Category Theory for ZFC in HOL I: Foundations Design Patterns, Set Theory, Digraphs, Semicategories

Mihails Milehins

March 19, 2025

Abstract

This article provides a foundational framework for the formalization of category theory in the object logic *ZFC in HOL* ([51], also see [47]) of the formal proof assistant *Isabelle* [49]. More specifically, this article provides a formalization of canonical set-theoretic constructions internalized in the type V associated with the ZFC in HOL, establishes a design pattern for the formalization of mathematical structures using sequences and locales [33, 8, 9], and showcases the developed infrastructure by providing formalizations of the elementary theories of digraphs and semicategories. The methodology chosen for the formalization of the theories of digraphs and semicategories (and categories in future articles) rests on the ideas that were originally expressed in [28]. Thus, in the context of this work, each of the aforementioned mathematical structures is represented as a term of the type V embedded into a stage of the von Neumann hierarchy [59].

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" [31] 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

Introduction

1.1 Background

This article presents a foundational framework that will be used for the formalization of elements of the theory of 1-categories in the object logic *ZFC in HOL* ([51], also see [47]) of the formal proof assistant *Isabelle* [49] in future articles. It is important to note that this chapter serves as an introduction to the entire development and not merely its foundational part.

There already exist several formalizations of the foundations of category theory in Isabelle. In the context of the work presented here, the most relevant formalizations (listed in the chronological order) are [25, 24], [48], [34] and [58]. Arguably, the most well developed and maintained entry is [58]: it subsumes the majority of the content of [48] and [34].

From the perspective of the methodology that was chosen for the formalization, this work differs significantly from the aforementioned previous work. In particular, the categories are modelled as terms of the type V and no attempt is made to generalize the concept of a category to arbitrary types. The inspiration for the chosen approach is drawn from [28], [52] and [57].

The primary references for this work are *Categories for the Working Mathematician* [39] by Saunders Mac Lane, *Category Theory in Context* by Emily Riehl [56] and *Categories and Functors* by Bodo Pareigis [15]. The secondary sources of information include the textbooks [7] and [32], as well as several online encyclopedias (including [3], [5], [4] and [2]). Of course, inspiration was also drawn from the previous formalizations of category theory in Isabelle.

It is likely that none of the content that is formalized in this work is original in nature. However, explicit citations are not provided for many results that were deemed to be trivial.

1.2 Related and previous work

To the best knowledge of the author, this work is the first attempt to develop a formalization of elements of category theory in the object logic ZFC in HOL by modelling categories as terms of the type V. However, it should be noted that the formalization of category theory in [34] largely rested on the object logic HOL/ZF [47], which is equiconsistent with the ZFC in HOL [51]. Nonetheless, in [34], the objects and arrows associated with categories were modelled as terms of arbitrary types. The object logic HOL/ZF was used for the exposition of the category *Set* of all sets and functions between them and a variety of closely related concepts. In this sense, the methodology employed in [34] could be seen as a combination of the methodology employed in this work and the methodology followed in [48] and [58]. Furthermore, in [26], the authors have experimented with the formalization of category theory in Higher-Order Tarski-Grothendieck (HOTG) theory [17] using a methodology that shares many similarities with the approach that was chosen in this study.

The formalizations of various elements of category theory in other proof assistants are abundant. While a survey of such formalizations is outside of the scope of this work, it is important to note that there exist at least two examples of the formalization of elements of category theory in a set-theoretic setting similar to the one that is used in this work. More specifically, elements of category theory were formalized in the Tarski-Grothendieck Set Theory in the Mizar proof

assistant [1] (and published in the associated electronic journal [30]) and the proof assistant Metamath [40]. The following references contain some of the relevant articles in [30], but the list may not be exhaustive: [18, 19, 21, 61, 20, 43, 60, 44, 45, 13, 22, 62, 23, 10, 63, 11, 64, 46, 36, 37, 12, 38, 53, 29, 54, 55].

Set Theory for Category Theory

2.1 Introduction

2.1.1 Background

This chapter presents a formalization of the elements of set theory in the object logic *ZFC in HOL* ([51], also see [47]) of the formal proof assistant Isabelle [49]. The emphasis of this work is on the improvement of the convenience of the formalization of abstract mathematics internalized in the type *V*.

2.1.2 References, related and previous work

The results that are presented in this chapter cannot be traced to a single source of information. Nonetheless, the results are not original. A significant number of these results was carried over (with amendments) from the main library of Isabelle/HOL [6]. Other results were inspired by elements of the content of the books *Introduction to Axiomatic Set Theory* by G. Takeuti and W. M. Zaring [59], *Theory of Sets* by N. Bourbaki [16] and *Algebra* by T. W. Hungerford [32]. Furthermore, several online encyclopedias and forums (including Wikipedia [5], ProofWiki [4], Encyclopedia of Mathematics [2], nLab [3] and Mathematics Stack Exchange) were used consistently throughout the development of this chapter. Inspiration for the work presented in this chapter has also been drawn from a similar ongoing project in the formalization of mathematics in the system HOTG (Higher Order Tarski-Grothendieck) [17, 26].

It should also be mentioned that the Isabelle/HOL and the Isabelle/ML code from the main distribution of Isabelle2020 and certain posts on the mailing list of Isabelle were frequently reused (with amendments) during the development of this chapter. Some of the specific examples of such reuse are

- The adoption of the implementation of the command **lemmas-with** that is available as part of the framework Types-To-Sets in the main distribution of Isabelle2020.
- The notation for the "multiway-if" was written by Manuel Eberl and appeared in a post on the mailing list of Isabelle: [27].

hide-const (**open**) *list.set Sum subset*

lemmas *ord-of-nat-zero* = *ord-of-nat.simps*(1)

2.1.3 Notation

abbreviation (*input*) *qm* (‹(- *?* - : -)› [0, 0, 10] 10)
  where *C ? A : B* ≡ *if C then A else B*
abbreviation (*input*) *if*2 **where** *if*2 *a b* ≡ (λ*i*. (*i* = 0 *? a* : *b*))

2.1.4 Further foundational results

lemma *theD*:
  assumes ∃!*x*. *P x* **and** *x* = (*THE x*. *P x*)
  shows *P x* **and** *P y* ⟹ *x* = *y*

using *assms* **by** (*metis theI*)+

lemmas [*intro*] = *bij-betw-imageI*

lemma *bij-betwE*[*elim*]:
assumes *bij-betw f A B* **and** ⟦ *inj-on f A*; *f ‘ A = B* ⟧ ⟹ *P*
shows *P*
using *assms* **unfolding** *bij-betw-def* **by** *auto*

lemma *bij-betwD*[*dest*]:
assumes *bij-betw f A B*
shows *inj-on f A* **and** *f ‘ A = B*
using *assms* **by** *auto*

lemma *ex1D*: ∃!*x*. *P x* ⟹ *P x* ⟹ *P y* ⟹ *x = y* **by** *clarsimp*

2.2 Further set algebra and other miscellaneous results

2.2.1 Background

This section presents further set algebra and various elementary properties of sets.

Many of the results that are presented in this section were carried over (with amendments) from the theories *Set* and *Complete-Lattices* in the main library.

declare *elts-sup-iff*[*simp del*]

2.2.2 Further notation

Set membership

abbreviation *vmember* :: $V \Rightarrow V \Rightarrow bool$ (‹(-/ $\in_\circ$ -)› [51, 51] 50)
  where *vmember x A* ≡ ($x \in elts\ A$)
notation *vmember* (‹$'(\in_\circ')$›)
  and *vmember* (‹(-/ $\in_\circ$ -)› [51, 51] 50)

abbreviation *not-vmember* :: $V \Rightarrow V \Rightarrow bool$ (‹(-/ $\notin_\circ$ -)› [51, 51] 50)
  where *not-vmember x A* ≡ ($x \notin elts\ A$)
notation
  not-vmember (‹$'(\notin_\circ')$›) **and**
  not-vmember (‹(-/ $\notin_\circ$ -)› [51, 51] 50)

Subsets

abbreviation *vsubset* :: $V \Rightarrow V \Rightarrow bool$
  where *vsubset* ≡ *less*
abbreviation *vsubset-eq* :: $V \Rightarrow V \Rightarrow bool$
  where *vsubset-eq* ≡ *less-eq*

notation *vsubset* (‹$'(\subset_\circ')$›)
  and *vsubset* (‹(-/ $\subset_\circ$ -)› [51, 51] 50)
  and *vsubset-eq* (‹$'(\subseteq_\circ')$›)
  and *vsubset-eq* (‹(-/ $\subseteq_\circ$ -)› [51, 51] 50)

Ball

syntax
  -*VBall* :: *pttrn* ⇒ $V \Rightarrow bool \Rightarrow bool$ (‹(3∀ (-/$\in_\circ$-)./ -)› [0, 0, 10] 10)
  -*VBex* :: *pttrn* ⇒ $V \Rightarrow bool \Rightarrow bool$ (‹(3∃ (-/$\in_\circ$-)./ -)› [0, 0, 10] 10)
  -*VBex*1 :: *pttrn* ⇒ $V \Rightarrow bool \Rightarrow bool$ (‹(3∃!(-/$\in_\circ$-)./ -)› [0, 0, 10] 10)

syntax-consts
  -*VBall* ⇌ *Ball* **and**
  -*VBex* ⇌ *Bex* **and**
  -*VBex*1 ⇌ *Ex*1

translations
  $\forall x \in_\circ A.\ P \rightleftharpoons CONST\ Ball\ (CONST\ elts\ A)\ (\lambda x.\ P)$
  $\exists x \in_\circ A.\ P \rightleftharpoons CONST\ Bex\ (CONST\ elts\ A)\ (\lambda x.\ P)$
  $\exists! x \in_\circ A.\ P \rightharpoonup \exists! x.\ x \in_\circ A \wedge P$

VLambda

The following notation was adapted from [50].

syntax -*vlam* :: [*pttrn*, V, V] ⇒ V (‹(3λ-$\in_\circ$-./ -)› 10)

syntax-consts *-vlam* ⇌ *VLambda*
translations $\lambda x \in_\circ A.\ f \rightleftharpoons CONST\ VLambda\ A\ (\lambda x.\ f)$

Intersection and union

abbreviation *vintersection* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\cap_\circ$› 70)
  where $(\cap_\circ) \equiv (\sqcap)$
notation *vintersection* (**infixl** ‹$\cap_\circ$› 70)

abbreviation *vunion* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\cup_\circ$› 65)
  where *vunion* ≡ *sup*
notation *vunion* (**infixl** ‹$\cup_\circ$› 65)

abbreviation *VInter* :: $V \Rightarrow V$ (‹$\bigcap_\circ$›)
  where $\bigcap_\circ\ A \equiv \bigsqcap\ (elts\ A)$
notation *VInter* (‹$\bigcap_\circ$›)

abbreviation *VUnion* :: $V \Rightarrow V$ (‹$\bigcup_\circ$›)
  where $\bigcup_\circ A \equiv \bigsqcup\ (elts\ A)$
notation *VUnion* (‹$\bigcup_\circ$›)

Miscellaneous

notation *app* (‹-(|-|)› [999, 50] 999)
notation *vtimes* (**infixr** ‹$\times_\circ$› 80)

2.2.3 Elementary results.

lemma *vempty-nin*[*simp*]: $a \notin_\circ 0$ **by** *simp*

lemma *vemptyE*:
  assumes $A \neq 0$
  obtains x **where** $x \in_\circ A$
  using *assms trad-foundation* **by** *auto*

lemma *in-set-CollectI*:
  assumes $P\ x$ **and** *small* $\{x.\ P\ x\}$
  shows $x \in_\circ set\ \{x.\ P\ x\}$
  using *assms* **by** *simp*

lemma *small-setcompr2*:
  assumes *small* $\{f\ x\ y \mid x\ y.\ P\ x\ y\}$ **and** $a \in_\circ set\ \{f\ x\ y \mid x\ y.\ P\ x\ y\}$
  obtains $x'\ y'$ **where** $a = f\ x'\ y'$ **and** $P\ x'\ y'$
  using *assms* **by** *auto*

lemma *in-small-setI*:
  assumes *small* A **and** $x \in A$
  shows $x \in_\circ set\ A$
  using *assms* **by** *simp*

lemma *in-small-setD*:
  assumes $x \in_\circ set\ A$ **and** *small* A
  shows $x \in A$
  using *assms* **by** *simp*

lemma *in-small-setE*:
  assumes $a \in_\circ set\ A$ **and** *small* A
  obtains $a \in A$
  using *assms* **by** *auto*

lemma *small-set-vsubset*:
assumes *small A* **and** $A \subseteq$ *elts B*
shows *set A* $\subseteq_\circ$ *B*
using *assms* **by** *auto*

lemma *some-in-set-if-set-neq-vempty*[*simp*]:
assumes $A \neq 0$
shows (*SOME x. x* $\in_\circ$ *A*) $\in_\circ$ *A*
by (*meson assms someI-ex vemptyE*)

lemma *small-vsubset-set*[*intro*, *simp*]:
assumes *small B* **and** $A \subseteq B$
shows *set A* $\subseteq_\circ$ *set B*
using *assms* **by** (*auto simp*: *subset-iff-less-eq-V*)

lemma *vset-neq*-1:
assumes $b \notin_\circ A$ **and** $a \in_\circ A$
shows $b \neq a$
using *assms* **by** *auto*

lemma *vset-neq*-2:
assumes $b \in_\circ A$ **and** $a \notin_\circ A$
shows $b \neq a$
using *assms* **by** *auto*

lemma *nin-vinsertI*:
assumes $a \neq b$ **and** $a \notin_\circ A$
shows $a \notin_\circ$ *vinsert b A*
using *assms* **by** *clarsimp*

lemma *vsubset-if-subset*:
assumes *elts A* $\subseteq$ *elts B*
shows $A \subseteq_\circ B$
using *assms* **by** *auto*

lemma *small-set-comprehension*[*simp*]: *small* $\{A\ i \mid i.\ i \in_\circ I\}$
proof(*rule smaller-than-small*)
show *small* (*A* ‘ *elts I*) **by** *auto*
qed *auto*

2.2.4 *VBall*

lemma *vball-cong*:
assumes $A = B$ **and** $\bigwedge x.\ x \in_\circ B \Longrightarrow P\ x \longleftrightarrow Q\ x$
shows $(\forall\, x{\in_\circ}A.\ P\ x) \longleftrightarrow (\forall\, x{\in_\circ}B.\ Q\ x)$
by (*simp add*: *assms*)

lemma *vball-cong-simp*[*cong*]:
assumes $A = B$ **and** $\bigwedge x.\ x \in_\circ B$ =*simp*=> $P\ x \longleftrightarrow Q\ x$
shows $(\forall\, x{\in_\circ}A.\ P\ x) \longleftrightarrow (\forall\, x{\in_\circ}B.\ Q\ x)$
using *assms* **by** (*simp add*: *simp-implies-def*)

2.2.5 *VBex*

lemma *vbex-cong*:
assumes $A = B$ **and** $\bigwedge x.\ x \in_\circ B \Longrightarrow P\ x \longleftrightarrow Q\ x$
shows $(\exists\, x{\in_\circ}A.\ P\ x) \longleftrightarrow (\exists\, x{\in_\circ}B.\ Q\ x)$

using *assms* **by** (*simp cong*: *conj-cong*)

lemma *vbex-cong-simp*[*cong*]:
assumes $A = B$ **and** $\bigwedge x.\ x \in_\circ B \Rightarrow P\ x \longleftrightarrow Q\ x$
shows $(\exists\, x{\in_\circ}A.\ P\ x) \longleftrightarrow (\exists\, x{\in_\circ}B.\ Q\ x)$
using *assms* **by** (*simp add*: *simp-implies-def*)

2.2.6 Subset

Rules.

lemma *vsubset-antisym*:
assumes $A \subseteq_\circ B$ **and** $B \subseteq_\circ A$
shows $A = B$
using *assms* **by** *simp*

lemma *vsubsetI*:
assumes $\bigwedge x.\ x \in_\circ A \Longrightarrow x \in_\circ B$
shows $A \subseteq_\circ B$
using *assms* **by** *auto*

lemma *vpsubsetI*:
assumes $A \subseteq_\circ B$ **and** $x \notin_\circ A$ **and** $x \in_\circ B$
shows $A \subset_\circ B$
using *assms* **unfolding** *less-V-def* **by** *auto*

lemma *vsubsetD*:
assumes $A \subseteq_\circ B$
shows $\bigwedge x.\ x \in_\circ A \Longrightarrow x \in_\circ B$
using *assms* **by** *auto*

lemma *vsubsetE*:
assumes $A \subseteq_\circ B$ **and** $(\bigwedge x.\ x \in_\circ A \Longrightarrow x \in_\circ B) \Longrightarrow P$
shows P
using *assms* **by** *auto*

lemma *vpsubsetE*:
assumes $A \subset_\circ B$
obtains x **where** $A \subseteq_\circ B$ **and** $x \notin_\circ A$ **and** $x \in_\circ B$
using *assms* **unfolding** *less-V-def* **by** (*meson V-equalityI vsubsetE*)

lemma *vsubset-iff*: $A \subseteq_\circ B \longleftrightarrow (\forall\, t.\ t \in_\circ A \longrightarrow t \in_\circ B)$ **by** *blast*

Elementary properties.

lemma *vsubset-eq*: $A \subseteq_\circ B \longleftrightarrow (\forall\, x{\in_\circ}A.\ x \in_\circ B)$ **by** *auto*

lemma *vsubset-transitive*[*intro*]:
assumes $A \subseteq_\circ B$ **and** $B \subseteq_\circ C$
shows $A \subseteq_\circ C$
using *assms* **by** *simp*

lemma *vsubset-reflexive*: $A \subseteq_\circ A$ **by** *simp*

Set operations.

lemma *vsubset-vempty*: $0 \subseteq_\circ A$ **by** *simp*

lemma *vsubset-vsingleton-left*: *set* $\{a\} \subseteq_\circ A \longleftrightarrow a \in_\circ A$ **by** *auto*

lemmas *vsubset-vsingleton-leftD*[*dest*] = *vsubset-vsingleton-left*[*THEN iffD*1]
and *vsubset-vsingleton-leftI*[*intro*] = *vsubset-vsingleton-left*[*THEN iffD*2]

lemma *vsubset-vsingleton-right*: $A \subseteq_\circ set\ \{a\} \longleftrightarrow A = set\ \{a\} \vee A = 0$
by (*auto intro*!: *vsubset-antisym*)

lemmas *vsubset-vsingleton-rightD*[*dest*] = *vsubset-vsingleton-right*[*THEN iffD*1]
and *vsubset-vsingleton-rightI*[*intro*] = *vsubset-vsingleton-right*[*THEN iffD*2]

lemma *vsubset-vdoubleton-leftD*[*dest*]:
assumes *set* $\{a, b\} \subseteq_\circ A$
shows $a \in_\circ A$ **and** $b \in_\circ A$
using *assms* **by** *auto*

lemma *vsubset-vdoubleton-leftI*[*intro*]:
assumes $a \in_\circ A$ **and** $b \in_\circ A$
shows *set* $\{a, b\} \subseteq_\circ A$
using *assms* **by** *auto*

lemma *vsubset-vinsert-leftD*[*dest*]:
assumes *vinsert a A* $\subseteq_\circ B$
shows $A \subseteq_\circ B$
using *assms* **by** *auto*

lemma *vsubset-vinsert-leftI*[*intro*]:
assumes $A \subseteq_\circ B$ **and** $a \in_\circ B$
shows *vinsert a A* $\subseteq_\circ B$
using *assms* **by** *auto*

lemma *vsubset-vinsert-vinsertI*[*intro*]:
assumes $A \subseteq_\circ$ *vinsert b B*
shows *vinsert b A* $\subseteq_\circ$ *vinsert b B*
using *assms* **by** *auto*

lemma *vsubset-vinsert-rightI*[*intro*]:
assumes $A \subseteq_\circ B$
shows $A \subseteq_\circ$ *vinsert b B*
using *assms* **by** *auto*

lemmas *vsubset-VPow* = *VPow-le-VPow-iff*

lemmas *vsubset-VPowD* = *vsubset-VPow*[*THEN iffD*1]
and *vsubset-VPowI* = *vsubset-VPow*[*THEN iffD*2]

Special properties.

lemma *vsubset-contraD*:
assumes $A \subseteq_\circ B$ **and** $c \notin_\circ B$
shows $c \notin_\circ A$
using *assms* **by** *auto*

2.2.7 Equality

Rules.

lemma *vequalityD*1:
assumes $A = B$
shows $A \subseteq_\circ B$
using *assms* **by** *simp*

lemma *vequalityD2*:
assumes $A = B$
shows $B \subseteq_\circ A$
using *assms* **by** *simp*

lemma *vequalityE*:
assumes $A = B$ **and** $[\![A \subseteq_\circ B; B \subseteq_\circ A]\!] \Longrightarrow P$
shows P
using *assms* **by** *simp*

lemma *vequalityCE*[*elim*]:
assumes $A = B$ **and** $[\![c \in_\circ A; c \in_\circ B]\!] \Longrightarrow P$ **and** $[\![c \notin_\circ A; c \notin_\circ B]\!] \Longrightarrow P$
shows P
using *assms* **by** *auto*

2.2.8 Binary intersection

lemma *vintersection-def*: $A \cap_\circ B = set\ \{x.\ x \in_\circ A \wedge x \in_\circ B\}$
by (*metis Int-def inf-V-def*)

lemma *small-vintersection-set*[*simp*]: *small* $\{x.\ x \in_\circ A \wedge x \in_\circ B\}$
by (*rule down*[*of* - A]) *auto*

Rules.

lemma *vintersection-iff*[*simp*]: $x \in_\circ A \cap_\circ B \longleftrightarrow x \in_\circ A \wedge x \in_\circ B$
unfolding *vintersection-def* **by** *simp*

lemma *vintersectionI*[*intro*!]:
assumes $x \in_\circ A$ **and** $x \in_\circ B$
shows $x \in_\circ A \cap_\circ B$
using *assms* **by** *simp*

lemma *vintersectionD1*[*dest*]:
assumes $x \in_\circ A \cap_\circ B$
shows $x \in_\circ A$
using *assms* **by** *simp*

lemma *vintersectionD2*[*dest*]:
assumes $x \in_\circ A \cap_\circ B$
shows $x \in_\circ B$
using *assms* **by** *simp*

lemma *vintersectionE*[*elim*!]:
assumes $x \in_\circ A \cap_\circ B$ **and** $x \in_\circ A \Longrightarrow x \in_\circ B \Longrightarrow P$
shows P
using *assms* **by** *simp*

Elementary properties.

lemma *vintersection-intersection*: $A \cap_\circ B = set\ (elts\ A \cap elts\ B)$
unfolding *inf-V-def* **by** *simp*

lemma *vintersection-assoc*: $A \cap_\circ (B \cap_\circ C) = (A \cap_\circ B) \cap_\circ C$ **by** *auto*

lemma *vintersection-commute*: $A \cap_\circ B = B \cap_\circ A$ **by** *auto*

Previous set operations.

lemma *vsubset-vintersection-right*: $A \subseteq_\circ (B \cap_\circ C) = (A \subseteq_\circ B \wedge A \subseteq_\circ C)$

by *clarsimp*

lemma *vsubset-vintersection-rightD*[*dest*]:
assumes $A \subseteq_\circ (B \cap_\circ C)$
shows $A \subseteq_\circ B$ **and** $A \subseteq_\circ C$
using *assms* **by** *auto*

lemma *vsubset-vintersection-rightI*[*intro*]:
assumes $A \subseteq_\circ B$ **and** $A \subseteq_\circ C$
shows $A \subseteq_\circ (B \cap_\circ C)$
using *assms* **by** *auto*

Set operations.

lemma *vintersection-vempty*: $0 \subseteq_\circ A$ **by** *simp*

lemma *vintersection-vsingleton*: $a \in_\circ$ *set* $\{a\}$ **by** *simp*

lemma *vintersection-vdoubleton*: $a \in_\circ$ *set* $\{a, b\}$ **and** $b \in_\circ$ *set* $\{a, b\}$
by *simp-all*

lemma *vintersection-VPow*[*simp*]: *VPow* $(A \cap_\circ B) =$ *VPow* $A \cap_\circ$ *VPow* B **by** *auto*

Special properties.

lemma *vintersection-function-mono*:
assumes *mono* f
shows $f\ (A \cap_\circ B) \subseteq_\circ f\ A \cap_\circ f\ B$
using *assms* **by** (*fact mono-inf*)

2.2.9 Binary union

lemma *small-vunion-set*: *small* $\{x.\ x \in_\circ A \vee x \in_\circ B\}$
by (*rule down*[*of* - ‹$A \cup_\circ B$›]) (*auto simp*: *elts-sup-iff*)

Rules.

lemma *vunion-def*: $A \cup_\circ B =$ *set* $\{x.\ x \in_\circ A \vee x \in_\circ B\}$
unfolding *Un-def sup-V-def* **by** *simp*

lemma *vunion-iff*[*simp*]: $x \in_\circ A \cup_\circ B \longleftrightarrow x \in_\circ A \vee x \in_\circ B$
by (*simp add*: *elts-sup-iff*)

lemma *vunionI*1:
assumes $a \in_\circ A$
shows $a \in_\circ A \cup_\circ B$
using *assms* **by** *simp*

lemma *vunionI*2:
assumes $a \in_\circ B$
shows $a \in_\circ A \cup_\circ B$
using *assms* **by** *simp*

lemma *vunionCI*[*intro*!]:
assumes $x \notin_\circ B \Longrightarrow x \in_\circ A$
shows $x \in_\circ A \cup_\circ B$
using *assms* **by** *clarsimp*

lemma *vunionE*[*elim*!]:
assumes $x \in_\circ A \cup_\circ B$ **and** $x \in_\circ A \Longrightarrow P$ **and** $x \in_\circ B \Longrightarrow P$
shows P

using *assms* **by** *auto*

Elementary properties.

lemma *vunion-union*: $A \cup_\circ B = set\ (elts\ A \cup elts\ B)$ **by** *auto*

lemma *vunion-assoc*: $A \cup_\circ (B \cup_\circ C) = (A \cup_\circ B) \cup_\circ C$ **by** *auto*

lemma *vunion-commute*: $A \cup_\circ B = B \cup_\circ A$ **by** *auto*

Previous set operations.

lemma *vsubset-vunion-left*: $(A \cup_\circ B) \subseteq_\circ C \longleftrightarrow (A \subseteq_\circ C \wedge B \subseteq_\circ C)$ **by** *simp*

lemma *vsubset-vunion-leftD*[*dest*]:
assumes $(A \cup_\circ B) \subseteq_\circ C$
shows $A \subseteq_\circ C$ **and** $B \subseteq_\circ C$
using *assms* **by** *auto*

lemma *vsubset-vunion-leftI*[*intro*]:
assumes $A \subseteq_\circ C$ **and** $B \subseteq_\circ C$
shows $(A \cup_\circ B) \subseteq_\circ C$
using *assms* **by** *auto*

lemma *vintersection-vunion-left*: $(A \cup_\circ B) \cap_\circ C = (A \cap_\circ C) \cup_\circ (B \cap_\circ C)$
by *auto*

lemma *vintersection-vunion-right*: $A \cap_\circ (B \cup_\circ C) = (A \cap_\circ B) \cup_\circ (A \cap_\circ C)$
by *auto*

Set operations.

lemmas *vunion-vempty-left* = *sup-V-0-left*
and *vunion-vempty-right* = *sup-V-0-right*

lemma *vunion-vsingleton*[*simp*]: $set\ \{a\} \cup_\circ A = vinsert\ a\ A$ **by** *auto*

lemma *vunion-vdoubleton*[*simp*]: $set\ \{a, b\} \cup_\circ A = vinsert\ a\ (vinsert\ b\ A)$
by *auto*

lemma *vunion-vinsert-comm-left*:
$(vinsert\ a\ A) \cup_\circ B = A \cup_\circ (vinsert\ a\ B)$
by *auto*

lemma *vunion-vinsert-comm-right*:
$A \cup_\circ (vinsert\ a\ B) = (vinsert\ a\ A) \cup_\circ B$
by *auto*

lemma *vinsert-def*: $vinsert\ y\ B = set\ \{x.\ x = y\} \cup_\circ B$ **by** *auto*

lemma *vunion-vinsert-left*: $(vinsert\ a\ A) \cup_\circ B = vinsert\ a\ (A \cup_\circ B)$ **by** *auto*

lemma *vunion-vinsert-right*: $A \cup_\circ (vinsert\ a\ B) = vinsert\ a\ (A \cup_\circ B)$ **by** *auto*

