**in** *category*) *cat-cf-Hom-fst-is-functor*:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $Hom_{O.C\alpha}\mathfrak{C}(-,b) : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
proof–
have *op-*$\mathfrak{C}$: *category* α (*op-cat* $\mathfrak{C}$) **by** (*auto intro*: *cat-cs-intros*)
from *op-*$\mathfrak{C}$ *category-axioms cat-Hom-is-functor assms* **show** *?thesis*
unfolding *cf-Hom-fst-def* **by** (*rule bifunctor-proj-fst-is-functor*)
qed

lemma (**in** *category*) *cat-cf-Hom-fst-is-functor'*:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $\beta = \alpha$ **and** $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$ **and** $\mathfrak{D}' = cat\text{-}Set\ \alpha$
shows $Hom_{O.C\alpha}\mathfrak{C}(-,b) : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}'$
using *assms*(*1*) **unfolding** *assms*(*2–4*) **by** (*rule cat-cf-Hom-fst-is-functor*)

lemmas [*cat-cs-intros*] = *category.cat-cf-Hom-fst-is-functor'*

26.5.3 Object maps

lemma (**in** *category*) *cat-cf-Hom-snd-ObjMap-vsv*[*cat-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *vsv* ($Hom_{O.C\alpha}\mathfrak{C}(a,-)(\!|ObjMap|\!)$)
unfolding *cf-Hom-snd-def*
using *category-axioms assms*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-intros*] = *category.cat-cf-Hom-snd-ObjMap-vsv*

lemma (**in** *category*) *cat-cf-Hom-fst-ObjMap-vsv*[*cat-cs-intros*]:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *vsv* ($Hom_{O.C\alpha}\mathfrak{C}(-,b)(\!|ObjMap|\!)$)
unfolding *cf-Hom-fst-def*
using *category-axioms assms*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-prod-cs-simps cat-cs-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)

lemmas [*cat-cs-intros*] = *category.cat-cf-Hom-fst-ObjMap-vsv*

lemma (**in** *category*) *cat-cf-Hom-snd-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}$ ($Hom_{O.C\alpha}\mathfrak{C}(a,-)(\!|ObjMap|\!)$) = $\mathfrak{C}(\!|Obj|\!)$
using *category-axioms assms*
unfolding *cf-Hom-snd-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-simps*] = *category.cat-cf-Hom-snd-ObjMap-vdomain*

lemma (**in** *category*) *cat-cf-Hom-fst-ObjMap-vdomain*[*cat-cs-simps*]:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}\ (Hom_{O.C\alpha}\mathfrak{C}(-,b)(\!|ObjMap|\!)) = op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
using *category-axioms assms*
unfolding *cf-Hom-fst-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-simps*] = *category.cat-cf-Hom-fst-ObjMap-vdomain*

lemma (**in** *category*) *cat-cf-Hom-snd-ObjMap-app*[*cat-cs-simps*]:
assumes $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $Hom_{O.C\alpha}\mathfrak{C}(a,-)(\!|ObjMap|\!)(\!|b|\!) = Hom\ \mathfrak{C}\ a\ b$
proof–
from *assms* **have** *ab*: $[a,\ b]_{\circ} \in_{\circ} (op\text{-}cat\ \mathfrak{C} \times_C \mathfrak{C})(\!|Obj|\!)$
by (*intro cat-prod-2-ObjI*) (*auto intro*: *cat-cs-intros*)
show *?thesis*
unfolding
cf-Hom-snd-def
bifunctor-proj-snd-ObjMap-app[*OF category-op category-axioms ab*]
cf-Hom-ObjMap-app[*OF ab*]
..
qed

lemmas [*cat-cs-simps*] = *category.cat-cf-Hom-snd-ObjMap-app*

lemma (**in** *category*) *cat-cf-Hom-fst-ObjMap-app*[*cat-cs-simps*]:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
shows $Hom_{O.C\alpha}\mathfrak{C}(-,b)(\!|ObjMap|\!)(\!|a|\!) = Hom\ \mathfrak{C}\ a\ b$
proof–
from *assms* **have** *ab*: $[a,\ b]_{\circ} \in_{\circ} (op\text{-}cat\ \mathfrak{C} \times_C \mathfrak{C})(\!|Obj|\!)$
by (*intro cat-prod-2-ObjI*) (*auto intro*: *cat-cs-intros*)
show *?thesis*
unfolding
cf-Hom-fst-def
bifunctor-proj-fst-ObjMap-app[*OF category-op category-axioms ab*]
cf-Hom-ObjMap-app[*OF ab*]
..
qed

lemmas [*cat-cs-simps*] = *category.cat-cf-Hom-fst-ObjMap-app*

26.5.4 Arrow maps

lemma (**in** *category*) *cat-cf-Hom-snd-ArrMap-vsv*[*cat-cs-intros*]:
assumes $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
shows *vsv* $(Hom_{O.C\alpha}\mathfrak{C}(a,-)(\!|ArrMap|\!))$
unfolding *cf-Hom-snd-def*
using *category-axioms assms*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*

cs-intro: *bifunctor-proj-snd-ArrMap-vsv cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-intros*] = *category.cat-cf-Hom-snd-ArrMap-vsv*

lemma (**in** *category*) *cat-cf-Hom-fst-ArrMap-vsv*[*cat-cs-intros*]:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *vsv* ($Hom_{O.C\alpha}\mathfrak{C}(-,b)(\!|ArrMap|\!)$)
unfolding *cf-Hom-fst-def*
using *category-axioms assms*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *bifunctor-proj-fst-ArrMap-vsv cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-intros*] = *category.cat-cf-Hom-fst-ArrMap-vsv*

lemma (**in** *category*) *cat-cf-Hom-snd-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}$ ($Hom_{O.C\alpha}\mathfrak{C}(a,-)(\!|ArrMap|\!)$) = $\mathfrak{C}(\!|Arr|\!)$
using *category-axioms assms*
unfolding *cf-Hom-snd-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-simps*] = *category.cat-cf-Hom-snd-ArrMap-vdomain*

lemma (**in** *category*) *cat-cf-Hom-fst-ArrMap-vdomain*[*cat-cs-simps*]:
assumes $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}$ ($Hom_{O.C\alpha}\mathfrak{C}(-,b)(\!|ArrMap|\!)$) = $op\text{-}cat\ \mathfrak{C}(\!|Arr|\!)$
using *category-axioms assms*
unfolding *cf-Hom-fst-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemmas [*cat-cs-simps*] = *category.cat-cf-Hom-fst-ArrMap-vdomain*

lemma (**in** *category*) *cat-cf-Hom-snd-ArrMap-app*[*cat-cs-simps*]:
assumes $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$ **and** $f : b \mapsto_{\mathfrak{C}} b'$
shows $Hom_{O.C\alpha}\mathfrak{C}(a,-)(\!|ArrMap|\!)(\!|f|\!)$ = *cf-hom* $\mathfrak{C}$ [$op\text{-}cat\ \mathfrak{C}(\!|CId|\!)(\!|a|\!)$, f]$_{\circ}$
proof–
from *assms*(*2*) **have** *f*: $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)$ **by** (*simp add*: *cat-cs-intros*)
from *category-axioms assms* **show** *?thesis*
unfolding
cf-Hom-snd-def
bifunctor-proj-snd-ArrMap-app[*OF category-op category-axioms assms*(*1*) *f*]
cat-op-simps
by
(
cs-concl **cs-shallow**

```
      cs-simp: cat-cs-simps
      cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
    )
qed

lemmas [cat-cs-simps] = category.cat-cf-Hom-snd-ArrMap-app

lemma (in category) cat-cf-Hom-fst-ArrMap-app[cat-cs-simps]:
  assumes b ∈₀ 𝔈(|Obj|) and f : a ↦_{op-cat 𝔈} a'
  shows Hom_{O.Cα}𝔈(−,b)(|ArrMap|)(|f|) = cf-hom 𝔈 [f, 𝔈(|CId|)(|b|)]₀
proof–
  from assms(2) have f: f ∈₀ op-cat 𝔈(|Arr|) by (simp add: cat-cs-intros)
  with category-axioms assms show ?thesis
    unfolding
      cf-Hom-fst-def
      bifunctor-proj-fst-ArrMap-app[OF category-op category-axioms assms(1) f]
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed

lemmas [cat-cs-simps] = category.cat-cf-Hom-fst-ArrMap-app
```

26.5.5 Opposite *Hom*-functor projections

```
lemma (in category) cat-op-cat-cf-Hom-snd:
  assumes a ∈₀ 𝔈(|Obj|)
  shows Hom_{O.Cα}op-cat 𝔈(a,−) = Hom_{O.Cα}𝔈(−,a)
proof(rule cf-eqI[of α])

  from assms category-axioms show
    Hom_{O.Cα}op-cat 𝔈(a,−) : op-cat 𝔈 ↦↦_{Cα} cat-Set α
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
  from assms category-axioms show
    Hom_{O.Cα}𝔈(−,a) : op-cat 𝔈 ↦↦_{Cα} cat-Set α
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )

  show Hom_{O.Cα}op-cat 𝔈(a,−)(|ObjMap|) = Hom_{O.Cα}𝔈(−,a)(|ObjMap|)
  proof(rule vsv-eqI)
    from assms category-axioms show vsv (Hom_{O.Cα}op-cat 𝔈(a,−)(|ObjMap|))
      by (intro is-functor.cf-ObjMap-vsv)
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
```

```
    )
  from assms category-axioms show vsv (Hom_{O.C α} 𝔈(−,a)(|ObjMap|))
    by (intro is-functor.cf-ObjMap-vsv)
      (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms category-axioms show
    𝒟∘ (Hom_{O.C α} op-cat 𝔈(a,−)(|ObjMap|)) = 𝒟∘ (Hom_{O.C α} 𝔈(−,a)(|ObjMap|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
  show Hom_{O.C α} op-cat 𝔈(a,−)(|ObjMap|)(|b|) = Hom_{O.C α} 𝔈(−,a)(|ObjMap|)(|b|)
    if b ∈∘ 𝒟∘ (Hom_{O.C α} op-cat 𝔈(a,−)(|ObjMap|)) for b
  proof–
    from that have b ∈∘ 𝔈(|Obj|)
      by
        (
          simp add:
            category.cat-cf-Hom-snd-ObjMap-vdomain[
              OF category-op, unfolded cat-op-simps, OF assms
            ]
        )
    from category-axioms assms this show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-op-intros
        )
  qed
qed

show Hom_{O.C α} op-cat 𝔈(a,−)(|ArrMap|) = Hom_{O.C α} 𝔈(−,a)(|ArrMap|)
proof(rule vsv-eqI)
  from assms category-axioms show vsv (Hom_{O.C α} op-cat 𝔈(a,−)(|ArrMap|))
    by (intro is-functor.cf-ArrMap-vsv)
      (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)
  from assms category-axioms show vsv (Hom_{O.C α} 𝔈(−,a)(|ArrMap|))
    by (intro is-functor.cf-ArrMap-vsv)
      (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms category-axioms show
    𝒟∘ (Hom_{O.C α} op-cat 𝔈(a,−)(|ArrMap|)) = 𝒟∘ (Hom_{O.C α} 𝔈(−,a)(|ArrMap|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-op-intros
      )
  show Hom_{O.C α} op-cat 𝔈(a,−)(|ArrMap|)(|f|) = Hom_{O.C α} 𝔈(−,a)(|ArrMap|)(|f|)
    if f ∈∘ 𝒟∘ (Hom_{O.C α} op-cat 𝔈(a,−)(|ArrMap|)) for f
  proof–
    from that have f ∈∘ 𝔈(|Arr|)
      by
        (
          simp add:
            category.cat-cf-Hom-snd-ArrMap-vdomain[
              OF category-op, unfolded cat-op-simps, OF assms
            ]
        )
```

```
    then obtain a b where f : a ↦_𝔈 b by auto
    from category-axioms assms this show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros
        )
  qed
qed

qed simp-all

lemmas [cat-op-simps] = category.cat-op-cat-cf-Hom-snd

lemma (in category) cat-op-cat-cf-Hom-fst:
  assumes a ∈₀ 𝔈⦇Obj⦈
  shows Hom_{O.C α} op-cat 𝔈(−,a) = Hom_{O.C α} 𝔈(a,−)
proof–
  from assms have a: a ∈₀ op-cat 𝔈⦇Obj⦈ unfolding cat-op-simps .
  have Hom_{O.C α} 𝔈(a,−) = Hom_{O.C α} op-cat (op-cat 𝔈)(a,−)
    unfolding cat-op-simps ..
  also have ... = Hom_{O.C α}(op-cat 𝔈)(−,a)
    unfolding category.cat-op-cat-cf-Hom-snd[OF category-op a] by simp
  finally show Hom_{O.C α}(op-cat 𝔈)(−,a) = Hom_{O.C α} 𝔈(a,−) by simp
qed

lemmas [cat-op-simps] = category.cat-op-cat-cf-Hom-fst
```

26.5.6 *Hom*-functors are injections on objects

```
lemma (in category) cat-cf-Hom-snd-inj:
  assumes Hom_{O.C α} 𝔈(a,−) = Hom_{O.C α} 𝔈(b,−)
    and a ∈₀ 𝔈⦇Obj⦈
    and b ∈₀ 𝔈⦇Obj⦈
  shows a = b
proof(rule ccontr)
  assume prems: a ≠ b
  from assms(1) have Hom_{O.C α} 𝔈(a,−)⦇ObjMap⦈⦇b⦈ = Hom_{O.C α} 𝔈(b,−)⦇ObjMap⦈⦇b⦈
    by simp
  then have Hom 𝔈 a b = Hom 𝔈 b b
    unfolding
      cat-cf-Hom-snd-ObjMap-app[unfolded cat-op-simps, OF assms(2,3)]
      cat-cf-Hom-snd-ObjMap-app[unfolded cat-op-simps, OF assms(3,3)]
    by simp
  with assms prems show False by (force intro: cat-cs-intros)
qed

lemma (in category) cat-cf-Hom-fst-inj:
  assumes Hom_{O.C α} 𝔈(−,a) = Hom_{O.C α} 𝔈(−,b) and a ∈₀ 𝔈⦇Obj⦈ and b ∈₀ 𝔈⦇Obj⦈
  shows a = b
proof(rule ccontr)
  assume prems: a ≠ b
  from assms(1) have Hom_{O.C α} 𝔈(−,a)⦇ObjMap⦈⦇b⦈ = Hom_{O.C α} 𝔈(−,b)⦇ObjMap⦈⦇b⦈
    by simp
  then have Hom 𝔈 b a = Hom 𝔈 b b
    unfolding
      cat-cf-Hom-fst-ObjMap-app[unfolded cat-op-simps, OF assms(2,3)]
```

cat-cf-Hom-fst-ObjMap-app[*unfolded cat-op-simps*, *OF assms*(*3*,*3*)]
by *simp*
with *assms prems* **show** *False* **by** (*force intro*: *cat-cs-intros*)
qed

26.5.7 *Hom*-functor is an array bifunctor

lemma (**in** *category*) *cat-cf-Hom-is-cf-array*:
— See Chapter II-3 in [7].
$Hom_{O.C\alpha}\mathfrak{C}(-,-)$ =
cf-array (*op-cat* $\mathfrak{C}$) $\mathfrak{C}$ (*cat-Set* α) (*cf-Hom-fst* α $\mathfrak{C}$) (*cf-Hom-snd* α $\mathfrak{C}$)
proof(*rule cf-eqI*[*of* α])

show $Hom_{O.C\alpha}\mathfrak{C}(-,-)$: *op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
by (*rule cat-Hom-is-functor*)

have *c1*: *category* α (*op-cat* $\mathfrak{C}$) **by** (*auto intro*: *cat-cs-intros*)
have *c2*: *category* α $\mathfrak{C}$ **by** (*auto intro*: *cat-cs-intros*)
have *c3*: *category* α (*cat-Set* α) **by** (*simp add*: *category-cat-Set*)
have *c4*: $Hom_{O.C\alpha}\mathfrak{C}(-,c)$: *op-cat* $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
if $c \in_\circ \mathfrak{C}(\!|Obj|\!)$ **for** c
using *that* **by** (*rule cat-cf-Hom-fst-is-functor*)
have *c5*: $Hom_{O.C\alpha}\mathfrak{C}(b,-)$: $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
if $b \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$ **for** b
using *that* **unfolding** *cat-op-simps* **by** (*rule cat-cf-Hom-snd-is-functor*)
have *c6*: $Hom_{O.C\alpha}\mathfrak{C}(b,-)(\!|ObjMap|\!)(\!|c|\!) = Hom_{O.C\alpha}\mathfrak{C}(-,c)(\!|ObjMap|\!)(\!|b|\!)$
if $b \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$ **and** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$ **for** b c
using *that category-axioms*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
have *c7*:
$Hom_{O.C\alpha}\mathfrak{C}(b',-)(\!|ArrMap|\!)(\!|g|\!) \circ_{A\,cat\text{-}Set\ \alpha}\ Hom_{O.C\alpha}\mathfrak{C}(-,c)(\!|ArrMap|\!)(\!|f|\!) =$
$Hom_{O.C\alpha}\mathfrak{C}(-,c')(\!|ArrMap|\!)(\!|f|\!) \circ_{A\,cat\text{-}Set\ \alpha}\ Hom_{O.C\alpha}\mathfrak{C}(b,-)(\!|ArrMap|\!)(\!|g|\!)$
if $f : b \mapsto_{op\text{-}cat\ \mathfrak{C}} b'$ **and** $g : c \mapsto_{\mathfrak{C}} c'$ **for** b c b' c' f g
using *that category-axioms*
unfolding *cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros*
)

let *?cfa* =
‹*cf-array* (*op-cat* $\mathfrak{C}$) $\mathfrak{C}$ (*cat-Set* α) (*cf-Hom-fst* α $\mathfrak{C}$) (*cf-Hom-snd* α $\mathfrak{C}$)›

note *cf-array-specification* =
cf-array-specification[*OF c1 c2 c3 c4 c5 c6 c7*, *simplified*]

from *c1 c2 c3 c4 c5 c6 c7* **show** *?cfa* : *op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
by (*rule cf-array-is-functor*)

show $Hom_{O.C\alpha}\mathfrak{C}(-,-)(\!|ObjMap|\!) = ?cfa(\!|ObjMap|\!)$
proof(*rule vsv-eqI*, *unfold cat-cs-simps*)
fix *aa'* **assume** $aa' \in_\circ$ (*op-cat* $\mathfrak{C}$ $\times_C$ $\mathfrak{C}$)$(\!|Obj|\!)$
then obtain *a a'*
where *aa'-def*: $aa' = [a, a']_\circ$
and *a*: $a \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$
and *a'*: $a' \in_\circ \mathfrak{C}(\!|Obj|\!)$

```
    by (elim cat-prod-2-ObjE[OF c1 c2])
  from category-axioms a a' show
    Hom_{O.Cα}ℭ(−,−)(|ObjMap|)(|aa'|) = ?cfa(|ObjMap|)(|aa'|)
    unfolding aa'-def cf-array-specification(2)[OF a a'] cat-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-op-intros cat-prod-cs-intros
      )
qed (auto simp: cf-array-ObjMap-vsv cf-Hom-ObjMap-vsv cat-cs-simps)

show Hom_{O.Cα}ℭ(−,−)(|ArrMap|) = ?cfa(|ArrMap|)
proof(rule vsv-eqI, unfold cat-cs-simps)
  fix ff' assume ff' ∈∘ (op-cat ℭ ×_C ℭ)(|Arr|)
  then obtain f f'
    where ff'-def: ff' = [f, f']∘
      and f: f ∈∘ op-cat ℭ(|Arr|)
      and f': f' ∈∘ ℭ(|Arr|)
    by (elim cat-prod-2-ArrE[OF c1 c2])
  then obtain a b a' b'
    where f: f : a ↦_{op-cat ℭ} b and f': f' : a' ↦_ℭ b'
    by (blast intro: is-arrI)
  from category-axioms f f' show cf-hom ℭ ff' = ?cfa(|ArrMap|)(|ff'|)
    unfolding ff'-def cat-op-simps
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros
      )
qed (auto simp: cf-array-ArrMap-vsv cf-Hom-ArrMap-vsv cat-cs-simps)

qed simp-all
```

26.5.8 Projections of the compositions of a *Hom*-functor and a functor are projections of the *Hom*-functor

```
lemma (in category) cat-cf-rcomp-Hom-cf-Hom-snd:
  assumes 𝔊 : 𝔅 ↦↦_{Cα} ℭ and a ∈∘ ℭ(|Obj|)
  shows Hom_{O.Cα}ℭ(−,𝔊−)_{op-cat ℭ,𝔅}(a,−)_{CF} = Hom_{O.Cα}ℭ(a,−) ∘_{CF} 𝔊
  using category-axioms assms
  unfolding cf-rcomp-Hom-def cf-Hom-snd-def
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
    )

lemmas [cat-cs-simps] = category.cat-cf-rcomp-Hom-cf-Hom-snd

lemma (in category) cat-cf-lcomp-Hom-cf-Hom-snd:
  assumes 𝔉 : 𝔅 ↦↦_{Cα} ℭ and b ∈∘ 𝔅(|Obj|)
  shows Hom_{O.Cα}ℭ(𝔉−,−)_{op-cat 𝔅,ℭ}(b,−)_{CF} = Hom_{O.Cα}ℭ(𝔉(|ObjMap|)(|b|),−)
  using category-axioms assms
  unfolding cf-lcomp-Hom-def cf-Hom-snd-def
  by
    (
      cs-concl cs-shallow
```

```
    cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
  )

lemmas [cat-cs-simps] = category.cat-cf-lcomp-Hom-cf-Hom-snd

lemma (in category) cat-cf-rcomp-Hom-cf-Hom-fst:
  assumes 𝔉 : 𝔅 ↦↦_{Cα} ℭ and b ∈_∘ 𝔅(|Obj|)
  shows Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF} = Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|))
proof–

  from category-axioms assms have H𝔉b:
    Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF} : op-cat ℭ ↦↦_{Cα} cat-Set α
    by (cs-concl cs-intro: cat-cs-intros)
  from category-axioms assms have H𝔉b′:
    Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|)) : op-cat ℭ ↦↦_{Cα} cat-Set α
    by (cs-concl cs-intro: cat-cs-intros)

  from category-axioms assms have [cat-cs-simps]:
    𝒟_∘ ((Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF})(|ObjMap|)) = op-cat ℭ(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros)+
  from category-axioms assms have [cat-cs-simps]:
    𝒟_∘ (Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|))(|ObjMap|)) = op-cat ℭ(|Obj|)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  from category-axioms assms have [cat-cs-simps]:
    𝒟_∘ ((Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF})(|ArrMap|)) = op-cat ℭ(|Arr|)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros)+
  from category-axioms assms have [cat-cs-simps]:
    𝒟_∘ (Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|))(|ArrMap|)) = op-cat ℭ(|Arr|)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  show ?thesis
  proof(rule cf-eqI[OF H𝔉b H𝔉b′])

    show
      (Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF})(|ObjMap|) =
        Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|))(|ObjMap|)
    proof(rule vsv-eqI, unfold cat-cs-simps)
      from category-axioms assms show
        vsv ((Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF})(|ObjMap|))
        by (intro bifunctor-proj-fst-ObjMap-vsv[of α])
          (cs-concl cs-intro: cat-cs-intros)+
      from assms show vsv (Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|))(|ObjMap|))
        by (intro cat-cf-Hom-fst-ObjMap-vsv)
          (cs-concl cs-intro: cat-cs-intros)+
      fix a assume prems: a ∈_∘ op-cat ℭ(|Obj|)
      with category-axioms assms show
        (Hom_{O.Cα}ℭ(−,𝔉−)_{op-cat ℭ,𝔅}(−,b)_{CF})(|ObjMap|)(|a|) =
          Hom_{O.Cα}ℭ(−,𝔉(|ObjMap|)(|b|))(|ObjMap|)(|a|)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps
              cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
          )
    qed simp
```

show
($Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}-)_{op\text{-}cat\ \mathfrak{C},\mathfrak{B}}(-,b)_{CF}$)(|*ArrMap*|) =
$Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}$(|*ObjMap*|)(|$b$|))(|*ArrMap*|)
proof(*rule vsv-eqI*, *unfold cat-cs-simps cat-op-simps*)
from *category-axioms assms* **show**
vsv (($Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}-)_{op\text{-}cat\ \mathfrak{C},\mathfrak{B}}(-,b)_{CF}$)(|*ArrMap*|))
by (*intro bifunctor-proj-fst-ArrMap-vsv*[*of* α])
(*cs-concl* **cs-intro**: *cat-cs-intros*)+
from *assms* **show** *vsv* ($Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}$(|*ObjMap*|)(|$b$|))(|*ArrMap*|))
by (*intro cat-cf-Hom-fst-ArrMap-vsv*)
(*cs-concl* **cs-intro**: *cat-cs-intros*)+
fix f **assume** $f \in_\circ \mathfrak{C}$(|*Arr*|)
then obtain a' b' **where** $f : a' \mapsto_{\mathfrak{C}} b'$ **by** (*auto simp*: *cat-op-simps*)
from *category-axioms assms this* **show**
($Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}-)_{op\text{-}cat\ \mathfrak{C},\mathfrak{B}}(-,b)_{CF}$)(|*ArrMap*|)(|$f$|) =
$Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}$(|*ObjMap*|)(|$b$|))(|*ArrMap*|)(|$f$|)
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed *simp*

qed *simp-all*

qed

lemmas [*cat-cs-simps*] = *category.cat-cf-rcomp-Hom-cf-Hom-fst*

27 Cones and cocones

27.1 Cone and cocone

27.1.1 Definition and elementary properties

In the context of this work, the concept of a cone corresponds to that of a cone to the base of a functor from a vertex, as defined in Chapter III-4 in [7]; the concept of a cocone corresponds to that of a cone from the base of a functor to a vertex, as defined in Chapter III-3 in [7].

locale *is-cat-cone* = *is-ntcf* α $\mathfrak{J}$ $\mathfrak{C}$ ‹*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*› $\mathfrak{F}$ $\mathfrak{N}$ **for** α *c* $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$ +
  assumes *cat-cone-obj*[*cat-cs-intros*]: *c* $\in_{\circ}$ $\mathfrak{C}$(|*Obj*|)

syntax *-is-cat-cone* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
  (‹(- :/ - <$_{CF.cone}$ - :/ - ↦↦$_{C1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-cone* ⇌ *is-cat-cone*
translations $\mathfrak{N}$: *c* <$_{CF.cone}$ $\mathfrak{F}$: $\mathfrak{J}$ ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌
  CONST is-cat-cone α *c* $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$

locale *is-cat-cocone* = *is-ntcf* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ ‹*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*› $\mathfrak{N}$ **for** α *c* $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$ +
  assumes *cat-cocone-obj*[*cat-cs-intros*]: *c* $\in_{\circ}$ $\mathfrak{C}$(|*Obj*|)

syntax *-is-cat-cocone* :: *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *V* ⇒ *bool*
  (‹(- :/ - >$_{CF.cocone}$ - :/ - ↦↦$_{C1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-cat-cocone* ⇌ *is-cat-cocone*
translations $\mathfrak{N}$: $\mathfrak{F}$ >$_{CF.cocone}$ *c* : $\mathfrak{J}$ ↦↦$_{C\alpha}$ $\mathfrak{C}$ ⇌
  CONST is-cat-cocone α *c* $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$

Rules.

lemma (**in** *is-cat-cone*) *is-cat-cone-axioms'*[*cat-cs-intros*]:
  assumes α' = α **and** *c'* = *c* **and** $\mathfrak{J}'$ = $\mathfrak{J}$ **and** $\mathfrak{C}'$ = $\mathfrak{C}$ **and** $\mathfrak{F}'$ = $\mathfrak{F}$
  shows $\mathfrak{N}$: *c'* <$_{CF.cone}$ $\mathfrak{F}'$: $\mathfrak{J}'$ ↦↦$_{C\alpha'}$ $\mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-cone-axioms*)

mk-ide rf *is-cat-cone-def*[*unfolded is-cat-cone-axioms-def*]
  |*intro is-cat-coneI*|
  |*dest is-cat-coneD*[*dest*!]|
  |*elim is-cat-coneE*[*elim*!]|

lemma (**in** *is-cat-cone*) *is-cat-coneD'*[*cat-cs-intros*]:
  assumes *c'* = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*
  shows $\mathfrak{N}$: *c'* ↦$_{CF}$ $\mathfrak{F}$: $\mathfrak{J}$ ↦↦$_{C\alpha}$ $\mathfrak{C}$
  unfolding *assms* **by** (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

lemma (**in** *is-cat-cocone*) *is-cat-cocone-axioms'*[*cat-cs-intros*]:
  assumes α' = α **and** *c'* = *c* **and** $\mathfrak{J}'$ = $\mathfrak{J}$ **and** $\mathfrak{C}'$ = $\mathfrak{C}$ **and** $\mathfrak{F}'$ = $\mathfrak{F}$
  shows $\mathfrak{N}$: $\mathfrak{F}'$ >$_{CF.cocone}$ *c'* : $\mathfrak{J}'$ ↦↦$_{C\alpha'}$ $\mathfrak{C}'$
  unfolding *assms* **by** (*rule is-cat-cocone-axioms*)

mk-ide rf *is-cat-cocone-def*[*unfolded is-cat-cocone-axioms-def*]
  |*intro is-cat-coconeI*|
  |*dest is-cat-coconeD*[*dest*!]|
  |*elim is-cat-coconeE*[*elim*!]|

lemma (**in** *is-cat-cocone*) *is-cat-coconeD'*[*cat-cs-intros*]:
  assumes *c'* = *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*
  shows $\mathfrak{N}$: $\mathfrak{F}$ ↦$_{CF}$ *c'* : $\mathfrak{J}$ ↦↦$_{C\alpha}$ $\mathfrak{C}$
  unfolding *assms* **by** (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)

Duality.

lemma (**in** *is-cat-cone*) *is-cat-cocone-op*:
op-ntcf $\mathfrak{N}$: *op-cf* $\mathfrak{F}$ $>_{CF.cocone}$ *c* : *op-cat* $\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-coconeI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)+

lemma (**in** *is-cat-cone*) *is-cat-cocone-op'*[*cat-op-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{J}' =$ *op-cat* $\mathfrak{J}$ **and** $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$ **and** $\mathfrak{F}' =$ *op-cf* $\mathfrak{F}$
shows *op-ntcf* $\mathfrak{N}$: $\mathfrak{F}'$ $>_{CF.cocone}$ *c* : $\mathfrak{J}'$ $\mapsto\!\!\mapsto_{C\alpha'}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-cocone-op*)

lemmas [*cat-op-intros*] = *is-cat-cone.is-cat-cocone-op'*

lemma (**in** *is-cat-cocone*) *is-cat-cone-op*:
op-ntcf $\mathfrak{N}$: *c* $<_{CF.cone}$ *op-cf* $\mathfrak{F}$: *op-cat* $\mathfrak{J}$ $\mapsto\!\!\mapsto_{C\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-cat-coneI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemma (**in** *is-cat-cocone*) *is-cat-cone-op'*[*cat-op-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{J}' =$ *op-cat* $\mathfrak{J}$ **and** $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$ **and** $\mathfrak{F}' =$ *op-cf* $\mathfrak{F}$
shows *op-ntcf* $\mathfrak{N}$: *c* $<_{CF.cone}$ $\mathfrak{F}'$: $\mathfrak{J}'$ $\mapsto\!\!\mapsto_{C\alpha'}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule is-cat-cone-op*)

lemmas [*cat-op-intros*] = *is-cat-cocone.is-cat-cone-op'*

Elementary properties.

lemma (**in** *is-cat-cone*) *cat-cone-LArr-app-is-arr*:
assumes *j* $\in_\circ$ $\mathfrak{J}$(|*Obj*|)
shows $\mathfrak{N}$(|*NTMap*|)(|*j*|) : *c* $\mapsto_{\mathfrak{C}}$ $\mathfrak{F}$(|*ObjMap*|)(|*j*|)
proof–
from *assms* **have** [*simp*]: *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*(|*ObjMap*|)(|*j*|) = *c*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *ntcf-NTMap-is-arr*[*OF assms*] **show** *?thesis* **by** *simp*
qed

lemma (**in** *is-cat-cone*) *cat-cone-LArr-app-is-arr'*[*cat-cs-intros*]:
assumes *j* $\in_\circ$ $\mathfrak{J}$(|*Obj*|) **and** $\mathfrak{F}j = \mathfrak{F}$(|*ObjMap*|)(|*j*|)
shows $\mathfrak{N}$(|*NTMap*|)(|*j*|) : *c* $\mapsto_{\mathfrak{C}}$ $\mathfrak{F}j$
using *assms*(*1*) **unfolding** *assms*(*2*) **by** (*rule cat-cone-LArr-app-is-arr*)

lemmas [*cat-cs-intros*] = *is-cat-cone.cat-cone-LArr-app-is-arr'*

lemma (**in** *is-cat-cocone*) *cat-cocone-LArr-app-is-arr*:
assumes *j* $\in_\circ$ $\mathfrak{J}$(|*Obj*|)
shows $\mathfrak{N}$(|*NTMap*|)(|*j*|) : $\mathfrak{F}$(|*ObjMap*|)(|*j*|) $\mapsto_{\mathfrak{C}}$ *c*
proof–
from *assms* **have** [*simp*]: *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*(|*ObjMap*|)(|*j*|) = *c*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *ntcf-NTMap-is-arr*[*OF assms*] **show** *?thesis* **by** *simp*
qed

lemma (**in** *is-cat-cocone*) *cat-cocone-LArr-app-is-arr'*[*cat-cs-intros*]:

assumes $j \in_{\circ} \mathfrak{J}(\!|Obj|\!)$ **and** $\mathfrak{F}j = \mathfrak{F}(\!|ObjMap|\!)(\!|j|\!)$
shows $\mathfrak{N}(\!|NTMap|\!)(\!|j|\!) : \mathfrak{F}j \mapsto_{\mathfrak{C}} c$
using *assms*(*1*) **unfolding** *assms*(*2*) **by** (*rule cat-cocone-LArr-app-is-arr*)

lemmas [*cat-cs-intros*] = *is-cat-cocone.cat-cocone-LArr-app-is-arr'*

lemma (**in** *is-cat-cone*) *cat-cone-Comp-commute*[*cat-cs-simps*]:
assumes $f : a \mapsto_{\mathfrak{J}} b$
shows $\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{C}} \mathfrak{N}(\!|NTMap|\!)(\!|a|\!) = \mathfrak{N}(\!|NTMap|\!)(\!|b|\!)$
using *ntcf-Comp-commute*[*symmetric*, *OF assms*] *assms*
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

thm *is-cat-cone.cat-cone-Comp-commute*

lemma (**in** *is-cat-cocone*) *cat-cocone-Comp-commute*[*cat-cs-simps*]:
assumes $f : a \mapsto_{\mathfrak{J}} b$
shows $\mathfrak{N}(\!|NTMap|\!)(\!|b|\!) \circ_{A\mathfrak{C}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{N}(\!|NTMap|\!)(\!|a|\!)$
using *ntcf-Comp-commute*[*OF assms*] *assms*
by
(
cs-prems
cs-simp: *cat-cs-simps dghm-const-ArrMap-app* **cs-intro**: *cat-cs-intros*
)

thm *is-cat-cocone.cat-cocone-Comp-commute*

Utilities/helper lemmas.

lemma (**in** *is-cat-cone*) *helper-cat-cone-ntcf-vcomp-Comp*:
assumes $\mathfrak{N}' : c' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $f' : c' \mapsto_{\mathfrak{C}} c$
and $\mathfrak{N}' = \mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f'$
and $j \in_{\circ} \mathfrak{J}(\!|Obj|\!)$
shows $\mathfrak{N}'(\!|NTMap|\!)(\!|j|\!) = \mathfrak{N}(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{C}} f'$
proof–
from *assms*(*3*) **have** $\mathfrak{N}'(\!|NTMap|\!)(\!|j|\!) = (\mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f')(\!|NTMap|\!)(\!|j|\!)$
by *simp*
from *this assms*(*1,2,4*) **show** $\mathfrak{N}'(\!|NTMap|\!)(\!|j|\!) = \mathfrak{N}(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{C}} f'$
by (*cs-prems* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma (**in** *is-cat-cone*) *helper-cat-cone-Comp-ntcf-vcomp*:
assumes $\mathfrak{N}' : c' <_{CF.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $f' : c' \mapsto_{\mathfrak{C}} c$
and $\bigwedge j.\ j \in_{\circ} \mathfrak{J}(\!|Obj|\!) \Longrightarrow \mathfrak{N}'(\!|NTMap|\!)(\!|j|\!) = \mathfrak{N}(\!|NTMap|\!)(\!|j|\!) \circ_{A\mathfrak{C}} f'$
shows $\mathfrak{N}' = \mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f'$
proof–
interpret $\mathfrak{N}'$: *is-cat-cone* $\alpha\ c'\ \mathfrak{J}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{N}'$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*rule ntcf-eqI*[*OF* $\mathfrak{N}'$*.is-ntcf-axioms*])
from *assms*(*2*) **show**
$\mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f' : cf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ c' \mapsto_{CF} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show $\mathfrak{N}'(\!|NTMap|\!) = (\mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f')(\!|NTMap|\!)$
proof(*rule vsv-eqI*, *unfold cat-cs-simps*)
show *vsv* $((\mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f')(\!|NTMap|\!))$
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
from *assms* **show** $\mathfrak{J}(\!|Obj|\!) = \mathcal{D}_{\circ}((\mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{J}\ \mathfrak{C}\ f')(\!|NTMap|\!))$
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

```
    fix j assume prems': j ∈∘ ℑ(|Obj|)
    with assms(1,2) show 𝔑'(|NTMap|)(|j|) = (𝔑 ∙NTCF ntcf-const ℑ ℭ f')(|NTMap|)(|j|)
      by (cs-concl cs-simp: cat-cs-simps assms(3) cs-intro: cat-cs-intros)
  qed auto
 qed simp-all
qed

lemma (in is-cat-cone) helper-cat-cone-Comp-ntcf-vcomp-iff:
 assumes 𝔑' : c' <CF.cone 𝔉 : ℑ ↦↦Cα ℭ
 shows f' : c' ↦ℭ c ∧ 𝔑' = 𝔑 ∙NTCF ntcf-const ℑ ℭ f' ⟷
  f' : c' ↦ℭ c ∧ (∀ j∈∘ℑ(|Obj|). 𝔑'(|NTMap|)(|j|) = 𝔑(|NTMap|)(|j|) ∘A ℭ f')
 using
  helper-cat-cone-ntcf-vcomp-Comp[OF assms]
  helper-cat-cone-Comp-ntcf-vcomp[OF assms]
 by (intro iffI; elim conjE; intro conjI) metis+

lemma (in is-cat-cocone) helper-cat-cocone-ntcf-vcomp-Comp:
 assumes 𝔑' : 𝔉 >CF.cocone c' : ℑ ↦↦Cα ℭ
  and f' : c ↦ℭ c'
  and 𝔑' = ntcf-const ℑ ℭ f' ∙NTCF 𝔑
  and j ∈∘ ℑ(|Obj|)
 shows 𝔑'(|NTMap|)(|j|) = f' ∘A ℭ 𝔑(|NTMap|)(|j|)
proof–
 interpret 𝔑': is-cat-cocone α c' ℑ ℭ 𝔉 𝔑' by (rule assms(1))
 from assms(3) have op-ntcf 𝔑' = op-ntcf (ntcf-const ℑ ℭ f' ∙NTCF 𝔑) by simp
 from this assms(2) have op-𝔑':
  op-ntcf 𝔑' = op-ntcf 𝔑 ∙NTCF ntcf-const (op-cat ℑ) (op-cat ℭ) f'
  by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-op-intros)
 have 𝔑'(|NTMap|)(|j|) = 𝔑(|NTMap|)(|j|) ∘A op-cat ℭ f'
  by
   (
    rule is-cat-cone.helper-cat-cone-ntcf-vcomp-Comp[
     OF is-cat-cone-op 𝔑'.is-cat-cone-op,
     unfolded cat-op-simps,
     OF assms(2) op-𝔑' assms(4)
     ]
   )
 from this assms(2,4) show 𝔑'(|NTMap|)(|j|) = f' ∘A ℭ 𝔑(|NTMap|)(|j|)
  by
   (
    cs-prems cs-shallow
     cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
   )
qed

lemma (in is-cat-cocone) helper-cat-cocone-Comp-ntcf-vcomp:
 assumes 𝔑' : 𝔉 >CF.cocone c' : ℑ ↦↦Cα ℭ
  and f' : c ↦ℭ c'
  and ⋀j. j ∈∘ ℑ(|Obj|) ⟹ 𝔑'(|NTMap|)(|j|) = f' ∘A ℭ 𝔑(|NTMap|)(|j|)
 shows 𝔑' = ntcf-const ℑ ℭ f' ∙NTCF 𝔑
proof–
 interpret 𝔑': is-cat-cocone α c' ℑ ℭ 𝔉 𝔑' by (rule assms(1))
 from assms(2) have 𝔑'j: 𝔑'(|NTMap|)(|j|) = 𝔑(|NTMap|)(|j|) ∘A op-cat ℭ f'
  if j ∈∘ ℑ(|Obj|) for j
  using that
  unfolding assms(3)[OF that]
  by
   (
```

```
      cs-concl cs-shallow
        cs-simp: cat-op-simps cat-cs-simps cs-intro: cat-cs-intros
    )
  have op-𝔑': op-ntcf 𝔑' = op-ntcf 𝔑 ⊙NTCF ntcf-const (op-cat 𝔍) (op-cat ℭ) f'
    by
      (
      rule is-cat-cone.helper-cat-cone-Comp-ntcf-vcomp[
        OF is-cat-cone-op 𝔑'.is-cat-cone-op,
        unfolded cat-op-simps,
        OF assms(2) 𝔑'j,
        simplified
        ]
      )
  from assms(2) show 𝔑' = (ntcf-const 𝔍 ℭ f' ⊙NTCF 𝔑)
    by
      (
      cs-concl
        cs-simp:
          cat-op-simps op-𝔑' eq-op-ntcf-iff[symmetric, OF 𝔑'.is-ntcf-axioms]
        cs-intro: cat-cs-intros
      )
qed

lemma (in is-cat-cocone) helper-cat-cocone-Comp-ntcf-vcomp-iff:
  assumes 𝔑' : 𝔉 >CF.cocone c' : 𝔍 ↦↦Cα ℭ
  shows f' : c ↦ℭ c' ∧ 𝔑' = ntcf-const 𝔍 ℭ f' ⊙NTCF 𝔑 ⟷
    f' : c ↦ℭ c' ∧ (∀ j∈∘𝔍(|Obj|). 𝔑'(|NTMap|)(|j|) = f' ∘AℭA 𝔑(|NTMap|)(|j|))
  using
    helper-cat-cocone-ntcf-vcomp-Comp[OF assms]
    helper-cat-cocone-Comp-ntcf-vcomp[OF assms]
  by (intro iffI; elim conjE; intro conjI) metis+
```

27.1.2 Vertical composition of a natural transformation and a cone

```
lemma ntcf-vcomp-is-cat-cone[cat-cs-intros]:
  assumes 𝔐 : 𝔊 ↦CF 𝔎 : 𝔄 ↦↦Cα 𝔅
    and 𝔑 : a <CF.cone 𝔊 : 𝔄 ↦↦Cα 𝔅
  shows 𝔐 ⊙NTCF 𝔑 : a <CF.cone 𝔎 : 𝔄 ↦↦Cα 𝔅
  by
    (
      intro is-cat-coneI ntcf-vcomp-is-ntcf, rule assms(1);
      rule is-cat-coneD[OF assms(2)]
    )
```

27.1.3 Composition of a functor and a cone, composition of a functor and a cocone

```
lemma cf-ntcf-comp-cf-cat-cone:
  assumes 𝔑 : c <CF.cone 𝔉 : 𝔄 ↦↦Cα 𝔅 and 𝔊 : 𝔅 ↦↦Cα ℭ
  shows 𝔊 ∘CF−NTCF 𝔑 : 𝔊(|ObjMap|)(|c|) <CF.cone 𝔊 ∘CF 𝔉 : 𝔄 ↦↦Cα ℭ
proof–
  interpret 𝔑: is-cat-cone α c 𝔄 𝔅 𝔉 𝔑 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  show ?thesis
    by (intro is-cat-coneI)
      (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros )+
qed
```

lemma *cf-ntcf-comp-cf-cat-cone'*[*cat-cs-intros*]:
  assumes $\mathfrak{N} : c <_{CF.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $c' = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)$
    and $\mathfrak{G}\mathfrak{F} = \mathfrak{G} \circ_{CF} \mathfrak{F}$
  shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : c' <_{CF.cone} \mathfrak{G}\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*rule cf-ntcf-comp-cf-cat-cone*)

lemma *cf-ntcf-comp-cf-cat-cocone*:
  assumes $\mathfrak{N} : \mathfrak{F} >_{CF.cocone} c : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{G} \circ_{CF} \mathfrak{F} >_{CF.cocone} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor) : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–
  interpret $\mathfrak{N}$: *is-cat-cocone* α c $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
  show *?thesis*
    by
      (
        rule is-cat-cone.is-cat-cocone-op
          [
            OF cf-ntcf-comp-cf-cat-cone[
              OF $\mathfrak{N}$.*is-cat-cone-op* $\mathfrak{G}$.*is-functor-op*, *unfolded cat-op-simps*
              ],
            unfolded cat-op-simps
          ]
)
qed

lemma *cf-ntcf-comp-cf-cat-cocone'*[*cat-cs-intros*]:
  assumes $\mathfrak{N} : \mathfrak{F} >_{CF.cocone} c : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $c' = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)$
    and $\mathfrak{G}\mathfrak{F} = \mathfrak{G} \circ_{CF} \mathfrak{F}$
  shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{G}\mathfrak{F} >_{CF.cocone} c' : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*rule cf-ntcf-comp-cf-cat-cocone*)

27.1.4 Cones, cocones and constant natural transformations

lemma *ntcf-vcomp-ntcf-const-is-cat-cone*:
  assumes $\mathfrak{N} : b <_{CF.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $f : a \mapsto_{\mathfrak{B}} b$
  shows $\mathfrak{N} \bullet_{NTCF}$ *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ $f : a <_{CF.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof–
  interpret $\mathfrak{N}$: *is-cat-cone* α b $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
  from *assms*(*2*) **show** *?thesis*
    by (*intro is-cat-coneI*) (*cs-concl* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-vcomp-ntcf-const-is-cat-cone'*[*cat-cs-intros*]:
  assumes $\mathfrak{N} : b <_{CF.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
    and $\mathfrak{M} =$ *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ f
    and $f : a \mapsto_{\mathfrak{B}} b$
  shows $\mathfrak{N} \bullet_{NTCF} \mathfrak{M} : a <_{CF.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
  using *assms*(*1*,*3*) **unfolding** *assms*(*2*) **by** (*rule ntcf-vcomp-ntcf-const-is-cat-cone*)

lemma *ntcf-vcomp-ntcf-const-is-cat-cocone*:
  assumes $\mathfrak{N} : \mathfrak{F} >_{CF.cocone} a : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$ **and** $f : a \mapsto_{\mathfrak{B}} b$
  shows *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ $f \bullet_{NTCF} \mathfrak{N} : \mathfrak{F} >_{CF.cocone} b : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{B}$
proof–
  interpret $\mathfrak{N}$: *is-cat-cocone* α a $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))

```
  from is-cat-cone.is-cat-cocone-op
    [
      OF ntcf-vcomp-ntcf-const-is-cat-cone[
        OF 𝔑.is-cat-cone-op, unfolded cat-op-simps, OF assms(2)
        ],
      unfolded cat-op-simps,
      folded op-ntcf-ntcf-const
    ]
    assms(2)
  show ?thesis
    by (cs-prems cs-simp: cat-op-simps cs-intro: cat-cs-intros cat-op-intros)
qed

lemma ntcf-vcomp-ntcf-const-is-cat-cocone'[cat-cs-intros]:
  assumes 𝔑 : 𝔉 >CF.cocone a : 𝔄 ↦↦Cα 𝔅
    and 𝔐 = ntcf-const 𝔄 𝔅 f
    and f : a ↦𝔅 b
  shows 𝔐 ∘NTCF 𝔑 : 𝔉 >CF.cocone b : 𝔄 ↦↦Cα 𝔅
  using assms(1,3)
  unfolding assms(2)
  by (rule ntcf-vcomp-ntcf-const-is-cat-cocone)

lemma ntcf-vcomp-ntcf-const-CId:
  assumes 𝔑 : b <CF.cone 𝔉 : 𝔄 ↦↦Cα 𝔅
  shows 𝔑 ∘NTCF ntcf-const 𝔄 𝔅 (𝔅(|CId|)(|b|)) = 𝔑
proof–

  interpret 𝔑: is-cat-cone α b 𝔄 𝔅 𝔉 𝔑 by (rule assms)

  show ?thesis
  proof(rule ntcf-eqI)

    from 𝔑.cat-cone-obj show lhs-is-ntcf:
      𝔑 ∘NTCF ntcf-const 𝔄 𝔅 (𝔅(|CId|)(|b|)) :
        cf-const 𝔄 𝔅 b ↦CF 𝔉 : 𝔄 ↦↦Cα 𝔅
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    then have dom-lhs:
      𝒟∘ ((𝔑 ∘NTCF ntcf-const 𝔄 𝔅 (𝔅(|CId|)(|b|)))(|NTMap|)) = 𝔄(|Obj|)
      by (simp add: cat-cs-simps)

    from 𝔑.cat-cone-obj show 𝔑 : cf-const 𝔄 𝔅 b ↦CF 𝔉 : 𝔄 ↦↦Cα 𝔅
      by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    then have dom-rhs: 𝒟∘ (𝔑(|NTMap|)) = 𝔄(|Obj|)
      by (simp add: cat-cs-simps)

    show (𝔑 ∘NTCF ntcf-const 𝔄 𝔅 (𝔅(|CId|)(|b|)))(|NTMap|) = 𝔑(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume prems: a ∈∘ 𝔄(|Obj|)
      from prems 𝔑.cat-cone-obj show
        (𝔑 ∘NTCF ntcf-const 𝔄 𝔅 (𝔅(|CId|)(|b|)))(|NTMap|)(|a|) = 𝔑(|NTMap|)(|a|)
        by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    qed (use lhs-is-ntcf in ‹cs-concl cs-intro: cat-cs-intros›)+

  qed simp-all

qed

lemma ntcf-vcomp-ntcf-const-CId'[cat-cs-simps]:
```

assumes $\mathfrak{N} : b <_{CF.cone} \mathfrak{F} : \mathfrak{A} \mapsto\!\!\!\mapsto_{C\alpha} \mathfrak{B}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{N} \cdot_{NTCF} ntcf\text{-}const\ \mathfrak{A}\ \mathfrak{B}\ (\mathfrak{B}'(\!|CId|\!)(\!|b|\!)) = \mathfrak{N}$
using *assms*(*1*) **unfolding** *assms*(*2*) **by** (*rule ntcf-vcomp-ntcf-const-CId*)

27.2 Cone and cocone functors

27.2.1 Definition and elementary properties

See Chapter V-1 in [7].

definition *cf-Cone* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-Cone* $\alpha\ \beta\ \mathfrak{F}$ =
$Hom_{O.C\beta} cat\text{-}FUNCT\ \alpha\ (\mathfrak{F}(\!|HomDom|\!))\ (\mathfrak{F}(\!|HomCod|\!))(-, cf\text{-}map\ \mathfrak{F}) \circ_{CF}$
$op\text{-}cf\ (\Delta_{CF}\ \alpha\ (\mathfrak{F}(\!|HomDom|\!))\ (\mathfrak{F}(\!|HomCod|\!)))$

definition *cf-Cocone* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-Cocone* $\alpha\ \beta\ \mathfrak{F}$ =
$Hom_{O.C\beta} cat\text{-}FUNCT\ \alpha\ (\mathfrak{F}(\!|HomDom|\!))\ (\mathfrak{F}(\!|HomCod|\!))(cf\text{-}map\ \mathfrak{F}, -) \circ_{CF}$
$(\Delta_{CF}\ \alpha\ (\mathfrak{F}(\!|HomDom|\!))\ (\mathfrak{F}(\!|HomCod|\!)))$

An alternative form of the definition.

context *is-functor*
begin

lemma *cf-Cone-def'*:
cf-Cone $\alpha\ \beta\ \mathfrak{F}$ = $Hom_{O.C\beta} cat\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}(-, cf\text{-}map\ \mathfrak{F}) \circ_{CF} op\text{-}cf\ (\Delta_{CF}\ \alpha\ \mathfrak{A}\ \mathfrak{B})$
unfolding *cf-Cone-def cat-cs-simps* **by** *simp*

lemma *cf-Cocone-def'*:
cf-Cocone $\alpha\ \beta\ \mathfrak{F}$ = $Hom_{O.C\beta} cat\text{-}FUNCT\ \alpha\ \mathfrak{A}\ \mathfrak{B}(cf\text{-}map\ \mathfrak{F}, -) \circ_{CF} (\Delta_{CF}\ \alpha\ \mathfrak{A}\ \mathfrak{B})$
unfolding *cf-Cocone-def cat-cs-simps* **by** *simp*

end

27.2.2 Object map

lemma (**in** *is-functor*) *cf-Cone-ObjMap-vsv*[*cat-cs-intros*]:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows *vsv* (*cf-Cone* $\alpha\ \beta\ \mathfrak{F}(\!|ObjMap|\!)$)
proof–
from *assms* **interpret** β: $\mathcal{Z}\ \beta$ **by** *simp*
from *assms* **interpret** Δ: *is-functor* $\beta\ \mathfrak{B}$ ‹*cat-FUNCT* $\alpha\ \mathfrak{A}\ \mathfrak{B}$› ‹$\Delta_{CF}\ \alpha\ \mathfrak{A}\ \mathfrak{B}$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-op-intros*)+
from *assms*(*2*) **show** *?thesis*
unfolding *cf-Cone-def'*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-FUNCT-components*(*1*) *cat-op-simps*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros cat-op-intros*
)
qed

lemmas [*cat-cs-intros*] = *is-functor.cf-Cone-ObjMap-vsv*

lemma (**in** *is-functor*) *cf-Cocone-ObjMap-vsv*[*cat-cs-intros*]:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows *vsv* (*cf-Cocone* $\alpha\ \beta\ \mathfrak{F}(\!|ObjMap|\!)$)
proof–

```
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2) show ?thesis
    unfolding cf-Cocone-def'
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-intros] = is-functor.cf-Cocone-ObjMap-vsv

lemma (in is-functor) cf-Cone-ObjMap-vdomain[cat-cs-simps]:
  assumes 𝒵 β and α ∈₀ β and b ∈₀ 𝔅(|Obj|)
  shows 𝒟₀ (cf-Cone α β ℱ(|ObjMap|)) = 𝔅(|Obj|)
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms show ?thesis
    unfolding cf-Cone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cone-ObjMap-vdomain

lemma (in is-functor) cf-Cocone-ObjMap-vdomain[cat-cs-simps]:
  assumes 𝒵 β and α ∈₀ β and b ∈₀ 𝔅(|Obj|)
  shows 𝒟₀ (cf-Cocone α β ℱ(|ObjMap|)) = 𝔅(|Obj|)
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms show ?thesis
    unfolding cf-Cocone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cocone-ObjMap-vdomain

lemma (in is-functor) cf-Cone-ObjMap-app[cat-cs-simps]:
  assumes 𝒵 β and α ∈₀ β and b ∈₀ 𝔅(|Obj|)
  shows cf-Cone α β ℱ(|ObjMap|)(|b|) =
    Hom (cat-FUNCT α 𝔄 𝔅) (cf-map (cf-const 𝔄 𝔅 b)) (cf-map ℱ)
proof–
```

```
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2,3) show ?thesis
    unfolding cf-Cone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cone-ObjMap-app

lemma (in is-functor) cf-Cocone-ObjMap-app[cat-cs-simps]:
  assumes 𝒵 β and α ∈₀ β and b ∈₀ 𝔅(|Obj|)
  shows cf-Cocone α β 𝔉(|ObjMap|)(|b|) =
    Hom (cat-FUNCT α 𝔄 𝔅) (cf-map 𝔉) (cf-map (cf-const 𝔄 𝔅 b))
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2,3) show ?thesis
    unfolding cf-Cocone-def'
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cocone-ObjMap-app
```

27.2.3 Arrow map

```
lemma (in is-functor) cf-Cone-ArrMap-vsv[cat-cs-intros]:
  assumes 𝒵 β and α ∈₀ β
  shows vsv (cf-Cone α β 𝔉(|ArrMap|))
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2) show ?thesis
    unfolding cf-Cone-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-intros] = is-functor.cf-Cone-ArrMap-vsv

lemma (in is-functor) cf-Cocone-ArrMap-vsv[cat-cs-intros]:
  assumes 𝒵 β and α ∈₀ β
```

```
  shows vsv (cf-Cocone α β 𝔉(|ArrMap|))
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2) show ?thesis
    unfolding cf-Cocone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-intros] = is-functor.cf-Cocone-ArrMap-vsv

lemma (in is-functor) cf-Cone-ArrMap-vdomain[cat-cs-simps]:
  assumes 𝒵 β and α ∈ₒ β and b ∈ₒ 𝔅(|Obj|)
  shows 𝒟ₒ (cf-Cone α β 𝔉(|ArrMap|)) = 𝔅(|Arr|)
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2) show ?thesis
    unfolding cf-Cone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cone-ArrMap-vdomain

lemma (in is-functor) cf-Cocone-ArrMap-vdomain[cat-cs-simps]:
  assumes 𝒵 β and α ∈ₒ β and b ∈ₒ 𝔅(|Obj|)
  shows 𝒟ₒ (cf-Cocone α β 𝔉(|ArrMap|)) = 𝔅(|Arr|)
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2) show ?thesis
    unfolding cf-Cocone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cocone-ArrMap-vdomain

lemma (in is-functor) cf-Cone-ArrMap-app[cat-cs-simps]:
  assumes 𝒵 β
    and α ∈ₒ β
```

```
    and f : a ↦_𝔅 b
  shows cf-Cone α β 𝔉(|ArrMap|)(|f|) = cf-hom
    (cat-FUNCT α 𝔄 𝔅)
    [ntcf-arrow (ntcf-const 𝔄 𝔅 f), cat-FUNCT α 𝔄 𝔅(|CId|)(|cf-map 𝔉|)]∘
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2,3) show ?thesis
    unfolding cf-Cone-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cone-ArrMap-app

lemma (in is-functor) cf-Cocone-ArrMap-app[cat-cs-simps]:
  assumes 𝒵 β
    and α ∈∘ β
    and f : a ↦_𝔅 b
  shows cf-Cocone α β 𝔉(|ArrMap|)(|f|) = cf-hom
    (cat-FUNCT α 𝔄 𝔅)
    [cat-FUNCT α 𝔄 𝔅(|CId|)(|cf-map 𝔉|), ntcf-arrow (ntcf-const 𝔄 𝔅 f)]∘
proof–
  from assms interpret β: 𝒵 β by simp
  from assms interpret Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_CF α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  from assms(2,3) show ?thesis
    unfolding cf-Cocone-def'
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-components(1) cat-op-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-Cocone-ArrMap-app
```

27.2.4 The cone functor is a functor

```
lemma (in is-functor) tm-cf-cf-Cone-is-functor-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows cf-Cone α β 𝔉 : op-cat 𝔅 ↦↦_Cβ cat-Set β
proof–
  from assms interpret FUNCT: category β ‹cat-FUNCT α 𝔄 𝔅›
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from assms interpret op-Δ:
    is-functor β ‹op-cat 𝔅› ‹op-cat (cat-FUNCT α 𝔄 𝔅)› ‹op-cf (Δ_CF α 𝔄 𝔅)›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
```

```
  have Hom_{O.Cβ} cat-FUNCT α 𝔄 𝔅(−,cf-map 𝔉) :
    op-cat (cat-FUNCT α 𝔄 𝔅) ↦↦_{Cβ} cat-Set β
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  then show cf-Cone α β 𝔉 : op-cat 𝔅 ↦↦_{Cβ} cat-Set β
    unfolding cf-Cone-def′ by (cs-concl cs-intro: cat-cs-intros)
qed

lemma (in is-functor) tm-cf-cf-Cocone-is-functor-if-ge-Limit:
  assumes 𝒵 β and α ∈₀ β
  shows cf-Cocone α β 𝔉 : 𝔅 ↦↦_{Cβ} cat-Set β
proof–
  from assms interpret Funct: category β ‹cat-FUNCT α 𝔄 𝔅›
    by
      (
        cs-concl cs-intro:
          cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  from assms interpret op-Δ: is-functor β 𝔅 ‹cat-FUNCT α 𝔄 𝔅› ‹Δ_{CF} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)+
  have Hom_{O.Cβ} cat-FUNCT α 𝔄 𝔅(cf-map 𝔉,−) :
    cat-FUNCT α 𝔄 𝔅 ↦↦_{Cβ} cat-Set β
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  then show cf-Cocone α β  𝔉 : 𝔅 ↦↦_{Cβ} cat-Set β
    unfolding cf-Cocone-def′ by (cs-concl cs-intro: cat-cs-intros)
qed
```

28 Smallness for cones and cocones

28.1 Cone with tiny maps and cocone with tiny maps

28.1.1 Definition and elementary properties

locale *is-tm-cat-cone* =
  is-ntcf α $\mathfrak{J}$ $\mathfrak{C}$ ‹*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ c› $\mathfrak{F}$ $\mathfrak{N}$ + *NTCod*: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$
  for α c $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$ +
  assumes *tm-cat-cone-obj*[*cat-cs-intros*, *cat-small-cs-intros*]: $c \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$

syntax *-is-tm-cat-cone* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  (‹(- :/ - $<_{CF.tm.cone}$ - :/ - $\mapsto\mapsto_{C.tm1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-tm-cat-cone* $\rightleftharpoons$ *is-tm-cat-cone*
translations $\mathfrak{N} : c <_{CF.tm.cone} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-tm-cat-cone α c $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$

locale *is-tm-cat-cocone* =
  is-ntcf α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ ‹*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ c› $\mathfrak{N}$ + *NTDom*: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$
  for α c $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$ +
  assumes *tm-cat-cocone-obj*[*cat-cs-intros*, *cat-small-cs-intros*]: $c \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$

syntax *-is-tm-cat-cocone* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  (‹(- :/ - $>_{CF.tm.cocone}$ - :/ - $\mapsto\mapsto_{C.tm1}$ -)› [*51*, *51*, *51*, *51*, *51*] *51*)
syntax-consts *-is-tm-cat-cocone* $\rightleftharpoons$ *is-tm-cat-cocone*
translations $\mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} c : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C} \rightleftharpoons$
  CONST is-tm-cat-cocone α c $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{N}$

Rules.

lemma (**in** *is-tm-cat-cone*) *is-tm-cat-cone-axioms'*[
    cat-cs-intros, *cat-small-cs-intros*
    ]:
  assumes $\alpha' = \alpha$ **and** $c' = c$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
  shows $\mathfrak{N} : c' <_{CF.tm.cone} \mathfrak{F}' : \mathfrak{J}' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-tm-cat-cone-axioms*)

mk-ide rf *is-tm-cat-cone-def*[*unfolded is-tm-cat-cone-axioms-def*]
  |*intro is-tm-cat-coneI*|
  |*dest is-tm-cat-coneD*[*dest*!]|
  |*elim is-tm-cat-coneE*[*elim*!]|

lemma (**in** *is-tm-cat-cocone*) *is-tm-cat-cocone-axioms'*[
    cat-cs-intros, *cat-small-cs-intros*
    ]:
  assumes $\alpha' = \alpha$ **and** $c' = c$ **and** $\mathfrak{J}' = \mathfrak{J}$ **and** $\mathfrak{C}' = \mathfrak{C}$ **and** $\mathfrak{F}' = \mathfrak{F}$
  shows $\mathfrak{N} : \mathfrak{F}' >_{CF.tm.cocone} c' : \mathfrak{J}' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule is-tm-cat-cocone-axioms*)

mk-ide rf *is-tm-cat-cocone-def*[*unfolded is-tm-cat-cocone-axioms-def*]
  |*intro is-tm-cat-coconeI*|
  |*dest is-tm-cat-coconeD*[*dest*!]|
  |*elim is-tm-cat-coconeE*[*elim*!]|

Duality.

lemma (**in** *is-tm-cat-cone*) *is-tm-cat-cocone-op*:
  op-ntcf $\mathfrak{N}$: *op-cf* $\mathfrak{F}$ $>_{CF.tm.cocone}$ c : *op-cat* $\mathfrak{J}$ $\mapsto\mapsto_{C.tm\alpha}$ *op-cat* $\mathfrak{C}$
  by (*intro is-tm-cat-coconeI*)
    (
      cs-concl **cs-shallow**

cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)+

lemma (**in** *is-tm-cat-cone*) *is-tm-cat-cocone-op'*[*cat-op-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{J}' =$ *op-cat* $\mathfrak{J}$ **and** $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$ **and** $\mathfrak{F}' =$ *op-cf* $\mathfrak{F}$
shows *op-ntcf* $\mathfrak{N} : \mathfrak{F}' >_{CF.tm.cocone} c : \mathfrak{J}' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-cocone-op*)

lemmas [*cat-op-intros*] = *is-tm-cat-cone.is-tm-cat-cocone-op'*

lemma (**in** *is-tm-cat-cocone*) *is-tm-cat-cone-op*:
op-ntcf $\mathfrak{N} : c <_{CF.tm.cone}$ *op-cf* $\mathfrak{F} :$ *op-cat* $\mathfrak{J} \mapsto\mapsto_{C.tm\alpha}$ *op-cat* $\mathfrak{C}$
by (*intro is-tm-cat-coneI*)
(
cs-concl **cs-shallow**
cs-simp: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)

lemma (**in** *is-tm-cat-cocone*) *is-tm-cat-cone-op'*[*cat-op-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{J}' =$ *op-cat* $\mathfrak{J}$ **and** $\mathfrak{C}' =$ *op-cat* $\mathfrak{C}$ **and** $\mathfrak{F}' =$ *op-cf* $\mathfrak{F}$
shows *op-ntcf* $\mathfrak{N} : c <_{CF.tm.cone} \mathfrak{F}' : \mathfrak{J}' \mapsto\mapsto_{C.tm_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule is-tm-cat-cone-op*)

lemmas [*cat-op-intros*] = *is-cat-cocone.is-cat-cone-op'*

Elementary properties.

lemma (**in** *is-tm-cat-cone*) *tm-cat-cone-is-tm-ntcf'*[
cat-cs-intros, *cat-small-cs-intros*
]:
assumes $c' =$ *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ c
shows $\mathfrak{N} : c' \mapsto_{CF.tm} \mathfrak{F} : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
unfolding *assms*
proof(*intro is-tm-ntcfI'*)
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule NTCod.is-tm-functor-axioms*)
show *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ $c : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros*)
qed (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-cs-intros assms*)+

lemmas [*cat-small-cs-intros*] = *is-tm-cat-cone.tm-cat-cone-is-tm-ntcf'*

sublocale *is-tm-cat-cone* $\subseteq$ *is-tm-ntcf* α $\mathfrak{J}$ $\mathfrak{C}$ ‹*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ c› $\mathfrak{F}$ $\mathfrak{N}$
by (*intro tm-cat-cone-is-tm-ntcf'*) *simp*

lemma (**in** *is-tm-cat-cocone*) *tm-cat-cocone-is-tm-ntcf'*[
cat-cs-intros, *cat-small-cs-intros*
]:
assumes $c' =$ *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ c
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{CF.tm} c' : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
unfolding *assms*
proof(*intro is-tm-ntcfI'*)
interpret $\mathfrak{F}$: *is-tm-functor* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule NTDom.is-tm-functor-axioms*)
show *cf-const* $\mathfrak{J}$ $\mathfrak{C}$ $c : \mathfrak{J} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros*)
qed (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-cs-intros assms*)+

lemmas [*cat-small-cs-intros*, *cat-cs-intros*] =
is-tm-cat-cocone.tm-cat-cocone-is-tm-ntcf'

sublocale *is-tm-cat-cocone* ⊆ *is-tm-ntcf* α $\mathfrak{J}$ $\mathfrak{C}$ $\mathfrak{F}$ ‹*cf-const* $\mathfrak{J}$ $\mathfrak{C}$ *c*› $\mathfrak{N}$
by (*intro tm-cat-cocone-is-tm-ntcf'*) *simp*

sublocale *is-tm-cat-cone* ⊆ *is-cat-cone*
by (*intro is-cat-coneI*, *rule is-ntcf-axioms*, *rule tm-cat-cone-obj*)

lemmas (**in** *is-tm-cat-cone*) *tm-cat-cone-is-cat-cone* = *is-cat-cone-axioms*
lemmas [*cat-small-cs-intros*] = *is-tm-cat-cone.tm-cat-cone-is-cat-cone*

sublocale *is-tm-cat-cocone* ⊆ *is-cat-cocone*
by (*intro is-cat-coconeI*, *rule is-ntcf-axioms*, *rule tm-cat-cocone-obj*)

lemmas (**in** *is-tm-cat-cocone*) *tm-cat-cocone-is-cat-cocone* = *is-cat-cocone-axioms*
lemmas [*cat-small-cs-intros*] = *is-tm-cat-cocone.tm-cat-cocone-is-cat-cocone*

28.1.2 Vertical composition of a natural transformation with tiny maps and a cone with tiny maps

lemma *ntcf-vcomp-is-tm-cat-cone*[*cat-cs-intros*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{CF.tm} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{N} : a <_{CF.tm.cone} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
shows $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} : a <_{CF.tm.cone} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
by
(
intro is-tm-cat-coneI ntcf-vcomp-is-ntcf;
(*rule is-tm-ntcfD'*[*OF assms*(*1*)])*?*;
(*intro is-tm-cat-coneD*[*OF assms*(*2*)])*?*
)

28.1.3 Composition of a functor and a cone with tiny maps, composition of a functor and a cocone with tiny maps

lemma *cf-ntcf-comp-tm-cf-tm-cat-cone*:
assumes $\mathfrak{N} : c <_{CF.tm.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor) <_{CF.tm.cone} \mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-tm-cat-cone* α *c* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{G}\mathfrak{F}$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{C}$ ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› **by** (*rule assms*(*3*))
show *?thesis*
by (*intro is-tm-cat-coneI*)
(*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros is-cat-coneD*)+
qed

lemma *cf-ntcf-comp-tm-cf-tm-cat-cone'*[*cat-small-cs-intros*]:
assumes $\mathfrak{N} : c <_{CF.tm.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and $c' = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)$
and $\mathfrak{G}\mathfrak{F} = \mathfrak{G} \circ_{CF} \mathfrak{F}$
shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : c' <_{CF.tm.cone} \mathfrak{G}\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
using *assms*(*1*,*2*,*3*)
unfolding *assms*(*4*,*5*)
by (*rule cf-ntcf-comp-tm-cf-tm-cat-cone*)

lemma *cf-ntcf-comp-tm-cf-tm-cat-cocone*:

assumes $\mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} c : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{G} \circ_{CF} \mathfrak{F} >_{CF.tm.cocone} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor) : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-tm-cat-cocone* α c $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{GF}$: *is-tm-functor* α $\mathfrak{A}$ $\mathfrak{C}$ ‹$\mathfrak{G} \circ_{CF} \mathfrak{F}$› **by** (*rule assms*(*3*))
show *?thesis*
by
(
rule is-tm-cat-cone.is-tm-cat-cocone-op
[
OF cf-ntcf-comp-tm-cf-tm-cat-cone[
OF $\mathfrak{N}$.*is-tm-cat-cone-op* $\mathfrak{G}$.*is-functor-op*, *unfolded cat-op-simps*
],
OF $\mathfrak{GF}$.*is-tm-functor-op*[*unfolded cat-op-simps*],
unfolded cat-op-simps
]
)
qed

lemma *cf-ntcf-comp-tm-cf-tm-cat-cocone'*[*cat-small-cs-intros*]:
assumes $\mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} c : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
and $c' = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor c \rfloor)$
and $\mathfrak{GF} = \mathfrak{G} \circ_{CF} \mathfrak{F}$
shows $\mathfrak{G} \circ_{CF-NTCF} \mathfrak{N} : \mathfrak{GF} >_{CF.tm.cocone} c' : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{C}$
using *assms*(*1–3*)
unfolding *assms*(*4,5*)
by (*rule cf-ntcf-comp-tm-cf-tm-cat-cocone*)

28.1.4 Cones and cocones with tiny maps and constant natural transformations

lemma *ntcf-vcomp-ntcf-const-is-tm-cat-cone*:
assumes $\mathfrak{N} : b <_{CF.tm.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$ **and** $f : a \mapsto_{\mathfrak{B}} b$
shows $\mathfrak{N} \bullet_{NTCF}$ *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ $f : a <_{CF.tm.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{N}$: *is-tm-cat-cone* α b $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
from *assms*(*2*) **show** *?thesis*
by (*intro is-tm-cat-coneI*)
(*cs-concl* **cs-intro**: *cat-small-cs-intros cat-cs-intros*)
qed

lemma *ntcf-vcomp-ntcf-const-is-tm-cat-cone'*[*cat-small-cs-intros*]:
assumes $\mathfrak{N} : b <_{CF.tm.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{M} =$ *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ f
and $f : a \mapsto_{\mathfrak{B}} b$
shows $\mathfrak{N} \bullet_{NTCF} \mathfrak{M} : a <_{CF.tm.cone} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
using *assms*(*1,3*)
unfolding *assms*(*2*)
by (*rule ntcf-vcomp-ntcf-const-is-tm-cat-cone*)

lemma *ntcf-vcomp-ntcf-const-is-tm-cat-cocone*:
assumes $\mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} a : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$ **and** $f : a \mapsto_{\mathfrak{B}} b$
shows *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ $f \bullet_{NTCF} \mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} b : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
proof–

interpret $\mathfrak{N}$: *is-tm-cat-cocone* α a $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{N}$ **by** (*rule assms*(*1*))
from *is-tm-cat-cone.is-tm-cat-cocone-op*
[
OF ntcf-vcomp-ntcf-const-is-tm-cat-cone[
OF $\mathfrak{N}$.*is-tm-cat-cone-op*, *unfolded cat-op-simps*, *OF assms*(*2*)
],
unfolded cat-op-simps,
folded op-ntcf-ntcf-const
]
assms(*2*)
show *?thesis*
by (*cs-prems* **cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros cat-op-intros*)
qed

lemma *ntcf-vcomp-ntcf-const-is-tm-cat-cocone'*[*cat-cs-intros*]:
assumes $\mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} a : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
and $\mathfrak{M}$ = *ntcf-const* $\mathfrak{A}$ $\mathfrak{B}$ f
and $f : a \mapsto_{\mathfrak{B}} b$
shows $\mathfrak{M} \bullet_{NTCF} \mathfrak{N} : \mathfrak{F} >_{CF.tm.cocone} b : \mathfrak{A} \mapsto\mapsto_{C.tm\alpha} \mathfrak{B}$
using *assms*(*1*,*3*)
unfolding *assms*(*2*)
by (*rule ntcf-vcomp-ntcf-const-is-tm-cat-cocone*)

28.2 Small cone and small cocone functors

28.2.1 Definition and elementary properties

definition *tm-cf-Cone* :: $V \Rightarrow V \Rightarrow V$
where *tm-cf-Cone* α $\mathfrak{F}$ =
$Hom_{O.C\alpha}$*cat-Funct* α ($\mathfrak{F}$(|*HomDom*|)) ($\mathfrak{F}$(|*HomCod*|))(−,*cf-map* $\mathfrak{F}$) $\circ_{CF}$
op-cf ($\Delta_{CF.tm}$ α ($\mathfrak{F}$(|*HomDom*|)) ($\mathfrak{F}$(|*HomCod*|)))

definition *tm-cf-Cocone* :: $V \Rightarrow V \Rightarrow V$
where *tm-cf-Cocone* α $\mathfrak{F}$ =
$Hom_{O.C\alpha}$*cat-Funct* α ($\mathfrak{F}$(|*HomDom*|)) ($\mathfrak{F}$(|*HomCod*|))(*cf-map* $\mathfrak{F}$,−) $\circ_{CF}$
($\Delta_{CF.tm}$ α ($\mathfrak{F}$(|*HomDom*|)) ($\mathfrak{F}$(|*HomCod*|)))

Alternative definitions.

context *is-tm-functor*
begin

lemma *tm-cf-Cone-def'*:
tm-cf-Cone α $\mathfrak{F}$ =
$Hom_{O.C\alpha}$*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(−,*cf-map* $\mathfrak{F}$) $\circ_{CF}$ *op-cf* ($\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$)
unfolding *tm-cf-Cone-def cat-cs-simps* **by** *simp*

lemma *tm-cf-Cocone-def'*:
tm-cf-Cocone α $\mathfrak{F}$ =
$Hom_{O.C\alpha}$*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(*cf-map* $\mathfrak{F}$,−) $\circ_{CF}$ ($\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$)
unfolding *tm-cf-Cocone-def cat-cs-simps* **by** *simp*

end

28.2.2 Object map

lemma (**in** *is-tm-functor*) *tm-cf-Cone-ObjMap-vsv*[*cat-small-cs-intros*]:
vsv (*tm-cf-Cone* α $\mathfrak{F}$(|*ObjMap*|))
proof–
interpret Δ: *is-functor* α $\mathfrak{B}$ ‹*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$› ‹$\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$›

```
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  show ?thesis
    unfolding tm-cf-Cone-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-small-cs-intros] = is-tm-functor.tm-cf-Cone-ObjMap-vsv

lemma (in is-tm-functor) tm-cf-Cocone-ObjMap-vsv[cat-small-cs-intros]:
  vsv (tm-cf-Cocone α 𝔉(|ObjMap|))
proof–
  interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_{CF.tm} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  show ?thesis
    unfolding tm-cf-Cocone-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
       )
qed

lemmas [cat-small-cs-intros] = is-tm-functor.tm-cf-Cocone-ObjMap-vsv

lemma (in is-tm-functor) tm-cf-Cone-ObjMap-vdomain[cat-small-cs-simps]:
  assumes b ∈∘ 𝔅(|Obj|)
  shows 𝒟∘ (tm-cf-Cone α 𝔉(|ObjMap|)) = 𝔅(|Obj|)
proof–
  from assms interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_{CF.tm} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-small-cs-simps] = is-tm-functor.tm-cf-Cone-ObjMap-vdomain

lemma (in is-tm-functor) tm-cf-Cocone-ObjMap-vdomain[cat-small-cs-simps]:
  assumes b ∈∘ 𝔅(|Obj|)
```

```
  shows 𝒟ₒ (tm-cf-Cocone α 𝔉(|ObjMap|)) = 𝔅(|Obj|)
proof–
  from assms interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_{CF.tm} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cocone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-small-cs-simps] = is-tm-functor.tm-cf-Cocone-ObjMap-vdomain

lemma (in is-tm-functor) tm-cf-Cone-ObjMap-app[cat-small-cs-simps]:
  assumes b ∈ₒ 𝔅(|Obj|)
  shows tm-cf-Cone α 𝔉(|ObjMap|)(|b|) =
    Hom (cat-Funct α 𝔄 𝔅) (cf-map (cf-const 𝔄 𝔅 b)) (cf-map 𝔉)
proof–
  from assms interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_{CF.tm} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cone-def
    by
      (
        cs-concl
          cs-simp:
            cat-small-cs-simps
            cat-cs-simps
            cat-FUNCT-cs-simps
            cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-small-cs-simps] = is-tm-functor.tm-cf-Cone-ObjMap-app

lemma (in is-tm-functor) tm-cf-Cocone-ObjMap-app[cat-small-cs-simps]:
  assumes b ∈ₒ 𝔅(|Obj|)
  shows tm-cf-Cocone α 𝔉(|ObjMap|)(|b|) =
    Hom (cat-Funct α 𝔄 𝔅) (cf-map 𝔉) (cf-map (cf-const 𝔄 𝔅 b))
proof–
  from assms interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_{CF.tm} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cocone-def
    by
      (
```

cs-concl **cs-shallow**
cs-simp:
cat-small-cs-simps cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
qed

lemmas [*cat-small-cs-simps*] = *is-tm-functor.tm-cf-Cocone-ObjMap-app*

28.2.3 Arrow map

lemma (**in** *is-tm-functor*) *tm-cf-Cone-ArrMap-vsv*[*cat-small-cs-intros*]:
vsv (*tm-cf-Cone* α $\mathfrak{F}(\!|ArrMap|\!)$)
proof–
interpret Δ: *is-functor* α $\mathfrak{B}$ ‹*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$› ‹$\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-cs-intros*)
show *?thesis*
unfolding *tm-cf-Cone-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps cat-op-simps*
cs-intro:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
)
qed

lemmas [*cat-small-cs-intros*] = *is-tm-functor.tm-cf-Cone-ArrMap-vsv*

lemma (**in** *is-tm-functor*) *tm-cf-Cocone-ArrMap-vsv*[*cat-small-cs-intros*]:
vsv (*tm-cf-Cocone* α $\mathfrak{F}(\!|ArrMap|\!)$)
proof–
interpret Δ: *is-functor* α $\mathfrak{B}$ ‹*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$› ‹$\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-cs-intros*)
show *?thesis*
unfolding *tm-cf-Cocone-def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps cat-op-simps*
cs-intro:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
)
qed

lemmas [*cat-small-cs-intros*] = *is-tm-functor.tm-cf-Cocone-ArrMap-vsv*

lemma (**in** *is-tm-functor*) *tm-cf-Cone-ArrMap-vdomain*[*cat-small-cs-simps*]:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}$ (*tm-cf-Cone* α $\mathfrak{F}(\!|ArrMap|\!)$) = $\mathfrak{B}(\!|Arr|\!)$
proof–
interpret Δ: *is-functor* α $\mathfrak{B}$ ‹*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$› ‹$\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$›
by (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-cs-intros*)
from *assms* **show** *?thesis*
unfolding *tm-cf-Cone-def′*
by
(
cs-concl **cs-shallow**

```
      cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
      cs-intro:
        cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
  )
qed

lemmas [cat-small-cs-simps] = is-tm-functor.tm-cf-Cone-ArrMap-vdomain

lemma (in is-tm-functor) tm-cf-Cocone-ArrMap-vdomain[cat-small-cs-simps]:
  assumes b ∈∘ 𝔅(|Obj|)
  shows 𝒟∘ (tm-cf-Cocone α 𝔉(|ArrMap|)) = 𝔅(|Arr|)
proof–
  interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_CF.tm α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cocone-def'
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
qed

lemmas [cat-small-cs-simps] = is-tm-functor.tm-cf-Cocone-ArrMap-vdomain

lemma (in is-tm-functor) tm-cf-Cone-ArrMap-app[cat-small-cs-simps]:
  assumes f : a ↦_𝔅 b
  shows tm-cf-Cone α 𝔉(|ArrMap|)(|f|) = cf-hom
    (cat-Funct α 𝔄 𝔅)
    [ntcf-arrow (ntcf-const 𝔄 𝔅 f), cat-Funct α 𝔄 𝔅(|CId|)(|cf-map 𝔉|)]∘
proof–
  interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_CF.tm α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cone-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros cat-op-intros
      )
qed

lemmas [cat-small-cs-simps] = is-tm-functor.tm-cf-Cone-ArrMap-app

lemma (in is-tm-functor) tm-cf-Cocone-ArrMap-app[cat-small-cs-simps]:
  assumes f : a ↦_𝔅 b
  shows tm-cf-Cocone α 𝔉(|ArrMap|)(|f|) = cf-hom
    (cat-Funct α 𝔄 𝔅)
    [cat-Funct α 𝔄 𝔅(|CId|)(|cf-map 𝔉|), ntcf-arrow (ntcf-const 𝔄 𝔅 f)]∘
proof–
  interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_CF.tm α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  from assms show ?thesis
    unfolding tm-cf-Cocone-def
    by
```

(
cs-concl
cs-simp: *cat-cs-simps cat-FUNCT-cs-simps cat-op-simps cat-op-simps*
cs-intro:
cat-small-cs-intros
cat-cs-intros
cat-FUNCT-cs-intros
cat-op-intros
)
qed

lemmas [*cat-small-cs-simps*] = *is-tm-functor.tm-cf-Cocone-ArrMap-app*

28.2.4 Small cone functor and small cocone functor are functors

lemma (**in** *is-tm-functor*) *tm-cf-cf-Cone-is-functor*:
tm-cf-Cone α $\mathfrak{F}$: *op-cat* $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof–
interpret $\mathfrak{AB}$: *category* α ‹*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$›
by
(
cs-concl **cs-shallow cs-intro**:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)
interpret *op-*Δ:
is-functor α ‹*op-cat* $\mathfrak{B}$› ‹*op-cat* (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$)› ‹*op-cf* ($\Delta_{CF.tm}$ α $\mathfrak{A}$ $\mathfrak{B}$)›
by
(
cs-concl **cs-shallow cs-intro**:
cat-small-cs-intros cat-cs-intros cat-op-intros
)
have $Hom_{O.C\alpha}$*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$(−,*cf-map* $\mathfrak{F}$) :
op-cat (*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$) $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
by
(
cs-concl **cs-shallow**
cs-simp: *cat-FUNCT-cs-simps*
cs-intro: *cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros*
)
then show *tm-cf-Cone* α $\mathfrak{F}$: *op-cat* $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
unfolding *tm-cf-Cone-def'* **by** (*cs-concl* **cs-intro**: *cat-cs-intros*)
qed

lemma (**in** *is-tm-functor*) *tm-cf-cf-Cone-is-functor'*[*cat-small-cs-intros*]:
assumes $\mathfrak{A}'$ = *op-cat* $\mathfrak{B}$ **and** $\mathfrak{B}'$ = *cat-Set* α **and** α' = α
shows *tm-cf-Cone* α $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{C\alpha'}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule tm-cf-cf-Cone-is-functor*)

lemmas [*cat-small-cs-intros*] = *is-tm-functor.tm-cf-cf-Cone-is-functor'*

lemma (**in** *is-tm-functor*) *tm-cf-cf-Cocone-is-functor*:
tm-cf-Cocone α $\mathfrak{F}$: $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof–
interpret *Funct*: *category* α ‹*cat-Funct* α $\mathfrak{A}$ $\mathfrak{B}$›
by
(
cs-concl **cs-shallow cs-intro**:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros

```
    )
  interpret Δ: is-functor α 𝔅 ‹cat-Funct α 𝔄 𝔅› ‹Δ_{CF.tm} α 𝔄 𝔅›
    by (cs-concl cs-shallow cs-intro: cat-small-cs-intros cat-cs-intros)
  have Hom_{O.Cα}cat-Funct α 𝔄 𝔅(cf-map 𝔉,–) :
    cat-Funct α 𝔄 𝔅 ↦↦_{Cα} cat-Set α
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-FUNCT-cs-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )
  then show tm-cf-Cocone α 𝔉 : 𝔅 ↦↦_{Cα} cat-Set α
    unfolding tm-cf-Cocone-def' by (cs-concl cs-intro: cat-cs-intros)
qed

lemma (in is-tm-functor) tm-cf-cf-Cocone-is-functor'[cat-small-cs-intros]:
  assumes 𝔅' = cat-Set α and α' = α
  shows tm-cf-Cocone α 𝔉 : 𝔅 ↦↦_{Cα'} 𝔅'
  unfolding assms by (rule tm-cf-cf-Cocone-is-functor)

lemmas [cat-small-cs-intros] = is-tm-functor.tm-cf-cf-Cocone-is-functor'
```

29 Yoneda Lemma

29.1 Yoneda map

The Yoneda map is the bijection that is used in the statement of the Yoneda Lemma, as presented, for example, in Chapter III-2 in [7] or in subsection 1.15 in [3].

definition *Yoneda-map* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *Yoneda-map* α $\mathfrak{K}$ r =
    (
      $\lambda\psi\in_\circ$*these-ntcfs* α ($\mathfrak{K}$(|*HomDom*|)) (*cat-Set* α) $Hom_{O.C\alpha}\mathfrak{K}$(|*HomDom*|)($r$,−) $\mathfrak{K}$.
        ψ(|*NTMap*|)(|r|)(|*ArrVal*|)($\mathfrak{K}$(|*HomDom*|)(|*CId*|)(|r|))
)

Elementary properties.

mk-VLambda *Yoneda-map-def*
  |*vsv Yoneda-map-vsv*[*cat-cs-intros*]|

mk-VLambda (**in** *is-functor*) *Yoneda-map-def*[**where** α=α **and** $\mathfrak{K}$=$\mathfrak{F}$, *unfolded cf-HomDom*]
  |*vdomain Yoneda-map-vdomain*|
  |*app Yoneda-map-app*[*unfolded these-ntcfs-iff*]|

lemmas [*cat-cs-simps*] = *is-functor.Yoneda-map-vdomain*

lemmas *Yoneda-map-app*[*cat-cs-simps*] =
  is-functor.Yoneda-map-app[*unfolded these-ntcfs-iff*]

29.2 Yoneda component

29.2.1 Definition and elementary properties

The Yoneda components are the components of the natural transformations that appear in the statement of the Yoneda Lemma (e.g., see Chapter III-2 in [7] or subsection 1.15 in [3]).

definition *Yoneda-component* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *Yoneda-component* $\mathfrak{K}$ r u d =
    [
      ($\lambda f\in_\circ Hom$ ($\mathfrak{K}$(|*HomDom*|)) r d. $\mathfrak{K}$(|*ArrMap*|)(|f|)(|*ArrVal*|)(|u|)),
      Hom ($\mathfrak{K}$(|*HomDom*|)) r d,
      $\mathfrak{K}$(|*ObjMap*|)(|d|)
    ]$_\circ$

Components.

lemma (**in** *is-functor*) *Yoneda-component-components*:
  shows *Yoneda-component* $\mathfrak{F}$ r u d(|*ArrVal*|) =
    ($\lambda f\in_\circ Hom$ $\mathfrak{A}$ r d. $\mathfrak{F}$(|*ArrMap*|)(|f|)(|*ArrVal*|)(|u|))
    and *Yoneda-component* $\mathfrak{F}$ r u d(|*ArrDom*|) = Hom $\mathfrak{A}$ r d
    and *Yoneda-component* $\mathfrak{F}$ r u d(|*ArrCod*|) = $\mathfrak{F}$(|*ObjMap*|)(|d|)
  unfolding *Yoneda-component-def arr-field-simps*
  by (*simp-all add*: *nat-omega-simps cat-cs-simps*)

29.2.2 Arrow value

mk-VLambda (**in** *is-functor*) *Yoneda-component-components*(*1*)
  |*vsv Yoneda-component-ArrVal-vsv*|
  |*vdomain Yoneda-component-ArrVal-vdomain*|
  |*app Yoneda-component-ArrVal-app*[*unfolded in-Hom-iff*]|

lemmas [*cat-cs-simps*] = *is-functor.Yoneda-component-ArrVal-vdomain*

lemmas *Yoneda-component-ArrVal-app*[*cat-cs-simps*] =
is-functor.Yoneda-component-ArrVal-app[*unfolded in-Hom-iff*]

29.2.3 Yoneda component is an arrow in the category *Set*

lemma (**in** *category*) *cat-Yoneda-component-is-arr*:
assumes $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $r \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $u \in_\circ \mathfrak{K}(\!|ObjMap|\!)(\!|r|\!)$
and $d \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *Yoneda-component* $\mathfrak{K}$ r u d : *Hom* $\mathfrak{C}$ r d $\mapsto_{cat\text{-}Set\ \alpha}$ $\mathfrak{K}(\!|ObjMap|\!)(\!|d|\!)$
proof–
interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{K}$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro cat-Set-is-arrI arr-SetI*, *unfold* $\mathfrak{K}$.*Yoneda-component-components*)
show *vfsequence* (*Yoneda-component* $\mathfrak{K}$ r u d)
unfolding *Yoneda-component-def* **by** *simp*
show *vcard* (*Yoneda-component* $\mathfrak{K}$ r u d) = $3_{\mathbb{N}}$
unfolding *Yoneda-component-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ ($\lambda f \in_\circ$ *Hom* $\mathfrak{C}$ r d. $\mathfrak{K}(\!|ArrMap|\!)(\!|f|\!)(\!|ArrVal|\!)(\!|u|\!)$) $\subseteq_\circ$ $\mathfrak{K}(\!|ObjMap|\!)(\!|d|\!)$
proof(*rule vrange-VLambda-vsubset*)
fix f **assume** $f \in_\circ$ *Hom* $\mathfrak{C}$ r d
then have $\mathfrak{K}f$: $\mathfrak{K}(\!|ArrMap|\!)(\!|f|\!) : \mathfrak{K}(\!|ObjMap|\!)(\!|r|\!) \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{K}(\!|ObjMap|\!)(\!|d|\!)$
by (*auto simp*: *cat-cs-intros*)
note $\mathfrak{K}f$-*simps* = *cat-Set-is-arrD*[*OF* $\mathfrak{K}f$]
interpret $\mathfrak{K}f$: *arr-Set* α ‹$\mathfrak{K}(\!|ArrMap|\!)(\!|f|\!)$› **by** (*rule* $\mathfrak{K}f$-*simps*(*1*))
have $u \in_\circ \mathcal{D}_\circ$ ($\mathfrak{K}(\!|ArrMap|\!)(\!|f|\!)(\!|ArrVal|\!)$)
by (*simp add*: $\mathfrak{K}f$-*simps assms cat-Set-cs-simps*)
with $\mathfrak{K}f$.*arr-Set-ArrVal-vrange*[*unfolded* $\mathfrak{K}f$-*simps*] **show**
$\mathfrak{K}(\!|ArrMap|\!)(\!|f|\!)(\!|ArrVal|\!)(\!|u|\!) \in_\circ \mathfrak{K}(\!|ObjMap|\!)(\!|d|\!)$
by (*blast elim*: $\mathfrak{K}f$.*ArrVal.vsv-value*)
qed
from *assms* $\mathfrak{K}$.*HomCod.cat-Obj-vsubset-Vset* **show** $\mathfrak{K}(\!|ObjMap|\!)(\!|d|\!) \in_\circ$ *Vset* α
by (*auto dest*: $\mathfrak{K}$.*cf-ObjMap-app-in-HomCod-Obj*)
qed (*auto simp*: *assms cat-cs-intros*)
qed

lemma (**in** *category*) *cat-Yoneda-component-is-arr′*:
assumes $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $r \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $u \in_\circ \mathfrak{K}(\!|ObjMap|\!)(\!|r|\!)$
and $d \in_\circ \mathfrak{C}(\!|Obj|\!)$
and s = *Hom* $\mathfrak{C}$ r d
and $t = \mathfrak{K}(\!|ObjMap|\!)(\!|d|\!)$
and $\mathfrak{D}$ = *cat-Set* α
shows *Yoneda-component* $\mathfrak{K}$ r u d : $s \mapsto_{\mathfrak{D}} t$
unfolding *assms*(*5–7*) **using** *assms*(*1–4*) **by** (*rule cat-Yoneda-component-is-arr*)

lemmas [*cat-cs-intros*] = *category.cat-Yoneda-component-is-arr′*[*rotated 1*]

29.3 Yoneda arrow

29.3.1 Definition and elementary properties

The Yoneda arrows are the natural transformations that appear in the statement of the Yoneda Lemma in Chapter III-2 in [7] and subsection 1.15 in [3].

definition *Yoneda-arrow* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$

where *Yoneda-arrow* α $\mathfrak{K}$ r u =
[
$(\lambda d \in_\circ \mathfrak{K}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).$ *Yoneda-component* $\mathfrak{K}$ r u $d)$,
$Hom_{O.C\alpha}\mathfrak{K}(\lfloor HomDom \rfloor)(r,-)$,
$\mathfrak{K}$,
$\mathfrak{K}(\lfloor HomDom \rfloor)$,
cat-Set α
]$_\circ$

Components.

lemma (**in** *is-functor*) *Yoneda-arrow-components*:
shows *Yoneda-arrow* α $\mathfrak{F}$ r $u(\lfloor NTMap \rfloor)$ =
$(\lambda d \in_\circ \mathfrak{A}(\lfloor Obj \rfloor).$ *Yoneda-component* $\mathfrak{F}$ r u $d)$
and *Yoneda-arrow* α $\mathfrak{F}$ r $u(\lfloor NTDom \rfloor) = Hom_{O.C\alpha}\mathfrak{A}(r,-)$
and *Yoneda-arrow* α $\mathfrak{F}$ r $u(\lfloor NTCod \rfloor) = \mathfrak{F}$
and *Yoneda-arrow* α $\mathfrak{F}$ r $u(\lfloor NTDGDom \rfloor) = \mathfrak{A}$
and *Yoneda-arrow* α $\mathfrak{F}$ r $u(\lfloor NTDGCod \rfloor) =$ *cat-Set* α
unfolding *Yoneda-arrow-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps cat-cs-simps*)

29.3.2 Natural transformation map

mk-VLambda (**in** *is-functor*) *Yoneda-arrow-components*(*1*)
|*vsv Yoneda-arrow-NTMap-vsv*|
|*vdomain Yoneda-arrow-NTMap-vdomain*|
|*app Yoneda-arrow-NTMap-app*|

lemmas [*cat-cs-simps*] = *is-functor.Yoneda-arrow-NTMap-vdomain*

lemmas *Yoneda-arrow-NTMap-app*[*cat-cs-simps*] =
is-functor.Yoneda-arrow-NTMap-app

29.3.3 Yoneda arrow is a natural transformation

lemma (**in** *category*) *cat-Yoneda-arrow-is-ntcf*:
assumes $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $r \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
and $u \in_\circ \mathfrak{K}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$
shows *Yoneda-arrow* α $\mathfrak{K}$ r $u : Hom_{O.C\alpha}\mathfrak{C}(r,-) \mapsto_{CF} \mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof–

interpret $\mathfrak{K}$: *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{K}$ **by** (*rule assms*(*1*))
note $\mathfrak{K}ru$ = *cat-Yoneda-component-is-arr*[*OF assms*]
let *?*$\mathfrak{K}ru$ = ‹*Yoneda-component* $\mathfrak{K}$ r u›

show *?thesis*
proof(*intro is-ntcfI'*, *unfold* $\mathfrak{K}$.*Yoneda-arrow-components*)

show *vfsequence* (*Yoneda-arrow* α $\mathfrak{K}$ r u)
unfolding *Yoneda-arrow-def* **by** *simp*
show *vcard* (*Yoneda-arrow* α $\mathfrak{K}$ r u) = $5_{\mathbb{N}}$
unfolding *Yoneda-arrow-def* **by** (*simp add*: *nat-omega-simps*)

show
$(\lambda d \in_\circ \mathfrak{C}(\lfloor Obj \rfloor).$ *?*$\mathfrak{K}ru$ $d)(\lfloor a \rfloor)$:
$Hom_{O.C\alpha}\mathfrak{C}(r,-)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{K}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
if $a \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$ **for** a
using *that assms category-axioms*

```
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-op-simps V-cs-simps
        cs-intro: cat-cs-intros
    )
show
  (λd∈∘ℭ(|Obj|). ?ℜru d)(|b|) ∘A cat-Set α Hom_O.Cα ℭ(r,−)(|ArrMap|)(|f|) =
    ℜ(|ArrMap|)(|f|) ∘A cat-Set α (λd∈∘ℭ(|Obj|). ?ℜru d)(|a|)
  if f : a ↦ℭ b for a b f
proof–

  note 𝔐a = ℜru[OF cat-is-arrD(2)[OF that]]
  note 𝔐b = ℜru[OF cat-is-arrD(3)[OF that]]

  from category-axioms assms that 𝔐b have b-f:
    ?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘ :
      Hom ℭ r a ↦cat-Set α ℜ(|ObjMap|)(|b|)
    by
      (
        cs-concl cs-shallow
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  then have dom-lhs:
    𝒟∘ ((?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘)(|ArrVal|)) =
      Hom ℭ r a
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms that 𝔐a have f-a:
    ℜ(|ArrMap|)(|f|) ∘A cat-Set α ?ℜru a :
      Hom ℭ r a ↦cat-Set α ℜ(|ObjMap|)(|b|)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  then have dom-rhs:
    𝒟∘ ((ℜ(|ArrMap|)(|f|) ∘A cat-Set α ?ℜru a)(|ArrVal|)) = Hom ℭ r a
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)

  have [cat-cs-simps]:
    ?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘ =
      ℜ(|ArrMap|)(|f|) ∘A cat-Set α ?ℜru a
  proof(rule arr-Set-eqI[of α])

    from b-f show arr-Set-b-f:
      arr-Set α (?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘)
      by (auto simp: cat-Set-is-arrD(1))
    interpret b-f: arr-Set α ‹?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘›
      by (rule arr-Set-b-f)
    from f-a show arr-Set-f-a:
      arr-Set α (ℜ(|ArrMap|)(|f|) ∘A cat-Set α ?ℜru a)
      by (auto simp: cat-Set-is-arrD(1))
    interpret f-a: arr-Set α ‹ℜ(|ArrMap|)(|f|) ∘A cat-Set α ?ℜru a›
      by (rule arr-Set-f-a)

    show
      (?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘)(|ArrVal|) =
        (ℜ(|ArrMap|)(|f|) ∘A cat-Set α ?ℜru a)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix q assume q : r ↦ℭ a
      from category-axioms assms that this 𝔐a 𝔐b show
        (?ℜru b ∘A cat-Set α cf-hom ℭ [ℭ(|CId|)(|r|), f]∘)(|ArrVal|)(|q|) =
```

```
        (ℜ(|ArrMap|)(|f|) ∘_A cat-Set α ?ℜru a)(|ArrVal|)(|q|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
    qed (use arr-Set-b-f arr-Set-f-a in auto)

  qed (use b-f f-a in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

  from that category-axioms assms 𝔐a 𝔐b show ?thesis
    by
      (
        cs-concl
          cs-simp: V-cs-simps cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros
      )

 qed

qed (auto simp: assms(2) cat-cs-intros)

qed
```

29.4 Yoneda Lemma

The following lemma is approximately equivalent to the Yoneda Lemma stated in subsection 1.15 in [3] (the first two conclusions correspond to the statement of the Yoneda lemma in Chapter III-2 in [7]).

```
lemma (in category) cat-Yoneda-Lemma:
  assumes ℜ : ℭ ↦↦_Cα cat-Set α and r ∈_∘ ℭ(|Obj|)
  shows v11 (Yoneda-map α ℜ r)
    and ℛ_∘ (Yoneda-map α ℜ r) = ℜ(|ObjMap|)(|r|)
    and (Yoneda-map α ℜ r)⁻¹_∘ = (λu∈_∘ℜ(|ObjMap|)(|r|). Yoneda-arrow α ℜ r u)
proof–

  interpret ℜ: is-functor α ℭ ‹cat-Set α› ℜ by (rule assms(1))

  from assms(2) ℜ.HomCod.cat-Obj-vsubset-Vset ℜ.cf-ObjMap-app-in-HomCod-Obj
  have ℜr-in-Vset: ℜ(|ObjMap|)(|r|) ∈_∘ Vset α
    by auto

  show Ym: v11 (Yoneda-map α ℜ r)
  proof(intro vsv.vsv-valeq-v11I, unfold ℜ.Yoneda-map-vdomain these-ntcfs-iff)

    fix 𝔐 𝔑
    assume prems:
      𝔐 : Hom_O.Cα ℭ(r,−) ↦_CF ℜ : ℭ ↦↦_Cα cat-Set α
      𝔑 : Hom_O.Cα ℭ(r,−) ↦_CF ℜ : ℭ ↦↦_Cα cat-Set α
      Yoneda-map α ℜ r(|𝔐|) = Yoneda-map α ℜ r(|𝔑|)
    from prems(3) have 𝔐r-𝔑r:
      𝔐(|NTMap|)(|r|)(|ArrVal|)(|ℭ(|CId|)(|r|)|) = 𝔑(|NTMap|)(|r|)(|ArrVal|)(|ℭ(|CId|)(|r|)|)
      unfolding
        Yoneda-map-app[OF assms(1) prems(1)]
        Yoneda-map-app[OF assms(1) prems(2)]
      by simp
```

```
interpret 𝔐: is-ntcf α ℭ ‹cat-Set α› ‹Hom_{O.Cα}ℭ(r,−)› ℜ 𝔐
  by (rule prems(1))
interpret 𝔑: is-ntcf α ℭ ‹cat-Set α› ‹Hom_{O.Cα}ℭ(r,−)› ℜ 𝔑
  by (rule prems(2))

show 𝔐 = 𝔑
proof
  (
    rule ntcf-eqI[OF prems(1,2)];
    (rule refl)?;
    rule vsv-eqI,
    unfold 𝔐.ntcf-NTMap-vdomain 𝔑.ntcf-NTMap-vdomain
  )

  fix d assume prems': d ∈∘ ℭ⦇Obj⦈

  note 𝔐d-simps = cat-Set-is-arrD[OF 𝔐.ntcf-NTMap-is-arr[OF prems']]
  interpret 𝔐d: arr-Set α ‹𝔐⦇NTMap⦈⦇d⦈› by (rule 𝔐d-simps(1))

  note 𝔑d-simps = cat-Set-is-arrD[OF 𝔑.ntcf-NTMap-is-arr[OF prems']]
  interpret 𝔑d: arr-Set α ‹𝔑⦇NTMap⦈⦇d⦈› by (rule 𝔑d-simps(1))

  show 𝔐⦇NTMap⦈⦇d⦈ = 𝔑⦇NTMap⦈⦇d⦈
  proof(rule arr-Set-eqI[of α])
    show 𝔐⦇NTMap⦈⦇d⦈⦇ArrVal⦈ = 𝔑⦇NTMap⦈⦇d⦈⦇ArrVal⦈
    proof
      (
        rule vsv-eqI,
        unfold
          𝔑d.arr-Set-ArrVal-vdomain
          𝔐d.arr-Set-ArrVal-vdomain
          𝔐d-simps
          𝔑d-simps
      )
      fix f assume prems'': f ∈∘ Hom_{O.Cα}ℭ(r,−)⦇ObjMap⦈⦇d⦈
      from prems'' prems' category-axioms assms(2) have f: f : r ↦_ℭ d
        by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-op-intros)
      from 𝔐.ntcf-Comp-commute[OF f] have
        (
          𝔐⦇NTMap⦈⦇d⦈ ∘_{A cat-Set α} Hom_{O.Cα}ℭ(r,−)⦇ArrMap⦈⦇f⦈
        )⦇ArrVal⦈⦇ℭ⦇CId⦈⦇r⦈⦈ =
          (ℜ⦇ArrMap⦈⦇f⦈ ∘_{A cat-Set α} 𝔐⦇NTMap⦈⦇r⦈)⦇ArrVal⦈⦇ℭ⦇CId⦈⦇r⦈⦈
        by simp
      from this category-axioms assms(2) f prems prems' have 𝔐df:
        𝔐⦇NTMap⦈⦇d⦈⦇ArrVal⦈⦇f⦈ =
          ℜ⦇ArrMap⦈⦇f⦈⦇ArrVal⦈⦇𝔐⦇NTMap⦈⦇r⦈⦇ArrVal⦈⦇ℭ⦇CId⦈⦇r⦈⦈⦈⦈
        by
          (
            cs-prems cs-shallow
              cs-simp: cat-cs-simps cat-op-simps
              cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
          )
      from 𝔑.ntcf-Comp-commute[OF f] have
        (
          𝔑⦇NTMap⦈⦇d⦈ ∘_{A cat-Set α}
          Hom_{O.Cα}ℭ(r,−)⦇ArrMap⦈⦇f⦈
        )⦇ArrVal⦈⦇ℭ⦇CId⦈⦇r⦈⦈ =
```

```
      (ℜ(|ArrMap|)(|f|) ∘_A cat-Set α 𝔑(|NTMap|)(|r|))(|ArrVal|)(|ℭ(|CId|)(|r|)|)
    by simp
  from this category-axioms assms(2) f prems prems' have 𝔑df:
    𝔑(|NTMap|)(|d|)(|ArrVal|)(|f|) =
      ℜ(|ArrMap|)(|f|)(|ArrVal|)(|𝔑(|NTMap|)(|r|)(|ArrVal|)(|ℭ(|CId|)(|r|)|)|)
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  show 𝔐(|NTMap|)(|d|)(|ArrVal|)(|f|) = 𝔑(|NTMap|)(|d|)(|ArrVal|)(|f|)
    unfolding 𝔐df 𝔑df 𝔐r-𝔑r by simp
 qed auto

 qed (simp-all add: 𝔐d-simps 𝔑d-simps)

 qed auto

qed (auto simp: Yoneda-map-vsv)

interpret Ym: v11 ‹Yoneda-map α ℜ r› by (rule Ym)

have YY: Yoneda-map α ℜ r(|Yoneda-arrow α ℜ r a|) = a
 if a ∈₀ ℜ(|ObjMap|)(|r|) for a
proof–
 note cat-Yoneda-arrow-is-ntcf[OF assms that]
 moreover with assms have Ya: Yoneda-arrow α ℜ r a ∈₀ 𝒟₀ (Yoneda-map α ℜ r)
  by
   (
    cs-concl cs-shallow
      cs-simp: these-ntcfs-iff cat-cs-simps cs-intro: cat-cs-intros
   )
 ultimately show Yoneda-map α ℜ r(|Yoneda-arrow α ℜ r a|) = a
  using assms that ℜr-in-Vset
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
qed

show [simp]: ℛ₀ (Yoneda-map α ℜ r) = ℜ(|ObjMap|)(|r|)
proof(intro vsubset-antisym)

 show ℛ₀ (Yoneda-map α ℜ r) ⊆₀ ℜ(|ObjMap|)(|r|)
  unfolding Yoneda-map-def
 proof(intro vrange-VLambda-vsubset, unfold these-ntcfs-iff ℜ.cf-HomDom)
  fix 𝔐 assume prems: 𝔐 : Hom_{O.CA}ℭ(r,-) ↦_CF ℜ : ℭ ↦↦_CA cat-Set α
  then interpret 𝔐: is-ntcf α ℭ ‹cat-Set α› ‹Hom_{O.CA}ℭ(r,-)› ℜ 𝔐 .
  note 𝔐r-simps = cat-Set-is-arrD[OF 𝔐.ntcf-NTMap-is-arr[OF assms(2)]]
  interpret 𝔐r: arr-Set α ‹𝔐(|NTMap|)(|r|)› by (rule 𝔐r-simps(1))
  from prems category-axioms assms(2) have
    ℭ(|CId|)(|r|) ∈₀ 𝒟₀ (𝔐(|NTMap|)(|r|)(|ArrVal|))
    unfolding 𝔐r.arr-Set-ArrVal-vdomain 𝔐r-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
      )
  then have 𝔐(|NTMap|)(|r|)(|ArrVal|)(|ℭ(|CId|)(|r|)|) ∈₀ ℛ₀ (𝔐(|NTMap|)(|r|)(|ArrVal|))
    by (blast elim: 𝔐r.ArrVal.vsv-value)
```

```
    then show 𝔐(|NTMap|)(|r|)(|ArrVal|)(|ℭ(|CId|)(|r|)|) ∈∘ ℜ(|ObjMap|)(|r|)
      by (auto simp: 𝔐r-simps dest!: vsubsetD[OF 𝔐r.arr-Set-ArrVal-vrange])
  qed

  show ℜ(|ObjMap|)(|r|) ⊆∘ ℛ∘ (Yoneda-map α ℜ r)
  proof(intro vsubsetI)
    fix u assume prems: u ∈∘ ℜ(|ObjMap|)(|r|)
    from cat-Yoneda-arrow-is-ntcf[OF assms prems] have
      Yoneda-arrow α ℜ r u ∈∘ 𝒟∘ (Yoneda-map α ℜ r)
      by
        (
          cs-concl cs-shallow
            cs-simp: these-ntcfs-iff cat-cs-simps cs-intro: cat-cs-intros
        )
    with YY[OF prems] show u ∈∘ ℛ∘ (Yoneda-map α ℜ r)
      by (force dest!: vdomain-atD)
  qed

qed

show (Yoneda-map α ℜ r)⁻¹∘ = (λu∈∘ℜ(|ObjMap|)(|r|). Yoneda-arrow α ℜ r u)
proof(rule vsv-eqI, unfold vdomain-vconverse vdomain-VLambda)
  from Ym show vsv ((Yoneda-map α ℜ r)⁻¹∘) by auto
  show (Yoneda-map α ℜ r)⁻¹∘(|a|) = (λu∈∘ℜ(|ObjMap|)(|r|). Yoneda-arrow α ℜ r u)(|a|)
    if a ∈∘ ℛ∘ (Yoneda-map α ℜ r) for a
  proof–
    from that have a: a ∈∘ ℜ(|ObjMap|)(|r|) by simp
    note Ya = cat-Yoneda-arrow-is-ntcf[OF assms a]
    then have Yoneda-arrow α ℜ r a ∈∘ 𝒟∘ (Yoneda-map α ℜ r)
      by
        (
          cs-concl cs-shallow
            cs-simp: these-ntcfs-iff cat-cs-simps cs-intro: cat-cs-intros
        )
    with Ya YY[OF a] a show ?thesis
      by
        (
          intro Ym.v11-vconverse-app[
            unfolded ℜ.Yoneda-map-vdomain these-ntcfs-iff
            ]
        )
        (simp-all add: these-ntcfs-iff cat-cs-simps)
  qed
qed auto

qed
```

29.5 Inverse of the Yoneda map

```
lemma (in category) inv-Yoneda-map-v11:
  assumes ℜ : ℭ ↦↦_Cα cat-Set α and r ∈∘ ℭ(|Obj|)
  shows v11 ((Yoneda-map α ℜ r)⁻¹∘)
  using cat-Yoneda-Lemma(1)[OF assms] by (simp add: v11.v11-vconverse)

lemma (in category) inv-Yoneda-map-vdomain:
  assumes ℜ : ℭ ↦↦_Cα cat-Set α and r ∈∘ ℭ(|Obj|)
  shows 𝒟∘ ((Yoneda-map α ℜ r)⁻¹∘) = ℜ(|ObjMap|)(|r|)
  unfolding cat-Yoneda-Lemma(3)[OF assms] by simp
```

lemmas [*cat-cs-simps*] = *category.inv-Yoneda-map-vdomain*

lemma (**in** *category*) *inv-Yoneda-map-app*:
assumes $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α **and** $r \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$ **and** $u \in_\circ \mathfrak{K}(\lfloor ObjMap \rfloor)(\lfloor r \rfloor)$
shows $(Yoneda\text{-}map\ \alpha\ \mathfrak{K}\ r)^{-1}_\circ(\lfloor u \rfloor) = Yoneda\text{-}arrow\ \alpha\ \mathfrak{K}\ r\ u$
using *assms*(*3*) **unfolding** *cat-Yoneda-Lemma*(*3*)[*OF assms*(*1,2*)] **by** *simp*

lemmas [*cat-cs-simps*] = *category.inv-Yoneda-map-app*

lemma (**in** *category*) *inv-Yoneda-map-vrange*:
assumes $\mathfrak{K} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
shows $\mathcal{R}_\circ\ ((Yoneda\text{-}map\ \alpha\ \mathfrak{K}\ r)^{-1}_\circ) =$
$these\text{-}ntcfs\ \alpha\ \mathfrak{C}\ (cat\text{-}Set\ \alpha)\ Hom_{O.C\alpha}\mathfrak{C}(r,-)\ \mathfrak{K}$
proof–
interpret $\mathfrak{K}$: *is-functor* $\alpha\ \mathfrak{C}\ \langle cat\text{-}Set\ \alpha\rangle\ \mathfrak{K}$ **by** (*rule assms*(*1*))
show *?thesis* **unfolding** *Yoneda-map-def* **by** (*simp add*: *cat-cs-simps*)
qed

29.6 Component of a composition of a *Hom*-natural transformation with natural transformations

29.6.1 Definition and elementary properties

The following definition is merely a technical generalization that is used in the context of the description of the composition of a *Hom*-natural transformation with a natural transformation later in this section (also see subsection 1.15 in [3]).

definition *ntcf-Hom-component* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-Hom-component* $\varphi\ \psi\ a\ b$ =
[
(
$\lambda f \in_\circ Hom\ (\varphi(\lfloor NTDGCod \rfloor))\ (\varphi(\lfloor NTCod \rfloor)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))\ (\psi(\lfloor NTDom \rfloor)(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)).$
$\psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\psi(\lfloor NTDGCod \rfloor)} f \circ_{A\psi(\lfloor NTDGCod \rfloor)} \varphi(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
),
$Hom\ (\varphi(\lfloor NTDGCod \rfloor))\ (\varphi(\lfloor NTCod \rfloor)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))\ (\psi(\lfloor NTDom \rfloor)(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)),$
$Hom\ (\varphi(\lfloor NTDGCod \rfloor))\ (\varphi(\lfloor NTDom \rfloor)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))\ (\psi(\lfloor NTCod \rfloor)(\lfloor ObjMap \rfloor)(\lfloor b \rfloor))$
]$_\circ$

Components.

lemma *ntcf-Hom-component-components*:
shows *ntcf-Hom-component* $\varphi\ \psi\ a\ b(\lfloor ArrVal \rfloor) =$
(
$\lambda f \in_\circ Hom\ (\varphi(\lfloor NTDGCod \rfloor))\ (\varphi(\lfloor NTCod \rfloor)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))\ (\psi(\lfloor NTDom \rfloor)(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)).$
$\psi(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\psi(\lfloor NTDGCod \rfloor)} f \circ_{A\psi(\lfloor NTDGCod \rfloor)} \varphi(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
)
and *ntcf-Hom-component* $\varphi\ \psi\ a\ b(\lfloor ArrDom \rfloor) =$
$Hom\ (\varphi(\lfloor NTDGCod \rfloor))\ (\varphi(\lfloor NTCod \rfloor)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))\ (\psi(\lfloor NTDom \rfloor)(\lfloor ObjMap \rfloor)(\lfloor b \rfloor))$
and *ntcf-Hom-component* $\varphi\ \psi\ a\ b(\lfloor ArrCod \rfloor) =$
$Hom\ (\varphi(\lfloor NTDGCod \rfloor))\ (\varphi(\lfloor NTDom \rfloor)(\lfloor ObjMap \rfloor)(\lfloor a \rfloor))\ (\psi(\lfloor NTCod \rfloor)(\lfloor ObjMap \rfloor)(\lfloor b \rfloor))$
unfolding *ntcf-Hom-component-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

29.6.2 Arrow value

mk-VLambda *ntcf-Hom-component-components*(*1*)
|*vsv ntcf-Hom-component-ArrVal-vsv*[*intro*]|

context
  fixes α φ ψ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
  assumes φ: $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and ψ: $\psi : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule* φ)
interpretation ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ ψ **by** (*rule* ψ)

mk-VLambda
  ntcf-Hom-component-components(*1*)
    [
      of φ ψ,
      unfolded
        φ.*ntcf-NTDom* ψ.*ntcf-NTDom*
        φ.*ntcf-NTCod* ψ.*ntcf-NTCod*
        φ.*ntcf-NTDGDom* ψ.*ntcf-NTDGDom*
        φ.*ntcf-NTDGCod* ψ.*ntcf-NTDGCod*
    ]
  |*vdomain ntcf-Hom-component-ArrVal-vdomain*|
  |*app ntcf-Hom-component-ArrVal-app*[*unfolded in-Hom-iff*]|

lemmas [*cat-cs-simps*] =
  ntcf-Hom-component-ArrVal-vdomain
  ntcf-Hom-component-ArrVal-app

lemma *ntcf-Hom-component-ArrVal-vrange*:
  assumes $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
  shows
    $\mathcal{R}_\circ$ (*ntcf-Hom-component* φ ψ a $b(\!|ArrVal|\!)$) $\subseteq_\circ$
      Hom $\mathfrak{C}$ ($\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$) ($\mathfrak{G}'(\!|ObjMap|\!)(\!|b|\!)$)
proof
  (
    rule vsv.vsv-vrange-vsubset,
    unfold ntcf-Hom-component-ArrVal-vdomain in-Hom-iff
)
  fix f **assume** $f : \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!)$
  with *assms* φ ψ **show**
    ntcf-Hom-component φ ψ a $b(\!|ArrVal|\!)(\!|f|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{G}'(\!|ObjMap|\!)(\!|b|\!)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*rule ntcf-Hom-component-ArrVal-vsv*)

end

29.6.3 Arrow domain and codomain

context
  fixes α φ ψ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
  assumes φ: $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and ψ: $\psi : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule* φ)
interpretation ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ ψ **by** (*rule* ψ)

lemma *ntcf-Hom-component-ArrDom*[*cat-cs-simps*]:
  ntcf-Hom-component φ ψ a $b(\!|ArrDom|\!)$ = *Hom* $\mathfrak{C}$ ($\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$) ($\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!)$)
  unfolding *ntcf-Hom-component-components* **by** (*simp add*: *cat-cs-simps*)

lemma *ntcf-Hom-component-ArrCod*[*cat-cs-simps*]:
ntcf-Hom-component φ ψ *a b*(|*ArrCod*|) = *Hom* $\mathfrak{C}$ ($\mathfrak{F}$(|*ObjMap*|)(|*a*|)) ($\mathfrak{G}'$(|*ObjMap*|)(|*b*|))
unfolding *ntcf-Hom-component-components* **by** (*simp add*: *cat-cs-simps*)

end

29.6.4 Component of a composition of a *Hom*-natural transformation with natural transformations is an arrow in the category *Set*

lemma (**in** *category*) *cat-ntcf-Hom-component-is-arr*:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\psi : \mathfrak{F}' \mapsto\mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ$ *op-cat* $\mathfrak{A}$(|*Obj*|)
and $b \in_\circ \mathfrak{B}$(|*Obj*|)
shows
ntcf-Hom-component φ ψ *a b* :
Hom $\mathfrak{C}$ ($\mathfrak{G}$(|*ObjMap*|)(|*a*|)) ($\mathfrak{F}'$(|*ObjMap*|)(|*b*|)) $\mapsto_{cat\text{-}Set\ \alpha}$
Hom $\mathfrak{C}$ ($\mathfrak{F}$(|*ObjMap*|)(|*a*|)) ($\mathfrak{G}'$(|*ObjMap*|)(|*b*|))
proof–

interpret φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule assms*(*1*))
interpret ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ ψ **by** (*rule assms*(*2*))

from *assms* **have** *a*: $a \in_\circ \mathfrak{A}$(|*Obj*|) **unfolding** *cat-op-simps* **by** *simp*

show *?thesis*
proof(*intro cat-Set-is-arrI arr-SetI*)
show *vfsequence* (*ntcf-Hom-component* φ ψ *a b*)
unfolding *ntcf-Hom-component-def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* (*ntcf-Hom-component* φ ψ *a b*) = $3_{\mathbb{N}}$
unfolding *ntcf-Hom-component-def* **by** (*simp add*: *nat-omega-simps*)
from *assms ntcf-Hom-component-ArrVal-vrange*[*OF assms*(*1,2*) *a assms*(*4*)] **show**
$\mathcal{R}_\circ$ (*ntcf-Hom-component* φ ψ *a b*(|*ArrVal*|)) $\subseteq_\circ$
ntcf-Hom-component φ ψ *a b*(|*ArrCod*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *assms*(*1,2,4*) *a* **show** *ntcf-Hom-component* φ ψ *a b*(|*ArrDom*|) $\in_\circ$ *Vset* α
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms*(*1,2,4*) *a* **show** *ntcf-Hom-component* φ ψ *a b*(|*ArrCod*|) $\in_\circ$ *Vset* α
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*use assms* **in** ‹*auto simp*: *ntcf-Hom-component-components cat-cs-simps*›)

qed

lemma (**in** *category*) *cat-ntcf-Hom-component-is-arr'*:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\psi : \mathfrak{F}' \mapsto\mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ$ *op-cat* $\mathfrak{A}$(|*Obj*|)
and $b \in_\circ \mathfrak{B}$(|*Obj*|)
and $\mathfrak{A}'$ = *Hom* $\mathfrak{C}$ ($\mathfrak{G}$(|*ObjMap*|)(|*a*|)) ($\mathfrak{F}'$(|*ObjMap*|)(|*b*|))
and $\mathfrak{B}'$ = *Hom* $\mathfrak{C}$ ($\mathfrak{F}$(|*ObjMap*|)(|*a*|)) ($\mathfrak{G}'$(|*ObjMap*|)(|*b*|))
and $\mathfrak{C}'$ = *cat-Set* α
shows *ntcf-Hom-component* φ ψ *a b* : $\mathfrak{A}' \mapsto_{\mathfrak{C}'} \mathfrak{B}'$
using *assms*(*1–4*) **unfolding** *assms*(*5–7*) **by** (*rule cat-ntcf-Hom-component-is-arr*)

lemmas [*cat-cs-intros*] = *category.cat-ntcf-Hom-component-is-arr'*

29.6.5 Naturality of the components of a composition of a *Hom*-natural transformation with natural transformations

lemma (in *category*) *cat-ntcf-Hom-component-nat*:
assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\psi : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $g : a \mapsto_{op\text{-}cat} \mathfrak{A}\ a'$
and $f : b \mapsto_{\mathfrak{B}} b'$
shows
ntcf-Hom-component $\varphi\ \psi\ a'\ b' \circ_{A\, cat\text{-}Set\ \alpha}$
cf-hom $\mathfrak{C}\ [\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor),\ \mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)]_\circ =$
cf-hom $\mathfrak{C}\ [\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor),\ \mathfrak{G}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)]_\circ \circ_{A\, cat\text{-}Set\ \alpha}$
ntcf-Hom-component $\varphi\ \psi\ a\ b$
proof–

let *?Y-ab* = ‹*ntcf-Hom-component* $\varphi\ \psi\ a\ b$›
and *?Y-a'b'* = ‹*ntcf-Hom-component* $\varphi\ \psi\ a'\ b'$›
and $?\mathfrak{G}g$ = ‹$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor)$›
and $?\mathfrak{F}'f$ = ‹$\mathfrak{F}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$›
and $?\mathfrak{F}g$ = ‹$\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor g \rfloor)$›
and $?\mathfrak{G}'f$ = ‹$\mathfrak{G}'(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$›
and $?\mathfrak{G}a$ = ‹$\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$›
and $?\mathfrak{F}'b$ = ‹$\mathfrak{F}'(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$›
and $?\mathfrak{F}a'$ = ‹$\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a' \rfloor)$›
and $?\mathfrak{G}'b'$ = ‹$\mathfrak{G}'(\lfloor ObjMap \rfloor)(\lfloor b' \rfloor)$›

interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}'\ \mathfrak{G}'\ \psi$ **by** (*rule assms*(*2*))
interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)

from *assms*(*3*) **have** *g*: $g : a' \mapsto_{\mathfrak{A}} a$ **unfolding** *cat-op-simps* **by** *simp*

from *Set.category-axioms category-axioms assms g* **have** *a'b-Gg𝔉'f*:
?Y-a'b' $\circ_{A\, cat\text{-}Set\ \alpha}$ *cf-hom* $\mathfrak{C}\ [?\mathfrak{G}g,\ ?\mathfrak{F}'f]_\circ$:
Hom $\mathfrak{C}\ ?\mathfrak{G}a\ ?\mathfrak{F}'b \mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}\ ?\mathfrak{F}a'\ ?\mathfrak{G}'b'$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
then have *dom-lhs*:
$\mathcal{D}_\circ$ ((*?Y-a'b'* $\circ_{A\, cat\text{-}Set\ \alpha}$ *cf-hom* $\mathfrak{C}\ [?\mathfrak{G}g,\ ?\mathfrak{F}'f]_\circ$)$(\lfloor ArrVal \rfloor)$ =
Hom $\mathfrak{C}\ ?\mathfrak{G}a\ ?\mathfrak{F}'b$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *Set.category-axioms category-axioms assms g* **have** *𝔉g𝔊'f-ab*:
cf-hom $\mathfrak{C}\ [?\mathfrak{F}g,\ ?\mathfrak{G}'f]_\circ \circ_{A\, cat\text{-}Set\ \alpha}$ *?Y-ab* :
Hom $\mathfrak{C}\ ?\mathfrak{G}a\ ?\mathfrak{F}'b \mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}\ ?\mathfrak{F}a'\ ?\mathfrak{G}'b'$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
then have *dom-rhs*:
$\mathcal{D}_\circ$ ((*cf-hom* $\mathfrak{C}\ [?\mathfrak{F}g,\ ?\mathfrak{G}'f]_\circ \circ_{A\, cat\text{-}Set\ \alpha}$ *?Y-ab*)$(\lfloor ArrVal \rfloor)$ =
Hom $\mathfrak{C}\ ?\mathfrak{G}a\ ?\mathfrak{F}'b$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)

```
  show ?thesis
  proof(rule arr-Set-eqI[of α])
    from a'b-Gg𝔉'f show arr-Set-a'b-Gg𝔉'f:
      arr-Set α (?Y-a'b' ∘_A cat-Set α cf-hom ℭ [?𝔊g, ?𝔉'f]_∘)
      by (auto dest: cat-Set-is-arrD(1))
    from 𝔉g𝔊'f-ab show arr-Set-𝔉g𝔊'f-ab:
      arr-Set α (cf-hom ℭ [?𝔉g, ?𝔊'f]_∘ ∘_A cat-Set α ?Y-ab)
      by (auto dest: cat-Set-is-arrD(1))
    show
      (?Y-a'b' ∘_A cat-Set α cf-hom ℭ [?𝔊g, ?𝔉'f]_∘)(|ArrVal|) =
        (cf-hom ℭ [?𝔉g, ?𝔊'f]_∘ ∘_A cat-Set α ?Y-ab)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix h assume prems: h : 𝔊(|ObjMap|)(|a|) ↦_ℭ 𝔉'(|ObjMap|)(|b|)
      from assms(1,2) g have [cat-cs-simps]:
        ψ(|NTMap|)(|b'|) ∘_Aℭ (?𝔉'f ∘_Aℭ (h ∘_Aℭ (?𝔊g ∘_Aℭ φ(|NTMap|)(|a'|)))) =
          ψ(|NTMap|)(|b'|) ∘_Aℭ (?𝔉'f ∘_Aℭ (h ∘_Aℭ (φ(|NTMap|)(|a|) ∘_Aℭ ?𝔉g)))
        by
          (
            cs-concl cs-shallow
              cs-simp: is-ntcf.ntcf-Comp-commute cs-intro: cat-cs-intros
          )
        also from assms(1,2,4) prems g have ... =
          (((ψ(|NTMap|)(|b'|) ∘_Aℭ ?𝔉'f) ∘_Aℭ h) ∘_Aℭ φ(|NTMap|)(|a|)) ∘_Aℭ ?𝔉g
          by (cs-concl cs-shallow cs-simp: cat-Comp-assoc cs-intro: cat-cs-intros)
        also from assms(1,2,4) have ... =
          (((?𝔊'f ∘_Aℭ ψ(|NTMap|)(|b|)) ∘_Aℭ h) ∘_Aℭ φ(|NTMap|)(|a|)) ∘_Aℭ ?𝔉g
          by
            (
              cs-concl cs-shallow
                cs-simp: is-ntcf.ntcf-Comp-commute cs-intro: cat-cs-intros
            )
        also from assms(1,2,4) prems g have ... =
          ?𝔊'f ∘_Aℭ (ψ(|NTMap|)(|b|) ∘_Aℭ (h ∘_Aℭ (φ(|NTMap|)(|a|) ∘_Aℭ ?𝔉g)))
          by (cs-concl cs-simp: cat-Comp-assoc cs-intro: cat-cs-intros)
        finally have nat:
          ψ(|NTMap|)(|b'|) ∘_Aℭ (?𝔉'f ∘_Aℭ (h ∘_Aℭ (?𝔊g ∘_Aℭ φ(|NTMap|)(|a'|)))) =
            ?𝔊'f ∘_Aℭ (ψ(|NTMap|)(|b|) ∘_Aℭ (h ∘_Aℭ (φ(|NTMap|)(|a|) ∘_Aℭ ?𝔉g))).
        from prems Set.category-axioms category-axioms assms(1,2,4) g show
          (?Y-a'b' ∘_A cat-Set α cf-hom ℭ [?𝔊g, ?𝔉'f]_∘)(|ArrVal|)(|h|) =
            (cf-hom ℭ [?𝔉g, ?𝔊'f]_∘ ∘_A cat-Set α ?Y-ab)(|ArrVal|)(|h|)
          by
            (
              cs-concl
                cs-simp: nat cat-cs-simps cat-op-simps
                cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
            )
      qed (use arr-Set-a'b-Gg𝔉'f arr-Set-𝔉g𝔊'f-ab in auto)

    qed (use a'b-Gg𝔉'f 𝔉g𝔊'f-ab in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed
```

29.6.6 Composition of the components of a composition of a *Hom*-natural transformation with natural transformations

```
lemma (in category) cat-ntcf-Hom-component-Comp:
  assumes φ' : 𝔊 ↦_CF 𝔎 : 𝔄 ↦↦_Cα ℭ
```

```
    and φ : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα ℭ
    and ψ' : 𝔊' ↦_CF ℌ' : 𝔅 ↦↦_Cα ℭ
    and ψ : 𝔉' ↦_CF 𝔊' : 𝔅 ↦↦_Cα ℭ
    and a ∈_∘ 𝔄(|Obj|)
    and b ∈_∘ 𝔅(|Obj|)
  shows
    ntcf-Hom-component φ ψ' a b ∘_A cat-Set α ntcf-Hom-component φ' ψ a b =
      ntcf-Hom-component (φ' ⊙_NTCF φ) (ψ' ⊙_NTCF ψ) a b
    (is ‹?φψ' ∘_A cat-Set α ?φ'ψ = ?φ'φψ'ψ›)
proof–

  interpret Set: category α ‹cat-Set α› by (rule category-cat-Set)

  from assms Set.category-axioms category-axioms have φψ'-φ'ψ:
    ?φψ' ∘_A cat-Set α ?φ'ψ :
      Hom ℭ (ℌ(|ObjMap|)(|a|)) (𝔉'(|ObjMap|)(|b|)) ↦_cat-Set α
      Hom ℭ (𝔉(|ObjMap|)(|a|)) (ℌ'(|ObjMap|)(|b|))
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)
  then have dom-lhs:
    𝒟_∘ ((?φψ' ∘_A cat-Set α ?φ'ψ)(|ArrVal|)) =
      Hom ℭ (ℌ(|ObjMap|)(|a|)) (𝔉'(|ObjMap|)(|b|))
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms Set.category-axioms category-axioms have φ'φψ'ψ:
    ?φ'φψ'ψ :
      Hom ℭ (ℌ(|ObjMap|)(|a|)) (𝔉'(|ObjMap|)(|b|)) ↦_cat-Set α
      Hom ℭ (𝔉(|ObjMap|)(|a|)) (ℌ'(|ObjMap|)(|b|))
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-op-intros)
  then have dom-rhs:
    𝒟_∘ (?φ'φψ'ψ(|ArrVal|)) = Hom ℭ (ℌ(|ObjMap|)(|a|)) (𝔉'(|ObjMap|)(|b|))
    by (cs-concl cs-simp: cat-cs-simps)

  show ?thesis
  proof(rule arr-Set-eqI[of α])
    from φψ'-φ'ψ show arr-Set-φψ'-φ'ψ: arr-Set α (?φψ' ∘_A cat-Set α ?φ'ψ)
      by (auto dest: cat-Set-is-arrD(1))
    from φ'φψ'ψ show arr-Set-φ'φψ'ψ: arr-Set α ?φ'φψ'ψ
      by (auto dest: cat-Set-is-arrD(1))
    show (?φψ' ∘_A cat-Set α ?φ'ψ)(|ArrVal|) = ?φ'φψ'ψ(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix f assume f : ℌ(|ObjMap|)(|a|) ↦_ℭ 𝔉'(|ObjMap|)(|b|)
      with category-axioms assms Set.category-axioms show
        (?φψ' ∘_A cat-Set α ?φ'ψ)(|ArrVal|)(|f|) = ?φ'φψ'ψ(|ArrVal|)(|f|)
        by
          (
            cs-concl cs-shallow
              cs-simp: cat-cs-simps
              cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
          )
    qed (use arr-Set-φ'φψ'ψ arr-Set-φψ'-φ'ψ in auto)

  qed (use φψ'-φ'ψ φ'φψ'ψ in ‹cs-concl cs-simp: cat-cs-simps›)+

qed

lemmas [cat-cs-simps] = category.cat-ntcf-Hom-component-Comp
```

29.6.7 Component of a composition of *Hom*-natural transformation with the identity natural transformations

lemma (**in** *category*) *cat-ntcf-Hom-component-ntcf-id*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{F}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows
ntcf-Hom-component (*ntcf-id* $\mathfrak{F}$) (*ntcf-id* $\mathfrak{F}'$) a b =
cat-Set $\alpha(\!|CId|\!)(\!|Hom\ \mathfrak{C}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!))|\!)$
(**is** $\langle ?\mathfrak{F}\mathfrak{F}' = cat\text{-}Set\ \alpha(\!|CId|\!)(\!|?\mathfrak{F}a\mathfrak{F}'b|\!)\rangle$)
proof–

interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}'$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ **by** (*rule assms*(*2*))
interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)

from *assms Set.category-axioms category-axioms* **have** $\mathfrak{F}\mathfrak{F}'$:
$?\mathfrak{F}\mathfrak{F}'$:
Hom $\mathfrak{C}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!)) \mapsto_{cat\text{-}Set\ \alpha}$
Hom $\mathfrak{C}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!))$
by (*cs-concl* **cs-intro**: *cat-cs-intros cat-op-intros*)
then have *dom-lhs*: $\mathcal{D}_\circ$ $(?\mathfrak{F}\mathfrak{F}'(\!|ArrVal|\!))$ = *Hom* $\mathfrak{C}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!))$
by (*cs-concl* **cs-simp**: *cat-cs-simps*)

from *category-axioms assms Set.category-axioms* **have** $\mathfrak{F}a\mathfrak{F}'b$:
cat-Set $\alpha(\!|CId|\!)(\!|?\mathfrak{F}a\mathfrak{F}'b|\!)$:
Hom $\mathfrak{C}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!)) \mapsto_{cat\text{-}Set\ \alpha}$
Hom $\mathfrak{C}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!))$
by
(
cs-concl
cs-simp: *cat-Set-cs-simps cat-Set-components*(*1*)
cs-intro: *cat-cs-intros*
)
then have *dom-rhs*:
$\mathcal{D}_\circ$ (*cat-Set* $\alpha(\!|CId|\!)(\!|?\mathfrak{F}a\mathfrak{F}'b|\!)(\!|ArrVal|\!)$) = *Hom* $\mathfrak{C}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!))$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)

show *?thesis*
proof(*rule arr-Set-eqI*[*of* α])
from $\mathfrak{F}\mathfrak{F}'$ **show** *arr-Set-*$\mathfrak{F}\psi$: *arr-Set* α $?\mathfrak{F}\mathfrak{F}'$
by (*auto dest*: *cat-Set-is-arrD*(*1*))
from $\mathfrak{F}a\mathfrak{F}'b$ **show** *arr-Set-*$\mathfrak{F}a\mathfrak{F}'b$: *arr-Set* α (*cat-Set* $\alpha(\!|CId|\!)(\!|?\mathfrak{F}a\mathfrak{F}'b|\!)$)
by (*auto dest*: *cat-Set-is-arrD*(*1*))
show $?\mathfrak{F}\mathfrak{F}'(\!|ArrVal|\!)$ = *cat-Set* $\alpha(\!|CId|\!)(\!|?\mathfrak{F}a\mathfrak{F}'b|\!)(\!|ArrVal|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs in-Hom-iff*)
fix f **assume** $f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{C}} \mathfrak{F}'(\!|ObjMap|\!)(\!|b|\!)$
with *category-axioms Set.category-axioms assms* **show**
$?\mathfrak{F}\mathfrak{F}'(\!|ArrVal|\!)(\!|f|\!)$ = *cat-Set* $\alpha(\!|CId|\!)(\!|?\mathfrak{F}a\mathfrak{F}'b|\!)(\!|ArrVal|\!)(\!|f|\!)$
by
(
cs-concl
cs-simp: *cat-cs-simps cat-Set-components*(*1*)
cs-intro: *cat-cs-intros*
)
qed (*use arr-Set-*$\mathfrak{F}a\mathfrak{F}'b$ **in** *auto*)

qed (*use* $\mathfrak{F}\mathfrak{F}'\ \mathfrak{F}a\mathfrak{F}'b$ **in** ‹*cs-concl cs-simp*: *cat-cs-simps*›)+

qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-Hom-component-ntcf-id*

29.7 Component of a composition of a *Hom*-natural transformation with a natural transformation

29.7.1 Definition and elementary properties

definition *ntcf-lcomp-Hom-component* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-lcomp-Hom-component* $\varphi\ a\ b$ =
    ntcf-Hom-component φ (*ntcf-id* (*cf-id* (φ(|*NTDGCod*|)))) $a\ b$

definition *ntcf-rcomp-Hom-component* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-rcomp-Hom-component* $\psi\ a\ b$ =
    ntcf-Hom-component (*ntcf-id* (*cf-id* (ψ(|*NTDGCod*|)))) $\psi\ a\ b$

29.7.2 Arrow value

lemma *ntcf-lcomp-Hom-component-ArrVal-vsv*:
  vsv (*ntcf-lcomp-Hom-component* $\varphi\ a\ b$(|*ArrVal*|))
  unfolding *ntcf-lcomp-Hom-component-def* **by** (*rule ntcf-Hom-component-ArrVal-vsv*)

lemma *ntcf-rcomp-Hom-component-ArrVal-vsv*:
  vsv (*ntcf-rcomp-Hom-component* $\psi\ a\ b$(|*ArrVal*|))
  unfolding *ntcf-rcomp-Hom-component-def* **by** (*rule ntcf-Hom-component-ArrVal-vsv*)

lemma *ntcf-lcomp-Hom-component-ArrVal-vdomain*[*cat-cs-simps*]:
  assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $b \in_\circ \mathfrak{C}$(|*Obj*|)
  shows $\mathcal{D}_\circ$ (*ntcf-lcomp-Hom-component* $\varphi\ a\ b$(|*ArrVal*|)) = *Hom* $\mathfrak{C}$ ($\mathfrak{G}$(|*ObjMap*|)(|a|)) b
proof–
  interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
  show *?thesis*
    using *assms*
    unfolding *ntcf-lcomp-Hom-component-def* φ.*ntcf-NTDGCod*
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-rcomp-Hom-component-ArrVal-vdomain*[*cat-cs-simps*]:
  assumes $\psi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $a \in_\circ$ *op-cat* $\mathfrak{C}$(|*Obj*|)
  shows $\mathcal{D}_\circ$ (*ntcf-rcomp-Hom-component* $\psi\ a\ b$(|*ArrVal*|)) = *Hom* $\mathfrak{C}\ a$ ($\mathfrak{F}$(|*ObjMap*|)(|b|))
proof–
  interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \psi$ **by** (*rule assms*(*1*))
  show *?thesis*
    using *assms*
    unfolding *cat-op-simps ntcf-rcomp-Hom-component-def* ψ.*ntcf-NTDGCod*
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-lcomp-Hom-component-ArrVal-app*[*cat-cs-simps*]:
  assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $a \in_\circ$ *op-cat* $\mathfrak{A}$(|*Obj*|)
    and $b \in_\circ \mathfrak{C}$(|*Obj*|)
    and $h : \mathfrak{G}$(|*ObjMap*|)(|a|) $\mapsto_{\mathfrak{C}} b$
  shows *ntcf-lcomp-Hom-component* $\varphi\ a\ b$(|*ArrVal*|)(|h|) = $h \circ_{A\mathfrak{C}} \varphi$(|*NTMap*|)(|$a$|)
proof–

interpret φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule assms*(*1*))
show *?thesis*
using *assms*
unfolding *cat-op-simps ntcf-lcomp-Hom-component-def* φ*.ntcf-NTDGCod*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-rcomp-Hom-component-ArrVal-app*[*cat-cs-simps*]:
assumes $\psi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $h : a \mapsto_{\mathfrak{C}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
shows *ntcf-rcomp-Hom-component* ψ a $b(\!|ArrVal|\!)(\!|h|\!) = \psi(\!|NTMap|\!)(\!|b|\!) \circ_{A\mathfrak{C}} h$
proof–
interpret ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ ψ **by** (*rule assms*(*1*))
show *?thesis*
using *assms*
unfolding *cat-op-simps ntcf-rcomp-Hom-component-def* ψ*.ntcf-NTDGCod*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-lcomp-Hom-component-ArrVal-vrange*:
assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ$ *op-cat* $\mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows $\mathcal{R}_\circ$ (*ntcf-lcomp-Hom-component* φ a $b(\!|ArrVal|\!)$) $\subseteq_\circ$ *Hom* $\mathfrak{C}$ ($\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$) b
proof–
interpret φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule assms*(*1*))
from *assms*(*2*) **have** *a*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **unfolding** *cat-op-simps* **by** *simp*
from *assms*(*1*,*3*) *a* **have**
$\mathcal{R}_\circ$ (*ntcf-lcomp-Hom-component* φ a $b(\!|ArrVal|\!)$) $\subseteq_\circ$
Hom $\mathfrak{C}$ ($\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$) (*cf-id* $\mathfrak{C}(\!|ObjMap|\!)(\!|b|\!)$)
by
(
unfold cat-op-simps ntcf-lcomp-Hom-component-def φ*.ntcf-NTDGCod*,
intro ntcf-Hom-component-ArrVal-vrange
)
(*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)+
from *this assms*(*3*) **show** *?thesis* **by** (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps*)
qed

lemma *ntcf-rcomp-Hom-component-ArrVal-vrange*:
assumes $\psi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_\circ$ *op-cat* $\mathfrak{C}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $\mathcal{R}_\circ$ (*ntcf-rcomp-Hom-component* ψ a $b(\!|ArrVal|\!)$) $\subseteq_\circ$ *Hom* $\mathfrak{C}$ a ($\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$)
proof–
interpret ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ ψ **by** (*rule assms*(*1*))
from *assms*(*2*) **have** *a*: $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **unfolding** *cat-op-simps* **by** *simp*
from *assms*(*1*,*3*) *a* **have**
$\mathcal{R}_\circ$ (*ntcf-rcomp-Hom-component* ψ a $b(\!|ArrVal|\!)$) $\subseteq_\circ$
Hom $\mathfrak{C}$ (*cf-id* $\mathfrak{C}(\!|ObjMap|\!)(\!|a|\!)$) ($\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!)$)
by
(
unfold ntcf-rcomp-Hom-component-def ψ*.ntcf-NTDGCod*,
intro ntcf-Hom-component-ArrVal-vrange
)
(*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

from *this a* **show** *?thesis* **by** (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps*)
qed

29.7.3 Arrow domain and codomain

lemma *ntcf-lcomp-Hom-component-ArrDom*[*cat-cs-simps*]:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-lcomp-Hom-component* φ a $b(\!|ArrDom|\!) = Hom\ \mathfrak{C}\ (\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!))\ b$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
from *assms* **show** *?thesis*
unfolding *ntcf-lcomp-Hom-component-def* φ.*ntcf-NTDGCod*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-rcomp-Hom-component-ArrDom*[*cat-cs-simps*]:
assumes $\psi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-rcomp-Hom-component* ψ a $b(\!|ArrDom|\!) = Hom\ \mathfrak{C}\ a\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
proof–
interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \psi$ **by** (*rule assms*(*1*))
from *assms* **show** *?thesis*
unfolding *cat-op-simps ntcf-rcomp-Hom-component-def* ψ.*ntcf-NTDGCod*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-lcomp-Hom-component-ArrCod*[*cat-cs-simps*]:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-lcomp-Hom-component* φ a $b(\!|ArrCod|\!) = Hom\ \mathfrak{C}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ b$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
from *assms* **show** *?thesis*
unfolding *ntcf-lcomp-Hom-component-def* φ.*ntcf-NTDGCod*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-rcomp-Hom-component-ArrCod*[*cat-cs-simps*]:
assumes $\psi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $a \in_{\circ} op\text{-}cat\ \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-rcomp-Hom-component* ψ a $b(\!|ArrCod|\!) = Hom\ \mathfrak{C}\ a\ (\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))$
proof–
interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \psi$ **by** (*rule assms*(*1*))
from *assms* **show** *?thesis*
unfolding *cat-op-simps ntcf-rcomp-Hom-component-def* ψ.*ntcf-NTDGCod*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

29.7.4 Component of a composition of a *Hom*-natural transformation with a natural transformation is an arrow in the category *Set*

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-component-is-arr*:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $a \in_{\circ} op\text{-}cat\ \mathfrak{A}(\!|Obj|\!)$
and $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-lcomp-Hom-component* φ a b :
$Hom\ \mathfrak{C}\ (\mathfrak{G}(\!|ObjMap|\!)(\!|a|\!))\ b \mapsto_{cat\text{-}Set\ \alpha} Hom\ \mathfrak{C}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ b$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
from *assms* **have** *a*: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ **unfolding** *cat-op-simps* **by** *simp*
from *assms*(*1*,*3*) *a* **have**

```
    ntcf-lcomp-Hom-component φ a b :
      Hom ℭ (𝔊(‖ObjMap‖)(‖a‖)) (cf-id ℭ(‖ObjMap‖)(‖b‖)) ↦_cat-Set α
      Hom ℭ (𝔉(‖ObjMap‖)(‖a‖)) (cf-id ℭ(‖ObjMap‖)(‖b‖))
    unfolding ntcf-lcomp-Hom-component-def φ.ntcf-NTDGCod
    by (intro cat-ntcf-Hom-component-is-arr
      (cs-concl cs-intro: cat-cs-intros cat-op-intros)+
  from this assms(1,3) a show ?thesis
    by (cs-prems cs-shallow cs-simp: cat-cs-simps)
qed

lemma (in category) cat-ntcf-lcomp-Hom-component-is-arr':
  assumes φ : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα ℭ
    and a ∈₀ op-cat 𝔄(‖Obj‖)
    and b ∈₀ ℭ(‖Obj‖)
    and 𝔄' = Hom ℭ (𝔊(‖ObjMap‖)(‖a‖)) b
    and 𝔅' = Hom ℭ (𝔉(‖ObjMap‖)(‖a‖)) b
    and ℭ' = cat-Set α
  shows ntcf-lcomp-Hom-component φ a b : 𝔄' ↦_ℭ' 𝔅'
  using assms(1–3)
  unfolding assms(4–6)
  by (rule cat-ntcf-lcomp-Hom-component-is-arr)

lemmas [cat-cs-intros] = category.cat-ntcf-lcomp-Hom-component-is-arr'

lemma (in category) cat-ntcf-rcomp-Hom-component-is-arr:
  assumes ψ : 𝔉 ↦_CF 𝔊 : 𝔅 ↦↦_Cα ℭ
    and a ∈₀ op-cat ℭ(‖Obj‖)
    and b ∈₀ 𝔅(‖Obj‖)
  shows ntcf-rcomp-Hom-component ψ a b :
    Hom ℭ a (𝔉(‖ObjMap‖)(‖b‖)) ↦_cat-Set α Hom ℭ a (𝔊(‖ObjMap‖)(‖b‖))
proof–
  interpret ψ: is-ntcf α 𝔅 ℭ 𝔉 𝔊 ψ by (rule assms(1))
  from assms have a: a ∈₀ ℭ(‖Obj‖) unfolding cat-op-simps by simp
  from assms(1,3) a have
    ntcf-rcomp-Hom-component ψ a b :
      Hom ℭ (cf-id ℭ(‖ObjMap‖)(‖a‖)) (𝔉(‖ObjMap‖)(‖b‖)) ↦_cat-Set α
      Hom ℭ (cf-id ℭ(‖ObjMap‖)(‖a‖)) (𝔊(‖ObjMap‖)(‖b‖))
    unfolding ntcf-rcomp-Hom-component-def ψ.ntcf-NTDGCod
    by (intro cat-ntcf-Hom-component-is-arr)
      (cs-concl cs-intro: cat-cs-intros cat-op-intros)
  from this assms(1,3) a show ?thesis
    by (cs-prems cs-shallow cs-simp: cat-cs-simps)
qed

lemma (in category) cat-ntcf-rcomp-Hom-component-is-arr':
  assumes ψ : 𝔉 ↦_CF 𝔊 : 𝔅 ↦↦_Cα ℭ
    and a ∈₀ op-cat ℭ(‖Obj‖)
    and b ∈₀ 𝔅(‖Obj‖)
    and 𝔄' = Hom ℭ a (𝔉(‖ObjMap‖)(‖b‖))
    and 𝔅' = Hom ℭ a (𝔊(‖ObjMap‖)(‖b‖))
    and ℭ' = cat-Set α
  shows ntcf-rcomp-Hom-component ψ a b : 𝔄' ↦_ℭ' 𝔅'
  using assms(1–3)
  unfolding assms(4–6)
  by (rule cat-ntcf-rcomp-Hom-component-is-arr)

lemmas [cat-cs-intros] = category.cat-ntcf-rcomp-Hom-component-is-arr'
```

29.8 Composition of a *Hom*-natural transformation with two natural transformations

29.8.1 Definition and elementary properties

See subsection 1.15 in [3].

definition *ntcf-Hom* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ ($\langle Hom_{A.C}1'(/--,--/')\rangle$)
  where $Hom_{A.C\alpha}(\varphi-,\psi-)$ =
    [
      (
        $\lambda ab\in_\circ$(*op-cat* ($\varphi(\!|NTDGDom|\!)$) $\times_C$ $\psi(\!|NTDGDom|\!)$)$(\!|Obj|\!)$.
          ntcf-Hom-component φ ψ (*vpfst ab*) (*vpsnd ab*)
      ),
      $Hom_{O.C\alpha}\psi(\!|NTDGCod|\!)(\varphi(\!|NTCod|\!)-,\psi(\!|NTDom|\!)-)$,
      $Hom_{O.C\alpha}\psi(\!|NTDGCod|\!)(\varphi(\!|NTDom|\!)-,\psi(\!|NTCod|\!)-)$,
      op-cat ($\varphi(\!|NTDGDom|\!)$) $\times_C$ $\psi(\!|NTDGDom|\!)$,
      cat-Set α
    ]$_\circ$

Components.

lemma *ntcf-Hom-components*:
  shows $Hom_{A.C\alpha}(\varphi-,\psi-)(\!|NTMap|\!)$ =
    (
      $\lambda ab\in_\circ$(*op-cat* ($\varphi(\!|NTDGDom|\!)$) $\times_C$ $\psi(\!|NTDGDom|\!)$)$(\!|Obj|\!)$.
        ntcf-Hom-component φ ψ (*vpfst ab*) (*vpsnd ab*)
)
    and $Hom_{A.C\alpha}(\varphi-,\psi-)(\!|NTDom|\!)$ =
      $Hom_{O.C\alpha}\psi(\!|NTDGCod|\!)(\varphi(\!|NTCod|\!)-,\psi(\!|NTDom|\!)-)$
    and $Hom_{A.C\alpha}(\varphi-,\psi-)(\!|NTCod|\!)$ =
      $Hom_{O.C\alpha}\psi(\!|NTDGCod|\!)(\varphi(\!|NTDom|\!)-,\psi(\!|NTCod|\!)-)$
    and $Hom_{A.C\alpha}(\varphi-,\psi-)(\!|NTDGDom|\!)$ = *op-cat* ($\varphi(\!|NTDGDom|\!)$) $\times_C$ $\psi(\!|NTDGDom|\!)$
    and $Hom_{A.C\alpha}(\varphi-,\psi-)(\!|NTDGCod|\!)$ = *cat-Set* α
  unfolding *ntcf-Hom-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