Special properties.

lemma *vunion-fun-mono*:
assumes *mono f*
shows $f\ A \cup_\circ f\ B \subseteq_\circ f\ (A \cup_\circ B)$
using *assms* **by** (*fact mono-sup*)

2.2.10 Set difference

definition *vdiff* :: $V \Rightarrow V \Rightarrow V$ (**infixl** $\langle -_\circ \rangle$ 65)
  where $A -_\circ B = set\ \{x.\ x \in_\circ A \wedge x \notin_\circ B\}$
notation *vdiff* (**infixl** $\langle -_\circ \rangle$ 65)

lemma *small-set-vdiff*[*simp*]: *small* $\{x.\ x \in_\circ A \wedge x \notin_\circ B\}$
  by (*rule down*[*of* - *A*]) *simp*

Rules.

lemma *vdiff-iff*[*simp*]: $x \in_\circ A -_\circ B \longleftrightarrow x \in_\circ A \wedge x \notin_\circ B$
  unfolding *vdiff-def* **by** *simp*

lemma *vdiffI*[*intro*!]:
  assumes $x \in_\circ A$ **and** $x \notin_\circ B$
  shows $x \in_\circ A -_\circ B$
  using *assms* **by** *simp*

lemma *vdiffD*1:
  assumes $x \in_\circ A -_\circ B$
  shows $x \in_\circ A$
  using *assms* **by** *simp*

lemma *vdiffD*2:
  assumes $x \in_\circ A -_\circ B$ **and** $x \in_\circ B$
  shows P
  using *assms* **by** *simp*

lemma *vdiffE*[*elim*!]:
  assumes $x \in_\circ A -_\circ B$ **and** $[\![\ x \in_\circ A;\ x \notin_\circ B\ ]\!] \Longrightarrow P$
  shows P
  using *assms* **by** *simp*

Previous set operations.

lemma *vsubset-vdiff*:
  assumes $A \subseteq_\circ B -_\circ C$
  shows $A \subseteq_\circ B$
  using *assms* **by** *auto*

lemma *vinsert-vdiff-nin*[*simp*]:
  assumes $a \notin_\circ B$
  shows $vinsert\ a\ (A -_\circ B) = vinsert\ a\ A -_\circ B$
  using *assms* **by** *auto*

Set operations.

lemma *vdiff-vempty-left*[*simp*]: $0 -_\circ A = 0$ **by** *auto*

lemma *vdiff-vempty-right*[*simp*]: $A -_\circ 0 = A$ **by** *auto*

lemma *vdiff-vsingleton-vinsert*[*simp*]: $set\ \{a\} -_\circ vinsert\ a\ A = 0$ **by** *auto*

lemma *vdiff-vsingleton-in*[*simp*]:
  assumes $a \in_\circ A$
  shows $set\ \{a\} -_\circ A = 0$
  using *assms* **by** *auto*

lemma *vdiff-vsingleton-nin*[*simp*]:
  assumes $a \notin_\circ A$

shows *set* $\{a\} -_\circ A = set\ \{a\}$
using *assms* **by** *auto*

lemma *vdiff-vinsert-vsingleton*[*simp*]: *vinsert a A* $-_\circ$ *set* $\{a\} = A -_\circ set\ \{a\}$
by *auto*

lemma *vdiff-vsingleton*[*simp*]:
assumes $a \notin_\circ A$
shows $A -_\circ set\ \{a\} = A$
using *assms* **by** *auto*

lemma *vdiff-vsubset*:
assumes $A \subseteq_\circ B$ **and** $D \subseteq_\circ C$
shows $A -_\circ C \subseteq_\circ B -_\circ D$
using *assms* **by** *auto*

lemma *vdiff-vinsert-left-in*[*simp*]:
assumes $a \in_\circ B$
shows (*vinsert a A*) $-_\circ B = A -_\circ B$
using *assms* **by** *auto*

lemma *vdiff-vinsert-left-nin*:
assumes $a \notin_\circ B$
shows (*vinsert a A*) $-_\circ B$ = *vinsert a* $(A -_\circ B)$
using *assms* **by** *auto*

lemma *vdiff-vinsert-right-in*: $A -_\circ$ (*vinsert a B*) = $A -_\circ B -_\circ$ *set* $\{a\}$ **by** *auto*

lemma *vdiff-vinsert-right-nin*[*simp*]:
assumes $a \notin_\circ A$
shows $A -_\circ$ (*vinsert a B*) = $A -_\circ B$
using *assms* **by** *auto*

lemma *vdiff-vintersection-left*: $(A \cap_\circ B) -_\circ C = (A -_\circ C) \cap_\circ (B -_\circ C)$ **by** *auto*

lemma *vdiff-vunion-left*: $(A \cup_\circ B) -_\circ C = (A -_\circ C) \cup_\circ (B -_\circ C)$ **by** *auto*

Special properties.

lemma *complement-vsubset*: $I -_\circ J \subseteq_\circ I$ **by** *auto*

lemma *vintersection-complement*: $(I -_\circ J) \cap_\circ J = 0$ **by** *auto*

lemma *vunion-complement*:
assumes $J \subseteq_\circ I$
shows $I -_\circ J \cup_\circ J = I$
using *assms* **by** *auto*

2.2.11 Augmenting a set with an element

lemma *vinsert-compr*: *vinsert y A* = *set* $\{x.\ x = y \vee x \in_\circ A\}$
unfolding *vunion-def vinsert-def* **by** *simp-all*

Rules.

lemma *vinsert-iff*[*simp*]: $x \in_\circ$ *vinsert y A* $\longleftrightarrow x = y \vee x \in_\circ A$ **by** *simp*

lemma *vinsertI1*: $x \in_\circ$ *vinsert x A* **by** *simp*

lemma *vinsertI2*:

assumes $x \in_\circ A$
shows $x \in_\circ$ *vinsert* y A
using *assms* **by** *simp*

lemma *vinsertE*1[*elim*!]:
assumes $x \in_\circ$ *vinsert* y A **and** $x = y \Longrightarrow P$ **and** $x \in_\circ A \Longrightarrow P$
shows P
using *assms* **unfolding** *vinsert-def* **by** *auto*

lemma *vinsertCI*[*intro*!]:
assumes $x \notin_\circ A \Longrightarrow x = y$
shows $x \in_\circ$ *vinsert* y A
using *assms* **by** *clarsimp*

Elementary properties.

lemma *vinsert-insert*: *vinsert* a A = *set* (*insert* a (*elts* A)) **by** *auto*

lemma *vinsert-commute*: *vinsert* a (*vinsert* b C) = *vinsert* b (*vinsert* a C)
by *auto*

lemma *vinsert-ident*:
assumes $x \notin_\circ A$ **and** $x \notin_\circ B$
shows *vinsert* x A = *vinsert* x $B \longleftrightarrow A = B$
using *assms* **by** *force*

lemmas *vinsert-identD*[*dest*] = *vinsert-ident*[*THEN iffD*1, *rotated* 2]
and *vinsert-identI*[*intro*] = *vinsert-ident*[*THEN iffD*2]

Set operations.

lemma *vinsert-vempty*: *vinsert* a 0 = *set* $\{a\}$ **by** *auto*

lemma *vinsert-vsingleton*: *vinsert* a (*set* $\{b\}$) = *set* $\{a, b\}$ **by** *auto*

lemma *vinsert-vdoubleton*: *vinsert* a (*set* $\{b, c\}$) = *set* $\{a, b, c\}$ **by** *auto*

lemma *vinsert-vinsert*: *vinsert* a (*vinsert* b C) = *set* $\{a, b\} \cup_\circ C$ **by** *auto*

lemma *vinsert-vunion-left*: *vinsert* a ($A \cup_\circ B$) = *vinsert* a $A \cup_\circ B$ **by** *auto*

lemma *vinsert-vunion-right*: *vinsert* a ($A \cup_\circ B$) = $A \cup_\circ$ *vinsert* a B **by** *auto*

lemma *vinsert-vintersection*: *vinsert* a ($A \cap_\circ B$) = *vinsert* a $A \cap_\circ$ *vinsert* a B
by *auto*

Special properties.

lemma *vinsert-set-insert-empty-anyI*:
assumes P (*vinsert* a 0)
shows P (*set* (*insert* a $\{\}$))
using *assms* **by** (*simp add*: *vinsert-def*)

lemma *vinsert-set-insert-anyI*:
assumes *small* B **and** P (*vinsert* a (*set* (*insert* b B)))
shows P (*set* (*insert* a (*insert* b B)))
using *assms* **by** (*simp add*: *ZFC-in-HOL.vinsert-def*)

lemma *vinsert-set-insert-eq*:
assumes *small* B

shows *set* (*insert a* (*insert b B*)) = *vinsert a* (*set* (*insert b B*))
using *assms* **by** (*simp add*: *ZFC-in-HOL.vinsert-def*)

lemma *vsubset-vinsert*:
$A \subseteq_\circ$ *vinsert x B* $\longleftrightarrow$ (*if* $x \in_\circ A$ *then* $A -_\circ$ *set* $\{x\} \subseteq_\circ B$ *else* $A \subseteq_\circ B$)
by *auto*

lemma *vinsert-obtain-ne*:
assumes $A \neq 0$
obtains $a\ A'$ **where** A = *vinsert a* A' **and** $a \notin_\circ A'$
proof–
from *assms mem-not-refl* **obtain** a **where** $a \in_\circ A$
by (*auto intro*!: *vsubset-antisym*)
with ‹$a \in_\circ A$› **have** A = *vinsert a* ($A -_\circ$ *set* $\{a\}$) **by** *auto*
then show *?thesis* **using** *that* **by** *auto*
qed

2.2.12 Power set

Rules.

lemma *VPowI*:
assumes $A \subseteq_\circ B$
shows $A \in_\circ$ *VPow B*
using *assms* **by** *simp*

lemma *VPowD*:
assumes $A \in_\circ$ *VPow B*
shows $A \subseteq_\circ B$
using *assms* **by** (*simp add*: *Pow-def*)

lemma *VPowE*[*elim*]:
assumes $A \in_\circ$ *VPow B* **and** $A \subseteq_\circ B \Longrightarrow P$
shows *P*
using *assms* **by** *auto*

Elementary properties.

lemma *VPow-bottom*: $0 \in_\circ$ *VPow B* **by** *simp*

lemma *VPow-top*: $A \in_\circ$ *VPow A* **by** *simp*

Set operations.

lemma *VPow-vempty*[*simp*]: *VPow* 0 = *set* $\{0\}$ **by** *auto*

lemma *VPow-vsingleton*[*simp*]: *VPow* (*set* $\{a\}$) = *set* $\{0,$ *set* $\{a\}\}$
by (*rule vsubset-antisym*; *rule vsubsetI*) *auto*

lemma *VPow-not-vempty*: *VPow A* $\neq 0$ **by** *auto*

lemma *VPow-mono*:
assumes $A \subseteq_\circ B$
shows *VPow A* $\subseteq_\circ$ *VPow B*
using *assms* **by** *simp*

lemma *VPow-vunion-subset*: *VPow A* $\cup_\circ$ *VPow B* $\subseteq_\circ$ *VPow* ($A \cup_\circ B$) **by** *simp*

2.2.13 Singletons, using insert

Rules.

lemma *vsingletonI*[*intro*!]: $x \in_\circ set\ \{x\}$ **by** *auto*

lemma *vsingletonD*[*dest*!]:
assumes $y \in_\circ set\ \{x\}$
shows $y = x$
using *assms* **by** *auto*

lemma *vsingleton-iff*: $y \in_\circ set\ \{x\} \longleftrightarrow y = x$ **by** *simp*

Previous set operations.

lemma *VPow-vdoubleton*[*simp*]:
$VPow\ (set\ \{a, b\}) = set\ \{0, set\ \{a\}, set\ \{b\}, set\ \{a, b\}\}$
by (*intro vsubset-antisym vsubsetI*)
(*auto intro*!: *vsubset-antisym simp*: *vinsert-set-insert-eq*)

lemma *vsubset-vinsertI*:
assumes $A -_\circ set\ \{x\} \subseteq_\circ B$
shows $A \subseteq_\circ vinsert\ x\ B$
using *assms* **by** *auto*

Special properties.

lemma *vsingleton-inject*:
assumes $set\ \{x\} = set\ \{y\}$
shows $x = y$
using *assms* **by** *simp*

lemma *vsingleton-insert-inj-eq*[*iff*]:
$set\ \{y\} = vinsert\ x\ A \longleftrightarrow x = y \wedge A \subseteq_\circ set\ \{y\}$
by *auto*

lemma *vsingleton-insert-inj-eq'*[*iff*]:
$vinsert\ x\ A = set\ \{y\} \longleftrightarrow x = y \wedge A \subseteq_\circ set\ \{y\}$
by *auto*

lemma *vsubset-vsingletonD*:
assumes $A \subseteq_\circ set\ \{x\}$
shows $A = 0 \vee A = set\ \{x\}$
using *assms* **by** *auto*

lemma *vsubset-vsingleton-iff*: $a \subseteq_\circ set\ \{x\} \longleftrightarrow a = 0 \vee a = set\ \{x\}$ **by** *auto*

lemma *vsubset-vdiff-vinsert*: $A \subseteq_\circ B -_\circ vinsert\ x\ C \longleftrightarrow A \subseteq_\circ B -_\circ C \wedge x \notin_\circ A$
by *auto*

lemma *vunion-vsingleton-iff*:
$A \cup_\circ B = set\ \{x\} \longleftrightarrow$
$A = 0 \wedge B = set\ \{x\} \vee A = set\ \{x\} \wedge B = 0 \vee A = set\ \{x\} \wedge B = set\ \{x\}$
by
(
metis
vsubset-vsingletonD inf-sup-ord(4) *sup.idem sup-V*-0-*right sup-commute*
)

lemma *vsingleton-Un-iff*:

set $\{x\} = A \cup_\circ B \longleftrightarrow$
$A = 0 \wedge B = set\ \{x\} \vee A = set\ \{x\} \wedge B = 0 \vee A = set\ \{x\} \wedge B = set\ \{x\}$
by (*metis vunion-vsingleton-iff sup-V-0-left sup-V-0-right sup-idem*)

lemma *VPow-vsingleton-iff*[*simp*]: $VPow\ X = set\ \{Y\} \longleftrightarrow X = 0 \wedge Y = 0$
by (*auto intro*!: *vsubset-antisym*)

2.2.14 Intersection of elements

lemma *small-VInter*[*simp*]:
assumes $A \neq 0$
shows $small\ \{a.\ \forall\, x \in_\circ A.\ a \in_\circ x\}$
by (*metis* (*no-types*, *lifting*) *assms down eq0-iff mem-Collect-eq subsetI*)

lemma *VInter-def*: $\bigcap_\circ A = (if\ A = 0\ then\ 0\ else\ set\ \{a.\ \forall\, x \in_\circ A.\ a \in_\circ x\})$
proof(*cases* ‹$A = 0$›)
case *True* **show** *?thesis* **unfolding** *True Inf-V-def* **by** *simp*
next
case *False*
from *False* **have** $(\bigcap\ (elts\ `\ elts\ A)) = \{a.\ \forall\, x \in_\circ A.\ a \in_\circ x\}$ **by** *auto*
with *False* **show** *?thesis* **unfolding** *Inf-V-def* **by** *auto*
qed

Rules.

lemma *VInter-iff*[*simp*]:
assumes [*simp*]: $A \neq 0$
shows $a \in_\circ \bigcap_\circ A \longleftrightarrow (\forall\, x \in_\circ A.\ a \in_\circ x)$
unfolding *VInter-def* **by** *auto*

lemma *VInterI*[*intro*]:
assumes $A \neq 0$ **and** $\bigwedge x.\ x \in_\circ A \Longrightarrow a \in_\circ x$
shows $a \in_\circ \bigcap_\circ A$
using *assms* **by** *auto*

lemma *VInter0I*[*intro*]:
assumes $A = 0$
shows $\bigcap_\circ A = 0$
using *assms* **unfolding** *VInter-def* **by** *simp*

lemma *VInterD*[*dest*]:
assumes $a \in_\circ \bigcap_\circ A$ **and** $x \in_\circ A$
shows $a \in_\circ x$
using *assms* **by** (*cases* ‹$A = 0$›) *auto*

lemma *VInterE1*[*elim*]:
assumes $a \in_\circ \bigcap_\circ A$ **and** $x \notin_\circ A \Longrightarrow R$ **and** $a \in_\circ x \Longrightarrow R$
shows R
using *assms elts-0* **unfolding** *Inter-eq* **by** *blast*

lemma *VInterE2*[*elim*]:
assumes $a \in_\circ \bigcap_\circ A$
obtains x **where** $a \in_\circ x$ **and** $x \in_\circ A$
proof(*cases* ‹$A = 0$›)
show $(\bigwedge x.\ a \in_\circ x \Longrightarrow x \in_\circ A \Longrightarrow thesis) \Longrightarrow A = 0 \Longrightarrow thesis$
using *assms* **unfolding** *Inf-V-def* **by** *auto*
show $(\bigwedge x.\ a \in_\circ x \Longrightarrow x \in_\circ A \Longrightarrow thesis) \Longrightarrow A \neq 0 \Longrightarrow thesis$
using *assms* **by** (*meson assms VInterE1 that trad-foundation*)
qed

lemma *VInterE3*:
assumes $a \in_\circ \bigcap_\circ A$ **and** $(\bigwedge y.\ y \in_\circ A \Longrightarrow a \in_\circ y) \Longrightarrow P$
shows P
using *assms* **by** *auto*

Elementary properties.

lemma *VInter-Inter*: $\bigcap_\circ A = set\ (\bigcap\ (elts\ `\ (elts\ A)))$
by (*simp add*: *Inf-V-def ext*)

lemma *VInter-eq*:
assumes [*simp*]: $A \neq 0$
shows $\bigcap_\circ A = set\ \{a.\ \forall x \in_\circ A.\ a \in_\circ x\}$
unfolding *VInter-def* **by** *auto*

Set operations.

lemma *VInter-vempty*[*simp*]: $\bigcap_\circ 0 = 0$ **using** *VInter0I* **by** *auto*

lemma *VInf-vempty*[*simp*]: $\bigsqcap\{\} = (0::V)$ **by** (*simp add*: *Inf-V-def*)

lemma *VInter-vdoubleton*: $\bigcap_\circ (set\ \{a, b\}) = a \cap_\circ b$
proof(*intro vsubset-antisym vsubsetI*)
show $x \in_\circ \bigcap_\circ (set\ \{a, b\}) \Longrightarrow x \in_\circ a \cap_\circ b$ **for** x **by** (*elim VInterE3*) *auto*
show $x \in_\circ a \cap_\circ b \Longrightarrow x \in_\circ \bigcap_\circ (set\ \{a, b\})$ **for** x **by** (*intro VInterI*) *force+*
qed

lemma *VInter-antimono*:
assumes $B \neq 0$ **and** $B \subseteq_\circ A$
shows $\bigcap_\circ A \subseteq_\circ \bigcap_\circ B$
using *assms* **by** *blast*

lemma *VInter-vsubset*:
assumes $\bigwedge x.\ x \in_\circ A \Longrightarrow x \subseteq_\circ B$ **and** $A \neq 0$
shows $\bigcap_\circ A \subseteq_\circ B$
using *assms* **by** *auto*

lemma *VInter-vinsert*:
assumes $A \neq 0$
shows $\bigcap_\circ (vinsert\ a\ A) = a \cap_\circ \bigcap_\circ A$
using *assms* **by** (*blast intro*!: *vsubset-antisym*)

lemma *VInter-vunion*:
assumes $A \neq 0$ **and** $B \neq 0$
shows $\bigcap_\circ(A \cup_\circ B) = \bigcap_\circ A \cap_\circ \bigcap_\circ B$
using *assms* **by** (*blast intro*!: *vsubset-antisym*)

lemma *VInter-vintersection*:
assumes $A \cap_\circ B \neq 0$
shows $\bigcap_\circ A \cup_\circ \bigcap_\circ B \subseteq_\circ \bigcap_\circ (A \cap_\circ B)$
using *assms* **by** *auto*

lemma *VInter-VPow*: $\bigcap_\circ (VPow\ A) \subseteq_\circ VPow\ (\bigcap_\circ A)$ **by** *auto*

Elementary properties.

lemma *VInter-lower*:
assumes $x \in_\circ A$
shows $\bigcap_\circ A \subseteq_\circ x$

using *assms* **by** *auto*

lemma *VInter-greatest*:
assumes $A \neq 0$ **and** $\bigwedge x.\ x \in_\circ A \Longrightarrow B \subseteq_\circ x$
shows $B \subseteq_\circ \bigcap_\circ A$
using *assms* **by** *auto*

2.2.15 Union of elements

lemma *Union-eq-VUnion*: $\bigcup(elts\ `\ elts\ A) = \{a.\ \exists x \in_\circ A.\ a \in_\circ x\}$ **by** *auto*

lemma *small-VUnion*[*simp*]: *small* $\{a.\ \exists x \in_\circ A.\ a \in_\circ x\}$
by (*fold Union-eq-VUnion*) *simp*

lemma *VUnion-def*: $\bigcup_\circ A = set\ \{a.\ \exists x \in_\circ A.\ a \in_\circ x\}$
unfolding *Sup-V-def* **by** *auto*

Rules.

lemma *VUnion-iff*[*simp*]: $A \in_\circ \bigcup_\circ C \longleftrightarrow (\exists x \in_\circ C.\ A \in_\circ x)$ **by** *auto*

lemma *VUnionI*[*intro*]:
assumes $x \in_\circ A$ **and** $a \in_\circ x$
shows $a \in_\circ \bigcup_\circ A$
using *assms* **by** *auto*

lemma *VUnionE*[*elim*!]:
assumes $a \in_\circ \bigcup_\circ A$ **and** $\bigwedge x.\ a \in_\circ x \Longrightarrow x \in_\circ A \Longrightarrow R$
shows R
using *assms* **by** *clarsimp*

Elementary properties.

lemma *VUnion-Union*: $\bigcup_\circ A = set\ (\bigcup\ (elts\ `\ (elts\ A)))$
by (*simp add*: *Inf-V-def ext*)

Set operations.

lemma *VUnion-vempty*[*simp*]: $\bigcup_\circ 0 = 0$ **by** *simp*

lemma *VUnion-vsingleton*[*simp*]: $\bigcup_\circ(set\ \{a\}) = a$ **by** *simp*

lemma *VUnion-vdoubleton*[*simp*]: $\bigcup_\circ(set\ \{a,\ b\}) = a \cup_\circ b$ **by** *auto*

lemma *VUnion-mono*:
assumes $A \subseteq_\circ B$
shows $\bigcup_\circ A \subseteq_\circ \bigcup_\circ B$
using *assms* **by** *auto*

lemma *VUnion-vinsert*: $\bigcup_\circ(vinsert\ x\ A) = x \cup_\circ \bigcup_\circ A$ **by** *auto*

lemma *VUnion-vintersection*: $\bigcup_\circ(A \cap_\circ B) \subseteq_\circ \bigcup_\circ A \cap_\circ \bigcup_\circ B$ **by** *auto*

lemma *VUnion-vunion*[*simp*]: $\bigcup_\circ(A \cup_\circ B) = \bigcup_\circ A \cup_\circ \bigcup_\circ B$ **by** *auto*

lemma *VUnion-VPow*[*simp*]: $\bigcup_\circ(VPow\ A) = A$ **by** *auto*

Special properties.

lemma *VUnion-vempty-conv-left*: $0 = \bigcup_\circ A \longleftrightarrow (\forall x \in_\circ A.\ x = 0)$ **by** *auto*

lemma *VUnion-vempty-conv-right*: $\bigcup_\circ A = 0 \longleftrightarrow (\forall\, x \in_\circ A.\ x = 0)$ **by** *auto*

lemma *vsubset-VPow-VUnion*: $A \subseteq_\circ VPow\ (\bigcup_\circ A)$ **by** *auto*

lemma *VUnion-vsubsetI*:
assumes $\bigwedge x.\ x \in_\circ A \Longrightarrow \exists\, y.\ y \in_\circ B \wedge x \subseteq_\circ y$
shows $\bigcup_\circ A \subseteq_\circ \bigcup_\circ B$
using *assms* **by** *auto*

lemma *VUnion-upper*:
assumes $x \in_\circ A$
shows $x \subseteq_\circ \bigcup_\circ A$
using *assms* **by** *auto*

lemma *VUnion-least*:
assumes $\bigwedge x.\ x \in_\circ A \Longrightarrow x \subseteq_\circ B$
shows $\bigcup_\circ A \subseteq_\circ B$
using *assms* **by** (*fact Sup-least*)

2.2.16 Pairs

Further results

lemma *small-elts-of-set*[*simp*, *intro*]:
assumes *small x*
shows *elts* (*set x*) = *x*
by (*simp add*: *assms*)

lemma *small-vpair*[*intro*, *simp*]:
assumes *small* {*a*. *P a*}
shows *small* {⟨*a*, *b*⟩ | *a*. *P a*}
by (*subgoal-tac* ‹{⟨*a*, *b*⟩ | *a*. *P a*} = (λ*a*. ⟨*a*, *b*⟩) ‘ {*a*. *P a*}›)
(*auto simp*: *assms*)

vpairs

definition *vpairs* :: $V \Rightarrow V$ **where**
vpairs r = *set* {*x*. *x* ∈∘ *r* ∧ (∃ *a b*. *x* = ⟨*a*, *b*⟩)}

lemma *small-vpairs*[*simp*]: *small* {⟨*a*, *b*⟩ | *a b*. ⟨*a*, *b*⟩ ∈∘ *r*}
by (*rule down*[*of* - *r*]) *clarsimp*

Rules.

lemma *vpairsI*[*intro*]:
assumes *x* ∈∘ *r* **and** *x* = ⟨*a*, *b*⟩
shows *x* ∈∘ *vpairs r*
using *assms* **unfolding** *vpairs-def* **by** *auto*

lemma *vpairsD*[*dest*]:
assumes *x* ∈∘ *vpairs r*
shows *x* ∈∘ *r* **and** ∃ *a b*. *x* = ⟨*a*, *b*⟩
using *assms* **unfolding** *vpairs-def* **by** *auto*

lemma *vpairsE*[*elim*]:
assumes *x* ∈∘ *vpairs r*
obtains *a b* **where** *x* = ⟨*a*, *b*⟩ **and** ⟨*a*, *b*⟩ ∈∘ *r*
using *assms* **unfolding** *vpairs-def* **by** *auto*

lemma *vpairs-iff*: *x* ∈∘ *vpairs r* ⟷ *x* ∈∘ *r* ∧ (∃ *a b*. *x* = ⟨*a*, *b*⟩) **by** *auto*

Elementary properties.

lemma *vpairs-iff-elts*: $\langle a,\ b\rangle \in_\circ \mathit{vpairs}\ r \longleftrightarrow \langle a,\ b\rangle \in_\circ r$ **by** *auto*

lemma *vpairs-iff-pairs*: $\langle a,\ b\rangle \in_\circ \mathit{vpairs}\ r \longleftrightarrow (a,\ b) \in \mathit{pairs}\ r$
by (*simp add*: *vpairs-iff-elts pairs-iff-elts*)

Set operations.

lemma *vpairs-vempty*[*simp*]: $\mathit{vpairs}\ 0 = 0$ **by** *auto*

lemma *vpairs-vsingleton*[*simp*]: $\mathit{vpairs}\ (\mathit{set}\ \{\langle a,\ b\rangle\}) = \mathit{set}\ \{\langle a,\ b\rangle\}$ **by** *auto*

lemma *vpairs-vinsert*: $\mathit{vpairs}\ (\mathit{vinsert}\ \langle a,\ b\rangle\ A) = \mathit{set}\ \{\langle a,\ b\rangle\} \cup_\circ \mathit{vpairs}\ A$
by *auto*

lemma *vpairs-mono*:
assumes $r \subseteq_\circ s$
shows $\mathit{vpairs}\ r \subseteq_\circ \mathit{vpairs}\ s$
using *assms* **by** *blast*

lemma *vpairs-vunion*: $\mathit{vpairs}\ (A \cup_\circ B) = \mathit{vpairs}\ A \cup_\circ \mathit{vpairs}\ B$ **by** *auto*

lemma *vpairs-vintersection*: $\mathit{vpairs}\ (A \cap_\circ B) = \mathit{vpairs}\ A \cap_\circ \mathit{vpairs}\ B$ **by** *auto*

lemma *vpairs-vdiff*: $\mathit{vpairs}\ (A -_\circ B) = \mathit{vpairs}\ A -_\circ \mathit{vpairs}\ B$ **by** *auto*