29.8.2 Natural transformation map

mk-VLambda *ntcf-Hom-components*(*1*)
  |*vsv ntcf-Hom-NTMap-vsv*|

context
  fixes α φ ψ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
  assumes φ: $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and ψ: $\psi : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule* φ)
interpretation ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ ψ **by** (*rule* ψ)

mk-VLambda *ntcf-Hom-components*(*1*)[*of* - φ ψ, *simplified*]
  |*vdomain ntcf-Hom-NTMap-vdomain*[*unfolded in-Hom-iff*]|

lemmas [*cat-cs-simps*] = *ntcf-Hom-NTMap-vdomain*

lemma *ntcf-Hom-NTMap-app*[*cat-cs-simps*]:
  assumes $[a, b]_\circ \in_\circ$ (*op-cat* $\mathfrak{A}$ $\times_C$ $\mathfrak{B}$)$(\!|Obj|\!)$
  shows $Hom_{A.C\alpha}(\varphi-,\psi-)(\!|NTMap|\!)(\!|a, b|\!)_\bullet$ = *ntcf-Hom-component* φ ψ *a b*
  using *assms*
  unfolding *ntcf-Hom-components*

```
  by (simp add: nat-omega-simps cat-cs-simps)

end

lemma (in category) ntcf-Hom-NTMap-vrange:
  assumes φ : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{Cα} ℭ and ψ : 𝔉' ↦_{CF} 𝔊' : 𝔅 ↦↦_{Cα} ℭ
  shows ℛ∘ (Hom_{A.Cα}(φ-,ψ-)(|NTMap|)) ⊆∘ cat-Set α(|Arr|)
proof–
  interpret φ: is-ntcf α 𝔄 ℭ 𝔉 𝔊 φ by (rule assms(1))
  interpret ψ: is-ntcf α 𝔅 ℭ 𝔉' 𝔊' ψ by (rule assms(2))
  show ?thesis
  proof
    (
      rule vsv.vsv-vrange-vsubset,
      unfold ntcf-Hom-NTMap-vdomain[OF assms] cat-cs-simps
    )
    fix ab assume ab ∈∘ (op-cat 𝔄 ×_C 𝔅)(|Obj|)
    then obtain a b
      where ab-def: ab = [a, b]∘
        and a: a ∈∘ op-cat 𝔄(|Obj|)
        and b: b ∈∘ 𝔅(|Obj|)
      by
        (
          rule cat-prod-2-ObjE[
            OF φ.NTDom.HomDom.category-op ψ.NTDom.HomDom.category-axioms
            ]
        )
    from assms a b category-cat-Set category-axioms show
      Hom_{A.Cα}(φ-,ψ-)(|NTMap|)(|ab|) ∈∘ cat-Set α(|Arr|)
      unfolding ab-def cat-op-simps
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
  qed (simp add: ntcf-Hom-NTMap-vsv)
qed
```

29.8.3 Composition of a *Hom*-natural transformation with two natural transformations is a natural transformation

```
lemma (in category) cat-ntcf-Hom-is-ntcf:
  assumes φ : 𝔉 ↦_{CF} 𝔊 : 𝔄 ↦↦_{Cα} ℭ and ψ : 𝔉' ↦_{CF} 𝔊' : 𝔅 ↦↦_{Cα} ℭ
  shows Hom_{A.Cα}(φ-,ψ-) :
    Hom_{O.Cα}ℭ(𝔊-,𝔉'-) ↦_{CF} Hom_{O.Cα}ℭ(𝔉-,𝔊'-) :
    op-cat 𝔄 ×_C 𝔅 ↦↦_{Cα} cat-Set α
proof–

  interpret φ: is-ntcf α 𝔄 ℭ 𝔉 𝔊 φ by (rule assms(1))
  interpret ψ: is-ntcf α 𝔅 ℭ 𝔉' 𝔊' ψ by (rule assms(2))

  show ?thesis
  proof(intro is-ntcfI')
    show vfsequence (Hom_{A.Cα}(φ-,ψ-)) unfolding ntcf-Hom-def by simp
    show vcard (Hom_{A.Cα}(φ-,ψ-)) = 5_ℕ
      unfolding ntcf-Hom-def by (simp add: nat-omega-simps)
    from assms category-axioms show
```

```
  Hom_{O.Cα}𝔈(𝔊−,𝔉′−) : op-cat 𝔄 ×_C 𝔅 ↦↦_{Cα} cat-Set α
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
from assms category-axioms show
  Hom_{O.Cα}𝔈(𝔉−,𝔊′−) : op-cat 𝔄 ×_C 𝔅 ↦↦_{Cα} cat-Set α
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
from assms show 𝔇∘ (Hom_{A.Cα}(φ−,ψ−)(⌊NTMap⌋)) = (op-cat 𝔄 ×_C 𝔅)(⌊Obj⌋)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
show Hom_{A.Cα}(φ−,ψ−)(⌊NTMap⌋)(⌊ab⌋) :
  Hom_{O.Cα}𝔈(𝔊−,𝔉′−)(⌊ObjMap⌋)(⌊ab⌋) ↦_{cat-Set α}
  Hom_{O.Cα}𝔈(𝔉−,𝔊′−)(⌊ObjMap⌋)(⌊ab⌋)
  if ab ∈∘ (op-cat 𝔄 ×_C 𝔅)(⌊Obj⌋) for ab
proof–
  from that obtain a b
    where ab-def: ab = [a, b]∘
      and a: a ∈∘ op-cat 𝔄(⌊Obj⌋)
      and b: b ∈∘ 𝔅(⌊Obj⌋)
    by
      (
        rule cat-prod-2-ObjE[
          OF φ.NTDom.HomDom.category-op ψ.NTDom.HomDom.category-axioms
          ]
      )
  from category-axioms assms a b show
    Hom_{A.Cα}(φ−,ψ−)(⌊NTMap⌋)(⌊ab⌋) :
      Hom_{O.Cα}𝔈(𝔊−,𝔉′−)(⌊ObjMap⌋)(⌊ab⌋) ↦_{cat-Set α}
      Hom_{O.Cα}𝔈(𝔉−,𝔊′−)(⌊ObjMap⌋)(⌊ab⌋)
    unfolding ab-def cat-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed
show
  Hom_{A.Cα}(φ−,ψ−)(⌊NTMap⌋)(⌊a′b′⌋) ∘_{A cat-Set α}
   Hom_{O.Cα}𝔈(𝔊−,𝔉′−)(⌊ArrMap⌋)(⌊gf⌋) =
    Hom_{O.Cα}𝔈(𝔉−,𝔊′−)(⌊ArrMap⌋)(⌊gf⌋) ∘_{A cat-Set α}
    Hom_{A.Cα}(φ−,ψ−)(⌊NTMap⌋)(⌊ab⌋)
  if gf : ab ↦_{op-cat 𝔄 ×_C 𝔅} a′b′ for ab a′b′ gf
proof–
  from that obtain g f a b a′ b′
    where gf-def: gf = [g, f]∘
      and ab-def: ab = [a, b]∘
      and a′b′-def: a′b′ = [a′, b′]∘
      and g: g : a ↦_{op-cat 𝔄} a′
      and f: f : b ↦_𝔅 b′
    by
      (
        elim
          cat-prod-2-is-arrE[
            OF φ.NTDom.HomDom.category-op ψ.NTDom.HomDom.category-axioms
            ]
      )
  from assms category-axioms that g f show ?thesis
    unfolding gf-def ab-def a′b′-def cat-op-simps
    by
      (
```

cs-concl
cs-simp: *cat-ntcf-Hom-component-nat cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-op-intros cat-prod-cs-intros*
)
qed
qed (*auto simp*: *ntcf-Hom-components cat-cs-simps*)

qed

lemma (**in** *category*) *cat-ntcf-Hom-is-ntcf'*:
assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\psi : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\beta = \alpha$
and $\mathfrak{A}' = \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,\mathfrak{F}'-)$
and $\mathfrak{B}' = \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{G}'-)$
and $\mathfrak{C}' = \mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{B}$
and $\mathfrak{D}' = \mathit{cat\text{-}Set}\ \alpha$
shows $\mathit{Hom}_{A.C\alpha}(\varphi-,\psi-) : \mathfrak{A}' \mapsto_{CF} \mathfrak{B}' : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}'$
using *assms*(*1–2*) **unfolding** *assms*(*3–7*) **by** (*rule cat-ntcf-Hom-is-ntcf*)

lemmas [*cat-cs-intros*] = *category.cat-ntcf-Hom-is-ntcf'*

29.8.4 Composition of a *Hom*-natural transformation with two vertical compositions of natural transformations

lemma (**in** *category*) *cat-ntcf-Hom-vcomp*:
assumes $\varphi' : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\psi' : \mathfrak{G}' \mapsto_{CF} \mathfrak{H}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\psi : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows
$\mathit{Hom}_{A.C\alpha}(\varphi' \cdot_{NTCF} \varphi-,\psi' \cdot_{NTCF} \psi-) =$
$\mathit{Hom}_{A.C\alpha}(\varphi-,\psi'-) \cdot_{NTCF} \mathit{Hom}_{A.C\alpha}(\varphi'-,\psi-)$
proof(*rule ntcf-eqI*[*of* α])

interpret φ': *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{G}\ \mathfrak{H}\ \varphi'$ **by** (*rule assms*(*1*))
interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*2*))
interpret ψ': *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{G}'\ \mathfrak{H}'\ \psi'$ **by** (*rule assms*(*3*))
interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}'\ \mathfrak{G}'\ \psi$ **by** (*rule assms*(*4*))

from *category-axioms assms* **show** *H-vcomp*:
$\mathit{Hom}_{A.C\alpha}(\varphi' \cdot_{NTCF} \varphi-,\psi' \cdot_{NTCF} \psi-) :$
$\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{H}-,\mathfrak{F}'-) \mapsto_{CF} \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{H}'-) :$
$\mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathit{cat\text{-}Set}\ \alpha$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *category-axioms assms* **show** *vcomp-H*:
$\mathit{Hom}_{A.C\alpha}(\varphi-,\psi'-) \cdot_{NTCF} \mathit{Hom}_{A.C\alpha}(\varphi'-,\psi-) :$
$\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{H}-,\mathfrak{F}'-) \mapsto_{CF} \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,\mathfrak{H}'-) :$
$\mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathit{cat\text{-}Set}\ \alpha$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *category-axioms assms H-vcomp* **have** *dom-H-vcomp*:
$\mathcal{D}_\circ\ (\mathit{Hom}_{A.C\alpha}(\varphi' \cdot_{NTCF} \varphi-,\psi' \cdot_{NTCF} \psi-)(\!|\mathit{NTMap}|\!)) = (\mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{B})(\!|\mathit{Obj}|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *category-axioms assms H-vcomp* **have** *dom-vcomp-H*:
$\mathcal{D}_\circ\ ((\mathit{Hom}_{A.C\alpha}(\varphi-,\psi'-) \cdot_{NTCF} \mathit{Hom}_{A.C\alpha}(\varphi'-,\psi-))(\!|\mathit{NTMap}|\!)) =$
$(\mathit{op\text{-}cat}\ \mathfrak{A} \times_C \mathfrak{B})(\!|\mathit{Obj}|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

```
  show Hom_{A.C α}(φ' ∙_{NTCF} φ−,ψ' ∙_{NTCF} ψ−)(|NTMap|) =
    (Hom_{A.C α}(φ−,ψ'−) ∙_{NTCF} Hom_{A.C α}(φ'−,ψ−))(|NTMap|)
  proof(rule vsv-eqI, unfold dom-H-vcomp dom-vcomp-H)
    fix ab assume prems: ab ∈_∘ (op-cat 𝔄 ×_C 𝔅)(|Obj|)
    then obtain a b
      where ab-def: ab = [a, b]_∘
        and a: a ∈_∘ 𝔄(|Obj|)
        and b: b ∈_∘ 𝔅(|Obj|)
      by
        (
          auto
            elim:
              cat-prod-2-ObjE[
                OF φ'.NTDom.HomDom.category-op ψ'.NTDom.HomDom.category-axioms
                ]
            simp: cat-op-simps
        )
    from
      assms a b
      category-axioms
      φ'.NTDom.HomDom.category-axioms
      ψ'.NTDom.HomDom.category-axioms
    show
      Hom_{A.C α}(φ' ∙_{NTCF} φ−,ψ' ∙_{NTCF} ψ−)(|NTMap|)(|ab|) =
        (Hom_{A.C α}(φ−,ψ'−) ∙_{NTCF} Hom_{A.C α}(φ'−,ψ−))(|NTMap|)(|ab|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps ab-def
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
  qed (auto simp: ntcf-Hom-NTMap-vsv cat-cs-intros)

qed simp-all

lemmas [cat-cs-simps] = category.cat-ntcf-Hom-vcomp

lemma (in category) cat-ntcf-Hom-ntcf-id:
  assumes 𝔉 : 𝔄 ↦↦_{Cα} ℭ and 𝔉': 𝔅 ↦↦_{Cα} ℭ
  shows Hom_{A.C α}(ntcf-id 𝔉−,ntcf-id 𝔉'−) = ntcf-id Hom_{O.C α}ℭ(𝔉−,𝔉'−)
proof(rule ntcf-eqI[of α])

  interpret 𝔉: is-functor α 𝔄 ℭ 𝔉 by (rule assms(1))
  interpret 𝔉': is-functor α 𝔅 ℭ 𝔉' by (rule assms(2))

  from category-axioms assms show H-id:
    Hom_{A.C α}(ntcf-id 𝔉−,ntcf-id 𝔉'−) :
      Hom_{O.C α}ℭ(𝔉−,𝔉'−) ↦_{CF} Hom_{O.C α}ℭ(𝔉−,𝔉'−) :
      op-cat 𝔄 ×_C 𝔅 ↦↦_{Cα} cat-Set α
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from category-axioms assms show id-H:
    ntcf-id Hom_{O.C α}ℭ(𝔉−,𝔉'−) :
      Hom_{O.C α}ℭ(𝔉−,𝔉'−) ↦_{CF} Hom_{O.C α}ℭ(𝔉−,𝔉'−) :
      op-cat 𝔄 ×_C 𝔅 ↦↦_{Cα} cat-Set α
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  from category-axioms assms H-id have dom-H-id:
    𝒟_∘ (Hom_{A.C α}(ntcf-id 𝔉−,ntcf-id 𝔉'−)(|NTMap|)) = (op-cat 𝔄 ×_C 𝔅)(|Obj|)
```

```
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from category-axioms assms H-id have dom-id-H:
    𝒟∘ (ntcf-id Hom_{O.C α}𝔈(𝔉−,𝔉′−)(|NTMap|)) = (op-cat 𝔄 ×_C 𝔅)(|Obj|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  show
    Hom_{A.C α}(ntcf-id 𝔉−,ntcf-id 𝔉′−)(|NTMap|) =
      ntcf-id Hom_{O.C α}𝔈(𝔉−,𝔉′−)(|NTMap|)
  proof(rule vsv-eqI, unfold dom-H-id dom-id-H)
    show vsv (Hom_{A.C α}(ntcf-id 𝔉−,ntcf-id 𝔉′−)(|NTMap|))
      by (rule ntcf-Hom-NTMap-vsv)
    from id-H show vsv (ntcf-id Hom_{O.C α}𝔈(𝔉−,𝔉′−)(|NTMap|))
      by (intro is-functor.ntcf-id-NTMap-vsv)
        (cs-concl cs-shallow cs-intro: cat-cs-intros)
    fix ab assume ab ∈∘ (op-cat 𝔄 ×_C 𝔅)(|Obj|)
    then obtain a b
      where ab-def: ab = [a, b]∘
        and a: a ∈∘ 𝔄(|Obj|)
        and b: b ∈∘ 𝔅(|Obj|)
      by
        (
          auto
            elim:
              cat-prod-2-ObjE[OF 𝔉.HomDom.category-op 𝔉′.HomDom.category-axioms]
            simp: cat-op-simps
        )
    from category-axioms assms a b H-id id-H show
      Hom_{A.C α}(ntcf-id 𝔉−,ntcf-id 𝔉′−)(|NTMap|)(|ab|) =
      ntcf-id Hom_{O.C α}𝔈(𝔉−,𝔉′−)(|NTMap|)(|ab|)
      unfolding ab-def
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
  qed simp

qed simp-all

lemmas [cat-cs-simps] = category.cat-ntcf-Hom-ntcf-id
```

29.9 Composition of a *Hom*-natural transformation with a natural transformation

29.9.1 Definition and elementary properties

See subsection 1.15 in [3].

```
definition ntcf-lcomp-Hom :: V ⇒ V ⇒ V (‹Hom_{A.C}1′(/−−,−/′)›)
  where Hom_{A.C α}(φ−,−) = Hom_{A.C α}(φ−,ntcf-id (cf-id (φ(|NTDGCod|)))−)

definition ntcf-rcomp-Hom :: V ⇒ V ⇒ V (‹Hom_{A.C}1′(/−,−−/′)›)
  where Hom_{A.C α}(−,ψ−) = Hom_{A.C α}(ntcf-id (cf-id (ψ(|NTDGCod|)))−,ψ−)
```

29.9.2 Natural transformation map

```
lemma ntcf-lcomp-Hom-NTMap-vsv: vsv (Hom_{A.C α}(φ−,−)(|NTMap|))
  unfolding ntcf-lcomp-Hom-def by (rule ntcf-Hom-NTMap-vsv)
```

lemma *ntcf-rcomp-Hom-NTMap-vsv*: *vsv* (*Hom*$_{A.C\alpha}$(−,ψ−)(|*NTMap*|))
  unfolding *ntcf-rcomp-Hom-def* **by** (*rule ntcf-Hom-NTMap-vsv*)

lemma *ntcf-lcomp-Hom-NTMap-vdomain*[*cat-cs-simps*]:
  assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows $\mathcal{D}_\circ$ (*Hom*$_{A.C\alpha}$(φ−,−)(|*NTMap*|)) = (*op-cat* $\mathfrak{A}$ ×$_C$ $\mathfrak{C}$)(|*Obj*|)
proof–
  interpret φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule assms*(*1*))
  from *assms* **show** *?thesis*
    unfolding *ntcf-lcomp-Hom-def* φ.*ntcf-NTDGCod*
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-rcomp-Hom-NTMap-vdomain*[*cat-cs-simps*]:
  assumes $\psi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows $\mathcal{D}_\circ$ (*Hom*$_{A.C\alpha}$(−,ψ−)(|*NTMap*|)) = (*op-cat* $\mathfrak{C}$ ×$_C$ $\mathfrak{B}$)(|*Obj*|)
proof–
  interpret ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ ψ **by** (*rule assms*(*1*))
  from *assms* **show** *?thesis*
    unfolding *ntcf-rcomp-Hom-def* ψ.*ntcf-NTDGCod*
    by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemma *ntcf-lcomp-Hom-NTMap-app*[*cat-cs-simps*]:
  assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $a \in_\circ$ *op-cat* $\mathfrak{A}$(|*Obj*|)
    and $b \in_\circ \mathfrak{C}$(|*Obj*|)
  shows *Hom*$_{A.C\alpha}$(φ−,−)(|*NTMap*|)(|a, b|)$_\bullet$ = *ntcf-lcomp-Hom-component* φ a b
proof–
  interpret φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule assms*(*1*))
  from *assms* **show** *?thesis*
    unfolding
      ntcf-lcomp-Hom-def ntcf-lcomp-Hom-component-def φ.*ntcf-NTDGCod*
      cat-op-simps
    by
      (
        cs-concl
          cs-simp: *cat-cs-simps cat-op-simps*
          cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed

lemma *ntcf-rcomp-Hom-NTMap-app*[*cat-cs-simps*]:
  assumes $\psi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $a \in_\circ$ *op-cat* $\mathfrak{C}$(|*Obj*|)
    and $b \in_\circ \mathfrak{B}$(|*Obj*|)
  shows *Hom*$_{A.C\alpha}$(−,ψ−)(|*NTMap*|)(|a, b|)$_\bullet$ = *ntcf-rcomp-Hom-component* ψ a b
proof–
  interpret ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ ψ **by** (*rule assms*(*1*))
  from *assms* **show** *?thesis*
    unfolding
      ntcf-rcomp-Hom-def ntcf-rcomp-Hom-component-def ψ.*ntcf-NTDGCod*
      cat-op-simps
    by
      (
        cs-concl
          cs-simp: *cat-cs-simps cat-op-simps*

cs-intro: *cat-cs-intros cat-prod-cs-intros*
)
qed

lemma (**in** *category*) *ntcf-lcomp-Hom-NTMap-vrange*:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}\ (Hom_{A.C\alpha}(\varphi -,-)(\lVert NTMap \rVert)) \subseteq_{\circ} cat\text{-}Set\ \alpha(\lVert Arr \rVert)$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
from *assms* **show** *?thesis*
unfolding *ntcf-lcomp-Hom-def ntcf-lcomp-Hom-component-def* φ*.ntcf-NTDGCod*
by (*intro ntcf-Hom-NTMap-vrange*) (*cs-concl* **cs-intro**: *cat-cs-intros*)+
qed

lemma (**in** *category*) *ntcf-rcomp-Hom-NTMap-vrange*:
assumes $\psi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}\ (Hom_{A.C\alpha}(-,\psi -)(\lVert NTMap \rVert)) \subseteq_{\circ} cat\text{-}Set\ \alpha(\lVert Arr \rVert)$
proof–
interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \psi$ **by** (*rule assms*(*1*))
from *assms* **show** *?thesis*
unfolding *ntcf-rcomp-Hom-def ntcf-rcomp-Hom-component-def* ψ*.ntcf-NTDGCod*
by (*intro ntcf-Hom-NTMap-vrange*) (*cs-concl* **cs-intro**: *cat-cs-intros*)+
qed

29.9.3 Composition of a *Hom*-natural transformation with a natural transformation is a natural transformation

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-is-ntcf*:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{A.C\alpha}(\varphi -,-) :$
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G} -,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F} -,-) : op\text{-}cat\ \mathfrak{A} \times_{C} \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{A}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
from *assms category-axioms* **show** *?thesis*
unfolding
ntcf-lcomp-Hom-def cf-bcomp-Hom-cf-lcomp-Hom[*symmetric*] φ*.ntcf-NTDGCod*
by (*intro category.cat-ntcf-Hom-is-ntcf*)
(*cs-concl* **cs-intro**: *cat-cs-intros*)+
qed

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-is-ntcf'*:
assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\beta = \alpha$
and $\mathfrak{A}' = Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G} -,-)$
and $\mathfrak{B}' = Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F} -,-)$
and $\mathfrak{C}' = op\text{-}cat\ \mathfrak{A} \times_{C} \mathfrak{C}$
and $\mathfrak{D}' = cat\text{-}Set\ \alpha$
shows $Hom_{A.C\alpha}(\varphi -,-) : \mathfrak{A}' \mapsto_{CF} \mathfrak{B}' : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}'$
using *assms*(*1*) **unfolding** *assms*(*2–6*) **by** (*rule cat-ntcf-lcomp-Hom-is-ntcf*)

lemmas [*cat-cs-intros*] = *category.cat-ntcf-lcomp-Hom-is-ntcf'*

lemma (**in** *category*) *cat-ntcf-rcomp-Hom-is-ntcf*:
assumes $\psi : \mathfrak{F} \mapsto\mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{A.C\alpha}(-,\psi -) :$
$Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F} -) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{G} -) : op\text{-}cat\ \mathfrak{C} \times_{C} \mathfrak{B} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
proof–
interpret ψ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \psi$ **by** (*rule assms*(*1*))

```
  from assms category-axioms show ?thesis
    unfolding
      ntcf-rcomp-Hom-def cf-bcomp-Hom-cf-rcomp-Hom[symmetric] ψ.ntcf-NTDGCod
    by (intro category.cat-ntcf-Hom-is-ntcf)
      (cs-concl cs-intro: cat-cs-intros)+
qed

lemma (in category) cat-ntcf-rcomp-Hom-is-ntcf':
  assumes ψ : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα ℭ
    and β = α
    and 𝔄' = HomO.Cα ℭ(−,𝔉−)
    and 𝔅' = HomO.Cα ℭ(−,𝔊−)
    and ℭ' = op-cat ℭ ×C 𝔅
    and 𝔇' = cat-Set α
  shows HomA.Cα(−,ψ−) : 𝔄' ↦CF 𝔅' : ℭ' ↦↦Cα 𝔇'
  using assms(1) unfolding assms(2–6) by (rule cat-ntcf-rcomp-Hom-is-ntcf)

lemmas [cat-cs-intros] = category.cat-ntcf-rcomp-Hom-is-ntcf'
```

29.9.4 Component of a composition of a *Hom*-natural transformation with a natural transformation and the Yoneda component

```
lemma (in category) cat-ntcf-lcomp-Hom-component-is-Yoneda-component:
  assumes φ : 𝔉 ↦CF 𝔊 : 𝔅 ↦↦Cα ℭ
    and b ∈∘ op-cat 𝔅(|Obj|)
    and c ∈∘ ℭ(|Obj|)
  shows
    ntcf-lcomp-Hom-component φ b c =
      Yoneda-component HomO.Cα ℭ(𝔉(|ObjMap|)(|b|),−) (𝔊(|ObjMap|)(|b|)) (φ(|NTMap|)(|b|)) c
  (is ‹?lcomp = ?Yc›)
proof–

  interpret φ: is-ntcf α 𝔅 ℭ 𝔉 𝔊 φ by (rule assms(1))

  from assms(2) have b: b ∈∘ 𝔅(|Obj|) unfolding cat-op-simps by clarsimp
  from b have 𝔉b: 𝔉(|ObjMap|)(|b|) ∈∘ ℭ(|Obj|) and 𝔊b: 𝔊(|ObjMap|)(|b|) ∈∘ ℭ(|Obj|)
    by (auto intro: cat-cs-intros)
  from assms(1,3) b category-axioms have φb:
    φ(|NTMap|)(|b|) ∈∘ HomO.Cα ℭ(𝔉(|ObjMap|)(|b|),−)(|ObjMap|)(𝔊(|ObjMap|)(|b|))
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
      )

  have lcomp:
    ?lcomp : Hom ℭ (𝔊(|ObjMap|)(|b|)) c ↦cat-Set α Hom ℭ (𝔉(|ObjMap|)(|b|)) c
    by (rule cat-ntcf-lcomp-Hom-component-is-arr[OF assms])
  then have dom-lhs: 𝒟∘ (?lcomp(|ArrVal|)) = Hom ℭ (𝔊(|ObjMap|)(|b|)) c
    by (cs-concl cs-simp: cat-cs-simps)

  have Yc: ?Yc :
    Hom ℭ (𝔊(|ObjMap|)(|b|)) c ↦cat-Set α HomO.Cα ℭ(𝔉(|ObjMap|)(|b|),−)(|ObjMap|)(|c|)
    by
      (
        rule cat-Yoneda-component-is-arr[
          OF cat-cf-Hom-snd-is-functor[OF 𝔉b] 𝔊b φb assms(3)
          ]
```

```
    )
  then have dom-rhs: 𝒟ₒ (?Yc(|ArrVal|)) = Hom 𝔈 (𝔊(|ObjMap|)(|b|)) c
    by (cs-concl cs-simp: cat-cs-simps)

  show ?thesis
  proof(rule arr-Set-eqI[of α])

    from lcomp show arr-Set α ?lcomp by (auto dest: cat-Set-is-arrD(1))
    from Yc show arr-Set α ?Yc by (auto dest: cat-Set-is-arrD(1))

    show ?lcomp(|ArrVal|) = ?Yc(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      from assms(1) b category-axioms show vsv (?Yc(|ArrVal|))
        by (intro is-functor.Yoneda-component-ArrVal-vsv)
          (cs-concl cs-shallow cs-intro: cat-cs-intros)
      show ?lcomp(|ArrVal|)(|f|) = ?Yc(|ArrVal|)(|f|)
        if f ∈ₒ Hom 𝔈 (𝔊(|ObjMap|)(|b|)) c for f
      proof–
        from that have f : 𝔊(|ObjMap|)(|b|) ↦_𝔈 c by simp
        with category-axioms assms(1,3) b show ?thesis
          by
            (
              cs-concl cs-shallow
                cs-simp: cat-cs-simps cat-op-simps
                cs-intro: cat-cs-intros cat-op-intros
            )
      qed
    qed (simp-all add: ntcf-lcomp-Hom-component-ArrVal-vsv)

    from Yc category-axioms assms(1,3) b have
      ?Yc : Hom 𝔈 (𝔊(|ObjMap|)(|b|)) c ↦_cat-Set α Hom 𝔈 (𝔉(|ObjMap|)(|b|)) c
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
          )
    with lcomp show ?lcomp(|ArrCod|) = ?Yc(|ArrCod|)
      by (cs-concl cs-simp: cat-cs-simps)

  qed (use lcomp Yc in ‹cs-concl cs-simp: cat-cs-simps›)

qed
```

29.9.5 Composition of a *Hom*-natural transformation with a vertical composition of natural transformations

```
lemma (in category) cat-ntcf-lcomp-Hom-vcomp:
  assumes φ' : 𝔊 ↦_CF 𝔎 : 𝔄 ↦↦_Cα 𝔈 and φ : 𝔉 ↦_CF 𝔊 : 𝔄 ↦↦_Cα 𝔈
  shows Hom_A.Cα(φ' ∘_NTCF φ−,−) = Hom_A.Cα(φ−,−) ∘_NTCF Hom_A.Cα(φ'−,−)
proof–
  interpret φ': is-ntcf α 𝔄 𝔈 𝔊 𝔎 φ' by (rule assms(1))
  interpret φ: is-ntcf α 𝔄 𝔈 𝔉 𝔊 φ by (rule assms(2))
  from category-axioms have ntcf-id-cf-id:
    ntcf-id (cf-id 𝔈) = ntcf-id (cf-id 𝔈) ∘_NTCF ntcf-id (cf-id 𝔈)
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from category-axioms assms show ?thesis
    unfolding
      ntcf-lcomp-Hom-def
```

ntsmcf-vcomp-components
dghm-id-components
φ'*.ntcf-NTDGCod*
φ*.ntcf-NTDGCod*
by (*subst ntcf-id-cf-id*)
(*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-lcomp-Hom-vcomp*

lemma (**in** *category*) *cat-ntcf-rcomp-Hom-vcomp*:
assumes $\varphi' : \mathfrak{G} \mapsto_{CF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{A.C\alpha}(-,\varphi' \bullet_{NTCF} \varphi-) = Hom_{A.C\alpha}(-,\varphi'-) \bullet_{NTCF} Hom_{A.C\alpha}(-,\varphi-)$
proof–
interpret φ': *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{G}$ $\mathfrak{H}$ φ' **by** (*rule assms*(*1*))
interpret φ: *is-ntcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule assms*(*2*))
from *category-axioms* **have** *ntcf-id-cf-id*:
ntcf-id (*cf-id* $\mathfrak{C}$) = *ntcf-id* (*cf-id* $\mathfrak{C}$) $\bullet_{NTCF}$ *ntcf-id* (*cf-id* $\mathfrak{C}$)
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *category-axioms assms* **show** *?thesis*
unfolding
ntcf-rcomp-Hom-def
ntsmcf-vcomp-components
dghm-id-components
φ'*.ntcf-NTDGCod*
φ*.ntcf-NTDGCod*
by (*subst ntcf-id-cf-id*)
(*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-rcomp-Hom-vcomp*

29.9.6 Composition of a *Hom*-natural transformation with an identity natural transformation

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-ntcf-id*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{A.C\alpha}(ntcf\text{-}id\ \mathfrak{F}-,-) = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{A}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
from *category-axioms assms* **show** *?thesis*
unfolding *ntcf-lcomp-Hom-def ntcf-id-components* $\mathfrak{F}$*.cf-HomCod*
by
(
cs-concl
cs-simp: *ntcf-lcomp-Hom-def cat-cs-simps*
cs-intro: *cat-cs-intros*
)
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-lcomp-Hom-ntcf-id*

lemma (**in** *category*) *cat-ntcf-rcomp-Hom-ntcf-id*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $Hom_{A.C\alpha}(-,ntcf\text{-}id\ \mathfrak{F}-) = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(-,\mathfrak{F}-)$
proof–
interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
from *category-axioms assms* **show** *?thesis*

unfolding *ntcf-rcomp-Hom-def ntcf-id-components* $\mathfrak{F}$*.cf-HomCod*
by (*cs-concl* **cs-simp**: *ntcf-rcomp-Hom-def cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-rcomp-Hom-ntcf-id*

29.10 Projections of a *Hom*-natural transformation

The concept of a projection of a *Hom*-natural transformation appears in the corollary to the Yoneda Lemma in Chapter III-2 in [7] (although the concept has not been given any specific name in the aforementioned reference).

29.10.1 Definition and elementary properties

definition *ntcf-Hom-snd* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (‹$Hom_{A.C}$1-′(/-,-/′)›)
where $Hom_{A.C\alpha}\mathfrak{C}(f,-)$ =
Yoneda-arrow α ($Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{C}(\lfloor Dom \rfloor)(\lfloor f \rfloor),-)$) ($\mathfrak{C}(\lfloor Cod \rfloor)(\lfloor f \rfloor)$) f

definition *ntcf-Hom-fst* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (‹$Hom_{A.C}$1-′(/-,-/′)›)
where $Hom_{A.C\alpha}\mathfrak{C}(-,f)$ = $Hom_{A.C\alpha}$*op-cat* $\mathfrak{C}(f,-)$

Components.

lemma (**in** *category*) *cat-ntcf-Hom-snd-components*:
assumes $f : s \mapsto_{\mathfrak{C}} r$
shows $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTMap \rfloor)$ =
($\lambda d \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$. *Yoneda-component* $Hom_{O.C\alpha}\mathfrak{C}(s,-)$ r f d)
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTDom \rfloor)$ = $Hom_{O.C\alpha}\mathfrak{C}(r,-)$
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTCod \rfloor)$ = $Hom_{O.C\alpha}\mathfrak{C}(s,-)$
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTDGDom \rfloor)$ = $\mathfrak{C}$
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTDGCod \rfloor)$ = *cat-Set* α
proof–
interpret *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› ‹$Hom_{O.C\alpha}\mathfrak{C}(s,-)$›
using *assms category-axioms* **by** (*cs-concl* **cs-intro**: *cat-cs-intros*)
show $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTMap \rfloor)$ =
($\lambda d \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$. *Yoneda-component* $Hom_{O.C\alpha}\mathfrak{C}(s,-)$ r f d)
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTDom \rfloor)$ = $Hom_{O.C\alpha}\mathfrak{C}(r,-)$
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTCod \rfloor)$ = $Hom_{O.C\alpha}\mathfrak{C}(s,-)$
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTDGDom \rfloor)$ = $\mathfrak{C}$
and $Hom_{A.C\alpha}\mathfrak{C}(f,-)(\lfloor NTDGCod \rfloor)$ = *cat-Set* α
unfolding *ntcf-Hom-snd-def cat-is-arrD*[*OF assms*] *Yoneda-arrow-components*
by *simp-all*
qed

lemma (**in** *category*) *cat-ntcf-Hom-fst-components*:
assumes $f : r \mapsto_{\mathfrak{C}} s$
shows $Hom_{A.C\alpha}\mathfrak{C}(-,f)(\lfloor NTMap \rfloor)$ =
($\lambda d \in_{\circ} op\text{-}cat\ \mathfrak{C}(\lfloor Obj \rfloor)$. *Yoneda-component* $Hom_{O.C\alpha}\mathfrak{C}(-,s)$ r f d)
and $Hom_{A.C\alpha}\mathfrak{C}(-,f)(\lfloor NTDom \rfloor)$ = $Hom_{O.C\alpha}\mathfrak{C}(-,r)$
and $Hom_{A.C\alpha}\mathfrak{C}(-,f)(\lfloor NTCod \rfloor)$ = $Hom_{O.C\alpha}\mathfrak{C}(-,s)$
and $Hom_{A.C\alpha}\mathfrak{C}(-,f)(\lfloor NTDGDom \rfloor)$ = *op-cat* $\mathfrak{C}$
and $Hom_{A.C\alpha}\mathfrak{C}(-,f)(\lfloor NTDGCod \rfloor)$ = *cat-Set* α
using *category-axioms assms*
unfolding
ntcf-Hom-fst-def
category.cat-ntcf-Hom-snd-components[
OF category-op, *unfolded cat-op-simps*, *OF assms*
]

cat-op-simps
by (*cs-concl* **cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*)+

Alternative definition.

lemma (**in** *category*) *ntcf-Hom-snd-def'*:
assumes $f : r \mapsto_{\mathfrak{C}} s$
shows $Hom_{A.C\alpha}\mathfrak{C}(f,-) = Yoneda\text{-}arrow\ \alpha\ (Hom_{O.C\alpha}\mathfrak{C}(r,-))\ s\ f$
using *assms* **unfolding** *ntcf-Hom-snd-def* **by** (*simp add*: *cat-cs-simps*)

lemma (**in** *category*) *ntcf-Hom-fst-def'*:
assumes $f : r \mapsto_{\mathfrak{C}} s$
shows $Hom_{A.C\alpha}\mathfrak{C}(-,f) = Yoneda\text{-}arrow\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(-,s)\ r\ f$
proof–
from *assms category-axioms* **show** *?thesis*
unfolding *ntcf-Hom-fst-def ntcf-Hom-snd-def cat-op-simps*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)
qed

29.10.2 Natural transformation map

context *category*
begin

context
fixes *s r f*
assumes *f*: $f : s \mapsto_{\mathfrak{C}} r$
begin

mk-VLambda *cat-ntcf-Hom-snd-components*(*1*)[*OF f*]
|*vsv ntcf-Hom-snd-NTMap-vsv*[*intro*]|
|*vdomain ntcf-Hom-snd-NTMap-vdomain*|
|*app ntcf-Hom-snd-NTMap-app*|

end

context
fixes *s r f*
assumes *f*: $f : r \mapsto_{\mathfrak{C}} s$
begin

mk-VLambda *cat-ntcf-Hom-fst-components*(*1*)[*OF f*]
|*vsv ntcf-Hom-fst-NTMap-vsv*[*intro*]|
|*vdomain ntcf-Hom-fst-NTMap-vdomain*|
|*app ntcf-Hom-fst-NTMap-app*|

end

end

lemmas [*cat-cs-simps*] =
category.ntcf-Hom-snd-NTMap-vdomain
category.ntcf-Hom-fst-NTMap-vdomain

lemmas *ntcf-Hom-snd-NTMap-app*[*cat-cs-simps*] =

category.ntcf-Hom-snd-NTMap-app
category.ntcf-Hom-fst-NTMap-app

29.10.3 *Hom*-natural transformation projections are natural transformations

lemma (**in** *category*) *cat-ntcf-Hom-snd-is-ntcf*:
assumes $f : s \mapsto_{\mathfrak{C}} r$
shows $Hom_{A.C\alpha}\mathfrak{C}(f,-) :$
$Hom_{O.C\alpha}\mathfrak{C}(r,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(s,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
proof–
note f = *cat-is-arrD*[*OF assms*]
show *?thesis*
unfolding *ntcf-Hom-snd-def f*
proof(*rule category.cat-Yoneda-arrow-is-ntcf*)
from *assms category-axioms* **show** $f \in_{\circ} Hom_{O.C\alpha}\mathfrak{C}(s,-)(\!|ObjMap|\!)(\!|r|\!)$
by (*cs-concl* **cs-simp**: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*)
qed (*intro category-axioms cat-cf-Hom-snd-is-functor f*)+
qed

lemma (**in** *category*) *cat-ntcf-Hom-snd-is-ntcf'*:
assumes $f : s \mapsto_{\mathfrak{C}} r$
and $\beta = \alpha$
and $\mathfrak{A}' = Hom_{O.C\alpha}\mathfrak{C}(r,-)$
and $\mathfrak{B}' = Hom_{O.C\alpha}\mathfrak{C}(s,-)$
and $\mathfrak{C}' = \mathfrak{C}$
and $\mathfrak{D}' = cat\text{-}Set\ \alpha$
shows $Hom_{A.C\alpha}\mathfrak{C}(f,-) : \mathfrak{A}' \mapsto_{CF} \mathfrak{B}' : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}'$
using *assms*(*1*) **unfolding** *assms*(*2–6*) **by** (*rule cat-ntcf-Hom-snd-is-ntcf*)

lemmas [*cat-cs-intros*] = *category.cat-ntcf-Hom-snd-is-ntcf'*

lemma (**in** *category*) *cat-ntcf-Hom-fst-is-ntcf*:
assumes $f : r \mapsto_{\mathfrak{C}} s$
shows $Hom_{A.C\alpha}\mathfrak{C}(-,f) :$
$Hom_{O.C\alpha}\mathfrak{C}(-,r) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(-,s) : op\text{-}cat\ \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
proof–
from *assms* **have** *r*: $r \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** *s*: $s \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
from
category.cat-ntcf-Hom-snd-is-ntcf[
OF category-op,
unfolded cat-op-simps,
OF assms,
unfolded cat-op-cat-cf-Hom-snd[*OF r*] *cat-op-cat-cf-Hom-snd*[*OF s*],
folded ntcf-Hom-fst-def
]
show *?thesis* **.**
qed

lemma (**in** *category*) *cat-ntcf-Hom-fst-is-ntcf'*:
assumes $f : r \mapsto_{\mathfrak{C}} s$
and $\beta = \alpha$
and $\mathfrak{A}' = Hom_{O.C\alpha}\mathfrak{C}(-,r)$
and $\mathfrak{B}' = Hom_{O.C\alpha}\mathfrak{C}(-,s)$
and $\mathfrak{C}' = op\text{-}cat\ \mathfrak{C}$
and $\mathfrak{D}' = cat\text{-}Set\ \alpha$
shows $Hom_{A.C\alpha}\mathfrak{C}(-,f) : \mathfrak{A}' \mapsto_{CF} \mathfrak{B}' : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}'$
using *assms*(*1*) **unfolding** *assms*(*2–6*) **by** (*rule cat-ntcf-Hom-fst-is-ntcf*)

lemmas [*cat-cs-intros*] = *category.cat-ntcf-Hom-fst-is-ntcf'*

29.10.4 Opposite *Hom*-natural transformation projections

lemma (**in** *category*) *cat-op-cat-ntcf-Hom-snd*:
Hom$_{A.C\alpha}$*op-cat* $\mathfrak{C}(f,-)$ = *Hom*$_{A.C\alpha}\mathfrak{C}(-,f)$
unfolding *ntcf-Hom-fst-def* **by** *simp*

lemmas [*cat-op-simps*] = *category.cat-op-cat-ntcf-Hom-snd*

lemma (**in** *category*) *cat-op-cat-ntcf-Hom-fst*:
Hom$_{A.C\alpha}$*op-cat* $\mathfrak{C}(-,f)$ = *Hom*$_{A.C\alpha}\mathfrak{C}(f,-)$
unfolding *ntcf-Hom-fst-def cat-op-simps* **by** *simp*

lemmas [*cat-op-simps*] = *category.cat-op-cat-ntcf-Hom-fst*

29.10.5 *Hom*-natural transformation projections and the Yoneda component

lemma (**in** *category*) *cat-Yoneda-component-cf-Hom-snd-Comp*:
assumes $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **and** $d \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows
Yoneda-component Hom$_{O.C\alpha}\mathfrak{C}(a,-)$ *b f d* $\circ_{A\,cat\text{-}Set\ \alpha}$
Yoneda-component Hom$_{O.C\alpha}\mathfrak{C}(b,-)$ *c g d* =
Yoneda-component Hom$_{O.C\alpha}\mathfrak{C}(a,-)$ *c* (*g* $\circ_{A\,\mathfrak{C}}$ *f*) *d*
(**is** ‹*?Ya b f d* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?Yb c g d* = *?Ya c* (*g* $\circ_{A\,\mathfrak{C}}$ *f*) *d*›)
proof–

interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)

note *gD* = *cat-is-arrD*[*OF assms*(*1*)]
note *fD* = *cat-is-arrD*[*OF assms*(*2*)]

from *assms category-axioms* **have** *Y-f*:
?Ya b f d : *Hom* $\mathfrak{C}$ *b d* $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}$ *a d*
by (*cs-concl* **cs-simp**: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*)
moreover from *assms category-axioms* **have** *Y-g*:
?Yb c g d : *Hom* $\mathfrak{C}$ *c d* $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}$ *b d*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*
)
ultimately have *Yf-Yg*:
?Ya b f d $\circ_{A\,cat\text{-}Set\ \alpha}$ *?Yb c g d* : *Hom* $\mathfrak{C}$ *c d* $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}$ *a d*
by (*auto intro*: *cat-cs-intros*)
from *assms category-axioms* **have** *Y-gf*:
?Ya c (*g* $\circ_{A\,\mathfrak{C}}$ *f*) *d* : *Hom* $\mathfrak{C}$ *c d* $\mapsto_{cat\text{-}Set\ \alpha}$ *Hom* $\mathfrak{C}$ *a d*
by (*cs-concl* **cs-simp**: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*)
from *Yf-Yg* **have** *dom-rhs*:
$\mathcal{D}_{\circ}$ ((*?Ya b f d* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?Yb c g d*)(|*ArrVal*|)) = *Hom* $\mathfrak{C}$ *c d*
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *Y-gf* **have** *dom-lhs*: $\mathcal{D}_{\circ}$ (*?Ya c* (*g* $\circ_{A\,\mathfrak{C}}$ *f*) *d*(|*ArrVal*|)) = *Hom* $\mathfrak{C}$ *c d*
by (*cs-concl* **cs-simp**: *cat-cs-simps*)

show *?thesis*
proof(*rule arr-Set-eqI*[*of* α])
from *Yf-Yg* **show** *arr-Set-Yf-Yg*:
arr-Set α (*?Ya b f d* $\circ_{A\,cat\text{-}Set\ \alpha}$ *?Yb c g d*)

```
    by (auto dest: cat-Set-is-arrD(1))
  interpret Yf-Yg: arr-Set α ‹?Ya b f d ∘_A cat-Set α ?Yb c g d›
    by (rule arr-Set-Yf-Yg)
  from Y-gf show arr-Set-Y-gf: arr-Set α (?Ya c (g ∘_A 𝔈 f) d)
    by (auto dest: cat-Set-is-arrD(1))
  interpret Yf-Yg: arr-Set α ‹?Ya c (g ∘_A 𝔈 f) d› by (rule arr-Set-Y-gf)
  show
    (?Ya b f d ∘_A cat-Set α ?Yb c g d)(|ArrVal|) =
      ?Ya c (g ∘_A 𝔈 f) d(|ArrVal|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
    fix h assume h : c ↦_𝔈 d
    with Y-gf Y-g Y-f category-axioms assms show
      (?Ya b f d ∘_A cat-Set α ?Yb c g d)(|ArrVal|)(|h|) =
        ?Ya c (g ∘_A 𝔈 f) d(|ArrVal|)(|h|)

      by (cs-concl cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros)
  qed auto

qed (use Y-gf Yf-Yg in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed

lemmas [cat-cs-simps] =
  category.cat-Yoneda-component-cf-Hom-snd-Comp[symmetric]

lemma (in category) cat-Yoneda-component-cf-Hom-snd-CId:
  assumes c ∈_∘ 𝔈(|Obj|) and d ∈_∘ 𝔈(|Obj|)
  shows
    Yoneda-component Hom_O.Cα𝔈(c,−) c (𝔈(|CId|)(|c|)) d =
      cat-Set α(|CId|)(|Hom 𝔈 c d|)
  (is ‹?Ycd = cat-Set α(|CId|)(|Hom 𝔈 c d|)›)
proof–

  interpret Set: category α ‹cat-Set α› by (rule category-cat-Set)

  from assms category-axioms have Y-CId-c:
    ?Ycd : Hom 𝔈 c d ↦_cat-Set α Hom 𝔈 c d
    by (cs-concl cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros)
  from Y-CId-c Set.category-axioms assms category-axioms have CId-cd:
    cat-Set α(|CId|)(|Hom 𝔈 c d|) : Hom 𝔈 c d ↦_cat-Set α Hom 𝔈 c d
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from Y-CId-c have dom-lhs: 𝒟_∘ (?Ycd(|ArrVal|)) = Hom 𝔈 c d
    by (cs-concl cs-simp: cat-cs-simps)
  from CId-cd have dom-rhs: 𝒟_∘ (cat-Set α(|CId|)(|Hom 𝔈 c d|)(|ArrVal|)) = Hom 𝔈 c d
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)

  show ?thesis
  proof(rule arr-Set-eqI[of α])
    from Y-CId-c show arr-Set-Y-CId-c: arr-Set α ?Ycd
      by (auto dest: cat-Set-is-arrD(1))
    interpret Yf-Yg: arr-Set α ?Ycd by (rule arr-Set-Y-CId-c)
    from CId-cd show arr-Set-CId-cd: arr-Set α (cat-Set α(|CId|)(|Hom 𝔈 c d|))
      by (auto dest: cat-Set-is-arrD(1))
    interpret CId-cd: arr-Set α ‹cat-Set α(|CId|)(|Hom 𝔈 c d|)›
      by (rule arr-Set-CId-cd)
    show ?Ycd(|ArrVal|) = cat-Set α(|CId|)(|Hom 𝔈 c d|)(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)
      fix h assume h : c ↦_𝔈 d
```

with *CId-cd Y-CId-c category-axioms assms* **show**
?Ycd(|*ArrVal*|)(|*h*|) = *cat-Set* α(|*CId*|)(|*Hom* $\mathfrak{C}$ *c d*|)(|*ArrVal*|)(|*h*|)
by (*cs-concl* **cs-simp**: *cat-cs-simps cat-op-simps* **cs-intro**: *cat-cs-intros*)
qed *auto*
qed (*use Y-CId-c CId-cd* **in** ‹*cs-concl cs-shallow cs-simp*: *cat-cs-simps*›)+

qed

lemmas [*cat-cs-simps*] = *category.cat-Yoneda-component-cf-Hom-snd-CId*

29.10.6 *Hom*-natural transformation projection of a composition

lemma (**in** *category*) *cat-ntcf-Hom-snd-Comp*:
assumes $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
shows $Hom_{A.C\alpha}\mathfrak{C}(g \circ_{A\mathfrak{C}} f,-) = Hom_{A.C\alpha}\mathfrak{C}(f,-) \cdot_{NTCF} Hom_{A.C\alpha}\mathfrak{C}(g,-)$
(**is** ‹*?H-gf* = *?H-f* $\cdot_{NTCF}$ *?H-g*›)
proof(*rule ntcf-eqI*[*of* α])
from *assms category-axioms* **show**
$?H\text{-}gf : Hom_{O.C\alpha}\mathfrak{C}(c,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(a,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms category-axioms* **show** *?H-f* $\cdot_{NTCF}$ *?H-g* :
$Hom_{O.C\alpha}\mathfrak{C}(c,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(a,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms category-axioms* **have** *lhs-dom*: $\mathcal{D}_\circ$ (*?H-gf*(|*NTMap*|)) = $\mathfrak{C}$(|*Obj*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms category-axioms* **have** *rhs-dom*:
$\mathcal{D}_\circ$ ((*?H-f* $\cdot_{NTCF}$ *?H-g*)(|*NTMap*|)) = $\mathfrak{C}$(|*Obj*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show *?H-gf*(|*NTMap*|) = (*?H-f* $\cdot_{NTCF}$ *?H-g*)(|*NTMap*|)
proof(*rule vsv-eqI*, *unfold lhs-dom rhs-dom*)
fix *d* **assume** $d \in_\circ \mathfrak{C}$(|*Obj*|)
with *assms category-axioms* **show**
?H-gf(|*NTMap*|)(|*d*|) = (*?H-f* $\cdot_{NTCF}$ *?H-g*)(|*NTMap*|)(|*d*|)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*use assms* **in** ‹*auto intro*: *cat-cs-intros*›)
qed *auto*

lemmas [*cat-cs-simps*] = *category.cat-ntcf-Hom-snd-Comp*

lemma (**in** *category*) *cat-ntcf-Hom-fst-Comp*:
assumes $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
shows $Hom_{A.C\alpha}\mathfrak{C}(-,g \circ_{A\mathfrak{C}} f) = Hom_{A.C\alpha}\mathfrak{C}(-,g) \cdot_{NTCF} Hom_{A.C\alpha}\mathfrak{C}(-,f)$
proof–
note *category.cat-ntcf-Hom-snd-Comp*[
OF category-op, *unfolded cat-op-simps*, *OF assms*(*2*,*1*)
]
from *this category-axioms assms* **show** *?thesis*
by (*cs-prems* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*) *simp*
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-Hom-fst-Comp*

29.10.7 *Hom*-natural transformation projection of an identity

lemma (**in** *category*) *cat-ntcf-Hom-snd-CId*:
assumes $c \in_\circ \mathfrak{C}$(|*Obj*|)
shows $Hom_{A.C\alpha}\mathfrak{C}(\mathfrak{C}$(|*CId*|)(|*c*|),-) = *ntcf-id* $Hom_{O.C\alpha}\mathfrak{C}(c,-)$
(**is** ‹*?H-c* = *?id-H-c*›)

proof(*rule ntcf-eqI*[*of* α])
  from *assms* **have** $\mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor) : c \mapsto_{\mathfrak{C}} c$ **by** (*auto simp*: *cat-cs-intros*)
  from *assms category-axioms* **show**
    ?H-c : $Hom_{O.C\alpha}\mathfrak{C}(c,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(c,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *assms category-axioms* **show**
    ?id-H-c : $Hom_{O.C\alpha}\mathfrak{C}(c,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(c,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *assms category-axioms* **have** *lhs-dom*: $\mathcal{D}_{\circ}$ (*?H-c*($\lfloor NTMap \rfloor$)) = $\mathfrak{C}(\lfloor Obj \rfloor)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  from *assms category-axioms* **have** *rhs-dom*: $\mathcal{D}_{\circ}$ (*?id-H-c*($\lfloor NTMap \rfloor$)) = $\mathfrak{C}(\lfloor Obj \rfloor)$
    by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
  show *?H-c*($\lfloor NTMap \rfloor$) = *?id-H-c*($\lfloor NTMap \rfloor$)
  proof(*rule vsv-eqI*, *unfold lhs-dom rhs-dom*)
    from *assms category-axioms* **show** *vsv* (*?id-H-c*($\lfloor NTMap \rfloor$))
      by (*intro is-functor.ntcf-id-NTMap-vsv*)
        (*cs-concl* **cs-shallow cs-simp**: **cs-intro**: *cat-cs-intros*)
    fix *d* **assume** *d* $\in_{\circ}$ $\mathfrak{C}(\lfloor Obj \rfloor)$
    with *assms category-axioms* **show** *?H-c*($\lfloor NTMap \rfloor$)($\lfloor d \rfloor$) = *?id-H-c*($\lfloor NTMap \rfloor$)($\lfloor d \rfloor$)
      by
        (
          cs-concl **cs-shallow**
            cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)
  qed (*use assms* **in** ‹*auto intro*: *cat-cs-intros*›)
qed *auto*

lemmas [*cat-cs-simps*] = *category.cat-ntcf-Hom-snd-CId*

lemma (**in** *category*) *cat-ntcf-Hom-fst-CId*:
  assumes *c* $\in_{\circ}$ $\mathfrak{C}(\lfloor Obj \rfloor)$
  shows $Hom_{A.C\alpha}\mathfrak{C}(-,\mathfrak{C}(\lfloor CId \rfloor)(\lfloor c \rfloor)) = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(-,c)$
proof–
  note *category.cat-ntcf-Hom-snd-CId*[
      OF category-op, *unfolded cat-op-simps*, *OF assms*
      ]
  from *this category-axioms assms* **show** *?thesis*
    by (*cs-prems* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*) *simp*
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-Hom-fst-CId*

29.10.8 *Hom*-natural transformation and the Yoneda map

lemma (**in** *category*) *cat-Yoneda-map-of-ntcf-Hom-snd*:
  assumes *f* : *s* $\mapsto_{\mathfrak{C}}$ *r*
  shows *Yoneda-map* α ($Hom_{O.C\alpha}\mathfrak{C}(s,-)$) *r*($\lfloor Hom_{A.C\alpha}\mathfrak{C}(f,-) \rfloor$) = *f*
  using *category-axioms assms*
  by
    (
      cs-concl
        cs-simp: *cat-cs-simps cat-op-simps cat-Set-components*(*1*)
        cs-intro: *cat-cs-intros cat-prod-cs-intros*
)

lemmas [*cat-cs-simps*] = *category.cat-Yoneda-map-of-ntcf-Hom-snd*

lemma (**in** *category*) *cat-Yoneda-map-of-ntcf-Hom-fst*:

assumes $f : r \mapsto_{\mathfrak{C}} s$
shows *Yoneda-map* α $(Hom_{O.C\alpha}\mathfrak{C}(-,s))$ $r(\!|Hom_{A.C\alpha}\mathfrak{C}(-,f)|\!) = f$
proof–
note *category.cat-Yoneda-map-of-ntcf-Hom-snd*[
OF category-op, *unfolded cat-op-simps*, *OF assms*
]
from *this category-axioms assms* **show** *?thesis*
by (*cs-prems* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*) *simp*
qed

lemmas [*cat-cs-simps*] = *category.cat-Yoneda-map-of-ntcf-Hom-fst*

29.11 Evaluation arrow

29.11.1 Definition and elementary properties

The evaluation arrow is a part of the definition of the evaluation functor. The evaluation functor appears in Chapter III-2 in [7].

definition *cf-eval-arrow* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-eval-arrow* $\mathfrak{C}$ $\mathfrak{N}$ f =
[
(
$\lambda x \in_{\circ} \mathfrak{N}(\!|NTDom|\!)(\!|ObjMap|\!)(\!|\mathfrak{C}(\!|Dom|\!)(\!|f|\!)|\!).$
$\mathfrak{N}(\!|NTCod|\!)(\!|ArrMap|\!)(\!|f|\!)(\!|ArrVal|\!)(\!|\mathfrak{N}(\!|NTMap|\!)(\!|\mathfrak{C}(\!|Dom|\!)(\!|f|\!)|\!)(\!|ArrVal|\!)(\!|x|\!)|\!)$
),
$\mathfrak{N}(\!|NTDom|\!)(\!|ObjMap|\!)(\!|\mathfrak{C}(\!|Dom|\!)(\!|f|\!)|\!)$,
$\mathfrak{N}(\!|NTCod|\!)(\!|ObjMap|\!)(\!|\mathfrak{C}(\!|Cod|\!)(\!|f|\!)|\!)$
]$_{\circ}$

Components.

lemma *cf-eval-arrow-components*:
shows *cf-eval-arrow* $\mathfrak{C}$ $\mathfrak{N}$ $f(\!|ArrVal|\!)$ =
(
$\lambda x \in_{\circ} \mathfrak{N}(\!|NTDom|\!)(\!|ObjMap|\!)(\!|\mathfrak{C}(\!|Dom|\!)(\!|f|\!)|\!).$
$\mathfrak{N}(\!|NTCod|\!)(\!|ArrMap|\!)(\!|f|\!)(\!|ArrVal|\!)(\!|\mathfrak{N}(\!|NTMap|\!)(\!|\mathfrak{C}(\!|Dom|\!)(\!|f|\!)|\!)(\!|ArrVal|\!)(\!|x|\!)|\!)$
)
and *cf-eval-arrow* $\mathfrak{C}$ $\mathfrak{N}$ $f(\!|ArrDom|\!) = \mathfrak{N}(\!|NTDom|\!)(\!|ObjMap|\!)(\!|\mathfrak{C}(\!|Dom|\!)(\!|f|\!)|\!)$
and *cf-eval-arrow* $\mathfrak{C}$ $\mathfrak{N}$ $f(\!|ArrCod|\!) = \mathfrak{N}(\!|NTCod|\!)(\!|ObjMap|\!)(\!|\mathfrak{C}(\!|Cod|\!)(\!|f|\!)|\!)$
unfolding *cf-eval-arrow-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context
fixes α $\mathfrak{N}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ a b f
assumes $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
and f: $f : a \mapsto_{\mathfrak{C}} b$
begin

interpretation $\mathfrak{N}$: *is-ntcf* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule* $\mathfrak{N}$)

lemmas *cf-eval-arrow-components′* = *cf-eval-arrow-components*[
where $\mathfrak{C}$=$\mathfrak{C}$ **and** $\mathfrak{N}$=‹*ntcf-arrow* $\mathfrak{N}$› **and** f=f,
unfolded
ntcf-arrow-components
cf-map-components
$\mathfrak{N}$.*NTDom.HomDom.cat-is-arrD*[*OF f*]
cat-cs-simps
]

lemmas [*cat-cs-simps*] = *cf-eval-arrow-components'*(*2*,*3*)

end

29.11.2 Arrow value

context
  fixes α $\mathfrak{N}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ *a b f*
  assumes $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
    and *f*: $f : a \mapsto_{\mathfrak{C}} b$
begin

mk-VLambda *cf-eval-arrow-components'*(*1*)[*OF* $\mathfrak{N}$ *f*]
  |*vsv cf-eval-arrow-ArrVal-vsv*[*cat-cs-intros*]|
  |*vdomain cf-eval-arrow-ArrVal-vdomain*[*cat-cs-simps*]|
  |*app cf-eval-arrow-ArrVal-app*[*cat-cs-simps*]|

end

29.11.3 Evaluation arrow is an arrow in the category *Set*

lemma *cf-eval-arrow-is-arr*:
  assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α **and** $f : a \mapsto_{\mathfrak{C}} b$
  shows *cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* $\mathfrak{N}$) *f* :
    $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{cat\text{-}Set\ \alpha} \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
proof–
  interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*)
  show *?thesis*
  proof
    (
      intro cat-Set-is-arrI arr-SetI,
      unfold cf-eval-arrow-components'(*2*,*3*)[*OF assms*]
)
    show *vfsequence* (*cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* $\mathfrak{N}$) *f*)
      unfolding *cf-eval-arrow-def* **by** *simp*
    show *vcard* (*cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* $\mathfrak{N}$) *f*) = $3_{\mathbb{N}}$
      unfolding *cf-eval-arrow-def* **by** (*simp add*: *nat-omega-simps*)
    show $\mathcal{R}_{\circ}$ (*cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* $\mathfrak{N}$) *f*(|*ArrVal*|)) $\subseteq_{\circ}$ $\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
      by
      (
        unfold cf-eval-arrow-components'[*OF assms*],
        intro vrange-VLambda-vsubset
)
      (
        use assms **in**
          ‹*cs-concl cs-intro*: *cat-cs-intros cat-Set-cs-intros*›
)+
  qed
    (
      use assms(*2*) **in**
        ‹*cs-concl cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*›
)+
qed

lemma *cf-eval-arrow-is-arr'*[*cat-cs-intros*]:
  assumes $\mathfrak{N}' = ntcf\text{-}arrow\ \mathfrak{N}$
    and $\mathfrak{F}a = \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
    and $\mathfrak{G}b = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$

```
    and 𝔑 : 𝔉 ↦_CF 𝔊 : ℭ ↦↦_Cα cat-Set α
    and f : a ↦_ℭ b
  shows cf-eval-arrow ℭ 𝔑′ f : 𝔉a ↦_cat-Set α 𝔊b
  using assms(4,5) unfolding assms(1–3) by (rule cf-eval-arrow-is-arr)

lemma (in category) cat-cf-eval-arrow-ntcf-vcomp[cat-cs-simps]:
  assumes 𝔐 : 𝔊 ↦_CF ℌ : ℭ ↦↦_Cα cat-Set α
    and 𝔑 : 𝔉 ↦_CF 𝔊 : ℭ ↦↦_Cα cat-Set α
    and g : b ↦_ℭ c
    and f : a ↦_ℭ b
  shows
    cf-eval-arrow ℭ (ntcf-arrow (𝔐 ∙_NTCF 𝔑)) (g ∘_Aℭ f) =
      cf-eval-arrow ℭ (ntcf-arrow 𝔐) g ∘_A cat-Set α
      cf-eval-arrow ℭ (ntcf-arrow 𝔑) f
proof–

  interpret 𝔐: is-ntcf α ℭ ‹cat-Set α› 𝔊 ℌ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntcf α ℭ ‹cat-Set α› 𝔉 𝔊 𝔑 by (rule assms(2))

  have 𝔐𝔑: 𝔐 ∙_NTCF 𝔑 : 𝔉 ↦_CF ℌ : ℭ ↦↦_Cα cat-Set α
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms(3,4) have gf: g ∘_Aℭ f : a ↦_ℭ c
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from 𝔐𝔑 gf have cf-eval-gf:
    cf-eval-arrow ℭ (ntcf-arrow (𝔐 ∙_NTCF 𝔑)) (g ∘_Aℭ f) :
      𝔉(|ObjMap|)(|a|) ↦_cat-Set α ℌ(|ObjMap|)(|c|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from assms(3,4) have cf-eval-g-cf-eval-f:
    cf-eval-arrow ℭ (ntcf-arrow 𝔐) g ∘_A cat-Set α
      cf-eval-arrow ℭ (ntcf-arrow 𝔑) f :
      𝔉(|ObjMap|)(|a|) ↦_cat-Set α ℌ(|ObjMap|)(|c|)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  note cf-eval-gf = cf-eval-gf cat-Set-is-arrD[OF cf-eval-gf]
  note cf-eval-g-cf-eval-f =
    cf-eval-g-cf-eval-f cat-Set-is-arrD[OF cf-eval-g-cf-eval-f]

  interpret arr-Set-cf-eval-gf:
    arr-Set α ‹cf-eval-arrow ℭ (ntcf-arrow (𝔐 ∙_NTCF 𝔑)) (g ∘_Aℭ f)›
    by (rule cf-eval-gf(2))
  interpret arr-Set-cf-eval-g-cf-eval-f:
    arr-Set
      α
      ‹
        cf-eval-arrow ℭ (ntcf-arrow 𝔐) g ∘_A cat-Set α
        cf-eval-arrow ℭ (ntcf-arrow 𝔑) f
      ›
    by (rule cf-eval-g-cf-eval-f(2))

  show ?thesis
  proof(rule arr-Set-eqI)
    from 𝔐𝔑 gf have dom-lhs:
      𝒟∘ (cf-eval-arrow ℭ (ntcf-arrow (𝔐 ∙_NTCF 𝔑)) (g ∘_Aℭ f)(|ArrVal|)) =
        𝔉(|ObjMap|)(|a|)
      by (cs-concl cs-shallow cs-simp: cat-cs-simps)
    from cf-eval-g-cf-eval-f(1) have dom-rhs:
      𝒟∘
        (
          (
```

```
        cf-eval-arrow 𝔈 (ntcf-arrow 𝔐) g ∘A cat-Set α
        cf-eval-arrow 𝔈 (ntcf-arrow 𝔑) f
      )(|ArrVal|)
    ) = 𝔉(|ObjMap|)(|a|)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)
show
  cf-eval-arrow 𝔈 (ntcf-arrow (𝔐 ⊙NTCF 𝔑)) (g ∘A𝔈 f)(|ArrVal|) =
  (
    cf-eval-arrow 𝔈 (ntcf-arrow 𝔐) g ∘A cat-Set α
    cf-eval-arrow 𝔈 (ntcf-arrow 𝔑) f
  )(|ArrVal|)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
  fix 𝔉a assume prems: 𝔉a ∈∘ 𝔉(|ObjMap|)(|a|)
  from
    ArrVal-eq-helper
      [
        OF 𝔐.ntcf-Comp-commute[OF assms(4), symmetric],
        where a=‹𝔑(|NTMap|)(|a|)(|ArrVal|)(|𝔉a|)›
      ]
    prems
    assms(3,4)
  have [cat-cs-simps]:
    ℌ(|ArrMap|)(|f|)(|ArrVal|)(|𝔐(|NTMap|)(|a|)(|ArrVal|)(|𝔑(|NTMap|)(|a|)(|ArrVal|)(|𝔉a|)|)|) =
      𝔐(|NTMap|)(|b|)(|ArrVal|)(|𝔊(|ArrMap|)(|f|)(|ArrVal|)(|𝔑(|NTMap|)(|a|)(|ArrVal|)(|𝔉a|)|)|)
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cs-intro: cat-Set-cs-intros cat-cs-intros
      )
  from prems assms(3,4) show
    cf-eval-arrow 𝔈 (ntcf-arrow (𝔐 ⊙NTCF 𝔑)) (g ∘A𝔈 f)(|ArrVal|)(|𝔉a|) =
      (
        cf-eval-arrow 𝔈 (ntcf-arrow 𝔐) g ∘A cat-Set α
        cf-eval-arrow 𝔈 (ntcf-arrow 𝔑) f
      )(|ArrVal|)(|𝔉a|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cs-intro: cat-Set-cs-intros cat-cs-intros
      )
  qed (cs-concl cs-shallow cs-intro: V-cs-intros)
qed
  (
    auto
      simp: cf-eval-gf cf-eval-g-cf-eval-f
      intro: cf-eval-gf(2) cf-eval-g-cf-eval-f(2)
  )

qed

lemmas [cat-cs-simps] = category.cat-cf-eval-arrow-ntcf-vcomp

lemma (in category) cat-cf-eval-arrow-ntcf-id[cat-cs-simps]:
  assumes 𝔉 : 𝔈 ↦↦C α cat-Set α and c ∈∘ 𝔈(|Obj|)
  shows
    cf-eval-arrow 𝔈 (ntcf-arrow (ntcf-id 𝔉)) (𝔈(|CId|)(|c|)) =
      cat-Set α(|CId|)(|𝔉(|ObjMap|)(|c|)|)
proof–
```

interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{F}$ **by** (*rule assms*)

from *assms*(*2*) **have** *ntcf-id-CId-c*:
cf-eval-arrow $\mathfrak{C}$ (*ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)) ($\mathfrak{C}$⦇*CId*⦈⦇*c*⦈) :
$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈ $\mapsto_{cat\text{-}Set\ \alpha}$ $\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈
by (*cs-concl* **cs-intro**: *cat-cs-intros*)
from *assms*(*2*) **have** *CId-*$\mathfrak{F}$*c*:
cat-Set α⦇*CId*⦈⦇$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈⦈ : $\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈ $\mapsto_{cat\text{-}Set\ \alpha}$ $\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈
by (*cs-concl* **cs-intro**: *cat-cs-intros*)

show *?thesis*
proof(*rule arr-Set-eqI*[*of* α])

from *ntcf-id-CId-c* **show** *arr-Set-ntcf-id-CId-c*:
arr-Set α (*cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)) ($\mathfrak{C}$⦇*CId*⦈⦇*c*⦈))
by (*auto dest*: *cat-Set-is-arrD*(*1*))
from *ntcf-id-CId-c* **have** *dom-lhs*:
$\mathcal{D}_\circ$ (*cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)) ($\mathfrak{C}$⦇*CId*⦈⦇*c*⦈)⦇*ArrVal*⦈) =
$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈
by (*cs-concl* **cs-simp**: *cat-cs-simps*)+
interpret *ntcf-id-CId-c*:
arr-Set α ‹*cf-eval-arrow* $\mathfrak{C}$ (*ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)) ($\mathfrak{C}$⦇*CId*⦈⦇*c*⦈)›
by (*rule arr-Set-ntcf-id-CId-c*)

from *CId-*$\mathfrak{F}$*c* **show** *arr-Set-CId-*$\mathfrak{F}$*c*: *arr-Set* α (*cat-Set* α⦇*CId*⦈⦇$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈⦈)
by (*auto dest*: *cat-Set-is-arrD*(*1*))
from *CId-*$\mathfrak{F}$*c assms*(*2*) **have** *dom-rhs*:
$\mathcal{D}_\circ$ ((*cat-Set* α⦇*CId*⦈⦇$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈⦈)⦇*ArrVal*⦈) = $\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)

show
cf-eval-arrow $\mathfrak{C}$ (*ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)) ($\mathfrak{C}$⦇*CId*⦈⦇*c*⦈)⦇*ArrVal*⦈ =
cat-Set α⦇*CId*⦈⦇$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈⦈⦇*ArrVal*⦈
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** *a* $\in_\circ$ $\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈
with *category-axioms assms*(*2*) **show**
cf-eval-arrow $\mathfrak{C}$ (*ntcf-arrow* (*ntcf-id* $\mathfrak{F}$)) ($\mathfrak{C}$⦇*CId*⦈⦇*c*⦈)⦇*ArrVal*⦈⦇*a*⦈ =
cat-Set α⦇*CId*⦈⦇$\mathfrak{F}$⦇*ObjMap*⦈⦇*c*⦈⦈⦇*ArrVal*⦈⦇*a*⦈
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*use arr-Set-ntcf-id-CId-c arr-Set-CId-*$\mathfrak{F}$*c* **in** *auto*)

qed (*use ntcf-id-CId-c CId-*$\mathfrak{F}$*c* **in** ‹*cs-concl cs-simp*: *cat-cs-simps*›)+

qed

lemmas [*cat-cs-simps*] = *category.cat-cf-eval-arrow-ntcf-id*

29.12 *HOM*-functor

29.12.1 Definition and elementary properties

The following definition is a technical generalization that is used later in this section.

definition *cf-HOM-snd* :: $V \Rightarrow V \Rightarrow V$ (‹$HOM_{C1}'(/,--/')$›)
where $HOM_{C\alpha}(,\mathfrak{F}-)$ =
[
($\lambda a \in_\circ$ *op-cat* ($\mathfrak{F}$⦇*HomCod*⦈)⦇*Obj*⦈. *cf-map* ($Hom_{O.C\alpha}$($\mathfrak{F}$⦇*HomCod*⦈)(*a*,−) $\circ_{CF}$ $\mathfrak{F}$)),
(

```
    λf∈₀op-cat (𝔉(⦇HomCod⦈))(⦇Arr⦈).
      ntcf-arrow (Hom_{A.Cα}(𝔉(⦇HomCod⦈))(f,–) ∘_{NTCF-CF} 𝔉)
  ),
  op-cat (𝔉(⦇HomCod⦈)),
  cat-FUNCT α (𝔉(⦇HomDom⦈)) (cat-Set α)
]₀
```

definition *cf-HOM-fst* :: $V \Rightarrow V \Rightarrow V$ (‹$HOM_{C1}'(/-–,/')$›)
where $HOM_{C\alpha}(\mathfrak{F}-,) =$

```
[
  (λa∈₀(𝔉(⦇HomCod⦈))(⦇Obj⦈). cf-map (Hom_{O.Cα}(𝔉(⦇HomCod⦈))(–,a) ∘_{CF} op-cf 𝔉)),
  (
    λf∈₀(𝔉(⦇HomCod⦈))(⦇Arr⦈).
      ntcf-arrow (Hom_{A.Cα}(𝔉(⦇HomCod⦈))(–,f) ∘_{NTCF-CF} op-cf 𝔉)
  ),
  𝔉(⦇HomCod⦈),
  cat-FUNCT α (op-cat (𝔉(⦇HomDom⦈))) (cat-Set α)
]₀
```

Components.

lemma *cf-HOM-snd-components*:
shows $HOM_{C\alpha}(,\mathfrak{F}-)(\llbracket ObjMap \rrbracket) =$
(λa∈₀op-cat (𝔉(⦇HomCod⦈))(⦇Obj⦈). cf-map ($Hom_{O.C\alpha}$(𝔉(⦇HomCod⦈))(a,–) ∘$_{CF}$ 𝔉))
and $HOM_{C\alpha}(,\mathfrak{F}-)(\llbracket ArrMap \rrbracket) =$
(
λf∈₀op-cat (𝔉(⦇HomCod⦈))(⦇Arr⦈).
ntcf-arrow ($Hom_{A.C\alpha}$(𝔉(⦇HomCod⦈))(f,–) ∘$_{NTCF-CF}$ 𝔉)
)
and [*cat-cs-simps*]: $HOM_{C\alpha}(,\mathfrak{F}-)(\llbracket HomDom \rrbracket) =$ *op-cat* (𝔉(⦇HomCod⦈))
and [*cat-cs-simps*]:
$HOM_{C\alpha}(,\mathfrak{F}-)(\llbracket HomCod \rrbracket) =$ *cat-FUNCT* α (𝔉(⦇HomDom⦈)) (*cat-Set* α)
unfolding *cf-HOM-snd-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *cf-HOM-fst-components*:
shows $HOM_{C\alpha}(\mathfrak{F}-,)(\llbracket ObjMap \rrbracket) =$
(λa∈₀(𝔉(⦇HomCod⦈))(⦇Obj⦈). cf-map ($Hom_{O.C\alpha}$(𝔉(⦇HomCod⦈))(–,a) ∘$_{CF}$ op-cf 𝔉))
and $HOM_{C\alpha}(\mathfrak{F}-,)(\llbracket ArrMap \rrbracket) =$
(
λf∈₀(𝔉(⦇HomCod⦈))(⦇Arr⦈).
ntcf-arrow ($Hom_{A.C\alpha}$(𝔉(⦇HomCod⦈))(–,f) ∘$_{NTCF-CF}$ op-cf 𝔉)
)
and $HOM_{C\alpha}(\mathfrak{F}-,)(\llbracket HomDom \rrbracket) = \mathfrak{F}(\llbracket HomCod \rrbracket)$
and $HOM_{C\alpha}(\mathfrak{F}-,)(\llbracket HomCod \rrbracket) =$ *cat-FUNCT* α (*op-cat* (𝔉(⦇HomDom⦈))) (*cat-Set* α)
unfolding *cf-HOM-fst-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

context *is-functor*
begin

lemmas *cf-HOM-snd-components'* =
cf-HOM-snd-components[**where** 𝔉=𝔉, *unfolded cf-HomDom cf-HomCod*]

lemmas [*cat-cs-simps*] = *cf-HOM-snd-components'*(*3*,*4*)

lemmas *cf-HOM-fst-components'* =
cf-HOM-fst-components[**where** 𝔉=𝔉, *unfolded cf-HomDom cf-HomCod*]

lemmas [*cat-cs-simps*] = *cf-HOM-snd-components'*(*3*,*4*)

end

29.12.2 Object map

mk-VLambda *cf-HOM-snd-components*(*1*)
|*vsv cf-HOM-snd-ObjMap-vsv*[*cat-cs-intros*]|

mk-VLambda (**in** *is-functor*) *cf-HOM-snd-components′*(*1*)[*unfolded cat-op-simps*]
|*vdomain cf-HOM-snd-ObjMap-vdomain*[*cat-cs-simps*]|
|*app cf-HOM-snd-ObjMap-app*[*cat-cs-simps*]|

mk-VLambda *cf-HOM-snd-components*(*1*)
|*vsv cf-HOM-fst-ObjMap-vsv*[*cat-cs-intros*]|

mk-VLambda (**in** *is-functor*) *cf-HOM-fst-components′*(*1*)[*unfolded cat-op-simps*]
|*vdomain cf-HOM-fst-ObjMap-vdomain*[*cat-cs-simps*]|
|*app cf-HOM-fst-ObjMap-app*[*cat-cs-simps*]|