Special properties.

lemma *vpairs-ex-vfst*:
assumes $x \in_\circ \mathit{vpairs}\ r$
shows $\exists\, b.\ \langle \mathit{vfst}\ x,\ b\rangle \in_\circ r$
using *assms* **by** *force*

lemma *vpairs-ex-vsnd*:
assumes $y \in_\circ \mathit{vpairs}\ r$
shows $\exists\, a.\ \langle a,\ \mathit{vsnd}\ y\rangle \in_\circ r$
using *assms* **by** *force*

2.2.17 Cartesian products

The following lemma is based on Theorem 6.2 from [59].

lemma *vtimes-vsubset-VPowVPow*: $A \times_\circ B \subseteq_\circ \mathit{VPow}\ (\mathit{VPow}\ (A \cup_\circ B))$
proof(*intro vsubsetI*)
fix x **assume** $x \in_\circ A \times_\circ B$
then obtain $a\ b$ **where** *x-def*: $x = \langle a,\ b\rangle$ **and** $a \in_\circ A$ **and** $b \in_\circ B$ **by** *clarsimp*
then show $x \in_\circ \mathit{VPow}\ (\mathit{VPow}\ (A \cup_\circ B))$
unfolding *x-def vpair-def* **by** *auto*
qed

2.2.18 Pairwise

definition $\mathit{vpairwise} :: (V \Rightarrow V \Rightarrow \mathit{bool}) \Rightarrow V \Rightarrow \mathit{bool}$
where $\mathit{vpairwise}\ R\ S \longleftrightarrow (\forall\, x{\in_\circ}S.\ \forall\, y{\in_\circ}S.\ x \neq y \longrightarrow R\ x\ y)$

Rules.

lemma *vpairwiseI*[*intro?*]:
assumes $\bigwedge x\ y.\ x \in_\circ S \Longrightarrow y \in_\circ S \Longrightarrow x \neq y \Longrightarrow R\ x\ y$
shows $\mathit{vpairwise}\ R\ S$

using *assms* **by** (*simp add*: *vpairwise-def*)

lemma *vpairwiseD*[*dest*]:
assumes *vpairwise R S* **and** $x \in_\circ S$ **and** $y \in_\circ S$ **and** $x \neq y$
shows *R x y* **and** *R y x*
using *assms* **unfolding** *vpairwise-def* **by** *auto*

Elementary properties.

lemma *vpairwise-trivial*[*simp*]: *vpairwise* $(\lambda i\ j.\ j \neq i)$ *I*
by (*auto simp*: *vpairwise-def*)

Set operations.

lemma *vpairwise-vempty*[*simp*]: *vpairwise P* 0 **by** (*force intro*: *vpairwiseI*)

lemma *vpairwise-vsingleton*[*simp*]: *vpairwise P* (*set* $\{A\}$)
by (*simp add*: *vpairwise-def*)

lemma *vpairwise-vinsert*:
vpairwise r (*vinsert x s*) $\longleftrightarrow$
$(\forall\, y.\ y \in_\circ s \wedge y \neq x \longrightarrow r\ x\ y \wedge r\ y\ x) \wedge$ *vpairwise r s*
by (*intro iffI conjI allI impI*; (*elim conjE* | *tactic*‹*all-tac*›))
(*auto simp*: *vpairwise-def*)

lemma *vpairwise-vsubset*:
assumes *vpairwise P S* **and** $T \subseteq_\circ S$
shows *vpairwise P T*
using *assms* **by** (*metis less-eq-V-def subset-eq vpairwiseD*(2) *vpairwiseI*)

lemma *vpairwise-mono*:
assumes *vpairwise P A* **and** $\bigwedge x\ y.\ P\ x\ y \Longrightarrow Q\ x\ y$ **and** $B \subseteq_\circ A$
shows *vpairwise Q B*
using *assms* **by** (*simp add*: *less-eq-V-def subset-eq vpairwiseD*(2) *vpairwiseI*)

2.2.19 Disjoint sets

abbreviation *vdisjnt* :: $V \Rightarrow V \Rightarrow$ *bool*
where *vdisjnt A B* $\equiv A \cap_\circ B = 0$

Elementary properties.

lemma *vdisjnt-sym*:
assumes *vdisjnt A B*
shows *vdisjnt B A*
using *assms* **by** *blast*

lemma *vdisjnt-iff*: *vdisjnt A B* $\longleftrightarrow$ $(\forall\, x.\ \sim (x \in_\circ A \wedge x \in_\circ B))$ **by** *auto*

Set operations.

lemma *vdisjnt-vempty*1[*simp*]: *vdisjnt* 0 *A*
and *vdisjnt-vempty*2[*simp*]: *vdisjnt A* 0
by *auto*

lemma *vdisjnt-singleton*0[*simp*]: *vdisjnt* (*set* $\{a\}$) (*set* $\{b\}$) $\longleftrightarrow a \neq b$
and *vdisjnt-singleton*1[*simp*]: *vdisjnt* (*set* $\{a\}$) *A* $\longleftrightarrow a \notin_\circ A$
and *vdisjnt-singleton*2[*simp*]: *vdisjnt A* (*set* $\{a\}$) $\longleftrightarrow a \notin_\circ A$
by *force+*

lemma *vdisjnt-vinsert-left*: *vdisjnt* (*vinsert a X*) *Y* $\longleftrightarrow a \notin_\circ Y \wedge$ *vdisjnt X Y*

by (*metis vdisjnt-iff vdisjnt-sym vinsertE*1 *vinsertI*2 *vinsert-iff*)

lemma *vdisjnt-vinsert-right*: *vdisjnt Y* (*vinsert a X*) ⟷ *a* ∉∘ *Y* ∧ *vdisjnt Y X*
using *vdisjnt-sym vdisjnt-vinsert-left* **by** *meson*

lemma *vdisjnt-vsubset-left*:
assumes *vdisjnt X Y* **and** *Z* ⊆∘ *X*
shows *vdisjnt Z Y*
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vdisjnt-vsubset-right*:
assumes *vdisjnt X Y* **and** *Z* ⊆∘ *Y*
shows *vdisjnt X Z*
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vdisjnt-vunion-left*: *vdisjnt* (*A* ∪∘ *B*) *C* ⟷ *vdisjnt A C* ∧ *vdisjnt B C*
by *auto*

lemma *vdisjnt-vunion-right*: *vdisjnt C* (*A* ∪∘ *B*) ⟷ *vdisjnt C A* ∧ *vdisjnt C B*
by *auto*

Special properties.

lemma *vdisjnt-vemptyI*[*intro*]:
assumes ⋀*x*. *x* ∈∘ *A* ⟹ *x* ∈∘ *B* ⟹ *False*
shows *vdisjnt A B*
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vdisjnt-self-iff-vempty*[*simp*]: *vdisjnt S S* ⟷ *S* = 0 **by** *auto*

lemma *vdisjntI*:
assumes ⋀*x y*. *x* ∈∘ *A* ⟹ *y* ∈∘ *B* ⟹ *x* ≠ *y*
shows *vdisjnt A B*
using *assms* **by** *auto*

lemma *vdisjnt-nin-right*:
assumes *vdisjnt A B* **and** *a* ∈∘ *A*
shows *a* ∉∘ *B*
using *assms* **by** *auto*

lemma *vdisjnt-nin-left*:
assumes *vdisjnt B A* **and** *a* ∈∘ *A*
shows *a* ∉∘ *B*
using *assms* **by** *auto*

2.3 Further properties of natural numbers

2.3.1 Background

The section exposes certain fundamental properties of natural numbers and provides convenience utilities for doing arithmetic within the type V.

Many of the results that are presented in this sections were carried over (with amendments) from the theory *Nat* that can be found in the main library of Isabelle/HOL.

notation *ord-of-nat* ($\langle$-$_{\mathbb{N}}\rangle$ [999] 999)

named-theorems *nat-omega-simps*

declare *One-nat-def*[*simp del*]

abbreviation (*input*) *vpfst* **where** *vpfst a* ≡ $a(\!|0|\!)$
abbreviation (*input*) *vpsnd* **where** *vpsnd a* ≡ $a(\!|1_{\mathbb{N}}|\!)$
abbreviation (*input*) *vpthrd* **where** *vpthrd a* ≡ $a(\!|2_{\mathbb{N}}|\!)$

2.3.2 Conversion between V and *nat*

Primitive arithmetic

lemma *ord-of-nat-plus*[*nat-omega-simps*]: $a_{\mathbb{N}} + b_{\mathbb{N}} = (a + b)_{\mathbb{N}}$
by (*induct b*) (*simp-all add*: *plus-V-succ-right*)

lemma *ord-of-nat-times*[*nat-omega-simps*]: $a_{\mathbb{N}} * b_{\mathbb{N}} = (a * b)_{\mathbb{N}}$
by (*induct b*) (*simp-all add*: *mult-succ nat-omega-simps*)

lemma *ord-of-nat-succ*[*nat-omega-simps*]: *succ* ($a_{\mathbb{N}}$) = (*Suc a*)$_{\mathbb{N}}$ **by** *auto*

lemmas [*nat-omega-simps*] = *nat-cadd-eq-add*

lemma *ord-of-nat-csucc*[*nat-omega-simps*]: *csucc* ($a_{\mathbb{N}}$) = *succ* ($a_{\mathbb{N}}$)
using *finite-csucc* **by** *blast*

lemma *ord-of-nat-succ-vempty*[*nat-omega-simps*]: *succ* 0 = $1_{\mathbb{N}}$ **by** *auto*

lemma *ord-of-nat-vone*[*nat-omega-simps*]: 1 = $1_{\mathbb{N}}$ **by** *auto*

Transfer

definition *cr-omega* :: $V \Rightarrow nat \Rightarrow bool$
where *cr-omega a b* ⟷ (*a* = *ord-of-nat b*)

Transfer setup.

lemma *cr-omega-right-total*[*transfer-rule*]: *right-total cr-omega*
unfolding *cr-omega-def right-total-def* **by** *simp*

lemma *cr-omega-bi-unqie*[*transfer-rule*]: *bi-unique cr-omega*
unfolding *cr-omega-def bi-unique-def*
by (*simp add*: *inj-eq inj-ord-of-nat*)

lemma *omega-transfer-domain-rule*[*transfer-domain-rule*]:
Domainp cr-omega = ($\lambda x.\ x \in_{\circ} \omega$)
unfolding *cr-omega-def* **by** (*auto simp*: *elts-ω*)

lemma *omega-transfer*[*transfer-rule*]:
(*rel-set cr-omega*) (*elts ω*) (*UNIV*::*nat set*)

unfolding *cr-omega-def rel-set-def* **by** (*simp add*: *elts-ω*)

lemma *omega-of-real-transfer*[*transfer-rule*]: *cr-omega* (*ord-of-nat a*) *a*
unfolding *cr-omega-def* **by** *auto*

Operations.

lemma *omega-succ-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-omega* ===> *cr-omega*) *succ Suc*
proof(*intro rel-funI*, *unfold cr-omega-def*)
fix *x y* **assume** *prems*: $x = y_{\mathbb{N}}$
show *succ x* = *Suc* $y_{\mathbb{N}}$ **unfolding** *prems ord-of-nat-succ*[*symmetric*] **..**
qed

lemma *omega-plus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-omega* ===> *cr-omega* ===> *cr-omega*) (+) (+)
by (*intro rel-funI*, *unfold cr-omega-def*) (*simp add*: *nat-omega-simps*)

lemma *omega-mult-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-omega* ===> *cr-omega* ===> *cr-omega*) (*) (*)
by (*intro rel-funI*, *unfold cr-omega-def*) (*simp add*: *nat-omega-simps*)

lemma *ord-of-nat-card-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*rel-set* (=) ===> *cr-omega*) (λ*x*. *ord-of-nat* (*card x*)) *card*
by (*intro rel-funI*) (*simp add*: *cr-omega-def rel-set-eq*)

lemma *ord-of-nat-transfer*[*transfer-rule*]:
(*rel-fun cr-omega* (=)) *id ord-of-nat*
unfolding *cr-omega-def* **by** *auto*

2.3.3 Elementary results

lemma *ord-of-nat-vempty*: $0 = 0_{\mathbb{N}}$ **by** *auto*

lemma *set-vzero-eq-ord-of-nat-vone*: *set* {0} = $1_{\mathbb{N}}$
by (*metis elts*-1 *set-of-elts ord-of-nat-vone*)

lemma *vone-in-omega*[*simp*]: 1 ∈$_\circ$ *ω* **unfolding** *ω-def* **by** *force*

lemma *nat-of-omega*:
assumes *n* ∈$_\circ$ *ω*
obtains *m* **where** $n = m_{\mathbb{N}}$
using *assms* **unfolding** *ω-def* **by** *clarsimp*

lemma *omega-prev*:
assumes *n* ∈$_\circ$ *ω* **and** 0 ∈$_\circ$ *n*
obtains *k* **where** *n* = *succ k*
proof–
from *assms nat-of-omega* **obtain** *m* **where** $n = m_{\mathbb{N}}$ **by** *auto*
with *assms*(2) **obtain** *m′* **where** *m* = *Suc m′*
unfolding *less-V-def* **by** (*auto dest*: *gr*0-*implies-Suc*)
with *that* **show** *?thesis* **unfolding** ‹$n = m_{\mathbb{N}}$› **using** *ord-of-nat.simps*(2) **by** *blast*
qed

lemma *omega-vplus-commutative*:

assumes $a \in_\circ \omega$ **and** $b \in_\circ \omega$
shows $a + b = b + a$
using *assms* **by** (*metis Groups.add-ac*(2) *nat-of-omega ord-of-nat-plus*)

lemma *omega-vinetrsection*[*intro*]:
assumes $m \in_\circ \omega$ **and** $n \in_\circ \omega$
shows $m \cap_\circ n \in_\circ \omega$
proof–
from *nat-into-Ord*[*OF assms*(1)] *nat-into-Ord*[*OF assms*(2)] *Ord-linear-le*
consider $m \subseteq_\circ n \mid n \subseteq_\circ m$
by *auto*
then show *?thesis* **by** *cases* (*simp-all add*: *assms inf.absorb*1 *inf.absorb*2)
qed

2.3.4 Induction

lemma *omega-induct-all*[*consumes* 1, *case-names step*]:
assumes $n \in_\circ \omega$ **and** $\bigwedge x.\ [\![x \in_\circ \omega;\ \bigwedge y.\ y \in_\circ x \Longrightarrow P\ y]\!] \Longrightarrow P\ x$
shows $P\ n$
using *assms* **by** (*metis Ord-ω Ord-induct Ord-linear Ord-trans nat-into-Ord*)

lemma *omega-induct*[*consumes* 1, *case-names* 0 *succ*]:
assumes $n \in_\circ \omega$ **and** $P\ 0$ **and** $\bigwedge n.\ [\![\ n \in_\circ \omega;\ P\ n\ ]\!] \Longrightarrow P\ (succ\ n)$
shows $P\ n$
using *assms*(1,3)
proof(*induct rule*: *omega-induct-all*)
case (*step x*) **show** *?case*
proof(*cases* ‹$x = 0$›)
case *True* **with** *assms*(2) **show** *?thesis* **by** *simp*
next
case *False*
with *step*(1) **have** $0 \in_\circ x$ **by** (*simp add*: *mem-*0*-Ord*)
with ‹$x \in_\circ \omega$› **obtain** y **where** *x-def*: $x = succ\ y$ **by** (*elim omega-prev*)
with *elts-succ step.hyps*(1) **have** $y \in_\circ \omega$ **by** (*blast intro*: *Ord-trans*)
have $y \in_\circ x$ **by** (*simp add*: ‹$x = succ\ y$›)
have $P\ y$ **by** (*auto intro*: *step.prems step.hyps*(2)[*OF* ‹$y \in_\circ x$›])
from *step.prems*[*OF* ‹$y \in_\circ \omega$› ‹$P\ y$›, *folded x-def*] **show** $P\ x$ **.**
qed
qed

2.3.5 Methods

The following methods provide an infrastructure for working with goals of the form $a \in_\circ n_{\mathbf{N}} \Longrightarrow P\ a$.

lemma *in-succE*:
assumes $a \in_\circ succ\ n$ **and** $\bigwedge a.\ a \in_\circ n \Longrightarrow P\ a$ **and** $P\ n$
shows $P\ a$
using *assms* **by** *auto*

method *Suc-of-numeral* =
(
unfold numeral.simps add.assoc,
use nothing **in** ‹*unfold Suc-eq-plus*1*-left*[*symmetric*], *unfold One-nat-def*›
)

method *succ-of-numeral* =
(

Suc-of-numeral,
use nothing **in** ‹*unfold ord-of-nat-succ*[*symmetric*] *ord-of-nat-zero*›
)

method *numeral-of-succ* =
(
unfold nat-omega-simps,
use nothing **in**
‹
unfold numeral.simps[*symmetric*] *Suc-numeral add-num-simps*,
(*unfold numerals*(1))*?*
›
)

method *elim-in-succ* =
(
(
elim in-succE;
use nothing **in** ‹(*unfold triv-forall-equality*)*?*; (*numeral-of-succ*)*?*›
),
simp
)

method *elim-in-numeral* = (*succ-of-numeral*, *use nothing* **in** ‹*elim-in-succ*›)

2.3.6 Auxiliary

lemma *one*: $1_{\mathbb{N}}$ = *set* {0} **by** *auto*

lemma *two*: $2_{\mathbb{N}}$ = *set* {0, $1_{\mathbb{N}}$} **by** *force*

lemma *three*: $3_{\mathbb{N}}$ = *set* {0, $1_{\mathbb{N}}$, $2_{\mathbb{N}}$} **by** *force*

lemma *four*: $4_{\mathbb{N}}$ = *set* {0, $1_{\mathbb{N}}$, $2_{\mathbb{N}}$, $3_{\mathbb{N}}$} **by** *force*

lemma *two-vdiff-zero*[*simp*]: *set* {0, $1_{\mathbb{N}}$} −∘ *set* {0} = *set* {$1_{\mathbb{N}}$} **by** *auto*
lemma *two-vdiff-one*[*simp*]: *set* {0, $1_{\mathbb{N}}$} −∘ *set* {$1_{\mathbb{N}}$} = *set* {0} **by** *auto*

2.4 Elementary binary relations

2.4.1 Background

This section presents a theory of binary relations internalized in the type V and exposes elementary properties of two special types of binary relations: single-valued binary relations and injective single-valued binary relations.

Many of the results that are presented in this section were carried over (with amendments) from the theories *Set* and *Relation* in the main library.

2.4.2 Constructors

Identity relation

definition *vid-on* :: $V \Rightarrow V$
  where *vid-on* A = *set* $\{\langle a, a\rangle \mid a.\ a \in_\circ A\}$

lemma *vid-on-small*[*simp*]: *small* $\{\langle a, a\rangle \mid a.\ a \in_\circ A\}$
  by (*rule down*[*of* - ‹$A \times_\circ A$›]) *blast*

Rules.

lemma *vid-on-eqI*:
  assumes $a = b$ **and** $a \in_\circ A$
  shows $\langle a, b\rangle \in_\circ$ *vid-on* A
  using *assms* **by** (*simp add*: *vid-on-def*)

lemma *vid-onI*[*intro*!]:
  assumes $a \in_\circ A$
  shows $\langle a, a\rangle \in_\circ$ *vid-on* A
  by (*rule vid-on-eqI*) (*simp-all add*: *assms*)

lemma *vid-onD*[*dest*!]:
  assumes $\langle a, a\rangle \in_\circ$ *vid-on* A
  shows $a \in_\circ A$
  using *assms* **unfolding** *vid-on-def* **by** *auto*

lemma *vid-onE*[*elim*!]:
  assumes $x \in_\circ$ *vid-on* A **and** $\exists a{\in_\circ}A.\ x = \langle a, a\rangle \Longrightarrow P$
  shows P
  using *assms* **unfolding** *vid-on-def* **by** *auto*

lemma *vid-on-iff*: $\langle a, b\rangle \in_\circ$ *vid-on* $A \longleftrightarrow a = b \wedge a \in_\circ A$ **by** *auto*

Set operations.

lemma *vid-on-vempty*[*simp*]: *vid-on* $0 = 0$ **by** *auto*

lemma *vid-on-vsingleton*[*simp*]: *vid-on* (*set* $\{a\}$) = *set* $\{\langle a, a\rangle\}$ **by** *auto*

lemma *vid-on-vdoubleton*[*simp*]: *vid-on* (*set* $\{a, b\}$) = *set* $\{\langle a, a\rangle, \langle b, b\rangle\}$
  by (*auto simp*: *vinsert-set-insert-eq*)

lemma *vid-on-mono*:
  assumes $A \subseteq_\circ B$
  shows *vid-on* $A \subseteq_\circ$ *vid-on* B
  using *assms* **by** *auto*

lemma *vid-on-vinsert*: (*vinsert* $\langle a, a\rangle$ (*vid-on* A)) = (*vid-on* (*vinsert* a A))

by *auto*

lemma *vid-on-vintersection*: *vid-on* $(A \cap_\circ B)$ = *vid-on* $A \cap_\circ$ *vid-on* B **by** *auto*

lemma *vid-on-vunion*: *vid-on* $(A \cup_\circ B)$ = *vid-on* $A \cup_\circ$ *vid-on* B **by** *auto*

lemma *vid-on-vdiff*: *vid-on* $(A -_\circ B)$ = *vid-on* $A -_\circ$ *vid-on* B **by** *auto*

Special properties.

lemma *vid-on-vsubset-vtimes*: *vid-on* $A \subseteq_\circ A \times_\circ A$ **by** *clarsimp*

lemma *VLambda-id*[*simp*]: *VLambda* A *id* = *vid-on* A
by (*simp add*: *id-def vid-on-def Setcompr-eq-image VLambda-def*)

Constant function

definition *vconst-on* :: $V \Rightarrow V \Rightarrow V$
where *vconst-on* A c = *set* $\{\langle a, c\rangle \mid a.\ a \in_\circ A\}$

lemma *small-vconst-on*[*simp*]: *small* $\{\langle a, c\rangle \mid a.\ a \in_\circ A\}$
by (*rule down*[*of* - ‹$A \times_\circ$ *set* $\{c\}$›]) *auto*

Rules.

lemma *vconst-onI*[*intro*!]:
assumes $a \in_\circ A$
shows $\langle a, c\rangle \in_\circ$ *vconst-on* A c
using *assms* **unfolding** *vconst-on-def* **by** *simp*

lemma *vconst-onD*[*dest*!]:
assumes $\langle a, c\rangle \in_\circ$ *vconst-on* A c
shows $a \in_\circ A$
using *assms* **unfolding** *vconst-on-def* **by** *simp*

lemma *vconst-onE*[*elim*!]:
assumes $x \in_\circ$ *vconst-on* A c
obtains a **where** $a \in_\circ A$ **and** $x = \langle a, c\rangle$
using *assms* **unfolding** *vconst-on-def* **by** *auto*

lemma *vconst-on-iff*: $\langle a, c\rangle \in_\circ$ *vconst-on* A $c \longleftrightarrow a \in_\circ A$ **by** *auto*

Set operations.

lemma *vconst-on-vempty*[*simp*]: *vconst-on* 0 c = 0
unfolding *vconst-on-def* **by** *auto*

lemma *vconst-on-vsingleton*[*simp*]: *vconst-on* (*set* $\{a\}$) c = *set* $\{\langle a, c\rangle\}$ **by** *auto*

lemma *vconst-on-vdoubleton*[*simp*]: *vconst-on* (*set* $\{a, b\}$) c = *set* $\{\langle a, c\rangle, \langle b, c\rangle\}$
by (*auto simp*: *vinsert-set-insert-eq*)

lemma *vconst-on-mono*:
assumes $A \subseteq_\circ B$
shows *vconst-on* A $c \subseteq_\circ$ *vconst-on* B c
using *assms* **by** *auto*

lemma *vconst-on-vinsert*:
(*vinsert* $\langle a, c\rangle$ (*vconst-on* A c)) = (*vconst-on* (*vinsert* a A) c)
by *auto*

lemma *vconst-on-vintersection*:
vconst-on $(A \cap_\circ B)$ c = *vconst-on* A $c \cap_\circ$ *vconst-on* B c
by *auto*

lemma *vconst-on-vunion*: *vconst-on* $(A \cup_\circ B)$ c = *vconst-on* A $c \cup_\circ$ *vconst-on* B c
by *auto*

lemma *vconst-on-vdiff*: *vconst-on* $(A -_\circ B)$ c = *vconst-on* A $c -_\circ$ *vconst-on* B c
by *auto*

Special properties.

lemma *vconst-on-eq-vtimes*: *vconst-on* A c = $A \times_\circ$ *set* $\{c\}$
by *standard* (*auto intro*!: *vsubset-antisym*)

VLambda

Rules.

lemma *VLambdaI*[*intro*!]:
assumes $a \in_\circ A$
shows $\langle a, f\ a\rangle \in_\circ (\lambda a \in_\circ A.\ f\ a)$
using *assms* **unfolding** *VLambda-def* **by** *auto*

lemma *VLambdaD*[*dest*!]:
assumes $\langle a, f\ a\rangle \in_\circ (\lambda a \in_\circ A.\ f\ a)$
shows $a \in_\circ A$
using *assms* **unfolding** *VLambda-def* **by** *auto*

lemma *VLambdaE*[*elim*!]:
assumes $x \in_\circ (\lambda a \in_\circ A.\ f\ a)$
obtains a **where** $a \in_\circ A$ **and** $x = \langle a, f\ a\rangle$
using *assms* **unfolding** *VLambda-def* **by** *auto*

lemma *VLambda-iff*1: $x \in_\circ (\lambda a \in_\circ A.\ f\ a) \longleftrightarrow (\exists\, a \in_\circ A.\ x = \langle a, f\ a\rangle)$ **by** *auto*

lemma *VLambda-iff*2: $\langle a, b\rangle \in_\circ (\lambda a \in_\circ A.\ f\ a) \longleftrightarrow b = f\ a \wedge a \in_\circ A$ **by** *auto*

lemma *small-VLambda*[*simp*]: *small* $\{\langle a, f\ a\rangle \mid a.\ a \in_\circ A\}$ **by** *auto*

lemma *VLambda-set-def*: $(\lambda a \in_\circ A.\ f\ a)$ = *set* $\{\langle a, f\ a\rangle \mid a.\ a \in_\circ A\}$ **by** *auto*

Set operations.

lemma *VLambda-vempty*[*simp*]: $(\lambda a \in_\circ 0.\ f\ a) = 0$ **by** *auto*

lemma *VLambda-vsingleton*: $(\lambda a \in_\circ$ *set* $\{a\}.\ f\ a)$ = *set* $\{\langle a, f\ a\rangle\}$
by *auto*

lemma *VLambda-vdoubleton*:
$(\lambda a \in_\circ$ *set* $\{a, b\}.\ f\ a)$ = *set* $\{\langle a, f\ a\rangle, \langle b, f\ b\rangle\}$
by (*auto simp*: *vinsert-set-insert-eq*)

lemma *VLambda-mono*:
assumes $A \subseteq_\circ B$
shows $(\lambda a \in_\circ A.\ f\ a) \subseteq_\circ (\lambda a \in_\circ B.\ f\ a)$
using *assms* **by** *auto*

lemma *VLambda-vinsert*:
$(\lambda a \in_\circ$ *vinsert* a $A.\ f\ a) = (\lambda a \in_\circ$ *set* $\{a\}.\ f\ a) \cup_\circ (\lambda a \in_\circ A.\ f\ a)$

by *auto*

lemma *VLambda-vintersection*: $(\lambda a \in_\circ A \cap_\circ B.\ f\ a) = (\lambda a \in_\circ A.\ f\ a) \cap_\circ (\lambda a \in_\circ B.\ f\ a)$
by *auto*

lemma *VLambda-vunion*: $(\lambda a \in_\circ A \cup_\circ B.\ f\ a) = (\lambda a \in_\circ A.\ f\ a) \cup_\circ (\lambda a \in_\circ B.\ f\ a)$ **by** *auto*

lemma *VLambda-vdiff*: $(\lambda a \in_\circ A -_\circ B.\ f\ a) = (\lambda a \in_\circ A.\ f\ a) -_\circ (\lambda a \in_\circ B.\ f\ a)$ **by** *auto*