29.12.3 Arrow map

mk-VLambda *cf-HOM-snd-components*(*2*)
|*vsv cf-HOM-snd-ArrMap-vsv*[*cat-cs-intros*]|

mk-VLambda (**in** *is-functor*) *cf-HOM-snd-components′*(*2*)[*unfolded cat-op-simps*]
|*vdomain cf-HOM-snd-ArrMap-vdomain*[*cat-cs-simps*]|
|*app cf-HOM-snd-ArrMap-app*[*cat-cs-simps*]|

mk-VLambda *cf-HOM-fst-components*(*2*)
|*vsv cf-HOM-fst-ArrMap-vsv*[*cat-cs-intros*]|

mk-VLambda (**in** *is-functor*) *cf-HOM-fst-components′*(*2*)[*unfolded cat-op-simps*]
|*vdomain cf-HOM-fst-ArrMap-vdomain*[*cat-cs-simps*]|
|*app cf-HOM-fst-ArrMap-app*[*cat-cs-simps*]|

29.12.4 Opposite *HOM*-functor

lemma (**in** *is-functor*) *cf-HOM-snd-op*[*cat-op-simps*]:
$HOM_{C\alpha}(,op\text{-}cf\ \mathfrak{F}-) = HOM_{C\alpha}(\mathfrak{F}-,)$
proof–
have *dom-lhs*: $\mathcal{D}_\circ\ HOM_{C\alpha}(,op\text{-}cf\ \mathfrak{F}-) = \mathcal{4}_{\mathbb{N}}$
unfolding *cf-HOM-snd-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ\ HOM_{C\alpha}(\mathfrak{F}-,) = \mathcal{4}_{\mathbb{N}}$
unfolding *cf-HOM-fst-def* **by** (*simp add*: *nat-omega-simps*)
show *?thesis*
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** $a \in_\circ \mathcal{4}_{\mathbb{N}}$
then show $HOM_{C\alpha}(,op\text{-}cf\ \mathfrak{F}-)(\lfloor a \rfloor) = HOM_{C\alpha}(\mathfrak{F}-,)(\lfloor a \rfloor)$
proof
(
elim-in-numeral,
use nothing **in** ‹*fold dghm-field-simps*, *unfold cat-cs-simps*›
)
show $HOM_{C\alpha}(,op\text{-}cf\ \mathfrak{F}-)(\lfloor ObjMap \rfloor) = HOM_{C\alpha}(\mathfrak{F}-,)(\lfloor ObjMap \rfloor)$
unfolding
cf-HOM-fst-components′
is-functor.cf-HOM-snd-components′[*OF is-functor-op*]
by (*rule VLambda-eqI*, *unfold cat-op-simps*)
(
cs-concl **cs-shallow**

```
        cs-simp: cat-cs-simps cat-op-simps cs-intro: cat-cs-intros
      )+
    show HOM_{Cα}(,op-cf 𝔉−)(|ArrMap|) = HOM_{Cα}(𝔉−,)(|ArrMap|)
      unfolding
        cf-HOM-fst-components'
        is-functor.cf-HOM-snd-components'[OF is-functor-op]
      by (rule VLambda-eqI, unfold cat-op-simps)
        (cs-concl cs-shallow cs-simp: cat-op-simps cs-intro: cat-cs-intros)+
  qed
    (
      auto simp:
        cf-HOM-fst-components' cat-cs-simps cat-op-simps cat-op-intros
    )
 qed (auto simp: cf-HOM-snd-def cf-HOM-fst-def)
qed

lemmas [cat-op-simps] = is-functor.cf-HOM-snd-op

context is-functor
begin

lemmas cf-HOM-fst-op[cat-op-simps] =
  is-functor.cf-HOM-snd-op[OF is-functor-op, unfolded cat-op-simps, symmetric]

end

lemmas [cat-op-simps] = is-functor.cf-HOM-fst-op
```

29.12.5 *HOM*-functor is a functor

```
lemma (in is-functor) cf-HOM-snd-is-functor:
  assumes 𝒵 β and α ∈₀ β
  shows HOM_{Cα}(,𝔉−) : op-cat 𝔅 ↦↦_{Cβ} cat-FUNCT α 𝔄 (cat-Set α)
proof–

  interpret β: 𝒵 β by (rule assms(1))
  interpret βℭ: category β 𝔅
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+

  show ?thesis
  proof(intro is-functorI', unfold cat-op-simps)
    show vfsequence HOM_{Cα}(,𝔉−) unfolding cf-HOM-snd-def by auto
    show vcard HOM_{Cα}(,𝔉−) = ℶ_ℕ
      unfolding cf-HOM-snd-def by (simp add: nat-omega-simps)
    show ℛ₀ (HOM_{Cα}(,𝔉−)(|ObjMap|)) ⊆₀ cat-FUNCT α 𝔄 (cat-Set α)(|Obj|)
      unfolding cf-HOM-snd-components'
    proof(rule vrange-VLambda-vsubset, unfold cat-op-simps)
      fix b assume prems: b ∈₀ 𝔅(|Obj|)
      with assms(2) show
        cf-map (Hom_{O.Cα}𝔅(b,−) ∘_{CF} 𝔉) ∈₀ cat-FUNCT α 𝔄 (cat-Set α)(|Obj|)
        by
          (
            cs-concl
              cs-simp: cat-FUNCT-cs-simps
              cs-intro: cat-cs-intros cat-FUNCT-cs-intros
          )
    qed
```

```
    show
      HOM_{Cα}(,𝔉−)(|ArrMap|)(|f ∘_{A 𝔅} g|) =
        HOM_{Cα}(,𝔉−)(|ArrMap|)(|g|) ∘_{A cat-FUNCT α 𝔄 (cat-Set α)}
        HOM_{Cα}(,𝔉−)(|ArrMap|)(|f|)
      if g : c ↦_𝔅 b and f : b ↦_𝔅 a for b c g a f
      using that
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
    show
      HOM_{Cα}(,𝔉−)(|ArrMap|)(|𝔅(|CId|)(|c|)|) =
        cat-FUNCT α 𝔄 (cat-Set α)(|CId|)(|HOM_{Cα}(,𝔉−)(|ObjMap|)(|c|)|)
      if c ∈_∘ 𝔅(|Obj|) for c
      using that
      by
        (
          cs-concl
            cs-simp: cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
  qed
    (
      use assms(2) in
        ‹
          cs-concl
            cs-simp: cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
            cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
        ›
    )+

qed

lemma (in is-functor) cf-HOM-snd-is-functor'[cat-cs-intros]:
  assumes 𝒵 β
    and α ∈_∘ β
    and 𝔇' = op-cat 𝔅
    and 𝔇 = cat-FUNCT α 𝔄 (cat-Set α)
  shows HOM_{Cα}(,𝔉−) : 𝔇' ↦↦_{Cβ} 𝔇
  using assms(1,2) unfolding assms(3,4) by (rule cf-HOM-snd-is-functor)

lemmas [cat-cs-intros] = is-functor.cf-HOM-snd-is-functor'

lemma (in is-functor) cf-HOM-fst-is-functor:
  assumes 𝒵 β and α ∈_∘ β
  shows HOM_{Cα}(𝔉−,) : 𝔅 ↦↦_{Cβ} cat-FUNCT α (op-cat 𝔄) (cat-Set α)
  by
    (
      rule is-functor.cf-HOM-snd-is-functor[
        OF is-functor-op assms, unfolded cat-op-simps
        ]
    )

lemma (in is-functor) cf-HOM-fst-is-functor'[cat-cs-intros]:
  assumes 𝒵 β
    and α ∈_∘ β
```

and $\mathfrak{C}' = \mathfrak{B}$
and $\mathfrak{D} = \mathit{cat\text{-}FUNCT}\ \alpha\ (\mathit{op\text{-}cat}\ \mathfrak{A})\ (\mathit{cat\text{-}Set}\ \alpha)$
shows $\mathit{HOM}_{C\alpha}(\mathfrak{F}-,) : \mathfrak{C}' \mapsto\mapsto_{C\beta} \mathfrak{D}$
using *assms*(*1*,*2*) **unfolding** *assms*(*3*,*4*) **by** (*rule cf-HOM-fst-is-functor*)

lemmas [*cat-cs-intros*] = *is-functor.cf-HOM-fst-is-functor'*

29.13 Evaluation functor

29.13.1 Definition and elementary properties

See Chapter III-2 in [7].

definition *cf-eval* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *cf-eval* $\alpha\ \beta\ \mathfrak{C}$ =
[
($\lambda\mathfrak{F} d\in_\circ (\mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C})(\!|\mathit{Obj}|\!).\ \mathfrak{F}d(\!|0|\!)(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{F}d(\!|1_{\mathbb{N}}|\!)|\!)$),
(
$\lambda\mathfrak{N} f\in_\circ (\mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C})(\!|\mathit{Arr}|\!).$
$\mathit{cf\text{-}eval\text{-}arrow}\ \mathfrak{C}\ (\mathfrak{N}f(\!|0|\!))\ (\mathfrak{N}f(\!|1_{\mathbb{N}}|\!))$
),
$\mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C}$,
$\mathit{cat\text{-}Set}\ \beta$
]$_\circ$

Components.

lemma *cf-eval-components*:
shows $\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{ObjMap}|\!) =$
$(\lambda\mathfrak{F} d\in_\circ (\mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C})(\!|\mathit{Obj}|\!).\ \mathfrak{F}d(\!|0|\!)(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{F}d(\!|1_{\mathbb{N}}|\!)|\!))$
and $\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{ArrMap}|\!) =$
(
$\lambda\mathfrak{N} f\in_\circ (\mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C})(\!|\mathit{Arr}|\!).$
$\mathit{cf\text{-}eval\text{-}arrow}\ \mathfrak{C}\ (\mathfrak{N}f(\!|0|\!))\ (\mathfrak{N}f(\!|1_{\mathbb{N}}|\!))$
)
and [*cat-cs-simps*]:
$\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{HomDom}|\!) = \mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C}$
and [*cat-cs-simps*]: $\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{HomCod}|\!) = \mathit{cat\text{-}Set}\ \beta$
unfolding *cf-eval-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

29.13.2 Object map

lemma *cf-eval-ObjMap-vsv*[*cat-cs-intros*]: *vsv* ($\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{ObjMap}|\!)$)
unfolding *cf-eval-components* **by** *simp*

lemma *cf-eval-ObjMap-vdomain*[*cat-cs-simps*]:
$\mathcal{D}_\circ\ (\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{ObjMap}|\!)) = (\mathit{cat\text{-}FUNCT}\ \alpha\ \mathfrak{C}\ (\mathit{cat\text{-}Set}\ \alpha) \times_C \mathfrak{C})(\!|\mathit{Obj}|\!)$
unfolding *cf-eval-components* **by** *simp*

lemma (**in** *category*) *cf-eval-ObjMap-app*[*cat-cs-simps*]:
assumes $\mathfrak{F}c = [\mathit{cf\text{-}map}\ \mathfrak{F},\ c]_\circ$
and $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathit{cat\text{-}Set}\ \alpha$
and $c \in_\circ \mathfrak{C}(\!|\mathit{Obj}|\!)$
shows $\mathit{cf\text{-}eval}\ \alpha\ \beta\ \mathfrak{C}(\!|\mathit{ObjMap}|\!)(\!|\mathfrak{F}c|\!) = \mathfrak{F}(\!|\mathit{ObjMap}|\!)(\!|c|\!)$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}\ \langle \mathit{cat\text{-}Set}\ \alpha\rangle\ \mathfrak{F}$ **by** (*rule assms*(*2*))
define β **where** $\beta = \alpha + \omega$
have $\mathcal{Z}\ \beta$ **and** $\alpha\beta$: $\alpha \in_\circ \beta$
by (*simp-all add*: β*-def* $\mathcal{Z}$*-Limit-αω* $\mathcal{Z}$*-ω-αω* $\mathcal{Z}$*-def* $\mathcal{Z}$*-α-αω*)
then interpret β: $\mathcal{Z}\ \beta$ **by** *simp*

```
  note [cat-small-cs-intros] = cat-category-if-ge-Limit
  from assms(2,3) αβ have ℑc ∈∘ (cat-FUNCT α ℭ (cat-Set α) ×_C ℭ)⦇Obj⦈
    by
      (
        cs-concl cs-shallow
          cs-simp: assms(1) cat-FUNCT-components(1)
          cs-intro:
            cat-cs-intros
            cat-small-cs-intros
            cat-prod-cs-intros
            cat-FUNCT-cs-intros
      )
  then show ?thesis
    by (simp add: assms(1) cf-map-components cf-eval-components nat-omega-simps)
qed

lemmas [cat-cs-simps] = category.cf-eval-ObjMap-app
```

29.13.3 Arrow map

```
lemma cf-eval-ArrMap-vsv[cat-cs-intros]: vsv (cf-eval α β ℭ⦇ArrMap⦈)
  unfolding cf-eval-components by simp

lemma cf-eval-ArrMap-vdomain[cat-cs-simps]:
  𝒟∘ (cf-eval α β ℭ⦇ArrMap⦈) = (cat-FUNCT α ℭ (cat-Set α) ×_C ℭ)⦇Arr⦈
  unfolding cf-eval-components by simp

lemma (in category) cf-eval-ArrMap-app[cat-cs-simps]:
  assumes 𝔑f = [ntcf-arrow 𝔑, f]∘
    and 𝔑 : ℑ ↦_CF 𝔊 : ℭ ↦↦_Cα cat-Set α
    and f : a ↦_ℭ b
  shows cf-eval α β ℭ⦇ArrMap⦈⦇𝔑f⦈ = cf-eval-arrow ℭ (ntcf-arrow 𝔑) f
proof–
  interpret ℑ: is-ntcf α ℭ ‹cat-Set α› ℑ 𝔊 𝔑 by (rule assms(2))
  define β where β = α + ω
  have 𝒵 β and αβ: α ∈∘ β
    by (simp-all add: β-def 𝒵-Limit-αω 𝒵-ω-αω 𝒵-def 𝒵-α-αω)
  then interpret β: 𝒵 β by simp
  note [cat-small-cs-intros] = cat-category-if-ge-Limit
  from assms(1,3) αβ have 𝔑f ∈∘ (cat-FUNCT α ℭ (cat-Set α) ×_C ℭ)⦇Arr⦈
    by
      (
        cs-concl
          cs-simp: assms(1) cat-FUNCT-components(1)
          cs-intro:
            cat-cs-intros
            cat-small-cs-intros
            cat-prod-cs-intros
            cat-FUNCT-cs-intros
      )
  then show ?thesis
    by (simp add: assms(1) cf-map-components cf-eval-components nat-omega-simps)
qed

lemmas [cat-cs-simps] = category.cf-eval-ArrMap-app
```

29.13.4 Evaluation functor is a functor

lemma (**in** *category*) *cat-cf-eval-is-functor*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows *cf-eval* $\alpha\ \beta\ \mathfrak{C}$: *cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α) $\times_C\ \mathfrak{C} \mapsto\mapsto_{C\beta}$ *cat-Set* β
proof–

interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(*1*))
from *assms*(*2*) *cat-category-if-ge-Limit*[*OF assms*] **interpret** *FUNCT*:
category β ‹(*cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α))›
by
(
cs-concl **cs-intro**:
cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
)
interpret $\beta\mathfrak{C}$: *category* $\beta\ \mathfrak{C}$
by (*rule category.cat-category-if-ge-Limit*)
(*use assms*(*2*) **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+
interpret *cat-Set-*$\alpha\beta$: *subcategory* β ‹*cat-Set* α› ‹*cat-Set* β›
by (*rule subcategory-cat-Set-cat-Set*[*OF assms*])

show *?thesis*
proof(*intro is-functorI'*)
show *vfsequence* (*cf-eval* $\alpha\ \beta\ \mathfrak{C}$) **unfolding** *cf-eval-def* **by** *simp*
from *cat-category-if-ge-Limit*[*OF assms*] **show**
category β ((*cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α)) $\times_C\ \mathfrak{C}$)
by (*cs-concl* **cs-shallow cs-intro**: *cat-small-cs-intros cat-cs-intros*)
show *vcard* (*cf-eval* $\alpha\ \beta\ \mathfrak{C}$) = $\mathcal{4}_{\mathbb{N}}$
unfolding *cf-eval-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*cf-eval* $\alpha\ \beta\ \mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_{\circ}$ *cat-Set* $\beta(\!|Obj|\!)$
proof(*intro vsv.vsv-vrange-vsubset*, *unfold cat-cs-simps*)
fix $\mathfrak{F}c$ **assume** *prems*: $\mathfrak{F}c \in_{\circ}$ (*cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α) $\times_C\ \mathfrak{C}$)$(\!|Obj|\!)$
then obtain $\mathfrak{F}\ c$
where $\mathfrak{F}c$-*def*: $\mathfrak{F}c = [\mathfrak{F},\ c]_{\circ}$
and $\mathfrak{F}$: $\mathfrak{F} \in_{\circ}$ *cf-maps* $\alpha\ \mathfrak{C}$ (*cat-Set* α)
and *c*: $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
by
(
auto
elim: *cat-prod-2-ObjE*[*rotated 2*]
intro: *FUNCT.category-axioms* $\beta\mathfrak{C}$.*category-axioms*
simp: *cat-FUNCT-components*(*1*)
)
from $\mathfrak{F}$ **obtain** $\mathfrak{G}$ **where** $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *cf-map* $\mathfrak{G}$
and $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
by (*elim cf-mapsE*)
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)
from $\mathfrak{G}\ c$ **show** *cf-eval* $\alpha\ \beta\ \mathfrak{C}(\!|ObjMap|\!)(\!|\mathfrak{F}c|\!) \in_{\circ}$ *cat-Set* $\beta(\!|Obj|\!)$
unfolding $\mathfrak{F}c$-*def* $\mathfrak{F}$-*def*
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps*
cs-intro: *cat-cs-intros cat-Set-*$\alpha\beta$.*subcat-Obj-vsubset*
)
qed (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
show *cf-eval* $\alpha\ \beta\ \mathfrak{C}(\!|ArrMap|\!)(\!|\mathfrak{N}f|\!)$:
cf-eval $\alpha\ \beta\ \mathfrak{C}(\!|ObjMap|\!)(\!|\mathfrak{F}a|\!) \mapsto_{cat\text{-}Set\ \beta}$ *cf-eval* $\alpha\ \beta\ \mathfrak{C}(\!|ObjMap|\!)(\!|\mathfrak{G}b|\!)$

```
  if 𝔑f: 𝔑f : ℱa ↦_cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈 𝔊b for ℱa 𝔊b 𝔑f
proof–
  obtain 𝔑 f ℱ a 𝔊 b
    where 𝔑f-def: 𝔑f = [𝔑, f]∘
      and ℱa-def: ℱa = [ℱ, a]∘
      and 𝔊b-def: 𝔊b = [𝔊, b]∘
      and 𝔑: 𝔑 : ℱ ↦_cat-FUNCT α 𝔈 (cat-Set α) 𝔊
      and f: f : a ↦_𝔈 b
    by
      (
        auto intro:
          cat-prod-2-is-arrE[rotated 2, OF 𝔑f]
          FUNCT.category-axioms
          β𝔈.category-axioms
      )
  note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
  from 𝔑(1) f assms(2) show cf-eval α β 𝔈(|ArrMap|)(|𝔑f|) :
    cf-eval α β 𝔈(|ObjMap|)(|ℱa|) ↦_cat-Set β cf-eval α β 𝔈(|ObjMap|)(|𝔊b|)
    unfolding 𝔑f-def ℱa-def 𝔊b-def
    by
      (
        intro cat-Set-αβ.subcat-is-arrD,
        use nothing in ‹subst 𝔑(2), subst 𝔑(3), subst 𝔑(4)›
      )
      (
        cs-concl
          cs-simp: cat-FUNCT-cs-simps cat-cs-simps cs-intro: cat-cs-intros
      )
qed
show
  cf-eval α β 𝔈(|ArrMap|)(|𝔐g ∘_A cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈 𝔑f|) =
    cf-eval α β 𝔈(|ArrMap|)(|𝔐g|) ∘_A cat-Set β cf-eval α β 𝔈(|ArrMap|)(|𝔑f|)
  if 𝔐g: 𝔐g : 𝔊b ↦_cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈 ℌc
    and 𝔑f: 𝔑f : ℱa ↦_cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈 𝔊b
  for 𝔑f 𝔐g ℱa 𝔊b ℌc
proof–
  obtain 𝔑 f ℱ a 𝔊 b
    where 𝔑f-def: 𝔑f = [𝔑, f]∘
      and ℱa-def: ℱa = [ℱ, a]∘
      and 𝔊b-def: 𝔊b = [𝔊, b]∘
      and 𝔑: 𝔑 : ℱ ↦_cat-FUNCT α 𝔈 (cat-Set α) 𝔊
      and f: f : a ↦_𝔈 b
    by
      (
        auto intro:
          cat-prod-2-is-arrE[rotated 2, OF 𝔑f]
          FUNCT.category-axioms
          β𝔈.category-axioms
      )
  then obtain 𝔐 g ℌ c
    where 𝔐g-def: 𝔐g = [𝔐, g]∘
      and ℌc-def: ℌc = [ℌ, c]∘
      and 𝔐: 𝔐 : 𝔊 ↦_cat-FUNCT α 𝔈 (cat-Set α) ℌ
      and g: g : b ↦_𝔈 c
    by
      (
        auto intro:
```

```
        cat-prod-2-is-arrE[rotated 2, OF 𝔐g]
        FUNCT.category-axioms
        βℭ.category-axioms
      )
    note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
     and 𝔐 = cat-FUNCT-is-arrD[OF 𝔐]
    from 𝔑(1) 𝔐(1) f g show
      cf-eval α β ℭ(|ArrMap|)(|𝔐g ∘A cat-FUNCT α ℭ (cat-Set α) ×C ℭ 𝔑f|) =
        cf-eval α β ℭ(|ArrMap|)(|𝔐g|) ∘A cat-Set β cf-eval α β ℭ(|ArrMap|)(|𝔑f|)
      unfolding 𝔐g-def 𝔑f-def 𝔉a-def 𝔊b-def ℌc-def
      by
        (
          subst (1 2) 𝔐(2), use nothing in ‹subst (1 2) 𝔑(2)›,
          cs-concl-step cs-shallow cat-Set-αβ.subcat-Comp-simp[symmetric]
        )
        (
          cs-concl
            cs-simp: cat-cs-simps cat-prod-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros
        )
  qed
  show
    cf-eval α β ℭ(|ArrMap|)(|(cat-FUNCT α ℭ (cat-Set α) ×C ℭ)(|CId|)(|𝔉c|)|) =
      cat-Set β(|CId|)(|cf-eval α β ℭ(|ObjMap|)(|𝔉c|)|)
    if 𝔉c ∈∘ (cat-FUNCT α ℭ (cat-Set α) ×C ℭ)(|Obj|) for 𝔉c
  proof–
    from that obtain 𝔉 c where 𝔉c-def: 𝔉c = [𝔉, c]∘
      and 𝔉: 𝔉 ∈∘ cf-maps α ℭ (cat-Set α)
      and c: c ∈∘ ℭ(|Obj|)
      by
        (
          auto
            elim: cat-prod-2-ObjE[rotated 2]
            intro: FUNCT.category-axioms βℭ.category-axioms
            simp: cat-FUNCT-components(1)
        )
    from 𝔉 obtain 𝔊 where 𝔉-def: 𝔉 = cf-map 𝔊
      and 𝔊: 𝔊 : ℭ ↦↦C α cat-Set α
      by (elim cf-mapsE)
    interpret 𝔊: is-functor α ℭ ‹cat-Set α› 𝔊 by (rule 𝔊)
    from 𝔊 c show
      cf-eval α β ℭ(|ArrMap|)(|(cat-FUNCT α ℭ (cat-Set α) ×C ℭ)(|CId|)(|𝔉c|)|) =
        cat-Set β(|CId|)(|cf-eval α β ℭ(|ObjMap|)(|𝔉c|)|)
      unfolding 𝔉c-def 𝔉-def
      by (cs-concl-step cat-Set-αβ.subcat-CId[symmetric])
        (
          cs-concl
            cs-simp: cat-cs-simps cat-prod-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros
        )
  qed

 qed (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

qed

lemma (in category) cat-cf-eval-is-functor':
 assumes 𝒵 β
```

and $\alpha \in_\circ \beta$
and $\mathfrak{A}' = \textit{cat-FUNCT}\ \alpha\ \mathfrak{C}\ (\textit{cat-Set}\ \alpha) \times_C \mathfrak{C}$
and $\mathfrak{B}' = \textit{cat-Set}\ \beta$
and $\beta' = \beta$
shows $\textit{cf-eval}\ \alpha\ \beta\ \mathfrak{C} : \mathfrak{A}' \mapsto\mapsto_{C\beta'} \mathfrak{B}'$
using *assms*(*1,2*) **unfolding** *assms*(*3–5*) **by** (*rule cat-cf-eval-is-functor*)

lemmas [*cat-cs-intros*] = *category.cat-cf-eval-is-functor'*

29.14 N-functor

29.14.1 Definition and elementary properties

See Chapter III-2 in [7].

definition $\textit{cf-nt} :: V \Rightarrow V \Rightarrow V \Rightarrow V$
where $\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F} =$
$\textit{bifunctor-flip}\ (\mathfrak{F}(\!|\textit{HomCod}|\!))\ (\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$

Alternative definition.

lemma (**in** *is-functor*) *cf-nt-def'*:
$\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F} =$
$\textit{bifunctor-flip}\ \mathfrak{B}\ (\textit{cat-FUNCT}\ \alpha\ \mathfrak{A}\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ \mathfrak{A}\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$
unfolding *cf-nt-def cf-HomDom cf-HomCod* **by** *simp*

Components.

lemma *cf-nt-components*:
shows $\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F}(\!|\textit{ObjMap}|\!) =$
$($
$\textit{bifunctor-flip}\ (\mathfrak{F}(\!|\textit{HomCod}|\!))\ (\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$
$)(\!|\textit{ObjMap}|\!)$
and $\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F}(\!|\textit{ArrMap}|\!) =$
$($
$\textit{bifunctor-flip}\ (\mathfrak{F}(\!|\textit{HomCod}|\!))\ (\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$
$)(\!|\textit{ArrMap}|\!)$
and $\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F}(\!|\textit{HomDom}|\!) =$
$($
$\textit{bifunctor-flip}\ (\mathfrak{F}(\!|\textit{HomCod}|\!))\ (\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$
$)(\!|\textit{HomDom}|\!)$
and $\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F}(\!|\textit{HomCod}|\!) =$
$($
$\textit{bifunctor-flip}\ (\mathfrak{F}(\!|\textit{HomCod}|\!))\ (\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ (\mathfrak{F}(\!|\textit{HomDom}|\!))\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$
$)(\!|\textit{HomCod}|\!)$
unfolding *cf-nt-def* **by** *simp-all*

lemma (**in** *is-functor*) *cf-nt-components'*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows $\textit{cf-nt}\ \alpha\ \beta\ \mathfrak{F}(\!|\textit{ObjMap}|\!) =$
$($
$\textit{bifunctor-flip}\ \mathfrak{B}\ (\textit{cat-FUNCT}\ \alpha\ \mathfrak{A}\ (\textit{cat-Set}\ \alpha))$
$(\textit{Hom}_{O.C\beta}\textit{cat-FUNCT}\ \alpha\ \mathfrak{A}\ (\textit{cat-Set}\ \alpha)(\textit{HOM}_{C\alpha}(,\mathfrak{F}-)-,-))$
$)(\!|\textit{ObjMap}|\!)$

```
    and cf-nt α β 𝔉(⁅ArrMap⁆) =
      (
        bifunctor-flip 𝔙 (cat-FUNCT α 𝔄 (cat-Set α))
          (Hom_{O.Cβ} cat-FUNCT α 𝔄 (cat-Set α)(HOM_{Cα}(,𝔉−)−,−))
      )(⁅ArrMap⁆)
    and [cat-cs-simps]:
      cf-nt α β 𝔉(⁅HomDom⁆) = cat-FUNCT α 𝔄 (cat-Set α) ×_C 𝔙
    and [cat-cs-simps]:
      cf-nt α β 𝔉(⁅HomCod⁆) = cat-Set β
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret β𝔄: category β 𝔄
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret β𝔙: category β 𝔙
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  show
    cf-nt α β 𝔉(⁅ObjMap⁆) =
      (
        bifunctor-flip 𝔙 (cat-FUNCT α 𝔄 (cat-Set α))
          (Hom_{O.Cβ} cat-FUNCT α 𝔄 (cat-Set α)(HOM_{Cα}(,𝔉−)−,−))
      )(⁅ObjMap⁆)
    cf-nt α β 𝔉(⁅ArrMap⁆) =
      (
        bifunctor-flip 𝔙 (cat-FUNCT α 𝔄 (cat-Set α))
          (Hom_{O.Cβ} cat-FUNCT α 𝔄 (cat-Set α)(HOM_{Cα}(,𝔉−)−,−))
      )(⁅ArrMap⁆)
    cf-nt α β 𝔉(⁅HomDom⁆) = cat-FUNCT α 𝔄 (cat-Set α) ×_C 𝔙
    cf-nt α β 𝔉(⁅HomCod⁆) = cat-Set β
    unfolding cf-nt-def
    using assms(2)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro: cat-small-cs-intros cat-cs-intros cat-FUNCT-cs-intros
      )+
qed

lemmas [cat-cs-simps] = is-functor.cf-nt-components'(3,4)
```

29.14.2 Object map

```
lemma cf-nt-ObjMap-vsv[cat-cs-intros]: vsv (cf-nt α β 𝔈(⁅ObjMap⁆))
  unfolding cf-nt-components by (cs-intro-step cat-cs-intros)

lemma (in is-functor) cf-nt-ObjMap-vdomain[cat-cs-simps]:
  assumes 𝒵 β and α ∈∘ β
  shows 𝒟∘ (cf-nt α β 𝔉(⁅ObjMap⁆)) = (cat-FUNCT α 𝔄 (cat-Set α) ×_C 𝔙)(⁅Obj⁆)
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret β𝔄: category β 𝔄
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret β𝔙: category β 𝔙
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
```

```
  from assms(2) show ?thesis
    unfolding cf-nt-components
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-prod-cs-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-nt-ObjMap-vdomain

lemma (in is-functor) cf-nt-ObjMap-app[cat-cs-simps]:
  assumes 𝒵 β
    and α ∈∘ β
    and 𝔊b = [cf-map 𝔊, b]∘
    and 𝔊 : 𝔄 ↦↦_{C α} cat-Set α
    and b ∈∘ 𝔅(|Obj|)
  shows cf-nt α β 𝔉(|ObjMap|)(|𝔊b|) = Hom
    (cat-FUNCT α 𝔄 (cat-Set α))
    (cf-map (Hom_{O.C α}𝔅(b,−) ∘_{CF} 𝔉))
    (cf-map 𝔊)
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret β𝔄: category β 𝔄
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret β𝔅: category β 𝔅
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret 𝔊: is-functor α 𝔄 ‹cat-Set α› 𝔊 by (rule assms(4))
  from assms(2,5) show ?thesis
    unfolding assms(3) cf-nt-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-cs-intros
            cat-small-cs-intros
            cat-FUNCT-cs-intros
            cat-prod-cs-intros
            cat-op-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-nt-ObjMap-app
```

29.14.3 Arrow map

```
lemma cf-nt-ArrMap-vsv[cat-cs-intros]: vsv (cf-nt α β 𝔈(|ArrMap|))
  unfolding cf-nt-components by (cs-intro-step cat-cs-intros)

lemma (in is-functor) cf-nt-ArrMap-vdomain[cat-cs-simps]:
```

```
  assumes 𝒵 β and α ∈∘ β
  shows 𝒟∘ (cf-nt α β 𝔉(|ArrMap|)) = (cat-FUNCT α 𝔄 (cat-Set α) ×C 𝔅)(|Arr|)
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret β𝔄: category β 𝔄
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret β𝔅: category β 𝔅
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  from assms(2) show ?thesis
    unfolding cf-nt-components
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-small-cs-intros
            cat-cs-intros
            cat-FUNCT-cs-intros
            cat-prod-cs-intros
      )
qed

lemmas [cat-cs-simps] = is-functor.cf-nt-ArrMap-vdomain

lemma (in is-functor) cf-nt-ArrMap-app[cat-cs-simps]:
  assumes 𝒵 β
    and α ∈∘ β
    and 𝔑f = [ntcf-arrow 𝔑, f]∘
    and 𝔑 : 𝔊 ↦CF 𝔎 : 𝔄 ↦↦Cα cat-Set α
    and f : a ↦𝔅 b
  shows cf-nt α β 𝔉(|ArrMap|)(|𝔑f|) = cf-hom
    (cat-FUNCT α 𝔄 (cat-Set α))
    [ntcf-arrow (HomA.Cα𝔅(f,−) ∘NTCF−CF 𝔉), ntcf-arrow 𝔑]∘
proof–
  interpret β: 𝒵 β by (rule assms(1))
  interpret β𝔄: category β 𝔄
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret β𝔅: category β 𝔅
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  interpret 𝔑: is-ntcf α 𝔄 ‹cat-Set α› 𝔊 𝔎 𝔑 by (rule assms(4))
  from assms(2,5) show ?thesis
    unfolding assms(3) cf-nt-def
    by
      (
        cs-concl
          cs-simp: cat-cs-simps cat-FUNCT-cs-simps cat-op-simps
          cs-intro:
            cat-cs-intros
            cat-small-cs-intros
            cat-FUNCT-cs-intros
            cat-prod-cs-intros
            cat-op-intros
      )
qed
```

lemmas [*cat-cs-simps*] = *is-functor.cf-nt-ArrMap-app*

29.14.4 *N*-functor is a functor

lemma (**in** *is-functor*) *cf-nt-is-functor*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows *cf-nt* $\alpha\ \beta\ \mathfrak{F} : \textit{cat-FUNCT}\ \alpha\ \mathfrak{A}\ (\textit{cat-Set}\ \alpha) \times_{C} \mathfrak{B} \mapsto\mapsto_{C\beta} \textit{cat-Set}\ \beta$
proof–
interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(*1*))
interpret $\beta\mathfrak{A}$: *category* $\beta\ \mathfrak{A}$
by (*rule category.cat-category-if-ge-Limit*)
(*use assms*(*2*) **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+
interpret $\beta\mathfrak{B}$: *category* $\beta\ \mathfrak{B}$
by (*rule category.cat-category-if-ge-Limit*)
(*use assms*(*2*) **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+
from *assms*(*2*) **show** *?thesis*
unfolding *cf-nt-def′*
by
(
cs-concl
cs-simp: *cat-op-simps*
cs-intro: *cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros*
)
qed

lemma (**in** *is-functor*) *cf-nt-is-functor′*:
assumes $\mathcal{Z}\ \beta$
and $\alpha \in_{\circ} \beta$
and $\mathfrak{A}' = \textit{cat-FUNCT}\ \alpha\ \mathfrak{A}\ (\textit{cat-Set}\ \alpha) \times_{C} \mathfrak{B}$
and $\mathfrak{B}' = \textit{cat-Set}\ \beta$
and $\beta' = \beta$
shows *cf-nt* $\alpha\ \beta\ \mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{C\beta'} \mathfrak{B}'$
using *assms*(*1,2*) **unfolding** *assms*(*3–5*) **by** (*rule cf-nt-is-functor*)

lemmas [*cat-cs-intros*] = *is-functor.cf-nt-is-functor′*

29.15 Yoneda natural transformation arrow

29.15.1 Definition and elementary properties

The following subsection is based on the elements of the content of Chapter III-2 in [7].

definition *ntcf-Yoneda-arrow* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *ntcf-Yoneda-arrow* $\alpha\ \mathfrak{C}\ \mathfrak{F}\ r$ =
[
(
$\lambda\psi \in_{\circ} \textit{Hom}\ (\textit{cat-FUNCT}\ \alpha\ \mathfrak{C}\ (\textit{cat-Set}\ \alpha))\ (\textit{cf-map}\ (\textit{Hom}_{O.C\alpha}\mathfrak{C}(r,-)))\ \mathfrak{F}.$
Yoneda-map $\alpha\ (\textit{cf-of-cf-map}\ \mathfrak{C}\ (\textit{cat-Set}\ \alpha)\ \mathfrak{F})\ r\lfloor\!\lfloor$
ntcf-of-ntcf-arrow $\mathfrak{C}\ (\textit{cat-Set}\ \alpha)\ \psi$
$\rfloor\!\rfloor$
),
$\textit{Hom}\ (\textit{cat-FUNCT}\ \alpha\ \mathfrak{C}\ (\textit{cat-Set}\ \alpha))\ (\textit{cf-map}\ (\textit{Hom}_{O.C\alpha}\mathfrak{C}(r,-)))\ \mathfrak{F},$
$\mathfrak{F}(\lfloor\!\lfloor \textit{ObjMap} \rfloor\!\rfloor)(\lfloor\!\lfloor r \rfloor\!\rfloor)$
]$_{\circ}$

Components

lemma *ntcf-Yoneda-arrow-components*:
shows *ntcf-Yoneda-arrow* $\alpha\ \mathfrak{C}\ \mathfrak{F}\ r(\lfloor\!\lfloor \textit{ArrVal} \rfloor\!\rfloor)$ =

(
$\lambda\psi\in_\circ Hom\ (cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ (cat\text{-}Set\ \alpha))\ (cf\text{-}map\ (Hom_{O.C\alpha}\mathfrak{C}(r,-)))\ \mathfrak{F}.$
Yoneda-map α (*cf-of-cf-map* $\mathfrak{C}$ (*cat-Set* α) $\mathfrak{F}$) r(|
ntcf-of-ntcf-arrow $\mathfrak{C}$ (*cat-Set* α) ψ
|)
)
and [*cat-cs-simps*]: *ntcf-Yoneda-arrow* α $\mathfrak{C}$ $\mathfrak{F}$ r(|*ArrDom*|) =
Hom (*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α)) (*cf-map* ($Hom_{O.C\alpha}\mathfrak{C}(r,-)$)) $\mathfrak{F}$
and [*cat-cs-simps*]: *ntcf-Yoneda-arrow* α $\mathfrak{C}$ $\mathfrak{F}$ r(|*ArrCod*|) = $\mathfrak{F}$(|*ObjMap*|)(|r|)
unfolding *ntcf-Yoneda-arrow-def arr-field-simps*
by (*simp-all add*: *nat-omega-simps*)

29.15.2 Arrow map

mk-VLambda *ntcf-Yoneda-arrow-components*(*1*)
|*vsv ntcf-Yoneda-arrow-vsv*[*cat-cs-intros*]|
|*vdomain ntcf-Yoneda-arrow-vdomain*[*cat-cs-simps*]|

context *category*
begin

context
fixes $\mathfrak{F}$ :: *V*
begin

mk-VLambda *ntcf-Yoneda-arrow-components*(*1*)[**where** α=α **and** $\mathfrak{C}$=$\mathfrak{C}$ **and** $\mathfrak{F}$=‹*cf-map* $\mathfrak{F}$›]
|*app ntcf-Yoneda-arrow-app′*|

lemmas *ntcf-Yoneda-arrow-app* =
ntcf-Yoneda-arrow-app′[*unfolded in-Hom-iff*, *cat-cs-simps*]

end

end

lemmas [*cat-cs-simps*] = *category.ntcf-Yoneda-arrow-app*

29.15.3 Several technical lemmas

lemma (**in** *vsv*) *vsv-vrange-VLambda-app*:
assumes g ‘ *elts* A = *elts* ($\mathcal{D}_\circ$ r)
shows $\mathcal{R}_\circ$ ($\lambda x\in_\circ A.\ r$(|g x|)) = $\mathcal{R}_\circ$ r
proof(*intro vsubset-antisym vsv.vsv-vrange-vsubset*, *unfold vdomain-VLambda*)
show ($\lambda x\in_\circ A.\ r$(|g x|))(|x|) $\in_\circ$ $\mathcal{R}_\circ$ r **if** x $\in_\circ$ A **for** x
proof–
from *assms that* **have** g x $\in_\circ$ $\mathcal{D}_\circ$ r **by** *auto*
then have r(|g x|) $\in_\circ$ $\mathcal{R}_\circ$ r **by** *force*
with *that* **show** *?thesis* **by** *simp*
qed
show r(|x|) $\in_\circ$ $\mathcal{R}_\circ$ ($\lambda x\in_\circ A.\ r$(|g x|)) **if** x $\in_\circ$ $\mathcal{D}_\circ$ r **for** x
proof–
from *that assms* **have** x ∈ g ‘ *elts* A **by** *simp*
then obtain c **where** c: c $\in_\circ$ A **and** *x-def*: x = g c **by** *clarsimp*
from c **show** *?thesis* **unfolding** *x-def* **by** *auto*
qed
qed *auto*

lemma (**in** *vsv*) *vsv-vrange-VLambda-app′*:

assumes g ‘ $elts\ A = elts\ (\mathcal{D}_\circ\ r)$
and $R = \mathcal{R}_\circ\ r$
shows $\mathcal{R}_\circ\ (\lambda x \in_\circ A.\ r(\!|g\ x|\!)) = R$
using *assms*(*1*) **unfolding** *assms*(*2*) **by** (*rule vsv-vrange-VLambda-app*)

lemma (**in** *v11*) *v11-VLambda-v11-bij-betw-comp*:
assumes *bij-betw g* (*elts A*) (*elts* ($\mathcal{D}_\circ$ *r*))
shows *v11* ($\lambda x \in_\circ A.\ r(\!|g\ x|\!)$)
proof(*rule vsv.vsv-valeq-v11I*, *unfold vdomain-VLambda beta*)
fix *x y* **assume** *prems*: $x \in_\circ A\ y \in_\circ A\ r(\!|g\ x|\!) = r(\!|g\ y|\!)$
from *assms prems*(*1,2*) **have** $g\ x \in_\circ \mathcal{D}_\circ\ r$ **and** $g\ y \in_\circ \mathcal{D}_\circ\ r$ **by** *auto*
from *v11-injective*[*OF this prems*(*3*)] **have** $g\ x = g\ y$**.**
with *assms prems*(*1,2*) **show** $x = y$ **unfolding** *bij-betw-def inj-on-def* **by** *simp*
qed *simp*

29.15.4 Yoneda natural transformation arrow is an arrow in the category *Set*

lemma (**in** *category*) *cat-ntcf-Yoneda-arrow-is-arr-isomoprhism*:
assumes $\mathcal{Z}\ \beta$
and $\alpha \in_\circ \beta$
and $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $r \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *ntcf-Yoneda-arrow* $\alpha\ \mathfrak{C}$ (*cf-map* $\mathfrak{F}$) r :
Hom
(*cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α))
(*cf-map* ($Hom_{O.C\alpha}\mathfrak{C}(r,-)$))
(*cf-map* $\mathfrak{F}$) $\mapsto_{iso}$ *cat-Set* β
$\mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
proof–

interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{F}$ **by** (*rule assms*)

from *assms*(*2*) **interpret** *FUNCT*: *tiny-category* β ‹*cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α)›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)

let *?Hom-r* = ‹$Hom_{O.C\alpha}\mathfrak{C}(r,-)$›

from *assms* **have** [*cat-cs-simps*]: *cf-of-cf-map* $\mathfrak{C}$ (*cat-Set* α) (*cf-map* $\mathfrak{F}$) = $\mathfrak{F}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-FUNCT-cs-simps*)

note *Yoneda* = *cat-Yoneda-Lemma*[*OF assms*(*3,4*)]

show *?thesis*
proof
(
intro cat-Set-is-iso-arrI cat-Set-is-arrI arr-SetI,
unfold cat-cs-simps cf-map-components
)
show *vfsequence* (*ntcf-Yoneda-arrow* $\alpha\ \mathfrak{C}$ (*cf-map* $\mathfrak{F}$) r)
unfolding *ntcf-Yoneda-arrow-def* **by** *simp*
show *vcard* (*ntcf-Yoneda-arrow* $\alpha\ \mathfrak{C}$ (*cf-map* $\mathfrak{F}$) r) = $3_{\mathbb{N}}$
unfolding *ntcf-Yoneda-arrow-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*ntcf-Yoneda-arrow* $\alpha\ \mathfrak{C}$ (*cf-map* $\mathfrak{F}$) $r(\!|ArrVal|\!)$) = $\mathfrak{F}(\!|ObjMap|\!)(\!|r|\!)$
unfolding *cat-cs-simps cf-map-components ntcf-Yoneda-arrow-components*
by (*intro vsv.vsv-vrange-VLambda-app′*, *unfold Yoneda*(*2*))
(
use assms(*4*) **in**

```
        ‹
          cs-concl cs-shallow
            cs-simp:
              cat-cs-simps bij-betwD(2)[OF bij-betw-ntcf-of-ntcf-arrow-Hom]
            cs-intro: cat-cs-intros
        ›
      )+
    then show 𝓡∘ (ntcf-Yoneda-arrow α ℭ (cf-map 𝔉) r(|ArrVal|)) ⊆∘ 𝔉(|ObjMap|)(|r|)
      by auto
    from assms(4) show v11 (ntcf-Yoneda-arrow α ℭ (cf-map 𝔉) r(|ArrVal|))
      unfolding ntcf-Yoneda-arrow-components
      by
        (
          intro v11.v11-VLambda-v11-bij-betw-comp,
          unfold cat-cs-simps 𝔉.Yoneda-map-vdomain;
          intro Yoneda bij-betw-ntcf-of-ntcf-arrow-Hom
        )
        (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    from assms(4) show
      Hom (cat-FUNCT α ℭ (cat-Set α)) (cf-map ?Hom-r) (cf-map 𝔉) ∈∘ Vset β
      by (intro FUNCT.cat-Hom-in-Vset)
        (
          cs-concl cs-shallow
            cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros cat-FUNCT-cs-intros
        )
    from assms(4) have 𝔉(|ObjMap|)(|r|) ∈∘ Vset α
      by (cs-concl cs-intro: cat-cs-intros)
    then show 𝔉(|ObjMap|)(|r|) ∈∘ Vset β
      by (auto simp: assms(2) Vset-trans Vset-in-mono)
  qed (auto intro: cat-cs-intros)

qed

lemma (in category) cat-ntcf-Yoneda-arrow-is-arr-isomoprhism':
  assumes 𝒵 β
    and α ∈∘ β
    and 𝔉' = cf-map 𝔉
    and B = 𝔉(|ObjMap|)(|r|)
    and A = Hom
      (cat-FUNCT α ℭ (cat-Set α))
      (cf-map (Hom_{O.Cα}ℭ(r,-)))
      (cf-map 𝔉)
    and 𝔉 : ℭ ↦↦_{Cα} cat-Set α
    and r ∈∘ ℭ(|Obj|)
  shows ntcf-Yoneda-arrow α ℭ 𝔉' r : A ↦_{iso cat-Set β} B
  using assms(1,2,6,7)
  unfolding assms(3–5)
  by (rule cat-ntcf-Yoneda-arrow-is-arr-isomoprhism)

lemmas [cat-arrow-cs-intros] =
  category.cat-ntcf-Yoneda-arrow-is-arr-isomoprhism'

lemma (in category) cat-ntcf-Yoneda-arrow-is-arr:
  assumes 𝒵 β
    and α ∈∘ β
    and 𝔉 : ℭ ↦↦_{Cα} cat-Set α
    and r ∈∘ ℭ(|Obj|)
  shows ntcf-Yoneda-arrow α ℭ (cf-map 𝔉) r :
```

Hom
(*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α))
(*cf-map* (*Hom*$_{O.C\alpha}\mathfrak{C}(r,-)$))
(*cf-map* $\mathfrak{F}$) $\mapsto_{\textit{cat-Set}\ \beta}$
$\mathfrak{F}(\!|\textit{ObjMap}|\!)(\!|r|\!)$
by
(
rule cat-Set-is-iso-arrD[
OF cat-ntcf-Yoneda-arrow-is-arr-isomoprhism[*OF assms*]
]
)

lemma (**in** *category*) *cat-ntcf-Yoneda-arrow-is-arr'*[*cat-cs-intros*]:
assumes $\mathcal{Z}$ β
and $\alpha \in_\circ \beta$
and $\mathfrak{F}' =$ *cf-map* $\mathfrak{F}$
and $B = \mathfrak{F}(\!|\textit{ObjMap}|\!)(\!|r|\!)$
and $A =$ *Hom*
(*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α))
(*cf-map* (*Hom*$_{O.C\alpha}\mathfrak{C}(r,-)$))
(*cf-map* $\mathfrak{F}$)
and $\mathfrak{F} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $r \in_\circ \mathfrak{C}(\!|\textit{Obj}|\!)$
shows *ntcf-Yoneda-arrow* α $\mathfrak{C}$ $\mathfrak{F}'$ $r : A \mapsto_{\textit{cat-Set}\ \beta} B$
using *assms*(*1*,*2*,*6*,*7*)
unfolding *assms*(*3–5*)
by (*rule cat-ntcf-Yoneda-arrow-is-arr*)

lemmas [*cat-arrow-cs-intros*] = *category.cat-ntcf-Yoneda-arrow-is-arr'*

29.16 Commutativity law for the Yoneda natural transformation arrow

lemma (**in** *category*) *cat-ntcf-Yoneda-arrow-commute*:
assumes $\mathcal{Z}$ β
and $\alpha \in_\circ \beta$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{C\,F} \mathfrak{G} : \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $f : a \mapsto_{\mathfrak{C}} b$
shows
ntcf-Yoneda-arrow α $\mathfrak{C}$ (*cf-map* $\mathfrak{G}$) $b \circ_{A\,\textit{cat-Set}\ \beta}$
cf-hom
(*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α))
[*ntcf-arrow* *Hom*$_{A.C\alpha}\mathfrak{C}(f,-)$, *ntcf-arrow* $\mathfrak{N}$]$_\circ$ =
cf-eval-arrow $\mathfrak{C}$ (*ntcf-arrow* $\mathfrak{N}$) $f \circ_{A\,\textit{cat-Set}\ \beta}$
ntcf-Yoneda-arrow α $\mathfrak{C}$ (*cf-map* $\mathfrak{F}$) a
proof–

let *?hom* =
‹
cf-hom
(*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α))
[*ntcf-arrow* *Hom*$_{A.C\alpha}\mathfrak{C}(f,-)$, *ntcf-arrow* $\mathfrak{N}$]$_\circ$
›

interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
interpret $\mathfrak{N}$: *is-ntcf* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(*3*))
interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)
interpret $\beta\mathfrak{C}$: *category* β $\mathfrak{C}$
by (*rule category.cat-category-if-ge-Limit*)

```
    (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
interpret cat-Set-αβ: subcategory β ‹cat-Set α› ‹cat-Set β›
  by (rule subcategory-cat-Set-cat-Set[OF assms(1,2)])

from assms(2,4) have 𝔊b-𝔑f:
  ntcf-Yoneda-arrow α ℭ (cf-map 𝔊) b ∘_A cat-Set β ?hom :
    Hom
      (cat-FUNCT α ℭ (cat-Set α))
      (cf-map (Hom_O.Cα ℭ(a,−)))
      (cf-map 𝔉) ↦_cat-Set β
    𝔊(|ObjMap|)(|b|)
  by
    (
      cs-concl
        cs-intro:
          cat-small-cs-intros
          cat-cs-intros
          cat-prod-cs-intros
          cat-op-intros
          cat-FUNCT-cs-intros
    )

from assms(2,4) have 𝔑f-𝔉a:
  cf-eval-arrow ℭ (ntcf-arrow 𝔑) f ∘_A cat-Set β
    ntcf-Yoneda-arrow α ℭ (cf-map 𝔉) a :
    Hom
      (cat-FUNCT α ℭ (cat-Set α))
      (cf-map (Hom_O.Cα ℭ(a,−)))
      (cf-map 𝔉) ↦_cat-Set β
    𝔊(|ObjMap|)(|b|)
  by (cs-concl cs-intro: cat-cs-intros cat-Set-αβ.subcat-is-arrD)

show ?thesis
proof(rule arr-Set-eqI[of β])

  from 𝔊b-𝔑f show arr-Set-𝔊b-𝔑f:
    arr-Set β (ntcf-Yoneda-arrow α ℭ (cf-map 𝔊) b ∘_A cat-Set β ?hom)
    by (auto dest: cat-Set-is-arrD(1))
  from 𝔊b-𝔑f have dom-lhs:
    𝒟∘ ((ntcf-Yoneda-arrow α ℭ (cf-map 𝔊) b ∘_A cat-Set β ?hom)(|ArrVal|)) =
      Hom
        (cat-FUNCT α ℭ (cat-Set α))
        (cf-map (Hom_O.Cα ℭ(a,−)))
        (cf-map 𝔉)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)+
  interpret 𝔑f-𝔉a: arr-Set
    β ‹ntcf-Yoneda-arrow α ℭ (cf-map 𝔊) b ∘_A cat-Set β ?hom›
    by (rule arr-Set-𝔊b-𝔑f)

  from 𝔑f-𝔉a show arr-Set-𝔑f-𝔉a:
    arr-Set
      β
      (
        cf-eval-arrow ℭ (ntcf-arrow 𝔑) f ∘_A cat-Set β
        ntcf-Yoneda-arrow α ℭ (cf-map 𝔉) a
      )
    by (auto dest: cat-Set-is-arrD(1))
```

```
from 𝔑f-𝔉a have dom-rhs:
  𝒟∘
    (
      (
        cf-eval-arrow 𝔈 (ntcf-arrow 𝔑) f ∘A cat-Set β
        ntcf-Yoneda-arrow α 𝔈 (cf-map 𝔉) a
      )(|ArrVal|)
    ) = Hom
      (cat-FUNCT α 𝔈 (cat-Set α))
      (cf-map (Hom_O.Cα𝔈(a,-)))
      (cf-map 𝔉)
  by (cs-concl cs-shallow cs-simp: cat-cs-simps)

show
  (ntcf-Yoneda-arrow α 𝔈 (cf-map 𝔊) b ∘A cat-Set β ?hom)(|ArrVal|) =
    (
      cf-eval-arrow 𝔈 (ntcf-arrow 𝔑) f ∘A cat-Set β
      ntcf-Yoneda-arrow α 𝔈 (cf-map 𝔉) a
    )(|ArrVal|)
proof(rule vsv-eqI, unfold dom-lhs dom-rhs in-Hom-iff)

  fix 𝔐 assume prems:
    𝔐 : cf-map Hom_O.Cα𝔈(a,-) ↦cat-FUNCT α 𝔈 (cat-Set α) cf-map 𝔉

  from assms(4) have [cat-cs-simps]:
    cf-of-cf-map 𝔈 (cat-Set α) (cf-map Hom_O.Cα𝔈(a,-)) = Hom_O.Cα𝔈(a,-)
    cf-of-cf-map 𝔈 (cat-Set α) (cf-map 𝔉) = 𝔉
    by (cs-concl cs-simp: cat-FUNCT-cs-simps cs-intro: cat-cs-intros)

  note 𝔐 = cat-FUNCT-is-arrD[OF prems, unfolded cat-cs-simps]

  interpret 𝔐: is-ntcf
    α 𝔈 ‹cat-Set α› ‹Hom_O.Cα𝔈(a,-)› 𝔉 ‹ntcf-of-ntcf-arrow 𝔈 (cat-Set α) 𝔐›
    by (rule 𝔐(1))

  have 𝔊𝔑-eq-𝔑𝔉:
    𝔊(|ArrMap|)(|f|)(|ArrVal|)(|𝔑(|NTMap|)(|a|)(|ArrVal|)(|A|)|) =
      𝔑(|NTMap|)(|b|)(|ArrVal|)(|𝔉(|ArrMap|)(|f|)(|ArrVal|)(|A|)|)
    if A ∈∘ 𝔉(|ObjMap|)(|a|) for A
    using
      ArrVal-eq-helper[
        OF 𝔑.ntcf-Comp-commute[OF assms(4), symmetric], where a=A
        ]
      assms(4)
      that
    by (cs-prems cs-simp: cat-cs-simps cs-intro: cat-cs-intros)

  from 𝔐(1) assms(2,3,4) have 𝔐a-CId-a:
    𝔐(|NTMap|)(|a|)(|ArrVal|)(|𝔈(|CId|)(|a|)|) ∈∘ 𝔉(|ObjMap|)(|a|)
    by (subst 𝔐)
      (
        cs-concl
          cs-simp: cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
          cs-intro: cat-Set-cs-intros cat-cs-intros
      )

  have 𝔉f-𝔐a-eq-𝔐b:
    𝔉(|ArrMap|)(|f|)(|ArrVal|)(|𝔐(|NTMap|)(|a|)(|ArrVal|)(|h|)|) =
```

```
    𝔐(⦇NTMap⦈)(⦇b⦈)(⦇ArrVal⦈)(⦇f ∘_A ℭ h⦈)
  if h : a ↦_ℭ a for h
  using
    ArrVal-eq-helper[
      OF 𝔐.ntcf-Comp-commute[OF assms(4), symmetric], where a=h
      ]
    that
    assms(4)
    category-axioms
  by
    (
      cs-prems cs-shallow
        cs-simp:
          cat-FUNCT-cs-simps
          cat-map-extra-cs-simps
          cat-cs-simps
          cat-op-simps
        cs-intro: cat-cs-intros cat-prod-cs-intros cat-op-intros
    )

from 𝔐(1) assms(2,3,4) 𝔐a-CId-a category-axioms show
  (ntcf-Yoneda-arrow α ℭ (cf-map 𝔊) b ∘_A cat-Set β ?hom)(⦇ArrVal⦈)(⦇𝔐⦈) =
    (
      cf-eval-arrow ℭ (ntcf-arrow 𝔑) f ∘_A cat-Set β
      ntcf-Yoneda-arrow α ℭ (cf-map 𝔉) a
    )(⦇ArrVal⦈)(⦇𝔐⦈)
  by (subst (1 2) 𝔐(2))
    (
      cs-concl
        cs-simp:
          𝔉f-𝔐a-eq-𝔐b 𝔊𝔑-eq-𝔑𝔉
          cat-map-extra-cs-simps
          cat-FUNCT-cs-simps
          cat-cs-simps
          cat-op-simps
          cat-Set-components(1)
        cs-intro:
          cat-Set-αβ.subcat-is-arrD
          cat-small-cs-intros
          cat-cs-intros
          cat-FUNCT-cs-intros
          cat-prod-cs-intros
          cat-op-intros
    )

  qed (use arr-Set-𝔊b-𝔑f arr-Set-𝔑f-𝔉a in auto)

qed (use 𝔊b-𝔑f 𝔑f-𝔉a in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+

qed
```

29.17 Yoneda Lemma: naturality

29.17.1 The Yoneda natural transformation: definition and elementary properties

The main result of this subsection corresponds to the corollary to the Yoneda Lemma on page 61 in [7].

definition *ntcf-Yoneda* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntcf-Yoneda* α β $\mathfrak{C}$ =
    [
      (
        $\lambda\mathfrak{F} r\in_\circ$(*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α) $\times_C$ $\mathfrak{C}$)(⌊*Obj*⌋).
          ntcf-Yoneda-arrow α $\mathfrak{C}$ ($\mathfrak{F} r$(⌊*0*⌋)) ($\mathfrak{F} r$(⌊$1_{\mathbb{N}}$⌋))
      ),
      cf-nt α β (*cf-id* $\mathfrak{C}$),
      cf-eval α β $\mathfrak{C}$,
      cat-FUNCT α $\mathfrak{C}$ (*cat-Set* α) $\times_C$ $\mathfrak{C}$,
      cat-Set β
    ]$_\circ$

Components.

lemma *ntcf-Yoneda-components*:
  shows *ntcf-Yoneda* α β $\mathfrak{C}$(⌊*NTMap*⌋) =
    (
      $\lambda\mathfrak{F} r\in_\circ$(*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α) $\times_C$ $\mathfrak{C}$)(⌊*Obj*⌋).
        ntcf-Yoneda-arrow α $\mathfrak{C}$ ($\mathfrak{F} r$(⌊*0*⌋)) ($\mathfrak{F} r$(⌊$1_{\mathbb{N}}$⌋))
)
    and [*cat-cs-simps*]: *ntcf-Yoneda* α β $\mathfrak{C}$(⌊*NTDom*⌋) = *cf-nt* α β (*cf-id* $\mathfrak{C}$)
    and [*cat-cs-simps*]: *ntcf-Yoneda* α β $\mathfrak{C}$(⌊*NTCod*⌋) = *cf-eval* α β $\mathfrak{C}$
    and [*cat-cs-simps*]:
      ntcf-Yoneda α β $\mathfrak{C}$(⌊*NTDGDom*⌋) = *cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α) $\times_C$ $\mathfrak{C}$
    and [*cat-cs-simps*]: *ntcf-Yoneda* α β $\mathfrak{C}$(⌊*NTDGCod*⌋) = *cat-Set* β
  unfolding *ntcf-Yoneda-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

29.17.2 Natural transformation map

mk-VLambda *ntcf-Yoneda-components*(*1*)
  |*vsv ntcf-Yoneda-NTMap-vsv*[*cat-cs-intros*]|
  |*vdomain ntcf-Yoneda-NTMap-vdomain*[*cat-cs-intros*]|

lemma (**in** *category*) *ntcf-Yoneda-NTMap-app*[*cat-cs-simps*]:
  assumes $\mathcal{Z}$ β
    and α $\in_\circ$ β
    and $\mathfrak{F} r$ = [*cf-map* $\mathfrak{F}$, r]$_\circ$
    and $\mathfrak{F}$: $\mathfrak{C}$ $\mapsto\!\!\!\mapsto_{C\alpha}$ *cat-Set* α
    and r $\in_\circ$ $\mathfrak{C}$(⌊*Obj*⌋)
  shows *ntcf-Yoneda* α β $\mathfrak{C}$(⌊*NTMap*⌋)(⌊$\mathfrak{F} r$⌋) = *ntcf-Yoneda-arrow* α $\mathfrak{C}$ (*cf-map* $\mathfrak{F}$) r
proof–
  interpret β: $\mathcal{Z}$ β **by** (*rule assms*(*1*))
  interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{C}$ ‹*cat-Set* α› $\mathfrak{F}$ **by** (*rule assms*(*4*))
  interpret $\beta\mathfrak{C}$: *category* β $\mathfrak{C}$
    by (*rule category.cat-category-if-ge-Limit*)
      (*use assms*(*2*) **in** ‹*cs-concl cs-shallow cs-intro*: *cat-cs-intros*›)+
  from *assms*(*2*) **interpret** *FUNCT*: *category* β ‹*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α)›
    by
      (
        cs-concl **cs-shallow**
          cs-intro: *cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros*
)
  from *assms*(*5*) **have** [*cf-map* $\mathfrak{F}$, r]$_\circ$ $\in_\circ$ (*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α) $\times_C$ $\mathfrak{C}$)(⌊*Obj*⌋)
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-FUNCT-cs-simps*
          cs-intro: *cat-cs-intros cat-prod-cs-intros cat-FUNCT-cs-intros*

```
    )
  then show ?thesis
    unfolding assms(3) ntcf-Yoneda-components by (simp add: nat-omega-simps)
qed

lemmas [cat-cs-simps] = category.ntcf-Yoneda-NTMap-app
```

29.17.3 The Yoneda natural transformation is a natural transformation

```
lemma (in category) cat-ntcf-Yoneda-is-ntcf:
  assumes 𝒵 β and α ∈₀ β
  shows ntcf-Yoneda α β 𝔈 :
    cf-nt α β (cf-id 𝔈) ↦_{CF.iso} cf-eval α β 𝔈 :
    cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈 ↦↦_{Cβ} cat-Set β
proof–

  interpret β: 𝒵 β by (rule assms(1))
  interpret β𝔈: category β 𝔈
    by (rule category.cat-category-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: cat-cs-intros›)+
  from assms(2) interpret FUNCT: category β ‹cat-FUNCT α 𝔈 (cat-Set α)›
    by
      (
        cs-concl cs-shallow
          cs-intro: cat-cs-intros cat-small-cs-intros cat-FUNCT-cs-intros
      )

  show ?thesis
  proof(intro is-iso-ntcfI is-ntcfI')
    show vfsequence (ntcf-Yoneda α β 𝔈) unfolding ntcf-Yoneda-def by simp
    show vcard (ntcf-Yoneda α β 𝔈) = 5_ℕ
      unfolding ntcf-Yoneda-def by (simp add: nat-omega-simps)
    show ntcf-Yoneda-𝔉r: ntcf-Yoneda α β 𝔈(|NTMap|)(|𝔉r|) :
      cf-nt α β (cf-id 𝔈)(|ObjMap|)(|𝔉r|) ↦_{iso cat-Set β} cf-eval α β 𝔈(|ObjMap|)(|𝔉r|)
      if 𝔉r ∈₀ (cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈)(|Obj|) for 𝔉r
    proof–
      from that obtain 𝔉 r
        where 𝔉r-def: 𝔉r = [𝔉, r]₀
          and 𝔉: 𝔉 ∈₀ cf-maps α 𝔈 (cat-Set α)
          and r: r ∈₀ 𝔈(|Obj|)
        by
          (
            auto
              elim: cat-prod-2-ObjE[rotated 2]
              simp: cat-FUNCT-cs-simps
              intro: cat-cs-intros
          )
      from 𝔉 obtain 𝔊
        where 𝔉-def: 𝔉 = cf-map 𝔊 and 𝔊: 𝔊 : 𝔈 ↦↦_{Cα} cat-Set α
        by clarsimp
      from assms(2) 𝔊 r show ?thesis
        unfolding 𝔉r-def 𝔉-def
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-arrow-cs-intros
          )
    qed
```

```
    show ntcf-Yoneda α β 𝔈(|NTMap|)(|𝔉r|) :
      cf-nt α β (cf-id 𝔈)(|ObjMap|)(|𝔉r|) ↦_cat-Set β cf-eval α β 𝔈(|ObjMap|)(|𝔉r|)
      if 𝔉r ∈₀ (cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈)(|Obj|) for 𝔉r
      by (rule is-iso-arrD[OF ntcf-Yoneda-𝔉r[OF that]])
    show
      ntcf-Yoneda α β 𝔈(|NTMap|)(|𝔊b|) ∘_A cat-Set β
        cf-nt α β (cf-id 𝔈)(|ArrMap|)(|𝔑f|) =
          cf-eval α β 𝔈(|ArrMap|)(|𝔑f|) ∘_A cat-Set β
            ntcf-Yoneda α β 𝔈(|NTMap|)(|𝔉a|)
      if 𝔑f: 𝔑f : 𝔉a ↦_cat-FUNCT α 𝔈 (cat-Set α) ×_C 𝔈 𝔊b for 𝔉a 𝔊b 𝔑f
    proof–
      obtain 𝔑 f 𝔉 a 𝔊 b
        where 𝔑f-def: 𝔑f = [𝔑, f]₀
          and 𝔉a-def: 𝔉a = [𝔉, a]₀
          and 𝔊b-def: 𝔊b = [𝔊, b]₀
          and 𝔑: 𝔑 : 𝔉 ↦_cat-FUNCT α 𝔈 (cat-Set α) 𝔊
          and f: f : a ↦_𝔈 b
        by
          (
            auto intro:
              cat-prod-2-is-arrE[rotated 2, OF 𝔑f]
              FUNCT.category-axioms
              β𝔈.category-axioms
          )
      note 𝔑 = cat-FUNCT-is-arrD[OF 𝔑]
      note [cat-cs-simps] =
        cat-ntcf-Yoneda-arrow-commute[OF assms 𝔑(1) f, folded 𝔑(2,3,4)]
      from 𝔑(1) assms(2) f show ?thesis
        unfolding 𝔑f-def 𝔉a-def 𝔊b-def
        by (subst (1 2) 𝔑(2), use nothing in ‹subst 𝔑(3), subst 𝔑(4)›)
          (
            cs-concl
              cs-simp: 𝔑(2,3,4)[symmetric] cat-cs-simps cs-intro: cat-cs-intros
          )+
    qed

  qed (use assms(2) in ‹cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros›)+

qed
```

29.18 *Hom*-map

This subsection presents some of the results stated as Corollary 2 in subsection 1.15 in [3] and the corollary following the statement of the Yoneda Lemma on page 61 in [7] in a variety of forms.

29.18.1 Definition and elementary properties

The following function makes an explicit appearance in subsection 1.15 in [3].

```
definition ntcf-Hom-map :: V ⇒ V ⇒ V ⇒ V ⇒ V
  where ntcf-Hom-map α 𝔈 a b = (λf∈₀Hom 𝔈 a b. Hom_A.Cα𝔈(f,−))
```

Elementary properties.

```
mk-VLambda ntcf-Hom-map-def
  |vsv ntcf-Hom-map-vsv|
  |vdomain ntcf-Hom-map-vdomain[cat-cs-simps]|
```

|*app ntcf-Hom-map-app*[*unfolded in-Hom-iff*, *cat-cs-simps*]|

29.18.2 *Hom*-map is a bijection

lemma (in *category*) *cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique*:
— The following lemma approximately corresponds to the corollary on page 61 in [7].
assumes $r \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $s \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $\mathfrak{N} : Hom_{O.C\alpha}\mathfrak{C}(r,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(s,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
shows *Yoneda-map* $\alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r(\!|\mathfrak{N}|\!) : s \mapsto_{\mathfrak{C}} r$
and $\mathfrak{N} = Hom_{A.C\alpha}\mathfrak{C}(Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r(\!|\mathfrak{N}|\!),-)$
and $\bigwedge f.\ [\![\ f \in_{\circ} \mathfrak{C}(\!|Arr|\!);\ \mathfrak{N} = Hom_{A.C\alpha}\mathfrak{C}(f,-)\ ]\!] \Longrightarrow$
$f = Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r(\!|\mathfrak{N}|\!)$
proof–

interpret $\mathfrak{N}$: *is-ntcf* $\alpha\ \mathfrak{C}\ \langle cat\text{-}Set\ \alpha\rangle\ \langle Hom_{O.C\alpha}\mathfrak{C}(r,-)\rangle\ \langle Hom_{O.C\alpha}\mathfrak{C}(s,-)\rangle\ \mathfrak{N}$
by (*rule assms*(*3*))
let *?Y-Hom-s* = $\langle Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r\rangle$
note *Yoneda* =
cat-Yoneda-Lemma[*OF cat-cf-Hom-snd-is-functor*[*OF assms*(*2*)] *assms*(*1*)]
interpret *Y*: *v11* $\langle$*?Y-Hom-s*$\rangle$ **by** (*rule Yoneda*(*1*))

from *category-axioms assms* **have** $\mathfrak{N}$*-in-vdomain*: $\mathfrak{N} \in_{\circ} \mathcal{D}_{\circ}$ (*?Y-Hom-s*)
by (*cs-concl* **cs-shallow cs-simp**: *these-ntcfs-iff cat-cs-simps* **cs-intro**: *cat-cs-intros*)
then have *?Y-Hom-s*$(\!|\mathfrak{N}|\!) \in_{\circ} \mathcal{R}_{\circ}$ (*?Y-Hom-s*) **by** (*simp add*: *Y.vsv-vimageI2*)
from *this category-axioms assms* **show** *Ym-*$\mathfrak{N}$: *?Y-Hom-s*$(\!|\mathfrak{N}|\!) : s \mapsto_{\mathfrak{C}} r$
unfolding *Yoneda*(*2*)
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps cat-op-simps*)
then have *?Y-Hom-s*$(\!|\mathfrak{N}|\!) \in_{\circ} \mathfrak{C}(\!|Arr|\!)$ **by** (*simp add*: *cat-cs-intros*)

have $Hom_{A.C\alpha}\mathfrak{C}($*?Y-Hom-s*$(\!|\mathfrak{N}|\!),-) :$
$Hom_{O.C\alpha}\mathfrak{C}(r,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(s,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*intro cat-ntcf-Hom-snd-is-ntcf Ym-*$\mathfrak{N}$)

from *assms Ym-*$\mathfrak{N}$ *this category-axioms assms* **have**
$(?Y\text{-}Hom\text{-}s)^{-1}{}_{\circ}(\!|?Y\text{-}Hom\text{-}s(\!|\mathfrak{N}|\!)|\!) =$
Yoneda-arrow $\alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r\ (?Y\text{-}Hom\text{-}s(\!|\mathfrak{N}|\!))$
by (*intro category.inv-Yoneda-map-app*)
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)
then have $(?Y\text{-}Hom\text{-}s)^{-1}{}_{\circ}(\!|?Y\text{-}Hom\text{-}s(\!|\mathfrak{N}|\!)|\!) = Hom_{A.C\alpha}\mathfrak{C}($*?Y-Hom-s*$(\!|\mathfrak{N}|\!),-)$
by (*simp add*: *ntcf-Hom-snd-def'*[*OF Ym-*$\mathfrak{N}$])
with $\mathfrak{N}$*-in-vdomain* **show** $\mathfrak{N} = Hom_{A.C\alpha}\mathfrak{C}($*?Y-Hom-s*$(\!|\mathfrak{N}|\!),-)$ **by** *auto*

fix *f* **assume** *prems*: $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)\ \mathfrak{N} = Hom_{A.C\alpha}\mathfrak{C}(f,-)$
then obtain *a b* **where** *f*: $f : a \mapsto_{\mathfrak{C}} b$ **by** *auto*

have $\mathfrak{N} : Hom_{O.C\alpha}\mathfrak{C}(b,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(a,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*rule cat-ntcf-Hom-snd-is-ntcf*[*OF f*, *folded prems*(*2*)])
with *f* $\mathfrak{N}$*.ntcf-NTDom* $\mathfrak{N}$*.ntcf-NTCod assms cat-is-arrD*(*2*,*3*)[*OF f*]
have *ba-simps*: $b = r\ a = s$
by
(
simp-all add:
prems(*2*) *cat-cf-Hom-snd-inj cat-ntcf-Hom-snd-components*(*2*,*3*)
)

```
  from f have f : s ↦_𝔈 r unfolding ba-simps .

  with category-axioms show f = ?Y-Hom-s(|𝔑|)
    unfolding prems(2)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cat-op-simps)

qed

lemma (in category) cat-ntcf-Hom-fst-is-ntcf-Hom-fst-unique:
  assumes r ∈₀ 𝔈(|Obj|)
    and s ∈₀ 𝔈(|Obj|)
    and 𝔑 : Hom_{O.Cα}𝔈(−,r) ↦_CF Hom_{O.Cα}𝔈(−,s) : op-cat 𝔈 ↦↦_Cα cat-Set α
  shows Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|𝔑|) : r ↦_𝔈 s
    and 𝔑 = Hom_{A.Cα}𝔈(−,Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|𝔑|))
    and ⋀f. [[ f ∈₀ 𝔈(|Arr|); 𝔑 = Hom_{A.Cα}𝔈(−,f) ]] ⟹
      f = Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|𝔑|)
  by
    (
      intro
        category.cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique[
          OF category-op,
          unfolded cat-op-simps cat-op-cat-ntcf-Hom-snd,
          OF assms(1,2),
          unfolded assms(1,2)[THEN cat-op-cat-cf-Hom-snd],
          OF assms(3)
          ]
    )+

lemma (in category) cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique′:
  assumes r ∈₀ 𝔈(|Obj|)
    and s ∈₀ 𝔈(|Obj|)
    and 𝔑 : Hom_{O.Cα}𝔈(r,−) ↦_CF Hom_{O.Cα}𝔈(s,−) : 𝔈 ↦↦_Cα cat-Set α
  shows ∃!f. f ∈₀ 𝔈(|Arr|) ∧ 𝔑 = Hom_{A.Cα}𝔈(f,−)
  using cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique[OF assms] by blast

lemma (in category) cat-ntcf-Hom-fst-is-ntcf-Hom-fst-unique′:
  assumes r ∈₀ 𝔈(|Obj|)
    and s ∈₀ 𝔈(|Obj|)
    and 𝔑 : Hom_{O.Cα}𝔈(−,r) ↦_CF Hom_{O.Cα}𝔈(−,s) : op-cat 𝔈 ↦↦_Cα cat-Set α
  shows ∃!f. f ∈₀ 𝔈(|Arr|) ∧ 𝔑 = Hom_{A.Cα}𝔈(−,f)
  using cat-ntcf-Hom-fst-is-ntcf-Hom-fst-unique[OF assms] by blast

lemma (in category) cat-ntcf-Hom-snd-inj:
  assumes Hom_{A.Cα}𝔈(g,−) = Hom_{A.Cα}𝔈(f,−)
    and g : a ↦_𝔈 b
    and f : a ↦_𝔈 b
  shows g = f
proof–
  from assms have
    Yoneda-map α (Hom_{O.Cα}𝔈(a,−)) b(|Hom_{A.Cα}𝔈(g,−)|) =
      Yoneda-map α (Hom_{O.Cα}𝔈(a,−)) b(|Hom_{A.Cα}𝔈(f,−)|)
    by simp
  from this assms category-axioms show g = f
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
      )
```

```
    simp
qed

lemma (in category) cat-ntcf-Hom-fst-inj:
  assumes Hom_{A.CA}𝔈(−,g) = Hom_{A.CA}𝔈(−,f)
    and g : a ↦_𝔈 b
    and f : a ↦_𝔈 b
  shows g = f
proof–
  from category.cat-ntcf-Hom-snd-inj
    [
      OF category-op,
      unfolded cat-op-simps,
      unfolded cat-op-cat-ntcf-Hom-snd,
      OF assms
    ]
  show ?thesis .
qed

lemma (in category) cat-ntcf-Hom-map:
  assumes a ∈₀ 𝔈⦇Obj⦈ and b ∈₀ 𝔈⦇Obj⦈
  shows v11 (ntcf-Hom-map α 𝔈 a b)
    and ℛ₀ (ntcf-Hom-map α 𝔈 a b) =
      these-ntcfs α 𝔈 (cat-Set α) Hom_{O.CA}𝔈(b,−) Hom_{O.CA}𝔈(a,−)
    and (ntcf-Hom-map α 𝔈 a b)⁻¹₀ =
      (λ𝔑∈₀these-ntcfs α 𝔈 (cat-Set α) Hom_{O.CA}𝔈(b,−) Hom_{O.CA}𝔈(a,−).
        Yoneda-map α Hom_{O.CA}𝔈(a,−) b⦇𝔑⦈)
proof–

  show v11 (ntcf-Hom-map α 𝔈 a b)
  proof(rule vsv.vsv-valeq-v11I, unfold ntcf-Hom-map-vdomain in-Hom-iff)
    show vsv (ntcf-Hom-map α 𝔈 a b) unfolding ntcf-Hom-map-def by simp
    fix g f assume prems:
      g : a ↦_𝔈 b
      f : a ↦_𝔈 b
      ntcf-Hom-map α 𝔈 a b⦇g⦈ = ntcf-Hom-map α 𝔈 a b⦇f⦈
    from prems(3,1,2) have Hom_{A.CA}𝔈(g,−) = Hom_{A.CA}𝔈(f,−)
      by (cs-prems cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    with prems(1,2) show g = f by (intro cat-ntcf-Hom-snd-inj[of g f])
  qed
  then interpret Hm: v11 ‹ntcf-Hom-map α 𝔈 a b› .

  show Hm-vrange: ℛ₀ (ntcf-Hom-map α 𝔈 a b) =
    these-ntcfs α 𝔈 (cat-Set α) Hom_{O.CA}𝔈(b,−) Hom_{O.CA}𝔈(a,−)
  proof(intro vsubset-antisym)
    show ℛ₀ (ntcf-Hom-map α 𝔈 a b) ⊆₀
      these-ntcfs α 𝔈 (cat-Set α) Hom_{O.CA}𝔈(b,−) Hom_{O.CA}𝔈(a,−)
      by
        (
          unfold ntcf-Hom-map-def,
          intro vrange-VLambda-vsubset,
          unfold these-ntcfs-iff in-Hom-iff,
          intro cat-ntcf-Hom-snd-is-ntcf
        )
    show these-ntcfs α 𝔈 (cat-Set α) Hom_{O.CA}𝔈(b,−) Hom_{O.CA}𝔈(a,−) ⊆₀
      ℛ₀ (ntcf-Hom-map α 𝔈 a b)
    proof(intro vsubsetI, unfold these-ntcfs-iff)
      fix 𝔑 assume prems:
```

```
    𝔑 : Hom_{O.Cα}𝔈(b,–) ↦_{CF} Hom_{O.Cα}𝔈(a,–) : 𝔈 ↦↦_{Cα} cat-Set α
   note unique =
    cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique[OF assms(2,1) prems]
   from unique(1) have
    Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|) ∈_∘ 𝒟_∘ (ntcf-Hom-map α 𝔈 a b)
    by (cs-concl cs-simp: cat-cs-simps)
   moreover from
    cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique(1,2)[OF assms(2,1) prems]
   have 𝔑-def: 𝔑 = ntcf-Hom-map α 𝔈 a b(| Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|) |)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
   ultimately show 𝔑 ∈_∘ ℛ_∘ (ntcf-Hom-map α 𝔈 a b) by force
  qed
qed

show (ntcf-Hom-map α 𝔈 a b)^{-1}_∘ =
  (
   λ𝔑∈_∘these-ntcfs α 𝔈 (cat-Set α) Hom_{O.Cα}𝔈(b,–) Hom_{O.Cα}𝔈(a,–).
    Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|)
  )
proof
  (
   rule vsv-eqI,
   unfold vdomain-vconverse vdomain-VLambda Hm-vrange these-ntcfs-iff
  )

  from Hm.v11-axioms show vsv ((ntcf-Hom-map α 𝔈 a b)^{-1}_∘) by auto
  show vsv
   (
    λ𝔑∈_∘these-ntcfs α 𝔈 (cat-Set α) Hom_{O.Cα}𝔈(b,–) Hom_{O.Cα}𝔈(a,–).
     Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|)
   )
   by simp

  fix 𝔑 assume prems:
   𝔑 : Hom_{O.Cα}𝔈(b,–) ↦_{CF} Hom_{O.Cα}𝔈(a,–) : 𝔈 ↦↦_{Cα} cat-Set α
  then have 𝔑:
   𝔑 ∈_∘ these-ntcfs α 𝔈 (cat-Set α) Hom_{O.Cα}𝔈(b,–) Hom_{O.Cα}𝔈(a,–)
   unfolding these-ntcfs-iff by simp
  show (ntcf-Hom-map α 𝔈 a b)^{-1}_∘(|𝔑|) =
   (
    λ𝔑∈_∘these-ntcfs α 𝔈 (cat-Set α) Hom_{O.Cα}𝔈(b,–) Hom_{O.Cα}𝔈(a,–).
     Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|)
   )(|𝔑|)
  proof
   (
    intro Hm.v11-vconverse-app,
    unfold ntcf-Hom-map-vdomain in-Hom-iff beta[OF 𝔑]
   )
   note unique =
    cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique[OF assms(2,1) prems]
   show Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|) : a ↦_𝔈 b by (rule unique(1))
   then show
    ntcf-Hom-map α 𝔈 a b(| Yoneda-map α Hom_{O.Cα}𝔈(a,–) b(|𝔑|) |) = 𝔑
    by (cs-concl cs-simp: unique(2)[symmetric] cat-cs-simps)
  qed

qed simp
```

qed

29.18.3 Inverse of a *Hom*-map

lemma (**in** *category*) *inv-ntcf-Hom-map-v11*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows *v11* $((ntcf\text{-}Hom\text{-}map\ \alpha\ \mathfrak{C}\ a\ b)^{-1}{}_{\circ})$
using *cat-ntcf-Hom-map*(*1*)[*OF assms*] **by** (*simp add*: *v11.v11-vconverse*)

lemma (**in** *category*) *inv-ntcf-Hom-map-vdomain*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $\mathcal{D}_{\circ}\ ((ntcf\text{-}Hom\text{-}map\ \alpha\ \mathfrak{C}\ a\ b)^{-1}{}_{\circ}) =$
these-ntcfs $\alpha\ \mathfrak{C}\ (cat\text{-}Set\ \alpha)\ Hom_{O.C\alpha}\mathfrak{C}(b,-)\ Hom_{O.C\alpha}\mathfrak{C}(a,-)$
unfolding *cat-ntcf-Hom-map*(*3*)[*OF assms*] **by** *simp*

lemmas [*cat-cs-simps*] = *category.inv-ntcf-Hom-map-vdomain*

lemma (**in** *category*) *inv-ntcf-Hom-map-app*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $\mathfrak{N} : Hom_{O.C\alpha}\mathfrak{C}(b,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(a,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
shows $(ntcf\text{-}Hom\text{-}map\ \alpha\ \mathfrak{C}\ a\ b)^{-1}{}_{\circ}(\!|\mathfrak{N}|\!) = Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(a,-)\ b(\!|\mathfrak{N}|\!)$
using *assms*(*3*) **unfolding** *cat-ntcf-Hom-map*(*3*)[*OF assms*(*1,2*)] **by** *simp*

lemmas [*cat-cs-simps*] = *category.inv-ntcf-Hom-map-app*

lemma *inv-ntcf-Hom-map-vrange*: $\mathcal{R}_{\circ}\ ((ntcf\text{-}Hom\text{-}map\ \alpha\ \mathfrak{C}\ a\ b)^{-1}{}_{\circ}) = Hom\ \mathfrak{C}\ a\ b$
unfolding *ntcf-Hom-map-def* **by** *simp*

29.18.4 *Hom*-natural transformation and isomorphisms

This subsection presents further results that were stated as Corollary 2 in subsection 1.15 in [3].

lemma (**in** *category*) *cat-is-iso-arr-ntcf-Hom-snd-is-iso-ntcf*:
assumes $f : s \mapsto_{iso\mathfrak{C}} r$
shows $Hom_{A.C\alpha}\mathfrak{C}(f,-) :$
$Hom_{O.C\alpha}\mathfrak{C}(r,-) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{C}(s,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
proof–
from *assms* **obtain** *g*
where *iso-g*: $g : r \mapsto_{iso\mathfrak{C}} s$
and *gf*: $g \circ_{A\mathfrak{C}} f = \mathfrak{C}(\!|CId|\!)(\!|s|\!)$
and *fg*: $f \circ_{A\mathfrak{C}} g = \mathfrak{C}(\!|CId|\!)(\!|r|\!)$
by
(
auto intro:
cat-the-inverse-Comp-CId-left
cat-the-inverse-Comp-CId-right
cat-the-inverse-is-iso-arr'
)
then have *g*: $g : r \mapsto_{\mathfrak{C}} s$ **by** *auto*
show *?thesis*
proof(*intro is-iso-arr-is-iso-ntcf*)
from *assms* **have** *f*: $f : s \mapsto_{\mathfrak{C}} r$ **by** *auto*
with *category-axioms* **show** $Hom_{A.C\alpha}\mathfrak{C}(f,-) :$
$Hom_{O.C\alpha}\mathfrak{C}(r,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(s,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *category-axioms g* **show** $Hom_{A.C\alpha}\mathfrak{C}(g,-) :$
$Hom_{O.C\alpha}\mathfrak{C}(s,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(r,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$

```
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  from category-axioms f g have
    Hom_{A.C α}𝔈(f,–) ∙_{NTCF} Hom_{A.C α}𝔈(g,–) = Hom_{A.C α}𝔈(g ∘_{A𝔈} f,–)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  also from category-axioms f g have ... = ntcf-id Hom_{O.C α}𝔈(s,–)
    by (cs-concl cs-simp: gf cat-cs-simps cs-intro: cat-cs-intros)
  finally show
    Hom_{A.C α}𝔈(f,–) ∙_{NTCF} Hom_{A.C α}𝔈(g,–) = ntcf-id Hom_{O.C α}𝔈(s,–)
    by simp
  from category-axioms f g have
    Hom_{A.C α}𝔈(g,–) ∙_{NTCF} Hom_{A.C α}𝔈(f,–) = Hom_{A.C α}𝔈(f ∘_{A𝔈} g,–)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  also from category-axioms f g have ... = ntcf-id Hom_{O.C α}𝔈(r,–)
    by (cs-concl cs-simp: fg cat-cs-simps cs-intro: cat-cs-intros)
  finally show
    Hom_{A.C α}𝔈(g,–) ∙_{NTCF} Hom_{A.C α}𝔈(f,–) = ntcf-id Hom_{O.C α}𝔈(r,–)
    by simp
 qed
qed

lemma (in category) cat-is-iso-arr-ntcf-Hom-fst-is-iso-ntcf:
 assumes f : r ↦_{iso 𝔈} s
 shows Hom_{A.C α}𝔈(–,f) :
   Hom_{O.C α}𝔈(–,r) ↦_{CF.iso} Hom_{O.C α}𝔈(–,s) : op-cat 𝔈 ↦↦_{C α} cat-Set α
proof–
 from assms have r: r ∈_∘ 𝔈(|Obj|) and s: s ∈_∘ 𝔈(|Obj|) by auto
 from
   category.cat-is-iso-arr-ntcf-Hom-snd-is-iso-ntcf
     [
       OF category-op,
       unfolded cat-op-simps,
       OF assms,
       unfolded
         category.cat-op-cat-cf-Hom-snd[OF category-axioms r]
         category.cat-op-cat-cf-Hom-snd[OF category-axioms s]
         category.cat-op-cat-ntcf-Hom-snd[OF category-axioms]
     ]
 show ?thesis.
qed

lemma (in category) cat-ntcf-Hom-snd-is-iso-ntcf-Hom-snd-unique:
 assumes r ∈_∘ 𝔈(|Obj|)
   and s ∈_∘ 𝔈(|Obj|)
   and 𝔑 : Hom_{O.C α}𝔈(r,–) ↦_{CF.iso} Hom_{O.C α}𝔈(s,–) : 𝔈 ↦↦_{C α} cat-Set α
 shows Yoneda-map α Hom_{O.C α}𝔈(s,–) r(|𝔑|) : s ↦_{iso 𝔈} r
   and 𝔑 = Hom_{A.C α}𝔈( Yoneda-map α Hom_{O.C α}𝔈(s,–) r(|𝔑|),–)
   and ⋀f. [[ f ∈_∘ 𝔈(|Arr|); 𝔑 = Hom_{A.C α}𝔈(f,–) ]] ⟹
     f = Yoneda-map α Hom_{O.C α}𝔈(s,–) r(|𝔑|)
proof–

 let ?Ym-𝔑 = ‹Yoneda-map α Hom_{O.C α}𝔈(s,–) r(|𝔑|)›
   and ?Ym-inv-𝔑 = ‹Yoneda-map α Hom_{O.C α}𝔈(r,–) s(|inv-ntcf 𝔑|)›

 from assms(3) have 𝔑:
   𝔑 : Hom_{O.C α}𝔈(r,–) ↦_{CF} Hom_{O.C α}𝔈(s,–) : 𝔈 ↦↦_{C α} cat-Set α
   by auto
 from iso-ntcf-is-iso-arr[OF assms(3)]
 have iso-inv-𝔑: inv-ntcf 𝔑 :
```

$Hom_{O.C\alpha}\mathfrak{C}(s,-) \mapsto_{CF.iso} Hom_{O.C\alpha}\mathfrak{C}(r,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
and [*simp*]: $\mathfrak{N} \bullet_{NTCF} inv\text{-}ntcf\ \mathfrak{N} = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(s,-)$
and [*simp*]: $inv\text{-}ntcf\ \mathfrak{N} \bullet_{NTCF} \mathfrak{N} = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(r,-)$
by *auto*
from *iso-inv-*$\mathfrak{N}$ **have** *inv-*$\mathfrak{N}$:
$inv\text{-}ntcf\ \mathfrak{N} : Hom_{O.C\alpha}\mathfrak{C}(s,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(r,-) : \mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by *auto*
note *unique* = *cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique*[*OF assms*(*1,2*) $\mathfrak{N}$]
and *inv-unique* =
cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique[*OF assms*(*2,1*) *inv-*$\mathfrak{N}$]
have *Ym-*$\mathfrak{N}$: *?Ym-*$\mathfrak{N} : s \mapsto_{\mathfrak{C}} r$ **by** (*rule unique*(*1*))

show $\mathfrak{N} = Hom_{A.C\alpha}\mathfrak{C}(Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r(\!|\mathfrak{N}|\!),-)$
and $\bigwedge f.\ [\![f \in_{\circ} \mathfrak{C}(\!|Arr|\!);\ \mathfrak{N} = Hom_{A.C\alpha}\mathfrak{C}(f,-)]\!] \Longrightarrow$
$f = Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r(\!|\mathfrak{N}|\!)$
by (*intro unique*)+

show $Yoneda\text{-}map\ \alpha\ Hom_{O.C\alpha}\mathfrak{C}(s,-)\ r(\!|\mathfrak{N}|\!) : s \mapsto_{iso\mathfrak{C}} r$
proof(*intro is-iso-arrI*[*OF Ym-*$\mathfrak{N}$, *of* ‹*?Ym-inv-*$\mathfrak{N}$›] *is-inverseI*)

show *Ym-inv-*$\mathfrak{N}$: *?Ym-inv-*$\mathfrak{N} : r \mapsto_{\mathfrak{C}} s$ **by** (*rule inv-unique*(*1*))

have $ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(s,-) = \mathfrak{N} \bullet_{NTCF} inv\text{-}ntcf\ \mathfrak{N}$ **by** *simp*
also have $\ldots = Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}\mathfrak{N},-) \bullet_{NTCF} Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}inv\text{-}\mathfrak{N},-)$
by (*subst unique*(*2*), *subst inv-unique*(*2*)) *simp*
also from *category-axioms Ym-*$\mathfrak{N}$ *inv-unique*(*1*) *assms*(*3*) **have**
$\ldots = Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}inv\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}\mathfrak{N},-)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
finally have $Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}inv\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}\mathfrak{N},-) = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(s,-)$
by *simp*
also from *category-axioms assms*(*1,2*) **have** $\ldots = Hom_{A.C\alpha}\mathfrak{C}(\mathfrak{C}(\!|CId|\!)(\!|s|\!),-)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
finally have $Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}inv\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}\mathfrak{N},-) = Hom_{A.C\alpha}\mathfrak{C}(\mathfrak{C}(\!|CId|\!)(\!|s|\!),-)$
by *simp*
then show $?Ym\text{-}inv\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}\mathfrak{N} = \mathfrak{C}(\!|CId|\!)(\!|s|\!)$
by (*rule cat-ntcf-Hom-snd-inj*)
(
all‹
use category-axioms Ym-$\mathfrak{N}$ *Ym-inv-*$\mathfrak{N}$ *assms in*
‹*cs-concl cs-shallow cs-simp*: *cat-cs-simps cs-intro*: *cat-cs-intros*›
›
)
have $ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(r,-) = inv\text{-}ntcf\ \mathfrak{N} \bullet_{NTCF} \mathfrak{N}$ **by** *simp*
also have $\ldots = Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}inv\text{-}\mathfrak{N},-) \bullet_{NTCF} Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}\mathfrak{N},-)$
by (*subst unique*(*2*), *subst inv-unique*(*2*)) *simp*
also from *category-axioms Ym-*$\mathfrak{N}$ *inv-unique*(*1*) **have**
$\ldots = Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}inv\text{-}\mathfrak{N},-)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
finally have
$Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}inv\text{-}\mathfrak{N},-) = ntcf\text{-}id\ Hom_{O.C\alpha}\mathfrak{C}(r,-)$
by *simp*
also from *category-axioms assms*(*1,2*) **have** $\ldots = Hom_{A.C\alpha}\mathfrak{C}(\mathfrak{C}(\!|CId|\!)(\!|r|\!),-)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
finally have
$Hom_{A.C\alpha}\mathfrak{C}(?Ym\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}inv\text{-}\mathfrak{N},-) = Hom_{A.C\alpha}\mathfrak{C}(\mathfrak{C}(\!|CId|\!)(\!|r|\!),-)$
by *simp*
then show $?Ym\text{-}\mathfrak{N} \circ_{A\mathfrak{C}} ?Ym\text{-}inv\text{-}\mathfrak{N} = \mathfrak{C}(\!|CId|\!)(\!|r|\!)$
by (*rule cat-ntcf-Hom-snd-inj*)

```
    (
      all‹
        use category-axioms Ym-𝔑 Ym-inv-𝔑 assms in
          ‹cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros›
        ›
    )

  qed (intro Ym-𝔑)

qed

lemma (in category) cat-ntcf-Hom-fst-is-iso-ntcf-Hom-fst-unique:
  assumes r ∈₀ 𝔈(|Obj|)
    and s ∈₀ 𝔈(|Obj|)
    and 𝔑 :
    Hom_{O.Cα}𝔈(−,r) ↦_{CF.iso} Hom_{O.Cα}𝔈(−,s) : op-cat 𝔈 ↦↦_{Cα} cat-Set α
  shows Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|𝔑|) : r ↦_{iso 𝔈} s
    and 𝔑 = Hom_{A.Cα}𝔈(−,Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|𝔑|))
    and ⋀f. [[ f ∈₀ 𝔈(|Arr|); 𝔑 = Hom_{A.Cα}𝔈(−,f) ]] ⟹
      f = Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|𝔑|)
  by
    (
      intro
        category.cat-ntcf-Hom-snd-is-iso-ntcf-Hom-snd-unique[
          OF category-op,
          unfolded cat-op-simps cat-op-cat-ntcf-Hom-snd,
          OF assms(1,2),
          unfolded assms(1,2)[THEN cat-op-cat-cf-Hom-snd],
          OF assms(3)
          ]
    )+

lemma (in category) cat-is-iso-arr-if-ntcf-Hom-snd-is-iso-ntcf:
  assumes f : s ↦_{𝔈} r
    and Hom_{A.Cα}𝔈(f,−) :
      Hom_{O.Cα}𝔈(r,−) ↦_{CF.iso} Hom_{O.Cα}𝔈(s,−) : 𝔈 ↦↦_{Cα} cat-Set α
  shows f : s ↦_{iso 𝔈} r
proof–
  from assms(1) have r: r ∈₀ 𝔈(|Obj|) and s: s ∈₀ 𝔈(|Obj|) by auto
  note unique = cat-ntcf-Hom-snd-is-iso-ntcf-Hom-snd-unique[OF r s assms(2)]
  from unique(1) have Ym-Hf:
    Yoneda-map α Hom_{O.Cα}𝔈(s,−) r(|Hom_{A.Cα}𝔈(f,−)|) : s ↦_{𝔈} r
    by auto
  from unique(1) show ?thesis
    unfolding cat-ntcf-Hom-snd-inj[OF unique(2) assms(1) Ym-Hf, symmetric]
    by simp
qed

lemma (in category) cat-is-iso-arr-if-ntcf-Hom-fst-is-iso-ntcf:
  assumes f : r ↦_{𝔈} s
    and Hom_{A.Cα}𝔈(−,f) :
      Hom_{O.Cα}𝔈(−,r) ↦_{CF.iso} Hom_{O.Cα}𝔈(−,s) : op-cat 𝔈 ↦↦_{Cα} cat-Set α
  shows f : r ↦_{iso 𝔈} s
proof–
  from assms(1) have r: r ∈₀ 𝔈(|Obj|) and s: s ∈₀ 𝔈(|Obj|) by auto
  note unique = cat-ntcf-Hom-fst-is-iso-ntcf-Hom-fst-unique[OF r s assms(2)]
  from unique(1) have Ym-Hf:
    Yoneda-map α Hom_{O.Cα}𝔈(−,s) r(|Hom_{A.Cα}𝔈(−,f)|) : r ↦_{𝔈} s
```

by *auto*
from *unique*(*1*) **show** *?thesis*
unfolding *cat-ntcf-Hom-fst-inj*[*OF unique*(*2*) *assms*(*1*) *Ym-Hf*, *symmetric*]
by *simp*
qed

29.18.5 The relationship between a *Hom*-natural transformation and the compositions of a *Hom*-natural transformation and a natural transformation

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-ntcf-Hom-snd-NTMap-app*:
assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
and $c \in_\circ \mathfrak{C}(\lfloor Obj \rfloor)$
shows $Hom_{A.C\alpha}(\varphi -,-)(\lfloor NTMap \rfloor)(\lfloor b,\ c \rfloor)_\bullet = Hom_{A.C\alpha}\mathfrak{C}(\varphi(\lfloor NTMap \rfloor)(\lfloor b \rfloor),-)(\lfloor NTMap \rfloor)(\lfloor c \rfloor)$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
from *assms*(*2*) **have** *b*: $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$ **unfolding** *cat-op-simps* **by** *simp*
from *category-axioms assms*(*1*,*3*) *b* **show** *?thesis*
by
(
cs-concl **cs-shallow**
cs-simp:
cat-ntcf-lcomp-Hom-component-is-Yoneda-component cat-cs-simps
cs-intro: *cat-cs-intros cat-op-intros*
)
qed

lemmas [*cat-cs-simps*] = *category.cat-ntcf-lcomp-Hom-ntcf-Hom-snd-NTMap-app*

lemma (**in** *category*) *cat-bnt-proj-snd-tcf-lcomp-Hom-ntcf-Hom-snd*:
assumes $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $b \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
shows $Hom_{A.C\alpha}(\varphi -,-)_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(b,-)_{NTCF} = Hom_{A.C\alpha}\mathfrak{C}(\varphi(\lfloor NTMap \rfloor)(\lfloor b \rfloor),-)$
proof–
interpret φ: *is-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*rule ntcf-eqI*[*of* α])
from *category-axioms assms* **show**
$Hom_{A.C\alpha}(\varphi -,-)_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(b,-)_{NTCF} :$
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor),-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor),-) :$
$\mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*
)
from *assms this* **have** *dom-lhs*:
$\mathcal{D}_\circ\ ((Hom_{A.C\alpha}(\varphi -,-)_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(b,-)_{NTCF})(\lfloor NTMap \rfloor)) = \mathfrak{C}(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *category-axioms assms* **show**
$Hom_{A.C\alpha}\mathfrak{C}(\varphi(\lfloor NTMap \rfloor)(\lfloor b \rfloor),-) :$
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor),-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor),-) :$
$\mathfrak{C} \mapsto\mapsto_{C\alpha} cat\text{-}Set\ \alpha$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms this* **have** *dom-rhs*:
$\mathcal{D}_\circ\ (Hom_{A.C\alpha}\mathfrak{C}(\varphi(\lfloor NTMap \rfloor)(\lfloor b \rfloor),-)(\lfloor NTMap \rfloor)) = \mathfrak{C}(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show

```
    (Hom_{A.Cα}(φ−,−)_{op-cat 𝔅,𝔈}(b,−)_{NTCF})(⌊NTMap⌋) =
      Hom_{A.Cα}𝔈(φ(⌊NTMap⌋)(⌊b⌋),−)(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume a ∈∘ 𝔈(⌊Obj⌋)
    with category-axioms assms show
      (Hom_{A.Cα}(φ−,−)_{op-cat 𝔅,𝔈}(b,−)_{NTCF})(⌊NTMap⌋)(⌊a⌋) =
        Hom_{A.Cα}𝔈(φ(⌊NTMap⌋)(⌊b⌋),−)(⌊NTMap⌋)(⌊a⌋)
      by (cs-concl cs-simp: cat-cs-simps)
  qed (use assms(2) in ‹auto intro: cat-cs-intros›)
 qed simp-all
qed

lemmas [cat-cs-simps] = category.cat-bnt-proj-snd-tcf-lcomp-Hom-ntcf-Hom-snd
```

29.18.6 The relationship between the *Hom*-natural isomorphisms and the compositions of a *Hom*-natural isomorphism and a natural transformation

```
lemma (in category) cat-ntcf-lcomp-Hom-if-ntcf-Hom-snd-is-iso-ntcf:
  assumes φ : 𝔉 ↦_{CF} 𝔊 : 𝔅 ↦↦_{Cα} 𝔈
    and ⋀b. b ∈∘ 𝔅(⌊Obj⌋) ⟹ Hom_{A.Cα}𝔈(φ(⌊NTMap⌋)(⌊b⌋),−) :
      Hom_{O.Cα}𝔈(𝔊(⌊ObjMap⌋)(⌊b⌋),−) ↦_{CF.iso} Hom_{O.Cα}𝔈(𝔉(⌊ObjMap⌋)(⌊b⌋),−) :
      𝔈 ↦↦_{Cα} cat-Set α
  shows Hom_{A.Cα}(φ−,−) :
    Hom_{O.Cα}𝔈(𝔊−,−) ↦_{CF.iso} Hom_{O.Cα}𝔈(𝔉−,−) :
    op-cat 𝔅 ×_C 𝔈 ↦↦_{Cα} cat-Set α
proof–
  interpret φ: is-ntcf α 𝔅 𝔈 𝔉 𝔊 φ by (rule assms(1))
  have Hom_{A.Cα}(φ−,−)_{op-cat 𝔅,𝔈}(b,−)_{NTCF} :
    Hom_{O.Cα}𝔈(𝔊−,−)_{op-cat 𝔅,𝔈}(b,−)_{CF} ↦_{CF.iso}
    Hom_{O.Cα}𝔈(𝔉−,−)_{op-cat 𝔅,𝔈}(b,−)_{CF} :
    𝔈 ↦↦_{Cα} cat-Set α
    if b ∈∘ 𝔅(⌊Obj⌋) for b
    unfolding
      cat-bnt-proj-snd-tcf-lcomp-Hom-ntcf-Hom-snd[OF assms(1) that]
      cat-cf-lcomp-Hom-cf-Hom-snd[OF φ.NTDom.is-functor-axioms that]
      cat-cf-lcomp-Hom-cf-Hom-snd[OF φ.NTCod.is-functor-axioms that]
    by (intro assms(2) that)
  from
    is-iso-ntcf-if-bnt-proj-snd-is-iso-ntcf[
      OF
        φ.NTDom.HomDom.category-op category-axioms
        cat-ntcf-lcomp-Hom-is-ntcf[OF assms(1)],
      unfolded cat-op-simps, OF this
      ]
  show ?thesis .
qed

lemma (in category) cat-ntcf-Hom-snd-if-ntcf-lcomp-Hom-is-iso-ntcf:
  assumes φ : 𝔉 ↦_{CF} 𝔊 : 𝔅 ↦↦_{Cα} 𝔈
    and Hom_{A.Cα}(φ−,−) :
      Hom_{O.Cα}𝔈(𝔊−,−) ↦_{CF.iso} Hom_{O.Cα}𝔈(𝔉−,−) :
      op-cat 𝔅 ×_C 𝔈 ↦↦_{Cα} cat-Set α
    and b ∈∘ 𝔅(⌊Obj⌋)
  shows Hom_{A.Cα}𝔈(φ(⌊NTMap⌋)(⌊b⌋),−) :
    Hom_{O.Cα}𝔈(𝔊(⌊ObjMap⌋)(⌊b⌋),−) ↦_{CF.iso} Hom_{O.Cα}𝔈(𝔉(⌊ObjMap⌋)(⌊b⌋),−) :
    𝔈 ↦↦_{Cα} cat-Set α
proof–
  interpret φ: is-ntcf α 𝔅 𝔈 𝔉 𝔊 φ by (rule assms(1))
```

from *category-axioms assms* **show** *?thesis*
by
(
fold
cat-bnt-proj-snd-tcf-lcomp-Hom-ntcf-Hom-snd[*OF assms*(*1*,*3*)]
cat-cf-lcomp-Hom-cf-Hom-snd[*OF* φ*.NTDom.is-functor-axioms assms*(*3*)]
cat-cf-lcomp-Hom-cf-Hom-snd[*OF* φ*.NTCod.is-functor-axioms assms*(*3*)],
intro bnt-proj-snd-is-iso-ntcf-if-is-iso-ntcf
)
(*cs-concl* **cs-shallow cs-simp**: *cat-op-simps* **cs-intro**: *cat-cs-intros*)
qed

29.19 Yoneda map for arbitrary functors

The concept of the Yoneda map for arbitrary functors was developed based on the function that was used in the statement of Lemma 3 in subsection 1.15 in [3].

definition *af-Yoneda-map* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$ =
($\lambda\varphi\in_{\circ}$*these-ntcfs* α ($\mathfrak{F}$(|*HomDom*|)) ($\mathfrak{F}$(|*HomCod*|)) $\mathfrak{F}$ $\mathfrak{G}$. $Hom_{A.C\alpha}(\varphi -,-)$)

Elementary properties.

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$
assumes $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
interpretation $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule* $\mathfrak{G}$)

mk-VLambda
af-Yoneda-map-def[**where** $\mathfrak{F}$=$\mathfrak{F}$ **and** $\mathfrak{G}$=$\mathfrak{G}$, *unfolded* $\mathfrak{F}$*.cf-HomDom* $\mathfrak{F}$*.cf-HomCod*]
|*vsv af-Yoneda-map-vsv*|
|*vdomain af-Yoneda-map-vdomain*[*cat-cs-simps*]|
|*app af-Yoneda-map-app*[*unfolded these-ntcfs-iff*, *cat-cs-simps*]|

end

29.20 Yoneda arrow for arbitrary functors

29.20.1 Definition and elementary properties

The following natural transformation is used in the proof of Lemma 3 in subsection 1.15 in [3].

definition *af-Yoneda-arrow* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ =
[
(
$\lambda b\in_{\circ}$($\mathfrak{F}$(|*HomDom*|))(|*Obj*|).
Yoneda-map α $Hom_{O.C\alpha}$$\mathfrak{F}$(|*HomCod*|)($\mathfrak{F}$(|*ObjMap*|)(|$b$|),−) ($\mathfrak{G}$(|*ObjMap*|)(|$b$|))(|
$\mathfrak{N}_{op\text{-}cat\ (\mathfrak{F}(|HomDom|)),\mathfrak{F}(|HomCod|)}(b,-)_{NTCF}$
|)
),
$\mathfrak{F}$,
$\mathfrak{G}$,
$\mathfrak{F}$(|*HomDom*|),
$\mathfrak{F}$(|*HomCod*|)
]$_{\circ}$

Components.

lemma *af-Yoneda-arrow-components*:
shows *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}(\!|NTMap|\!)$ =
(
$\lambda b \in_\circ \mathfrak{F}(\!|HomDom|\!)(\!|Obj|\!)$.
Yoneda-map α $Hom_{O.C\alpha}\mathfrak{F}(\!|HomCod|\!)(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),-)$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|$
$\mathfrak{N}_{op\text{-}cat\ (\mathfrak{F}(\!|HomDom|\!)),\mathfrak{F}(\!|HomCod|\!)}(b,-)_{NTCF}$
$|\!|$
)
and *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}(\!|NTDom|\!)$ = $\mathfrak{F}$
and *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}(\!|NTCod|\!)$ = $\mathfrak{G}$
and *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}(\!|NTDGDom|\!)$ = $\mathfrak{F}(\!|HomDom|\!)$
and *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}(\!|NTDGCod|\!)$ = $\mathfrak{F}(\!|HomCod|\!)$
unfolding *af-Yoneda-arrow-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

29.20.2 Natural transformation map

mk-VLambda *af-Yoneda-arrow-components*(*1*)
|*vsv af-Yoneda-arrow-NTMap-vsv*|

context
fixes α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$
assumes $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
begin

interpretation $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)

mk-VLambda
af-Yoneda-arrow-components(*1*)[**where** $\mathfrak{F}$=$\mathfrak{F}$, *unfolded* $\mathfrak{F}$.*cf-HomDom* $\mathfrak{F}$.*cf-HomCod*]
|*vdomain af-Yoneda-arrow-NTMap-vdomain*[*cat-cs-simps*]|
|*app af-Yoneda-arrow-NTMap-app*[*cat-cs-simps*]|

end

lemma (**in** *category*) *cat-af-Yoneda-arrow-is-ntcf*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N}$:
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
op-cat $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
shows *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$: $\mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

let *?H*$\mathfrak{G}$ = ‹$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$›
and *?H*$\mathfrak{F}$ = ‹$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$›
and *?Set* = ‹*cat-Set* α›
and *?Ym* =
‹
λb. *Yoneda-map*
α $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!),-)$ $(\mathfrak{G}(\!|ObjMap|\!)(\!|b|\!))(\!|\mathfrak{N}_{op\text{-}cat\ \mathfrak{B},\mathfrak{C}}(b,-)_{NTCF}|\!|)$
›

interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))
interpret $\mathfrak{N}$: *is-ntcf*
α ‹*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$› ‹*cat-Set* α› ‹$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$› ‹$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$› $\mathfrak{N}$
by (*rule assms*)

```
have comm[unfolded cat-op-simps]:
  (𝔑(⁅NTMap⁆)(⁅c, d⁆)•)(⁅ArrVal⁆)(⁅f ∘A𝔈 (q ∘A𝔈 𝔊(⁅ArrMap⁆)(⁅g⁆))⁆) =
    f ∘A𝔈 ((𝔑(⁅NTMap⁆)(⁅a, b⁆)•)(⁅ArrVal⁆)(⁅q⁆) ∘A𝔈 𝔉(⁅ArrMap⁆)(⁅g⁆))
  if g : a ↦op-cat 𝔅 c and f : b ↦𝔈 d and q : 𝔊(⁅ObjMap⁆)(⁅a⁆) ↦𝔈 b
  for q g f a b c d
proof–
  from that(1) have g: g : c ↦𝔅 a unfolding cat-op-simps by simp
  from category-axioms assms g that(2) have ab:
    [a, b]∘ ∈∘ (op-cat 𝔅 ×C 𝔈)(⁅Obj⁆)
    by (cs-concl cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros)
  from 𝔑.ntcf-NTMap-is-arr[OF ab] category-axioms assms g that(2) have 𝔑ab:
    𝔑(⁅NTMap⁆)(⁅a, b⁆)• :
      Hom 𝔈 (𝔊(⁅ObjMap⁆)(⁅a⁆)) b ↦cat-Set α Hom 𝔈 (𝔉(⁅ObjMap⁆)(⁅a⁆)) b
    by
      (
        cs-prems
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  have 𝔑-abq: (𝔑(⁅NTMap⁆)(⁅a, b⁆)•)(⁅ArrVal⁆)(⁅q⁆) : 𝔉(⁅ObjMap⁆)(⁅a⁆) ↦𝔈 b
    by
      (
        rule cat-Set-ArrVal-app-vrange[
          OF 𝔑ab, unfolded in-Hom-iff, OF that(3)
          ]
      )
  have [g, f]∘ : [a, b]∘ ↦op-cat 𝔅 ×C 𝔈 [c, d]∘
    by
      (
        rule
          cat-prod-2-is-arrI[
            OF 𝔉.HomDom.category-op category-axioms that(1,2)
            ]
      )
  then have
    𝔑(⁅NTMap⁆)(⁅c, d⁆)• ∘A cat-Set α Hom_O.Cα𝔈(𝔊−,−)(⁅ArrMap⁆)(⁅g, f⁆)• =
      Hom_O.Cα𝔈(𝔉−,−)(⁅ArrMap⁆)(⁅g, f⁆)• ∘A cat-Set α 𝔑(⁅NTMap⁆)(⁅a, b⁆)•
    by (rule is-ntcf.ntcf-Comp-commute[OF assms(3)])
  then have
    (𝔑(⁅NTMap⁆)(⁅c, d⁆)• ∘A ?Set ?H𝔊(⁅ArrMap⁆)(⁅g, f⁆)•)(⁅ArrVal⁆)(⁅q⁆) =
      (?H𝔉(⁅ArrMap⁆)(⁅g, f⁆)• ∘A ?Set 𝔑(⁅NTMap⁆)(⁅a, b⁆)•)(⁅ArrVal⁆)(⁅q⁆)
    by auto

  from
    this that(2,3) assms
    category-axioms 𝔉.HomDom.category-axioms 𝔉.HomDom.category-op category-op
    g 𝔑ab 𝔑-abq
  show
    (𝔑(⁅NTMap⁆)(⁅c, d⁆)•)(⁅ArrVal⁆)(⁅f ∘A𝔈 (q ∘A𝔈 𝔊(⁅ArrMap⁆)(⁅g⁆))⁆) =
      f ∘A𝔈 ((𝔑(⁅NTMap⁆)(⁅a, b⁆)•)(⁅ArrVal⁆)(⁅q⁆) ∘A𝔈 𝔉(⁅ArrMap⁆)(⁅g⁆))
    by
      (
        cs-prems
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
qed
```

```
show ?thesis
proof(rule is-ntcfI')

  show vfsequence (af-Yoneda-arrow α 𝔉 𝔊 𝔑)
    unfolding af-Yoneda-arrow-def by simp
  show vcard (af-Yoneda-arrow α 𝔉 𝔊 𝔑) = 5_ℕ
    unfolding af-Yoneda-arrow-def by (simp add: nat-omega-simps)

  have 𝔑b: 𝔑_{op-cat 𝔅,ℭ}(b,−)_{NTCF} :
    Hom_{O.Cα}ℭ(𝔊(|ObjMap|)(|b|),−) ↦_{CF} Hom_{O.Cα}ℭ(𝔉(|ObjMap|)(|b|),−) :
    ℭ ↦↦_{Cα} cat-Set α
    if b ∈_∘ 𝔅(|Obj|) for b
    by
      (
        rule
          bnt-proj-snd-is-ntcf
            [
              OF 𝔉.HomDom.category-op category-axioms assms(3),
              unfolded cat-op-simps,
              OF that,
              unfolded
                cat-cf-lcomp-Hom-cf-Hom-snd[OF assms(1) that]
                cat-cf-lcomp-Hom-cf-Hom-snd[OF assms(2) that]
            ]
      )

  show af-Yoneda-arrow α 𝔉 𝔊 𝔑(|NTMap|)(|b|) : 𝔉(|ObjMap|)(|b|) ↦_ℭ 𝔊(|ObjMap|)(|b|)
    if b ∈_∘ 𝔅(|Obj|) for b
  proof–
    let ?𝔊b = ‹𝔊(|ObjMap|)(|b|)›
      and ?𝔉b = ‹𝔉(|ObjMap|)(|b|)›
      and ?ℭ𝔊b = ‹ℭ(|CId|)(|𝔊(|ObjMap|)(|b|)|)›
    from that have ℭ𝔊b: ?ℭ𝔊b : ?𝔊b ↦_ℭ ?𝔊b by (auto simp: cat-cs-intros)
    from assms that have [b, ?𝔊b]_∘ ∈_∘ (op-cat 𝔅 ×_C ℭ)(|Obj|)
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
        )
    from 𝔑.ntcf-NTMap-is-arr[OF this] category-axioms assms that have 𝔑-b𝔊b:
      𝔑(|NTMap|)(|b, ?𝔊b|)_• : Hom ℭ ?𝔊b ?𝔊b ↦_{cat-Set α} Hom ℭ ?𝔉b ?𝔊b
      by
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cat-op-simps
            cs-intro: cat-cs-intros cat-prod-cs-intros
        )
    from ℭ𝔊b have 𝔑-b𝔊b-ℭ𝔊b:
      (𝔑(|NTMap|)(|b, ?𝔊b|)_•)(|ArrVal|)(|?ℭ𝔊b|) : ?𝔉b ↦_ℭ ?𝔊b
      by (rule cat-Set-ArrVal-app-vrange[OF 𝔑-b𝔊b, unfolded in-Hom-iff])
    with category-axioms assms that 𝔑b[OF that] show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
        )
  qed
```

```
show
  af-Yoneda-arrow α 𝔉 𝔊 𝔑(|NTMap|)(|b|) ∘_A𝔈 𝔉(|ArrMap|)(|f|) =
    𝔊(|ArrMap|)(|f|) ∘_A𝔈 af-Yoneda-arrow α 𝔉 𝔊 𝔑(|NTMap|)(|a|)
  if f : a ↦_𝔅 b for a b f
proof–

  from that have a: a ∈_∘ 𝔅(|Obj|) and b: b ∈_∘ 𝔅(|Obj|) by auto

  let ?𝔅a = ‹𝔅(|CId|)(|a|)›
    and ?𝔅b = ‹𝔅(|CId|)(|b|)›
    and ?𝔊a = ‹𝔊(|ObjMap|)(|a|)›
    and ?𝔊b = ‹𝔊(|ObjMap|)(|b|)›
    and ?𝔉a = ‹𝔉(|ObjMap|)(|a|)›
    and ?𝔉b = ‹𝔉(|ObjMap|)(|b|)›
    and ?𝔈𝔊a = ‹𝔈(|CId|)(|𝔊(|ObjMap|)(|a|)|)›
    and ?𝔈𝔊b = ‹𝔈(|CId|)(|𝔊(|ObjMap|)(|b|)|)›

  from that have 𝔈𝔊a: ?𝔈𝔊a : ?𝔊a ↦_𝔈 ?𝔊a by (auto intro: cat-cs-intros)
  from that have 𝔈𝔊b: ?𝔈𝔊b : ?𝔊b ↦_𝔈 ?𝔊b by (auto intro: cat-cs-intros)
  from that have 𝔅a: ?𝔅a : a ↦_𝔅 a by (auto intro: cat-cs-intros)

  from assms that have [b, ?𝔊b]_∘ ∈_∘ (op-cat 𝔅 ×_C 𝔈)(|Obj|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  from 𝔑.ntcf-NTMap-is-arr[OF this] category-axioms assms that have 𝔑-b𝔊b:
    𝔑(|NTMap|)(|b, ?𝔊b|)_• : Hom 𝔈 ?𝔊b ?𝔊b ↦_cat-Set α Hom 𝔈 ?𝔉b ?𝔊b
    by
      (
        cs-prems cs-shallow
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  from 𝔈𝔊b have 𝔑-b𝔊b-𝔈𝔊b:
    (𝔑(|NTMap|)(|b, ?𝔊b|)_•)(|ArrVal|)(|?𝔈𝔊b|) : ?𝔉b ↦_𝔈 ?𝔊b
    by (rule cat-Set-ArrVal-app-vrange[OF 𝔑-b𝔊b, unfolded in-Hom-iff])

  from assms that have [a, ?𝔊a]_∘ ∈_∘ (op-cat 𝔅 ×_C 𝔈)(|Obj|)
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros cat-op-intros cat-prod-cs-intros
      )
  from 𝔑.ntcf-NTMap-is-arr[OF this] category-axioms assms that have 𝔑-a𝔊a:
    𝔑(|NTMap|)(|a, ?𝔊a|)_• : Hom 𝔈 ?𝔊a ?𝔊a ↦_cat-Set α Hom 𝔈 ?𝔉a ?𝔊a
    by
      (
        cs-prems
          cs-simp: cat-cs-simps cat-op-simps
          cs-intro: cat-cs-intros cat-prod-cs-intros
      )
  from 𝔈𝔊a have 𝔑-a𝔊a-𝔈𝔊a:
    (𝔑(|NTMap|)(|a, ?𝔊a|)_•)(|ArrVal|)(|?𝔈𝔊a|) : ?𝔉a ↦_𝔈 ?𝔊a
```

by (*rule cat-Set-ArrVal-app-vrange*[*OF* $\mathfrak{N}$-$a\mathfrak{G}a$, *unfolded in-Hom-iff*])

from
comm[*OF* $\mathfrak{B}a$ $\mathfrak{G}$.*cf-ArrMap-is-arr*[*OF that*] $\mathfrak{C}\mathfrak{G}a$]
category-axioms assms that $\mathfrak{N}$-$a\mathfrak{G}a$-$\mathfrak{C}\mathfrak{G}a$
have $\mathfrak{N}$-a-$\mathfrak{G}b$[*symmetric*, *cat-cs-simps*]:
$(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ ?\mathfrak{G}b \rfloor_{\bullet})(\lfloor ArrVal \rfloor)(\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)) =$
$\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ ?\mathfrak{G}a \rfloor_{\bullet})(\lfloor ArrVal \rfloor)(\lfloor ?\mathfrak{C}\mathfrak{G}a \rfloor)$
by (*cs-prems* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

from *comm*[*OF that* $\mathfrak{C}\mathfrak{G}b$ $\mathfrak{C}\mathfrak{G}b$] *category-axioms assms that* $\mathfrak{N}$-$b\mathfrak{G}b$-$\mathfrak{C}\mathfrak{G}b$
have $\mathfrak{N}$-a-$\mathfrak{G}b'$[*cat-cs-simps*]:
$(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a,\ ?\mathfrak{G}b \rfloor_{\bullet})(\lfloor ArrVal \rfloor)(\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)) =$
$(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor b,\ ?\mathfrak{G}b \rfloor_{\bullet})(\lfloor ArrVal \rfloor)(\lfloor ?\mathfrak{C}\mathfrak{G}b \rfloor) \circ_{A\mathfrak{C}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$
by (*cs-prems* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

from *category-axioms assms that* $\mathfrak{N}b$[*OF a*] $\mathfrak{N}b$[*OF b*] **show** *?thesis*
by (*cs-concl* **cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros cat-op-intros*)

qed

qed (*auto simp*: *af-Yoneda-arrow-components cat-cs-simps intro*: *cat-cs-intros*)

qed

lemma (**in** *category*) *cat-af-Yoneda-arrow-is-ntcf'*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N}$:
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
op-cat $\mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
and $\beta = \alpha$
and $\mathfrak{F}' = \mathfrak{F}$
and $\mathfrak{G}' = \mathfrak{G}$
shows *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N} : \mathfrak{F}' \mapsto_{CF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{C\beta} \mathfrak{C}$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule cat-af-Yoneda-arrow-is-ntcf*)

lemmas [*cat-cs-intros*] = *category.cat-af-Yoneda-arrow-is-ntcf'*

29.20.3 Yoneda Lemma for arbitrary functors

The following lemmas correspond to variants of the elements of Lemma 3 in subsection 1.15 in [3].

lemma (**in** *category*) *cat-af-Yoneda-map-af-Yoneda-arrow-app*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N}$:
$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF} Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
op-cat $\mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
shows $\mathfrak{N} = Hom_{A.C\alpha}(af\text{-}Yoneda\text{-}arrow\ \alpha\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}-,-)$
proof–

let *?H*$\mathfrak{G}$ = $\langle Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)\rangle$
and *?H*$\mathfrak{F}$ = $\langle Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)\rangle$
and *?aYa* = $\langle \lambda\mathfrak{N}.\ af\text{-}Yoneda\text{-}arrow\ \alpha\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}\rangle$

interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))

```
interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))

interpret 𝔑: is-ntcf α ‹op-cat 𝔅 ×_C ℭ› ‹cat-Set α› ‹?H𝔊› ‹?H𝔉› 𝔑
  by (rule assms(3))
interpret aY𝔑: is-ntcf α 𝔅 ℭ 𝔉 𝔊 ‹?aYa 𝔑›
  by (rule cat-af-Yoneda-arrow-is-ntcf[OF assms])
interpret HY𝔑: is-ntcf
  α ‹op-cat 𝔅 ×_C ℭ› ‹cat-Set α› ‹?H𝔊› ‹?H𝔉› ‹Hom_{A.Cα}(?aYa 𝔑-,-)›
  by (rule cat-ntcf-lcomp-Hom-is-ntcf[OF aY𝔑.is-ntcf-axioms])

show [cat-cs-simps]: 𝔑 = Hom_{A.Cα}(?aYa 𝔑-,-)
proof
  (
    rule sym,
    rule ntcf-eqI[OF HY𝔑.is-ntcf-axioms assms(3)],
    rule vsv-eqI;
    (intro HY𝔑.NTMap.vsv-axioms 𝔑.NTMap.vsv-axioms)?;
    (unfold 𝔑.ntcf-NTMap-vdomain HY𝔑.ntcf-NTMap-vdomain)?
  )
  fix bc assume prems': bc ∈₀ (op-cat 𝔅 ×_C ℭ)(|Obj|)
  then obtain b c
    where bc-def: bc = [b, c]₀
      and op-b: b ∈₀ op-cat 𝔅(|Obj|)
      and c: c ∈₀ ℭ(|Obj|)
    by (auto intro: cat-prod-2-ObjE cat-cs-intros)
  from op-b have b: b ∈₀ 𝔅(|Obj|) unfolding cat-op-simps by simp

  then have 𝔊b: 𝔊(|ObjMap|)(|b|) ∈₀ ℭ(|Obj|) and 𝔉b: 𝔉(|ObjMap|)(|b|) ∈₀ ℭ(|Obj|)
    by (auto intro: cat-cs-intros)
  have Ym-𝔑:
    Yoneda-map α Hom_{O.Cα}ℭ(𝔉(|ObjMap|)(|b|),-) (𝔊(|ObjMap|)(|b|))(|
      𝔑_{op-cat 𝔅,ℭ}(b,-)_{NTCF}
    |) = ?aYa 𝔑(|NTMap|)(|b|)
    unfolding af-Yoneda-arrow-NTMap-app[OF assms(1) b] by simp

  from
    bnt-proj-snd-is-ntcf
      [
        OF 𝔉.HomDom.category-op category-axioms assms(3) op-b,
        unfolded
          cat-cf-lcomp-Hom-cf-Hom-snd[OF assms(1) b]
          cat-cf-lcomp-Hom-cf-Hom-snd[OF assms(2) b]
      ]
  have 𝔑b: 𝔑_{op-cat 𝔅,ℭ}(b,-)_{NTCF} :
    Hom_{O.Cα}ℭ(𝔊(|ObjMap|)(|b|),-) ↦_{CF} Hom_{O.Cα}ℭ(𝔉(|ObjMap|)(|b|),-) :
    ℭ ↦↦_{Cα} cat-Set α
    by simp
  from c show Hom_{A.Cα}(?aYa 𝔑-,-)(|NTMap|)(|bc|) = 𝔑(|NTMap|)(|bc|)
    unfolding
      bc-def
      cat-ntcf-lcomp-Hom-ntcf-Hom-snd-NTMap-app[OF aY𝔑.is-ntcf-axioms b c]
      cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique(2)[
        OF 𝔊b 𝔉b 𝔑b, unfolded Ym-𝔑, symmetric
      ]
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)

qed simp-all
```

qed

lemma (**in** *category*) *cat-af-Yoneda-Lemma*:
  assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
  shows *v11* (*af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$)
    and $\mathcal{R}_\circ$ (*af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$) =
    these-ntcfs α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$ $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$
    and (*af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$)$^{-1}{}_\circ$ =
      (
        $\lambda\mathfrak{N}\in_\circ$ *these-ntcfs*
          α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$ $Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$.
          af-Yoneda-arrow α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$
)
proof–

  let *?H*$\mathfrak{G}$ = ‹$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$›
    and *?H*$\mathfrak{F}$ = ‹$Hom_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$›
    and *?aYm* = ‹*af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$›
    and *?aYa* = ‹$\lambda\mathfrak{N}$. *af-Yoneda-arrow* α $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$›

  interpret $\mathfrak{F}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{G}$: *is-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*2*))

  show *v11-aY*: *v11 ?aYm*
  proof
    (
      intro vsv.vsv-valeq-v11I,
      unfold af-Yoneda-map-vdomain[*OF assms*] *these-ntcfs-iff*
)

    show *vsv* (*af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$) **by** (*rule af-Yoneda-map-vsv*[*OF assms*])

    fix φ ψ **assume** *prems*:
      $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
      $\psi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
      ?aYm($\lfloor\varphi\rfloor$) = *?aYm*($\lfloor\psi\rfloor$)

    interpret φ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ φ **by** (*rule prems*(*1*))
    interpret ψ: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ ψ **by** (*rule prems*(*2*))

    from *prems*(*3*) **have** *H*φ-*H*ψ: $Hom_{A.C\alpha}(\varphi-,-) = Hom_{A.C\alpha}(\psi-,-)$
      unfolding
        af-Yoneda-map-app[*OF assms prems*(*1*)]
        af-Yoneda-map-app[*OF assms prems*(*2*)]
      by *simp*

    show $\varphi = \psi$
    proof
      (
        rule ntcf-eqI[*OF prems*(*1,2*)],
        rule vsv-eqI,
        unfold φ*.ntcf-NTMap-vdomain* ψ*.ntcf-NTMap-vdomain*
)
      fix *b* **assume** *prems'*: $b \in_\circ \mathfrak{B}(\lfloor Obj\rfloor)$
      from *prems'* **have** φb: $\varphi(\lfloor NTMap\rfloor)(\lfloor b\rfloor) : \mathfrak{F}(\lfloor ObjMap\rfloor)(\lfloor b\rfloor) \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor ObjMap\rfloor)(\lfloor b\rfloor)$
        and ψb: $\psi(\lfloor NTMap\rfloor)(\lfloor b\rfloor) : \mathfrak{F}(\lfloor ObjMap\rfloor)(\lfloor b\rfloor) \mapsto_{\mathfrak{C}} \mathfrak{G}(\lfloor ObjMap\rfloor)(\lfloor b\rfloor)$
        and $\mathfrak{G}b$: $\mathfrak{G}(\lfloor ObjMap\rfloor)(\lfloor b\rfloor) \in_\circ \mathfrak{C}(\lfloor Obj\rfloor)$
        and $\mathfrak{F}b$: $\mathfrak{F}(\lfloor ObjMap\rfloor)(\lfloor b\rfloor) \in_\circ \mathfrak{C}(\lfloor Obj\rfloor)$

```
    by (auto intro: cat-cs-intros cat-prod-cs-intros)
  have Hom_{A.COα}ℭ(φ(|NTMap|)(|b|),-) = Hom_{A.COα}ℭ(ψ(|NTMap|)(|b|),-)
  proof
    (
      rule
        ntcf-eqI
          [
            OF
              cat-ntcf-Hom-snd-is-ntcf[OF φb]
              cat-ntcf-Hom-snd-is-ntcf[OF ψb]
          ]
    )
    show Hom_{A.COα}ℭ(φ(|NTMap|)(|b|),-)(|NTMap|) = Hom_{A.COα}ℭ(ψ(|NTMap|)(|b|),-)(|NTMap|)
    proof
      (
        rule vsv-eqI,
        unfold
          ntcf-Hom-snd-NTMap-vdomain[OF φb]
          ntcf-Hom-snd-NTMap-vdomain[OF ψb]
      )
      fix c assume prems'': c ∈₀ ℭ(|Obj|)
      note H = cat-ntcf-lcomp-Hom-ntcf-Hom-snd-NTMap-app
      show
        Hom_{A.COα}ℭ(φ(|NTMap|)(|b|),-)(|NTMap|)(|c|) =
          Hom_{A.COα}ℭ(ψ(|NTMap|)(|b|),-)(|NTMap|)(|c|)
        unfolding
          H[OF prems(1) prems' prems'', symmetric]
          H[OF prems(2) prems' prems'', symmetric]
          Hφ-Hψ
        by simp
    qed
      (
        simp-all add:
          ntcf-Hom-snd-NTMap-vsv[OF ψb] ntcf-Hom-snd-NTMap-vsv[OF φb]
      )
  qed simp-all
  with φb ψb show φ(|NTMap|)(|b|) = ψ(|NTMap|)(|b|)
    by (auto intro: cat-ntcf-Hom-snd-inj)
qed auto

qed

interpret aYm: v11 ?aYm by (rule v11-aY)

have [cat-cs-simps]: ?aYm(|?aYa 𝔑|) = 𝔑
  if 𝔑 : ?Hℌ ↦↦_{CF} ?Hℱ : op-cat 𝔅 ×_C ℭ ↦↦_{Cα} cat-Set α for 𝔑
  using category-axioms assms that
  by
    (
      cs-concl cs-shallow
        cs-simp:
          cat-af-Yoneda-map-af-Yoneda-arrow-app[symmetric] cat-cs-simps
        cs-intro: cat-cs-intros
    )

show aYm-vrange:
  ℛ₀ ?aYm = these-ntcfs α (op-cat 𝔅 ×_C ℭ) (cat-Set α) ?Hℌ ?Hℱ
proof(intro vsubset-antisym)
```

show $\mathcal{R}_\circ$ *?aYm* $\subseteq_\circ$ *these-ntcfs* α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) *?HG ?HF*
proof
(
rule vsv.vsv-vrange-vsubset,
unfold these-ntcfs-iff af-Yoneda-map-vdomain[*OF assms*]
)
fix φ **assume** $\varphi : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
with *category-axioms assms* **show**
?aYm($\lfloor\varphi\rfloor$) : *?HG* $\mapsto_{CF}$ *?HF* : *op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
qed (*auto intro*: *af-Yoneda-map-vsv*)

show *these-ntcfs* α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) *?HG ?HF* $\subseteq_\circ$ $\mathcal{R}_\circ$ *?aYm*
proof(*rule vsubsetI*, *unfold these-ntcfs-iff*)
fix $\mathfrak{N}$ **assume** *prems*:
$\mathfrak{N}$: *?HG* $\mapsto_{CF}$ *?HF* : *op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
interpret *aY*$\mathfrak{N}$: *is-ntcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ ‹*?aYa* $\mathfrak{N}$›
by (*rule cat-af-Yoneda-arrow-is-ntcf*[*OF assms prems*])
from *prems* **have** $\mathfrak{N}$-*def*: $\mathfrak{N}$ = *?aYm*($\lfloor$*?aYa* $\mathfrak{N}\rfloor$)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
from *assms aY*$\mathfrak{N}$.*is-ntcf-axioms* **have** *?aYa* $\mathfrak{N}$ $\in_\circ$ $\mathcal{D}_\circ$ *?aYm*
by (*cs-concl* **cs-shallow cs-simp**: *these-ntcfs-iff cat-cs-simps*)
then show $\mathfrak{N}$ $\in_\circ$ $\mathcal{R}_\circ$ *?aYm* **by** (*subst* $\mathfrak{N}$-*def*, *intro aYm.vsv-vimageI2*) *auto*
qed

qed

show *?aYm*$^{-1}{}_\circ$ =
(*λ*$\mathfrak{N}$$\in_\circ$*these-ntcfs* α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) *?HG ?HF*. *?aYa* $\mathfrak{N}$)
proof
(
rule vsv-eqI,
unfold vdomain-vconverse vdomain-VLambda aYm-vrange these-ntcfs-iff
)
from *aYm.v11-axioms* **show** *vsv* ((*af-Yoneda-map* α $\mathfrak{F}$ $\mathfrak{G}$)$^{-1}{}_\circ$) **by** *auto*
fix $\mathfrak{N}$ **assume** *prems*: $\mathfrak{N}$: *?HG* $\mapsto_{CF}$ *?HF* : *op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$ $\mapsto\mapsto_{C\alpha}$ *cat-Set* α
then have $\mathfrak{N}$: $\mathfrak{N}$ $\in_\circ$ *these-ntcfs* α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) *?HG ?HF*
by *simp*
show *?aYm*$^{-1}{}_\circ$($\lfloor\mathfrak{N}\rfloor$) =
(*λ*$\mathfrak{N}$$\in_\circ$*these-ntcfs* α (*op-cat* $\mathfrak{B}$ $\times_C$ $\mathfrak{C}$) (*cat-Set* α) *?HG ?HF*. *?aYa* $\mathfrak{N}$)($\lfloor\mathfrak{N}\rfloor$)
proof
(
intro aYm.v11-vconverse-app,
unfold beta[*OF* $\mathfrak{N}$] *af-Yoneda-map-vdomain*[*OF assms*] *these-ntcfs-iff*
)
from *prems* **show** $\mathfrak{N}$-*def*: *?aYm*($\lfloor$*?aYa* $\mathfrak{N}\rfloor$) = $\mathfrak{N}$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps*)
show *?aYa* $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{CF}$ $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\mapsto_{C\alpha}$ $\mathfrak{C}$
by (*rule cat-af-Yoneda-arrow-is-ntcf*[*OF assms prems*])
qed
qed *simp-all*

qed

29.20.4 Inverse of the Yoneda map for arbitrary functors

lemma (**in** *category*) *inv-af-Yoneda-map-v11*:

assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows *v11* $((\textit{af-Yoneda-map}\ \alpha\ \mathfrak{F}\ \mathfrak{G})^{-1}{}_{\circ})$
using *cat-af-Yoneda-Lemma*(*1*)[*OF assms*] **by** (*simp add*: *v11.v11-vconverse*)

lemma (**in** *category*) *inv-af-Yoneda-map-vdomain*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{D}_{\circ}$ $((\textit{af-Yoneda-map}\ \alpha\ \mathfrak{F}\ \mathfrak{G})^{-1}{}_{\circ})$ =
these-ntcfs α (*op-cat* $\mathfrak{B} \times_C \mathfrak{C}$) (*cat-Set* α) $\textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$ $\textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$
unfolding *cat-af-Yoneda-Lemma*(*3*)[*OF assms*] **by** *simp*

lemmas [*cat-cs-simps*] = *category.inv-af-Yoneda-map-vdomain*

lemma (**in** *category*) *inv-af-Yoneda-map-app*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
and $\mathfrak{N}$:
$\textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF} \textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
op-cat $\mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
shows $(\textit{af-Yoneda-map}\ \alpha\ \mathfrak{F}\ \mathfrak{G})^{-1}{}_{\circ}(\lfloor\mathfrak{N}\rfloor) = \textit{af-Yoneda-arrow}\ \alpha\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$
using *assms*(*3*) **unfolding** *cat-af-Yoneda-Lemma*(*3*)[*OF assms*(*1,2*)] **by** *simp*

lemmas [*cat-cs-simps*] = *category.inv-af-Yoneda-map-app*

lemma (**in** *category*) *inv-af-Yoneda-map-vrange*:
assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$ **and** $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\mathcal{R}_{\circ}$ $((\textit{af-Yoneda-map}\ \alpha\ \mathfrak{F}\ \mathfrak{G})^{-1}{}_{\circ}) = \textit{these-ntcfs}\ \alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}$
proof–
interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
from *assms* **show** *?thesis*
unfolding *af-Yoneda-map-def* **by** (*simp add*: *cat-cs-simps*)
qed

29.20.5 Yoneda map for arbitrary functors and natural isomorphisms

The following lemmas correspond to variants of the elements of Lemma 3 in subsection 1.15 in [3].

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-is-iso-ntcf-if-is-iso-ntcf*:
assumes $\varphi : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
shows $\textit{Hom}_{A.C\alpha}(\varphi-,-)$:
$\textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF.iso} \textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
op-cat $\mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
proof–
interpret φ: *is-iso-ntcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \varphi$ **by** (*rule assms*(*1*))
show *?thesis*
proof(*intro cat-ntcf-lcomp-Hom-if-ntcf-Hom-snd-is-iso-ntcf*)
fix b **assume** $b \in_{\circ} \mathfrak{B}(\lfloor\textit{Obj}\rfloor)$
then show $\textit{Hom}_{A.C\alpha}\mathfrak{C}(\varphi(\lfloor\textit{NTMap}\rfloor)(\lfloor b\rfloor),-)$:
$\textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}(\lfloor\textit{ObjMap}\rfloor)(\lfloor b\rfloor),-) \mapsto_{CF.iso} \textit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}(\lfloor\textit{ObjMap}\rfloor)(\lfloor b\rfloor),-)$:
$\mathfrak{C} \mapsto\mapsto_{C\alpha}$ *cat-Set* α
by
(
auto intro!:
cat-is-iso-arr-ntcf-Hom-snd-is-iso-ntcf cat-arrow-cs-intros
)
qed (*auto simp*: *cat-cs-intros*)
qed

lemma (**in** *category*) *cat-ntcf-lcomp-Hom-is-iso-ntcf-if-is-iso-ntcf'*:
  assumes $\varphi : \mathfrak{F} \mapsto\mapsto_{CF.iso} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\beta = \alpha$
    and $\mathfrak{G}' = \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$
    and $\mathfrak{F}' = \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$
    and $\mathfrak{B}' = \mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C}$
    and $\mathfrak{C}' = \mathit{cat\text{-}Set}\ \alpha$
  shows $\mathit{Hom}_{A.C\alpha}(\varphi-,-) : \mathfrak{G}' \mapsto_{CF.iso} \mathfrak{F}' : \mathfrak{B}' \mapsto\mapsto_{C\beta} \mathfrak{C}'$
  using *assms*(*1*)
  unfolding *assms*(*2–6*)
  by (*rule cat-ntcf-lcomp-Hom-is-iso-ntcf-if-is-iso-ntcf*)

lemmas [*cat-cs-intros*] =
  category.cat-ntcf-lcomp-Hom-is-iso-ntcf-if-is-iso-ntcf'

lemma (**in** *category*) *cat-aYa-is-iso-ntcf-if-ntcf-lcomp-Hom-is-iso-ntcf*:
  assumes $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    and $\mathfrak{N}$:
      $\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF.iso} \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
      $\mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathit{cat\text{-}Set}\ \alpha$
  shows *af-Yoneda-arrow* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N} : \mathfrak{F} \mapsto_{CF.iso} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
proof–

  let *?aYa* = ‹*af-Yoneda-arrow* $\alpha\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$›

  interpret $\mathfrak{F}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{G}$: *is-functor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(*2*))
  interpret $\mathfrak{N}$: *is-iso-ntcf*
    α ‹*op-cat* $\mathfrak{B} \times_C \mathfrak{C}$› ‹*cat-Set* α› ‹$\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-)$› ‹$\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$› $\mathfrak{N}$
    by (*rule assms*(*3*))

  from *assms*(*1,2*) $\mathfrak{N}$.*is-ntcf-axioms* **have** $\mathfrak{N}$-*def*: $\mathfrak{N} = \mathit{Hom}_{A.C\alpha}(\mathit{?aYa}-,-)$
    by
      (
        cs-concl **cs-shallow**
          cs-simp: *cat-af-Yoneda-map-af-Yoneda-arrow-app*[*symmetric*]
)

  from *category-axioms assms* **have** *aYa*: $\mathit{?aYa} : \mathfrak{F} \mapsto_{CF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{C\alpha} \mathfrak{C}$
    by (*cs-concl* **cs-shallow** **cs-intro**: *cat-cs-intros*)
  have *Hom-aYa*: $\mathit{Hom}_{A.C\alpha}(\mathit{?aYa}-,-)$:
    $\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}-,-) \mapsto_{CF.iso} \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}-,-)$:
    $\mathit{op\text{-}cat}\ \mathfrak{B} \times_C \mathfrak{C} \mapsto\mapsto_{C\alpha} \mathit{cat\text{-}Set}\ \alpha$
    by (*auto intro*: *assms*(*3*) *simp add*: $\mathfrak{N}$-*def*[*symmetric*])
  have *Hb*:
    $\mathit{Hom}_{A.C\alpha}\mathfrak{C}(\mathit{?aYa}(\lfloor \mathit{NTMap} \rfloor)(\lfloor b \rfloor),-)$:
      $\mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{G}(\lfloor \mathit{ObjMap} \rfloor)(\lfloor b \rfloor),-) \mapsto_{CF.iso} \mathit{Hom}_{O.C\alpha}\mathfrak{C}(\mathfrak{F}(\lfloor \mathit{ObjMap} \rfloor)(\lfloor b \rfloor),-)$:
      $\mathfrak{C} \mapsto\mapsto_{C\alpha} \mathit{cat\text{-}Set}\ \alpha$
    if $b \in_\circ \mathfrak{B}(\lfloor \mathit{Obj} \rfloor)$ **for** b
    by
      (
        rule cat-ntcf-Hom-snd-if-ntcf-lcomp-Hom-is-iso-ntcf[
          OF aYa Hom-aYa that
          ]
)

  show *?thesis*

```
  proof(intro is-iso-ntcfI)
    from category-axioms assms show
      af-Yoneda-arrow α 𝔉 𝔊 𝔑 : 𝔉 ↦_CF 𝔊 : 𝔅 ↦↦_Cα ℭ
      by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
    fix b assume prems: b ∈₀ 𝔅(|Obj|)
    then have 𝔊b: 𝔊(|ObjMap|)(|b|) ∈₀ ℭ(|Obj|) and 𝔉b: 𝔉(|ObjMap|)(|b|) ∈₀ ℭ(|Obj|)
      by (auto intro: cat-cs-intros)
    from assms(1,2) aYa prems have aYa-b:
      ?aYa(|NTMap|)(|b|) : 𝔉(|ObjMap|)(|b|) ↦_ℭ 𝔊(|ObjMap|)(|b|)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros cs-simp: cat-cs-simps)
    show af-Yoneda-arrow α 𝔉 𝔊 𝔑(|NTMap|)(|b|) : 𝔉(|ObjMap|)(|b|) ↦_iso ℭ 𝔊(|ObjMap|)(|b|)
      by
        (
          rule cat-is-iso-arr-if-ntcf-Hom-snd-is-iso-ntcf[
            OF aYa-b Hb[OF prems]
            ]
        )
  qed

qed

lemma (in category) cat-aYa-is-iso-ntcf-if-ntcf-lcomp-Hom-is-iso-ntcf':
  assumes 𝔉 : 𝔅 ↦↦_Cα ℭ
    and 𝔊 : 𝔅 ↦↦_Cα ℭ
    and 𝔑 :
      Hom_O.Cα ℭ(𝔊−,−) ↦_CF.iso Hom_O.Cα ℭ(𝔉−,−) :
      op-cat 𝔅 ×_C ℭ ↦↦_Cα cat-Set α
    and β = α
    and 𝔉' = 𝔉
    and 𝔊' = 𝔊
  shows af-Yoneda-arrow α 𝔉 𝔊 𝔑 : 𝔉' ↦_CF.iso 𝔊' : 𝔅 ↦↦_Cα ℭ
  using assms(1–3)
  unfolding assms(4–6)
  by (rule cat-aYa-is-iso-ntcf-if-ntcf-lcomp-Hom-is-iso-ntcf)

lemmas [cat-cs-intros] =
  category.cat-aYa-is-iso-ntcf-if-ntcf-lcomp-Hom-is-iso-ntcf'

lemma (in category) cat-iso-functor-if-cf-lcomp-Hom-iso-functor:
  assumes 𝔉 : 𝔅 ↦↦_Cα ℭ
    and 𝔊 : 𝔅 ↦↦_Cα ℭ
    and Hom_O.Cα ℭ(𝔉−,−) ≈_CFα Hom_O.Cα ℭ(𝔊−,−)
  shows 𝔉 ≈_CFα 𝔊
proof–
  let ?H𝔊 = ‹Hom_O.Cα ℭ(𝔊−,−)›
    and ?H𝔉 = ‹Hom_O.Cα ℭ(𝔉−,−)›
    and ?aYa = ‹λ𝔑. af-Yoneda-arrow α 𝔉 𝔊 𝔑›
  interpret 𝔉: is-functor α 𝔅 ℭ 𝔉 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  from assms(3) obtain 𝔑 𝔄 𝔇 where 𝔑: 𝔑 : ?H𝔉 ↦_CF.iso ?H𝔊 : 𝔄 ↦↦_Cα 𝔇
    by auto
  interpret 𝔑: is-iso-ntcf α 𝔄 𝔇 ?H𝔉 ?H𝔊 𝔑 by (rule 𝔑)
  from category-axioms assms have ?H𝔉 : op-cat 𝔅 ×_C ℭ ↦↦_Cα cat-Set α
    by (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  then have 𝔄-def: 𝔄 = op-cat 𝔅 ×_C ℭ and 𝔇-def: 𝔇 = cat-Set α
    by (force simp: cat-cs-simps)+
  note 𝔑 = 𝔑[unfolded 𝔄-def 𝔇-def]
  from 𝔑 have 𝔑 : ?H𝔉 ↦_CF ?H𝔊 : op-cat 𝔅 ×_C ℭ ↦↦_Cα cat-Set α
```

```
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: cat-cs-intros ntcf-cs-intros
      )
  from category-axioms assms 𝔑 have
    af-Yoneda-arrow α 𝔊 𝔉 𝔑 : 𝔊 ↦_CF.iso 𝔉 : 𝔅 ↦↦_Cα ℭ
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  then have 𝔊 ≈_CFα 𝔉 by (clarsimp intro!: iso-functorI)
  then show ?thesis by (rule iso-functor-sym)
qed

lemma (in category) cat-cf-lcomp-Hom-iso-functor-if-iso-functor:
  assumes 𝔉 : 𝔅 ↦↦_Cα ℭ
    and 𝔊 : 𝔅 ↦↦_Cα ℭ
    and 𝔉 ≈_CFα 𝔊
  shows Hom_O.Cα ℭ(𝔉−,−) ≈_CFα Hom_O.Cα ℭ(𝔊−,−)
proof–
  let ?H𝔊 = ‹Hom_O.Cα ℭ(𝔊−,−)›
    and ?H𝔉 = ‹Hom_O.Cα ℭ(𝔉−,−)›
    and ?aYa = ‹λ𝔑. af-Yoneda-arrow α 𝔉 𝔊 𝔑›
  interpret 𝔉: is-functor α 𝔅 ℭ 𝔉 by (rule assms(1))
  interpret 𝔊: is-functor α 𝔅 ℭ 𝔊 by (rule assms(2))
  from assms obtain 𝔅' ℭ' φ where φ: φ : 𝔉 ↦_CF.iso 𝔊 : 𝔅' ↦↦_Cα ℭ'
    by auto
  interpret φ: is-iso-ntcf α 𝔅' ℭ' 𝔉 𝔊 φ by (rule φ)
  from assms φ.NTDom.is-functor-axioms
  have 𝔅'-def: 𝔅' = 𝔅 and ℭ'-def: ℭ' = ℭ
    by fast+
  note φ = φ[unfolded 𝔅'-def ℭ'-def]
  show ?thesis
    by (rule iso-functor-sym)
      (
        intro iso-functorI[
          OF cat-ntcf-lcomp-Hom-is-iso-ntcf-if-is-iso-ntcf[OF φ]
          ]
      )
qed

lemma (in category) cat-cf-lcomp-Hom-iso-functor-if-iso-functor':
  assumes 𝔉 : 𝔅 ↦↦_Cα ℭ
    and 𝔊 : 𝔅 ↦↦_Cα ℭ
    and 𝔉 ≈_CFα 𝔊
    and α' = α
    and ℭ' = ℭ
  shows Hom_O.Cα ℭ(𝔉−,−) ≈_CFα Hom_O.C_α' ℭ'(𝔊−,−)
  using assms(1–3)
  unfolding assms(4,5)
  by (rule cat-cf-lcomp-Hom-iso-functor-if-iso-functor)

lemmas [cat-cs-intros] =
  category.cat-cf-lcomp-Hom-iso-functor-if-iso-functor'
```