Connections.

lemma *VLambda-vid-on*: $(\lambda a \in_\circ A.\ a) = vid\text{-}on\ A$ **by** *auto*

lemma *VLambda-vconst-on*: $(\lambda a \in_\circ A.\ c) = vconst\text{-}on\ A\ c$ **by** *auto*

Composition

definition *vcomp* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\circ_\circ\rangle$ 75)
where $r \circ_\circ s = set\ \{\langle a, c\rangle \mid a\ c.\ \exists b.\ \langle a, b\rangle \in_\circ s \wedge \langle b, c\rangle \in_\circ r\}$
notation *vcomp* (**infixr** $\langle\circ_\circ\rangle$ 75)

lemma *vcomp-small*[*simp*]: *small* $\{\langle a, c\rangle \mid a\ c.\ \exists b.\ \langle a, b\rangle \in_\circ s \wedge \langle b, c\rangle \in_\circ r\}$
(**is** $\langle small\ ?s\rangle$)
proof–
define *comp'* **where** $comp' = (\lambda\langle\langle a, b\rangle, \langle c, d\rangle\rangle.\ \langle a, d\rangle)$
have *small* (*elts* (*vpairs* ($s \times_\circ r$))) **by** *simp*
then have *small-comp*: *small* (*comp'* ‘ *elts* (*vpairs* ($s \times_\circ r$))) **by** *simp*
have *ss*: $?s \subseteq$ (*comp'* ‘ *elts* (*vpairs* ($s \times_\circ r$)))
proof
fix *x* **assume** $x \in ?s$
then obtain *a b c* **where** *x-def*: $x = \langle a, c\rangle$
and $\langle a, b\rangle \in_\circ s$
and $\langle b, c\rangle \in_\circ r$
by *auto*
then have *abbc*: $\langle\langle a, b\rangle, \langle b, c\rangle\rangle \in_\circ vpairs\ (s \times_\circ r)$
by (*simp add*: *vpairs-iff-elts*)
have *x-def'*: $x = comp'\ \langle\langle a, b\rangle, \langle b, c\rangle\rangle$ **unfolding** *comp'-def x-def* **by** *auto*
then show $x \in$ *comp'* ‘ *elts* (*vpairs* ($s \times_\circ r$))
unfolding *x-def'* **using** *abbc* **by** *auto*
qed
with *small-comp* **show** *?thesis* **by** (*metis* (*lifting*) *smaller-than-small*)
qed

Rules.

lemma *vcompI*[*intro*!]:
assumes $\langle b, c\rangle \in_\circ r$ **and** $\langle a, b\rangle \in_\circ s$
shows $\langle a, c\rangle \in_\circ r \circ_\circ s$
using *assms* **unfolding** *vcomp-def* **by** *auto*

lemma *vcompD*[*dest*!]:
assumes $\langle a, c\rangle \in_\circ r \circ_\circ s$
shows $\exists b.\ \langle b, c\rangle \in_\circ r \wedge \langle a, b\rangle \in_\circ s$
using *assms* **unfolding** *vcomp-def* **by** *auto*

lemma *vcompE*[*elim*!]:
assumes $ac \in_\circ r \circ_\circ s$
obtains *a b c* **where** $ac = \langle a, c\rangle$ **and** $\langle a, b\rangle \in_\circ s$ **and** $\langle b, c\rangle \in_\circ r$
using *assms* **unfolding** *vcomp-def* **by** *clarsimp*

Elementary properties.

lemma *vcomp-assoc*: $(r \circ_\circ s) \circ_\circ t = r \circ_\circ (s \circ_\circ t)$ **by** *auto*

Set operations.

lemma *vcomp-vempty-left*[*simp*]: $0 \circ_\circ r = 0$ **by** *auto*

lemma *vcomp-vempty-right*[*simp*]: $r \circ_\circ 0 = 0$ **by** *auto*

lemma *vcomp-mono*:
assumes $r' \subseteq_\circ r$ **and** $s' \subseteq_\circ s$
shows $r' \circ_\circ s' \subseteq_\circ r \circ_\circ s$
using *assms* **by** *auto*

lemma *vcomp-vinsert-left*[*simp*]:
$(\mathit{vinsert}\ \langle a,\ b\rangle\ s) \circ_\circ r = (\mathit{set}\ \{\langle a,\ b\rangle\} \circ_\circ r) \cup_\circ (s \circ_\circ r)$
by *auto*

lemma *vcomp-vinsert-right*[*simp*]:
$r \circ_\circ (\mathit{vinsert}\ \langle a,\ b\rangle\ s) = (r \circ_\circ \mathit{set}\ \{\langle a,\ b\rangle\}) \cup_\circ (r \circ_\circ s)$
by *auto*

lemma *vcomp-vunion-left*[*simp*]: $(s \cup_\circ t) \circ_\circ r = (s \circ_\circ r) \cup_\circ (t \circ_\circ r)$ **by** *auto*

lemma *vcomp-vunion-right*[*simp*]: $r \circ_\circ (s \cup_\circ t) = (r \circ_\circ s) \cup_\circ (r \circ_\circ t)$ **by** *auto*

Connections.

lemma *vcomp-vid-on-idem*[*simp*]: *vid-on A* $\circ_\circ$ *vid-on A* = *vid-on A* **by** *auto*

lemma *vcomp-vid-on*[*simp*]: *vid-on A* $\circ_\circ$ *vid-on B* = *vid-on* $(A \cap_\circ B)$ **by** *auto*

lemma *vcomp-vconst-on-vid-on*[*simp*]: *vconst-on A c* $\circ_\circ$ *vid-on A* = *vconst-on A c*
by *auto*

lemma *vcomp-VLambda-vid-on*[*simp*]: $(\lambda a{\in_\circ} A.\ f\ a) \circ_\circ \mathit{vid\text{-}on}\ A = (\lambda a{\in_\circ} A.\ f\ a)$
by *auto*

Special properties.

lemma *vcomp-vsubset-vtimes*:
assumes $r \subseteq_\circ B \times_\circ C$ **and** $s \subseteq_\circ A \times_\circ B$
shows $r \circ_\circ s \subseteq_\circ A \times_\circ C$
using *assms* **by** *auto*

lemma *vcomp-obtain-middle*[*elim*]:
assumes $\langle a,\ c\rangle \in_\circ r \circ_\circ s$
obtains b **where** $\langle a,\ b\rangle \in_\circ s$ **and** $\langle b,\ c\rangle \in_\circ r$
using *assms* **by** *auto*

Converse relation

definition *vconverse* :: $V \Rightarrow V$
where $\mathit{vconverse}\ A = (\lambda r{\in_\circ} A.\ \mathit{set}\ \{\langle b,\ a\rangle \mid a\ b.\ \langle a,\ b\rangle \in_\circ r\})$

abbreviation *app-vconverse* $(\langle(\text{-}^{-1}{}_\circ)\rangle\ [1000]\ 999)$
where $r^{-1}{}_\circ \equiv \mathit{vconverse}\ (\mathit{set}\ \{r\})\ (\lfloor r \rfloor)$

lemma *app-vconverse-def*: $r^{-1}{}_\circ = \mathit{set}\ \{\langle b,\ a\rangle \mid a\ b.\ \langle a,\ b\rangle \in_\circ r\}$
unfolding *vconverse-def* **by** *simp*

lemma *vconverse-small*[*simp*]: *small* $\{\langle b,\ a\rangle \mid a\ b.\ \langle a,\ b\rangle \in_\circ r\}$
proof–
have *eq*: $\{\langle b,\ a\rangle \mid a\ b.\ \langle a,\ b\rangle \in_\circ r\} = (\lambda\langle a,\ b\rangle.\ \langle b,\ a\rangle)\ `\ elts\ (vpairs\ r)$
proof(*rule subset-antisym*; *rule subsetI*, *unfold mem-Collect-eq*)
fix *x* **assume** $x \in (\lambda\langle a,\ b\rangle.\ \langle b,\ a\rangle)\ `\ elts\ (vpairs\ r)$
then obtain *a b* **where** $\langle a,\ b\rangle \in_\circ vpairs\ r$ **and** $x = (\lambda\langle a,\ b\rangle.\ \langle b,\ a\rangle)\ \langle a,\ b\rangle$
by *blast*
then show $\exists\, a\ b.\ x = \langle b,\ a\rangle \wedge \langle a,\ b\rangle \in_\circ r$ **by** *auto*
qed (*use image-iff vpairs-iff-elts* **in** *fastforce*)
show *?thesis* **unfolding** *eq* **by** (*rule replacement*) *auto*
qed

Rules.

lemma *vconverseI*[*intro*!]:
assumes $r \in_\circ A$
shows $\langle r,\ r^{-1}{}_\circ\rangle \in_\circ vconverse\ A$
using *assms* **unfolding** *vconverse-def* **by** *auto*

lemma *vconverseD*[*dest*]:
assumes $\langle r,\ s\rangle \in_\circ vconverse\ A$
shows $r \in_\circ A$ **and** $s = r^{-1}{}_\circ$
using *assms* **unfolding** *vconverse-def* **by** *auto*

lemma *vconverseE*[*elim*]:
assumes $x \in_\circ vconverse\ A$
obtains *r* **where** $x = \langle r,\ r^{-1}{}_\circ\rangle$ **and** $r \in_\circ A$
using *assms* **unfolding** *vconverse-def* **by** *auto*

lemma *app-vconverseI*[*sym*, *intro*!]:
assumes $\langle a,\ b\rangle \in_\circ r$
shows $\langle b,\ a\rangle \in_\circ r^{-1}{}_\circ$
using *assms* **unfolding** *vconverse-def* **by** *auto*

lemma *app-vconverseD*[*sym*, *dest*]:
assumes $\langle a,\ b\rangle \in_\circ r^{-1}{}_\circ$
shows $\langle b,\ a\rangle \in_\circ r$
using *assms* **unfolding** *vconverse-def* **by** *simp*

lemma *app-vconverseE*[*elim*!]:
assumes $x \in_\circ r^{-1}{}_\circ$
obtains *a b* **where** $x = \langle b,\ a\rangle$ **and** $\langle a,\ b\rangle \in_\circ r$
using *assms* **unfolding** *vconverse-def* **by** *auto*

lemma *vconverse-iff*: $\langle b,\ a\rangle \in_\circ r^{-1}{}_\circ \longleftrightarrow \langle a,\ b\rangle \in_\circ r$ **by** *auto*

Set operations.

lemma *vconverse-vempty*[*simp*]: $0^{-1}{}_\circ = 0$ **by** *auto*

lemma *vconverse-vsingleton*: $(set\ \{\langle a,\ b\rangle\})^{-1}{}_\circ = set\ \{\langle b,\ a\rangle\}$ **by** *auto*

lemma *vconverse-vdoubleton*[*simp*]: $(set\ \{\langle a,\ b\rangle,\ \langle c,\ d\rangle\})^{-1}{}_\circ = set\ \{\langle b,\ a\rangle,\ \langle d,\ c\rangle\}$
by (*auto simp*: *vinsert-set-insert-eq*)

lemma *vconverse-vinsert*: $(vinsert\ \langle a,\ b\rangle\ r)^{-1}{}_\circ = vinsert\ \langle b,\ a\rangle\ (r^{-1}{}_\circ)$ **by** *auto*

lemma *vconverse-vintersection*: $(r \cap_\circ s)^{-1}{}_\circ = r^{-1}{}_\circ \cap_\circ s^{-1}{}_\circ$ **by** *auto*

lemma *vconverse-vunion*: $(r \cup_\circ s)^{-1}_\circ = r^{-1}_\circ \cup_\circ s^{-1}_\circ$ **by** *auto*

Connections.

lemma *vconverse-vid-on*[*simp*]: $(\textit{vid-on}\ A)^{-1}_\circ = \textit{vid-on}\ A$ **by** *auto*

lemma *vconverse-vconst-on*[*simp*]: $(\textit{vconst-on}\ A\ c)^{-1}_\circ = \textit{set}\ \{c\} \times_\circ A$ **by** *auto*

lemma *vconverse-vcomp*: $(r \circ_\circ s)^{-1}_\circ = s^{-1}_\circ \circ_\circ r^{-1}_\circ$ **by** *auto*

lemma *vconverse-vtimes*: $(A \times_\circ B)^{-1}_\circ = (B \times_\circ A)$ **by** *auto*

Left restriction

definition *vlrestriction* :: $V \Rightarrow V$
  where *vlrestriction D* =
    VLambda D $(\lambda\langle r,\ A\rangle.\ \textit{set}\ \{\langle a,\ b\rangle \mid a\ b.\ a \in_\circ A \wedge \langle a,\ b\rangle \in_\circ r\})$

abbreviation *app-vlrestriction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\restriction^l_\circ\rangle$ 80)
  where $r \restriction^l_\circ A \equiv \textit{vlrestriction}\ (\textit{set}\ \{\langle r,\ A\rangle\})\ (\!|\langle r,\ A\rangle|\!)$

lemma *app-vlrestriction-def*: $r \restriction^l_\circ A = \textit{set}\ \{\langle a,\ b\rangle \mid a\ b.\ a \in_\circ A \wedge \langle a,\ b\rangle \in_\circ r\}$
  unfolding *vlrestriction-def* **by** *simp*

lemma *vlrestriction-small*[*simp*]: *small* $\{\langle a,\ b\rangle \mid a\ b.\ a \in_\circ A \wedge \langle a,\ b\rangle \in_\circ r\}$
  by (*rule down*[*of* - r]) *auto*

Rules.

lemma *vlrestrictionI*[*intro*!]:
  assumes $\langle r,\ A\rangle \in_\circ D$
  shows $\langle\langle r,\ A\rangle,\ r \restriction^l_\circ A\rangle \in_\circ \textit{vlrestriction}\ D$
  using *assms* **unfolding** *vlrestriction-def* **by** (*simp add*: *VLambda-iff*2)

lemma *vlrestrictionD*[*dest*]:
  assumes $\langle\langle r,\ A\rangle,\ s\rangle \in_\circ \textit{vlrestriction}\ D$
  shows $\langle r,\ A\rangle \in_\circ D$ **and** $s = r \restriction^l_\circ A$
  using *assms* **unfolding** *vlrestriction-def* **by** *auto*

lemma *vlrestrictionE*[*elim*]:
  assumes $x \in_\circ \textit{vlrestriction}\ D$ **and** $D \subseteq_\circ R \times_\circ X$
  obtains $r\ A$ **where** $x = \langle\langle r,\ A\rangle,\ r \restriction^l_\circ A\rangle$ **and** $r \in_\circ R$ **and** $A \in_\circ X$
  using *assms* **unfolding** *vlrestriction-def* **by** *auto*

lemma *app-vlrestrictionI*[*intro*!]:
  assumes $a \in_\circ A$ **and** $\langle a,\ b\rangle \in_\circ r$
  shows $\langle a,\ b\rangle \in_\circ r \restriction^l_\circ A$
  using *assms* **unfolding** *vlrestriction-def* **by** *simp*

lemma *app-vlrestrictionD*[*dest*]:
  assumes $\langle a,\ b\rangle \in_\circ r \restriction^l_\circ A$
  shows $a \in_\circ A$ **and** $\langle a,\ b\rangle \in_\circ r$
  using *assms* **unfolding** *vlrestriction-def* **by** *auto*

lemma *app-vlrestrictionE*[*elim*]:
  assumes $x \in_\circ r \restriction^l_\circ A$
  obtains $a\ b$ **where** $x = \langle a,\ b\rangle$ **and** $a \in_\circ A$ **and** $\langle a,\ b\rangle \in_\circ r$
  using *assms* **unfolding** *vlrestriction-def* **by** *auto*

Set operations.

lemma *vlrestriction-on-vempty*[*simp*]: $r \restriction_{l\circ} 0 = 0$
by (*auto intro*!: *vsubset-antisym*)

lemma *vlrestriction-vempty*[*simp*]: $0 \restriction_{l\circ} A = 0$ **by** *auto*

lemma *vlrestriction-vsingleton-in*[*simp*]:
assumes $a \in_\circ A$
shows *set* $\{\langle a, b\rangle\} \restriction_{l\circ} A$ = *set* $\{\langle a, b\rangle\}$
using *assms* **by** *auto*

lemma *vlrestriction-vsingleton-nin*[*simp*]:
assumes $a \notin_\circ A$
shows *set* $\{\langle a, b\rangle\} \restriction_{l\circ} A = 0$
using *assms* **by** *auto*

lemma *vlrestriction-mono*:
assumes $A \subseteq_\circ B$
shows $r \restriction_{l\circ} A \subseteq_\circ r \restriction_{l\circ} B$
using *assms* **by** *auto*

lemma *vlrestriction-vinsert-nin*[*simp*]:
assumes $a \notin_\circ A$
shows (*vinsert* $\langle a, b\rangle$ r) $\restriction_{l\circ} A = r \restriction_{l\circ} A$
using *assms* **by** *auto*

lemma *vlrestriction-vinsert-in*:
assumes $a \in_\circ A$
shows (*vinsert* $\langle a, b\rangle$ r) $\restriction_{l\circ} A$ = *vinsert* $\langle a, b\rangle$ $(r \restriction_{l\circ} A)$
using *assms* **by** *auto*

lemma *vlrestriction-vintersection*: $(r \cap_\circ s) \restriction_{l\circ} A = r \restriction_{l\circ} A \cap_\circ s \restriction_{l\circ} A$ **by** *auto*

lemma *vlrestriction-vunion*: $(r \cup_\circ s) \restriction_{l\circ} A = r \restriction_{l\circ} A \cup_\circ s \restriction_{l\circ} A$ **by** *auto*

lemma *vlrestriction-vdiff*: $(r -_\circ s) \restriction_{l\circ} A = r \restriction_{l\circ} A -_\circ s \restriction_{l\circ} A$ **by** *auto*

Connections.

lemma *vlrestriction-vid-on*[*simp*]: (*vid-on* A) $\restriction_{l\circ} B$ = *vid-on* $(A \cap_\circ B)$ **by** *auto*

lemma *vlrestriction-vconst-on*: (*vconst-on* A c) $\restriction_{l\circ} B$ = (*vconst-on* B c) $\restriction_{l\circ} A$
by *auto*

lemma *vlrestriction-vconst-on-commute*:
assumes $x \in_\circ$ *vconst-on* A $c \restriction_{l\circ} B$
shows $x \in_\circ$ *vconst-on* B $c \restriction_{l\circ} A$
using *assms* **by** *auto*

lemma *vlrestriction-vcomp*[*simp*]: $(r \circ_\circ s) \restriction_{l\circ} A = r \circ_\circ (s \restriction_{l\circ} A)$ **by** *auto*

Previous connections.

lemma *vcomp-rel-vid-on*[*simp*]: $r \circ_\circ$ *vid-on* $A = r \restriction_{l\circ} A$ **by** *auto*

lemma *vcomp-vconst-on*:
$r \circ_\circ$ (*vconst-on* A c) = $(r \restriction_{l\circ}$ *set* $\{c\}) \circ_\circ$ (*vconst-on* A c)
by *auto*

Special properties.

lemma *vlrestriction-vsubset-vpairs*: $r \restriction_{l\circ} A \subseteq_\circ$ *vpairs* r

by (*rule vsubsetI*) *blast*

lemma *vlrestriction-vsubset-rel*: $r \upharpoonright^l_\circ A \subseteq_\circ r$ **by** *auto*

lemma *vlrestriction-VLambda*: $(\lambda a \in_\circ A.\ f\ a) \upharpoonright^l_\circ B = (\lambda a \in_\circ A \cap_\circ B.\ f\ a)$ **by** *auto*

Right restriction

definition *vrrestriction* :: $V \Rightarrow V$
where *vrrestriction D* =
VLambda D $(\lambda\langle r, A\rangle.\ set\ \{\langle a, b\rangle \mid a\ b.\ b \in_\circ A \wedge \langle a, b\rangle \in_\circ r\})$

abbreviation *app-vrrestriction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\upharpoonright^r_\circ\rangle$ 80)
where $r \upharpoonright^r_\circ A \equiv$ *vrrestriction* $(set\ \{\langle r, A\rangle\})$ $(\!|\langle r, A\rangle|\!)$

lemma *app-vrrestriction-def*: $r \upharpoonright^r_\circ A = set\ \{\langle a, b\rangle \mid a\ b.\ b \in_\circ A \wedge \langle a, b\rangle \in_\circ r\}$
unfolding *vrrestriction-def* **by** *simp*

lemma *vrrestriction-small*[*simp*]: *small* $\{\langle a, b\rangle \mid a\ b.\ b \in_\circ A \wedge \langle a, b\rangle \in_\circ r\}$
by (*rule down*[*of* - *r*]) *auto*

Rules.

lemma *vrrestrictionI*[*intro*!]:
assumes $\langle r, A\rangle \in_\circ D$
shows $\langle\langle r, A\rangle, r \upharpoonright^r_\circ A\rangle \in_\circ$ *vrrestriction D*
using *assms* **unfolding** *vrrestriction-def* **by** (*simp add*: *VLambda-iff*2)

lemma *vrrestrictionD*[*dest*]:
assumes $\langle\langle r, A\rangle, s\rangle \in_\circ$ *vrrestriction D*
shows $\langle r, A\rangle \in_\circ D$ **and** $s = r \upharpoonright^r_\circ A$
using *assms* **unfolding** *vrrestriction-def* **by** *auto*

lemma *vrrestrictionE*[*elim*]:
assumes $x \in_\circ$ *vrrestriction D* **and** $D \subseteq_\circ R \times_\circ X$
obtains *r A* **where** $x = \langle\langle r, A\rangle, r \upharpoonright^r_\circ A\rangle$ **and** $r \in_\circ R$ **and** $A \in_\circ X$
using *assms* **unfolding** *vrrestriction-def* **by** *auto*

lemma *app-vrrestrictionI*[*intro*!]:
assumes $b \in_\circ A$ **and** $\langle a, b\rangle \in_\circ r$
shows $\langle a, b\rangle \in_\circ r \upharpoonright^r_\circ A$
using *assms* **unfolding** *vrrestriction-def* **by** *simp*

lemma *app-vrrestrictionD*[*dest*]:
assumes $\langle a, b\rangle \in_\circ r \upharpoonright^r_\circ A$
shows $b \in_\circ A$ **and** $\langle a, b\rangle \in_\circ r$
using *assms* **unfolding** *vrrestriction-def* **by** *auto*

lemma *app-vrrestrictionE*[*elim*]:
assumes $x \in_\circ r \upharpoonright^r_\circ A$
obtains *a b* **where** $x = \langle a, b\rangle$ **and** $b \in_\circ A$ **and** $\langle a, b\rangle \in_\circ r$
using *assms* **unfolding** *vrrestriction-def* **by** *auto*

Set operations.

lemma *vrrestriction-on-vempty*[*simp*]: $r \upharpoonright^r_\circ 0 = 0$
by (*auto intro*!: *vsubset-antisym*)

lemma *vrrestriction-vempty*[*simp*]: $0 \upharpoonright^r_\circ A = 0$ **by** *auto*

lemma *vrrestriction-vsingleton-in*[*simp*]:
assumes $b \in_\circ A$
shows *set* $\{\langle a, b\rangle\} \restriction^r_\circ A$ = *set* $\{\langle a, b\rangle\}$
using *assms* **by** *auto*

lemma *vrrestriction-vsingleton-nin*[*simp*]:
assumes $b \notin_\circ A$
shows *set* $\{\langle a, b\rangle\} \restriction^r_\circ A = 0$
using *assms* **by** *auto*

lemma *vrrestriction-mono*:
assumes $A \subseteq_\circ B$
shows $r \restriction^r_\circ A \subseteq_\circ r \restriction^r_\circ B$
using *assms* **by** *auto*

lemma *vrrestriction-vinsert-nin*[*simp*]:
assumes $b \notin_\circ A$
shows (*vinsert* $\langle a, b\rangle$ r) $\restriction^r_\circ A = r \restriction^r_\circ A$
using *assms* **by** *auto*

lemma *vrrestriction-vinsert-in*:
assumes $b \in_\circ A$
shows (*vinsert* $\langle a, b\rangle$ r) $\restriction^r_\circ A$ = *vinsert* $\langle a, b\rangle$ $(r \restriction^r_\circ A)$
using *assms* **by** *auto*

lemma *vrrestriction-vintersection*: $(r \cap_\circ s) \restriction^r_\circ A = r \restriction^r_\circ A \cap_\circ s \restriction^r_\circ A$ **by** *auto*

lemma *vrrestriction-vunion*: $(r \cup_\circ s) \restriction^r_\circ A = r \restriction^r_\circ A \cup_\circ s \restriction^r_\circ A$ **by** *auto*

lemma *vrrestriction-vdiff*: $(r -_\circ s) \restriction^r_\circ A = r \restriction^r_\circ A -_\circ s \restriction^r_\circ A$ **by** *auto*

Connections.

lemma *vrrestriction-vid-on*[*simp*]: (*vid-on* A) $\restriction^r_\circ B$ = *vid-on* $(A \cap_\circ B)$ **by** *auto*

lemma *vrrestriction-vconst-on*:
assumes $c \in_\circ B$
shows (*vconst-on* A c) $\restriction^r_\circ B$ = *vconst-on* A c
using *assms* **by** *auto*

lemma *vrrestriction-vcomp*[*simp*]: $(r \circ_\circ s) \restriction^r_\circ A = (r \restriction^r_\circ A) \circ_\circ s$ **by** *auto*

Previous connections.

lemma *vcomp-vid-on-rel*[*simp*]: *vid-on* $A \circ_\circ r = r \restriction^r_\circ A$
by (*auto intro*!: *vsubset-antisym*)

lemma *vcomp-vconst-on-rel*: (*vconst-on* A c) $\circ_\circ r$ = (*vconst-on* A c) $\circ_\circ$ $(r \restriction^r_\circ A)$
by *auto*

lemma *vlrestriction-vconverse*: $r^{-1}_\circ \restriction^l_\circ A = (r \restriction^r_\circ A)^{-1}_\circ$ **by** *auto*

lemma *vrrestriction-vconverse*: $r^{-1}_\circ \restriction^r_\circ A = (r \restriction^l_\circ A)^{-1}_\circ$ **by** *auto*

Special properties.

lemma *vrrestriction-vsubset-rel*: $r \restriction^r_\circ A \subseteq_\circ r$ **by** *auto*

lemma *vrrestriction-vsubset-vpairs*: $r \restriction^r_\circ A \subseteq_\circ$ *vpairs* r **by** *auto*

Restriction

definition *vrestriction* :: $V \Rightarrow V$
where *vrestriction D* =
VLambda D ($\lambda\langle r,\ A\rangle$. *set* $\{\langle a,\ b\rangle \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge \langle a,\ b\rangle \in_\circ r\}$)

abbreviation *app-vrestriction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\restriction_\circ\rangle$ 80)
where $r \restriction_\circ A \equiv$ *vrestriction* (*set* $\{\langle r,\ A\rangle\}$) ($\|\langle r,\ A\rangle\|$)

lemma *app-vrestriction-def*:
$r \restriction_\circ A$ = *set* $\{\langle a,\ b\rangle \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge \langle a,\ b\rangle \in_\circ r\}$
unfolding *vrestriction-def* **by** *simp*

lemma *vrestriction-small*[*simp*]:
small $\{\langle a,\ b\rangle \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge \langle a,\ b\rangle \in_\circ r\}$
by (*rule down*[*of* - *r*]) *auto*

Rules.