29.21 The Yoneda Functor

29.21.1 Definition and elementary properties

See Chapter III-2 in [7].

definition *Yoneda-functor* :: $V \Rightarrow V \Rightarrow V$
where *Yoneda-functor* $\alpha\ \mathfrak{D}$ =
[
($\lambda r \in_\circ$ *op-cat* $\mathfrak{D}(\!|Obj|\!)$. *cf-map* ($Hom_{O.C\alpha}\mathfrak{D}(r,-)$)),
($\lambda f \in_\circ$ *op-cat* $\mathfrak{D}(\!|Arr|\!)$. *ntcf-arrow* ($Hom_{A.C\alpha}\mathfrak{D}(f,-)$)),
op-cat $\mathfrak{D}$,
cat-FUNCT $\alpha\ \mathfrak{D}$ (*cat-Set* α)
]$_\circ$

Components.

lemma *Yoneda-functor-components*:
shows *Yoneda-functor* $\alpha\ \mathfrak{D}(\!|ObjMap|\!)$ =
($\lambda r \in_\circ$ *op-cat* $\mathfrak{D}(\!|Obj|\!)$. *cf-map* ($Hom_{O.C\alpha}\mathfrak{D}(r,-)$))
and *Yoneda-functor* $\alpha\ \mathfrak{D}(\!|ArrMap|\!)$ =
($\lambda f \in_\circ$ *op-cat* $\mathfrak{D}(\!|Arr|\!)$. *ntcf-arrow* ($Hom_{A.C\alpha}\mathfrak{D}(f,-)$))
and *Yoneda-functor* $\alpha\ \mathfrak{D}(\!|HomDom|\!)$ = *op-cat* $\mathfrak{D}$
and *Yoneda-functor* $\alpha\ \mathfrak{D}(\!|HomCod|\!)$ = *cat-FUNCT* $\alpha\ \mathfrak{D}$ (*cat-Set* α)
unfolding *Yoneda-functor-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

29.21.2 Object map

mk-VLambda *Yoneda-functor-components*(*1*)
|*vsv Yoneda-functor-ObjMap-vsv*[*cat-cs-intros*]|
|*vdomain Yoneda-functor-ObjMap-vdomain*[*cat-cs-simps*]|
|*app Yoneda-functor-ObjMap-app*[*cat-cs-simps*]|

lemma (**in** *category*) *Yoneda-functor-ObjMap-vrange*:
$\mathcal{R}_\circ$ (*Yoneda-functor* $\alpha\ \mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_\circ$ *cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α)$(\!|Obj|\!)$
proof
(
unfold Yoneda-functor-components,
rule vrange-VLambda-vsubset,
unfold cat-op-simps
)
fix c **assume** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
with *category-axioms* **show**
cf-map $Hom_{O.C\alpha}\mathfrak{C}(c,-) \in_\circ$ *cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α)$(\!|Obj|\!)$
unfolding *cat-op-simps cat-FUNCT-components*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
qed

29.21.3 Arrow map

mk-VLambda *Yoneda-functor-components*(*2*)
|*vsv Yoneda-functor-ArrMap-vsv*[*cat-cs-intros*]|
|*vdomain Yoneda-functor-ArrMap-vdomain*[*cat-cs-simps*]|
|*app Yoneda-functor-ArrMap-app*[*cat-cs-simps*]|

lemma (**in** *category*) *Yoneda-functor-ArrMap-vrange*:
$\mathcal{R}_\circ$ (*Yoneda-functor* $\alpha\ \mathfrak{C}(\!|ArrMap|\!)$) $\subseteq_\circ$ *cat-FUNCT* $\alpha\ \mathfrak{C}$ (*cat-Set* α)$(\!|Arr|\!)$
proof
(
unfold Yoneda-functor-components,
rule vrange-VLambda-vsubset,
unfold cat-op-simps
)
fix f **assume** $f \in_\circ \mathfrak{C}(\!|Arr|\!)$

then obtain a b **where** f: $f : a \mapsto_{\mathfrak{C}} b$ **by** *auto*
define β **where** $\beta = \alpha + \omega$
have $\mathcal{Z}\beta$: $\mathcal{Z}\ \beta$ **and** $\alpha\beta$: $\alpha \in_{\circ} \beta$
by (*simp-all add*: $\mathcal{Z}$-α-$\alpha\omega$ $\mathcal{Z}$.*intro* $\mathcal{Z}$-*Limit*-$\alpha\omega$ $\mathcal{Z}$-ω-$\alpha\omega$ β-*def*)
from *tiny-category-cat-FUNCT category-axioms* $\mathcal{Z}\beta$ $\alpha\beta$ f **show**
ntcf-arrow $Hom_{A.C\alpha}\mathfrak{C}(f,-) \in_{\circ}$ *cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α)(|*Arr*|)
unfolding *cat-op-simps*
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-FUNCT-cs-intros*)
qed

29.21.4 The Yoneda Functor is a fully faithful functor

lemma (**in** *category*) *cat-Yoneda-functor-is-functor*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_{\circ} \beta$
shows *Yoneda-functor* α $\mathfrak{C}$: *op-cat* $\mathfrak{C} \mapsto\mapsto_{C.ff\beta}$ *cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α)
proof
(
intro
is-ff-functorI
is-ft-functorI'
is-fl-functorI'
vsubset-antisym
vsubsetI,
unfold cat-op-simps in-Hom-iff,
tactic‹*distinct-subgoals-tac*›
)

interpret *Set*: *category* α ‹*cat-Set* α› **by** (*rule category-cat-Set*)

let *?Yf* = ‹*Yoneda-functor* α $\mathfrak{C}$› **and** *?FUNCT* = ‹*cat-FUNCT* α $\mathfrak{C}$ (*cat-Set* α)›

show *Yf*: *?Yf* : *op-cat* $\mathfrak{C} \mapsto\mapsto_{C\beta}$ *?FUNCT*
proof(*intro is-functorI'*)
show *vfsequence ?Yf* **unfolding** *Yoneda-functor-def* **by** *simp*
from *assms* **have** *category* β $\mathfrak{C}$ **by** (*intro cat-category-if-ge-Limit*)
then show *category* β (*op-cat* $\mathfrak{C}$) **by** (*intro category.category-op*)
from *assms* **show** *category* β *?FUNCT*
by
(
cs-concl **cs-shallow**
cs-intro: *cat-small-cs-intros tiny-category-cat-FUNCT*
)
show *vcard ?Yf* = $\mathcal{4}_{\mathbb{N}}$
unfolding *Yoneda-functor-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*?Yf*(|*ObjMap*|)) $\subseteq_{\circ}$ *?FUNCT*(|*Obj*|)
by (*rule Yoneda-functor-ObjMap-vrange*)
show
?Yf(|*ArrMap*|)(|*f*|) : *?Yf*(|*ObjMap*|)(|*a*|) $\mapsto_{cat\text{-}FUNCT\ \alpha\ \mathfrak{C}\ (cat\text{-}Set\ \alpha)}$ *?Yf*(|*ObjMap*|)(|*b*|)
if $f : a \mapsto_{op\text{-}cat\ \mathfrak{C}} b$ **for** a b f
using *that category-axioms*
unfolding *cat-op-simps*
by
(
cs-concl
cs-simp: *cat-cs-simps cat-op-simps*
cs-intro: *cat-cs-intros cat-FUNCT-cs-intros*
)
show *?Yf*(|*ArrMap*|)(|$g \circ_{A\ op\text{-}cat\ \mathfrak{C}} f$|) =

```
    ?Yf(|ArrMap|)(|g|) ∘A ?FUNCT ?Yf(|ArrMap|)(|f|)
    if g : b ↦op-cat 𝔈 c and f : a ↦op-cat 𝔈 b for b c g a f
    using that category-axioms
    unfolding cat-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
  show ?Yf(|ArrMap|)(|op-cat 𝔈(|CId|)(|c|)|) = ?FUNCT(|CId|)(|?Yf(|ObjMap|)(|c|)|)
    if c ∈∘ op-cat 𝔈(|Obj|) for c
    using that category-axioms
    unfolding cat-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
          cs-intro: cat-cs-intros cat-FUNCT-cs-intros
      )
qed (auto simp: assms(1) Yoneda-functor-components 𝒵.intro 𝒵-Limit-αω 𝒵-ω-αω)

interpret Yf: is-functor β ‹op-cat 𝔈› ‹?FUNCT› ‹?Yf› by (rule Yf)

show v11 (?Yf(|ArrMap|) ↾l∘ Hom 𝔈 b a)
  if a ∈∘ 𝔈(|Obj|) and b ∈∘ 𝔈(|Obj|) for a b
proof–
  from that have dom-Y-ba: 𝒟∘ (?Yf(|ArrMap|) ↾l∘ Hom 𝔈 b a) = Hom 𝔈 b a
    by
      (
        fastforce simp:
          cat-op-simps
          in-Hom-iff vdomain-vlrestriction Yoneda-functor-components
      )

  show v11 (?Yf(|ArrMap|) ↾l∘ Hom 𝔈 b a)
  proof(intro vsv.vsv-valeq-v11I, unfold dom-Y-ba in-Hom-iff)
    fix g f assume prems:
      g : b ↦𝔈 a
      f : b ↦𝔈 a
      (?Yf(|ArrMap|) ↾l∘ Hom 𝔈 b a)(|g|) = (?Yf(|ArrMap|) ↾l∘ Hom 𝔈 b a)(|f|)
    from
      prems(3) category-axioms prems(1,2) Yoneda-functor-ArrMap-vsv[of α 𝔈]
    have Hom_{A.Cα}𝔈(g,−) = Hom_{A.Cα}𝔈(f,−)
      by
        (
          cs-prems cs-shallow
            cs-simp: V-cs-simps cat-cs-simps cat-op-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros
        )
    from this prems(1,2) show g = f by (rule cat-ntcf-Hom-snd-inj)
  qed (auto simp: Yoneda-functor-components)
qed

fix a b assume prems: a ∈∘ 𝔈(|Obj|) b ∈∘ 𝔈(|Obj|)
show 𝔑 : ?Yf(|ObjMap|)(|a|) ↦cat-FUNCT α 𝔈 (cat-Set α) ?Yf(|ObjMap|)(|b|)
  if 𝔑 ∈∘ ?Yf(|ArrMap|) ‘∘ Hom 𝔈 b a for 𝔑
proof–
```

```
    from that obtain f where ?Yf(|ArrMap|)(|f|) = 𝔑 and f: f : b ↦_𝔈 a
      by (force elim!: Yf.ArrMap.vsv-vimageE)
    then have 𝔑-def: 𝔑 = ntcf-arrow Hom_{A.Cα}𝔈(f,-)
      unfolding
        Yoneda-functor-ArrMap-app[
          unfolded cat-op-simps, OF cat-is-arrD(1)[OF f]
          ]
      by (simp add: cat-cs-simps cat-op-simps cat-cs-intros)
    from category-axioms f show ?thesis
      unfolding 𝔑-def
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros cat-op-intros cat-FUNCT-cs-intros
        )
  qed
  show 𝔑 ∈₀ ?Yf(|ArrMap|) ‘₀ Hom 𝔈 b a
    if 𝔑 : ?Yf(|ObjMap|)(|a|) ↦_{cat-FUNCT α 𝔈 (cat-Set α)} ?Yf(|ObjMap|)(|b|) for 𝔑
  proof–
    note 𝔑 = cat-FUNCT-is-arrD[OF that]
    from 𝔑(1) category-axioms prems have ntcf-𝔑:
      ntcf-of-ntcf-arrow 𝔈 (cat-Set α) 𝔑 :
        Hom_{O.Cα}𝔈(a,-) ↦_{CF} Hom_{O.Cα}𝔈(b,-) : 𝔈 ↦↦_{Cα} cat-Set α
      by (subst (asm) 𝔑(3), use nothing in ‹subst (asm) 𝔑(4)›)
        (
          cs-prems cs-shallow
            cs-simp: cat-cs-simps cat-FUNCT-cs-simps
            cs-intro: cat-cs-intros cat-op-intros cat-FUNCT-cs-intros
        )
    from cat-ntcf-Hom-snd-is-ntcf-Hom-snd-unique(1,2)[OF prems ntcf-𝔑] obtain f
      where f: f : b ↦_𝔈 a
        and 𝔑-def: ntcf-of-ntcf-arrow 𝔈 (cat-Set α) 𝔑 = Hom_{A.Cα}𝔈(f,-)
      by auto
    from 𝔑(2) f show 𝔑 ∈₀ Yoneda-functor α 𝔈(|ArrMap|) ‘₀ Hom 𝔈 b a
      unfolding 𝔑-def
      by (intro Yf.ArrMap.vsv-vimage-eqI[of f])
        (
          cs-concl cs-shallow
            cs-simp: cat-cs-simps cs-intro: cat-cs-intros cat-op-intros
        )+
  qed
qed
```

30 Orders

30.1 Background

named-theorems *cat-order-cs-simps*
named-theorems *cat-order-cs-intros*

30.2 Preorder category

See Chapter I-2 in [7].

locale *cat-preorder* = *category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *cat-peo*:
    $[\![\ a \in_\circ \mathfrak{C}(\!|Obj|\!);\ b \in_\circ \mathfrak{C}(\!|Obj|\!)\ ]\!] \Longrightarrow$
      $(\exists f.\ Hom\ \mathfrak{C}\ a\ b = set\ \{f\}) \vee (Hom\ \mathfrak{C}\ a\ b = 0)$

Rules.

lemma (**in** *cat-preorder*) *cat-preorder-axioms'*[*cat-order-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *cat-preorder* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule cat-preorder-axioms*)

mk-ide rf *cat-preorder-def*[*unfolded cat-preorder-axioms-def*]
  |*intro cat-preorderI*|
  |*dest cat-preorderD*[*dest*]|
  |*elim cat-preorderE*[*elim*]|

lemmas [*cat-order-cs-intros*] = *cat-preorderD*(*1*)

Elementary properties.

lemma (**in** *cat-preorder*) *cat-peo-HomE*:
  assumes $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
  obtains *f* **where** ‹*Hom* $\mathfrak{C}$ *a b* = *set* {*f*}› | ‹*Hom* $\mathfrak{C}$ *a b* = *0*›
  using *cat-peo*[*OF assms*] **by** *auto*

lemma (**in** *cat-preorder*) *cat-peo-is-thin-category*:
  — The statement of the lemma appears in nLab [1][16].
  assumes $f : a \mapsto_{\mathfrak{C}} b$ **and** $g : a \mapsto_{\mathfrak{C}} b$
  shows *f* = *g*
proof–
  note *f* = *cat-is-arrD*[*OF assms*(*1*)]
  from *assms* **have** *Hom* $\mathfrak{C}$ *a b* ≠ *0* **by** (*metis HomI eq0-iff*)
  with *cat-peo-HomE*[*OF f*(*2,3*)] **obtain** *h* **where** *Hom* $\mathfrak{C}$ *a b* = *set* {*h*} **by** *auto*
  moreover from *assms* **have** *f* ∈∘ *Hom* $\mathfrak{C}$ *a b* **and** *g* ∈∘ *Hom* $\mathfrak{C}$ *a b* **by** *auto*
  ultimately have *h* = *f* **and** *h* = *g* **by** *auto*
  then show *?thesis* **by** *auto*
qed

30.3 Order relation

30.3.1 Definition and elementary properties

definition *is-le* :: *V* ⇒ *V* ⇒ *V* ⇒ *bool* (**infix** ‹≤$_{O1}$› *50*)
  where $a \leq_{O\mathfrak{C}} b \longleftrightarrow Hom\ \mathfrak{C}\ a\ b \neq 0$

Rules.

mk-ide *is-le-def*

[16] https://ncatlab.org/nlab/show/preorder

|*intro is-leI*|
|*dest is-leD*[*dest*]|
|*elim is-leE*[*elim*]|

Elementary properties.

lemma (in *cat-preorder*) *cat-peo-is-le*[*cat-order-cs-intros*]:
assumes $f : a \mapsto_{\mathfrak{C}} b$
shows $a \leq_{O\mathfrak{C}} b$
using *assms* **by** (*force intro*: *is-leI*)

lemmas [*cat-order-cs-intros*] = *cat-preorder.cat-peo-is-le*

lemma (in *cat-preorder*) *cat-peo-is-le-ex1*:
assumes $a \leq_{O\mathfrak{C}} b$ **and** $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $\exists!f.\ f : a \mapsto_{\mathfrak{C}} b$
proof–
from *assms* **have** *Hom* $\mathfrak{C}$ a $b \neq 0$ **by** *auto*
with *assms cat-peo* **obtain** f **where** *Hom-ab*: *Hom* $\mathfrak{C}$ a b = *set* $\{f\}$ **by** *meson*
show $\exists!f.\ f : a \mapsto_{\mathfrak{C}} b$
proof(*intro ex1I*)
from *Hom-ab* **show** $f : a \mapsto_{\mathfrak{C}} b$ **by** *auto*
fix g **assume** $g : a \mapsto_{\mathfrak{C}} b$
with *Hom-ab* **show** $g = f$ **by** *auto*
qed
qed

lemma (in *cat-preorder*) *cat-peo-is-le-ex*[*elim*]:
assumes $a \leq_{O\mathfrak{C}} b$ **and** $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
obtains f **where** $f : a \mapsto_{\mathfrak{C}} b$
using *cat-peo-is-le-ex1*[*OF assms*] *that* **by** *clarsimp*

30.3.2 Order relation on a preorder category is a preorder

lemma (in *cat-preorder*) *is-le-refl*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
shows $a \leq_{O\mathfrak{C}} a$
proof(*intro is-leI*)
from *assms* **have** $\mathfrak{C}(\!|CId|\!)(\!|a|\!) \in_{\circ} Hom\ \mathfrak{C}\ a\ a$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
then show *Hom* $\mathfrak{C}$ a $a \neq 0$ **by** *force*
qed

lemma (in *cat-preorder*) *is-le-trans*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $b \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $c \in_{\circ} \mathfrak{C}(\!|Obj|\!)$
and $a \leq_{O\mathfrak{C}} b$
and $b \leq_{O\mathfrak{C}} c$
shows $a \leq_{O\mathfrak{C}} c$
proof(*intro is-leI*)
from *assms* **obtain** f **where** *f*: $f : a \mapsto_{\mathfrak{C}} b$ **by** *auto*
from *assms* **obtain** g **where** *g*: $g : b \mapsto_{\mathfrak{C}} c$ **by** *auto*
from f g **have** $g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
then show *Hom* $\mathfrak{C}$ a $c \neq 0$ **by** *force*
qed

30.4 Partial order category

See Chapter I-2 in [7].

locale *cat-partial-order* = *cat-preorder* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *cat-po*: $[\![\ a \in_\circ \mathfrak{C}(\!|Obj|\!);\ b \in_\circ \mathfrak{C}(\!|Obj|\!);\ a \le_{O\mathfrak{C}} b;\ b \le_{O\mathfrak{C}} a\ ]\!] \Longrightarrow a = b$

Rules.

lemma (**in** *cat-partial-order*) *cat-partial-order-axioms'*[*cat-order-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *cat-partial-order* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule cat-partial-order-axioms*)

mk-ide rf *cat-partial-order-def*[*unfolded cat-partial-order-axioms-def*]
  |*intro cat-partial-orderI*|
  |*dest cat-partial-orderD*[*dest*]|
  |*elim cat-partial-orderE*[*elim*]|

lemmas [*cat-order-cs-intros*] = *cat-partial-orderD*(*1*)

30.5 Linear order category

See Chapter I-2 in [7].

locale *cat-linear-order* = *cat-partial-order* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *cat-lo*: $[\![\ a \in_\circ \mathfrak{C}(\!|Obj|\!);\ b \in_\circ \mathfrak{C}(\!|Obj|\!)\ ]\!] \Longrightarrow a \le_{O\mathfrak{C}} b \vee b \le_{O\mathfrak{C}} a$

Rules.

lemma (**in** *cat-linear-order*) *cat-linear-order-axioms'*[*cat-order-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *cat-linear-order* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule cat-linear-order-axioms*)

mk-ide rf *cat-linear-order-def*[*unfolded cat-linear-order-axioms-def*]
  |*intro cat-linear-orderI*|
  |*dest cat-linear-orderD*[*dest*]|
  |*elim cat-linear-orderE*[*elim*]|

lemmas [*cat-order-cs-intros*] = *cat-linear-orderD*(*1*)

30.6 Preorder functor

30.6.1 Definition and elementary properties

See [1][17].

locale *is-preorder-functor* =
  is-functor α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ + *HomDom*: *cat-preorder* α $\mathfrak{A}$ + *HomCod*: *cat-preorder* α $\mathfrak{B}$
  for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

syntax *-is-preorder-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\le_{C.PEO1}$ -)$\rangle$ [*51*, *51*, *51*] *51*)
syntax-consts *-is-preorder-functor* $\rightleftharpoons$ *is-preorder-functor*
translations $\mathfrak{F} : \mathfrak{A} \le_{C.PEO\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-preorder-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

Rules.

lemma (**in** *is-preorder-functor*) *is-preorder-functor-axioms'*[*cat-order-cs-intros*]:

[17] https://ncatlab.org/nlab/show/monotone+function

assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto_{C.PEO_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-preorder-functor-axioms*)

mk-ide rf *is-preorder-functor-def*
|*intro is-preorder-functorI*|
|*dest is-preorder-functorD*[*dest*]|
|*elim is-preorder-functorE*[*elim*]|

lemmas [*cat-order-cs-intros*] = *is-preorder-functorD*

30.6.2 A preorder functor is a faithful functor

sublocale *is-preorder-functor* ⊆ *is-ft-functor*
proof(*intro is-ft-functorI'*)
fix *a b* **assume** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ $b \in_\circ \mathfrak{A}(\!|Obj|\!)$
show *v11* $(\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
proof
(
intro vsv.vsv-valeq-v11I,
unfold vdomain-vlrestriction cat-cs-simps vintersection-iff;
(*elim conjE*)*?*
)
fix *g f* **assume** $g : a \mapsto_{\mathfrak{A}} b$ $f : a \mapsto_{\mathfrak{A}} b$
then show $g = f$ **by** (*auto simp*: *HomDom.cat-peo-is-thin-category*)
qed *simp*
qed (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)

lemmas (**in** *is-preorder-functor*) *is-preorder-functor-is-ft-functor* =
is-ft-functor-axioms

lemmas [*cat-order-cs-intros*] =
is-preorder-functor.is-preorder-functor-is-ft-functor

30.6.3 A preorder functor is a monotone function

lemma (**in** *is-preorder-functor*) *cat-peo*:
— Based on [1][18]
assumes $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $a \leq_{O\mathfrak{A}} b$
shows $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \leq_{O\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
proof–
from *assms* **obtain** f **where** $f : a \mapsto_{\mathfrak{A}} b$ **by** *auto*
then have $\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
by (*simp add*: *cf-ArrMap-is-arr*)
then show $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \leq_{O\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
by (*cs-concl* **cs-shallow cs-intro**: *cat-order-cs-intros*)
qed

30.6.4 Composition of preorder functors

lemma *cf-comp-is-preorder-functor*[*cat-order-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto_{C.PEO\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto_{C.PEO\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{CF} \mathfrak{F} : \mathfrak{A} \mapsto_{C.PEO\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{G}$: *is-preorder-functor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(*1*))
interpret $\mathfrak{F}$: *is-preorder-functor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(*2*))

[18] https://ncatlab.org/nlab/show/monotone+function

show *?thesis*
by (*intro is-preorder-functorI*)
(*cs-concl* **cs-intro**: *cat-cs-intros cat-order-cs-intros*)+
qed

lemma (**in** *cat-preorder*) *cat-peo-cf-is-preorder-functor*:
cf-id $\mathfrak{C} : \mathfrak{C} \leq_{C.PEO\alpha} \mathfrak{C}$
by (*intro is-preorder-functorI*)
(*cs-concl* **cs-intro**: *cat-cs-intros cat-order-cs-intros*)+

lemma (**in** *cat-preorder*) *cat-peo-cf-is-preorder-functor'*[*cat-order-cs-intros*]:
assumes $\mathfrak{A}' = \mathfrak{C}$ **and** $\mathfrak{B}' = \mathfrak{C}$
shows *cf-id* $\mathfrak{C} : \mathfrak{A}' \leq_{C.PEO\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule cat-peo-cf-is-preorder-functor*)

lemmas [*cat-order-cs-intros*] = *cat-preorder.cat-peo-cf-is-preorder-functor'*

31 Smallness for orders

31.1 Background

named-theorems *cat-small-order-cs-simps*
named-theorems *cat-small-order-cs-intros*

31.2 Tiny preorder category

locale *cat-tiny-preorder* = *tiny-category* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
assumes *cat-tiny-peo*:
$[\![a \in_\circ \mathfrak{C}(\!|Obj|\!);\ b \in_\circ \mathfrak{C}(\!|Obj|\!)]\!] \Longrightarrow$
$(\exists f.\ Hom\ \mathfrak{C}\ a\ b = set\ \{f\}) \vee (Hom\ \mathfrak{C}\ a\ b = 0)$

Rules.

lemma (**in** *cat-tiny-preorder*) *cat-tiny-preorder-axioms'*[*cat-order-cs-intros*]:
assumes $\alpha' = \alpha$
shows *cat-tiny-preorder* α' $\mathfrak{C}$
unfolding *assms* **by** (*rule cat-tiny-preorder-axioms*)

mk-ide rf *cat-tiny-preorder-def*[*unfolded cat-tiny-preorder-axioms-def*]
|*intro cat-tiny-preorderI*|
|*dest cat-tiny-preorderD*[*dest*]|
|*elim cat-tiny-preorderE*[*elim*]|

lemmas [*cat-small-order-cs-intros*] = *cat-tiny-preorderD*(*1*)

Tiny preorder is a preorder.

sublocale *cat-tiny-preorder* ⊆ *cat-preorder*
by (*intro cat-preorderI cat-tiny-peo category-axioms*) *simp-all*

lemmas (**in** *cat-tiny-preorder*) *cat-tiny-peo-is-cat-preoder* = *cat-preorder-axioms*

lemmas [*cat-small-order-cs-intros*] =
cat-tiny-preorder.cat-tiny-peo-is-cat-preoder

31.3 Tiny partial order category

locale *cat-tiny-partial-order* = *cat-tiny-preorder* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
assumes *cat-tiny-po*:

$[\![\ a \in_{\circ} \mathfrak{C}(\!|Obj|\!);\ b \in_{\circ} \mathfrak{C}(\!|Obj|\!);\ a \le_{O\mathfrak{C}} b;\ b \le_{O\mathfrak{C}} a\ ]\!] \Longrightarrow a = b$

Rules.

lemma (**in** *cat-tiny-partial-order*)
cat-tiny-partial-order-axioms'[*cat-order-cs-intros*]:
assumes $\alpha' = \alpha$
shows *cat-tiny-partial-order* $\alpha'\ \mathfrak{C}$
unfolding *assms* **by** (*rule cat-tiny-partial-order-axioms*)

mk-ide rf *cat-tiny-partial-order-def*[*unfolded cat-tiny-partial-order-axioms-def*]
|*intro cat-tiny-partial-orderI*|
|*dest cat-tiny-partial-orderD*[*dest*]|
|*elim cat-tiny-partial-orderE*[*elim*]|

lemmas [*cat-small-order-cs-intros*] = *cat-tiny-partial-orderD*(*1*)

Tiny partial order is a partial order.

sublocale *cat-tiny-partial-order* ⊆ *cat-partial-order*
by (*intro cat-partial-orderI cat-tiny-po cat-preorder-axioms*) *simp-all*

lemmas (**in** *cat-tiny-preorder*) *cat-tiny-po-is-cat-preoder* = *cat-preorder-axioms*

lemmas [*cat-small-order-cs-intros*] =
cat-tiny-preorder.cat-tiny-peo-is-cat-preoder

lemma *cat-tiny-partial-orderI'*:
assumes *tiny-category* $\alpha\ \mathfrak{C}$
and *cat-partial-order* $\alpha\ \mathfrak{C}$
shows *cat-tiny-partial-order* $\alpha\ \mathfrak{C}$
proof–
interpret *tiny-category* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*1*))
interpret *cat-partial-order* $\alpha\ \mathfrak{C}$ **by** (*rule assms*(*2*))
show *?thesis*
by (*intro cat-tiny-partial-orderI cat-tiny-preorderI assms*(*1*) *cat-po cat-peo*)
qed

31.4 Tiny linear order category

locale *cat-tiny-linear-order* = *cat-tiny-partial-order* $\alpha\ \mathfrak{C}$ **for** $\alpha\ \mathfrak{C}$ +
assumes *cat-tiny-lo*: $[\![\ a \in_{\circ} \mathfrak{C}(\!|Obj|\!);\ b \in_{\circ} \mathfrak{C}(\!|Obj|\!)\ ]\!] \Longrightarrow a \le_{O\mathfrak{C}} b \vee b \le_{O\mathfrak{C}} a$

Rules.

lemma (**in** *cat-tiny-linear-order*)
cat-tiny-linear-order-axioms'[*cat-order-cs-intros*]:
assumes $\alpha' = \alpha$
shows *cat-tiny-linear-order* $\alpha'\ \mathfrak{C}$
unfolding *assms* **by** (*rule cat-tiny-linear-order-axioms*)

mk-ide rf *cat-tiny-linear-order-def*[*unfolded cat-tiny-linear-order-axioms-def*]
|*intro cat-tiny-linear-orderI*|
|*dest cat-tiny-linear-orderD*[*dest*]|
|*elim cat-tiny-linear-orderE*[*elim*]|

lemmas [*cat-small-order-cs-intros*] = *cat-tiny-linear-orderD*(*1*)

Tiny linear order is a partial order.

sublocale *cat-tiny-linear-order* ⊆ *cat-linear-order*

by (*intro cat-linear-orderI cat-tiny-lo cat-partial-order-axioms*) *simp-all*

lemmas (**in** *cat-tiny-linear-order*) *cat-tiny-lo-is-cat-partial-order* =
cat-linear-order-axioms

lemmas [*cat-small-order-cs-intros*] =
cat-tiny-linear-order.cat-tiny-lo-is-cat-partial-order

lemma *cat-tiny-linear-orderI'*:
assumes *tiny-category* α $\mathfrak{C}$ **and** *cat-linear-order* α $\mathfrak{C}$
shows *cat-tiny-linear-order* α $\mathfrak{C}$
proof–
interpret *tiny-category* α $\mathfrak{C}$ **by** (*rule assms*(*1*))
interpret *cat-linear-order* α $\mathfrak{C}$ **by** (*rule assms*(*2*))
show *?thesis*
by
(
intro
assms(*1*)
cat-tiny-linear-orderI
cat-tiny-partial-orderI'
cat-partial-order-axioms
cat-lo
)
qed

31.5 Tiny preorder functor

locale *is-tiny-preorder-functor* =
is-functor α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
HomDom: *cat-tiny-preorder* α $\mathfrak{A}$ +
HomCod: *cat-tiny-preorder* α $\mathfrak{B}$
for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

syntax *-is-tiny-preorder-functor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\leq_{C.PEO.tiny_1}$ -)› [*51*, *51*, *51*] *51*)
syntax-consts *-is-tiny-preorder-functor* $\rightleftharpoons$ *is-tiny-preorder-functor*
translations $\mathfrak{F} : \mathfrak{A} \leq_{C.PEO.tiny\alpha} \mathfrak{B} \rightleftharpoons$
CONST is-tiny-preorder-functor α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

Rules.

lemma (**in** *is-tiny-preorder-functor*)
is-tiny-preorder-functor-axioms'[*cat-order-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \leq_{C.PEO.tiny_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-tiny-preorder-functor-axioms*)

mk-ide rf *is-tiny-preorder-functor-def*
|*intro is-tiny-preorder-functorI*|
|*dest is-tiny-preorder-functorD*[*dest*]|
|*elim is-tiny-preorder-functorE*[*elim*]|

lemmas [*cat-small-order-cs-intros*] = *is-tiny-preorder-functorD*(*1*)

Tiny preorder functor is a tiny functor

sublocale *is-tiny-preorder-functor* ⊆ *is-tiny-functor*
by
(

intro
is-tiny-functorI'
is-functor-axioms
HomDom.tiny-category-axioms
HomCod.tiny-category-axioms
)

32 Ordinal numbers

32.1 Background

The content of this section is based on the treatment of the ordinal numbers from the perspective of category theory as exposed, for example, in Chapter I-2 in [7].

named-theorems *cat-ordinal-cs-simps*
named-theorems *cat-ordinal-cs-intros*

32.2 Arrows associated with an ordinal number

definition *ordinal-arrs* :: $V \Rightarrow V$
where *ordinal-arrs* $A \equiv$ *set* $\{[a, b]_\circ \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge a \leq b\}$

lemma *small-ordinal-arrs*[*simp*]:
small $\{[a, b]_\circ \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge a \leq b\}$
by (*rule down*[**where** x=$\langle A \times_\bullet A\rangle$]) *auto*

Rules.

lemma *ordinal-arrsI*[*cat-ordinal-cs-intros*]:
assumes $x = [a, b]_\circ$ **and** $a \in_\circ A$ **and** $b \in_\circ A$ **and** $a \leq b$
shows $x \in_\circ$ *ordinal-arrs* A
using *assms* **unfolding** *ordinal-arrs-def* **by** *auto*

lemma *ordinal-arrsD*[*dest*]:
assumes $[a, b]_\circ \in_\circ$ *ordinal-arrs* A
shows $a \in_\circ A$ **and** $b \in_\circ A$ **and** $a \leq b$
using *assms* **unfolding** *ordinal-arrs-def* **by** *auto*

lemma *ordinal-arrsE*[*elim*]:
assumes $x \in_\circ$ *ordinal-arrs* A
obtains $a\ b$ **where** $a \in_\circ A$ **and** $b \in_\circ A$ **and** $a \leq b$ **and** $x = [a, b]_\circ$
using *assms* **unfolding** *ordinal-arrs-def* **by** *clarsimp*

32.3 Composable arrows

abbreviation *ordinal-composable* :: $V \Rightarrow V$
where *ordinal-composable* $A \equiv$ *set*
{
$[[b, c]_\circ, [a, b]_\circ]_\circ \mid a\ b\ c.$
$a \in_\circ A \wedge b \in_\circ A \wedge c \in_\circ A \wedge a \leq b \wedge b \leq c$
}

lemma *small-ordinal-composable*[*simp*]:
small
{
$[[b, c]_\circ, [a, b]_\circ]_\circ \mid a\ b\ c.$
$a \in_\circ A \wedge b \in_\circ A \wedge c \in_\circ A \wedge a \leq b \wedge b \leq c$
}

by (*rule down*[**where** x=‹$(A \times_\bullet A) \times_\bullet (A \times_\bullet A)$›]) *auto*

Rules.

lemma *ordinal-composableI*[*cat-ordinal-cs-intros*]:
assumes $x = [[b,\ c]_\circ,\ [a,\ b]_\circ]_\circ$
and $a \in_\circ A$
and $b \in_\circ A$
and $c \in_\circ A$
and $a \leq b$
and $b \leq c$
shows $x \in_\circ$ *ordinal-composable* A
using *assms* **by** *auto*

lemma *ordinal-composableD*[*dest*]:
assumes $[[b,\ c]_\circ,\ [a,\ b]_\circ]_\circ \in_\circ$ *ordinal-composable* A
shows $a \in_\circ A$ **and** $b \in_\circ A$ **and** $c \in_\circ A$ **and** $a \leq b$ **and** $b \leq c$
using *assms* **by** *auto*

lemma *ordinal-composableE*[*elim*]:
assumes $x \in_\circ$ *ordinal-composable* A
obtains $a\ b\ c$
where $x = [[b,\ c]_\circ,\ [a,\ b]_\circ]_\circ$
and $a \in_\circ A$
and $b \in_\circ A$
and $c \in_\circ A$
and $a \leq b$
and $b \leq c$
using *assms* **by** *clarsimp*

32.4 Ordinal number as a category

32.4.1 Definition and elementary properties

definition *cat-ordinal* :: $V \Rightarrow V$
where *cat-ordinal* A =
[
A,
ordinal-arrs A,
$(\lambda f \in_\circ$ *ordinal-arrs* A. $f(\!|0|\!))$,
$(\lambda f \in_\circ$ *ordinal-arrs* A. $f(\!|1_{\mathbb{N}}|\!))$,
$(\lambda gf \in_\circ$ *ordinal-composable* A. $[gf(\!|1_{\mathbb{N}}|\!)(\!|0|\!),\ gf(\!|0|\!)(\!|1_{\mathbb{N}}|\!)]_\circ)$,
$(\lambda x \in_\circ A.\ [x,\ x]_\circ)$
]$_\circ$

Components.

lemma *cat-ordinal-components*:
shows [*cat-ordinal-cs-simps*]: *cat-ordinal* $A(\!|Obj|\!) = A$
and [*cat-ordinal-cs-simps*]: *cat-ordinal* $A(\!|Arr|\!) =$ *ordinal-arrs* A
and *cat-ordinal* $A(\!|Dom|\!) = (\lambda f \in_\circ$ *ordinal-arrs* A. $f(\!|0|\!))$
and *cat-ordinal* $A(\!|Cod|\!) = (\lambda f \in_\circ$ *ordinal-arrs* A. $f(\!|1_{\mathbb{N}}|\!))$
and *cat-ordinal* $A(\!|Comp|\!) =$
$(\lambda gf \in_\circ$ *ordinal-composable* A. $[gf(\!|1_{\mathbb{N}}|\!)(\!|0|\!),\ gf(\!|0|\!)(\!|1_{\mathbb{N}}|\!)]_\circ)$
and *cat-ordinal* $A(\!|CId|\!) = (\lambda x \in_\circ A.\ [x,\ x]_\circ)$
unfolding *cat-ordinal-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

32.4.2 Domain

mk-VLambda *cat-ordinal-components*(*3*)

```
  |vsv cat-ordinal-Dom-vsv[cat-ordinal-cs-intros]|
  |vdomain
    cat-ordinal-Dom-vdomain[
      folded cat-ordinal-components, cat-ordinal-cs-simps
      ]
  |

lemma cat-ordinal-Dom-app[cat-ordinal-cs-simps]:
  assumes x ∈∘ cat-ordinal A(|Arr|) and x = [a, b]∘
  shows cat-ordinal A(|Dom|)(|x|) = a
  using assms(1) unfolding assms(2) cat-ordinal-components by auto

lemma cat-ordinal-Dom-vrange: ℛ∘ (cat-ordinal A(|Dom|)) ⊆∘ cat-ordinal A(|Obj|)
  unfolding cat-ordinal-components
  by (intro vrange-VLambda-vsubset, elim ordinal-arrsE) simp
```

32.4.3 Codomain

```
mk-VLambda cat-ordinal-components(4)
  |vsv cat-ordinal-Cod-vsv[cat-ordinal-cs-intros]|
  |vdomain
    cat-ordinal-Cod-vdomain[
      folded cat-ordinal-components, cat-ordinal-cs-simps
      ]
  |

lemma cat-ordinal-Cod-app[cat-ordinal-cs-simps]:
  assumes x ∈∘ cat-ordinal A(|Arr|) and x = [a, b]∘
  shows cat-ordinal A(|Cod|)(|x|) = b
  using assms(1)
  unfolding assms(2) cat-ordinal-components
  by (auto simp: nat-omega-simps)

lemma cat-ordinal-Cod-vrange: ℛ∘ (cat-ordinal A(|Cod|)) ⊆∘ cat-ordinal A(|Obj|)
  unfolding cat-ordinal-components
  by (intro vrange-VLambda-vsubset, elim ordinal-arrsE)
    (simp add: nat-omega-simps)
```

32.4.4 Arrow with a domain and a codomain

Rules.

```
lemma cat-ordinal-is-arrI[cat-ordinal-cs-intros]:
  assumes a ∈∘ A and b ∈∘ A and a ≤ b and f = [a, b]∘
  shows f : a ↦_cat-ordinal A b
proof(intro is-arrI, unfold cat-ordinal-components(2))
  from assms show f ∈∘ ordinal-arrs A by (intro ordinal-arrsI)
  then show cat-ordinal A(|Dom|)(|f|) = a cat-ordinal A(|Cod|)(|f|) = b
    by (cs-concl cs-simp: cat-ordinal-cs-simps assms(4))+
qed

lemma cat-ordinal-is-arrD[dest]:
  assumes f : a ↦_cat-ordinal A b
  shows a ∈∘ A and b ∈∘ A and a ≤ b and f = [a, b]∘
proof–
  note f = is-arrD[OF assms, unfolded cat-ordinal-components(2)]
  from f(1) obtain a' b'
    where a': a' ∈∘ A
      and b': b' ∈∘ A
```

```
    and a'b': a' ≤ b'
    and f-def: f = [a', b']∘
  unfolding ordinal-arrs-def by clarsimp
from f(2) a' b' a'b' have a'-def: a' = a
  by
    (
      cs-prems cs-shallow
        cs-simp: cat-ordinal-cs-simps f-def cs-intro: cat-ordinal-cs-intros
    )
from f(3) a' b' a'b' have b'-def: b' = b
  by
    (
      cs-prems cs-shallow
        cs-simp: cat-ordinal-cs-simps f-def cs-intro: cat-ordinal-cs-intros
    )
from a' b' a'b' f-def show a ∈∘ A and b ∈∘ A and a ≤ b and f = [a, b]∘
  unfolding a'-def b'-def by auto
qed

lemma cat-ordinal-is-arrE[elim]:
  assumes f : a ↦_cat-ordinal A b
  obtains a ∈∘ A and b ∈∘ A and a ≤ b and f = [a, b]∘
  using assms by auto
```

Elementary properties.

```
lemma cat-ordinal-is-arr-not:
  assumes ¬a ≤ b
  shows ¬f : a ↦_cat-ordinal A b
proof(rule ccontr, unfold not-not)
  assume f : a ↦_cat-ordinal A b
  with cat-ordinal-is-arrD(3)[OF this] assms show False by simp
qed

lemma cat-ordinal-is-arr-is-unique:
  assumes f : a ↦_cat-ordinal A b and g : a ↦_cat-ordinal A b
  shows f = g
  by
    (
      simp add:
        cat-ordinal-is-arrD(4)[OF assms(1)]
        cat-ordinal-is-arrD(4)[OF assms(2)]
    )

lemma cat-ordinal-Hom-ne:
  assumes f : a ↦_cat-ordinal A b
  shows Hom (cat-ordinal A) a b = set {f}
  using assms cat-ordinal-is-arr-is-unique
  by (intro vsubset-antisym vsubsetI) auto

lemma cat-ordinal-Hom-empty:
  assumes ¬a ≤ b
  shows Hom (cat-ordinal A) a b = 0
  using assms by (intro vsubset-antisym vsubsetI) auto

lemma cat-ordinal-inj:
  assumes cat-ordinal m = cat-ordinal n
  shows m = n
  by (metis assms cat-ordinal-components(1))
```

32.4.5 Composition

mk-VLambda *cat-ordinal-components*(*5*)
|*vsv cat-ordinal-Comp-vsv*[*cat-ordinal-cs-intros*]|
|*vdomain cat-ordinal-Comp-vdomain*[*folded cat-ordinal-components*, *cat-cs-simps*]|

lemma *cat-ordinal-Comp-app*[*cat-ordinal-cs-simps*]:
assumes $g : b \mapsto_{\text{cat-ordinal } A} c$ **and** $f : a \mapsto_{\text{cat-ordinal } A} b$
shows $g \circ_{A\,\text{cat-ordinal } A} f = [a, c]_\circ$
proof–
from *assms*(*2*) **have** *a*: $a \in_\circ A$
and *b*: $b \in_\circ A$
and *ab*: $a \le b$
and *f-def*: $f = [a, b]_\circ$
by *auto*
from *assms*(*1*) **have** *c*: $c \in_\circ A$ **and** *bc*: $b \le c$ **and** *g-def*: $g = [b, c]_\circ$
by *auto*
from *a b c ab bc* **have** $[g, f]_\circ \in_\circ$ *ordinal-composable A*
by (*cs-concl* **cs-shallow cs-simp**: *g-def f-def* **cs-intro**: *cat-ordinal-cs-intros*)
then show $g \circ_{A\,\text{cat-ordinal } A} f = [a, c]_\circ$
unfolding *cat-ordinal-components* **by** (*simp add*: *g-def f-def nat-omega-simps*)
qed

32.4.6 Identity

mk-VLambda *cat-ordinal-components*(*6*)
|*vsv cat-ordinal-CId-vsv*[*cat-ordinal-cs-intros*]|
|*vdomain cat-ordinal-CId-vdomain*[*cat-ordinal-cs-simps*]|
|*app cat-ordinal-CId-app*[*cat-ordinal-cs-simps*]|

32.4.7 Order relation

lemma *cat-ordinal-is-leD*[*dest*]:
assumes $a \le_{O\,\text{cat-ordinal } A} b$
shows $[a, b]_\circ : a \mapsto_{\text{cat-ordinal } A} b$
proof(*intro cat-ordinal-is-arrI*)
from *is-leD*[*OF assms*] **obtain** *f* **where** $f : a \mapsto_{\text{cat-ordinal } A} b$ **by** *fastforce*
then show $a \in_\circ A$ $b \in_\circ A$ $a \subseteq_\circ b$ **by** *auto*
qed *simp*

lemma *cat-ordinal-is-leE*[*elim*]:
assumes $a \le_{O\,\text{cat-ordinal } A} b$
obtains $[a, b]_\circ : a \mapsto_{\text{cat-ordinal } A} b$
using *assms* **by** *auto*

lemma *cat-ordinal-is-le-iff*:
$a \le_{O\,\text{cat-ordinal } A} b \longleftrightarrow [a, b]_\circ : a \mapsto_{\text{cat-ordinal } A} b$
using *cat-ordinal-is-leD cat-ordinal-is-leE*
by (*metis HomI is-le-def vempty-nin*)

32.4.8 Every ordinal number is a category

lemma (**in** $\mathcal{Z}$) *cat-linear-order-cat-ordinal*[*cat-ordinal-cs-intros*]:
assumes *Ord A* **and** $A \subseteq_\circ \alpha$
shows *cat-linear-order* α (*cat-ordinal A*)
proof(*intro cat-linear-orderI cat-partial-orderI cat-preorderI categoryI'*)
show *vfsequence* (*cat-ordinal A*) **unfolding** *cat-ordinal-def* **by** *auto*
show *vcard* (*cat-ordinal A*) = $6_{\mathbb{N}}$
unfolding *cat-ordinal-def* **by** (*simp add*: *nat-omega-simps*)

```
show ℛ∘ (cat-ordinal A(|Dom|)) ⊆∘ cat-ordinal A(|Obj|)
  by (rule cat-ordinal-Dom-vrange)
show ℛ∘ (cat-ordinal A(|Cod|)) ⊆∘ cat-ordinal A(|Obj|)
  by (rule cat-ordinal-Cod-vrange)
show (gf ∈∘ 𝒟∘ (cat-ordinal A(|Comp|))) ⟷
  (
    ∃g f b c a.
      gf = [g, f]∘ ∧ g : b ↦cat-ordinal A c ∧ f : a ↦cat-ordinal A b
  )
  for gf
  unfolding cat-ordinal-Comp-vdomain
proof
  assume gf ∈∘ ordinal-composable A
  then obtain a b c
    where gf-def: gf = [[b, c]∘, [a, b]∘]∘
      and a: a ∈∘ A
      and b: b ∈∘ A
      and c: c ∈∘ A
      and ab: a ≤ b
      and bc: b ≤ c
    by clarsimp
  from a b c ab bc show
    ∃g f b c a.
      gf = [g, f]∘ ∧ g : b ↦cat-ordinal A c ∧ f : a ↦cat-ordinal A b
    unfolding gf-def
    by (intro exI conjI)
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps cs-intro: cat-ordinal-cs-intros
      )+
next
  assume
    ∃g f b c a.
      gf = [g, f]∘ ∧
      g : b ↦cat-ordinal A c ∧
      f : a ↦cat-ordinal A b
  then obtain g f b c a
    where gf-def: gf = [g, f]∘
      and g: g : b ↦cat-ordinal A c
      and f: f : a ↦cat-ordinal A b
    by clarsimp
  note g = cat-ordinal-is-arrD[OF g]
  note f = cat-ordinal-is-arrD[OF f]
  from g(1–3) f(1–3) show gf ∈∘ ordinal-composable A
    unfolding gf-def g(4) f(4)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps cs-intro: cat-ordinal-cs-intros
      )
qed
show [cat-ordinal-cs-intros]: g ∘A cat-ordinal A f : a ↦cat-ordinal A c
  if g : b ↦cat-ordinal A c and f : a ↦cat-ordinal A b for b c g a f
proof–
  note g = cat-ordinal-is-arrD[OF that(1)]
  note f = cat-ordinal-is-arrD[OF that(2)]
  show ?thesis
  proof(intro cat-ordinal-is-arrI g(1,2) f(1,2), unfold g(4) f(4))
```

```
    from g(3) f(3) show a ⊆∘ c by auto
    from g(1,2,3) f(1,2,3) show  [b, c]∘ ∘A cat-ordinal A [a, b]∘ = [a, c]∘
      by
        (
          cs-concl cs-shallow
            cs-simp: cat-ordinal-cs-simps cs-intro: cat-ordinal-cs-intros
        )
  qed
qed
show
  h ∘A cat-ordinal A g ∘A cat-ordinal A f =
    h ∘A cat-ordinal A (g ∘A cat-ordinal A f)
  if h : c ↦cat-ordinal A d
    and g : b ↦cat-ordinal A c
    and f : a ↦cat-ordinal A b
  for c d h b g a f
proof–
  note h = cat-ordinal-is-arrD[OF that(1)]
  note g = cat-ordinal-is-arrD[OF that(2)]
  note f = cat-ordinal-is-arrD[OF that(3)]
  from that(1–3) h(1–3) g(1–4) f(1–3) show ?thesis
    unfolding h(4) g(4) f(4)
    by (cs-concl cs-simp: cat-ordinal-cs-simps cs-intro: cat-ordinal-cs-intros)
qed
show cat-ordinal A(|CId|)(|a|) : a ↦cat-ordinal A a
  if a ∈∘ cat-ordinal A(|Obj|) for a
proof–
  from that have a ∈∘ A unfolding cat-ordinal-components by simp
  then show cat-ordinal A(|CId|)(|a|) : a ↦cat-ordinal A a
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps
          cs-intro: cat-ordinal-cs-intros V-cs-intros
      )
qed
show cat-ordinal A(|CId|)(|b|) ∘A cat-ordinal A f = f
  if f : a ↦cat-ordinal A b for a b f
proof–
  note f = cat-ordinal-is-arrD[OF that]
  from f(1–3) show ?thesis
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps f(4)
          cs-intro: cat-ordinal-cs-intros V-cs-intros
      )
qed
show f ∘A cat-ordinal A cat-ordinal A(|CId|)(|b|) = f
  if f : b ↦cat-ordinal A c for b c f
proof–
  note f = cat-ordinal-is-arrD[OF that]
  from f(1–3) show ?thesis
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-ordinal-cs-simps f(4)
          cs-intro: cat-ordinal-cs-intros V-cs-intros
```

```
    )
qed
from assms Ord-α show cat-ordinal A(|Obj|) ⊆∘ Vset α
  unfolding cat-ordinal-components by auto
show (⋃∘b∈∘B. ⋃∘c∈∘C. Hom (cat-ordinal A) b c) ∈∘ Vset α
  if B ⊆∘ cat-ordinal A(|Obj|)
    and C ⊆∘ cat-ordinal A(|Obj|)
    and B ∈∘ Vset α
    and C ∈∘ Vset α
  for B C
proof–
  have (⋃∘b∈∘B. ⋃∘c∈∘C. Hom (cat-ordinal A) b c) ⊆∘ B ×• C
  proof(rule vsubsetI)
    fix f assume f ∈∘ (⋃∘b∈∘B. ⋃∘c∈∘C. Hom (cat-ordinal A) b c)
    then obtain b c
      where b: b ∈∘ B and c: c ∈∘ C and f: f : b ↦_{cat-ordinal A} c
      by auto
    note f = cat-ordinal-is-arrD[OF f]
    from b c show f ∈∘ B ×• C
      unfolding f(4) by auto
  qed
  moreover from that(3,4) have B ×• C ∈∘ Vset α
    by (auto intro: Limit-ftimes-in-VsetI)
  ultimately show ?thesis by blast
qed
show (∃f. Hom (cat-ordinal A) a b = set {f}) ∨ Hom (cat-ordinal A) a b = 0
  if a ∈∘ cat-ordinal A(|Obj|) and b ∈∘ cat-ordinal A(|Obj|) for a b
proof–
  from that have a: a ∈∘ A and b: b ∈∘ A
    unfolding cat-ordinal-components by simp-all
  then consider ‹a ≤ b› | ‹¬a ≤ b› by auto
  then show ?thesis
  proof cases
    case 1
    with a b have [a, b]∘ : a ↦_{cat-ordinal A} b
      by (auto intro: cat-ordinal-is-arrI)
    then show ?thesis by (simp add: cat-ordinal-Hom-ne)
  qed (simp add: cat-ordinal-Hom-empty)
qed
show a = b
  if a ∈∘ cat-ordinal A(|Obj|)
    and b ∈∘ cat-ordinal A(|Obj|)
    and a ≤_{O cat-ordinal A} b
    and b ≤_{O cat-ordinal A} a
  for a b
  using
    that(3,4)[unfolded cat-ordinal-is-le-iff cat-ordinal-components]
    cat-ordinal-is-arr-is-unique
  by auto
show a ≤_{O cat-ordinal A} b ∨ b ≤_{O cat-ordinal A} a
  if a ∈∘ cat-ordinal A(|Obj|) and b ∈∘ cat-ordinal A(|Obj|) for a b
proof–
  from that[unfolded cat-ordinal-components] have a: a ∈∘ A and b: b ∈∘ A
    by simp-all
  with assms have Ord a Ord b by auto
  then consider ‹a ≤ b› | ‹b ≤ a› by (meson Ord-linear-le)
  then show a ≤_{O cat-ordinal A} b ∨ b ≤_{O cat-ordinal A} a
    by cases (auto simp: a b cat-ordinal-is-le-iff intro: cat-ordinal-is-arrI)
```

```
  qed
qed (auto intro: cat-ordinal-cs-intros simp: cat-ordinal-cs-simps)

lemmas [cat-ordinal-cs-intros] = 𝒵.cat-linear-order-cat-ordinal

lemma (in 𝒵) cat-tiny-linear-order-cat-ordinal[cat-ordinal-cs-intros]:
  assumes Ord A and A ∈₀ α
  shows cat-tiny-linear-order α (cat-ordinal A)
proof(intro cat-tiny-linear-orderI′ cat-linear-order-cat-ordinal assms(1))
  from assms show A ⊆₀ α
    by (meson Ord-α Ord-linear-le mem-not-refl vsubsetE)
  from assms(1) this interpret A: cat-linear-order α ‹cat-ordinal A›
    by (intro cat-linear-order-cat-ordinal)
  from assms(2) have A-in-Vset: A ∈₀ Vset α by (simp add: Ord-α Ord-in-in-VsetI)
  have cat-ordinal A(|Arr|) ⊆₀ A ×• A
    unfolding ordinal-arrs-def cat-ordinal-components by clarsimp
  moreover from A-in-Vset have A ×• A ∈₀ Vset α
    by (intro Limit-ftimes-in-VsetI) auto
  ultimately have cat-ordinal A(|Arr|) ∈₀ Vset α
    using vsubset-in-VsetI unfolding cat-ordinal-components by simp
  moreover have cat-ordinal A(|Obj|) ∈₀ Vset α
    unfolding cat-ordinal-components by (intro A-in-Vset)
  ultimately show tiny-category α (cat-ordinal A)
    by (cs-concl cs-shallow cs-intro: cat-cs-intros tiny-categoryI′)
qed

lemmas [cat-ordinal-cs-intros] = 𝒵.cat-linear-order-cat-ordinal

lemma (in 𝒵) finite-category-cat-ordinal[cat-ordinal-cs-intros]:
  assumes a ∈₀ ω
  shows finite-category α (cat-ordinal a)
proof–
  from assms have Ord a a ∈₀ α by (auto simp: Ord-α Ord-trans)
  then interpret cat-ordinal: cat-tiny-linear-order α ‹cat-ordinal a›
    by (cs-concl cs-shallow cs-intro: cat-ordinal-cs-intros)
  show ?thesis
  proof(intro finite-categoryI′)
    from assms show category α (cat-ordinal a)
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from assms show vfinite (cat-ordinal a(|Obj|))
      unfolding cat-ordinal-components by auto
    from assms show vfinite (cat-ordinal a(|Arr|))
    proof–
      have cat-ordinal a(|Arr|) ⊆₀ a ×• a
        unfolding cat-ordinal-components by auto
      moreover from assms have vfinite (a ×• a)
        by (intro vfinite-ftimesI) auto
      ultimately show ?thesis by (auto simp: vfinite-vsubset)
    qed
  qed
qed

lemmas [cat-ordinal-cs-intros] = 𝒵.finite-category-cat-ordinal
```

33 Simplicial category

33.1 Background

The content of this section is based, primarily, on the elements of the content of Chapter I-2 in [7].

named-theorems *cat-simplicial-cs-simps*
named-theorems *cat-simplicial-cs-intros*

33.2 Composable arrows for simplicial category

definition *composable-cat-simplicial* :: $V \Rightarrow V \Rightarrow V$
  where *composable-cat-simplicial* α A = *set*
    {
      $[g, f]_\circ$ | g f. $\exists$ m n p.
        g : *cat-ordinal* n $\leq_{C.PEO\alpha}$ *cat-ordinal* p $\wedge$
        f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n $\wedge$
        m $\in_\circ$ A $\wedge$ n $\in_\circ$ A $\wedge$ p $\in_\circ$ A
    }

lemma *small-composable-cat-simplicial*[*simp*]:
  small
    {
      $[g, f]_\circ$ | g f. $\exists$ m n p.
        g : *cat-ordinal* n $\leq_{C.PEO\alpha}$ *cat-ordinal* p $\wedge$
        f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n $\wedge$
        m $\in_\circ$ A $\wedge$ n $\in_\circ$ A $\wedge$ p $\in_\circ$ A
    }
  (**is** ‹*small ?S*›)
proof(*rule down*)
  show *?S* $\subseteq$ *elts* (*all-cfs* α $\times_\bullet$ *all-cfs* α)
  proof
    (
      intro subsetI,
      unfold mem-Collect-eq, *elim exE conjE*; *simp only*:; *intro ftimesI2*
)
    fix m n p g f
    assume *prems*:
      m $\in_\circ$ A
      n $\in_\circ$ A
      p $\in_\circ$ A
      g : *cat-ordinal* n $\leq_{C.PEO\alpha}$ *cat-ordinal* p
      f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n
    interpret g: *is-preorder-functor* α ‹*cat-ordinal n*› ‹*cat-ordinal p*› g
      by (*rule prems*(*4*))
    interpret f: *is-preorder-functor* α ‹*cat-ordinal m*› ‹*cat-ordinal n*› f
      by (*rule prems*(*5*))
    from *g.is-functor-axioms* **show** g $\in_\circ$ *all-cfs* α **by** *auto*
    from *f.is-functor-axioms* **show** f $\in_\circ$ *all-cfs* α **by** *auto*
  qed
qed

Rules.

lemma *composable-cat-simplicialI*:
  assumes g : *cat-ordinal* n $\leq_{C.PEO\alpha}$ *cat-ordinal* p
    and f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n
    and m $\in_\circ$ A
    and n $\in_\circ$ A

and $p \in_\circ A$
and $gf = [g, f]_\circ$
shows $gf \in_\circ$ *composable-cat-simplicial* α A
using *assms*
unfolding *composable-cat-simplicial-def*
by (*intro in-small-setI small-composable-cat-simplicial*) *auto*

lemma *composable-cat-simplicialE*[*elim*]:
assumes $gf \in_\circ$ *composable-cat-simplicial* α A
obtains $g\ f\ m\ n\ p$ **where** $gf = [g, f]_\circ$
and g : *cat-ordinal* $n \le_{C.PEO\alpha}$ *cat-ordinal* p
and f : *cat-ordinal* $m \le_{C.PEO\alpha}$ *cat-ordinal* n
and $m \in_\circ A$
and $n \in_\circ A$
and $p \in_\circ A$
proof–
from *assms* **obtain** $g\ f\ m\ n\ p$ **where**
$gf = [g, f]_\circ$
g : *cat-ordinal* $n \le_{C.PEO\alpha}$ *cat-ordinal* p
f : *cat-ordinal* $m \le_{C.PEO\alpha}$ *cat-ordinal* n
$m \in_\circ A$
$n \in_\circ A$
$p \in_\circ A$
unfolding *composable-cat-simplicial-def*
by (*elim in-small-setE*, *intro small-composable-cat-simplicial*) *auto*
with *that* **show** *?thesis* **by** *auto*
qed

33.3 Simplicial category

33.3.1 Definition and elementary properties

definition *cat-simplicial* :: $V \Rightarrow V \Rightarrow V$
where *cat-simplicial* α A =
[
set {*cat-ordinal* m | m. $m \in_\circ A$},
set
{
f. $\exists\, m\ n$.
f : *cat-ordinal* $m \le_{C.PEO\alpha}$ *cat-ordinal* $n \wedge m \in_\circ A \wedge n \in_\circ A$
},
(
$\lambda f \in_\circ$ *set*
{
f. $\exists\, m\ n$.
f : *cat-ordinal* $m \le_{C.PEO\alpha}$ *cat-ordinal* $n \wedge m \in_\circ A \wedge n \in_\circ A$
}. $f(\!|HomDom|\!)$
),
(
$\lambda f \in_\circ$ *set*
{
f. $\exists\, m\ n$.
f : *cat-ordinal* $m \le_{C.PEO\alpha}$ *cat-ordinal* $n \wedge m \in_\circ A \wedge n \in_\circ A$
}. $f(\!|HomCod|\!)$
),
($\lambda gf \in_\circ$ *composable-cat-simplicial* α A. $gf(\!|0|\!) \circ_{CF} gf(\!|1_{\mathbb{N}}|\!)$),
($\lambda m \in_\circ$ *set* {*cat-ordinal* m | m. $m \in_\circ A$}. *cf-id* m)
]$_\circ$

Components.

lemma *cat-simplicial-components*:
shows *cat-simplicial* α $A(\!|Obj|\!)$ = *set* $\{$*cat-ordinal* $m \mid m.\ m \in_\circ A\}$
and *cat-simplicial* α $A(\!|Arr|\!)$ =
set $\{f.\ \exists m\ n.\ f : \textit{cat-ordinal}\ m \leq_{C.PEO\alpha} \textit{cat-ordinal}\ n \wedge m \in_\circ A \wedge n \in_\circ A\}$
and *cat-simplicial* α $A(\!|Dom|\!)$ =
(
$\lambda f \in_\circ$ *set*
{
$f.\ \exists m\ n.$
f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* $n \wedge m \in_\circ A \wedge n \in_\circ A$
}. $f(\!|HomDom|\!)$
)
and *cat-simplicial* α $A(\!|Cod|\!)$ =
(
$\lambda f \in_\circ$ *set*
{
$f.\ \exists m\ n.$
f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* $n \wedge m \in_\circ A \wedge n \in_\circ A$
}. $f(\!|HomCod|\!)$
)
and *cat-simplicial* α $A(\!|Comp|\!)$ =
($\lambda gf \in_\circ$ *composable-cat-simplicial* α A. $gf(\!|0|\!) \circ_{CF} gf(\!|1_{\mathbb{N}}|\!)$)
and *cat-simplicial* α $A(\!|CId|\!)$ =
($\lambda m \in_\circ$ *set* $\{$*cat-ordinal* $m \mid m.\ m \in_\circ A\}$. *cf-id* m)
unfolding *cat-simplicial-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

33.3.2 Objects

lemma *cat-simplicial-ObjI*[*cat-simplicial-cs-intros*]:
assumes $m \in_\circ A$ **and** a = *cat-ordinal* m
shows $a \in_\circ$ *cat-simplicial* α $A(\!|Obj|\!)$
using *assms* **unfolding** *cat-simplicial-components* **by** *auto*

lemma *cat-simplicial-ObjD*:
assumes *cat-ordinal* $m \in_\circ$ *cat-simplicial* α $A(\!|Obj|\!)$
shows $m \in_\circ A$
using *assms cat-ordinal-inj* **unfolding** *cat-simplicial-components* **by** *auto*

lemma *cat-simplicial-ObjE*:
assumes $M \in_\circ$ *cat-simplicial* α $A(\!|Obj|\!)$
obtains m **where** M = *cat-ordinal* m **and** $m \in_\circ A$
using *assms cat-ordinal-inj that* **unfolding** *cat-simplicial-components* **by** *auto*

33.3.3 Arrows

lemma *small-cat-simplicial-Arr*[*simp*]:
small $\{f.\ \exists m\ n.\ f : \textit{cat-ordinal}\ m \leq_{C.PEO\alpha} \textit{cat-ordinal}\ n \wedge m \in_\circ A \wedge n \in_\circ A\}$
(**is** ‹*small ?S*›)
proof(*rule down*)
show *?S* $\subseteq$ *elts* (*all-cfs* α) **by** *auto*
qed

lemma *cat-simplicial-ArrI*[*cat-simplicial-cs-intros*]:
assumes f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* n **and** $m \in_\circ A$ **and** $n \in_\circ A$
shows $f \in_\circ$ *cat-simplicial* α $A(\!|Arr|\!)$
using *assms*
unfolding *cat-simplicial-components*

by (*intro in-small-setI small-cat-simplicial-Arr*) *auto*

lemma *cat-simplicial-ArrE*:
assumes $f \in_{\circ}$ *cat-simplicial* α $A(\!|Arr|\!)$
obtains m n
where f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* n **and** $m \in_{\circ} A$ **and** $n \in_{\circ} A$
proof–
from *assms cat-ordinal-inj* **obtain** m n
where $m \in_{\circ} A$ $n \in_{\circ} A$ f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* n
unfolding *cat-simplicial-components*
by (*elim in-small-setE*, *intro small-cat-simplicial-Arr*) *auto*
with *that* **show** *?thesis* **by** *simp*
qed

33.3.4 Domain

mk-VLambda *cat-simplicial-components*(*3*)
|*vsv cat-simplicial-Dom-vsv*[*cat-simplicial-cs-intros*]|
|*vdomain*
cat-simplicial-Dom-vdomain[
folded cat-simplicial-components, *cat-simplicial-cs-simps*
]
|
|*app cat-simplicial-Dom-app*[*folded cat-simplicial-components*]|

lemma *cat-simplicial-Dom-app'*[*cat-simplicial-cs-simps*]:
assumes f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* n **and** $m \in_{\circ} A$ **and** $n \in_{\circ} A$
shows *cat-simplicial* α $A(\!|Dom|\!)(\!|f|\!)$ = *cat-ordinal* m
proof–
interpret f: *is-preorder-functor* α ‹*cat-ordinal* m› ‹*cat-ordinal* n› f
by (*rule assms*(*1*))
from *assms* **show** *cat-simplicial* α $A(\!|Dom|\!)(\!|f|\!)$ = *cat-ordinal* m
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-simplicial-Dom-app*
cs-intro: *cat-simplicial-cs-intros*
)
qed

33.3.5 Codomain

mk-VLambda *cat-simplicial-components*(*4*)
|*vsv cat-simplicial-Cod-vsv*[*cat-simplicial-cs-intros*]|
|*vdomain*
cat-simplicial-Cod-vdomain[
folded cat-simplicial-components, *cat-simplicial-cs-simps*
]
|
|*app cat-simplicial-Cod-app*[*folded cat-simplicial-components*]|

lemma *cat-simplicial-Cod-app'*[*cat-simplicial-cs-simps*]:
assumes f : *cat-ordinal* $m \leq_{C.PEO\alpha}$ *cat-ordinal* n **and** $m \in_{\circ} A$ **and** $n \in_{\circ} A$
shows *cat-simplicial* α $A(\!|Cod|\!)(\!|f|\!)$ = *cat-ordinal* n
proof–
interpret f: *is-preorder-functor* α ‹*cat-ordinal* m› ‹*cat-ordinal* n› f
by (*rule assms*(*1*))

from *assms* **show** *cat-simplicial* α $A(\!|Cod|\!)(\!|f|\!)$ = *cat-ordinal* n
by
(
cs-concl **cs-shallow**
cs-simp: *cat-cs-simps cat-simplicial-Cod-app*
cs-intro: *cat-simplicial-cs-intros*
)
qed

33.3.6 Arrow with a domain and a codomain

lemma *cat-simplicial-is-arrI*:
assumes f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n
and $m \in_{\circ} A$
and $n \in_{\circ} A$
shows f : *cat-ordinal* m $\mapsto_{\textit{cat-simplicial } \alpha\ A}$ *cat-ordinal* n
proof(*intro is-arrI cat-simplicial-ArrI*, *rule assms*; (*intro assms*(*2*,*3*))*?*)
from *assms* **show**
cat-simplicial α $A(\!|Dom|\!)(\!|f|\!)$ = *cat-ordinal* m
cat-simplicial α $A(\!|Cod|\!)(\!|f|\!)$ = *cat-ordinal* n
by (*cs-concl* **cs-shallow cs-simp**: *cat-simplicial-cs-simps*)+
qed

lemma *cat-simplicial-is-arrI'*[*cat-simplicial-cs-intros*]:
assumes f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n
and $m \in_{\circ} A$
and $n \in_{\circ} A$
and a = *cat-ordinal* m
and b = *cat-ordinal* n
shows f : a $\mapsto_{\textit{cat-simplicial } \alpha\ A}$ b
using *assms*(*1–3*) **unfolding** *assms*(*4–5*) **by** (*rule cat-simplicial-is-arrI*)

lemma *cat-simplicial-is-arrD*[*dest*]:
assumes f : *cat-ordinal* m $\mapsto_{\textit{cat-simplicial } \alpha\ A}$ *cat-ordinal* n
and $m \in_{\circ} A$
and $n \in_{\circ} A$
shows f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n
proof–
note f = *is-arrD*[*OF assms*(*1*)]
from *f*(*1*) **obtain** m' n'
where *f-is-preorder-functor*: f : *cat-ordinal* m' $\leq_{C.PEO\alpha}$ *cat-ordinal* n'
and $m' \in_{\circ} A$
and $n' \in_{\circ} A$
by (*elim cat-simplicial-ArrE*)
then have
cat-simplicial α $A(\!|Dom|\!)(\!|f|\!)$ = *cat-ordinal* m'
cat-simplicial α $A(\!|Cod|\!)(\!|f|\!)$ = *cat-ordinal* n'
by (*cs-concl* **cs-shallow cs-simp**: *cat-simplicial-cs-simps*)+
with *f*(*2*,*3*) **have**
cat-ordinal m' = *cat-ordinal* m *cat-ordinal* n' = *cat-ordinal* n
by *auto*
with *f*(*2*,*3*) *cat-ordinal-inj* **have** [*simp*]: $m' = m$ $n' = n$ **by** *auto*
from *f-is-preorder-functor* **show** *?thesis* **by** *simp*
qed

lemma *cat-simplicial-is-arrE*[*elim*]:
assumes f : a $\mapsto_{\textit{cat-simplicial } \alpha\ A}$ b
obtains m n **where** f : *cat-ordinal* m $\leq_{C.PEO\alpha}$ *cat-ordinal* n

and $m \in_\circ A$
and $n \in_\circ A$
and $a = \textit{cat-ordinal}\ m$
and $b = \textit{cat-ordinal}\ n$
proof–
note $f = \textit{is-arrD}[\textit{OF assms}(1)]$
from $f(1)$ **obtain** $m\ n$
where *f-is-preorder-functor*: $f : \textit{cat-ordinal}\ m \leq_{C.PEO\alpha} \textit{cat-ordinal}\ n$
and *m*: $m \in_\circ A$
and *n*: $n \in_\circ A$
by (*elim cat-simplicial-ArrE*)
then have
$\textit{cat-simplicial}\ \alpha\ A(\!|\textit{Dom}|\!)(\!|f|\!) = \textit{cat-ordinal}\ m$
$\textit{cat-simplicial}\ \alpha\ A(\!|\textit{Cod}|\!)(\!|f|\!) = \textit{cat-ordinal}\ n$
by (*cs-concl* **cs-shallow cs-simp**: *cat-simplicial-cs-simps*)+
with $f(2,3)$ **have** $\textit{cat-ordinal}\ m = a\ \textit{cat-ordinal}\ n = b$
by *auto*
with *f-is-preorder-functor m n that* **show** *?thesis* **by** *auto*
qed

33.3.7 Composition

mk-VLambda *cat-simplicial-components*(5)
|*vsv cat-simplicial-Comp-vsv*[*cat-simplicial-cs-intros*]|
|*vdomain cat-simplicial-Comp-vdomain*[*cat-simplicial-cs-simps*]|

lemma *cat-simplicial-Comp-app*[*cat-simplicial-cs-simps*]:
assumes $g : \textit{cat-ordinal}\ n \mapsto_{\textit{cat-simplicial}\ \alpha\ A}\ \textit{cat-ordinal}\ p$
and $f : \textit{cat-ordinal}\ m \mapsto_{\textit{cat-simplicial}\ \alpha\ A}\ \textit{cat-ordinal}\ n$
and $m \in_\circ A$
and $n \in_\circ A$
and $p \in_\circ A$
shows $g \circ_{A\,\textit{cat-simplicial}\ \alpha\ A} f = g \circ_{CF} f$
proof–
note $g = \textit{cat-simplicial-is-arrD}[\textit{OF assms}(1,4,5)]$
note $f = \textit{cat-simplicial-is-arrD}[\textit{OF assms}(2,3,4)]$
interpret *g*: *is-preorder-functor* $\alpha\ \langle\textit{cat-ordinal}\ n\rangle\ \langle\textit{cat-ordinal}\ p\rangle\ g$
by (*rule g*)
interpret *f*: *is-preorder-functor* $\alpha\ \langle\textit{cat-ordinal}\ m\rangle\ \langle\textit{cat-ordinal}\ n\rangle\ f$
by (*rule f*)
have $[g, f]_\circ \in_\circ \textit{composable-cat-simplicial}\ \alpha\ A$
by
(
intro composable-cat-simplicialI, *rule g*, *rule f*;
(*intro assms refl*)*?*
)
then show $g \circ_{A\,\textit{cat-simplicial}\ \alpha\ A} f = g \circ_{CF} f$
unfolding *cat-simplicial-components* **by** (*simp add*: *nat-omega-simps*)
qed