lemma *vrestrictionI*[*intro*!]:
assumes $\langle r,\ A\rangle \in_\circ D$
shows $\langle\langle r,\ A\rangle,\ r \restriction_\circ A\rangle \in_\circ$ *vrestriction D*
using *assms* **unfolding** *vrestriction-def* **by** (*simp add*: *VLambda-iff*2)

lemma *vrestrictionD*[*dest*]:
assumes $\langle\langle r,\ A\rangle,\ s\rangle \in_\circ$ *vrestriction D*
shows $\langle r,\ A\rangle \in_\circ D$ **and** $s = r \restriction_\circ A$
using *assms* **unfolding** *vrestriction-def* **by** *auto*

lemma *vrestrictionE*[*elim*]:
assumes $x \in_\circ$ *vrestriction D* **and** $D \subseteq_\circ R \times_\circ X$
obtains *r A* **where** $x = \langle\langle r,\ A\rangle,\ r \restriction_\circ A\rangle$ **and** $r \in_\circ R$ **and** $A \in_\circ X$
using *assms* **unfolding** *vrestriction-def* **by** *auto*

lemma *app-vrestrictionI*[*intro*!]:
assumes $a \in_\circ A$ **and** $b \in_\circ A$ **and** $\langle a,\ b\rangle \in_\circ r$
shows $\langle a,\ b\rangle \in_\circ r \restriction_\circ A$
using *assms* **unfolding** *vrestriction-def* **by** *simp*

lemma *app-vrestrictionD*[*dest*]:
assumes $\langle a,\ b\rangle \in_\circ r \restriction_\circ A$
shows $a \in_\circ A$ **and** $b \in_\circ A$ **and** $\langle a,\ b\rangle \in_\circ r$
using *assms* **unfolding** *vrestriction-def* **by** *auto*

lemma *app-vrestrictionE*[*elim*]:
assumes $x \in_\circ r \restriction_\circ A$
obtains *a b* **where** $x = \langle a,\ b\rangle$ **and** $a \in_\circ A$ **and** $b \in_\circ A$ **and** $\langle a,\ b\rangle \in_\circ r$
using *assms* **unfolding** *vrestriction-def* **by** *clarsimp*

Set operations.

lemma *vrestriction-on-vempty*[*simp*]: $r \restriction_\circ 0 = 0$
by (*auto intro*!: *vsubset-antisym*)

lemma *vrestriction-vempty*[*simp*]: $0 \restriction_\circ A = 0$ **by** *auto*

lemma *vrestriction-vsingleton-in*[*simp*]:
assumes $a \in_\circ A$ **and** $b \in_\circ A$
shows *set* $\{\langle a,\ b\rangle\} \restriction_\circ A$ = *set* $\{\langle a,\ b\rangle\}$
using *assms* **by** *auto*

lemma *vrestriction-vsingleton-nin-left*[*simp*]:
assumes $a \notin_\circ A$
shows *set* $\{\langle a, b\rangle\} \restriction_\circ A = 0$
using *assms* **by** *auto*

lemma *vrestriction-vsingleton-nin-right*[*simp*]:
assumes $b \notin_\circ A$
shows *set* $\{\langle a, b\rangle\} \restriction_\circ A = 0$
using *assms* **by** *auto*

lemma *vrestriction-mono*:
assumes $A \subseteq_\circ B$
shows $r \restriction_\circ A \subseteq_\circ r \restriction_\circ B$
using *assms* **by** *auto*

lemma *vrestriction-vinsert-nin*[*simp*]:
assumes $a \notin_\circ A$ **and** $b \notin_\circ A$
shows $(\mathit{vinsert}\ \langle a, b\rangle\ r) \restriction_\circ A = r \restriction_\circ A$
using *assms* **by** *auto*

lemma *vrestriction-vinsert-in*:
assumes $a \in_\circ A$ **and** $b \in_\circ A$
shows $(\mathit{vinsert}\ \langle a, b\rangle\ r) \restriction_\circ A = \mathit{vinsert}\ \langle a, b\rangle\ (r \restriction_\circ A)$
using *assms* **by** *auto*

lemma *vrestriction-vintersection*: $(r \cap_\circ s) \restriction_\circ A = r \restriction_\circ A \cap_\circ s \restriction_\circ A$ **by** *auto*

lemma *vrestriction-vunion*: $(r \cup_\circ s) \restriction_\circ A = r \restriction_\circ A \cup_\circ s \restriction_\circ A$ **by** *auto*

lemma *vrestriction-vdiff*: $(r -_\circ s) \restriction_\circ A = r \restriction_\circ A -_\circ s \restriction_\circ A$ **by** *auto*

Connections.

lemma *vrestriction-vid-on*[*simp*]: $(\mathit{vid\text{-}on}\ A) \restriction_\circ B = \mathit{vid\text{-}on}\ (A \cap_\circ B)$ **by** *auto*

lemma *vrestriction-vconst-on-ex*:
assumes $c \in_\circ B$
shows $(\mathit{vconst\text{-}on}\ A\ c) \restriction_\circ B = \mathit{vconst\text{-}on}\ (A \cap_\circ B)\ c$
using *assms* **by** *auto*

lemma *vrestriction-vconst-on-nex*:
assumes $c \notin_\circ B$
shows $(\mathit{vconst\text{-}on}\ A\ c) \restriction_\circ B = 0$
using *assms* **by** *auto*

lemma *vrestriction-vcomp*[*simp*]: $(r \circ_\circ s) \restriction_\circ A = (r \restriction^r_\circ A) \circ_\circ (s \restriction^l_\circ A)$ **by** *auto*

lemma *vrestriction-vconverse*: $r^{-1}_\circ \restriction_\circ A = (r \restriction_\circ A)^{-1}_\circ$ **by** *auto*

Previous connections.

lemma *vrrestriction-vlrestriction*[*simp*]: $(r \restriction^r_\circ A) \restriction^l_\circ A = r \restriction_\circ A$ **by** *auto*

lemma *vlrestriction-vrrestriction*[*simp*]: $(r \restriction^l_\circ A) \restriction^r_\circ A = r \restriction_\circ A$ **by** *auto*

lemma *vrestriction-vlrestriction*[*simp*]: $(r \restriction_\circ A) \restriction^l_\circ A = r \restriction_\circ A$ **by** *auto*

lemma *vrestriction-vrrestriction*[*simp*]: $(r \restriction_\circ A) \restriction^r_\circ A = r \restriction_\circ A$ **by** *auto*

Special properties.

lemma *vrestriction-vsubset-vpairs*: $r \restriction_\circ A \subseteq_\circ vpairs\ r$ **by** *auto*

lemma *vrestriction-vsubset-vtimes*: $r \restriction_\circ A \subseteq_\circ A \times_\circ A$ **by** *auto*

lemma *vrestriction-vsubset-rel*: $r \restriction_\circ A \subseteq_\circ r$ **by** *auto*

2.4.3 Properties

Domain

definition *vdomain* :: $V \Rightarrow V$
  where $vdomain\ D = (\lambda r \in_\circ D.\ set\ \{a.\ \exists b.\ \langle a, b\rangle \in_\circ r\})$

abbreviation *app-vdomain* :: $V \Rightarrow V$ $(\langle\mathcal{D}_\circ\rangle)$
  where $\mathcal{D}_\circ\ r \equiv vdomain\ (set\ \{r\})\ (\!|r|\!)$

lemma *app-vdomain-def*: $\mathcal{D}_\circ\ r = set\ \{a.\ \exists b.\ \langle a, b\rangle \in_\circ r\}$
  unfolding *vdomain-def* **by** *simp*

lemma *vdomain-small*[*simp*]: $small\ \{a.\ \exists b.\ \langle a, b\rangle \in_\circ r\}$
proof–
  have *ss*: $\{a.\ \exists b.\ \langle a, b\rangle \in_\circ r\} \subseteq vfst\ `\ elts\ r$ **using** *image-iff* **by** *fastforce*
  have *small*: $small\ (vfst\ `\ elts\ r)$ **by** (*rule replacement*) *simp*
  show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

Rules.

lemma *vdomainI*[*intro*!]:
  assumes $r \in_\circ A$
  shows $\langle r, \mathcal{D}_\circ\ r\rangle \in_\circ vdomain\ A$
  using *assms* **unfolding** *vdomain-def* **by** *auto*

lemma *vdomainD*[*dest*]:
  assumes $\langle r, s\rangle \in_\circ vdomain\ A$
  shows $r \in_\circ A$ **and** $s = \mathcal{D}_\circ\ r$
  using *assms* **unfolding** *vdomain-def* **by** *auto*

lemma *vdomainE*[*elim*]:
  assumes $x \in_\circ vdomain\ A$
  obtains r **where** $x = \langle r, \mathcal{D}_\circ\ r\rangle$ **and** $r \in_\circ A$
  using *assms* **unfolding** *vdomain-def* **by** *auto*

lemma *app-vdomainI*[*intro*]:
  assumes $\langle a, b\rangle \in_\circ r$
  shows $a \in_\circ \mathcal{D}_\circ\ r$
  using *assms* **unfolding** *vdomain-def* **by** *auto*

lemma *app-vdomainD*[*dest*]:
  assumes $a \in_\circ \mathcal{D}_\circ\ r$
  shows $\exists b.\ \langle a, b\rangle \in_\circ r$
  using *assms* **unfolding** *vdomain-def* **by** *auto*

lemma *app-vdomainE*[*elim*]:
  assumes $a \in_\circ \mathcal{D}_\circ\ r$
  obtains b **where** $\langle a, b\rangle \in_\circ r$
  using *assms* **unfolding** *vdomain-def* **by** *clarsimp*

lemma *vdomain-iff*: $a \in_\circ \mathcal{D}_\circ\ r \longleftrightarrow (\exists\, y.\ \langle a,\ y\rangle \in_\circ r)$ **by** *auto*

Set operations.

lemma *vdomain-vempty*[*simp*]: $\mathcal{D}_\circ\ 0 = 0$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *vdomain-vsingleton*[*simp*]: $\mathcal{D}_\circ\ (set\ \{\langle a,\ b\rangle\}) = set\ \{a\}$ **by** *auto*

lemma *vdomain-vdoubleton*[*simp*]: $\mathcal{D}_\circ\ (set\ \{\langle a,\ b\rangle,\ \langle c,\ d\rangle\}) = set\ \{a,\ c\}$
by (*auto simp*: *vinsert-set-insert-eq*)

lemma *vdomain-mono*:
assumes $r \subseteq_\circ s$
shows $\mathcal{D}_\circ\ r \subseteq_\circ \mathcal{D}_\circ\ s$
using *assms* **by** *blast*

lemma *vdomain-vinsert*[*simp*]: $\mathcal{D}_\circ\ (vinsert\ \langle a,\ b\rangle\ r) = vinsert\ a\ (\mathcal{D}_\circ\ r)$
by (*auto intro*!: *vsubset-antisym*)

lemma *vdomain-vunion*: $\mathcal{D}_\circ\ (A \cup_\circ B) = \mathcal{D}_\circ\ A \cup_\circ \mathcal{D}_\circ\ B$
by (*auto intro*!: *vsubset-antisym*)

lemma *vdomain-vintersection-vsubset*: $\mathcal{D}_\circ\ (A \cap_\circ B) \subseteq_\circ \mathcal{D}_\circ\ A \cap_\circ \mathcal{D}_\circ\ B$ **by** *auto*

lemma *vdomain-vdiff-vsubset*: $\mathcal{D}_\circ\ A -_\circ \mathcal{D}_\circ\ B \subseteq_\circ \mathcal{D}_\circ\ (A -_\circ B)$ **by** *auto*

Connections.

lemma *vdomain-vid-on*[*simp*]: $\mathcal{D}_\circ\ (vid\text{-}on\ A) = A$
by (*auto intro*!: *vsubset-antisym*)

lemma *vdomain-vconst-on*[*simp*]: $\mathcal{D}_\circ\ (vconst\text{-}on\ A\ c) = A$
by (*auto intro*!: *vsubset-antisym*)

lemma *vdomain-VLambda*[*simp*]: $\mathcal{D}_\circ\ (\lambda a{\in_\circ}A.\ f\ a) = A$
by (*auto intro*!: *vsubset-antisym*)

lemma *vdomain-vlrestriction*: $\mathcal{D}_\circ\ (r \restriction^l_\circ A) = \mathcal{D}_\circ\ r \cap_\circ A$ **by** *auto*

lemma *vdomain-vlrestriction-vsubset*:
assumes $A \subseteq_\circ \mathcal{D}_\circ\ r$
shows $\mathcal{D}_\circ\ (r \restriction^l_\circ A) = A$
using *assms* **by** (*auto simp*: *vdomain-vlrestriction*)

Special properties.

lemma *vdomain-vsubset-vtimes*:
assumes *vpairs* $r \subseteq_\circ x \times_\circ y$
shows $\mathcal{D}_\circ\ r \subseteq_\circ x$
using *assms* **by** *auto*

Range

definition *vrange* :: $V \Rightarrow V$
where $vrange\ D = (\lambda r{\in_\circ}D.\ set\ \{b.\ \exists\, a.\ \langle a,\ b\rangle \in_\circ r\})$

abbreviation *app-vrange* :: $V \Rightarrow V$ $(\langle\mathcal{R}_\circ\rangle)$
where $\mathcal{R}_\circ\ r \equiv vrange\ (set\ \{r\})\ (\lfloor r \rfloor)$

lemma *app-vrange-def*: $\mathcal{R}_\circ\ r = set\ \{b.\ \exists\, a.\ \langle a,\ b\rangle \in_\circ r\}$
unfolding *vrange-def* **by** *simp*

lemma *vrange-small*[*simp*]: *small* {*b*. ∃ *a*. ⟨*a*, *b*⟩ ∈∘ *r*}
proof–
have *ss*: {*b*. ∃ *a*. ⟨*a*, *b*⟩ ∈∘ *r*} ⊆ *vsnd* ‘ *elts r* **using** *image-iff* **by** *fastforce*
have *small*: *small* (*vsnd* ‘ *elts r*) **by** (*rule replacement*) *simp*
show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

Rules.

lemma *vrangeI*[*intro*]:
assumes *r* ∈∘ *A*
shows ⟨*r*, ℛ∘ *r*⟩ ∈∘ *vrange A*
using *assms* **unfolding** *vrange-def* **by** *auto*

lemma *vrangeD*[*dest*]:
assumes ⟨*r*, *s*⟩ ∈∘ *vrange A*
shows *r* ∈∘ *A* **and** *s* = ℛ∘ *r*
using *assms* **unfolding** *vrange-def* **by** *auto*

lemma *vrangeE*[*elim*]:
assumes *x* ∈∘ *vrange A*
obtains *r* **where** *x* = ⟨*r*, ℛ∘ *r*⟩ **and** *r* ∈∘ *A*
using *assms* **unfolding** *vrange-def* **by** *auto*

lemma *app-vrangeI*[*intro*]:
assumes ⟨*a*, *b*⟩ ∈∘ *r*
shows *b* ∈∘ ℛ∘ *r*
using *assms* **unfolding** *vrange-def* **by** *auto*

lemma *app-vrangeD*[*dest*]:
assumes *b* ∈∘ ℛ∘ *r*
shows ∃ *a*. ⟨*a*, *b*⟩ ∈∘ *r*
using *assms* **unfolding** *vrange-def* **by** *simp*

lemma *app-vrangeE*[*elim*]:
assumes *b* ∈∘ ℛ∘ *r*
obtains *a* **where** ⟨*a*, *b*⟩ ∈∘ *r*
using *assms* **unfolding** *vrange-def* **by** *clarsimp*

lemma *vrange-iff*: *b* ∈∘ ℛ∘ *r* ⟷ (∃ *a*. ⟨*a*, *b*⟩ ∈∘ *r*) **by** *auto*

Set operations.

lemma *vrange-vempty*[*simp*]: ℛ∘ 0 = 0 **by** (*auto intro*!: *vsubset-antisym*)

lemma *vrange-vsingleton*[*simp*]: ℛ∘ (*set* {⟨*a*, *b*⟩}) = *set* {*b*} **by** *auto*

lemma *vrange-vdoubleton*[*simp*]: ℛ∘ (*set* {⟨*a*, *b*⟩, ⟨*c*, *d*⟩}) = *set* {*b*, *d*}
by (*auto simp*: *vinsert-set-insert-eq*)

lemma *vrange-mono*:
assumes *r* ⊆∘ *s*
shows ℛ∘ *r* ⊆∘ ℛ∘ *s*
using *assms* **by** *force*

lemma *vrange-vinsert*[*simp*]: ℛ∘ (*vinsert* ⟨*a*, *b*⟩ *r*) = *vinsert b* (ℛ∘ *r*)
by (*auto intro*!: *vsubset-antisym*)

lemma *vrange-vunion*: ℛ∘ (*r* ∪∘ *s*) = ℛ∘ *r* ∪∘ ℛ∘ *s*

by (*auto intro*!: *vsubset-antisym*)

lemma *vrange-vintersection-vsubset*: $\mathcal{R}_\circ\ (r \cap_\circ s) \subseteq_\circ \mathcal{R}_\circ\ r \cap_\circ \mathcal{R}_\circ\ s$ **by** *auto*

lemma *vrange-vdiff-vsubset*: $\mathcal{R}_\circ\ r -_\circ \mathcal{R}_\circ\ s \subseteq_\circ \mathcal{R}_\circ\ (r -_\circ s)$ **by** *auto*

Connections.

lemma *vrange-vid-on*[*simp*]: $\mathcal{R}_\circ\ (\mathit{vid\text{-}on}\ A) = A$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *vrange-vconst-on-vempty*[*simp*]: $\mathcal{R}_\circ\ (\mathit{vconst\text{-}on}\ 0\ c) = 0$ **by** *auto*

lemma *vrange-vconst-on-ne*[*simp*]:
assumes $A \neq 0$
shows $\mathcal{R}_\circ\ (\mathit{vconst\text{-}on}\ A\ c) = \mathit{set}\ \{c\}$
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vrange-VLambda*: $\mathcal{R}_\circ\ (\lambda a{\in_\circ} A.\ f\ a) = \mathit{set}\ (f\ `\ \mathit{elts}\ A)$
by (*intro vsubset-antisym vsubsetI*) *auto*

lemma *vrange-vrrestriction*: $\mathcal{R}_\circ\ (r\ \restriction^r_\circ\ A) = \mathcal{R}_\circ\ r \cap_\circ A$ **by** *auto*

Previous connections

lemma *vdomain-vconverse*[*simp*]: $\mathcal{D}_\circ\ ({r^{-1}}_\circ) = \mathcal{R}_\circ\ r$
by (*auto intro*!: *vsubset-antisym*)

lemma *vrange-vconverse*[*simp*]: $\mathcal{R}_\circ\ ({r^{-1}}_\circ) = \mathcal{D}_\circ\ r$
by (*auto intro*!: *vsubset-antisym*)

Special properties.

lemma *vrange-iff-vdomain*: $b \in_\circ \mathcal{R}_\circ\ r \longleftrightarrow (\exists\, a{\in_\circ}\mathcal{D}_\circ\ r.\ \langle a,\ b\rangle \in_\circ r)$ **by** *auto*

lemma *vrange-vsubset-vtimes*:
assumes $\mathit{vpairs}\ r \subseteq_\circ x \times_\circ y$
shows $\mathcal{R}_\circ\ r \subseteq_\circ y$
using *assms* **by** *auto*

lemma *vrange-VLambda-vsubset*:
assumes $\bigwedge x.\ x \in_\circ A \Longrightarrow f\ x \in_\circ B$
shows $\mathcal{R}_\circ\ (\mathit{VLambda}\ A\ f) \subseteq_\circ B$
using *assms* **by** *auto*

lemma *vpairs-vsubset-vdomain-vrange*[*simp*]: $\mathit{vpairs}\ r \subseteq_\circ \mathcal{D}_\circ\ r \times_\circ \mathcal{R}_\circ\ r$
by (*rule vsubsetI*) *auto*

lemma *vrange-vsubset*:
assumes $\bigwedge x\ y.\ \langle x,\ y\rangle \in_\circ r \Longrightarrow y \in_\circ A$
shows $\mathcal{R}_\circ\ r \subseteq_\circ A$
using *assms* **by** *auto*

Field

definition *vfield* :: $V \Rightarrow V$
where $\mathit{vfield}\ D = (\lambda r{\in_\circ} D.\ \mathcal{D}_\circ\ r \cup_\circ \mathcal{R}_\circ\ r)$

abbreviation *app-vfield* :: $V \Rightarrow V$ $(\langle\mathcal{F}_\circ\rangle)$
where $\mathcal{F}_\circ\ r \equiv \mathit{vfield}\ (\mathit{set}\ \{r\})\ (\lfloor r \rfloor)$

lemma *app-vfield-def*: $\mathcal{F}_\circ\ r = \mathcal{D}_\circ\ r \cup_\circ \mathcal{R}_\circ\ r$ **unfolding** *vfield-def* **by** *simp*

Rules.

lemma *vfieldI*[*intro*!]:
assumes $r \in_\circ A$
shows $\langle r, \mathcal{F}_\circ\ r\rangle \in_\circ$ *vfield* A
using *assms* **unfolding** *vfield-def* **by** *auto*

lemma *vfieldD*[*dest*]:
assumes $\langle r, s\rangle \in_\circ$ *vfield* A
shows $r \in_\circ A$ **and** $s = \mathcal{F}_\circ\ r$
using *assms* **unfolding** *vfield-def* **by** *auto*

lemma *vfieldE*[*elim*]:
assumes $x \in_\circ$ *vfield* A
obtains r **where** $x = \langle r, \mathcal{F}_\circ\ r\rangle$ **and** $r \in_\circ A$
using *assms* **unfolding** *vfield-def* **by** *auto*

lemma *app-vfieldI*1[*intro*]:
assumes $a \in_\circ \mathcal{D}_\circ\ r \cup_\circ \mathcal{R}_\circ\ r$
shows $a \in_\circ \mathcal{F}_\circ\ r$
using *assms* **unfolding** *vfield-def* **by** *simp*

lemma *app-vfieldI*2[*intro*]:
assumes $\langle a, b\rangle \in_\circ r$
shows $a \in_\circ \mathcal{F}_\circ\ r$
using *assms* **by** *auto*

lemma *app-vfieldI*3[*intro*]:
assumes $\langle a, b\rangle \in_\circ r$
shows $b \in_\circ \mathcal{F}_\circ\ r$
using *assms* **by** *auto*

lemma *app-vfieldD*[*dest*]:
assumes $a \in_\circ \mathcal{F}_\circ\ r$
shows $a \in_\circ \mathcal{D}_\circ\ r \cup_\circ \mathcal{R}_\circ\ r$
using *assms* **unfolding** *vfield-def* **by** *simp*

lemma *app-vfieldE*[*elim*]:
assumes $a \in_\circ \mathcal{F}_\circ\ r$ **and** $a \in_\circ \mathcal{D}_\circ\ r \cup_\circ \mathcal{R}_\circ\ r \Longrightarrow P$
shows P
using *assms* **by** *auto*

lemma *app-vfield-vpairE*[*elim*]:
assumes $a \in_\circ \mathcal{F}_\circ\ r$
obtains b **where** $\langle a, b\rangle \in_\circ r \vee \langle b, a\rangle \in_\circ r$
using *assms* **unfolding** *app-vfield-def* **by** *blast*

lemma *vfield-iff*: $a \in_\circ \mathcal{F}_\circ\ r \longleftrightarrow (\exists\, b.\ \langle a, b\rangle \in_\circ r \vee \langle b, a\rangle \in_\circ r)$ **by** *auto*

Set operations.

lemma *vfield-vempty*[*simp*]: $\mathcal{F}_\circ\ 0 = 0$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *vfield-vsingleton*[*simp*]: $\mathcal{F}_\circ\ (set\ \{\langle a, b\rangle\}) = set\ \{a, b\}$
by (*simp add*: *app-vfield-def vinsert-set-insert-eq*)

lemma *vfield-vdoubleton*[*simp*]: $\mathcal{F}_\circ\ (set\ \{\langle a, b\rangle, \langle c, d\rangle\}) = set\ \{a, b, c, d\}$
by (*auto simp*: *vinsert-set-insert-eq*)

lemma *vfield-mono*:

assumes $r \subseteq_\circ s$
shows $\mathcal{F}_\circ\ r \subseteq_\circ \mathcal{F}_\circ\ s$
using *assms* **by** *fastforce*

lemma *vfield-vinsert*[*simp*]: $\mathcal{F}_\circ\ (\mathit{vinsert}\ \langle a,\ b\rangle\ r) = \mathit{set}\ \{a,\ b\} \cup_\circ \mathcal{F}_\circ\ r$
by (*auto intro*!: *vsubset-antisym*)

lemma *vfield-vunion*[*simp*]: $\mathcal{F}_\circ\ (r \cup_\circ s) = \mathcal{F}_\circ\ r \cup_\circ \mathcal{F}_\circ\ s$
by (*auto intro*!: *vsubset-antisym*)

Connections.

lemma *vid-on-vfield*[*simp*]: $\mathcal{F}_\circ\ (\mathit{vid\text{-}on}\ A) = A$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *vconst-on-vfield-ne*[*intro*, *simp*]:
assumes $A \neq 0$
shows $\mathcal{F}_\circ\ (\mathit{vconst\text{-}on}\ A\ c) = \mathit{vinsert}\ c\ A$
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vconst-on-vfield-vempty*[*simp*]: $\mathcal{F}_\circ\ (\mathit{vconst\text{-}on}\ 0\ c) = 0$ **by** *auto*

lemma *vfield-vconverse*[*simp*]: $\mathcal{F}_\circ\ ({r^{-1}}_\circ) = \mathcal{F}_\circ\ r$
by (*auto intro*!: *vsubset-antisym*)

Image

definition *vimage* :: $V \Rightarrow V$
where $\mathit{vimage}\ D = \mathit{VLambda}\ D\ (\lambda\langle r,\ A\rangle.\ \mathcal{R}_\circ\ (r \restriction^l_\circ A))$

abbreviation *app-vimage* :: $V \Rightarrow V \Rightarrow V$ (**infixr** ‹‘$_\circ$› 90)
where $r\ `_\circ\ A \equiv \mathit{vimage}\ (\mathit{set}\ \{\langle r,\ A\rangle\})\ (\!|\langle r,\ A\rangle|\!)$

lemma *app-vimage-def*: $r\ `_\circ\ A = \mathcal{R}_\circ\ (r \restriction^l_\circ A)$ **unfolding** *vimage-def* **by** *simp*

lemma *vimage-small*[*simp*]: *small* $\{b.\ \exists\, a{\in_\circ}A.\ \langle a,\ b\rangle \in_\circ r\}$
proof–
have *ss*: $\{b.\ \exists\, a{\in_\circ}A.\ \langle a,\ b\rangle \in_\circ r\} \subseteq \mathit{vsnd}\ `\ \mathit{elts}\ r$
using *image-iff* **by** *fastforce*
have *small*: *small* $(\mathit{vsnd}\ `\ \mathit{elts}\ r)$ **by** (*rule replacement*) *simp*
show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

lemma *app-vimage-set-def*: $r\ `_\circ\ A = \mathit{set}\ \{b.\ \exists\, a{\in_\circ}A.\ \langle a,\ b\rangle \in_\circ r\}$
unfolding *vimage-def vrange-def* **by** *auto*

Rules.

lemma *vimageI*[*intro*!]:
assumes $\langle r,\ A\rangle \in_\circ D$
shows $\langle\langle r,\ A\rangle,\ r\ `_\circ\ A\rangle \in_\circ \mathit{vimage}\ D$
using *assms* **unfolding** *vimage-def* **by** (*simp add*: *VLambda-iff*2)

lemma *vimageD*[*dest*]:
assumes $\langle\langle r,\ A\rangle,\ s\rangle \in_\circ \mathit{vimage}\ D$
shows $\langle r,\ A\rangle \in_\circ D$ **and** $s = r\ `_\circ\ A$
using *assms* **unfolding** *vimage-def* **by** *auto*

lemma *vimageE*[*elim*]:
assumes $x \in_\circ \mathit{vimage}\ (R \times_\circ X)$
obtains $r\ A$ **where** $x = \langle\langle r,\ A\rangle,\ r\ `_\circ\ A\rangle$ **and** $r \in_\circ R$ **and** $A \in_\circ X$

using *assms* **unfolding** *vimage-def* **by** *auto*

lemma *app-vimageI1*:
assumes $x \in_\circ \mathcal{R}_\circ\ (r \restriction^l_\circ A)$
shows $x \in_\circ r\ `_\circ\ A$
using *assms* **unfolding** *vimage-def* **by** *simp*

lemma *app-vimageI2*[*intro*]:
assumes $\langle a,\ b\rangle \in_\circ r$ **and** $a \in_\circ A$
shows $b \in_\circ r\ `_\circ\ A$
using *assms app-vimageI1* **by** *auto*

lemma *app-vimageD*[*dest*]:
assumes $x \in_\circ r\ `_\circ\ A$
shows $x \in_\circ \mathcal{R}_\circ\ (r \restriction^l_\circ A)$
using *assms* **unfolding** *vimage-def* **by** *simp*

lemma *app-vimageE*[*elim*]:
assumes $b \in_\circ r\ `_\circ\ A$
obtains a **where** $\langle a,\ b\rangle \in_\circ r$ **and** $a \in_\circ A$
using *assms* **unfolding** *vimage-def* **by** *auto*

lemma *app-vimage-iff*: $b \in_\circ r\ `_\circ\ A \longleftrightarrow (\exists\, a{\in_\circ}A.\ \langle a,\ b\rangle \in_\circ r)$ **by** *auto*

Set operations.

lemma *vimage-vempty*[*simp*]: $0\ `_\circ\ A = 0$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *vimage-of-vempty*[*simp*]: $r\ `_\circ\ 0 = 0$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vsingleton*: $r\ `_\circ\ set\ \{a\} = set\ \{b.\ \langle a,\ b\rangle \in_\circ r\}$
proof–
have $\{b.\ \langle a,\ b\rangle \in_\circ r\} \subseteq \{b.\ \exists\, a.\ \langle a,\ b\rangle \in_\circ r\}$ **by** *auto*
then have [*simp*]: *small* $\{b.\ \langle a,\ b\rangle \in_\circ r\}$
by (*rule smaller-than-small*[*OF vrange-small*[*of r*]])
show *?thesis* **using** *app-vimage-set-def* **by** *auto*
qed

lemma *vimage-vsingleton-in*[*intro*, *simp*]:
assumes $a \in_\circ A$
shows $set\ \{\langle a,\ b\rangle\}\ `_\circ\ A = set\ \{b\}$
using *assms* **by** *auto*

lemma *vimage-vsingleton-nin*[*intro*, *simp*]:
assumes $a \notin_\circ A$
shows $set\ \{\langle a,\ b\rangle\}\ `_\circ\ A = 0$
using *assms* **by** *auto*

lemma *vimage-vsingleton-vinsert*[*simp*]: $set\ \{\langle a,\ b\rangle\}\ `_\circ\ vinsert\ a\ A = set\ \{b\}$
by *auto*

lemma *vimage-mono*:
assumes $r' \subseteq_\circ r$ **and** $A' \subseteq_\circ A$
shows $(r'\ `_\circ\ A') \subseteq_\circ (r\ `_\circ\ A)$
using *assms* **by** *fastforce*

lemma *vimage-vinsert*: $r\ `_\circ\ (vinsert\ a\ A) = r\ `_\circ\ set\ \{a\} \cup_\circ r\ `_\circ\ A$
by (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vunion-left*: $(r \cup_\circ s)$ ‘$_\circ$ $A = r$ ‘$_\circ$ $A \cup_\circ s$ ‘$_\circ$ A
by (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vunion-right*: r ‘$_\circ$ $(A \cup_\circ B) = r$ ‘$_\circ$ $A \cup_\circ r$ ‘$_\circ$ B
by (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vintersection*: r ‘$_\circ$ $(A \cap_\circ B) \subseteq_\circ r$ ‘$_\circ$ $A \cap_\circ r$ ‘$_\circ$ B **by** *auto*

lemma *vimage-vdiff*: r ‘$_\circ$ $A -_\circ r$ ‘$_\circ$ $B \subseteq_\circ r$ ‘$_\circ$ $(A -_\circ B)$ **by** *auto*

Previous set operations.

lemma *VPow-vinsert*:
$VPow\ (vinsert\ a\ A) = VPow\ A \cup_\circ ((\lambda x{\in_\circ} VPow\ A.\ vinsert\ a\ x)$ ‘$_\circ$ $VPow\ A)$
proof(*intro vsubset-antisym vsubsetI*)
fix x **assume** $x \in_\circ VPow\ (vinsert\ a\ A)$
then have $x \subseteq_\circ vinsert\ a\ A$ **by** *simp*
then consider $x \subseteq_\circ A \mid a \in_\circ x$ **by** *auto*
then show $x \in_\circ VPow\ A \cup_\circ (\lambda x{\in_\circ} VPow\ A.\ vinsert\ a\ x)$ ‘$_\circ$ $VPow\ A$
proof *cases*
case 1 **then show** *?thesis* **by** *simp*
next
case 2
define x' **where** $x' = x -_\circ set\ \{a\}$
with 2 **have** $x = vinsert\ a\ x'$ **and** $a \notin_\circ x'$ **by** *auto*
with ‹$x \subseteq_\circ vinsert\ a\ A$› **show** *?thesis*
unfolding *vimage-def*
by (*fastforce simp*: *vsubset-vinsert vlrestriction-VLambda*)
qed
qed (*elim vunionE*, *auto*)

Special properties.

lemma *vimage-vsingleton-iff*[*iff*]: $b \in_\circ r$ ‘$_\circ$ $set\ \{a\} \longleftrightarrow \langle a, b\rangle \in_\circ r$ **by** *auto*

lemma *vimage-is-vempty*[*iff*]: r ‘$_\circ$ $A = 0 \longleftrightarrow vdisjnt\ (\mathcal{D}_\circ\ r)\ A$ **by** *fastforce*

lemma *vcomp-vimage-vtimes-right*:
assumes r ‘$_\circ$ $Y = Z$
shows $r \circ_\circ (X \times_\circ Y) = X \times_\circ Z$
proof(*intro vsubset-antisym vsubsetI*)
fix x **assume** x: $x \in_\circ r \circ_\circ (X \times_\circ Y)$
then obtain $a\ c$ **where** *x-def*: $x = \langle a, c\rangle$ **and** $a \in_\circ X$ **and** $c \in_\circ \mathcal{R}_\circ\ r$ **by** *auto*
with x **obtain** b **where** $\langle a, b\rangle \in_\circ X \times_\circ Y$ **and** $\langle b, c\rangle \in_\circ r$ **by** *clarsimp*
then show $x \in_\circ X \times_\circ Z$ **unfolding** *x-def* **using** *assms* **by** *auto*
next
fix x **assume** $x \in_\circ X \times_\circ Z$
then obtain $a\ c$ **where** *x-def*: $x = \langle a, c\rangle$ **and** $a \in_\circ X$ **and** $c \in_\circ Z$ **by** *auto*
then show $x \in_\circ r \circ_\circ X \times_\circ Y$
using *assms* **unfolding** *x-def* **by** (*meson VSigmaI app-vimageE vcompI*)
qed

Connections.

lemma *vid-on-vimage*[*simp*]: *vid-on* A ‘$_\circ$ $B = A \cap_\circ B$
by (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vconst-on-ne*[*simp*]:
assumes $B \cap_\circ A \neq 0$
shows *vconst-on* $A\ c$ ‘$_\circ$ $B = set\ \{c\}$

using *assms* **by** *auto*

lemma *vimage-vconst-on-vempty*[*simp*]:
assumes *vdisjnt A B*
shows *vconst-on A c* ‘$_\circ$ *B* = 0
using *assms* **by** *auto*

lemma *vimage-vconst-on-vsubset-vconst*: *vconst-on A c* ‘$_\circ$ *B* $\subseteq_\circ$ *set* {*c*} **by** *auto*

lemma *vimage-VLambda-vrange*: (λ*a*∈$_\circ$*A*. *f a*) ‘$_\circ$ *B* = $\mathcal{R}_\circ$ (λ*a*∈$_\circ$*A* ∩$_\circ$ *B*. *f a*)
unfolding *vimage-def* **by** (*simp add*: *vlrestriction-VLambda*)

lemma *vimage-VLambda-vrange-rep*: (λ*a*∈$_\circ$*A*. *f a*) ‘$_\circ$ *A* = $\mathcal{R}_\circ$ (λ*a*∈$_\circ$*A*. *f a*)
by (*simp add*: *vimage-VLambda-vrange*)

lemma *vcomp-vimage*: (*r* ∘$_\circ$ *s*) ‘$_\circ$ *A* = *r* ‘$_\circ$ (*s* ‘$_\circ$ *A*)
by (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vlrestriction*[*simp*]: (*r* ↾$^l_\circ$ *A*) ‘$_\circ$ *B* = *r* ‘$_\circ$ (*A* ∩$_\circ$ *B*)
by (*auto intro*!: *vsubset-antisym*)

lemma *vimage-vrrestriction*[*simp*]: (*r* ↾$^r_\circ$ *A*) ‘$_\circ$ *B* = *A* ∩$_\circ$ *r* ‘$_\circ$ *B* **by** *auto*

lemma *vimage-vrestriction*[*simp*]: (*r* ↾$_\circ$ *A*) ‘$_\circ$ *B* = *A* ∩$_\circ$ (*r* ‘$_\circ$ (*A* ∩$_\circ$ *B*)) **by** *auto*

lemma *vimage-vdomain*: *r* ‘$_\circ$ $\mathcal{D}_\circ$ *r* = $\mathcal{R}_\circ$ *r* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vimage-eq-imp-vcomp*:
assumes *r* ‘$_\circ$ *A* = *s* ‘$_\circ$ *B*
shows (*t* ∘$_\circ$ *r*) ‘$_\circ$ *A* = (*t* ∘$_\circ$ *s*) ‘$_\circ$ *B*
using *assms* **by** (*metis vcomp-vimage*)

Previous connections.

lemma *vcomp-rel-vconst*: *r* ∘$_\circ$ (*vconst-on A c*) = *A* ×$_\circ$ (*r* ‘$_\circ$ *set* {*c*})
by *auto*

lemma *vcomp-VLambda*:
(λ*b*∈$_\circ$((λ*a*∈$_\circ$*A*. *g a*) ‘$_\circ$ *A*). *f b*) ∘$_\circ$ (λ*a*∈$_\circ$*A*. *g a*) = (λ*a*∈$_\circ$*A*. (*f* ∘ *g*) *a*)
using *VLambda-iff*1 **by** (*auto intro*!: *vsubset-antisym*)+

Further special properties.

lemma *vimage-vsubset*:
assumes *r* $\subseteq_\circ$ *A* ×$_\circ$ *B*
shows *r* ‘$_\circ$ *C* $\subseteq_\circ$ *B*
using *assms* **by** *auto*

lemma *vimage-vdomain-vsubset*: *r* ‘$_\circ$ *A* $\subseteq_\circ$ *r* ‘$_\circ$ $\mathcal{D}_\circ$ *r* **by** *auto*

lemma *vdomain-vsubset-VUnion*2: $\mathcal{D}_\circ$ *r* $\subseteq_\circ$ $\bigcup_\circ(\bigcup_\circ r)$
proof(*intro vsubsetI*)
fix *x* **assume** *x* ∈$_\circ$ $\mathcal{D}_\circ$ *r*
then obtain *y* **where** ⟨*x*, *y*⟩ ∈$_\circ$ *r* **by** *auto*
then have *set* {*set* {*x*}, *set* {*x*, *y*}} ∈$_\circ$ *r* **unfolding** *vpair-def* **by** *auto*
with *insert-commute* **have** *xy-Ur*: *set* {*x*, *y*} ∈$_\circ$ $\bigcup_\circ r$
unfolding *VUnion-iff* **by** *auto*
define *Ur* **where** *Ur* = $\bigcup_\circ r$
from *xy-Ur* **show** *x* ∈$_\circ$ $\bigcup_\circ(\bigcup_\circ r)$
unfolding *Ur-def*[*symmetric*] **by** (*auto dest*: *VUnionI*)

qed

lemma *vrange-vsubset-VUnion2*: $\mathcal{R}_\circ\ r \subseteq_\circ \bigcup_\circ(\bigcup_\circ r)$
proof(*intro vsubsetI*)
fix y **assume** $y \in_\circ \mathcal{R}_\circ\ r$
then obtain x **where** $\langle x, y\rangle \in_\circ r$ **by** *auto*
then have *set* $\{set\ \{x\}, set\ \{x, y\}\} \in_\circ r$ **unfolding** *vpair-def* **by** *auto*
with *insert-commute* **have** *xy-Ur*: *set* $\{x, y\} \in_\circ \bigcup_\circ r$
unfolding *VUnion-iff* **by** *auto*
define *Ur* **where** $Ur = \bigcup_\circ r$
from *xy-Ur* **show** $y \in_\circ \bigcup_\circ(\bigcup_\circ r)$
unfolding *Ur-def*[*symmetric*] **by** (*auto dest*: *VUnionI*)
qed

lemma *vfield-vsubset-VUnion2*: $\mathcal{F}_\circ\ r \subseteq_\circ \bigcup_\circ(\bigcup_\circ r)$
using *vdomain-vsubset-VUnion2 vrange-vsubset-VUnion2*
by (*auto simp*: *app-vfield-def*)

Inverse image

definition *invimage* :: $V \Rightarrow V$
where *invimage* D = *VLambda* D $(\lambda\langle r, A\rangle.\ r^{-1}{}_\circ\ `_\circ\ A)$

abbreviation *app-invimage* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle -`_\circ\rangle$ 90)
where $r -`_\circ A \equiv$ *invimage* (*set* $\{\langle r, A\rangle\}$) $(\!|\langle r, A\rangle|\!)$

lemma *app-invimage-def*: $r -`_\circ A = r^{-1}{}_\circ\ `_\circ\ A$ **unfolding** *invimage-def* **by** *simp*

lemma *invimage-small*[*simp*]: *small* $\{a.\ \exists\, b{\in_\circ}A.\ \langle a, b\rangle \in_\circ r\}$
proof–
have *ss*: $\{a.\ \exists\, b{\in_\circ}A.\ \langle a, b\rangle \in_\circ r\} \subseteq$ *vfst* ` *elts* r
using *image-iff* **by** *fastforce*
have *small*: *small* (*vfst* ` *elts* r) **by** (*rule replacement*) *simp*
show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

Rules.

lemma *invimageI*[*intro*!]:
assumes $\langle r, A\rangle \in_\circ D$
shows $\langle\langle r, A\rangle, r -`_\circ A\rangle \in_\circ$ *invimage* D
using *assms* **unfolding** *invimage-def* **by** (*simp add*: *VLambda-iff*2)

lemma *invimageD*[*dest*]:
assumes $\langle\langle r, A\rangle, s\rangle \in_\circ$ *invimage* D
shows $\langle r, A\rangle \in_\circ D$ **and** $s = r -`_\circ A$
using *assms* **unfolding** *invimage-def* **by** *auto*

lemma *invimageE*[*elim*]:
assumes $x \in_\circ$ *invimage* D **and** $D \subseteq_\circ R \times_\circ X$
obtains $r\ A$ **where** $x = \langle\langle r, A\rangle, r -`_\circ A\rangle$ **and** $r \in_\circ R$ **and** $A \in_\circ X$
using *assms* **unfolding** *invimage-def* **by** *auto*

lemma *app-invimageI*[*intro*]:
assumes $\langle a, b\rangle \in_\circ r$ **and** $b \in_\circ A$
shows $a \in_\circ r -`_\circ A$
using *assms invimage-def* **by** *auto*

lemma *app-invimageD*[*dest*]:

assumes $a \in_\circ r -`_\circ A$
shows $a \in_\circ \mathcal{D}_\circ (r \restriction^r_\circ A)$
using *assms* **using** *invimage-def* **by** *auto*

lemma *app-invimageE*[*elim*]:
assumes $a \in_\circ r -`_\circ A$
obtains b **where** $\langle a, b\rangle \in_\circ r$ **and** $b \in_\circ A$
using *assms* **unfolding** *invimage-def* **by** *auto*

lemma *app-invimageI*1:
assumes $a \in_\circ \mathcal{D}_\circ (r \restriction^r_\circ A)$
shows $a \in_\circ r -`_\circ A$
using *assms* **unfolding** *vimage-def*
by (*simp add*: *invimage-def app-vimageI*1 *vlrestriction-vconverse*)

lemma *app-invimageD*1:
assumes $a \in_\circ r -`_\circ A$
shows $a \in_\circ \mathcal{D}_\circ (r \restriction^r_\circ A)$
using *assms* **by** *fastforce*

lemma *app-invimageE*1:
assumes $a \in_\circ r -`_\circ A$ **and** $a \in_\circ \mathcal{D}_\circ (r \restriction^r_\circ A) \Longrightarrow P$
shows P
using *assms* **unfolding** *invimage-def* **by** *auto*

lemma *app-invimageI*2:
assumes $a \in_\circ r^{-1}_\circ `_\circ A$
shows $a \in_\circ r -`_\circ A$
using *assms* **unfolding** *invimage-def* **by** *simp*

lemma *app-invimageD*2:
assumes $a \in_\circ r -`_\circ A$
shows $a \in_\circ r^{-1}_\circ `_\circ A$
using *assms* **unfolding** *invimage-def* **by** *simp*

lemma *app-invimageE*2:
assumes $a \in_\circ r -`_\circ A$ **and** $a \in_\circ r^{-1}_\circ `_\circ A \Longrightarrow P$
shows P
unfolding *vimage-def* **by** (*simp add*: *assms app-invimageD*2)

lemma *invimage-iff*: $a \in_\circ r -`_\circ A \longleftrightarrow (\exists\, b{\in_\circ}A.\ \langle a, b\rangle \in_\circ r)$ **by** *auto*

lemma *invimage-iff*1: $a \in_\circ r -`_\circ A \longleftrightarrow a \in_\circ \mathcal{D}_\circ (r \restriction^r_\circ A)$ **by** *auto*

lemma *invimage-iff*2: $a \in_\circ r -`_\circ A \longleftrightarrow a \in_\circ r^{-1}_\circ `_\circ A$ **by** *auto*

Set operations.

lemma *invimage-vempty*[*simp*]: $0 -`_\circ A = 0$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *invimage-of-vempty*[*simp*]: $r -`_\circ 0 = 0$ **by** (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vsingleton-in*[*intro*, *simp*]:
assumes $b \in_\circ A$
shows *set* $\{\langle a, b\rangle\} -`_\circ A$ = *set* $\{a\}$
using *assms* **by** *auto*

lemma *invimage-vsingleton-nin*[*intro*, *simp*]:
assumes $b \notin_\circ A$

shows *set* $\{\langle a,\ b\rangle\} -`_\circ\ A = 0$
using *assms* **by** *auto*

lemma *invimage-vsingleton-vinsert*[*intro*, *simp*]:
$set\ \{\langle a,\ b\rangle\} -`_\circ\ vinsert\ b\ A = set\ \{a\}$
by *auto*

lemma *invimage-mono*:
assumes $r' \subseteq_\circ r$ **and** $A' \subseteq_\circ A$
shows $(r' -`_\circ\ A') \subseteq_\circ (r -`_\circ\ A)$
using *assms* **by** *fastforce*

lemma *invimage-vinsert*: $r -`_\circ\ (vinsert\ a\ A) = r -`_\circ\ set\ \{a\} \cup_\circ r -`_\circ\ A$
by (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vunion-left*: $(r \cup_\circ s) -`_\circ\ A = r -`_\circ\ A \cup_\circ s -`_\circ\ A$
by (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vunion-right*: $r -`_\circ\ (A \cup_\circ B) = r -`_\circ\ A \cup_\circ r -`_\circ\ B$
by (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vintersection*: $r -`_\circ\ (A \cap_\circ B) \subseteq_\circ r -`_\circ\ A \cap_\circ r -`_\circ\ B$ **by** *auto*

lemma *invimage-vdiff*: $r -`_\circ\ A -_\circ r -`_\circ\ B \subseteq_\circ r -`_\circ\ (A -_\circ B)$ **by** *auto*

Special properties.

lemma *invimage-set-def*: $r -`_\circ\ A = set\ \{a.\ \exists\, b{\in_\circ}A.\ \langle a,\ b\rangle \in_\circ r\}$ **by** *fastforce*

lemma *invimage-eq-vdomain-vrestriction*: $r -`_\circ\ A = \mathcal{D}_\circ\ (r \restriction^r_\circ A)$ **by** *fastforce*

lemma *invimage-vrange*[*simp*]: $r -`_\circ\ \mathcal{R}_\circ\ r = \mathcal{D}_\circ\ r$
unfolding *invimage-def* **by** (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vrange-vsubset*[*simp*]:
assumes $\mathcal{R}_\circ\ r \subseteq_\circ B$
shows $r -`_\circ\ B = \mathcal{D}_\circ\ r$
using *assms* **unfolding** *app-invimage-def* **by** (*blast intro*!: *vsubset-antisym*)

Connections.

lemma *invimage-vid-on*[*simp*]: $vid\text{-}on\ A -`_\circ\ B = A \cap_\circ B$
by (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vconst-on-vsubset-vdomain*[*simp*]: $vconst\text{-}on\ A\ c -`_\circ\ B \subseteq_\circ A$
unfolding *invimage-def* **by** *auto*

lemma *invimage-vconst-on-ne*[*simp*]:
assumes $c \in_\circ B$
shows $vconst\text{-}on\ A\ c -`_\circ\ B = A$
by (*simp add*: *assms invimage-eq-vdomain-vrestriction vrestriction-vconst-on*)

lemma *invimage-vconst-on-vempty*[*simp*]:
assumes $c \notin_\circ B$
shows $vconst\text{-}on\ A\ c -`_\circ\ B = 0$
using *assms* **by** *auto*

lemma *invimage-vcomp*: $(r \circ_\circ s) -`_\circ\ x = s -`_\circ\ (r -`_\circ\ x)$
by (*simp add*: *invimage-def vconverse-vcomp vcomp-vimage*)

lemma *invimage-vconverse*[*simp*]: $r^{-1}{}_\circ -`_\circ A = r\ `_\circ A$
by (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vlrestriction*[*simp*]: $(r \restriction^l_\circ A) -`_\circ B = A \cap_\circ r -`_\circ B$ **by** *auto*

lemma *invimage-vrrestriction*[*simp*]: $(r \restriction^r_\circ A) -`_\circ B = (r -`_\circ (A \cap_\circ B))$
by (*auto intro*!: *vsubset-antisym*)

lemma *invimage-vrestriction*[*simp*]: $(r \restriction_\circ A) -`_\circ B = A \cap_\circ (r -`_\circ (A \cap_\circ B))$
by *blast*

Previous connections.

lemma *vcomp-vconst-on-rel-vtimes*: *vconst-on* $A\ c \circ_\circ r = (r -`_\circ A) \times_\circ set\ \{c\}$
proof(*intro vsubset-antisym vsubsetI*)
fix x **assume** $x \in_\circ r -`_\circ A \times_\circ set\ \{c\}$
then obtain a **where** *x-def*: $x = \langle a, c\rangle$ **and** $a \in_\circ r -`_\circ A$ **by** *auto*
then obtain b **where** *ab*: $\langle a, b\rangle \in_\circ r$ **and** $b \in_\circ A$ **using** *invimage-iff* **by** *auto*
with $\langle b \in_\circ A\rangle$ **show** $x \in_\circ$ *vconst-on* $A\ c \circ_\circ r$ **unfolding** *x-def* **by** *auto*
qed *auto*

lemma *vdomain-vcomp*[*simp*]: $\mathcal{D}_\circ\ (r \circ_\circ s) = s -`_\circ \mathcal{D}_\circ\ r$ **by** *blast*

lemma *vrange-vcomp*[*simp*]: $\mathcal{R}_\circ\ (r \circ_\circ s) = r\ `_\circ \mathcal{R}_\circ\ s$ **by** *blast*

lemma *vdomain-vcomp-vsubset*:
assumes $\mathcal{R}_\circ\ s \subseteq_\circ \mathcal{D}_\circ\ r$
shows $\mathcal{D}_\circ\ (r \circ_\circ s) = \mathcal{D}_\circ\ s$
using *assms* **by** *simp*

2.4.4 Classification of relations

Binary relation

locale *vbrelation* =
fixes $r :: V$
assumes *vbrelation*: *vpairs* $r = r$

Rules.

lemma *vpairs-eqI*[*intro*!]:
assumes $\bigwedge x.\ x \in_\circ r \Longrightarrow \exists\, a\ b.\ x = \langle a, b\rangle$
shows *vpairs* $r = r$
using *assms* **by** *auto*

lemma *vpairs-eqD*[*dest*]:
assumes *vpairs* $r = r$
shows $\bigwedge x.\ x \in_\circ r \Longrightarrow \exists\, a\ b.\ x = \langle a, b\rangle$
using *assms* **by** *auto*

lemma *vpairs-eqE*[*elim*!]:
assumes *vpairs* $r = r$ **and** $(\bigwedge x.\ x \in_\circ r \Longrightarrow \exists\, a\ b.\ x = \langle a, b\rangle) \Longrightarrow P$
shows P
using *assms* **by** *auto*

lemmas *vbrelationI*[*intro*!] = *vbrelation.intro*
lemmas *vbrelationD*[*dest*!] = *vbrelation.vbrelation*

lemma *vbrelationE*[*elim*!]:
assumes *vbrelation* r **and** $(vpairs\ r = r) \Longrightarrow P$

shows P
using *assms* **unfolding** *vbrelation-def* **by** *auto*

lemma *vbrelationE1*[*elim*]:
assumes *vbrelation* r **and** $x \in_\circ r$
obtains a b **where** $x = \langle a, b \rangle$
using *assms* **by** *auto*

lemma *vbrelationD1*[*dest*]:
assumes *vbrelation* r **and** $x \in_\circ r$
shows $\exists\, a\ b.\ x = \langle a, b \rangle$
using *assms* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vinE*:
assumes $x \in_\circ r$
obtains a b **where** $x = \langle a, b \rangle$ **and** $a \in_\circ \mathcal{D}_\circ\ r$ **and** $b \in_\circ \mathcal{R}_\circ\ r$
using *assms vbrelation-axioms* **by** *blast*

Set operations.

lemma *vbrelation-vsubset*:
assumes *vbrelation* s **and** $r \subseteq_\circ s$
shows *vbrelation* r
using *assms* **by** *auto*

lemma *vbrelation-vinsert*[*simp*]: *vbrelation* (*vinsert* $\langle a, b \rangle$ r) $\longleftrightarrow$ *vbrelation* r
by *auto*

lemma (**in** *vbrelation*) *vbrelation-vinsertI*[*intro*, *simp*]:
vbrelation (*vinsert* $\langle a, b \rangle$ r)
using *vbrelation-axioms* **by** *auto*

lemma *vbrelation-vinsertD*[*dest*]:
assumes *vbrelation* (*vinsert* $\langle a, b \rangle$ r)
shows *vbrelation* r
using *assms* **by** *auto*

lemma *vbrelation-vunion*: *vbrelation* ($r \cup_\circ s$) $\longleftrightarrow$ *vbrelation* $r \wedge$ *vbrelation* s
by *auto*

lemma *vbrelation-vunionI*:
assumes *vbrelation* r **and** *vbrelation* s
shows *vbrelation* ($r \cup_\circ s$)
using *assms* **by** *auto*

lemma *vbrelation-vunionD*[*dest*]:
assumes *vbrelation* ($r \cup_\circ s$)
shows *vbrelation* r **and** *vbrelation* s
using *assms* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vintersectionI*: *vbrelation* ($r \cap_\circ s$)
using *vbrelation-axioms* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vdiffI*: *vbrelation* ($r -_\circ s$)
using *vbrelation-axioms* **by** *auto*

Connections.

lemma *vbrelation-vempty*: *vbrelation* 0 **by** *auto*

lemma *vbrelation-vsingleton*: *vbrelation* (*set* {⟨*a*, *b*⟩}) **by** *auto*

lemma *vbrelation-vdoubleton*: *vbrelation* (*set* {⟨*a*, *b*⟩, ⟨*c*, *d*⟩}) **by** *auto*

lemma *vbrelation-vid-on*[*simp*]: *vbrelation* (*vid-on A*) **by** *auto*

lemma *vbrelation-vconst-on*[*simp*]: *vbrelation* (*vconst-on A c*) **by** *auto*

lemma *vbrelation-VLambda*[*simp*]: *vbrelation* (*VLambda A f*)
unfolding *VLambda-def* **by** (*intro vbrelationI*) *auto*

global-interpretation *rel-VLambda*: *vbrelation* ‹*VLambda U f*›
by (*rule vbrelation-VLambda*)

lemma *vbrelation-vcomp*:
assumes *vbrelation r* **and** *vbrelation s*
shows *vbrelation* (*r* ∘ₒ *s*)
using *assms* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vconverse*: *vbrelation* ($r^{-1}{}_\circ$)
using *vbrelation-axioms* **by** *clarsimp*

lemma *vbrelation-vlrestriction*[*intro*, *simp*]: *vbrelation* (*r* ↾$^l_\circ$ *A*) **by** *auto*

lemma *vbrelation-vrrestriction*[*intro*, *simp*]: *vbrelation* (*r* ↾$^r_\circ$ *A*) **by** *auto*

lemma *vbrelation-vrestriction*[*intro*, *simp*]: *vbrelation* (*r* ↾$_\circ$ *A*) **by** *auto*