33.3.8 Identity

context
fixes $\alpha\ A :: V$
begin

mk-VLambda *cat-simplicial-components*(6)[**where** $\alpha{=}\alpha$ **and** $A{=}A$]
|*vsv cat-simplicial-CId-vsv*[*cat-simplicial-cs-intros*]|

|*vdomain*
cat-simplicial-CId-vdomain′[
folded cat-simplicial-components(*1*)[**where** α=α **and** *A*=*A*]
]
|
|*app cat-simplicial-CId-app′*[
folded cat-simplicial-components(*1*)[**where** α=α **and** *A*=*A*]
]
|

lemmas *cat-simplicial-CId-vdomain*[*cat-simplicial-cs-simps*] =
cat-simplicial-CId-vdomain′
lemmas *cat-simplicial-CId-app*[*cat-simplicial-cs-simps*] =
cat-simplicial-CId-app′

end

33.3.9 Simplicial category is a category

lemma (**in** $\mathcal{Z}$) *category-simplicial*:
assumes *Ord A* **and** $A \subseteq_\circ \alpha$
shows *category* α (*cat-simplicial* α *A*)
proof–

show *?thesis*
proof(*intro categoryI′*)

show *vfsequence* (*cat-simplicial* α *A*) **unfolding** *cat-simplicial-def* **by** *simp*
show *vcard* (*cat-simplicial* α *A*) = $6_{\mathbb{N}}$
unfolding *cat-simplicial-def* **by** (*simp add*: *nat-omega-simps*)

show $\mathcal{R}_\circ$ (*cat-simplicial* α *A*(|*Dom*|)) $\subseteq_\circ$ *cat-simplicial* α *A*(|*Obj*|)
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-simplicial-cs-simps*)
fix *f* **assume** *f* $\in_\circ$ *cat-simplicial* α *A*(|*Arr*|)
then obtain *m n*
where *f* : *cat-ordinal m* $\leq_{C.PEO\alpha}$ *cat-ordinal n*
and *m* $\in_\circ$ *A*
and *n* $\in_\circ$ *A*
by (*elim cat-simplicial-ArrE*)
then show *cat-simplicial* α *A*(|*Dom*|)(|*f*|) $\in_\circ$ *cat-simplicial* α *A*(|*Obj*|)
by (*auto simp*: *cat-simplicial-Dom-app′ intro*: *cat-simplicial-ObjI*)
qed (*auto simp*: *cat-simplicial-components*)

show $\mathcal{R}_\circ$ (*cat-simplicial* α *A*(|*Cod*|)) $\subseteq_\circ$ *cat-simplicial* α *A*(|*Obj*|)
proof(*rule vsv.vsv-vrange-vsubset*, *unfold cat-simplicial-cs-simps*)
fix *f* **assume** *f* $\in_\circ$ *cat-simplicial* α *A*(|*Arr*|)
then obtain *m n*
where *f* : *cat-ordinal m* $\leq_{C.PEO\alpha}$ *cat-ordinal n*
and *m* $\in_\circ$ *A*
and *n* $\in_\circ$ *A*
by (*elim cat-simplicial-ArrE*)
then show *cat-simplicial* α *A*(|*Cod*|)(|*f*|) $\in_\circ$ *cat-simplicial* α *A*(|*Obj*|)
by (*auto simp*: *cat-simplicial-Cod-app′ intro*: *cat-simplicial-ObjI*)
qed (*auto simp*: *cat-simplicial-components*)

show (*gf* $\in_\circ$ $\mathcal{D}_\circ$ (*cat-simplicial* α *A*(|*Comp*|))) $\longleftrightarrow$
(
$\exists\, g\ f\ b\ c\ a.$

```
      gf = [g, f]∘ ∧
      g : b ↦cat-simplicial α A c ∧
      f : a ↦cat-simplicial α A b
  )
  for gf
proof(intro iffI; (elim exE conjE)?)
  assume gf ∈∘ 𝒟∘ (cat-simplicial α A(|Comp|))
  then have gf ∈∘ composable-cat-simplicial α A
    unfolding cat-simplicial-components by simp
  then obtain g f m n p
    where gf-def: gf = [g, f]∘
      and g: g : cat-ordinal n ≤C.PEOα cat-ordinal p
      and f: f : cat-ordinal m ≤C.PEOα cat-ordinal n
      and m: m ∈∘ A
      and n: n ∈∘ A
      and p: p ∈∘ A
    by auto
  show ∃ g f b c a.
    gf = [g, f]∘ ∧
    g : b ↦cat-simplicial α A c ∧
    f : a ↦cat-simplicial α A b
  proof(intro exI conjI)
    from g n p show g : cat-ordinal n ↦cat-simplicial α A cat-ordinal p
      by (intro cat-simplicial-is-arrI) simp-all
    from f m n show f : cat-ordinal m ↦cat-simplicial α A cat-ordinal n
      by (intro cat-simplicial-is-arrI) simp-all
  qed (simp add: gf-def)
next
  fix g f a b c assume prems:
    gf = [g, f]∘
    g : b ↦cat-simplicial α A c
    f : a ↦cat-simplicial α A b
  from prems(2) obtain n p
    where g: g : cat-ordinal n ≤C.PEOα cat-ordinal p
      and n: n ∈∘ A
      and p: p ∈∘ A
      and b-def: b = cat-ordinal n
      and c = cat-ordinal p
    by auto
  from prems(3) obtain m n′
    where f: f : cat-ordinal m ≤C.PEOα cat-ordinal n′
      and m: m ∈∘ A
      and n′: n′ ∈∘ A
      and a-def: a = cat-ordinal m
      and b-def′: b = cat-ordinal n′
    by auto
  from b-def b-def′ have n′n: n′ = n by (auto simp: cat-ordinal-inj)
  show gf ∈∘ 𝒟∘ (cat-simplicial α A(|Comp|))
    unfolding prems(1) cat-simplicial-Comp-vdomain
    by (intro composable-cat-simplicialI, rule g, rule f[unfolded n′n])
      (simp-all add: m n p)
qed
show g ∘A cat-simplicial α A f : a ↦cat-simplicial α A c
  if g : b ↦cat-simplicial α A c and f : a ↦cat-simplicial α A b
  for b c g a f
  using that
  by (elim cat-simplicial-is-arrE; simp only: cat-ordinal-inj)
```

```
    (
      cs-concl cs-shallow
        cs-simp: cat-simplicial-cs-simps
        cs-intro: cat-order-cs-intros cat-simplicial-cs-intros
    )

show h ∘_A cat-simplicial α A g ∘_A cat-simplicial α A f =
  h ∘_A cat-simplicial α A (g ∘_A cat-simplicial α A f)
  if h : c ↦_cat-simplicial α A d
    and g : b ↦_cat-simplicial α A c
    and f : a ↦_cat-simplicial α A b
  for c d h b g a f
  using that
  apply(elim cat-simplicial-is-arrE; simp only:)
  subgoal for m n m' n' m'' n''
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-simplicial-cs-simps
          cs-intro: cat-order-cs-intros cat-simplicial-cs-intros
      )+
  done

show cat-simplicial α A(|CId|)(|a|) : a ↦_cat-simplicial α A a
  if a ∈_∘ cat-simplicial α A(|Obj|) for a
  using that
proof(elim cat-simplicial-ObjE; simp only:)
  fix m assume prems: m ∈_∘ A cat-ordinal m ∈_∘ cat-simplicial α A(|Obj|)
  moreover from prems(1) assms(1) have Ord m by auto
  moreover from prems assms have m ⊆_∘ α
    by (meson Ord-trans vsubsetI rev-vsubsetD)
  ultimately show cat-simplicial α A(|CId|)(|cat-ordinal m|) :
    cat-ordinal m ↦_cat-simplicial α A cat-ordinal m
    by
      (
        cs-concl
          cs-simp: cat-simplicial-cs-simps
          cs-intro:
            cat-ordinal-cs-intros
            cat-order-cs-intros
            cat-simplicial-cs-intros
      )
qed
show cat-simplicial α A(|CId|)(|b|) ∘_A cat-simplicial α A f = f
  if f : a ↦_cat-simplicial α A b for a b f
  using that
  by (elim cat-simplicial-is-arrE; simp only:)
    (
      cs-concl cs-shallow
        cs-simp: cat-cs-simps cat-simplicial-cs-simps
        cs-intro: cat-order-cs-intros cat-simplicial-cs-intros
    )
show f ∘_A cat-simplicial α A cat-simplicial α A(|CId|)(|b|) = f
  if f : b ↦_cat-simplicial α A c for b c f
  using that
  by (elim cat-simplicial-is-arrE; simp only:)
    (
      cs-concl
```

cs-simp: *cat-cs-simps cat-simplicial-cs-simps*
cs-intro: *cat-order-cs-intros cat-simplicial-cs-intros*
)
show *cat-simplicial* α $A(\!|Obj|\!) \subseteq_\circ$ *Vset* α
proof(*intro vsubsetI*, *elim cat-simplicial-ObjE*; *simp only*:)
fix m **assume** *prems*: $m \in_\circ A$
then have *Ord* m **using** *assms*(*1*) **by** *auto*
moreover from *prems* **have** $m \in_\circ \alpha$ **using** *assms*(*2*) **by** *auto*
ultimately interpret m: *cat-tiny-linear-order* α ‹*cat-ordinal m*›
by (*intro cat-tiny-linear-order-cat-ordinal*)
show *cat-ordinal* $m \in_\circ$ *Vset* α **by** (*rule m.tiny-cat-in-Vset*)
qed

show $(\bigcup_\circ a \in_\circ A'.\ \bigcup_\circ b \in_\circ B'.$ *Hom* (*cat-simplicial* α A) a $b) \in_\circ$ *Vset* α
if $A' \subseteq_\circ$ *cat-simplicial* α $A(\!|Obj|\!)$
and $B' \subseteq_\circ$ *cat-simplicial* α $A(\!|Obj|\!)$
and $A' \in_\circ$ *Vset* α
and $B' \in_\circ$ *Vset* α
for A' B'
proof–
define Q **where** Q i =
(
if $i = 0 \Rightarrow$ *VPow* $((\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!)) \times_\circ (\bigcup_\circ b' \in_\circ B'.\ b'(\!|Obj|\!)))$
$|\ i = 1_\mathbb{N} \Rightarrow$ *VPow*
$(((\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!)) \times_\bullet (\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!))) \times_\circ$
$((\bigcup_\circ a' \in_\circ B'.\ a'(\!|Obj|\!)) \times_\bullet (\bigcup_\circ a' \in_\circ B'.\ a'(\!|Obj|\!))))$
$|\ i = 2_\mathbb{N} \Rightarrow A'$
$|\ i = 3_\mathbb{N} \Rightarrow B'$
$|$ *otherwise* $\Rightarrow 0$
)
for i
let *?Q* =
‹{
$[fo, fa, a, b]_\circ \mid fo\ fa\ a\ b.$
$fo \subseteq_\circ ((\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!)) \times_\circ (\bigcup_\circ b' \in_\circ B'.\ b'(\!|Obj|\!))) \wedge$
$fa \subseteq_\circ$
$((\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!)) \times_\bullet (\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!))) \times_\circ$
$((\bigcup_\circ a' \in_\circ B'.\ a'(\!|Obj|\!)) \times_\bullet (\bigcup_\circ a' \in_\circ B'.\ a'(\!|Obj|\!))) \wedge$
$a \in_\circ A' \wedge$
$b \in_\circ B'$
}›

have *QQ*: *?Q* $\subseteq$ *elts* $(\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}, 3_\mathbb{N}\}.\ Q\ i)$
proof(*intro subsetI*, *unfold mem-Collect-eq*, *elim exE conjE*)
fix x fo fa a b **assume** *prems*:
$x = [fo, fa, a, b]_\circ$
$fo \subseteq_\circ ((\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!)) \times_\circ (\bigcup_\circ b' \in_\circ B'.\ b'(\!|Obj|\!)))$
$fa \subseteq_\circ$
$((\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!)) \times_\bullet (\bigcup_\circ a' \in_\circ A'.\ a'(\!|Obj|\!))) \times_\circ$
$((\bigcup_\circ a' \in_\circ B'.\ a'(\!|Obj|\!)) \times_\bullet (\bigcup_\circ a' \in_\circ B'.\ a'(\!|Obj|\!)))$
$a \in_\circ A'$
$b \in_\circ B'$
show $x \in_\circ (\prod_\circ i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}, 3_\mathbb{N}\}.\ Q\ i)$
proof(*intro vproductI*, *unfold Ball-def*; (*intro allI impI*)*?*)
show $\mathcal{D}_\circ\ x = set\ \{[]_\circ, 1_\mathbb{N}, 2_\mathbb{N}, 3_\mathbb{N}\}$
unfolding *prems*(*1*) **by** (*force simp*: *nat-omega-simps*)
fix i **assume** $i \in_\circ set\ \{0, 1_\mathbb{N}, 2_\mathbb{N}, 3_\mathbb{N}\}$
then consider ‹$i = 0$› | ‹$i = 1_\mathbb{N}$› | ‹$i = 2_\mathbb{N}$› | ‹$i = 3_\mathbb{N}$› **by** *auto*

```
    then show x(|i|) ∈∘ Q i
      by cases (auto simp: Q-def prems nat-omega-simps)
  qed (auto simp: prems)
qed
then have small-Q[simp]: small ?Q by (intro down)

have (⋃∘a∈∘A'. ⋃∘b∈∘B'. Hom (cat-simplicial α A) a b) ⊆∘ set ?Q
proof(intro vsubsetI in-small-setI small-Q)
  fix f assume f ∈∘ (⋃∘a∈∘A'. ⋃∘b∈∘B'. Hom (cat-simplicial α A) a b)
  then obtain a b
    where a: a ∈∘ A'
      and b: b ∈∘ B'
      and f : a ↦_{cat-simplicial α A} b
    by auto
  then obtain m n
    where f: f : cat-ordinal m ≤_{C.PEOα} cat-ordinal n
      and m: m ∈∘ A
      and n: n ∈∘ A
      and a-def: a = cat-ordinal m
      and b-def: b = cat-ordinal n
    by auto
  interpret f: is-preorder-functor α ‹cat-ordinal m› ‹cat-ordinal n› f
    by (rule f)
  show f ∈ ?Q
  proof(unfold mem-Collect-eq, intro exI conjI)
    show f(|ObjMap|) ⊆∘ (⋃∘a'∈∘A'. a'(|Obj|)) ×∘ (⋃∘b'∈∘B'. b'(|Obj|))
    proof(intro vsubsetI)
      fix x assume prems: x ∈∘ f(|ObjMap|)
      obtain xl xr
        where x-def: x = ⟨xl, xr⟩
          and xl: xl ∈∘ cat-ordinal m(|Obj|)
          and xr: xr ∈∘ (ℛ∘ (f(|ObjMap|)))
        by (elim f.ObjMap.vbrelation-vinE[OF prems, unfolded cat-cs-simps])
      show x ∈∘ (⋃∘a'∈∘A'. a'(|Obj|)) ×∘ (⋃∘b'∈∘B'. b'(|Obj|))
        unfolding x-def
      proof(standard; (intro vifunionI))
        from xr f.cf-ObjMap-vrange show xr ∈∘ cat-ordinal n(|Obj|) by auto
      qed (use a b in ‹auto intro: xl simp: a-def b-def›)
    qed
    show f(|ArrMap|) ⊆∘
      ((⋃∘a'∈∘A'. a'(|Obj|)) ×• (⋃∘a'∈∘A'. a'(|Obj|))) ×∘
      ((⋃∘a'∈∘B'. a'(|Obj|)) ×• (⋃∘a'∈∘B'. a'(|Obj|)))
    proof(intro vsubsetI)
      fix x assume prems: x ∈∘ f(|ArrMap|)
      obtain xl xr
        where x-def: x = ⟨xl, xr⟩
          and xl: xl ∈∘ cat-ordinal m(|Arr|)
          and xr: xr ∈∘ (ℛ∘ (f(|ArrMap|)))
        by (elim f.ArrMap.vbrelation-vinE[OF prems, unfolded cat-cs-simps])
      from xr vsubsetD have xr: xr ∈∘ cat-ordinal n(|Arr|)
        by (auto intro: f.cf-ArrMap-vrange)
      from xl obtain xll xlr where xl-def: xl = [xll, xlr]∘
        and xll-m: xll ∈∘ m
        and xlr-m: xlr ∈∘ m
        and xll ⊆∘ xlr
        unfolding ordinal-arrs-def cat-ordinal-components by clarsimp
      from xr obtain xrl xrr where xr-def: xr = [xrl, xrr]∘
        and xrl-n: xrl ∈∘ n
```

```
      and xrr-n:xrr ∈∘ n
      and xrl ⊆∘ xrr
      unfolding ordinal-arrs-def cat-ordinal-components by clarsimp
    show x ∈∘
      ((⋃∘a′∈∘A′. a′⦇Obj⦈) ×• (⋃∘a′∈∘A′. a′⦇Obj⦈)) ×∘
      ((⋃∘a′∈∘B′. a′⦇Obj⦈) ×• (⋃∘a′∈∘B′. a′⦇Obj⦈))
      unfolding x-def
      by (standard; (intro vifunionI ftimesI1)?)
        (
          use a b in ‹
            auto
              simp: xl-def xr-def a-def b-def cat-ordinal-components
              intro: xrr-n xrl-n xlr-m xll-m
            ›
        )
  qed
qed
(
  auto
    simp: cat-cs-simps
    intro: a[unfolded a-def] b[unfolded b-def] f.cf-def
)
moreover have set ?Q ⊆∘ (∏∘i∈∘set {0, 1ℕ, 2ℕ, 3ℕ}. Q i)
  by
    (
      intro vsubset-if-subset,
      unfold small-elts-of-set[OF small-Q],
      intro QQ
    )
moreover have (∏∘i∈∘set {0, 1ℕ, 2ℕ, 3ℕ}. Q i) ∈∘ Vset α
proof(intro Limit-vproduct-in-VsetI)
  show set {0, 1ℕ, 2ℕ, 3ℕ} ∈∘ Vset α
    unfolding four[symmetric] by simp
  have (⋃∘a′∈∘A′. a′⦇Obj⦈) ⊆∘ ⋃∘(⋃∘r∈∘A′. ℛ∘ r)
  proof(intro vsubsetI)
    fix x assume x ∈∘ (⋃∘a′∈∘A′. a′⦇Obj⦈)
    then obtain a′ where a′: a′ ∈∘ A′ and x: x ∈∘ a′⦇Obj⦈ by auto
    from a′ that(1) have a′ ∈∘ cat-simplicial α A⦇Obj⦈ by auto
    then obtain m where a′-def: a′ = cat-ordinal m and m: m ∈∘ A
      unfolding cat-simplicial-components by clarsimp
    show x ∈∘ ⋃∘(⋃∘r∈∘A′. ℛ∘ r)
    proof(rule VUnionI, rule vifunionI)
      from a′-def have vsv a′ and Obj ∈∘ 𝒟∘ a′
        unfolding a′-def cat-ordinal-def Obj-def by auto
      then show a′⦇Obj⦈ ∈∘ ℛ∘ a′ by auto
    qed (auto simp: x a′)
  qed
  moreover have (⋃∘r∈∘A′. ℛ∘ r) ∈∘ Vset α
    by (intro Limit-VUnion-vrange-in-VsetI[OF Limit-α] that)
  ultimately have UA′: (⋃∘a′∈∘A′. a′⦇Obj⦈) ∈∘ Vset α by blast
  have B′: (⋃∘b′∈∘B′. b′⦇Obj⦈) ⊆∘ ⋃∘(⋃∘r∈∘B′. ℛ∘ r)

  proof(intro vsubsetI)
    fix x assume x ∈∘ (⋃∘b′∈∘B′. b′⦇Obj⦈)
    then obtain b′ where b′: b′ ∈∘ B′ and x: x ∈∘ b′⦇Obj⦈ by auto
    from b′ that(2) have b′ ∈∘ cat-simplicial α A⦇Obj⦈ by auto
    then obtain m where b′-def: b′ = cat-ordinal m and m: m ∈∘ A
```

```
      unfolding cat-simplicial-components by clarsimp
    show x ∈∘ ⋃∘(⋃∘r∈∘B′. ℛ∘ r)
    proof(rule VUnionI, rule vifunionI)
      from b′-def have vsv b′ and Obj ∈∘ 𝒟∘ b′
        unfolding b′-def cat-ordinal-def Obj-def by auto
      then show b′(|Obj|) ∈∘ ℛ∘ b′ by auto
    qed (auto simp: x b′)
  qed
  moreover have (⋃∘r∈∘B′. ℛ∘ r) ∈∘ Vset α
    by (intro Limit-VUnion-vrange-in-VsetI[OF Limit-α] that)
  ultimately have UB′: (⋃∘a′∈∘B′. a′(|Obj|)) ∈∘ Vset α by blast
  have [simp]:
    VPow ((⋃∘a′∈∘A′. a′(|Obj|)) ×∘ (⋃∘b′∈∘B′. b′(|Obj|))) ∈∘ Vset α
    by (intro Limit-VPow-in-VsetI Limit-vtimes-in-VsetI UA′ UB′) auto
  have [simp]:
    VPow
      (
        ((⋃∘a′∈∘A′. a′(|Obj|)) ×• (⋃∘a′∈∘A′. a′(|Obj|))) ×∘
        ((⋃∘a′∈∘B′. a′(|Obj|)) ×• (⋃∘a′∈∘B′. a′(|Obj|)))
      ) ∈∘ Vset α
    by
      (
        intro
          Limit-VPow-in-VsetI
          Limit-vtimes-in-VsetI
          Limit-ftimes-in-VsetI
          UA′ UB′
      )
      auto
  fix i assume i ∈∘ set {0, 1ₙ, 2ₙ, 3ₙ}
  then consider ‹i = 0› | ‹i = 1ₙ› | ‹i = 2ₙ› | ‹i = 3ₙ› by auto
  then show Q i ∈∘ Vset α
    by cases (simp-all add: Q-def that nat-omega-simps)
qed auto
ultimately show ?thesis by (simp add: vsubset-in-VsetI)
qed
qed (auto simp: cat-simplicial-components)

qed
```

34 Example: categories with additional structure

34.1 Background

The examples that are presented in this section showcase how the framework developed in this article can be used for the formalization of the theory of categories with additional structure. The content of this section also indicates some of the potential future directions for this body of work.

34.2 Dagger category

named-theorems *dag-field-simps*

named-theorems *dagcat-cs-simps*
named-theorems *dagcat-cs-intros*

definition *DagCat* :: *V* **where** [*dag-field-simps*]: *DagCat* = *0*
definition *DagDag* :: *V* **where** [*dag-field-simps*]: *DagDag* = $1_{\mathbb{N}}$

abbreviation *DagDag-app* :: *V* ⇒ *V* (‹$\dagger_C$›)
  where $\dagger_C$ $\mathfrak{C}$ ≡ $\mathfrak{C}$(|*DagDag*|)

34.2.1 Definition and elementary properties

For further information see [1][19].

locale *dagger-category* =
  $\mathcal{Z}$ α +
  vfsequence $\mathfrak{C}$ +
  DagCat: *category* α ‹$\mathfrak{C}$(|*DagCat*|)› +
  DagDag: *is-functor* α ‹*op-cat* ($\mathfrak{C}$(|*DagCat*|))› ‹$\mathfrak{C}$(|*DagCat*|)› ‹$\dagger_C$ $\mathfrak{C}$›
  for α $\mathfrak{C}$ +
  assumes *dagcat-length*: *vcard* $\mathfrak{C}$ = $2_{\mathbb{N}}$
    and *dagcat-ObjMap-identity*[*dagcat-cs-simps*]:
      a ∈$_\circ$ $\mathfrak{C}$(|*DagCat*|)(|*Obj*|) ⟹ ($\dagger_C$ $\mathfrak{C}$)(|*ObjMap*|)(|*a*|) = *a*
    and *dagcat-DagCat-idem*[*dagcat-cs-simps*]:
      $\dagger_C$ $\mathfrak{C}$ $_{CF}\circ$ $\dagger_C$ $\mathfrak{C}$ = *cf-id* ($\mathfrak{C}$(|*DagCat*|))

lemmas [*dagcat-cs-simps*] =
  dagger-category.dagcat-ObjMap-identity
  dagger-category.dagcat-DagCat-idem

Rules.

lemma (**in** *dagger-category*) *dagger-category-axioms'*[*dagcat-cs-intros*]:
  assumes α' = α
  shows *dagger-category* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule dagger-category-axioms*)

mk-ide rf *dagger-category-def*[*unfolded dagger-category-axioms-def*]
  |*intro dagger-categoryI*|
  |*dest dagger-categoryD*[*dest*]|
  |*elim dagger-categoryE*[*elim*]|

lemma *category-if-dagger-category*[*dagcat-cs-intros*]:
  assumes $\mathfrak{C}'$ = ($\mathfrak{C}$(|*DagCat*|)) **and** *dagger-category* α $\mathfrak{C}$
  shows *category* α $\mathfrak{C}'$

[19] https://ncatlab.org/nlab/show/dagger+category

unfolding *assms*(*1*) **using** *assms*(*2*) **by** (*rule dagger-categoryD*(*3*))

lemma (**in** *dagger-category*) *dagcat-is-functor'*[*dagcat-cs-intros*]:
assumes $\mathfrak{A}' = op\text{-}cat\ (\mathfrak{C}(\!|DagCat|\!))$ **and** $\mathfrak{B}' = \mathfrak{C}(\!|DagCat|\!)$
shows $\dagger_C\ \mathfrak{C} : \mathfrak{A}' \mapsto\mapsto_{C\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule DagDag.is-functor-axioms*)

lemmas [*dagcat-cs-intros*] = *dagger-category.dagcat-is-functor'*

34.3 *Rel* as a dagger category

34.3.1 Definition and elementary properties

For further information see [1][20].

definition *dagcat-Rel* :: $V \Rightarrow V$
where *dagcat-Rel* α = $[cat\text{-}Rel\ \alpha, \dagger_{C.Rel}\ \alpha]_\circ$

Components.

lemma *dagcat-Rel-components*:
shows *dagcat-Rel* $\alpha(\!|DagCat|\!)$ = *cat-Rel* α
and *dagcat-Rel* $\alpha(\!|DagDag|\!) = \dagger_{C.Rel}\ \alpha$
unfolding *dagcat-Rel-def dag-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

34.3.2 *Rel* is a dagger category

lemma (**in** $\mathcal{Z}$) *dagger-category-dagcat-Rel*: *dagger-category* α (*dagcat-Rel* α)
proof(*intro dagger-categoryI*)
show *category* α (*dagcat-Rel* $\alpha(\!|DagCat|\!)$)
by
(
cs-concl **cs-shallow**
cs-simp: *dagcat-Rel-components* **cs-intro**: *cat-Rel-cs-intros*
)
show $\dagger_C$ (*dagcat-Rel* α) :
op-cat (*dagcat-Rel* $\alpha(\!|DagCat|\!)$) $\mapsto\mapsto_{C\alpha}$ *dagcat-Rel* $\alpha(\!|DagCat|\!)$
unfolding *dagcat-Rel-components*
by (*cs-concl* **cs-intro**: *cf-cs-intros cat-cs-intros*)
show *vcard* (*dagcat-Rel* α) = $2_{\mathbb{N}}$
unfolding *dagcat-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\dagger_C$ (*dagcat-Rel* α)$(\!|ObjMap|\!)(\!|a|\!)$ = *a*
if $a \in_\circ$ *dagcat-Rel* $\alpha(\!|DagCat|\!)(\!|Obj|\!)$ **for** *a*
using *that*
unfolding *dagcat-Rel-components cat-Rel-components*(*1*)
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps cat-Rel-cs-simps*)
show $\dagger_C$ (*dagcat-Rel* α) $_{CF}\circ$ $\dagger_C$ (*dagcat-Rel* α) = *dghm-id* (*dagcat-Rel* $\alpha(\!|DagCat|\!)$)
unfolding *dagcat-Rel-components*
by (*cs-concl* **cs-shallow cs-simp**: *cf-cn-comp-cf-dag-Rel-cf-dag-Rel*)
qed (*auto simp*: *dagcat-Rel-def*)

34.4 Monoidal category

For background information see Chapter 2 in [4].

[20] https://ncatlab.org/nlab/show/Rel

34.4.1 Background

named-theorems *mcat-field-simps*

named-theorems *mcat-cs-simps*
named-theorems *mcat-cs-intros*

definition *Mcat* :: *V* **where** [*mcat-field-simps*]: *Mcat* = *0*
definition *Mcf* :: *V* **where** [*mcat-field-simps*]: *Mcf* = $1_{\mathbb{N}}$
definition *Me* :: *V* **where** [*mcat-field-simps*]: *Me* = $2_{\mathbb{N}}$
definition *Mα* :: *V* **where** [*mcat-field-simps*]: *Mα* = $3_{\mathbb{N}}$
definition *Ml* :: *V* **where** [*mcat-field-simps*]: *Ml* = $4_{\mathbb{N}}$
definition *Mr* :: *V* **where** [*mcat-field-simps*]: *Mr* = $5_{\mathbb{N}}$

34.4.2 Definition and elementary properties

locale *monoidal-category* =
— See Definition 2.2.8 in [4].
$\mathcal{Z}$ *α* +
vfsequence 𝔈 +
Mcat: *category α* ‹𝔈(⌊*Mcat*⌋)› +
Mcf: *is-functor α* ‹(𝔈(⌊*Mcat*⌋)) ×$_C$ (𝔈(⌊*Mcat*⌋))› ‹𝔈(⌊*Mcat*⌋)› ‹𝔈(⌊*Mcf*⌋)› +
Mα: *is-iso-ntcf*
α ‹𝔈(⌊*Mcat*⌋)^$_{C3}$› ‹𝔈(⌊*Mcat*⌋)› ‹*cf-blcomp* (𝔈(⌊*Mcf*⌋))› ‹*cf-brcomp* (𝔈(⌊*Mcf*⌋))› ‹𝔈(⌊*Mα*⌋)› +
Ml: *is-iso-ntcf*
α
‹𝔈(⌊*Mcat*⌋)›
‹𝔈(⌊*Mcat*⌋)›
‹𝔈(⌊*Mcf*⌋)$_{𝔈(⌊Mcat⌋),𝔈(⌊Mcat⌋)}$(𝔈(⌊*Me*⌋),−)$_{CF}$›
‹*cf-id* (𝔈(⌊*Mcat*⌋))›
‹𝔈(⌊*Ml*⌋)› +
Mr: *is-iso-ntcf*
α
‹𝔈(⌊*Mcat*⌋)›
‹𝔈(⌊*Mcat*⌋)›
‹𝔈(⌊*Mcf*⌋)$_{𝔈(⌊Mcat⌋),𝔈(⌊Mcat⌋)}$(−,𝔈(⌊*Me*⌋))$_{CF}$›
‹*cf-id* (𝔈(⌊*Mcat*⌋))›
‹𝔈(⌊*Mr*⌋)›
for *α* 𝔈 +
assumes *mcat-length*[*mcat-cs-simps*]: *vcard* 𝔈 = $6_{\mathbb{N}}$
and *mcat-Me-is-obj*[*mcat-cs-intros*]: 𝔈(⌊*Me*⌋) ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋)
and *mcat-pentagon*:
⟦
a ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋);
b ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋);
c ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋);
d ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋)
⟧ ⟹
(𝔈(⌊*Mcat*⌋)(⌊*CId*⌋)(⌊*a*⌋) ⊗$_{HM.A}$𝔈(⌊*Mcf*⌋) 𝔈(⌊*Mα*⌋)(⌊*NTMap*⌋)(⌊*b*, *c*, *d*⌋$_\bullet$) ∘$_A$𝔈(⌊*Mcat*⌋)
𝔈(⌊*Mα*⌋)(⌊*NTMap*⌋)(⌊*a*, *b* ⊗$_{HM.O}$𝔈(⌊*Mcf*⌋) *c*, *d*⌋$_\bullet$ ∘$_A$𝔈(⌊*Mcat*⌋)
(𝔈(⌊*Mα*⌋)(⌊*NTMap*⌋)(⌊*a*, *b*, *c*⌋$_\bullet$ ⊗$_{HM.A}$𝔈(⌊*Mcf*⌋) 𝔈(⌊*Mcat*⌋)(⌊*CId*⌋)(⌊*d*⌋)) =
𝔈(⌊*Mα*⌋)(⌊*NTMap*⌋)(⌊*a*, *b*, *c* ⊗$_{HM.O}$𝔈(⌊*Mcf*⌋) *d*⌋$_\bullet$ ∘$_A$𝔈(⌊*Mcat*⌋)
𝔈(⌊*Mα*⌋)(⌊*NTMap*⌋)(⌊*a* ⊗$_{HM.O}$𝔈(⌊*Mcf*⌋) *b*, *c*, *d*⌋$_\bullet$
and *mcat-triangle*[*mcat-cs-simps*]:
⟦ *a* ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋); *b* ∈$_\circ$ 𝔈(⌊*Mcat*⌋)(⌊*Obj*⌋) ⟧ ⟹
(𝔈(⌊*Mcat*⌋)(⌊*CId*⌋)(⌊*a*⌋) ⊗$_{HM.A}$𝔈(⌊*Mcf*⌋) 𝔈(⌊*Ml*⌋)(⌊*NTMap*⌋)(⌊*b*⌋)) ∘$_A$𝔈(⌊*Mcat*⌋)
𝔈(⌊*Mα*⌋)(⌊*NTMap*⌋)(⌊*a*, 𝔈(⌊*Me*⌋), *b*⌋$_\bullet$ =

$$(\mathfrak{C}(\!|Mr|\!)(\!|NTMap|\!)(\!|a|\!) \otimes_{HM.A} \mathfrak{C}(\!|Mcf|\!)\ \mathfrak{C}(\!|Mcat|\!)(\!|CId|\!)(\!|b|\!))$$

lemmas [*mcat-cs-intros*] = *monoidal-category.mcat-Me-is-obj*
lemmas [*mcat-cs-simps*] = *monoidal-category.mcat-triangle*

Rules.

lemma (**in** *monoidal-category*) *monoidal-category-axioms'*[*mcat-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *monoidal-category* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule monoidal-category-axioms*)

mk-ide rf *monoidal-category-def*[*unfolded monoidal-category-axioms-def*]
  |*intro monoidal-categoryI*|
  |*dest monoidal-categoryD*[*dest*]|
  |*elim monoidal-categoryE*[*elim*]|

Elementary properties.

lemma *mcat-eqI*:
  assumes *monoidal-category* α $\mathfrak{A}$
    and *monoidal-category* α $\mathfrak{B}$
    and $\mathfrak{A}(\!|Mcat|\!) = \mathfrak{B}(\!|Mcat|\!)$
    and $\mathfrak{A}(\!|Mcf|\!) = \mathfrak{B}(\!|Mcf|\!)$
    and $\mathfrak{A}(\!|Me|\!) = \mathfrak{B}(\!|Me|\!)$
    and $\mathfrak{A}(\!|M\alpha|\!) = \mathfrak{B}(\!|M\alpha|\!)$
    and $\mathfrak{A}(\!|Ml|\!) = \mathfrak{B}(\!|Ml|\!)$
    and $\mathfrak{A}(\!|Mr|\!) = \mathfrak{B}(\!|Mr|\!)$
  shows $\mathfrak{A} = \mathfrak{B}$
proof–
  interpret $\mathfrak{A}$: *monoidal-category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))
  interpret $\mathfrak{B}$: *monoidal-category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
  show *?thesis*
  proof(*rule vsv-eqI*)
    have *dom*: $\mathcal{D}_\circ\ \mathfrak{A} = 6_{\mathbb{N}}$
      by (*cs-concl* **cs-shallow cs-simp**: *mcat-cs-simps V-cs-simps*)
    show $\mathcal{D}_\circ\ \mathfrak{A} = \mathcal{D}_\circ\ \mathfrak{B}$
      by (*cs-concl* **cs-shallow cs-simp**: *mcat-cs-simps V-cs-simps*)
    show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{A} \Longrightarrow \mathfrak{A}(\!|a|\!) = \mathfrak{B}(\!|a|\!)$ **for** *a*
      by (*unfold dom*, *elim-in-numeral*, *insert assms*)
        (*auto simp*: *mcat-field-simps*)
  qed *auto*
qed

34.5 Components for $M\alpha$ for *Rel*

34.5.1 Definition and elementary properties

definition $M\alpha$-*Rel-arrow-lr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where $M\alpha$-*Rel-arrow-lr* A B C =
    [
      ($\lambda ab\text{-}c \in_\circ (A \times_\circ B) \times_\circ C$. $\langle vfst\ (vfst\ ab\text{-}c),\ \langle vsnd\ (vfst\ ab\text{-}c),\ vsnd\ ab\text{-}c\rangle\rangle$),
      $(A \times_\circ B) \times_\circ C$,
      $A \times_\circ (B \times_\circ C)$
    ]$_\circ$

definition $M\alpha$-*Rel-arrow-rl* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where $M\alpha$-*Rel-arrow-rl* A B C =
    [
      ($\lambda a\text{-}bc \in_\circ A \times_\circ (B \times_\circ C)$. $\langle\langle vfst\ a\text{-}bc,\ vfst\ (vsnd\ a\text{-}bc)\rangle,\ vsnd\ (vsnd\ a\text{-}bc)\rangle$),

$A \times_\circ (B \times_\circ C)$,
$(A \times_\circ B) \times_\circ C$
$]_\circ$

Components.

lemma *Mα-Rel-arrow-lr-components*:
shows *Mα-Rel-arrow-lr A B C*(|*ArrVal*|) =
$(\lambda ab\text{-}c \in_\circ (A \times_\circ B) \times_\circ C.\ \langle vfst\ (vfst\ ab\text{-}c),\ \langle vsnd\ (vfst\ ab\text{-}c),\ vsnd\ ab\text{-}c\rangle\rangle)$
and [*cat-cs-simps*]: *Mα-Rel-arrow-lr A B C*(|*ArrDom*|) = $(A \times_\circ B) \times_\circ C$
and [*cat-cs-simps*]: *Mα-Rel-arrow-lr A B C*(|*ArrCod*|) = $A \times_\circ (B \times_\circ C)$
unfolding *Mα-Rel-arrow-lr-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *Mα-Rel-arrow-rl-components*:
shows *Mα-Rel-arrow-rl A B C*(|*ArrVal*|) =
$(\lambda a\text{-}bc \in_\circ A \times_\circ (B \times_\circ C).\ \langle\langle vfst\ a\text{-}bc,\ vfst\ (vsnd\ a\text{-}bc)\rangle,\ vsnd\ (vsnd\ a\text{-}bc)\rangle)$
and [*cat-cs-simps*]: *Mα-Rel-arrow-rl A B C*(|*ArrDom*|) = $A \times_\circ (B \times_\circ C)$
and [*cat-cs-simps*]: *Mα-Rel-arrow-rl A B C*(|*ArrCod*|) = $(A \times_\circ B) \times_\circ C$
unfolding *Mα-Rel-arrow-rl-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

34.5.2 Arrow value

mk-VLambda *Mα-Rel-arrow-lr-components*(*1*)
|*vsv Mα-Rel-arrow-lr-ArrVal-vsv*[*cat-cs-intros*]|
|*vdomain Mα-Rel-arrow-lr-ArrVal-vdomain*[*cat-cs-simps*]|
|*app Mα-Rel-arrow-lr-ArrVal-app′*|

lemma *Mα-Rel-arrow-lr-ArrVal-app*[*cat-cs-simps*]:
assumes *ab-c* = $\langle\langle a,\ b\rangle,\ c\rangle$ **and** *ab-c* $\in_\circ$ $(A \times_\circ B) \times_\circ C$
shows *Mα-Rel-arrow-lr A B C*(|*ArrVal*|)(|*ab-c*|) = $\langle a,\ \langle b,\ c\rangle\rangle$
using *assms*(*2*)
unfolding *assms*(*1*)
by (*simp-all add*: *Mα-Rel-arrow-lr-ArrVal-app′ nat-omega-simps*)

mk-VLambda *Mα-Rel-arrow-rl-components*(*1*)
|*vsv Mα-Rel-arrow-rl-ArrVal-vsv*[*cat-cs-intros*]|
|*vdomain Mα-Rel-arrow-rl-ArrVal-vdomain*[*cat-cs-simps*]|
|*app Mα-Rel-arrow-rl-ArrVal-app′*|

lemma *Mα-Rel-arrow-rl-ArrVal-app*[*cat-cs-simps*]:
assumes *a-bc* = $\langle a,\ \langle b,\ c\rangle\rangle$ **and** *a-bc* $\in_\circ$ $A \times_\circ (B \times_\circ C)$
shows *Mα-Rel-arrow-rl A B C*(|*ArrVal*|)(|*a-bc*|) = $\langle\langle a,\ b\rangle,\ c\rangle$
using *assms*(*2*)
unfolding *assms*(*1*)
by (*simp-all add*: *Mα-Rel-arrow-rl-ArrVal-app′ nat-omega-simps*)

34.5.3 Components for *Mα* for *Rel* are arrows

lemma (**in** $\mathcal{Z}$) *Mα-Rel-arrow-lr-is-cat-Set-arr-Vset*:
assumes *A* $\in_\circ$ *Vset α* **and** *B* $\in_\circ$ *Vset α* **and** *C* $\in_\circ$ *Vset α*
shows *Mα-Rel-arrow-lr A B C* : $(A \times_\circ B) \times_\circ C \mapsto_{cat\text{-}Set\ \alpha} A \times_\circ (B \times_\circ C)$
proof(*intro cat-Set-is-arrI arr-SetI*)
show *vfsequence* (*Mα-Rel-arrow-lr A B C*) **unfolding** *Mα-Rel-arrow-lr-def* **by** *auto*
show *vcard* (*Mα-Rel-arrow-lr A B C*) = $\mathfrak{Z}_\mathbb{N}$
unfolding *Mα-Rel-arrow-lr-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*Mα-Rel-arrow-lr A B C*(|*ArrVal*|)) $\subseteq_\circ$ *Mα-Rel-arrow-lr A B C*(|*ArrCod*|)
unfolding *Mα-Rel-arrow-lr-components* **by** *auto*
qed
(

```
    use assms in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
      ›
  )+

lemma (in 𝒵) Mα-Rel-arrow-rl-is-cat-Set-arr-Vset:
  assumes A ∈ₒ Vset α and B ∈ₒ Vset α and C ∈ₒ Vset α
  shows Mα-Rel-arrow-rl A B C : A ×ₒ (B ×ₒ C) ↦_{cat-Set α} (A ×ₒ B) ×ₒ C
proof(intro cat-Set-is-arrI arr-SetI)
  show vfsequence (Mα-Rel-arrow-rl A B C) unfolding Mα-Rel-arrow-rl-def by auto
  show vcard (Mα-Rel-arrow-rl A B C) = 3_ℕ
    unfolding Mα-Rel-arrow-rl-def by (simp add: nat-omega-simps)
  show ℛₒ (Mα-Rel-arrow-rl A B C⦇ArrVal⦈) ⊆ₒ Mα-Rel-arrow-rl A B C⦇ArrCod⦈
    unfolding Mα-Rel-arrow-rl-components by auto
qed
  (
    use assms in
      ‹
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
      ›
  )+

lemma (in 𝒵) Mα-Rel-arrow-lr-is-cat-Set-arr:
  assumes A ∈ₒ cat-Set α⦇Obj⦈
    and B ∈ₒ cat-Set α⦇Obj⦈
    and C ∈ₒ cat-Set α⦇Obj⦈
  shows Mα-Rel-arrow-lr A B C : (A ×ₒ B) ×ₒ C ↦_{cat-Set α} A ×ₒ (B ×ₒ C)
  using assms
  unfolding cat-Set-components
  by (rule Mα-Rel-arrow-lr-is-cat-Set-arr-Vset)

lemma (in 𝒵) Mα-Rel-arrow-lr-is-cat-Set-arr'[cat-rel-par-Set-cs-intros]:
  assumes A ∈ₒ cat-Set α⦇Obj⦈
    and B ∈ₒ cat-Set α⦇Obj⦈
    and C ∈ₒ cat-Set α⦇Obj⦈
    and A' = (A ×ₒ B) ×ₒ C
    and B' = A ×ₒ (B ×ₒ C)
    and 𝔊' = cat-Set α
  shows Mα-Rel-arrow-lr A B C : A' ↦_{𝔊'} B'
  using assms(1–3) unfolding assms(4–6) by (rule Mα-Rel-arrow-lr-is-cat-Set-arr)

lemmas [cat-rel-par-Set-cs-intros] = 𝒵.Mα-Rel-arrow-lr-is-cat-Set-arr'

lemma (in 𝒵) Mα-Rel-arrow-rl-is-cat-Set-arr:
  assumes A ∈ₒ cat-Set α⦇Obj⦈
    and B ∈ₒ cat-Set α⦇Obj⦈
    and C ∈ₒ cat-Set α⦇Obj⦈
  shows Mα-Rel-arrow-rl A B C : A ×ₒ (B ×ₒ C) ↦_{cat-Set α} (A ×ₒ B) ×ₒ C
  using assms
  unfolding cat-Set-components
  by (rule Mα-Rel-arrow-rl-is-cat-Set-arr-Vset)

lemma (in 𝒵) Mα-Rel-arrow-rl-is-cat-Set-arr'[cat-rel-par-Set-cs-intros]:
  assumes A ∈ₒ cat-Set α⦇Obj⦈
    and B ∈ₒ cat-Set α⦇Obj⦈
```

and $C \in_{\circ} \mathit{cat\text{-}Set}\ \alpha(\!|\mathit{Obj}|\!)$
and $A' = A \times_{\circ} (B \times_{\circ} C)$
and $B' = (A \times_{\circ} B) \times_{\circ} C$
and $\mathfrak{C}' = \mathit{cat\text{-}Set}\ \alpha$
shows $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule* $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Set\text{-}arr}$)

lemmas [*cat-rel-par-Set-cs-intros*] = $\mathcal{Z}.M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Set\text{-}arr}'$

lemma (in $\mathcal{Z}$**)** $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr\text{-}is\text{-}cat\text{-}Par\text{-}arr}$:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $C \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
shows $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C : (A \times_{\circ} B) \times_{\circ} C \mapsto_{\mathit{cat\text{-}Par}\ \alpha} A \times_{\circ} (B \times_{\circ} C)$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
from *assms* **show** *?thesis*
unfolding *cat-Par-components*(*1*)
by (*intro Set-Par.subcat-is-arrD* $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr\text{-}is\text{-}cat\text{-}Set\text{-}arr\text{-}Vset}$) *auto*
qed

lemma (in $\mathcal{Z}$**)** $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr\text{-}is\text{-}cat\text{-}Par\text{-}arr}'$[*cat-rel-Par-set-cs-intros*]:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $C \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $A' = (A \times_{\circ} B) \times_{\circ} C$
and $B' = A \times_{\circ} (B \times_{\circ} C)$
and $\mathfrak{C}' = \mathit{cat\text{-}Par}\ \alpha$
shows $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule* $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr\text{-}is\text{-}cat\text{-}Par\text{-}arr}$)

lemmas [*cat-rel-Par-set-cs-intros*] = $\mathcal{Z}.M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr\text{-}is\text{-}cat\text{-}Par\text{-}arr}'$

lemma (in $\mathcal{Z}$**)** $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Par\text{-}arr}$:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $C \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
shows $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C : A \times_{\circ} (B \times_{\circ} C) \mapsto_{\mathit{cat\text{-}Par}\ \alpha} (A \times_{\circ} B) \times_{\circ} C$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
from *assms* **show** *?thesis*
unfolding *cat-Par-components*(*1*)
by (*intro Set-Par.subcat-is-arrD* $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Set\text{-}arr\text{-}Vset}$) *auto*
qed

lemma (in $\mathcal{Z}$**)** $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Par\text{-}arr}'$[*cat-rel-Par-set-cs-intros*]:
assumes $A \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $C \in_{\circ} \mathit{cat\text{-}Par}\ \alpha(\!|\mathit{Obj}|\!)$
and $A' = A \times_{\circ} (B \times_{\circ} C)$
and $B' = (A \times_{\circ} B) \times_{\circ} C$
and $\mathfrak{C}' = \mathit{cat\text{-}Par}\ \alpha$
shows $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule* $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Par\text{-}arr}$)

lemmas [*cat-rel-Par-set-cs-intros*] = $\mathcal{Z}.M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Par\text{-}arr}'$

lemma (in $\mathcal{Z}$**)** *Mα-Rel-arrow-lr-is-cat-Rel-arr*:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $C \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
shows $M\alpha\text{-}Rel\text{-}arrow\text{-}lr\ A\ B\ C : (A \times_\circ B) \times_\circ C \mapsto_{cat\text{-}Rel\ \alpha} A \times_\circ (B \times_\circ C)$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
interpret *Par-Rel*: *wide-replete-subcategory* α ‹*cat-Par* α› ‹*cat-Rel* α›
by (*rule wide-replete-subcategory-cat-Par-cat-Rel*)
interpret *Set-Rel*: *subcategory* α ‹*cat-Set* α› ‹*cat-Rel* α›
by
(
rule subcat-trans[
OF Set-Par.subcategory-axioms Par-Rel.subcategory-axioms
]
)
from *assms* **show** *?thesis*
unfolding *cat-Rel-components*(*1*)
by (*intro Set-Rel.subcat-is-arrD Mα-Rel-arrow-lr-is-cat-Set-arr-Vset*) *auto*
qed

lemma (in $\mathcal{Z}$**)** *Mα-Rel-arrow-lr-is-cat-Rel-arr'*[*cat-Rel-par-set-cs-intros*]:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $C \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $A' = (A \times_\circ B) \times_\circ C$
and $B' = A \times_\circ (B \times_\circ C)$
and $\mathfrak{C}' =$ *cat-Rel* α
shows $M\alpha\text{-}Rel\text{-}arrow\text{-}lr\ A\ B\ C : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule Mα-Rel-arrow-lr-is-cat-Rel-arr*)

lemmas [*cat-Rel-par-set-cs-intros*] = $\mathcal{Z}$*.Mα-Rel-arrow-lr-is-cat-Rel-arr'*

lemma (in $\mathcal{Z}$**)** *Mα-Rel-arrow-rl-is-cat-Rel-arr*:
assumes $A \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $B \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
and $C \in_\circ$ *cat-Rel* $\alpha(\!|Obj|\!)$
shows $M\alpha\text{-}Rel\text{-}arrow\text{-}rl\ A\ B\ C : A \times_\circ (B \times_\circ C) \mapsto_{cat\text{-}Rel\ \alpha} (A \times_\circ B) \times_\circ C$
proof–
interpret *Set-Par*: *wide-replete-subcategory* α ‹*cat-Set* α› ‹*cat-Par* α›
by (*rule wide-replete-subcategory-cat-Set-cat-Par*)
interpret *Par-Rel*: *wide-replete-subcategory* α ‹*cat-Par* α› ‹*cat-Rel* α›
by (*rule wide-replete-subcategory-cat-Par-cat-Rel*)
interpret *Set-Rel*: *subcategory* α ‹*cat-Set* α› ‹*cat-Rel* α›
by
(
rule subcat-trans[
OF Set-Par.subcategory-axioms Par-Rel.subcategory-axioms
]
)
from *assms* **show** *?thesis*
unfolding *cat-Rel-components*(*1*)
by (*intro Set-Rel.subcat-is-arrD Mα-Rel-arrow-rl-is-cat-Set-arr-Vset*) *auto*
qed

lemma (in $\mathcal{Z}$**)** *Mα-Rel-arrow-rl-is-cat-Rel-arr'*[*cat-Rel-par-set-cs-intros*]:

assumes $A \in_{\circ} \mathit{cat\text{-}Rel}\ \alpha(\!|\mathit{Obj}|\!)$
and $B \in_{\circ} \mathit{cat\text{-}Rel}\ \alpha(\!|\mathit{Obj}|\!)$
and $C \in_{\circ} \mathit{cat\text{-}Rel}\ \alpha(\!|\mathit{Obj}|\!)$
and $A' = A \times_{\circ} (B \times_{\circ} C)$
and $B' = (A \times_{\circ} B) \times_{\circ} C$
and $\mathfrak{C}' = \mathit{cat\text{-}Rel}\ \alpha$
shows $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(*1–3*) **unfolding** *assms*(*4–6*) **by** (*rule* $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Rel\text{-}arr}$)

lemmas [*cat-Rel-par-set-cs-intros*] = $\mathcal{Z}.M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}is\text{-}cat\text{-}Rel\text{-}arr}'$

34.5.4 Further properties

lemma (in $\mathcal{Z}$**)** $M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl\text{-}}M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}$[*cat-cs-simps*]:
assumes $A \in_{\circ} \mathit{Vset}\ \alpha$ **and** $B \in_{\circ} \mathit{Vset}\ \alpha$ **and** $C \in_{\circ} \mathit{Vset}\ \alpha$
shows
$M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C \circ_{A\,\mathit{cat\text{-}Set}\ \alpha} M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C =$
$\mathit{cat\text{-}Set}\ \alpha(\!|\mathit{CId}|\!)(\!|(A \times_{\circ} B) \times_{\circ} C|\!)$
proof–
interpret *Set*: *category* α ‹*cat-Set* α›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros*)
from *assms* **have** *lhs*:
$M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C \circ_{A\,\mathit{cat\text{-}Set}\ \alpha} M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C :$
$(A \times_{\circ} B) \times_{\circ} C \mapsto_{\mathit{cat\text{-}Set}\ \alpha} (A \times_{\circ} B) \times_{\circ} C$
by
(
cs-concl **cs-shallow**
cs-simp: *cat-Set-components*(*1*)
cs-intro: *cat-rel-par-Set-cs-intros cat-cs-intros*
)
then have *dom-lhs*:
$\mathcal{D}_{\circ}((M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C \circ_{A\,\mathit{cat\text{-}Set}\ \alpha} M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C)(\!|\mathit{ArrVal}|\!)) =$
$(A \times_{\circ} B) \times_{\circ} C$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
from *assms Set.category-axioms* **have** *rhs*:
$\mathit{cat\text{-}Set}\ \alpha(\!|\mathit{CId}|\!)(\!|(A \times_{\circ} B) \times_{\circ} C|\!) :$
$(A \times_{\circ} B) \times_{\circ} C \mapsto_{\mathit{cat\text{-}Set}\ \alpha} (A \times_{\circ} B) \times_{\circ} C$
by
(
cs-concl
cs-simp: *cat-Set-components*(*1*) **cs-intro**: *V-cs-intros cat-cs-intros*
)
then have *dom-rhs*:
$\mathcal{D}_{\circ}((\mathit{cat\text{-}Set}\ \alpha(\!|\mathit{CId}|\!)(\!|(A \times_{\circ} B) \times_{\circ} C|\!))(\!|\mathit{ArrVal}|\!)) = (A \times_{\circ} B) \times_{\circ} C$
by (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)
show *?thesis*
proof(*rule arr-Set-eqI*)
from *lhs* **show** *arr-Set-lhs*:
$\mathit{arr\text{-}Set}\ \alpha\ (M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C \circ_{A\,\mathit{cat\text{-}Set}\ \alpha} M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C)$
by (*auto dest*: *cat-Set-is-arrD*(*1*))
from *rhs* **show** *arr-Set-rhs*: $\mathit{arr\text{-}Set}\ \alpha\ (\mathit{cat\text{-}Set}\ \alpha(\!|\mathit{CId}|\!)(\!|(A \times_{\circ} B) \times_{\circ} C|\!))$
by (*auto dest*: *cat-Set-is-arrD*(*1*))
show
$(M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}rl}\ A\ B\ C \circ_{A\,\mathit{cat\text{-}Set}\ \alpha} M\alpha\text{-}\mathit{Rel\text{-}arrow\text{-}lr}\ A\ B\ C)(\!|\mathit{ArrVal}|\!) =$
$\mathit{cat\text{-}Set}\ \alpha(\!|\mathit{CId}|\!)(\!|(A \times_{\circ} B) \times_{\circ} C|\!)(\!|\mathit{ArrVal}|\!)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *ab-c* **assume** *prems*: $\mathit{ab\text{-}c} \in_{\circ} (A \times_{\circ} B) \times_{\circ} C$
then obtain *a b c*

```
      where ab-c-def: ab-c = ⟨⟨a, b⟩, c⟩
        and a: a ∈∘ A
        and b: b ∈∘ B
        and c: c ∈∘ C
      by clarsimp
    from assms prems a b c lhs rhs show
      (Mα-Rel-arrow-rl A B C ∘A cat-Set α Mα-Rel-arrow-lr A B C)(|ArrVal|)(|ab-c|) =
        cat-Set α(|CId|)(|(A ×∘ B) ×∘ C|)(|ArrVal|)(|ab-c|)
      unfolding ab-c-def
      by
        (
          cs-concl
            cs-simp: cat-Set-components(1) cat-cs-simps
            cs-intro: cat-rel-par-Set-cs-intros V-cs-intros cat-cs-intros
        )
  qed (use arr-Set-lhs arr-Set-rhs in auto)
 qed (use lhs rhs in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
qed

lemma (in Z) Mα-Rel-arrow-rl-Mα-Rel-arrow-lr'[cat-cs-simps]:
 assumes A ∈∘ cat-Set α(|Obj|)
   and B ∈∘ cat-Set α(|Obj|)
   and C ∈∘ cat-Set α(|Obj|)
 shows
   Mα-Rel-arrow-rl A B C ∘A cat-Set α Mα-Rel-arrow-lr A B C =
     cat-Set α(|CId|)(|(A ×∘ B) ×∘ C|)
 using assms
 unfolding cat-Set-components(1)
 by (rule Mα-Rel-arrow-rl-Mα-Rel-arrow-lr)

lemmas [cat-cs-simps] = Z.Mα-Rel-arrow-rl-Mα-Rel-arrow-lr'

lemma (in Z) Mα-Rel-arrow-lr-Mα-Rel-arrow-rl[cat-cs-simps]:
 assumes A ∈∘ Vset α and B ∈∘ Vset α and C ∈∘ Vset α
 shows
   Mα-Rel-arrow-lr A B C ∘A cat-Set α Mα-Rel-arrow-rl A B C =
     cat-Set α(|CId|)(|A ×∘ (B ×∘ C)|)
proof–
 interpret Set: category α ‹cat-Set α›
   by (cs-concl cs-shallow cs-intro: cat-cs-intros)
 from assms have lhs:
   Mα-Rel-arrow-lr A B C ∘A cat-Set α Mα-Rel-arrow-rl A B C :
     A ×∘ (B ×∘ C) ↦cat-Set α A ×∘ (B ×∘ C)
   by
     (
       cs-concl cs-shallow
         cs-simp: cat-Set-components(1)
         cs-intro: cat-rel-par-Set-cs-intros cat-cs-intros
     )
 then have dom-lhs:
   𝒟∘ ((Mα-Rel-arrow-lr A B C ∘A cat-Set α Mα-Rel-arrow-rl A B C)(|ArrVal|)) =
     A ×∘ (B ×∘ C)
   by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
 from assms Set.category-axioms have rhs:
   cat-Set α(|CId|)(|A ×∘ (B ×∘ C)|) :
     A ×∘ (B ×∘ C) ↦cat-Set α A ×∘ (B ×∘ C)
   by
     (
```

```
      cs-concl
        cs-simp: cat-Set-components(1) cs-intro: V-cs-intros cat-cs-intros
    )
  then have dom-rhs:
    𝒟∘ ((cat-Set α(‖CId‖)(‖A ×∘ (B ×∘ C)‖))(‖ArrVal‖)) = A ×∘ (B ×∘ C)
    by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: cat-cs-intros)
  show ?thesis
  proof(rule arr-Set-eqI)
    from lhs show arr-Set-lhs:
      arr-Set α (Mα-Rel-arrow-lr A B C ∘_{A cat-Set α} Mα-Rel-arrow-rl A B C)
      by (auto dest: cat-Set-is-arrD(1))
    from rhs show arr-Set-rhs: arr-Set α (cat-Set α(‖CId‖)(‖A ×∘ (B ×∘ C)‖))
      by (auto dest: cat-Set-is-arrD(1))
    show
      (Mα-Rel-arrow-lr A B C ∘_{A cat-Set α} Mα-Rel-arrow-rl A B C)(‖ArrVal‖) =
        cat-Set α(‖CId‖)(‖A ×∘ (B ×∘ C)‖)(‖ArrVal‖)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a-bc assume prems: a-bc ∈∘ A ×∘ (B ×∘ C)
      then obtain a b c
        where a-bc-def: a-bc = ⟨a, ⟨b, c⟩⟩
          and a: a ∈∘ A
          and b: b ∈∘ B
          and c: c ∈∘ C
        by clarsimp
      from assms prems a b c lhs rhs show
        (Mα-Rel-arrow-lr A B C ∘_{A cat-Set α} Mα-Rel-arrow-rl A B C)(‖ArrVal‖)(‖a-bc‖) =
          cat-Set α(‖CId‖)(‖A ×∘ (B ×∘ C)‖)(‖ArrVal‖)(‖a-bc‖)
        unfolding a-bc-def
        by
          (
            cs-concl
              cs-simp: cat-Set-components(1) cat-cs-simps
              cs-intro: V-cs-intros cat-rel-par-Set-cs-intros cat-cs-intros
          )
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed (use lhs rhs in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
qed

lemma (in 𝒵) Mα-Rel-arrow-lr-Mα-Rel-arrow-rl'[cat-cs-simps]:
  assumes A ∈∘ cat-Set α(‖Obj‖)
    and B ∈∘ cat-Set α(‖Obj‖)
    and C ∈∘ cat-Set α(‖Obj‖)
  shows
    Mα-Rel-arrow-lr A B C ∘_{A cat-Set α} Mα-Rel-arrow-rl A B C =
      cat-Set α(‖CId‖)(‖A ×∘ (B ×∘ C)‖)
  using assms
  unfolding cat-Set-components(1)
  by (rule Mα-Rel-arrow-lr-Mα-Rel-arrow-rl)

lemmas [cat-cs-simps] = 𝒵.Mα-Rel-arrow-lr-Mα-Rel-arrow-rl'
```

34.5.5 Components for $M\alpha$ for Rel are isomorphisms

```
lemma (in 𝒵)
  assumes A ∈∘ Vset α and B ∈∘ Vset α and C ∈∘ Vset α
  shows Mα-Rel-arrow-lr-is-cat-Set-iso-arr-Vset:
    Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦_{iso cat-Set α} A ×∘ (B ×∘ C)
    and Mα-Rel-arrow-rl-is-cat-Set-iso-arr-Vset:
```

```
    Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Set α (A ×∘ B) ×∘ C
proof–
  interpret Set: category α ‹cat-Set α›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros)
  have lhs: Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦cat-Set α (A ×∘ B) ×∘ C
    by (intro Mα-Rel-arrow-rl-is-cat-Set-arr-Vset assms)
  from assms have [cat-cs-simps]:
    Mα-Rel-arrow-rl A B C ∘A cat-Set α Mα-Rel-arrow-lr A B C =
      cat-Set α(|CId|)(|(A ×∘ B) ×∘ C|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Set-components(1) cat-cs-simps cs-intro: cat-cs-intros
      )
  from assms have [cat-cs-simps]:
    Mα-Rel-arrow-lr A B C ∘A cat-Set α Mα-Rel-arrow-rl A B C =
      cat-Set α(|CId|)(|A ×∘ B ×∘ C|)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Set-components(1) cat-cs-simps cs-intro: cat-cs-intros
      )
  from
    Set.is-iso-arrI'
      [
        OF lhs Mα-Rel-arrow-lr-is-cat-Set-arr-Vset[OF assms],
        unfolded cat-cs-simps,
        simplified
      ]
  show Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦iso cat-Set α A ×∘ (B ×∘ C)
    and Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Set α (A ×∘ B) ×∘ C
    by auto
qed

lemma (in Z)
  assumes A ∈∘ cat-Set α(|Obj|)
    and B ∈∘ cat-Set α(|Obj|)
    and C ∈∘ cat-Set α(|Obj|)
  shows Mα-Rel-arrow-lr-is-cat-Set-iso-arr:
    Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦iso cat-Set α A ×∘ (B ×∘ C)
    and Mα-Rel-arrow-rl-is-cat-Set-iso-arr:
    Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Set α (A ×∘ B) ×∘ C
  using assms
  unfolding cat-Set-components(1)
  by
    (
      all
        ‹
          intro
            Mα-Rel-arrow-lr-is-cat-Set-iso-arr-Vset
            Mα-Rel-arrow-rl-is-cat-Set-iso-arr-Vset
        ›
    )

lemma (in Z)
  Mα-Rel-arrow-lr-is-cat-Set-iso-arr'[cat-rel-par-Set-cs-intros]:
  assumes A ∈∘ cat-Set α(|Obj|)
    and B ∈∘ cat-Set α(|Obj|)
```

```
    and C ∈∘ cat-Set α(|Obj|)
    and A′ = (A ×∘ B) ×∘ C
    and B′ = A ×∘ (B ×∘ C)
    and 𝔈′ = cat-Set α
  shows Mα-Rel-arrow-lr A B C : A′ ↦iso𝔈′ B′
  using assms(1–3)
  unfolding assms(4–6)
  by (rule Mα-Rel-arrow-lr-is-cat-Set-iso-arr)

lemmas [cat-rel-par-Set-cs-intros] =
  𝒵.Mα-Rel-arrow-lr-is-cat-Set-iso-arr′

lemma (in 𝒵)
  Mα-Rel-arrow-rl-is-cat-Set-iso-arr′[cat-rel-par-Set-cs-intros]:
  assumes A ∈∘ cat-Set α(|Obj|)
    and B ∈∘ cat-Set α(|Obj|)
    and C ∈∘ cat-Set α(|Obj|)
    and A′ = A ×∘ (B ×∘ C)
    and B′ = (A ×∘ B) ×∘ C
    and 𝔈′ = cat-Set α
  shows Mα-Rel-arrow-rl A B C : A′ ↦iso𝔈′ B′
  using assms(1–3)
  unfolding assms(4–6)
  by (rule Mα-Rel-arrow-rl-is-cat-Set-iso-arr)

lemmas [cat-rel-par-Set-cs-intros] =
  𝒵.Mα-Rel-arrow-rl-is-cat-Set-iso-arr′

lemma (in 𝒵)
  assumes A ∈∘ cat-Par α(|Obj|)
    and B ∈∘ cat-Par α(|Obj|)
    and C ∈∘ cat-Par α(|Obj|)
  shows Mα-Rel-arrow-lr-is-cat-Par-iso-arr:
    Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦iso cat-Par α A ×∘ (B ×∘ C)
    and Mα-Rel-arrow-rl-is-cat-Par-iso-arr:
    Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Par α (A ×∘ B) ×∘ C
proof–
  interpret Set-Par: wide-replete-subcategory α ‹cat-Set α› ‹cat-Par α›
    by (rule wide-replete-subcategory-cat-Set-cat-Par)
  show Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦iso cat-Par α A ×∘ (B ×∘ C)
    by
      (
        rule Set-Par.wr-subcat-is-iso-arr-is-iso-arr
          [
            THEN iffD1,
            OF Mα-Rel-arrow-lr-is-cat-Set-iso-arr-Vset[
              OF assms[unfolded cat-Par-components]
              ]
          ]
      )
  show Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Par α (A ×∘ B) ×∘ C
    by
      (
        rule Set-Par.wr-subcat-is-iso-arr-is-iso-arr
          [
            THEN iffD1,
            OF Mα-Rel-arrow-rl-is-cat-Set-iso-arr-Vset[
              OF assms[unfolded cat-Par-components]
```

```
          ]
        ]
    )
qed

lemma (in Z)
  Mα-Rel-arrow-lr-is-cat-Par-iso-arr'[cat-rel-Par-set-cs-intros]:
  assumes A ∈∘ cat-Par α(|Obj|)
    and B ∈∘ cat-Par α(|Obj|)
    and C ∈∘ cat-Par α(|Obj|)
    and A' = (A ×∘ B) ×∘ C
    and B' = A ×∘ (B ×∘ C)
    and 𝔈' = cat-Par α
  shows Mα-Rel-arrow-lr A B C : A' ↦iso𝔈' B'
  using assms(1–3)
  unfolding assms(4–6)
  by (rule Mα-Rel-arrow-lr-is-cat-Par-iso-arr)

lemmas [cat-rel-Par-set-cs-intros] =
  Z.Mα-Rel-arrow-lr-is-cat-Par-iso-arr'

lemma (in Z)
  Mα-Rel-arrow-rl-is-cat-Par-iso-arr'[cat-rel-Par-set-cs-intros]:
  assumes A ∈∘ cat-Par α(|Obj|)
    and B ∈∘ cat-Par α(|Obj|)
    and C ∈∘ cat-Par α(|Obj|)
    and A' = A ×∘ (B ×∘ C)
    and B' = (A ×∘ B) ×∘ C
    and 𝔈' = cat-Par α
  shows Mα-Rel-arrow-rl A B C : A' ↦iso𝔈' B'
  using assms(1–3)
  unfolding assms(4–6)
  by (rule Mα-Rel-arrow-rl-is-cat-Par-iso-arr)

lemmas [cat-rel-Par-set-cs-intros] =
  Z.Mα-Rel-arrow-rl-is-cat-Par-iso-arr'

lemma (in Z)
  assumes A ∈∘ cat-Rel α(|Obj|)
    and B ∈∘ cat-Rel α(|Obj|)
    and C ∈∘ cat-Rel α(|Obj|)
  shows Mα-Rel-arrow-lr-is-cat-Rel-iso-arr:
    Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦iso cat-Rel α A ×∘ (B ×∘ C)
    and Mα-Rel-arrow-rl-is-cat-Rel-iso-arr:
    Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Rel α (A ×∘ B) ×∘ C
proof–
  interpret Set-Par: wide-replete-subcategory α ‹cat-Set α› ‹cat-Par α›
    by (rule wide-replete-subcategory-cat-Set-cat-Par)
  interpret Par-Rel: wide-replete-subcategory α ‹cat-Par α› ‹cat-Rel α›
    by (rule wide-replete-subcategory-cat-Par-cat-Rel)
  interpret Set-Rel: wide-replete-subcategory α ‹cat-Set α› ‹cat-Rel α›
    by
      (
        rule wr-subcat-trans
          [
            OF
              Set-Par.wide-replete-subcategory-axioms
              Par-Rel.wide-replete-subcategory-axioms
```

```
        ]
    )
  show Mα-Rel-arrow-lr A B C : (A ×∘ B) ×∘ C ↦iso cat-Rel α A ×∘ (B ×∘ C)
    by
    (
      rule Set-Rel.wr-subcat-is-iso-arr-is-iso-arr
        [
          THEN iffD1,
          OF Mα-Rel-arrow-lr-is-cat-Set-iso-arr-Vset[
            OF assms[unfolded cat-Rel-components]
            ]
        ]
    )
  show Mα-Rel-arrow-rl A B C : A ×∘ (B ×∘ C) ↦iso cat-Rel α (A ×∘ B) ×∘ C
    by
    (
      rule Set-Rel.wr-subcat-is-iso-arr-is-iso-arr
        [
          THEN iffD1,
          OF Mα-Rel-arrow-rl-is-cat-Set-iso-arr-Vset[
            OF assms[unfolded cat-Rel-components]
            ]
        ]
    )
qed

lemma (in Z)
  Mα-Rel-arrow-lr-is-cat-Rel-iso-arr'[cat-Rel-par-set-cs-intros]:
  assumes A ∈∘ cat-Rel α(|Obj|)
    and B ∈∘ cat-Rel α(|Obj|)
    and C ∈∘ cat-Rel α(|Obj|)
    and A' = (A ×∘ B) ×∘ C
    and B' = A ×∘ (B ×∘ C)
    and 𝔈' = cat-Rel α
  shows Mα-Rel-arrow-lr A B C : A' ↦iso𝔈' B'
  using assms(1–3)
  unfolding assms(4–6)
  by (rule Mα-Rel-arrow-lr-is-cat-Rel-iso-arr)

lemmas [cat-Rel-par-set-cs-intros] =
  Z.Mα-Rel-arrow-lr-is-cat-Rel-iso-arr'

lemma (in Z)
  Mα-Rel-arrow-rl-is-cat-Rel-iso-arr'[cat-Rel-par-set-cs-intros]:
  assumes A ∈∘ cat-Rel α(|Obj|)
    and B ∈∘ cat-Rel α(|Obj|)
    and C ∈∘ cat-Rel α(|Obj|)
    and A' = A ×∘ (B ×∘ C)
    and B' = (A ×∘ B) ×∘ C
    and 𝔈' = cat-Rel α
  shows Mα-Rel-arrow-rl A B C : A' ↦iso𝔈' B'
  using assms(1–3)
  unfolding assms(4–6)
  by (rule Mα-Rel-arrow-rl-is-cat-Rel-iso-arr)

lemmas [cat-Rel-par-set-cs-intros] =
  Z.Mα-Rel-arrow-rl-is-cat-Rel-iso-arr'
```

34.6 $M\alpha$ for Rel

34.6.1 Definition and elementary properties

definition $M\alpha\text{-}Rel :: V \Rightarrow V$
  where $M\alpha\text{-}Rel\ \mathfrak{C} =$
    [
      $(\lambda abc \in_{\circ} (\mathfrak{C}\hat{}_{C3})(\lfloor Obj \rfloor).\ M\alpha\text{-}Rel\text{-}arrow\text{-}lr\ (abc(\lfloor 0 \rfloor))\ (abc(\lfloor 1_{\mathbb{N}} \rfloor))\ (abc(\lfloor 2_{\mathbb{N}} \rfloor)))$,
      $cf\text{-}blcomp\ (cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{C})$,
      $cf\text{-}brcomp\ (cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{C})$,
      $\mathfrak{C}\hat{}_{C3}$,
      $\mathfrak{C}$
    ]$_{\circ}$

Components.

lemma $M\alpha\text{-}Rel\text{-}components$:
  shows $M\alpha\text{-}Rel\ \mathfrak{C}(\lfloor NTMap \rfloor) =$
    $(\lambda abc \in_{\circ} (\mathfrak{C}\hat{}_{C3})(\lfloor Obj \rfloor).\ M\alpha\text{-}Rel\text{-}arrow\text{-}lr\ (abc(\lfloor 0 \rfloor))\ (abc(\lfloor 1_{\mathbb{N}} \rfloor))\ (abc(\lfloor 2_{\mathbb{N}} \rfloor)))$
    and [$cat\text{-}cs\text{-}simps$]: $M\alpha\text{-}Rel\ \mathfrak{C}(\lfloor NTDom \rfloor) = cf\text{-}blcomp\ (cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{C})$
    and [$cat\text{-}cs\text{-}simps$]: $M\alpha\text{-}Rel\ \mathfrak{C}(\lfloor NTCod \rfloor) = cf\text{-}brcomp\ (cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{C})$
    and [$cat\text{-}cs\text{-}simps$]: $M\alpha\text{-}Rel\ \mathfrak{C}(\lfloor NTDGDom \rfloor) = \mathfrak{C}\hat{}_{C3}$
    and [$cat\text{-}cs\text{-}simps$]: $M\alpha\text{-}Rel\ \mathfrak{C}(\lfloor NTDGCod \rfloor) = \mathfrak{C}$
  unfolding $M\alpha\text{-}Rel\text{-}def\ nt\text{-}field\text{-}simps$ **by** ($simp\text{-}all\ add$: $nat\text{-}omega\text{-}simps$)

34.6.2 Natural transformation map

mk-VLambda $M\alpha\text{-}Rel\text{-}components(1)$
  $|vsv\ M\alpha\text{-}Rel\text{-}NTMap\text{-}vsv[cat\text{-}cs\text{-}intros]|$
  $|vdomain\ M\alpha\text{-}Rel\text{-}NTMap\text{-}vdomain[cat\text{-}cs\text{-}simps]|$
  $|app\ M\alpha\text{-}Rel\text{-}NTMap\text{-}app'|$

lemma $M\alpha\text{-}Rel\text{-}NTMap\text{-}app[cat\text{-}cs\text{-}simps]$:
  assumes $ABC = [A, B, C]_{\circ}$ **and** $ABC \in_{\circ} (\mathfrak{C}\hat{}_{C3})(\lfloor Obj \rfloor)$
  shows $M\alpha\text{-}Rel\ \mathfrak{C}(\lfloor NTMap \rfloor)(\lfloor ABC \rfloor) = M\alpha\text{-}Rel\text{-}arrow\text{-}lr\ A\ B\ C$
  using $assms(2)$
  unfolding $assms(1)$
  by ($simp\ add$: $M\alpha\text{-}Rel\text{-}NTMap\text{-}app'\ nat\text{-}omega\text{-}simps$)