Previous connections.

lemma (**in** *vbrelation*) *vconverse-vconverse*[*simp*]: ($r^{-1}{}_\circ$)$^{-1}{}_\circ$ = *r*
using *vbrelation-axioms* **by** *auto*

lemma *vconverse-mono*[*simp*]:
assumes *vbrelation r* **and** *vbrelation s*
shows $r^{-1}{}_\circ \subseteq_\circ s^{-1}{}_\circ \longleftrightarrow r \subseteq_\circ s$
using *assms* **by** (*force intro*: *vconverse-vunion*)+

lemma *vconverse-inject*[*simp*]:
assumes *vbrelation r* **and** *vbrelation s*
shows $r^{-1}{}_\circ = s^{-1}{}_\circ \longleftrightarrow r = s$
using *assms* **by** *fast*

lemma (**in** *vbrelation*) *vconverse-vsubset-swap-2*:
assumes $r^{-1}{}_\circ \subseteq_\circ s$
shows $r \subseteq_\circ s^{-1}{}_\circ$
using *assms vbrelation-axioms* **by** *auto*

lemma (**in** *vbrelation*) *vlrestriction-vdomain*[*simp*]: *r* ↾$^l_\circ$ $\mathcal{D}_\circ$ *r* = *r*
using *vbrelation-axioms* **by** (*elim vbrelationE*) *auto*

lemma (**in** *vbrelation*) *vrrestriction-vrange*[*simp*]: *r* ↾$^r_\circ$ $\mathcal{R}_\circ$ *r* = *r*
using *vbrelation-axioms* **by** (*elim vbrelationE*) *auto*

Special properties.

lemma *brel-vsubset-vtimes*:
vbrelation r ⟷ *r* ⊆$_\circ$ *set* (*vfst* ‘ *elts r*) ×$_\circ$ *set* (*vsnd* ‘ *elts r*)
by *force*

lemma *vsubset-vtimes-vbrelation*:
assumes $r \subseteq_\circ A \times_\circ B$
shows *vbrelation r*
using *assms* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vintersection-vdomain*:
assumes *vdisjnt* $(\mathcal{D}_\circ\ r)$ $(\mathcal{D}_\circ\ s)$
shows *vdisjnt r s*
proof(*intro vsubset-antisym vsubsetI*)
fix *x* **assume** $x \in_\circ r \cap_\circ s$
then obtain *a b* **where** $\langle a, b\rangle \in_\circ r \cap_\circ s$
by (*metis vbrelationE*1 *vbrelation-vintersectionI*)
with *assms* **show** $x \in_\circ 0$ **by** *auto*
qed *simp*

lemma (**in** *vbrelation*) *vbrelation-vintersection-vrange*:
assumes *vdisjnt* $(\mathcal{R}_\circ\ r)$ $(\mathcal{R}_\circ\ s)$
shows *vdisjnt r s*
proof(*intro vsubset-antisym vsubsetI*)
fix *x* **assume** $x \in_\circ r \cap_\circ s$
then obtain *a b* **where** $\langle a, b\rangle \in_\circ r \cap_\circ s$
by (*metis vbrelationE*1 *vbrelation-vintersectionI*)
with *assms* **show** $x \in_\circ 0$ **by** *auto*
qed *simp*

lemma (**in** *vbrelation*) *vbrelation-vintersection-vfield*:
assumes *vdisjnt* (*vfield r*) (*vfield s*)
shows *vdisjnt r s*
proof(*intro vsubset-antisym vsubsetI*)
fix *x* **assume** $x \in_\circ r \cap_\circ s$
then obtain *a b* **where** $\langle a, b\rangle \in_\circ r \cap_\circ s$
by (*metis vbrelationE*1 *vbrelation-vintersectionI*)
with *assms* **show** $x \in_\circ 0$ **by** *auto*
qed *auto*

lemma (**in** *vbrelation*) *vdomain-vrange-vtimes*: $r \subseteq_\circ \mathcal{D}_\circ\ r \times_\circ \mathcal{R}_\circ\ r$
using *vbrelation* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vsubset-vtimes*:
assumes $\mathcal{D}_\circ\ r \subseteq_\circ A$ **and** $\mathcal{R}_\circ\ r \subseteq_\circ B$
shows $r \subseteq_\circ A \times_\circ B$
proof(*intro vsubsetI*)
fix *x* **assume** *prems*: $x \in_\circ r$
with *vbrelation* **obtain** *a b* **where** *x-def*: $x = \langle a, b\rangle$ **by** *auto*
from *prems* **have** *a*: $a \in_\circ \mathcal{D}_\circ\ r$ **and** *b*: $b \in_\circ \mathcal{R}_\circ\ r$ **unfolding** *x-def* **by** *auto*
with *assms* **have** $a \in_\circ A$ **and** $b \in_\circ B$ **by** *auto*
then show $x \in_\circ A \times_\circ B$ **unfolding** *x-def* **by** *simp*
qed

lemma (**in** *vbrelation*) *vlrestriction-vsubset-vrange*[*intro*, *simp*]:
assumes $\mathcal{D}_\circ\ r \subseteq_\circ A$
shows $r \restriction^l_\circ A = r$
proof(*intro vsubset-antisym*)
show $r \subseteq_\circ r \restriction^l_\circ A$
by (*rule vlrestriction-mono*[*OF assms*, *of r*, *unfolded vlrestriction-vdomain*])
qed *auto*

lemma (**in** *vbrelation*) *vrrestriction-vsubset-vrange*[*intro*, *simp*]:

assumes $\mathcal{R}_\circ\ r \subseteq_\circ B$
shows $r \restriction^r_\circ B = r$
proof(*intro vsubset-antisym*)
show $r \subseteq_\circ r \restriction^r_\circ B$
by (*rule vrrestriction-mono*[*OF assms*, *of r*, *unfolded vrrestriction-vrange*])
qed *auto*

lemma (**in** *vbrelation*) *vbrelation-vcomp-vid-on-left*[*simp*]:
assumes $\mathcal{R}_\circ\ r \subseteq_\circ A$
shows *vid-on A* $\circ_\circ$ $r = r$
using *assms* **by** *auto*

lemma (**in** *vbrelation*) *vbrelation-vcomp-vid-on-right*[*simp*]:
assumes $\mathcal{D}_\circ\ r \subseteq_\circ A$
shows $r \circ_\circ$ *vid-on A* $= r$
using *assms* **by** *auto*

Alternative forms of existing results.

lemmas [*intro*, *simp*] = *vbrelation.vconverse-vconverse*
and [*intro*, *simp*] = *vbrelation.vlrestriction-vsubset-vrange*
and [*intro*, *simp*] = *vbrelation.vrrestriction-vsubset-vrange*

Simple single-valued relation

locale *vsv* = *vbrelation r* **for** *r* +
assumes *vsv*: $[\![\ \langle a, b\rangle \in_\circ r;\ \langle a, c\rangle \in_\circ r\ ]\!] \Longrightarrow b = c$

Rules.

lemmas (**in** *vsv*) [*intro*] = *vsv-axioms*

mk-ide rf *vsv-def*[*unfolded vsv-axioms-def*]
|*intro vsvI*[*intro*]|
|*dest vsvD*[*dest*]|
|*elim vsvE*[*elim*]|

Set operations.

lemma (**in** *vsv*) *vsv-vinsert*[*simp*]:
assumes $a \notin_\circ \mathcal{D}_\circ\ r$
shows *vsv* (*vinsert* $\langle a, b\rangle$ *r*)
using *assms vsv-axioms* **by** *blast*

lemma *vsv-vinsertD*:
assumes *vsv* (*vinsert x r*)
shows *vsv r*
using *assms* **by** (*intro vsvI*) *auto*

lemma *vsv-vunion*[*intro*, *simp*]:
assumes *vsv r* **and** *vsv s* **and** *vdisjnt* $(\mathcal{D}_\circ\ r)$ $(\mathcal{D}_\circ\ s)$
shows *vsv* $(r \cup_\circ s)$
proof
from *assms* **have** *F*: $[\![\ \langle a, b\rangle \in_\circ r;\ \langle a, c\rangle \in_\circ s\ ]\!] \Longrightarrow$ *False* **for** *a b c*
using *elts-0* **by** *blast*
fix *a b c* **assume** $\langle a, b\rangle \in_\circ r \cup_\circ s$ **and** $\langle a, c\rangle \in_\circ r \cup_\circ s$
then consider
$\langle a, b\rangle \in_\circ r \wedge \langle a, c\rangle \in_\circ r$
| $\langle a, b\rangle \in_\circ r \wedge \langle a, c\rangle \in_\circ s$
| $\langle a, b\rangle \in_\circ s \wedge \langle a, c\rangle \in_\circ r$
| $\langle a, b\rangle \in_\circ s \wedge \langle a, c\rangle \in_\circ s$

by *blast*
then show $b = c$ **using** *assms* **by** *cases auto*
qed (*use assms* **in** *auto*)

lemma (**in** *vsv*) *vsv-vintersection*[*intro*, *simp*]: *vsv* $(r \cap_\circ s)$
using *vsv-axioms* **by** *blast*

lemma (**in** *vsv*) *vsv-vdiff*[*intro*, *simp*]: *vsv* $(r -_\circ s)$ **using** *vsv-axioms* **by** *blast*

Connections.

lemma *vsv-vempty*[*simp*]: *vsv* 0 **by** *auto*

lemma *vsv-vsingleton*[*simp*]: *vsv* (*set* $\{\langle a, b\rangle\}$) **by** *auto*

global-interpretation *rel-vsingleton*: *vsv* ‹*set* $\{\langle a, b\rangle\}$›
by (*rule vsv-vsingleton*)

lemma *vsv-vdoubleton*:
assumes $a \neq c$
shows *vsv* (*set* $\{\langle a, b\rangle, \langle c, d\rangle\}$)
using *assms* **by** (*auto simp*: *vinsert-set-insert-eq*)

lemma *vsv-vid-on*[*simp*]: *vsv* (*vid-on* A) **by** *auto*

lemma *vsv-vconst-on*[*simp*]: *vsv* (*vconst-on* A c) **by** *auto*

lemma *vsv-VLambda*[*simp*]: *vsv* $(\lambda a \in_\circ A.\ f\ a)$ **by** *auto*

global-interpretation *rel-VLambda*: *vsv* ‹$(\lambda a \in_\circ A.\ f\ a)$›
unfolding *VLambda-def* **by** (*intro vsvI*) *auto*

lemma *vsv-vcomp*:
assumes *vsv* r **and** *vsv* s
shows *vsv* $(r \circ_\circ s)$
using *assms*
by (*intro vsvI*; *elim vsvE*) (*simp add*: *vbrelation-vcomp*, *metis vcompD*)

lemma (**in** *vsv*) *vsv-vlrestriction*[*intro*, *simp*]: *vsv* $(r \restriction^l_\circ A)$
using *vsv-axioms* **by** *blast*

lemma (**in** *vsv*) *vsv-vrrestriction*[*intro*, *simp*]: *vsv* $(r \restriction^r_\circ A)$
using *vsv-axioms* **by** *blast*

lemma (**in** *vsv*) *vsv-vrestriction*[*intro*, *simp*]: *vsv* $(r \restriction_\circ A)$
using *vsv-axioms* **by** *blast*

Special properties.

lemma *small-vsv*[*simp*]: *small* $\{f.\ vsv\ f \wedge \mathcal{D}_\circ\ f = A \wedge \mathcal{R}_\circ\ f \subseteq_\circ B\}$
proof–
have *small* $\{f.\ f \subseteq_\circ A \times_\circ B\}$ **by** (*auto simp*: *small-iff*)
moreover have $\{f.\ vsv\ f \wedge \mathcal{D}_\circ\ f = A \wedge \mathcal{R}_\circ\ f \subseteq_\circ B\} \subseteq \{f.\ f \subseteq_\circ A \times_\circ B\}$
by *auto*
ultimately show *small* $\{f.\ vsv\ f \wedge \mathcal{D}_\circ\ f = A \wedge \mathcal{R}_\circ\ f \subseteq_\circ B\}$
by (*auto simp*: *smaller-than-small*)
qed

context *vsv*
begin

lemma *vsv-ex*1:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows $\exists!b.\ \langle a,\ b\rangle \in_\circ r$
using *vsv-axioms assms* **by** *auto*

lemma *vsv-ex*1-*app*1:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows $b = r(\!|a|\!) \longleftrightarrow \langle a,\ b\rangle \in_\circ r$
proof
assume *b-def*: $b = r(\!|a|\!)$ **show** $\langle a,\ b\rangle \in_\circ r$
unfolding *app-def b-def* **by** (*rule theI'*) (*rule vsv-ex*1[*OF assms*])
next
assume [*simp*]: $\langle a,\ b\rangle \in_\circ r$
from *assms vsv-axioms vsvD* **have** *THE-b*: (*THE y*. $\langle a,\ y\rangle \in_\circ r$) = b **by** *auto*
show $b = r(\!|a|\!)$ **unfolding** *app-def THE-b*[*symmetric*] **by** (*rule refl*)
qed

lemma *vsv-ex*1-*app*2[*iff*]:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows $r(\!|a|\!) = b \longleftrightarrow \langle a,\ b\rangle \in_\circ r$
using *vsv-ex*1-*app*1[*OF assms*] **by** *auto*

lemma *vsv-appI*[*intro*, *simp*]:
assumes $\langle a,\ b\rangle \in_\circ r$
shows $r(\!|a|\!) = b$
using *assms* **by** (*subgoal-tac* ‹$a \in_\circ \mathcal{D}_\circ\ r$›) *auto*

lemma *vsv-appE*:
assumes $r(\!|a|\!) = b$ **and** $a \in_\circ \mathcal{D}_\circ\ r$ **and** $\langle a,\ b\rangle \in_\circ r \Longrightarrow P$
shows P
using *assms vsv-ex*1-*app*1 **by** *blast*

lemma *vdomain-vrange-is-vempty*: $\mathcal{D}_\circ\ r = 0 \longleftrightarrow \mathcal{R}_\circ\ r = 0$ **by** *fastforce*

lemma *vsv-vrange-vempty*:
assumes $\mathcal{R}_\circ\ r = 0$
shows $r = 0$
using *assms vdomain-vrange-is-vempty vlrestriction-vdomain* **by** *auto*

lemma *vsv-vdomain-vempty-vrange-vempty*:
assumes $\mathcal{D}_\circ\ r \neq 0$
shows $\mathcal{R}_\circ\ r \neq 0$
using *assms* **by** *fastforce*

lemma *vsv-vdomain-vsingleton-vrange-vsingleton*:
assumes $\mathcal{D}_\circ\ r = set\ \{a\}$
obtains b **where** $\mathcal{R}_\circ\ r = set\ \{b\}$
proof–
from *assms* **obtain** b **where** *ab*: $\langle a,\ b\rangle \in_\circ r$ **by** *auto*
then have $\langle a,\ c\rangle \in_\circ r \Longrightarrow c = b$ **for** c **by** (*auto simp*: *vsv*)
moreover with *assms* **have** $\langle b,\ c\rangle \in_\circ r \Longrightarrow c = a$ **for** c **by** *force*
ultimately have $\langle c,\ d\rangle \in_\circ r \Longrightarrow d = b$ **for** $c\ d$
by (*metis app-vdomainI assms vsingletonD*)
with *ab* **have** $\mathcal{R}_\circ\ r = set\ \{b\}$ **by** *blast*
with *that* **show** *?thesis* **by** *simp*
qed

lemma *vsv-vsubset-vimageE*:
assumes $B \subseteq_\circ r \mathbin{`\!`}_\circ A$
obtains C **where** $C \subseteq_\circ A$ **and** $B = r \mathbin{`\!`}_\circ C$
proof–
define C **where** *C-def*: $C = (r^{-1}{}_\circ \mathbin{`\!`}_\circ B) \cap_\circ A$
then have $C \subseteq_\circ A$ **by** *auto*
moreover have $B = r \mathbin{`\!`}_\circ C$
unfolding *C-def*
proof(*intro vsubset-antisym vsubsetI*)
fix b **assume** $b \in_\circ B$
with *assms* **obtain** a **where** $a \in_\circ A$ **and** $\langle a, b\rangle \in_\circ r$
using *app-vimageE vsubsetD* **by** *metis*
then have $a \in_\circ r^{-1}{}_\circ \mathbin{`\!`}_\circ B \cap_\circ A$ **by** (*auto simp*: ‹$b \in_\circ B$›)
then show $b \in_\circ r \mathbin{`\!`}_\circ (r^{-1}{}_\circ \mathbin{`\!`}_\circ B \cap_\circ A)$ **by** (*auto intro*: ‹$\langle a, b\rangle \in_\circ r$›)
qed (*use vsv-axioms* **in** *auto*)
ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemma *vsv-vimage-eqI*[*intro*]:
assumes $a \in_\circ \mathcal{D}_\circ r$ **and** $r(\!|a|\!) = b$ **and** $a \in_\circ A$
shows $b \in_\circ r \mathbin{`\!`}_\circ A$
using *assms*(2)[*unfolded vsv-ex1-app2*[*OF assms*(1)]] *assms*(3) **by** *auto*

lemma *vsv-vimageI1*:
assumes $a \in_\circ \mathcal{D}_\circ r$ **and** $a \in_\circ A$
shows $r(\!|a|\!) \in_\circ r \mathbin{`\!`}_\circ A$
using *assms* **by** (*simp add*: *vsv-vimage-eqI*)

lemma *vsv-vimageI2*:
assumes $a \in_\circ \mathcal{D}_\circ r$
shows $r(\!|a|\!) \in_\circ \mathcal{R}_\circ r$
using *assms* **by** (*blast dest*: *vsv-ex1-app1*)

lemma *vsv-vimageI2′*:
assumes $b = r(\!|a|\!)$ **and** $a \in_\circ \mathcal{D}_\circ r$
shows $b \in_\circ \mathcal{R}_\circ r$
using *assms* **by** (*blast dest*: *vsv-ex1-app1*)

lemma *vsv-value*:
assumes $a \in_\circ \mathcal{D}_\circ r$
obtains b **where** $r(\!|a|\!) = b$ **and** $b \in_\circ \mathcal{R}_\circ r$
using *assms* **by** (*blast dest*: *vsv-ex1-app1*)

lemma *vsv-vimageE*:
assumes $b \in_\circ r \mathbin{`\!`}_\circ A$
obtains x **where** $r(\!|x|\!) = b$ **and** $x \in_\circ A$
using *assms vsv-axioms vsv-ex1-app2* **by** *blast*

lemma *vsv-vimage-iff*: $b \in_\circ r \mathbin{`\!`}_\circ A \longleftrightarrow (\exists a.\ a \in_\circ A \wedge a \in_\circ \mathcal{D}_\circ r \wedge r(\!|a|\!) = b)$
using *vsv-axioms* **by** (*blast intro*: *vsv-ex1-app1*[*THEN iffD1*])+

lemma *vsv-vimage-vsingleton*:
assumes $a \in_\circ \mathcal{D}_\circ r$
shows $r \mathbin{`\!`}_\circ set\ \{a\} = set\ \{r(\!|a|\!)\}$
using *assms* **by** *force*

lemma *vsv-vimage-vsubsetI*:
assumes $\bigwedge a.\ [\![\ a \in_\circ A;\ a \in_\circ \mathcal{D}_\circ r\ ]\!] \Longrightarrow r(\!|a|\!) \in_\circ B$

shows r ‘$_\circ$ $A \subseteq_\circ B$
using *assms* **by** (*metis vsv-vimage-iff vsubsetI*)

lemma *vsv-image-vsubset-iff*:
r ‘$_\circ$ $A \subseteq_\circ B \longleftrightarrow (\forall\, a{\in_\circ}A.\ a \in_\circ \mathcal{D}_\circ\ r \longrightarrow r(\!|a|\!) \in_\circ B)$
by (*auto simp*: *vsv-vimage-iff*)

lemma *vsv-vimage-vinsert*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows r ‘$_\circ$ *vinsert* $a\ A$ = *vinsert* $(r(\!|a|\!))\ (r$ ‘$_\circ$ $A)$
using *assms vsv-vimage-iff* **by** (*intro vsubset-antisym vsubsetI*) *auto*

lemma *vsv-vinsert-vimage*[*intro*, *simp*]:
assumes $a \in_\circ \mathcal{D}_\circ\ r$ **and** $a \in_\circ A$
shows *vinsert* $(r(\!|a|\!))\ (r$ ‘$_\circ$ $A) = r$ ‘$_\circ$ A
using *assms* **by** *auto*

lemma *vsv-is-VLambda*[*simp*]: $(\lambda x{\in_\circ}\mathcal{D}_\circ\ r.\ r(\!|x|\!)) = r$
using *vbrelation*
by (*auto simp*: *app-vdomainI VLambda-iff*2 *intro*!: *vsubset-antisym*)

lemma *vsv-is-VLambda-on-vlrestriction*[*intro*, *simp*]:
assumes $A \subseteq_\circ \mathcal{D}_\circ\ r$
shows $(\lambda x{\in_\circ}A.\ r(\!|x|\!)) = r \restriction^l_\circ A$
using *assms* **by** (*force simp*: *VLambda-iff*2)+

lemma *pairwise-vimageI*:
assumes $\bigwedge x\ y.$
$[\![\ x \in_\circ \mathcal{D}_\circ\ r;\ y \in_\circ \mathcal{D}_\circ\ r;\ x \neq y;\ r(\!|x|\!) \neq r(\!|y|\!)\ ]\!] \Longrightarrow P\ (r(\!|x|\!))\ (r(\!|y|\!))$
shows *vpairwise* $P\ (\mathcal{R}_\circ\ r)$
by (*intro vpairwiseI*) (*metis assms app-vdomainI app-vrangeE vsv-appI*)

lemma *vsv-vrange-vsubset*:
assumes $\bigwedge x.\ x \in_\circ \mathcal{D}_\circ\ r \Longrightarrow r(\!|x|\!) \in_\circ A$
shows $\mathcal{R}_\circ\ r \subseteq_\circ A$
using *assms* **by** *fastforce*

lemma *vsv-vlrestriction-vinsert*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows $r \restriction^l_\circ$ *vinsert* $a\ A$ = *vinsert* $\langle a,\ r(\!|a|\!)\rangle\ (r \restriction^l_\circ A)$
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

end

lemma *vsv-eqI*:
assumes *vsv* r
and *vsv* s
and $\mathcal{D}_\circ\ r = \mathcal{D}_\circ\ s$
and $\bigwedge a.\ a \in_\circ \mathcal{D}_\circ\ r \Longrightarrow r(\!|a|\!) = s(\!|a|\!)$
shows $r = s$
proof(*intro vsubset-antisym vsubsetI*)
interpret r: *vsv* r **by** (*rule assms*(1))
interpret s: *vsv* s **by** (*rule assms*(2))
fix x **assume** $x \in_\circ r$
then obtain $a\ b$ **where** x-*def*[*simp*]: $x = \langle a,\ b\rangle$ **and** $a \in_\circ \mathcal{D}_\circ\ r$
by (*elim r.vbrelation-vinE*)
with ‹$x \in_\circ r$› **have** $r(\!|a|\!) = b$ **by** *simp*
with *assms* ‹$a \in_\circ \mathcal{D}_\circ\ r$› **show** $x \in_\circ s$ **by** *fastforce*

next
interpret r: *vsv r* **by** (*rule assms*(1))
interpret s: *vsv s* **by** (*rule assms*(2))
fix x **assume** $x \in_\circ s$
with *assms*(2) **obtain** a b **where** *x-def*[*simp*]: $x = \langle a, b \rangle$ **and** $a \in_\circ \mathcal{D}_\circ s$
by (*elim vsvE*) *blast*
with *assms* ‹$x \in_\circ s$› **have** $s(\!|a|\!) = b$ **by** *blast*
with *assms* ‹$a \in_\circ \mathcal{D}_\circ s$› **show** $x \in_\circ r$ **by** *fastforce*
qed

lemma (**in** *vsv*) *vsv-VLambda-cong*:
assumes $\bigwedge a.\ a \in_\circ \mathcal{D}_\circ r \Longrightarrow r(\!|a|\!) = f\ a$
shows $(\lambda a \in_\circ \mathcal{D}_\circ r.\ f\ a) = r$
proof(*rule vsv-eqI*[*symmetric*])
show $\mathcal{D}_\circ\ r = \mathcal{D}_\circ\ (\mathit{VLambda}\ (\mathcal{D}_\circ\ r)\ f)$ **by** *simp*
fix a **assume** a: $a \in_\circ \mathcal{D}_\circ r$
then show $r(\!|a|\!) = \mathit{VLambda}\ (\mathcal{D}_\circ\ r)\ f\ (\!|a|\!)$ **using** *assms*(1)[*OF a*] **by** *auto*
qed *auto*

lemma *Axiom-of-Choice*:
obtains f **where** $\bigwedge x.\ x \in_\circ A \Longrightarrow x \neq 0 \Longrightarrow f(\!|x|\!) \in_\circ x$ **and** *vsv f*
proof–
obtain f **where** f: $x \in_\circ A \Longrightarrow x \neq 0 \Longrightarrow f(\!|x|\!) \in_\circ x$ **for** x
by (*metis beta vemptyE*)
define f' **where** $f' = (\lambda x \in_\circ A.\ f(\!|x|\!))$
have $x \in_\circ A \Longrightarrow x \neq 0 \Longrightarrow f'(\!|x|\!) \in_\circ x$ **for** x
unfolding f'-*def* **using** f **by** *simp*
moreover have *vsv* f' **unfolding** f'-*def* **by** *simp*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemma *VLambda-eqI*:
assumes $X = Y$ **and** $\bigwedge x.\ x \in_\circ X \Longrightarrow f\ x = g\ x$
shows $(\lambda x \in_\circ X.\ f\ x) = (\lambda y \in_\circ Y.\ g\ y)$
proof(*rule vsv-eqI*, *unfold vdomain-VLambda*; (*intro assms*(1) *vsv-VLambda*)*?*)
fix x **assume** $x \in_\circ X$
with *assms* **show** *VLambda X f*$(\!|x|\!)$ = *VLambda Y g*$(\!|x|\!)$ **by** *simp*
qed

lemma *VLambda-vsingleton-def*: $(\lambda i \in_\circ set\ \{j\}.\ f\ i) = (\lambda i \in_\circ set\ \{j\}.\ f\ j)$ **by** *auto*

Alternative forms of the available results.

lemmas [*iff*] = *vsv.vsv-ex1-app2*
and [*intro*, *simp*] = *vsv.vsv-appI*
and [*elim*] = *vsv.vsv-appE*
and [*intro*] = *vsv.vsv-vimage-eqI*
and [*simp*] = *vsv.vsv-vinsert-vimage*
and [*intro*] = *vsv.vsv-is-VLambda-on-vlrestriction*
and [*simp*] = *vsv.vsv-is-VLambda*
and [*intro*, *simp*] = *vsv.vsv-vintersection*
and [*intro*, *simp*] = *vsv.vsv-vdiff*
and [*intro*, *simp*] = *vsv.vsv-vlrestriction*
and [*intro*, *simp*] = *vsv.vsv-vrrestriction*
and [*intro*, *simp*] = *vsv.vsv-vrestriction*

Specialization of existing properties to single-valued relations.