34.6.3 $M\alpha$ for Rel is a natural isomorphism

lemma (**in** $\mathcal{Z}$) $M\alpha\text{-}Rel\text{-}is\text{-}iso\text{-}ntcf$:
  $M\alpha\text{-}Rel\ (cat\text{-}Rel\ \alpha) :$
    $cf\text{-}blcomp\ (cf\text{-}prod\text{-}2\text{-}Rel\ (cat\text{-}Rel\ \alpha)) \mapsto_{CF.iso}$
    $cf\text{-}brcomp\ (cf\text{-}prod\text{-}2\text{-}Rel\ (cat\text{-}Rel\ \alpha)) :$
    $cat\text{-}Rel\ \alpha\hat{}_{C3} \mapsto\mapsto_{C\alpha} cat\text{-}Rel\ \alpha$
proof–

  interpret $cf\text{-}prod$: $is\text{-}functor$
    $\alpha\ \langle cat\text{-}Rel\ \alpha \times_C cat\text{-}Rel\ \alpha\rangle\ \langle cat\text{-}Rel\ \alpha\rangle\ \langle cf\text{-}prod\text{-}2\text{-}Rel\ (cat\text{-}Rel\ \alpha)\rangle$
    by ($cs\text{-}concl$ **cs-shallow cs-intro**: $cat\text{-}cs\text{-}intros\ cat\text{-}Rel\text{-}cs\text{-}intros$)

  show $?thesis$
  proof($intro\ is\text{-}iso\text{-}ntcfI\ is\text{-}ntcfI'$)

    show $vfsequence\ (M\alpha\text{-}Rel\ (cat\text{-}Rel\ \alpha))$ **unfolding** $M\alpha\text{-}Rel\text{-}def$ **by** $auto$
    show $vcard\ (M\alpha\text{-}Rel\ (cat\text{-}Rel\ \alpha)) = 5_{\mathbb{N}}$
      unfolding $M\alpha\text{-}Rel\text{-}def$ **by** ($simp\ add$: $nat\text{-}omega\text{-}simps$)

    show $M\alpha\text{-}Rel\ (cat\text{-}Rel\ \alpha)(\lfloor NTMap \rfloor)(\lfloor ABC \rfloor) :$

```
  cf-blcomp (cf-prod-2-Rel (cat-Rel α))(⌊ObjMap⌋)(⌊ABC⌋) ↦_iso o cat-Rel α
  cf-brcomp (cf-prod-2-Rel (cat-Rel α))(⌊ObjMap⌋)(⌊ABC⌋)
  if ABC ∈_∘ (cat-Rel α ^_C3)(⌊Obj⌋) for ABC
proof–
  from that category-cat-Rel obtain A B C
    where ABC-def: ABC = [A, B, C]_∘
      and A: A ∈_∘ cat-Rel α(⌊Obj⌋)
      and B: B ∈_∘ cat-Rel α(⌊Obj⌋)
      and C: C ∈_∘ cat-Rel α(⌊Obj⌋)
    by (elim cat-prod-3-ObjE[rotated 3])
  from that A B C show ?thesis
    unfolding ABC-def
    by
      (
        cs-concl cs-shallow
          cs-intro:
            cat-cs-intros cat-Rel-par-set-cs-intros cat-prod-cs-intros
          cs-simp: cat-cs-simps cat-Rel-cs-simps
      )
qed
then show Mα-Rel (cat-Rel α)(⌊NTMap⌋)(⌊ABC⌋) :
  cf-blcomp (cf-prod-2-Rel (cat-Rel α))(⌊ObjMap⌋)(⌊ABC⌋) ↦_cat-Rel α
  cf-brcomp (cf-prod-2-Rel (cat-Rel α))(⌊ObjMap⌋)(⌊ABC⌋)
  if ABC ∈_∘ (cat-Rel α ^_C3)(⌊Obj⌋) for ABC
  using that by (simp add: cat-Rel-is-iso-arrD(1))
show
  Mα-Rel (cat-Rel α)(⌊NTMap⌋)(⌊ABC′⌋) ∘_A cat-Rel α
    cf-blcomp (cf-prod-2-Rel (cat-Rel α))(⌊ArrMap⌋)(⌊HGF⌋) =
      cf-brcomp (cf-prod-2-Rel (cat-Rel α))(⌊ArrMap⌋)(⌊HGF⌋) ∘_A cat-Rel α
        Mα-Rel (cat-Rel α)(⌊NTMap⌋)(⌊ABC⌋)
  if HGF : ABC ↦_cat-Rel α ^_C3 ABC′ for ABC ABC′ HGF
proof–

  from that obtain H G F A B C A′ B′ C′
    where HGF-def: HGF = [H, G, F]_∘
      and ABC-def: ABC = [A, B, C]_∘
      and ABC′-def: ABC′ = [A′, B′, C′]_∘
      and H-is-arr: H : A ↦_cat-Rel α A′
      and G-is-arr: G : B ↦_cat-Rel α B′
      and F-is-arr: F : C ↦_cat-Rel α C′
    by
      (
        elim cat-prod-3-is-arrE[
          OF category-cat-Rel category-cat-Rel category-cat-Rel
          ]
      )

  note H = cat-Rel-is-arrD[OF H-is-arr]
  note G = cat-Rel-is-arrD[OF G-is-arr]
  note F = cat-Rel-is-arrD[OF F-is-arr]

  interpret H: arr-Rel α H
    rewrites H(⌊ArrDom⌋) = A and H(⌊ArrCod⌋) = A′
    by (intro H)+
  interpret G: arr-Rel α G
    rewrites G(⌊ArrDom⌋) = B and G(⌊ArrCod⌋) = B′
    by (intro G)+
  interpret F: arr-Rel α F
```

```
  rewrites F(|ArrDom|) = C and F(|ArrCod|) = C'
  by (intro F)+

let ?ABC' = ‹Mα-Rel-arrow-lr A' B' C'›
  and ?ABC = ‹Mα-Rel-arrow-lr A B C›
  and ?HG-F =
    ‹
      prod-2-Rel-ArrVal
        (prod-2-Rel-ArrVal (H(|ArrVal|)) (G(|ArrVal|)))
        (F(|ArrVal|))
    ›
  and ?H-GF =
    ‹
      prod-2-Rel-ArrVal
        (H(|ArrVal|))
        (prod-2-Rel-ArrVal (G(|ArrVal|)) (F(|ArrVal|)))
    ›

have [cat-cs-simps]:
  ?ABC' ∘A cat-Rel α (H A×Rel G) A×Rel F =
    H A×Rel (G A×Rel F) ∘A cat-Rel α ?ABC
proof–

  from H-is-arr G-is-arr F-is-arr have lhs:
    ?ABC' ∘A cat-Rel α (H A×Rel G) A×Rel F :
      (A ×∘ B) ×∘ C ↦cat-Rel α A' ×∘ (B' ×∘ C')
    by
      (
        cs-concl cs-shallow
          cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros
      )
  from H-is-arr G-is-arr F-is-arr have rhs:
    H A×Rel (G A×Rel F) ∘A cat-Rel α ?ABC :
      (A ×∘ B) ×∘ C ↦cat-Rel α A' ×∘ (B' ×∘ C')
    by (cs-concl cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros)

  show ?thesis
  proof(rule arr-Rel-eqI)

    from lhs show arr-Rel-lhs:
      arr-Rel α (?ABC' ∘A cat-Rel α (H A×Rel G) A×Rel F)
      by (auto dest: cat-Rel-is-arrD)
    from rhs show arr-Rel-rhs:
      arr-Rel α (H A×Rel (G A×Rel F) ∘A cat-Rel α ?ABC)
      by (auto dest: cat-Rel-is-arrD)

    have [cat-cs-simps]: ?ABC'(|ArrVal|) ∘∘ ?HG-F = ?H-GF ∘∘ ?ABC(|ArrVal|)
    proof(intro vsubset-antisym vsubsetI)
      fix abc-abc'' assume prems: abc-abc'' ∈∘ ?ABC'(|ArrVal|) ∘∘ ?HG-F
      then obtain abc abc' abc''
        where abc-abc''-def: abc-abc'' = ⟨abc, abc''⟩
          and abc-abc': ⟨abc, abc'⟩ ∈∘ ?HG-F
          and abc'-abc'': ⟨abc', abc''⟩ ∈∘ ?ABC'(|ArrVal|)
        by (elim vcompE)
      from abc-abc' obtain ab c ab' c'
        where abc-abc'-def: ⟨abc, abc'⟩ = ⟨⟨ab, c⟩, ⟨ab', c'⟩⟩
          and ab-ab':
            ⟨ab, ab'⟩ ∈∘ prod-2-Rel-ArrVal (H(|ArrVal|)) (G(|ArrVal|))
```

```
    and cc': ⟨c, c'⟩ ∈∘ F(|ArrVal|)
   by (meson prod-2-Rel-ArrValE)
  then have abc-def: abc = ⟨ab, c⟩ and abc'-def: abc' = ⟨ab', c'⟩
   by auto
  from ab-ab' obtain a b a' b'
   where ab-ab'-def: ⟨ab, ab'⟩ = ⟨⟨a, b⟩, ⟨a', b'⟩⟩
    and aa': ⟨a, a'⟩ ∈∘ H(|ArrVal|)
    and bb': ⟨b, b'⟩ ∈∘ G(|ArrVal|)
   by auto
  then have ab-def: ab = ⟨a, b⟩ and ab'-def: ab' = ⟨a', b'⟩
   by auto
  from cc' F.arr-Rel-ArrVal-vdomain F.arr-Rel-ArrVal-vrange
  have c: c ∈∘ C and c': c' ∈∘ C'
   by auto
  from bb' G.arr-Rel-ArrVal-vdomain G.arr-Rel-ArrVal-vrange
  have b: b ∈∘ B and b': b' ∈∘ B'
   by auto
  from aa' H.arr-Rel-ArrVal-vdomain H.arr-Rel-ArrVal-vrange
  have a: a ∈∘ A and a': a' ∈∘ A'
   by auto
  from abc'-abc'' have abc'' = ?ABC'(|ArrVal|)(|abc'|)
   by (simp add: vsv.vsv-appI[OF Mα-Rel-arrow-lr-ArrVal-vsv])
  also from a' b' c' have ... = ⟨a', ⟨b', c'⟩⟩
   unfolding abc'-def ab'-def
   by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: V-cs-intros)
  finally have abc''-def: abc'' = ⟨a', ⟨b', c'⟩⟩ by auto
  from aa' bb' cc' a a' b b' c c' show
   abc-abc'' ∈∘ ?H-GF ∘∘ ?ABC(|ArrVal|)
   unfolding abc-abc''-def abc-def abc'-def abc''-def ab'-def ab-def
   by (intro vcompI prod-2-Rel-ArrValI)
    (
      cs-concl cs-shallow
       cs-simp: cat-cs-simps
       cs-intro:
        vsv.vsv-ex1-app2[THEN iffD1]
        V-cs-intros
        cat-cs-intros
        cat-Rel-cs-intros
    )+
next
  fix abc-abc'' assume prems: abc-abc'' ∈∘ ?H-GF ∘∘ ?ABC(|ArrVal|)
  then obtain abc abc' abc''
   where abc-abc''-def: abc-abc'' = ⟨abc, abc''⟩
    and abc-abc': ⟨abc, abc'⟩ ∈∘ ?ABC(|ArrVal|)
    and abc'-abc'': ⟨abc', abc''⟩ ∈∘ ?H-GF
   by (elim vcompE)
  from abc'-abc'' obtain a' bc' a'' bc''
   where abc'-abc''-def: ⟨abc', abc''⟩ = ⟨⟨a', bc'⟩, ⟨a'', bc''⟩⟩
    and aa'': ⟨a', a''⟩ ∈∘ H(|ArrVal|)
    and bc'-bc'':
      ⟨bc', bc''⟩ ∈∘ prod-2-Rel-ArrVal (G(|ArrVal|)) (F(|ArrVal|))
   by (meson prod-2-Rel-ArrValE)
  then have abc'-def: abc' = ⟨a', bc'⟩
   and abc''-def: abc'' = ⟨a'', bc''⟩
   by auto
  from bc'-bc'' obtain b' c' b'' c''
   where bc'-bc''-def: ⟨bc', bc''⟩ = ⟨⟨b', c'⟩, ⟨b'', c''⟩⟩
    and bb'': ⟨b', b''⟩ ∈∘ G(|ArrVal|)
```

```
    and cc'': ⟨c', c''⟩ ∈∘ F(|ArrVal|)
   by auto
 then have bc'-def: bc' = ⟨b', c'⟩
   and bc''-def: bc'' = ⟨b'', c''⟩
   by auto
 from cc'' F.arr-Rel-ArrVal-vdomain F.arr-Rel-ArrVal-vrange
 have c': c' ∈∘ C and c'': c'' ∈∘ C'
   by auto
 from bb'' G.arr-Rel-ArrVal-vdomain G.arr-Rel-ArrVal-vrange
 have b': b' ∈∘ B and b'': b'' ∈∘ B'
   by auto
 from aa'' H.arr-Rel-ArrVal-vdomain H.arr-Rel-ArrVal-vrange
 have a': a' ∈∘ A and a'': a'' ∈∘ A'
   by auto
 from abc-abc' have abc ∈∘ 𝒟∘ (?ABC(|ArrVal|)) by auto
 then have abc ∈∘ (A ×∘ B) ×∘ C by (simp add: cat-cs-simps)
 then obtain a b c
   where abc-def: abc = ⟨⟨a, b⟩, c⟩
     and a: a ∈∘ A
     and b: b ∈∘ B
     and c: c ∈∘ C
   by auto
 from abc-abc' have abc' = ?ABC(|ArrVal|)(|abc|)
   by (simp add: vsv.vsv-appI[OF Mα-Rel-arrow-lr-ArrVal-vsv])
 also from a b c have ... = ⟨a, ⟨b, c⟩⟩
   unfolding abc-def bc'-def
   by (cs-concl cs-shallow cs-simp: cat-cs-simps cs-intro: V-cs-intros)
 finally have abc'-def': abc' = ⟨a, ⟨b, c⟩⟩ by auto
 with abc'-def[unfolded bc'-def] have [cat-cs-simps]:
   a = a' b = b' c = c'
   by auto
 from a'' b'' c'' have ⟨⟨a'', b''⟩, c''⟩ ∈∘ (A' ×∘ B') ×∘ C'
   by (cs-concl cs-shallow cs-intro: V-cs-intros)
 with aa'' bb'' cc'' a a' b b' c c' show
   abc-abc'' ∈∘ ?ABC'(|ArrVal|) ∘∘ ?HG-F
   unfolding abc-abc''-def abc-def abc'-def abc''-def bc''-def
   by (intro vcompI prod-2-Rel-ArrValI)
    (
      cs-concl cs-shallow
       cs-simp: cat-cs-simps
       cs-intro:
         vsv.vsv-ex1-app2[THEN iffD1]
         V-cs-intros cat-cs-intros cat-Rel-cs-intros
    )+
qed

from that H-is-arr G-is-arr F-is-arr show
 (?ABC' ∘A cat-Rel α (H A×Rel G) A×Rel F)(|ArrVal|) =
   (H A×Rel (G A×Rel F) ∘A cat-Rel α ?ABC)(|ArrVal|)
 by
   (
     cs-concl
       cs-simp:
         prod-2-Rel-components comp-Rel-components
         cat-Rel-cs-simps cat-cs-simps
       cs-intro:
         cat-Rel-par-set-cs-intros cat-cs-intros cat-prod-cs-intros
   )
```

qed (*use lhs rhs* **in** ‹*cs-concl cs-simp*: *cat-cs-simps*›)+

qed

from *that H-is-arr G-is-arr F-is-arr* **show** *?thesis*
unfolding *HGF-def ABC-def ABC'-def*
by
(
cs-concl
cs-intro:
cat-Rel-par-set-cs-intros cat-cs-intros cat-prod-cs-intros
cs-simp: *cat-Rel-cs-simps cat-cs-simps*
)

qed

qed (*cs-concl* **cs-shallow cs-simp**: *cat-cs-simps* **cs-intro**: *cat-cs-intros*)+

qed

lemma (**in** $\mathcal{Z}$) *Mα-Rel-is-iso-ntcf'*[*cat-cs-intros*]:
assumes $\mathfrak{F}'$ = *cf-blcomp* (*cf-prod-2-Rel* (*cat-Rel* α))
and $\mathfrak{G}'$ = *cf-brcomp* (*cf-prod-2-Rel* (*cat-Rel* α))
and $\mathfrak{A}'$ = *cat-Rel* α^{+}_{C3}
and $\mathfrak{B}'$ = *cat-Rel* α
and α' = α
shows *Mα-Rel* (*cat-Rel* α) : $\mathfrak{F}' \mapsto_{CF.iso} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{C_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule Mα-Rel-is-iso-ntcf*)

lemmas [*cat-cs-intros*] = $\mathcal{Z}$.*Mα-Rel-is-iso-ntcf'*

34.7 *Ml* and *Mr* for *Rel*

34.7.1 Definition and elementary properties

definition *Ml-Rel* :: $V \Rightarrow V \Rightarrow V$
where *Ml-Rel* $\mathfrak{C}$ a =
[
($\lambda B \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$. *vsnd-arrow* (*set* {$a$}) B),
cf-prod-2-Rel $\mathfrak{C}_{\mathfrak{C},\mathfrak{C}}$(*set* {$a$},−)$_{CF}$,
cf-id $\mathfrak{C}$,
$\mathfrak{C}$,
$\mathfrak{C}$
]$_{\circ}$

definition *Mr-Rel* :: $V \Rightarrow V \Rightarrow V$
where *Mr-Rel* $\mathfrak{C}$ b =
[
($\lambda A \in_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$. *vfst-arrow* A (*set* {b})),
cf-prod-2-Rel $\mathfrak{C}_{\mathfrak{C},\mathfrak{C}}$(−,*set* {$b$})$_{CF}$,
cf-id $\mathfrak{C}$,
$\mathfrak{C}$,
$\mathfrak{C}$
]$_{\circ}$

Components.

lemma *Ml-Rel-components*:

shows *Ml-Rel* $\mathfrak{C}$ $a(\!|NTMap|\!) = (\lambda B \in_{\circ} \mathfrak{C}(\!|Obj|\!).\ vsnd\text{-}arrow\ (set\ \{a\})\ B)$
and [*cat-cs-simps*]: *Ml-Rel* $\mathfrak{C}$ $a(\!|NTDom|\!) = cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{C}_{\mathfrak{C},\mathfrak{C}}(set\ \{a\},-)_{CF}$
and [*cat-cs-simps*]: *Ml-Rel* $\mathfrak{C}$ $a(\!|NTCod|\!) = cf\text{-}id\ \mathfrak{C}$
and [*cat-cs-simps*]: *Ml-Rel* $\mathfrak{C}$ $a(\!|NTDGDom|\!) = \mathfrak{C}$
and [*cat-cs-simps*]: *Ml-Rel* $\mathfrak{C}$ $a(\!|NTDGCod|\!) = \mathfrak{C}$
unfolding *Ml-Rel-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

lemma *Mr-Rel-components*:
shows *Mr-Rel* $\mathfrak{C}$ $b(\!|NTMap|\!) = (\lambda A \in_{\circ} \mathfrak{C}(\!|Obj|\!).\ vfst\text{-}arrow\ A\ (set\ \{b\}))$
and [*cat-cs-simps*]: *Mr-Rel* $\mathfrak{C}$ $b(\!|NTDom|\!) = cf\text{-}prod\text{-}2\text{-}Rel\ \mathfrak{C}_{\mathfrak{C},\mathfrak{C}}(-,set\ \{b\})_{CF}$
and [*cat-cs-simps*]: *Mr-Rel* $\mathfrak{C}$ $b(\!|NTCod|\!) = cf\text{-}id\ \mathfrak{C}$
and [*cat-cs-simps*]: *Mr-Rel* $\mathfrak{C}$ $b(\!|NTDGDom|\!) = \mathfrak{C}$
and [*cat-cs-simps*]: *Mr-Rel* $\mathfrak{C}$ $b(\!|NTDGCod|\!) = \mathfrak{C}$
unfolding *Mr-Rel-def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

34.7.2 Natural transformation map

mk-VLambda *Ml-Rel-components*(*1*)
|*vsv Ml-Rel-components-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain Ml-Rel-components-NTMap-vdomain*[*cat-cs-simps*]|
|*app Ml-Rel-components-NTMap-app*[*cat-cs-simps*]|

mk-VLambda *Mr-Rel-components*(*1*)
|*vsv Mr-Rel-components-NTMap-vsv*[*cat-cs-intros*]|
|*vdomain Mr-Rel-components-NTMap-vdomain*[*cat-cs-simps*]|
|*app Mr-Rel-components-NTMap-app*[*cat-cs-simps*]|

34.7.3 *Ml* and *Mr* for *Rel* are natural isomorphisms

lemma (**in** $\mathcal{Z}$) *Ml-Rel-is-iso-ntcf*:
assumes $a \in_{\circ}$ *cat-Rel* $\alpha(\!|Obj|\!)$
shows *Ml-Rel* (*cat-Rel* α) *a*:
cf-prod-2-Rel (*cat-Rel* α)$_{cat\text{-}Rel\ \alpha,cat\text{-}Rel\ \alpha}(set\ \{a\},-)_{CF} \mapsto_{CF.iso}$
cf-id (*cat-Rel* α) :
cat-Rel $\alpha \mapsto\mapsto_{C\alpha}$ *cat-Rel* α
proof–

let *?cf-prod* = ‹*cf-prod-2-Rel* (*cat-Rel* α)$_{cat\text{-}Rel\ \alpha,cat\text{-}Rel\ \alpha}$ $(set\ \{a\},-)_{CF}$›
note [*cat-cs-simps*] = *set-empty*

interpret *cf-prod*: *is-functor*
α ‹*cat-Rel* $\alpha \times_C$ *cat-Rel* α› ‹*cat-Rel* α› ‹*cf-prod-2-Rel* (*cat-Rel* α)›
by (*cs-concl* **cs-shallow cs-intro**: *cat-cs-intros cat-Rel-cs-intros*)

show *?thesis*
proof(*intro is-iso-ntcfI is-ntcfI'*)
show *vfsequence* (*Ml-Rel* (*cat-Rel* α) *a*) **unfolding** *Ml-Rel-def* **by** *auto*
show *vcard* (*Ml-Rel* (*cat-Rel* α) *a*) = $5_{\mathbb{N}}$
unfolding *Ml-Rel-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** *?cf-prod* : *cat-Rel* $\alpha \mapsto\mapsto_{C\alpha}$ *cat-Rel* α
by
(
cs-concl
cs-simp: *cat-Rel-components*(*1*) *cat-cs-simps*
cs-intro: *cat-cs-intros V-cs-intros*
)
show *Ml-Rel* (*cat-Rel* α) $a(\!|NTMap|\!)(\!|B|\!)$:
?cf-prod$(\!|ObjMap|\!)(\!|B|\!) \mapsto_{iso} cat\text{-}Rel\ \alpha$ *cf-id* (*cat-Rel* α)$(\!|ObjMap|\!)(\!|B|\!)$

```
  if B ∈∘ cat-Rel α(|Obj|) for B
  using assms that
  by
    (
      cs-concl
        cs-simp: cat-Rel-components(1) V-cs-simps cat-cs-simps
        cs-intro:
          cat-Rel-par-set-cs-intros
          cat-cs-intros
          V-cs-intros
          cat-prod-cs-intros
    )
with cat-Rel-is-iso-arrD[OF this] show
  Ml-Rel (cat-Rel α) a(|NTMap|)(|B|) :
    ?cf-prod(|ObjMap|)(|B|) ↦_cat-Rel α cf-id (cat-Rel α)(|ObjMap|)(|B|)
  if B ∈∘ cat-Rel α(|Obj|) for B
  using that by simp
show
  Ml-Rel (cat-Rel α) a(|NTMap|)(|B|) ∘_A cat-Rel α ?cf-prod(|ArrMap|)(|F|) =
    cf-id (cat-Rel α)(|ArrMap|)(|F|) ∘_A cat-Rel α Ml-Rel (cat-Rel α) a(|NTMap|)(|A|)
  if F : A ↦_cat-Rel α B for A B F
proof–
  note F = cat-Rel-is-arrD[OF that]
  interpret F: arr-Rel α F
    rewrites F(|ArrDom|) = A and F(|ArrCod|) = B
    by (intro F)+
  have [cat-cs-simps]:
    vsnd-arrow (set {a}) B ∘_A cat-Rel α
      (cat-Rel α(|CId|)(|set {a}|)) _A×_Rel F =
        F ∘_A cat-Rel α vsnd-arrow (set {a}) A
    (is ‹?B2 ∘_A cat-Rel α ?aF = F ∘_A cat-Rel α ?A2›)
  proof–
    from assms that have lhs:
      ?B2 ∘_A cat-Rel α ?aF : set {a} ×∘ A ↦_cat-Rel α B
      by
        (
          cs-concl
            cs-simp: cat-Rel-components(1) cat-cs-simps
            cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros V-cs-intros
        )
    from assms that have rhs:
      F ∘_A cat-Rel α ?A2 : set {a} ×∘ A ↦_cat-Rel α B
      by
        (
          cs-concl
            cs-simp: cat-Rel-components(1) cat-cs-simps
            cs-intro: cat-Rel-par-set-cs-intros cat-cs-intros V-cs-intros
        )
    have [cat-cs-simps]:
      ?B2(|ArrVal|) ∘∘ prod-2-Rel-ArrVal (vid-on (set {a})) (F(|ArrVal|)) =
        F(|ArrVal|) ∘∘ ?A2(|ArrVal|)
    proof(intro vsubset-antisym vsubsetI)
      fix xx′-z assume xx′-z ∈∘
        ?B2(|ArrVal|) ∘∘ prod-2-Rel-ArrVal (vid-on (set {a})) (F(|ArrVal|))
      then obtain xx′ yy′ z
        where xx′-z-def: xx′-z = ⟨xx′, z⟩
          and xx′-yy′:
            ⟨xx′, yy′⟩ ∈∘ prod-2-Rel-ArrVal (vid-on (set {a})) (F(|ArrVal|))
```

```
      and yy'-z: ⟨yy', z⟩ ∈∘ ?B2(|ArrVal|)
    by (meson vcompE prod-2-Rel-ArrValE)
  from xx'-yy' obtain x x' y y'
    where ⟨xx', yy'⟩ = ⟨⟨x, x'⟩, ⟨y, y'⟩⟩
      and ⟨x, y⟩ ∈∘ vid-on (set {a})
      and xy': ⟨x', y'⟩ ∈∘ F(|ArrVal|)
    by auto
  then have xx'-def: xx' = ⟨a, x'⟩ and yy'-def: yy' = ⟨a, y'⟩
    by simp-all
  with yy'-z have y': y' ∈∘ B and z-def: z = y'
    unfolding vsnd-arrow-components by auto
  from xy' vsubsetD have x': x' ∈∘ A
    by (auto intro: F.arr-Rel-ArrVal-vdomain)
  show xx'-z ∈∘ F(|ArrVal|) ∘∘ ?A2(|ArrVal|)
    unfolding xx'-z-def z-def xx'-def
    by (intro vcompI, rule xy')
      (auto simp: vsnd-arrow-components x' VLambda-iff2)
next
  fix ay-z assume ay-z ∈∘ F(|ArrVal|) ∘∘ ?A2(|ArrVal|)
  then obtain ay y z
    where xx'-z-def: ay-z = ⟨ay, z⟩
      and ay-y: ⟨ay, y⟩ ∈∘ ?A2(|ArrVal|)
      and yz[cat-cs-intros]: ⟨y, z⟩ ∈∘ F(|ArrVal|)
    by auto
  then have ay-z-def: ay-z = ⟨⟨a, y⟩, z⟩
    and y: y ∈∘ A
    and ay-def: ay = ⟨a, y⟩
    unfolding vsnd-arrow-components by auto
  from yz vsubsetD have z: z ∈∘ B
    by (auto intro: F.arr-Rel-ArrVal-vrange)
  have [cat-cs-intros]: ⟨a, a⟩ ∈∘ vid-on (set {a}) by auto
  show ay-z ∈∘
    ?B2(|ArrVal|) ∘∘ prod-2-Rel-ArrVal (vid-on (set {a})) (F(|ArrVal|))
    unfolding ay-z-def
    by
      (
        intro vcompI prod-2-Rel-ArrValI,
        rule vsv.vsv-ex1-app1[THEN iffD1],
        unfold cat-cs-simps,
        insert z
      )
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
      )
qed
show ?thesis
proof(rule arr-Rel-eqI)
  from lhs show arr-Rel α (?B2 ∘A cat-Rel α ?aF)
    by (auto dest: cat-Rel-is-arrD)
  from rhs show arr-Rel α (F ∘A cat-Rel α ?A2)
    by (auto dest: cat-Rel-is-arrD)
  note cat-Rel-CId-app[cat-Rel-cs-simps del]
  note Z.cat-Rel-CId-app[cat-Rel-cs-simps del]
  from that assms show
    (?B2 ∘A cat-Rel α ?aF)(|ArrVal|) = (F ∘A cat-Rel α ?A2)(|ArrVal|)
    by
      (
```

```
            cs-concl
              cs-simp: cat-cs-simps cat-Rel-cs-simps
              cs-intro: cat-cs-intros cat-Rel-par-set-cs-intros V-cs-intros
              cs-simp:
                id-Rel-components
                cat-Rel-CId-app
                comp-Rel-components(1)
                prod-2-Rel-components
                cat-Rel-components(1)
          )
      qed (use lhs rhs in ‹cs-concl cs-simp: cat-cs-simps›)+
    qed
    from that assms show ?thesis
      by
        (
          cs-concl
            cs-simp: cat-cs-simps
            cs-intro: cat-cs-intros V-cs-intros cat-prod-cs-intros
            cs-simp: cat-Rel-components(1) V-cs-simps
        )
  qed
qed (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

qed

lemma (in 𝒵) Ml-Rel-is-iso-ntcf'[cat-cs-intros]:
  assumes a ∈∘ cat-Rel α(|Obj|)
    and 𝔉' = cf-prod-2-Rel (cat-Rel α)_{cat-Rel α,cat-Rel α}(set {a},−)_{CF}
    and 𝔊' = cf-id (cat-Rel α)
    and 𝔄' = cat-Rel α
    and 𝔅' = cat-Rel α
    and α' = α
  shows Ml-Rel (cat-Rel α) a : 𝔉' ↦_{CF.iso} 𝔊' : 𝔄' ↦↦_{C α'} 𝔅'
  using assms(1) unfolding assms(2–6) by (rule Ml-Rel-is-iso-ntcf)

lemmas [cat-cs-intros] = 𝒵.Ml-Rel-is-iso-ntcf'

lemma (in 𝒵) Mr-Rel-is-iso-ntcf:
  assumes b ∈∘ cat-Rel α(|Obj|)
  shows Mr-Rel (cat-Rel α) b :
    cf-prod-2-Rel (cat-Rel α)_{cat-Rel α,cat-Rel α}(−,set {b})_{CF} ↦_{CF.iso}
    cf-id (cat-Rel α) :
    cat-Rel α ↦↦_{Cα} cat-Rel α
proof–

  let ?cf-prod = ‹cf-prod-2-Rel (cat-Rel α)_{cat-Rel α,cat-Rel α} (−,set {b})_{CF}›
  note [cat-cs-simps] = set-empty

  interpret cf-prod: is-functor
    α ‹cat-Rel α ×_C cat-Rel α› ‹cat-Rel α› ‹cf-prod-2-Rel (cat-Rel α)›
    by (cs-concl cs-shallow cs-intro: cat-cs-intros cat-Rel-cs-intros)

  show ?thesis
  proof(intro is-iso-ntcfI is-ntcfI')
    show vfsequence (Mr-Rel (cat-Rel α) b) unfolding Mr-Rel-def by auto
    show vcard (Mr-Rel (cat-Rel α) b) = 5_ℕ
      unfolding Mr-Rel-def by (simp add: nat-omega-simps)
    from assms show ?cf-prod : cat-Rel α ↦↦_{Cα} cat-Rel α
```

```
  by
    (
      cs-concl
        cs-simp: cat-Rel-components(1) cat-cs-simps
        cs-intro: cat-cs-intros V-cs-intros
    )
show Mr-Rel (cat-Rel α) b(⁅NTMap⁆)(⁅B⁆) :
  ?cf-prod(⁅ObjMap⁆)(⁅B⁆) ↦_iso o cat-Rel α cf-id (cat-Rel α)(⁅ObjMap⁆)(⁅B⁆)
  if B ∈_∘ cat-Rel α(⁅Obj⁆) for B
  using assms that
  by
    (
      cs-concl
        cs-simp: cat-Rel-components(1) V-cs-simps cat-cs-simps
        cs-intro:
          cat-cs-intros
          cat-Rel-par-set-cs-intros
          V-cs-intros
          cat-prod-cs-intros
    )
with cat-Rel-is-iso-arrD[OF this] show
  Mr-Rel (cat-Rel α) b(⁅NTMap⁆)(⁅B⁆) :
    ?cf-prod(⁅ObjMap⁆)(⁅B⁆) ↦_cat-Rel α cf-id (cat-Rel α)(⁅ObjMap⁆)(⁅B⁆)
  if B ∈_∘ cat-Rel α(⁅Obj⁆) for B
  using that by simp
show
  Mr-Rel (cat-Rel α) b(⁅NTMap⁆)(⁅B⁆) ∘_A cat-Rel α ?cf-prod(⁅ArrMap⁆)(⁅F⁆) =
    cf-id (cat-Rel α)(⁅ArrMap⁆)(⁅F⁆) ∘_A cat-Rel α Mr-Rel (cat-Rel α) b(⁅NTMap⁆)(⁅A⁆)
  if F : A ↦_cat-Rel α B for A B F
proof–
  note F = cat-Rel-is-arrD[OF that]
  interpret F: arr-Rel α F
    rewrites F(⁅ArrDom⁆) = A and F(⁅ArrCod⁆) = B
    by (intro F)+
  have [cat-cs-simps]:
    vfst-arrow B (set {b}) ∘_A cat-Rel α
      F _A×_Rel (cat-Rel α(⁅CId⁆)(⁅set {b}⁆)) =
        F ∘_A cat-Rel α vfst-arrow A (set {b})
    (is ‹?B1 ∘_A cat-Rel α ?bF = F ∘_A cat-Rel α ?A1›)
  proof–
    from assms that have lhs:
      ?B1 ∘_A cat-Rel α ?bF : A ×_∘ set {b} ↦_cat-Rel α B
      by
        (
          cs-concl
            cs-simp: cat-Rel-components(1) cat-cs-simps
            cs-intro: cat-cs-intros cat-Rel-par-set-cs-intros V-cs-intros
        )
    from assms that have rhs:
      F ∘_A cat-Rel α ?A1 : A ×_∘ set {b} ↦_cat-Rel α B
      by
        (
          cs-concl
            cs-simp: cat-Rel-components(1) cat-cs-simps
            cs-intro: cat-cs-intros cat-Rel-par-set-cs-intros V-cs-intros
        )
    have [cat-cs-simps]:
      ?B1(⁅ArrVal⁆) ∘_∘ prod-2-Rel-ArrVal (F(⁅ArrVal⁆)) (vid-on (set {b})) =
```

```
    F(|ArrVal|) ∘ₒ ?A1(|ArrVal|)
proof(intro vsubset-antisym vsubsetI)
  fix xx′-z assume xx′-z ∈ₒ
    ?B1(|ArrVal|) ∘ₒ prod-2-Rel-ArrVal (F(|ArrVal|)) (vid-on (set {b}))
  then obtain xx′ yy′ z
    where xx′-z-def: xx′-z = ⟨xx′, z⟩
      and xx′-yy′:
        ⟨xx′, yy′⟩ ∈ₒ prod-2-Rel-ArrVal (F(|ArrVal|)) (vid-on (set {b}))
      and yy′-z: ⟨yy′, z⟩ ∈ₒ ?B1(|ArrVal|)
    by (meson vcompE prod-2-Rel-ArrValE)
  from xx′-yy′ obtain x x′ y y′
    where ⟨xx′, yy′⟩ = ⟨⟨x, x′⟩, ⟨y, y′⟩⟩
      and ⟨x′, y′⟩ ∈ₒ vid-on (set {b})
      and xy: ⟨x, y⟩ ∈ₒ F(|ArrVal|)
    by auto
  then have xx′-def: xx′ = ⟨x, b⟩ and yy′-def: yy′ = ⟨y, b⟩
    by simp-all
  with yy′-z have y′: y ∈ₒ B and z-def: z = y
    unfolding vfst-arrow-components by auto
  from xy vsubsetD have x: x ∈ₒ A
    by (auto intro: F.arr-Rel-ArrVal-vdomain)
  show xx′-z ∈ₒ F(|ArrVal|) ∘ₒ ?A1(|ArrVal|)
    unfolding xx′-z-def z-def xx′-def
    by (intro vcompI, rule xy)
      (auto simp: vfst-arrow-components x VLambda-iff2)
next
  fix xy-z assume xy-z ∈ₒ F(|ArrVal|) ∘ₒ ?A1(|ArrVal|)
  then obtain xy y z
    where xx′-z-def: xy-z = ⟨xy, z⟩
      and xy-y: ⟨xy, y⟩ ∈ₒ ?A1(|ArrVal|)
      and yz[cat-cs-intros]: ⟨y, z⟩ ∈ₒ F(|ArrVal|)
    by auto
  then have xy-z-def: xy-z = ⟨⟨y, b⟩, z⟩
    and y: y ∈ₒ A
    and xy-def: xy = ⟨y, b⟩
    unfolding vfst-arrow-components by auto
  from yz vsubsetD have z: z ∈ₒ B
    by (auto intro: F.arr-Rel-ArrVal-vrange)
  have [cat-cs-intros]: ⟨b, b⟩ ∈ₒ vid-on (set {b}) by auto
  show xy-z ∈ₒ
    ?B1(|ArrVal|) ∘ₒ prod-2-Rel-ArrVal (F(|ArrVal|)) (vid-on (set {b}))
    unfolding xy-z-def
    by
      (
        intro vcompI prod-2-Rel-ArrValI,
        rule vsv.vsv-ex1-app1[THEN iffD1],
        unfold cat-cs-simps,
        insert z
      )
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cs-intro: V-cs-intros cat-cs-intros
      )
qed
show ?thesis
proof(rule arr-Rel-eqI)
  from lhs show arr-Rel α (?B1 ∘_{A cat-Rel α} ?bF)
    by (auto dest: cat-Rel-is-arrD)
```

```
      from rhs show arr-Rel α (F ∘_A cat-Rel α ?A1)
        by (auto dest: cat-Rel-is-arrD)
      note cat-Rel-CId-app[cat-Rel-cs-simps del]
      note 𝒵.cat-Rel-CId-app[cat-Rel-cs-simps del]
      from that assms show
        (?B1 ∘_A cat-Rel α ?bF)(|ArrVal|) = (F ∘_A cat-Rel α ?A1)(|ArrVal|)
        by
          (
            cs-concl
              cs-simp: cat-cs-simps cat-Rel-cs-simps
              cs-intro: cat-cs-intros cat-Rel-par-set-cs-intros V-cs-intros
              cs-simp:
                id-Rel-components
                cat-Rel-CId-app
                comp-Rel-components(1)
                prod-2-Rel-components
                cat-Rel-components(1)
          )
    qed (use lhs rhs in ‹cs-concl cs-simp: cat-cs-simps›)+
  qed
  from that assms show ?thesis
    by
      (
        cs-concl
          cs-simp: cat-cs-simps
          cs-intro: cat-cs-intros V-cs-intros cat-prod-cs-intros
          cs-simp: cat-Rel-components(1) V-cs-simps
      )
 qed
qed (cs-concl cs-simp: cat-cs-simps cs-intro: cat-cs-intros)+

qed

lemma (in 𝒵) Mr-Rel-is-iso-ntcf'[cat-cs-intros]:
  assumes b ∈∘ cat-Rel α(|Obj|)
    and 𝔉' = cf-prod-2-Rel (cat-Rel α)_cat-Rel α,cat-Rel α(−,set {b})_CF
    and 𝔊' = cf-id (cat-Rel α)
    and 𝔄' = cat-Rel α
    and 𝔅' = cat-Rel α
    and α' = α
  shows Mr-Rel (cat-Rel α) b : 𝔉' ↦_CF.iso 𝔊' : 𝔄' ↦↦_C_α' 𝔅'
  using assms(1) unfolding assms(2–6) by (rule Mr-Rel-is-iso-ntcf)

lemmas [cat-cs-intros] = 𝒵.Mr-Rel-is-iso-ntcf'
```

34.8 *Rel* as a monoidal category

34.8.1 Definition and elementary properties

For further information see [2][21].

```
definition mcat-Rel :: V ⇒ V ⇒ V
  where mcat-Rel α a =
    [
      cat-Rel α,
      cf-prod-2-Rel (cat-Rel α),
      set {a},
```

[21] https://en.wikipedia.org/wiki/Category_of_relations

```
    Mα-Rel (cat-Rel α),
    Ml-Rel (cat-Rel α) a,
    Mr-Rel (cat-Rel α) a
  ]∘
```

Components.

```
lemma mcat-Rel-components:
  shows mcat-Rel α a(|Mcat|) = cat-Rel α
    and mcat-Rel α a(|Mcf|) = cf-prod-2-Rel (cat-Rel α)
    and mcat-Rel α a(|Me|) = set {a}
    and mcat-Rel α a(|Mα|) = Mα-Rel (cat-Rel α)
    and mcat-Rel α a(|Ml|) = Ml-Rel (cat-Rel α) a
    and mcat-Rel α a(|Mr|) = Mr-Rel (cat-Rel α) a
  unfolding mcat-Rel-def mcat-field-simps by (simp-all add: nat-omega-simps)
```

34.8.2 *Rel* is a monoidal category

```
lemma (in Z) monoidal-category-mcat-Rel:
  assumes a ∈∘ cat-Rel α(|Obj|)
  shows monoidal-category α (mcat-Rel α a)
proof–

  interpret Set-Par: wide-replete-subcategory α ‹cat-Set α› ‹cat-Par α›
    by (rule wide-replete-subcategory-cat-Set-cat-Par)
  interpret Par-Rel: wide-replete-subcategory α ‹cat-Par α› ‹cat-Rel α›
    by (rule wide-replete-subcategory-cat-Par-cat-Rel)
  interpret Set-Rel: wide-replete-subcategory α ‹cat-Set α› ‹cat-Rel α›
    by
      (
        rule wr-subcat-trans
          [
            OF
              Set-Par.wide-replete-subcategory-axioms
              Par-Rel.wide-replete-subcategory-axioms
          ]
      )

  show ?thesis
  proof(rule monoidal-categoryI)
    show vfsequence (mcat-Rel α a) unfolding mcat-Rel-def by auto
    show category α (mcat-Rel α a(|Mcat|))
      unfolding mcat-Rel-components
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show mcat-Rel α a(|Mcf|) :
      mcat-Rel α a(|Mcat|) ×_C mcat-Rel α a(|Mcat|) ↦↦_Cα mcat-Rel α a(|Mcat|)
      unfolding mcat-Rel-components
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    show mcat-Rel α a(|Mα|) :
      cf-blcomp (mcat-Rel α a(|Mcf|)) ↦_CF.iso cf-brcomp (mcat-Rel α a(|Mcf|)) :
      mcat-Rel α a(|Mcat|) ^_C3 ↦↦_Cα mcat-Rel α a(|Mcat|)
      unfolding mcat-Rel-components
      by (cs-concl cs-shallow cs-intro: cat-cs-intros)
    from assms show mcat-Rel α a(|Ml|) :
      mcat-Rel α a(|Mcf|)_mcat-Rel α a(|Mcat|),mcat-Rel α a(|Mcat|) (mcat-Rel α a(|Me|),–)_CF
        ↦_CF.iso
      cf-id (mcat-Rel α a(|Mcat|)) :
      mcat-Rel α a(|Mcat|) ↦↦_Cα mcat-Rel α a(|Mcat|)
      unfolding mcat-Rel-components
```

```
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
from assms show mcat-Rel α a(|Mr|) :
  mcat-Rel α a(|Mcf|)_{mcat-Rel α a(|Mcat|),mcat-Rel α a(|Mcat|)} (−,mcat-Rel α a(|Me|))_{CF
    ↦_{CF.iso}
  cf-id (mcat-Rel α a(|Mcat|)) : mcat-Rel α a(|Mcat|) ↦↦_{Cα} mcat-Rel α a(|Mcat|)
  unfolding mcat-Rel-components
  by (cs-concl cs-shallow cs-intro: cat-cs-intros)
show vcard (mcat-Rel α a) = 6_ℕ
  unfolding mcat-Rel-def by (simp add: nat-omega-simps)
from assms show mcat-Rel α a(|Me|) ∈_∘ mcat-Rel α a(|Mcat|)(|Obj|)
  unfolding mcat-Rel-components cat-Rel-components by force
show
  mcat-Rel α a(|Mcat|)(|CId|)(|A|) ⊗_{HM.A} mcat-Rel α a(|Mcf|)
    mcat-Rel α a(|Mα|)(|NTMap|)(|B, C, D|)• ∘_A mcat-Rel α a(|Mcat|)
      mcat-Rel α a(|Mα|)(|NTMap|)(|
        A, B ⊗_{HM.O} mcat-Rel α a(|Mcf|) C, D
        |)• ∘_A mcat-Rel α a(|Mcat|)
          (mcat-Rel α a(|Mα|)(|NTMap|)(|A, B, C|)• ⊗_{HM.A} mcat-Rel α a(|Mcf|)
            mcat-Rel α a(|Mcat|)(|CId|)(|D|)) =
              mcat-Rel α a(|Mα|)(|NTMap|)(|
                A, B, C ⊗_{HM.O} mcat-Rel α a(|Mcf|) D
                |)• ∘_A mcat-Rel α a(|Mcat|)
                  mcat-Rel α a(|Mα|)(|NTMap|)(|A ⊗_{HM.O} mcat-Rel α a(|Mcf|) B, C, D|)•
  if A ∈_∘ mcat-Rel α a(|Mcat|)(|Obj|)
    and B ∈_∘ mcat-Rel α a(|Mcat|)(|Obj|)
    and C ∈_∘ mcat-Rel α a(|Mcat|)(|Obj|)
    and D ∈_∘ mcat-Rel α a(|Mcat|)(|Obj|)
  for A B C D
proof–

  have [cat-cs-simps]:
    (cat-Rel α(|CId|)(|A|)) _A×_{Rel} (Mα-Rel-arrow-lr B C D) ∘_A cat-Rel α
      (
        Mα-Rel-arrow-lr A (B ×_∘ C) D ∘_A cat-Rel α
        (Mα-Rel-arrow-lr A B C) _A×_{Rel} (cat-Rel α(|CId|)(|D|))
      ) =
        Mα-Rel-arrow-lr A B (C ×_∘ D) ∘_A cat-Rel α
          Mα-Rel-arrow-lr (A ×_∘ B) C D
    (
      is
        ‹
          ?A-BCD ∘_A cat-Rel α (?A-BC-D ∘_A cat-Rel α ?ABC-D) =
          ?A-B-CD ∘_A cat-Rel α ?AB-C-D
        ›
    )
  proof–

    have [cat-cs-simps]:
      (cat-Set α(|CId|)(|A|)) _A×_{Rel} (Mα-Rel-arrow-lr B C D) ∘_A cat-Set α
        (
          ?A-BC-D ∘_A cat-Set α
          (Mα-Rel-arrow-lr A B C) _A×_{Rel} (cat-Set α(|CId|)(|D|))
        ) = ?A-B-CD ∘_A cat-Set α ?AB-C-D
      (
        is
          ‹
            ?A-BCD ∘_A cat-Set α (?A-BC-D ∘_A cat-Set α ?ABC-D) =
```

```
      ?A-B-CD ∘_A cat-Set α ?AB-C-D
    ›
  )
proof–
  from that have lhs:
    ?A-BCD ∘_A cat-Set α (?A-BC-D ∘_A cat-Set α ?ABC-D) :
      ((A ×_∘ B) ×_∘ C) ×_∘ D ↦_cat-Set α A ×_∘ B ×_∘ C ×_∘ D
    unfolding mcat-Rel-components cat-Rel-components(1)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Set-components(1)
          cs-intro: cat-rel-par-Set-cs-intros cat-cs-intros V-cs-intros
      )
  then have dom-lhs:
    𝒟_∘ ((?A-BCD ∘_A cat-Set α (?A-BC-D ∘_A cat-Set α ?ABC-D))(⦇ArrVal⦈)) =
      ((A ×_∘ B) ×_∘ C) ×_∘ D
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from that have rhs: ?A-B-CD ∘_A cat-Set α ?AB-C-D :
    ((A ×_∘ B) ×_∘ C) ×_∘ D ↦_cat-Set α A ×_∘ B ×_∘ C ×_∘ D
    unfolding mcat-Rel-components cat-Rel-components(1)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Rel-components(1) cat-Set-components(1)
          cs-intro:
            cat-cs-intros V-cs-intros Mα-Rel-arrow-lr-is-cat-Set-arr'
      )
  then have dom-rhs:
    𝒟_∘ ((?A-B-CD ∘_A cat-Set α ?AB-C-D)(⦇ArrVal⦈)) =
      ((A ×_∘ B) ×_∘ C) ×_∘ D
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
  proof(rule arr-Set-eqI)
    from lhs show arr-Set-lhs:
      arr-Set α (?A-BCD ∘_A cat-Set α (?A-BC-D ∘_A cat-Set α ?ABC-D))
      by (auto dest: cat-Set-is-arrD(1))
    from rhs show arr-Set-rhs:
      arr-Set α (?A-B-CD ∘_A cat-Set α ?AB-C-D)
      by (auto dest: cat-Set-is-arrD(1))
    show
      (?A-BCD ∘_A cat-Set α (?A-BC-D ∘_A cat-Set α ?ABC-D))(⦇ArrVal⦈) =
        (?A-B-CD ∘_A cat-Set α ?AB-C-D)(⦇ArrVal⦈)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix abcd assume prems: abcd ∈_∘ ((A ×_∘ B) ×_∘ C) ×_∘ D
      then obtain a b c d
        where abcd-def: abcd = ⟨⟨⟨a, b⟩, c⟩, d⟩
          and a: a ∈_∘ A
          and b: b ∈_∘ B
          and c: c ∈_∘ C
          and d: d ∈_∘ D
        by clarsimp
      from that prems a b c d show
        (
          ?A-BCD ∘_A cat-Set α
            (?A-BC-D ∘_A cat-Set α ?ABC-D)
        )(⦇ArrVal⦈)(⦇abcd⦈) =
          (?A-B-CD ∘_A cat-Set α ?AB-C-D)(⦇ArrVal⦈)(⦇abcd⦈)
```

```
      unfolding abcd-def mcat-Rel-components(1) cat-Rel-components(1)
      by
        (
          cs-concl cs-shallow
            cs-simp:
              cat-Set-components(1)
              cat-cs-simps
              cat-rel-par-Set-cs-simps
            cs-intro:
              cat-cs-intros cat-rel-par-Set-cs-intros V-cs-intros
        )
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed (use lhs rhs in ‹cs-concl cs-shallow cs-simp: cat-cs-simps›)+
qed

from assms that show ?thesis
  unfolding mcat-Rel-components cat-Rel-components(1)
  by
    (
      cs-concl cs-shallow
        cs-simp:
          cat-cs-simps
          cat-Rel-components(1)
          cat-Set-components(1)
          Set-Rel.subcat-CId[symmetric]
          Set-Rel.subcat-Comp-simp[symmetric]
        cs-intro: cat-cs-intros cat-rel-par-Set-cs-intros V-cs-intros
    )+

qed

from that show ?thesis
  unfolding mcat-Rel-components cat-Rel-components(1)
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-Rel-components(1) cat-cs-simps
        cs-intro:
          cat-cs-intros
          cat-Rel-par-set-cs-intros
          V-cs-intros
          cat-prod-cs-intros
    )

qed

show
  mcat-Rel α a⦇Mcat⦈⦇CId⦈⦇A⦈ ⊗HM.A mcat-Rel α a⦇Mcf⦈
    mcat-Rel α a⦇Ml⦈⦇NTMap⦈⦇B⦈ ∘A mcat-Rel α a⦇Mcat⦈
      mcat-Rel α a⦇Mα⦈⦇NTMap⦈⦇A, mcat-Rel α a⦇Me⦈, B⦈• =
        mcat-Rel α a⦇Mr⦈⦇NTMap⦈⦇A⦈ ⊗HM.A mcat-Rel α a⦇Mcf⦈
          mcat-Rel α a⦇Mcat⦈⦇CId⦈⦇B⦈
  if A ∈∘ mcat-Rel α a⦇Mcat⦈⦇Obj⦈ and B ∈∘ mcat-Rel α a⦇Mcat⦈⦇Obj⦈ for A B
proof–

  note [cat-cs-simps] = set-empty
```

```
have [cat-cs-simps]:
  (cat-Set α(|CId|)(|A|)) A×Rel (vsnd-arrow (set {a}) B) ∘A cat-Set α
    Mα-Rel-arrow-lr A (set {a}) B =
      (vfst-arrow A (set {a})) A×Rel (cat-Set α(|CId|)(|B|))
  (is ‹?A-aB ∘A cat-Set α ?AaB = ?Aa-B›)
proof–
  from assms that have lhs:
    ?A-aB ∘A cat-Set α ?AaB : (A ×∘ set {a}) ×∘ B ↦cat-Set α A ×∘ B
    unfolding mcat-Rel-components cat-Rel-components(1)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-Rel-components(1) cat-Set-components(1)
          cs-intro: cat-cs-intros cat-rel-par-Set-cs-intros V-cs-intros
      )
  then have dom-lhs:
    𝒟∘ ((?A-aB ∘A cat-Set α ?AaB)(|ArrVal|)) = (A ×∘ set {a}) ×∘ B
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  from assms that have rhs:
    ?Aa-B : (A ×∘ set {a}) ×∘ B ↦cat-Set α A ×∘ B
    unfolding mcat-Rel-components cat-Rel-components(1)
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-cs-simps cat-Set-components(1)
          cs-intro: cat-cs-intros cat-rel-par-Set-cs-intros V-cs-intros
      )
  then have dom-rhs: 𝒟∘ (?Aa-B(|ArrVal|)) = (A ×∘ set {a}) ×∘ B
    by (cs-concl cs-shallow cs-simp: cat-cs-simps)
  show ?thesis
  proof(rule arr-Set-eqI)
    from lhs show arr-Set-lhs: arr-Set α (?A-aB ∘A cat-Set α ?AaB)
      by (auto dest: cat-Set-is-arrD(1))
    from rhs show arr-Set-rhs: arr-Set α ?Aa-B
      by (auto dest: cat-Set-is-arrD(1))
    show (?A-aB ∘A cat-Set α ?AaB)(|ArrVal|) = ?Aa-B(|ArrVal|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix xay assume xay ∈∘ (A ×∘ set {a}) ×∘ B
      then obtain x y
        where xay-def: xay = ⟨⟨x, a⟩, y⟩ and x: x ∈∘ A and y: y ∈∘ B
        by auto
      from assms that x y show
        (?A-aB ∘A cat-Set α ?AaB)(|ArrVal|)(|xay|) = ?Aa-B(|ArrVal|)(|xay|)
        unfolding xay-def mcat-Rel-components cat-Rel-components(1)
        by
          (
            cs-concl cs-shallow
              cs-simp:
                cat-Rel-components(1) cat-Set-components(1)
                cat-cs-simps cat-rel-par-Set-cs-simps
              cs-intro:
                cat-cs-intros cat-rel-par-Set-cs-intros V-cs-intros
          )
    qed (use arr-Set-lhs arr-Set-rhs in auto)
  qed (use lhs rhs in ‹cs-concl cs-simp: cat-cs-simps›)+
qed

from assms that show ?thesis
```

unfolding *mcat-Rel-components cat-Rel-components*(*1*)
by
(
cs-concl **cs-shallow**
cs-simp:
cat-cs-simps
cat-Rel-components(*1*)
cat-Set-components(*1*)
Set-Rel.subcat-CId[*symmetric*]
Set-Rel.subcat-Comp-simp[*symmetric*]
cs-intro:
cat-cs-intros
cat-rel-par-Set-cs-intros
V-cs-intros
cat-prod-cs-intros
Set-Rel.subcat-is-arrD
)

qed

qed *auto*

qed

34.9 Dagger monoidal categories

34.9.1 Background

See [13] for further information.

named-theorems *dmcat-field-simps*

named-theorems *dmcat-cs-simps*
named-theorems *dmcat-cs-intros*

definition *DMcat* :: *V* **where** [*dmcat-field-simps*]: *DMcat* = *0*
definition *DMdag* :: *V* **where** [*dmcat-field-simps*]: *DMdag* = $1_{\mathbb{N}}$
definition *DMcf* :: *V* **where** [*dmcat-field-simps*]: *DMcf* = $2_{\mathbb{N}}$
definition *DMe* :: *V* **where** [*dmcat-field-simps*]: *DMe* = $3_{\mathbb{N}}$
definition *DMα* :: *V* **where** [*dmcat-field-simps*]: *DMα* = $4_{\mathbb{N}}$
definition *DMl* :: *V* **where** [*dmcat-field-simps*]: *DMl* = $5_{\mathbb{N}}$
definition *DMr* :: *V* **where** [*dmcat-field-simps*]: *DMr* = $6_{\mathbb{N}}$

abbreviation *DMDag-app* :: *V* ⇒ *V* (‹$\dagger_{MC}$›)
where $\dagger_{MC}$ $\mathfrak{C}$ ≡ $\mathfrak{C}(\!|DMdag|\!)$

34.9.2 Slicing

Dagger category.

definition *dmcat-dagcat* :: *V* ⇒ *V*
where *dmcat-dagcat* $\mathfrak{C}$ = [$\mathfrak{C}(\!|DMcat|\!)$, $\mathfrak{C}(\!|DMdag|\!)$]$_\circ$

lemma *dmcat-dagcat-components*[*slicing-simps*]:
shows *dmcat-dagcat* $\mathfrak{C}(\!|DagCat|\!)$ = $\mathfrak{C}(\!|DMcat|\!)$
and *dmcat-dagcat* $\mathfrak{C}(\!|DagDag|\!)$ = $\mathfrak{C}(\!|DMdag|\!)$
unfolding *dmcat-dagcat-def dmcat-field-simps dag-field-simps*
by (*auto simp*: *nat-omega-simps*)

Monoidal category.

definition *dmcat-mcat* :: $V \Rightarrow V$
  where *dmcat-mcat* $\mathfrak{C}$ = $[\mathfrak{C}(\!|DMcat|\!), \mathfrak{C}(\!|DMcf|\!), \mathfrak{C}(\!|DMe|\!), \mathfrak{C}(\!|DM\alpha|\!), \mathfrak{C}(\!|DMl|\!), \mathfrak{C}(\!|DMr|\!)]_{\circ}$

lemma *dmcat-mcat-components*[*slicing-simps*]:
  shows *dmcat-mcat* $\mathfrak{C}(\!|Mcat|\!)$ = $\mathfrak{C}(\!|DMcat|\!)$
    and *dmcat-mcat* $\mathfrak{C}(\!|Mcf|\!)$ = $\mathfrak{C}(\!|DMcf|\!)$
    and *dmcat-mcat* $\mathfrak{C}(\!|Me|\!)$ = $\mathfrak{C}(\!|DMe|\!)$
    and *dmcat-mcat* $\mathfrak{C}(\!|M\alpha|\!)$ = $\mathfrak{C}(\!|DM\alpha|\!)$
    and *dmcat-mcat* $\mathfrak{C}(\!|Ml|\!)$ = $\mathfrak{C}(\!|DMl|\!)$
    and *dmcat-mcat* $\mathfrak{C}(\!|Mr|\!)$ = $\mathfrak{C}(\!|DMr|\!)$
  unfolding *dmcat-mcat-def dmcat-field-simps mcat-field-simps*
  by (*auto simp*: *nat-omega-simps*)

34.9.3 Definition and elementary properties

locale *dagger-monoidal-category* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *dmcat-length*[*dmcat-cs-simps*]: *vcard* $\mathfrak{C}$ = $\mathcal{T}_{\mathbb{N}}$
    and *dmcat-dagger-category*: *dagger-category* α (*dmcat-dagcat* $\mathfrak{C}$)
    and *dmcat-monoidal-category*: *monoidal-category* α (*dmcat-mcat* $\mathfrak{C}$)
    and *dmcat-compatibility*:
      $[\![\ g : c \mapsto_{\mathfrak{C}(\!|DMcat|\!)} d; f : a \mapsto_{\mathfrak{C}(\!|DMcat|\!)} b\ ]\!] \Longrightarrow$
        $\dagger_{MC}\ \mathfrak{C}(\!|ArrMap|\!)(\!|g \otimes_{HM.A} \mathfrak{C}(\!|DMcf|\!)\ f|\!)$ =
          $\dagger_{MC}\ \mathfrak{C}(\!|ArrMap|\!)(\!|g|\!) \otimes_{HM.A} \mathfrak{C}(\!|DMcf|\!)\ \dagger_{MC}\ \mathfrak{C}(\!|ArrMap|\!)(\!|f|\!)$
    and *dmcat-Mα-unital*: $A \in_{\circ} (\mathfrak{C}(\!|DMcat|\!)\,\widehat{}_{C3})(\!|Obj|\!) \Longrightarrow$
      $\dagger_{MC}\ \mathfrak{C}(\!|ArrMap|\!)(\!|\mathfrak{C}(\!|DM\alpha|\!)(\!|NTMap|\!)(\!|A|\!)|\!)$ = $(\mathfrak{C}(\!|DM\alpha|\!)(\!|NTMap|\!)(\!|A|\!))^{-1}{}_{C}\mathfrak{C}(\!|DMcat|\!)$
    and *dmcat-Ml-unital*: $a \in_{\circ} \mathfrak{C}(\!|DMcat|\!)(\!|Obj|\!) \Longrightarrow$
      $\dagger_{MC}\ \mathfrak{C}(\!|ArrMap|\!)(\!|\mathfrak{C}(\!|DMl|\!)(\!|NTMap|\!)(\!|a|\!)|\!)$ = $(\mathfrak{C}(\!|DMl|\!)(\!|NTMap|\!)(\!|a|\!))^{-1}{}_{C}\mathfrak{C}(\!|DMcat|\!)$
    and *dmcat-Mr-unital*: $a \in_{\circ} \mathfrak{C}(\!|DMcat|\!)(\!|Obj|\!) \Longrightarrow$
      $\dagger_{MC}\ \mathfrak{C}(\!|ArrMap|\!)(\!|\mathfrak{C}(\!|DMr|\!)(\!|NTMap|\!)(\!|a|\!)|\!)$ = $(\mathfrak{C}(\!|DMr|\!)(\!|NTMap|\!)(\!|a|\!))^{-1}{}_{C}\mathfrak{C}(\!|DMcat|\!)$

Rules.

lemma (**in** *dagger-monoidal-category*)
  dagger-monoidal-category-axioms'[*dmcat-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *dagger-monoidal-category* α' $\mathfrak{C}$
  unfolding *assms* **by** (*rule dagger-monoidal-category-axioms*)

mk-ide rf
  dagger-monoidal-category-def[*unfolded dagger-monoidal-category-axioms-def*]
  |*intro dagger-monoidal-categoryI*[*intro*]|
  |*dest dagger-monoidal-categoryD*[*dest*]|
  |*elim dagger-monoidal-categoryE*[*elim*]|

Elementary properties.

lemma *dmcat-eqI*:
  assumes *dagger-monoidal-category* α $\mathfrak{A}$
    and *dagger-monoidal-category* α $\mathfrak{B}$
    and $\mathfrak{A}(\!|DMcat|\!)$ = $\mathfrak{B}(\!|DMcat|\!)$
    and $\mathfrak{A}(\!|DMdag|\!)$ = $\mathfrak{B}(\!|DMdag|\!)$
    and $\mathfrak{A}(\!|DMcf|\!)$ = $\mathfrak{B}(\!|DMcf|\!)$
    and $\mathfrak{A}(\!|DMe|\!)$ = $\mathfrak{B}(\!|DMe|\!)$
    and $\mathfrak{A}(\!|DM\alpha|\!)$ = $\mathfrak{B}(\!|DM\alpha|\!)$
    and $\mathfrak{A}(\!|DMl|\!)$ = $\mathfrak{B}(\!|DMl|\!)$
    and $\mathfrak{A}(\!|DMr|\!)$ = $\mathfrak{B}(\!|DMr|\!)$
  shows $\mathfrak{A}$ = $\mathfrak{B}$
proof–
  interpret $\mathfrak{A}$: *dagger-monoidal-category* α $\mathfrak{A}$ **by** (*rule assms*(*1*))

interpret $\mathfrak{B}$: *dagger-monoidal-category* α $\mathfrak{B}$ **by** (*rule assms*(*2*))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom*: $\mathcal{D}_\circ\ \mathfrak{A} = \mathcal{T}_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *dmcat-cs-simps V-cs-simps*)
show $\mathcal{D}_\circ\ \mathfrak{A} = \mathcal{D}_\circ\ \mathfrak{B}$
by (*cs-concl* **cs-shallow cs-simp**: *dmcat-cs-simps V-cs-simps*)
show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{A} \Longrightarrow \mathfrak{A}(\!|a|\!) = \mathfrak{B}(\!|a|\!)$ **for** a
by (*unfold dom*, *elim-in-numeral*, *insert assms*)
(*auto simp*: *dmcat-field-simps*)
qed *auto*
qed

Slicing.

context *dagger-monoidal-category*
begin

interpretation *dagcat*: *dagger-category* α ‹*dmcat-dagcat* $\mathfrak{C}$›
by (*rule dmcat-dagger-category*)

sublocale *DMCat*: *category* α ‹$\mathfrak{C}(\!|DMcat|\!)$›
by (*rule dagcat.DagCat.category-axioms*[*unfolded slicing-simps*])

sublocale *DMDag*: *is-functor* α ‹*op-cat* ($\mathfrak{C}(\!|DMcat|\!)$)› ‹$\mathfrak{C}(\!|DMcat|\!)$› ‹$\dagger_{MC}$ $\mathfrak{C}$›
by (*rule dagcat.DagDag.is-functor-axioms*[*unfolded slicing-simps*])

lemmas-with [*unfolded slicing-simps*]:
dmcat-Dom-vdomain[*dmcat-cs-simps*] = *dagcat.dagcat-ObjMap-identity*
and *dmcat-DagCat-idem*[*dmcat-cs-simps*] = *dagcat.dagcat-DagCat-idem*
and *dmcat-is-functor'*[*dmcat-cs-intros*] = *dagcat.dagcat-is-functor'*

end

lemmas [*dmcat-cs-simps*] =
dagger-monoidal-category.dmcat-Dom-vdomain
dagger-monoidal-category.dmcat-DagCat-idem

lemmas [*dmcat-cs-intros*] = *dagger-monoidal-category.dmcat-is-functor'*

context *dagger-monoidal-category*
begin

interpretation *mcat*: *monoidal-category* α ‹*dmcat-mcat* $\mathfrak{C}$›
by (*rule dmcat-monoidal-category*)

sublocale *DMcf*: *is-functor* α ‹$\mathfrak{C}(\!|DMcat|\!) \times_C \mathfrak{C}(\!|DMcat|\!)$› ‹$\mathfrak{C}(\!|DMcat|\!)$› ‹$\mathfrak{C}(\!|DMcf|\!)$›
by (*rule mcat.Mcf.is-functor-axioms*[*unfolded slicing-simps*])

sublocale *DM*α: *is-iso-ntcf*
α ‹$\mathfrak{C}(\!|DMcat|\!)\hat{}_{C3}$› ‹$\mathfrak{C}(\!|DMcat|\!)$› ‹*cf-blcomp* ($\mathfrak{C}(\!|DMcf|\!)$)› ‹*cf-brcomp* ($\mathfrak{C}(\!|DMcf|\!)$)› ‹$\mathfrak{C}(\!|DM\alpha|\!)$›
by (*rule mcat.M*α*.is-iso-ntcf-axioms*[*unfolded slicing-simps*])

sublocale *DMl*: *is-iso-ntcf*
α
‹$\mathfrak{C}(\!|DMcat|\!)$›
‹$\mathfrak{C}(\!|DMcat|\!)$›
‹$\mathfrak{C}(\!|DMcf|\!)_{\mathfrak{C}(\!|DMcat|\!),\mathfrak{C}(\!|DMcat|\!)}(\mathfrak{C}(\!|DMe|\!),-)_{CF}$›
‹*cf-id* ($\mathfrak{C}(\!|DMcat|\!)$)›

‹$\mathfrak{C}(\!|DMl|\!)$›
by (*rule mcat.Ml.is-iso-ntcf-axioms*[*unfolded slicing-simps*])

sublocale *DMr*: *is-iso-ntcf*
α
‹$\mathfrak{C}(\!|DMcat|\!)$›
‹$\mathfrak{C}(\!|DMcat|\!)$›
‹$\mathfrak{C}(\!|DMcf|\!)_{\mathfrak{C}(\!|DMcat|\!),\mathfrak{C}(\!|DMcat|\!)}(-,\mathfrak{C}(\!|DMe|\!))_{CF}$›
‹*cf-id* ($\mathfrak{C}(\!|DMcat|\!)$)›
‹$\mathfrak{C}(\!|DMr|\!)$›
by (*rule mcat.Mr.is-iso-ntcf-axioms*[*unfolded slicing-simps*])

lemmas-with [*unfolded slicing-simps*]:
dmcat-Me-is-obj[*dmcat-cs-intros*] = *mcat.mcat-Me-is-obj*
and *dmcat-pentagon* = *mcat.mcat-pentagon*
and *dmcat-triangle*[*dmcat-cs-simps*] = *mcat.mcat-triangle*

end

lemmas [*dmcat-cs-intros*] = *dagger-monoidal-category.dmcat-Me-is-obj*
lemmas [*dmcat-cs-simps*] = *dagger-monoidal-category.dmcat-triangle*

34.10 *Rel* as a dagger monoidal category

34.10.1 Definition and elementary properties

definition *dmcat-Rel* :: $V \Rightarrow V \Rightarrow V$
where *dmcat-Rel* α *a* =
[
cat-Rel α,
$\dagger_{C.Rel}$ α,
cf-prod-2-Rel (*cat-Rel* α),
set $\{a\}$,
Mα-Rel (*cat-Rel* α),
Ml-Rel (*cat-Rel* α) *a*,
Mr-Rel (*cat-Rel* α) *a*
]$_\circ$

Components.

lemma *dmcat-Rel-components*:
shows *dmcat-Rel* α *a*($\!|DMcat|\!$) = *cat-Rel* α
and *dmcat-Rel* α *a*($\!|DMdag|\!$) = $\dagger_{C.Rel}$ α
and *dmcat-Rel* α *a*($\!|DMcf|\!$) = *cf-prod-2-Rel* (*cat-Rel* α)
and *dmcat-Rel* α *a*($\!|DMe|\!$) = *set* $\{a\}$
and *dmcat-Rel* α *a*($\!|DM\alpha|\!$) = *Mα-Rel* (*cat-Rel* α)
and *dmcat-Rel* α *a*($\!|DMl|\!$) = *Ml-Rel* (*cat-Rel* α) *a*
and *dmcat-Rel* α *a*($\!|DMr|\!$) = *Mr-Rel* (*cat-Rel* α) *a*
unfolding *dmcat-Rel-def dmcat-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *dmcat-dagcat-dmcat-Rel*: *dmcat-dagcat* (*dmcat-Rel* α *a*) = *dagcat-Rel* α
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ$ (*dmcat-dagcat* (*dmcat-Rel* α *a*)) = $2_{\mathbb{N}}$
unfolding *dmcat-dagcat-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*dagcat-Rel* α) = $2_{\mathbb{N}}$
unfolding *dagcat-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*dmcat-dagcat* (*dmcat-Rel* α *a*)) = $\mathcal{D}_\circ$ (*dagcat-Rel* α)
unfolding *dom-lhs dom-rhs* **by** *simp*

show $A \in_\circ \mathcal{D}_\circ\ (dmcat\text{-}dagcat\ (dmcat\text{-}Rel\ \alpha\ a)) \Longrightarrow$
$dmcat\text{-}dagcat\ (dmcat\text{-}Rel\ \alpha\ a)(\!|A|\!) = dagcat\text{-}Rel\ \alpha(\!|A|\!)$
for A
by
(
unfold dom-lhs,
elim-in-numeral,
unfold dmcat-dagcat-def dmcat-field-simps dmcat-Rel-def dagcat-Rel-def
)
(*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *dmcat-dagcat-def dagcat-Rel-def*)

lemma *dmcat-mcat-dmcat-Rel*: $dmcat\text{-}mcat\ (dmcat\text{-}Rel\ \alpha\ a) = mcat\text{-}Rel\ \alpha\ a$
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ\ (dmcat\text{-}mcat\ (dmcat\text{-}Rel\ \alpha\ a)) = 6_{\mathbf{N}}$
unfolding *dmcat-mcat-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ\ (mcat\text{-}Rel\ \alpha\ a) = 6_{\mathbf{N}}$
unfolding *mcat-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ\ (dmcat\text{-}mcat\ (dmcat\text{-}Rel\ \alpha\ a)) = \mathcal{D}_\circ\ (mcat\text{-}Rel\ \alpha\ a)$
unfolding *dom-lhs dom-rhs* **by** *simp*
show $A \in_\circ \mathcal{D}_\circ\ (dmcat\text{-}mcat\ (dmcat\text{-}Rel\ \alpha\ a)) \Longrightarrow$
$dmcat\text{-}mcat\ (dmcat\text{-}Rel\ \alpha\ a)(\!|A|\!) = mcat\text{-}Rel\ \alpha\ a(\!|A|\!)$
for A
by
(
unfold dom-lhs,
elim-in-numeral,
unfold dmcat-mcat-def dmcat-field-simps dmcat-Rel-def mcat-Rel-def
)
(*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *dmcat-mcat-def mcat-Rel-def*)

34.10.2 *Rel* is a dagger monoidal category

lemma (**in** $\mathcal{Z}$) *dagger-monoidal-category-dmcat-Rel*:
assumes $A \in_\circ cat\text{-}Rel\ \alpha(\!|Obj|\!)$
shows $dagger\text{-}monoidal\text{-}category\ \alpha\ (dmcat\text{-}Rel\ \alpha\ A)$
proof–
interpret *Rel*: $category\ \alpha\ \langle cat\text{-}Rel\ \alpha\rangle$ **by** (*rule category-cat-Rel*)
interpret *dag-Rel*: $is\text{-}iso\text{-}functor\ \alpha\ \langle op\text{-}cat\ (cat\text{-}Rel\ \alpha)\rangle\ \langle cat\text{-}Rel\ \alpha\rangle\ \langle \dagger_{C.Rel}\ \alpha\rangle$
by (*rule cf-dag-Rel-is-iso-functor*)
show *?thesis*
proof(*rule dagger-monoidal-categoryI*)
show $\mathcal{Z}\ \alpha$ **by** *auto*
show *vfsequence* $(dmcat\text{-}Rel\ \alpha\ A)$ **unfolding** *dmcat-Rel-def* **by** *simp*
show $vcard\ (dmcat\text{-}Rel\ \alpha\ A) = \mathcal{T}_{\mathbf{N}}$
unfolding *dmcat-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $dagger\text{-}category\ \alpha\ (dmcat\text{-}dagcat\ (dmcat\text{-}Rel\ \alpha\ A))$
unfolding *dmcat-dagcat-dmcat-Rel* **by** (*rule dagger-category-dagcat-Rel*)
show $monoidal\text{-}category\ \alpha\ (dmcat\text{-}mcat\ (dmcat\text{-}Rel\ \alpha\ A))$
unfolding *dmcat-mcat-dmcat-Rel* **by** (*intro monoidal-category-mcat-Rel assms*)
show
$\dagger_{MC}\ (dmcat\text{-}Rel\ \alpha\ A)(\!|ArrMap|\!)(\!|g \otimes_{HM.A} dmcat\text{-}Rel\ \alpha\ A(\!|DMcf|\!)\ f|\!) =$
$\dagger_{MC}\ (dmcat\text{-}Rel\ \alpha\ A)(\!|ArrMap|\!)(\!|g|\!) \otimes_{HM.A} dmcat\text{-}Rel\ \alpha\ A(\!|DMcf|\!)$
$\dagger_{MC}\ (dmcat\text{-}Rel\ \alpha\ A)(\!|ArrMap|\!)(\!|f|\!)$
if $g : c \mapsto_{dmcat\text{-}Rel\ \alpha\ A(\!|DMcat|\!)} d$ **and** $f : a \mapsto_{dmcat\text{-}Rel\ \alpha\ A(\!|DMcat|\!)} b$
for $c\ d\ g\ a\ b\ f$
using *that*

```
  unfolding dmcat-Rel-components
  by
    (
      cs-concl cs-shallow
        cs-simp: cf-dag-Rel-ArrMap-app-prod-2-Rel cat-cs-simps cat-op-simps
        cs-intro: cat-cs-intros cat-prod-cs-intros cat-op-intros
    )
show
  †M C (dmcat-Rel α A)(|ArrMap|)(|dmcat-Rel α A(|DMα|)(|NTMap|)(|BCD|)|) =
    (dmcat-Rel α A(|DMα|)(|NTMap|)(|BCD|))⁻¹C dmcat-Rel α A(|DMcat|)
  if BCD ∈∘ (dmcat-Rel α A(|DMcat|)^C3)(|Obj|) for BCD
proof–
  from that obtain B C D
    where BCD-def: BCD = [B, C, D]∘
      and B: B ∈∘ cat-Rel α(|Obj|)
      and C: C ∈∘ cat-Rel α(|Obj|)
      and D: D ∈∘ cat-Rel α(|Obj|)
    unfolding dmcat-Rel-components
    by
      (
        elim cat-prod-3-ObjE
          [
            unfolded dmcat-Rel-components,
            OF Rel.category-axioms Rel.category-axioms Rel.category-axioms
          ]
      )
  from B C D show ?thesis
    unfolding dmcat-Rel-components BCD-def
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Rel-cs-simps cat-cs-simps
          cs-intro:
            cat-Rel-is-arrD
            cat-cs-intros
            cat-Rel-par-set-cs-intros
            cat-prod-cs-intros
      )
qed
show
  †M C (dmcat-Rel α A)(|ArrMap|)(|dmcat-Rel α A(|DMl|)(|NTMap|)(|B|)|) =
    (dmcat-Rel α A(|DMl|)(|NTMap|)(|B|))⁻¹C dmcat-Rel α A(|DMcat|)
  if B ∈∘ dmcat-Rel α A(|DMcat|)(|Obj|) for B
  using assms that
  unfolding dmcat-Rel-components
  by
    (
      cs-concl cs-shallow
        cs-simp: cat-Rel-cs-simps
        cs-intro: cat-Rel-is-arrD cat-cs-intros cat-arrow-cs-intros
    )+
show
  †M C (dmcat-Rel α A)(|ArrMap|)(|dmcat-Rel α A(|DMr|)(|NTMap|)(|B|)|) =
    (dmcat-Rel α A(|DMr|)(|NTMap|)(|B|))⁻¹C dmcat-Rel α A(|DMcat|)
  if B ∈∘ dmcat-Rel α A(|DMcat|)(|Obj|) for B
  using assms that
  unfolding dmcat-Rel-components
```

```
    by
      (
        cs-concl cs-shallow
          cs-simp: cat-Rel-cs-simps
          cs-intro: cat-Rel-is-arrD cat-cs-intros cat-arrow-cs-intros
      )+
  qed
qed
```

References

[1] nLab. URL https://ncatlab.org/nlab/show/HomePage.

[2] Wikipedia, 2001. URL https://www.wikipedia.org/.

[3] P. Bodo. *Categories and Functors*. Academic Press, New York, 1970.

[4] P. I. Etingof, S. Gelaki, D. Nikshych, and V. Ostrik. *Tensor Categories*. Number 205 in Mathematical Surveys and Monographs. American Mathematical Society, Providence, 2015. ISBN 978-1-4704-2024-6.

[5] S. Feferman and G. Kreisel. Set-Theoretical Foundations of Category Theory. In M. Barr, P. Berthiaume, B. J. Day, J. Duskin, S. Feferman, G. M. Kelly, S. Mac Lane, M. Tierney, and R. F. C. Walters, editors, *Reports of the Midwest Category Seminar III*, Lecture Notes in Mathematics, pages 201–247, Heidelberg, 1969. Springer.

[6] F. Haftmann. Sketch-and-Explore, 2021. URL https://isabelle.in.tum.de/library/HOL/HOL-ex/Sketch_and_Explore.html.

[7] S. Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer, New York, 2 edition, 2010. ISBN 978-1-4419-3123-8.

[8] M. Milehins. Category Theory for ZFC in HOL I: Foundations: Design Patterns, Set Theory, Digraphs, Semicategories. *Archive of Formal Proofs*, 2021.

[9] S. Obua. Partizan Games in Isabelle/HOLZF. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *ICTAC 2006*, volume 4281, pages 272–286. Springer, Berlin, 2006. ISBN 978-3-540-48815-6.

[10] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986.

[11] L. C. Paulson. Zermelo Fraenkel Set Theory in Higher-Order Logic. *Archive of Formal Proofs*, 2019.

[12] E. Riehl. *Category Theory in Context*. Emily Riehl, 2016.

[13] P. Selinger. A Survey of Graphical Languages for Monoidal Categories. In B. Coecke, editor, *New Structures for Physics*, volume 813, pages 289–355. Springer, Heidelberg, 2010. ISBN 978-3-642-12820-2.

[14] G. Takeuti and W. M. Zaring. *Introduction to Axiomatic Set Theory*. Springer-Verlag, Heidelberg, 1971. ISBN 0-387-05302-6.