Identity relation.

lemma *vid-on-eq-atI*[*intro*, *simp*]:
assumes $a = b$ **and** $a \in_{\circ} A$
shows *vid-on* A $(\!|a|\!) = b$
using *assms* **by** *auto*

lemma *vid-on-atI*[*intro*, *simp*]:
assumes $a \in_{\circ} A$
shows *vid-on* A $(\!|a|\!) = a$
using *assms* **by** *auto*

lemma *vid-on-at-iff*[*intro*, *simp*]:
assumes $a \in_{\circ} A$
shows *vid-on* A $(\!|a|\!) = b \longleftrightarrow a = b$
using *assms* **by** *auto*

Constant function.

lemma *vconst-on-atI*[*simp*]:
assumes $a \in_{\circ} A$
shows *vconst-on* A c $(\!|a|\!) = c$
using *assms* **by** *auto*

Composition.

lemma *vcomp-atI*[*intro*, *simp*]:
assumes *vsv* r
and *vsv* s
and $a \in_{\circ} \mathcal{D}_{\circ}\ r$
and $b \in_{\circ} \mathcal{D}_{\circ}\ s$
and $s(\!|b|\!) = c$
and $r(\!|a|\!) = b$
shows $(s \circ_{\circ} r)(\!|a|\!) = c$
using *assms* **by** (*auto simp*: *app-invimageI intro*!: *vsv-vcomp*)

lemma *vcomp-atD*[*dest*]:
assumes $(s \circ_{\circ} r)(\!|a|\!) = c$
and *vsv* r
and *vsv* s
and $a \in_{\circ} \mathcal{D}_{\circ}\ r$
and $r(\!|a|\!) \in_{\circ} \mathcal{D}_{\circ}\ s$
shows $\exists\, b.\ s(\!|b|\!) = c \wedge r(\!|a|\!) = b$
using *assms* **by** (*metis vcomp-atI*)

lemma *vcomp-atE*1:
assumes $(s \circ_{\circ} r)(\!|a|\!) = c$
and *vsv* r
and *vsv* s
and $a \in_{\circ} \mathcal{D}_{\circ}\ r$
and $r(\!|a|\!) \in_{\circ} \mathcal{D}_{\circ}\ s$
and $\exists\, b.\ s(\!|b|\!) = c \wedge r(\!|a|\!) = b \Longrightarrow P$
shows P
using *assms assms vcomp-atD* **by** *blast*

lemma *vcomp-atE*[*elim*]:
assumes $(s \circ_{\circ} r)(\!|a|\!) = c$
and *vsv* r
and *vsv* s
and $a \in_{\circ} \mathcal{D}_{\circ}\ r$
and $r(\!|a|\!) \in_{\circ} \mathcal{D}_{\circ}\ s$
obtains b **where** $r(\!|a|\!) = b$ **and** $s(\!|b|\!) = c$

using *assms that* **by** (*force elim*!: *vcomp-atE*1)

lemma *vsv-vcomp-at*[*simp*]:
assumes *vsv r* **and** *vsv s* **and** $a \in_{\circ} \mathcal{D}_{\circ}\ r$ **and** $r(\!|a|\!) \in_{\circ} \mathcal{D}_{\circ}\ s$
shows $(s \circ_{\circ} r)(\!|a|\!) = s(\!|r(\!|a|\!)|\!)$
using *assms* **by** *auto*

context *vsv*
begin

Converse relation.

lemma *vconverse-atI*[*intro*]:
assumes $a \in_{\circ} \mathcal{D}_{\circ}\ r$ **and** $r(\!|a|\!) = b$
shows $\langle b,\ a\rangle \in_{\circ} r^{-1}{}_{\circ}$
using *assms* **by** *auto*

lemma *vconverse-atD*[*dest*]:
assumes $\langle b,\ a\rangle \in_{\circ} r^{-1}{}_{\circ}$
shows $r(\!|a|\!) = b$
using *assms* **by** *auto*

lemma *vconverse-atE*[*elim*]:
assumes $\langle b,\ a\rangle \in_{\circ} r^{-1}{}_{\circ}$ **and** $r(\!|a|\!) = b \Longrightarrow P$
shows *P*
using *assms* **by** *auto*

lemma *vconverse-iff*:
assumes $a \in_{\circ} \mathcal{D}_{\circ}\ r$
shows $\langle b,\ a\rangle \in_{\circ} r^{-1}{}_{\circ} \longleftrightarrow r(\!|a|\!) = b$
using *assms* **by** *auto*

Left restriction.

interpretation *vlrestriction*: *vsv* $\langle r \restriction^{l}_{\circ} A\rangle$ **by** (*rule vsv-vlrestriction*)

lemma *vlrestriction-atI*[*intro*, *simp*]:
assumes $a \in_{\circ} \mathcal{D}_{\circ}\ r$ **and** $a \in_{\circ} A$ **and** $r(\!|a|\!) = b$
shows $(r \restriction^{l}_{\circ} A)(\!|a|\!) = b$
using *assms* **by** (*auto simp*: *vdomain-vlrestriction*)

lemma *vlrestriction-atD*[*dest*]:
assumes $(r \restriction^{l}_{\circ} A)(\!|a|\!) = b$ **and** $a \in_{\circ} \mathcal{D}_{\circ}\ r$ **and** $a \in_{\circ} A$
shows $r(\!|a|\!) = b$
using *assms* **by** (*auto simp*: *vdomain-vlrestriction*)

lemma *vlrestriction-atE*1[*elim*]:
assumes $(r \restriction^{l}_{\circ} A)(\!|a|\!) = b$
and $a \in_{\circ} \mathcal{D}_{\circ}\ r$
and $a \in_{\circ} A$
and $r(\!|a|\!) = b \Longrightarrow P$
shows *P*
using *assms vlrestrictionD* **by** *blast*

lemma *vlrestriction-atE*2[*elim*]:
assumes $x \in_{\circ} r \restriction^{l}_{\circ} A$
obtains *a b* **where** $x = \langle a,\ b\rangle$ **and** $a \in_{\circ} A$ **and** $r(\!|a|\!) = b$
using *assms* **by** *auto*

Right restriction.

interpretation *vrrestriction*: *vsv* ‹$r \restriction^r_\circ A$› **by** (*rule vsv-vrrestriction*)

lemma *vrrestriction-atI*[*intro*, *simp*]:
assumes $a \in_\circ \mathcal{D}_\circ\ r$ **and** $b \in_\circ A$ **and** $r(\!|a|\!) = b$
shows $(r \restriction^r_\circ A)(\!|a|\!) = b$
using *assms* **by** (*auto simp*: *app-vrrestrictionI*)

lemma *vrrestriction-atD*[*dest*]:
assumes $(r \restriction^r_\circ A)(\!|a|\!) = b$ **and** $a \in_\circ r -`_\circ A$
shows $b \in_\circ A$ **and** $r(\!|a|\!) = b$
using *assms* **by** *force+*

lemma *vrrestriction-atE*1[*elim*]:
assumes $(r \restriction^r_\circ A)(\!|a|\!) = b$ **and** $a \in_\circ r -`_\circ A$ **and** $r(\!|a|\!) = b \Longrightarrow P$
shows *P*
using *assms* **by** (*auto simp*: *vrrestriction-atD*(2))

lemma *vrrestriction-atE*2[*elim*]:
assumes $x \in_\circ r \restriction^r_\circ A$
obtains *a b* **where** $x = \langle a, b\rangle$ **and** $b \in_\circ A$ **and** $r(\!|a|\!) = b$
using *assms* **unfolding** *vrrestriction-def* **by** *auto*

Restriction.

interpretation *vrestriction*: *vsv* ‹$r \restriction_\circ A$› **by** (*rule vsv-vrestriction*)

lemma *vlrestriction-app*[*intro*, *simp*]:
assumes $a \in_\circ \mathcal{D}_\circ\ r$ **and** $a \in_\circ A$
shows $(r \restriction^l_\circ A)(\!|a|\!) = r(\!|a|\!)$
using *assms* **by** *auto*

lemma *vrestriction-atD*[*dest*]:
assumes $(r \restriction_\circ A)(\!|a|\!) = b$ **and** $a \in_\circ r -`_\circ A$ **and** $a \in_\circ A$
shows $b \in_\circ A$ **and** $r(\!|a|\!) = b$
proof–
from *assms* **have** $a \in_\circ \mathcal{D}_\circ\ r$ **by** *auto*
then show $r(\!|a|\!) = b$
by
(
metis
assms
*app-invimageD*1
vrrestriction.vlrestriction-atD
vrrestriction-atD(2)
vrrestriction-vlrestriction
)
then show $b \in_\circ A$ **using** *assms*(2) **by** *blast*
qed

lemma *vrestriction-atE*1[*elim*]:
assumes $(r \restriction_\circ A)(\!|a|\!) = b$
and $a \in_\circ r -`_\circ A$
and $a \in_\circ A$
and $r(\!|a|\!) = b \Longrightarrow P$
shows *P*
using *assms vrestriction-atD*(2) **by** *blast*

lemma *vrestriction-atE*2[*elim*]:
assumes $x \in_\circ r \restriction_\circ A$

obtains $a\ b$ **where** $x = \langle a,\ b\rangle$ **and** $a \in_\circ A$ **and** $b \in_\circ A$ **and** $r(\!|a|\!) = b$
using *assms* **unfolding** *vrestriction-def* **by** *clarsimp*

Domain.

lemma *vdomain-atD*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows $\exists\, b{\in_\circ}\mathcal{R}_\circ\ r.\ r(\!|a|\!) = b$
using *assms* **by** (*blast intro*: *vsv-vimageI2*)

lemma *vdomain-atE*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
obtains b **where** $b \in_\circ \mathcal{R}_\circ\ r$ **and** $r(\!|a|\!) = b$
using *assms* **by** *auto*

Range.

lemma *vrange-atD*:
assumes $b \in_\circ \mathcal{R}_\circ\ r$
shows $\exists\, a{\in_\circ}\mathcal{D}_\circ\ r.\ r(\!|a|\!) = b$
using *assms* **by** *auto*

lemma *vrange-atE*:
assumes $b \in_\circ \mathcal{R}_\circ\ r$
obtains a **where** $a \in_\circ \mathcal{D}_\circ\ r$ **and** $r(\!|a|\!) = b$
using *assms* **by** *auto*

Image.

lemma *vimage-set-eq-at*:
$\{b.\ \exists\, a{\in_\circ}A \cap_\circ \mathcal{D}_\circ\ r.\ r(\!|a|\!) = b\} = \{b.\ \exists\, a{\in_\circ}A.\ \langle a,\ b\rangle \in_\circ r\}$
by (*rule subset-antisym*; *rule subsetI*; *unfold mem-Collect-eq*) *auto*

lemma *vimage-small*[*simp*]: *small* $\{b.\ \exists\, a{\in_\circ}A \cap_\circ \mathcal{D}_\circ\ r.\ r(\!|a|\!) = b\}$
unfolding *vimage-set-eq-at* **by** *auto*

lemma *vimage-set-def*: $r\ `_\circ\ A = set\ \{b.\ \exists\, a{\in_\circ}A \cap_\circ \mathcal{D}_\circ\ r.\ r(\!|a|\!) = b\}$
unfolding *vimage-set-eq-at* **by** (*simp add*: *app-vimage-set-def*)

lemma *vimage-set-iff*: $b \in_\circ r\ `_\circ\ A \longleftrightarrow (\exists\, a{\in_\circ}A \cap_\circ \mathcal{D}_\circ\ r.\ r(\!|a|\!) = b)$
unfolding *vimage-set-eq-at* **using** *vsv-vimage-iff* **by** *auto*

Further derived results.

lemma *vimage-image*:
assumes $A \subseteq_\circ \mathcal{D}_\circ\ r$
shows *elts* $(r\ `_\circ\ A) = (\lambda x.\ r(\!|x|\!))\ `\ (elts\ A)$
using *vimage-def assms small-elts* **by** *blast*

lemma *vsv-vinsert-match-appI*[*intro*, *simp*]:
assumes $a \notin_\circ \mathcal{D}_\circ\ r$
shows *vinsert* $\langle a,\ b\rangle\ r\ (\!|a|\!) = b$
using *assms vsv-axioms* **by** *simp*

lemma *vsv-vinsert-no-match-appI*:
assumes $a \notin_\circ \mathcal{D}_\circ\ r$ **and** $c \in_\circ \mathcal{D}_\circ\ r$ **and** $r\ (\!|c|\!) = d$
shows *vinsert* $\langle a,\ b\rangle\ r\ (\!|c|\!) = d$
using *assms vsv-axioms* **by** *simp*

lemma *vsv-is-vconst-onI*:
assumes $\mathcal{D}_\circ\ r = A$ **and** $\mathcal{R}_\circ\ r = set\ \{a\}$

```
  shows r = vconst-on A a
  unfolding assms(1)[symmetric]
proof(cases ‹𝒟∘ r = 0›)
  case True
  with assms show r = vconst-on (𝒟∘ r) a
    by (auto simp: vdomain-vrange-is-vempty)
next
  case False
  show r = vconst-on (𝒟∘ r) a
  proof(rule vsv-eqI)
    fix a′ assume prems: a′ ∈∘ 𝒟∘ r
    then obtain b where r⦇a′⦈ = b and b ∈∘ ℛ∘ r by auto
    moreover then have b = a unfolding assms by simp
    ultimately show r⦇a′⦈ = vconst-on (𝒟∘ r) a⦇a′⦈ by (simp add: prems)
  qed auto
qed

lemma vsv-vdomain-vrange-vsingleton:
  assumes 𝒟∘ r = set {a} and ℛ∘ r = set{b}
  shows r = set {⟨a, b⟩}
  using assms vsv-is-vconst-onI by auto

end
```

Alternative forms of existing results.

```
lemmas [intro] = vsv.vconverse-atI
  and vsv-vconverse-atD[dest] = vsv.vconverse-atD[rotated]
  and vsv-vconverse-atE[elim] = vsv.vconverse-atE[rotated]
  and [intro, simp] = vsv.vlrestriction-atI
  and vsv-vlrestriction-atD[dest] = vsv.vlrestriction-atD[rotated]
  and vsv-vlrestriction-atE1[elim] = vsv.vlrestriction-atE1[rotated]
  and vsv-vlrestriction-atE2[elim] = vsv.vlrestriction-atE2[rotated]
  and [intro, simp] = vsv.vrrestriction-atI
  and vsv-vrrestriction-atD[dest] = vsv.vrrestriction-atD[rotated]
  and vsv-vrrestriction-atE1[elim] = vsv.vrrestriction-atE1[rotated]
  and vsv-vrrestriction-atE2[elim] = vsv.vrrestriction-atE2[rotated]
  and [intro, simp] = vsv.vlrestriction-app
  and vsv-vrestriction-atD[dest] = vsv.vrestriction-atD[rotated]
  and vsv-vrestriction-atE1[elim] = vsv.vrestriction-atE1[rotated]
  and vsv-vrestriction-atE2[elim] = vsv.vrestriction-atE2[rotated]
  and vsv-vdomain-atD = vsv.vdomain-atD[rotated]
  and vsv-vdomain-atE = vsv.vdomain-atE[rotated]
  and vrange-atD = vsv.vrange-atD[rotated]
  and vrange-atE = vsv.vrange-atE[rotated]
  and vsv-vinsert-match-appI[intro, simp] = vsv.vsv-vinsert-match-appI
  and vsv-vinsert-no-match-appI[intro, simp] =
    vsv.vsv-vinsert-no-match-appI[rotated 3]
```

Corollaries of the alternative forms of existing results.

```
lemma vsv-vlrestriction-vrange:
  assumes vsv s and vsv (r ↾ˡ∘ ℛ∘ s)
  shows vsv (r ∘∘ s)
proof(rule vsvI)
  show vbrelation (r ∘∘ s) by auto
  fix a c c′ assume ⟨a, c⟩ ∈∘ r ∘∘ s ⟨a, c′⟩ ∈∘ r ∘∘ s
  then obtain b and b′
    where ab: ⟨a, b⟩ ∈∘ s
      and bc: ⟨b, c⟩ ∈∘ r
```

and ab': $\langle a,\ b'\rangle \in_\circ s$
and $b'c'$: $\langle b',\ c'\rangle \in_\circ r$
by *clarsimp*
moreover then have $b \in_\circ \mathcal{R}_\circ\ s$ **and** $b' \in_\circ \mathcal{R}_\circ\ s$ **by** *auto*
ultimately have $\langle b,\ c\rangle \in_\circ (r \restriction^l_\circ \mathcal{R}_\circ\ s)$ **and** $\langle b',\ c'\rangle \in_\circ (r \restriction^l_\circ \mathcal{R}_\circ\ s)$ **by** *auto*
with *ab ab'* **have** $\langle a,\ c\rangle \in_\circ (r \restriction^l_\circ \mathcal{R}_\circ\ s) \circ_\circ s$ **and** $\langle a,\ c'\rangle \in_\circ (r \restriction^l_\circ \mathcal{R}_\circ\ s) \circ_\circ s$
by *blast+*
moreover from *assms* **have** *vsv* $((r \restriction^l_\circ \mathcal{R}_\circ\ s) \circ_\circ s)$ **by** (*intro vsv-vcomp*)
ultimately show $c = c'$ **by** *auto*
qed

lemma *vsv-vunion-app-right*[*simp*]:
assumes *vsv r* **and** *vsv s* **and** *vdisjnt* $(\mathcal{D}_\circ\ r)$ $(\mathcal{D}_\circ\ s)$ **and** $x \in_\circ \mathcal{D}_\circ\ s$
shows $(r \cup_\circ s)(\!|x|\!) = s(\!|x|\!)$
using *assms vsubsetD* **by** *blast*

lemma *vsv-vunion-app-left*[*simp*]:
assumes *vsv r* **and** *vsv s* **and** *vdisjnt* $(\mathcal{D}_\circ\ r)$ $(\mathcal{D}_\circ\ s)$ **and** $x \in_\circ \mathcal{D}_\circ\ r$
shows $(r \cup_\circ s)(\!|x|\!) = r(\!|x|\!)$
using *assms vsubsetD* **by** *blast*

One-to-one relation

locale *v11* = *vsv r* **for** *r* +
assumes *vsv-vconverse*: *vsv* $(r^{-1}{}_\circ)$

Rules.

lemmas (**in** *v11*) [*intro*] = *v11-axioms*

mk-ide rf *v11-def*[*unfolded v11-axioms-def*]
|*intro v11I*[*intro*]|
|*dest v11D*[*dest*]|
|*elim v11E*[*elim*]|

Set operations.

lemma (**in** *v11*) *v11-vinsert*[*intro*, *simp*]:
assumes $a \notin_\circ \mathcal{D}_\circ\ r$ **and** $b \notin_\circ \mathcal{R}_\circ\ r$
shows *v11* (*vinsert* $\langle a,\ b\rangle$ *r*)
using *assms v11-axioms*
by (*intro v11I*; *elim v11E*) (*simp-all add*: *vconverse-vinsert vsv.vsv-vinsert*)

lemma *v11-vinsertD*:
assumes *v11* (*vinsert x r*)
shows *v11 r*
using *assms* **by** (*intro v11I*) (*auto simp*: *vsv-vinsertD*)

lemma *v11-vunion*:
assumes *v11 r*
and *v11 s*
and *vdisjnt* $(\mathcal{D}_\circ\ r)$ $(\mathcal{D}_\circ\ s)$
and *vdisjnt* $(\mathcal{R}_\circ\ r)$ $(\mathcal{R}_\circ\ s)$
shows *v11* $(r \cup_\circ s)$
proof
interpret *r*: *v11 r* **by** (*rule assms*(1))
interpret *s*: *v11 s* **by** (*rule assms*(2))
show *vsv* $(r \cup_\circ s)$ **by** (*simp add*: *assms v11D*)
from *assms* **show** *vsv* $((r \cup_\circ s)^{-1}{}_\circ)$
by (*simp add*: *assms r.vsv-vconverse s.vsv-vconverse vconverse-vunion*)

qed

lemma (**in** *v11*) *v11-vintersection*[*intro*, *simp*]: *v11* (*r* ∩$_\circ$ *s*)
using *v11-axioms* **by** (*intro v11I*) *auto*

lemma (**in** *v11*) *v11-vdiff*[*intro*, *simp*]: *v11* (*r* −$_\circ$ *s*)
using *v11-axioms* **by** (*intro v11I*) *auto*

Special properties.

lemma (**in** *vsv*) *vsv-valneq-v11I*:
assumes ⋀*x y*. [[*x* ∈$_\circ$ $\mathcal{D}_\circ$ *r*; *y* ∈$_\circ$ $\mathcal{D}_\circ$ *r*; *x* ≠ *y*]] ⟹ *r*(|*x*|) ≠ *r*(|*y*|)
shows *v11 r*
proof(*intro v11I*)
from *vsv-axioms* **show** *vsv r* **by** *simp*
show *vsv* (*r*$^{-1}{}_\circ$)
by
(
metis
assms
vbrelation-vconverse
vconverse-atD
app-vrangeI
vrange-vconverse
vsvI
)
qed

lemma (**in** *vsv*) *vsv-valeq-v11I*:
assumes ⋀*x y*. [[*x* ∈$_\circ$ $\mathcal{D}_\circ$ *r*; *y* ∈$_\circ$ $\mathcal{D}_\circ$ *r*; *r*(|*x*|) = *r*(|*y*|)]] ⟹ *x* = *y*
shows *v11 r*
using *assms vsv-valneq-v11I* **by** *auto*

Connections.

lemma *v11-vempty*[*simp*]: *v11* 0 **by** (*simp add*: *v11I*)

lemma *v11-vsingleton*[*simp*]: *v11* (*set* {⟨*a*, *b*⟩}) **by** *auto*

lemma *v11-vdoubleton*:
assumes *a* ≠ *c* **and** *b* ≠ *d*
shows *v11* (*set* {⟨*a*, *b*⟩, ⟨*c*, *d*⟩})
using *assms* **by** (*auto simp*: *vinsert-set-insert-eq*)

lemma *v11-vid-on*[*simp*]: *v11* (*vid-on A*) **by** *auto*

lemma *v11-VLambda*[*intro*]:
assumes *inj-on f* (*elts A*)
shows *v11* (λ*a*∈$_\circ$*A*. *f a*)
proof(*rule rel-VLambda.vsv-valneq-v11I*)
fix *x y*
assume *x* ∈$_\circ$ $\mathcal{D}_\circ$ (λ*a*∈$_\circ$*A*. *f a*) **and** *y* ∈$_\circ$ $\mathcal{D}_\circ$ (λ*a*∈$_\circ$*A*. *f a*) **and** *x* ≠ *y*
then have *x* ∈$_\circ$ *A* **and** *y* ∈$_\circ$ *A* **by** *auto*
with *assms* ‹*x* ≠ *y*› **have** *f x* ≠ *f y* **by** (*auto dest*: *inj-onD*)
then show (λ*a*∈$_\circ$*A*. *f a*)(|*x*|) ≠ (λ*a*∈$_\circ$*A*. *f a*)(|*y*|)
by (*simp add*: ‹*x* ∈$_\circ$ *A*› ‹*y* ∈$_\circ$ *A*›)
qed

lemma *v11-vcomp*:
assumes *v11 r* **and** *v11 s*

shows $v11\ (r \circ_\circ s)$
using *assms* **by** (*intro* $v11I$; *elim* $v11E$) (*auto simp*: *vsv-vcomp vconverse-vcomp*)

context $v11$
begin

lemma $v11$-*vconverse*: $v11\ (r^{-1}{}_\circ)$ **by** (*auto simp*: *vsv-axioms vsv-vconverse*)

interpretation $v11\ \langle r^{-1}{}_\circ \rangle$ **by** (*rule* $v11$-*vconverse*)

lemma $v11$-*vlrestriction*[*intro*, *simp*]: $v11\ (r \restriction^l_\circ A)$
using *vsv-vrrestriction* **by** (*auto simp*: *vrrestriction-vconverse*)

lemma $v11$-*vrrestriction*[*intro*, *simp*]: $v11\ (r \restriction^r_\circ A)$
using *vsv-vlrestriction* **by** (*auto simp*: *vlrestriction-vconverse*)

lemma $v11$-*vrestriction*[*intro*, *simp*]: $v11\ (r \restriction_\circ A)$
using *vsv-vrestriction* **by** (*auto simp*: *vrestriction-vconverse*)

end

Further Special properties.

context $v11$
begin

lemma $v11$-*injective*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$ **and** $b \in_\circ \mathcal{D}_\circ\ r$ **and** $r(\!|a|\!) = r(\!|b|\!)$
shows $a = b$
using *assms* $v11$-*axioms* **by** *auto*

lemma $v11$-*double-pair*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$ **and** $a' \in_\circ \mathcal{D}_\circ\ r$ **and** $r(\!|a|\!) = b$ **and** $r(\!|a'|\!) = b'$
shows $a = a' \longleftrightarrow b = b'$
using *assms* $v11$-*axioms* **by** *auto*

lemma $v11$-*vrange-ex*1-*eq*: $b \in_\circ \mathcal{R}_\circ\ r \longleftrightarrow (\exists!a{\in_\circ}\mathcal{D}_\circ\ r.\ r(\!|a|\!) = b)$
proof(*rule iffI*)
from *app-vdomainI* $v11$-*injective* **show**
$b \in_\circ \mathcal{R}_\circ\ r \Longrightarrow \exists!a.\ a \in_\circ \mathcal{D}_\circ\ r \wedge r(\!|a|\!) = b$
by (*elim app-vrangeE*) *auto*
show $\exists!a.\ a \in_\circ \mathcal{D}_\circ\ r \wedge r(\!|a|\!) = b \Longrightarrow b \in_\circ \mathcal{R}_\circ\ r$
by (*auto intro*: *vsv-vimageI*2)
qed

lemma $v11$-*VLambda-iff*: *inj-on f* (*elts A*) $\longleftrightarrow$ $v11\ (\lambda a{\in_\circ}A.\ f\ a)$
by (*rule iffI*; (*intro inj-onI* | *tactic*‹*all-tac*›))
(*auto simp*: $v11$.$v11$-*injective*)

lemma $v11$-*vimage-vpsubset-neq*:
assumes $A \subseteq_\circ \mathcal{D}_\circ\ r$ **and** $B \subseteq_\circ \mathcal{D}_\circ\ r$ **and** $A \neq B$
shows $r\ `_\circ\ A \neq r\ `_\circ\ B$
proof–
from *assms* **obtain** a **where** AB: $a \in_\circ A \vee a \in_\circ B$ **and** nAB: $a \notin_\circ A \vee a \notin_\circ B$
by *auto*
then have $r(\!|a|\!) \notin_\circ r\ `_\circ\ A \vee r(\!|a|\!) \notin_\circ r\ `_\circ\ B$
unfolding *vsv-vimage-iff* **by** (*metis assms*(1,2) $v11$-*injective vsubsetD*)
moreover from AB nAB *assms*(1,2) **have** $r(\!|a|\!) \in_\circ r\ `_\circ\ A \vee r(\!|a|\!) \in_\circ r\ `_\circ\ B$
by *auto*

ultimately show $r \mathbin{`}_\circ A \neq r \mathbin{`}_\circ B$ **by** *clarsimp*
qed

lemma *v11-eq-iff*[*simp*]:
assumes $a \in_\circ \mathcal{D}_\circ\ r$ **and** $b \in_\circ \mathcal{D}_\circ\ r$
shows $r(\!|a|\!) = r(\!|b|\!) \longleftrightarrow a = b$
using *assms v11-double-pair* **by** *blast*

lemma *v11-vcomp-vconverse*: $r^{-1}_\circ \circ_\circ r = \textit{vid-on}\ (\mathcal{D}_\circ\ r)$
proof(*intro vsubset-antisym vsubsetI*)
fix x **assume** *prems*: $x \in_\circ r^{-1}_\circ \circ_\circ r$
then obtain $a\ c$ **where** *x-def*: $x = \langle a, c \rangle$ **and** a: $a \in_\circ \mathcal{D}_\circ\ r$ **by** *auto*
with *prems* **obtain** b **where** $\langle a, b \rangle \in_\circ r$ **and** $\langle b, c \rangle \in_\circ r^{-1}_\circ$ **by** *auto*
with *v11.vsv-vconverse v11-axioms* **have** *ca*: $c = a$ **by** *auto*
from a **show** $x \in_\circ \textit{vid-on}\ (\mathcal{D}_\circ\ r)$ **unfolding** *x-def ca* **by** *auto*
next
fix x **assume** $x \in_\circ \textit{vid-on}\ (\mathcal{D}_\circ\ r)$
then obtain a **where** *x-def*: $x = \langle a, a \rangle$ **and** a: $a \in_\circ \mathcal{D}_\circ\ r$ **by** *clarsimp*
then obtain b **where** $\langle a, b \rangle \in_\circ r$ **by** *auto*
then show $x \in_\circ r^{-1}_\circ \circ_\circ r$ **unfolding** *x-def* **using** a **by** *auto*
qed

lemma *v11-vcomp-vconverse'*: $r \circ_\circ r^{-1}_\circ = \textit{vid-on}\ (\mathcal{R}_\circ\ r)$
using *v11.v11-vcomp-vconverse v11-vconverse* **by** *force*

lemma *v11-vconverse-app*[*simp*]:
assumes $r(\!|a|\!) = b$ **and** $a \in_\circ \mathcal{D}_\circ\ r$
shows $r^{-1}_\circ(\!|b|\!) = a$
using *assms* **by** (*simp add*: *vsv.vconverse-iff vsv-axioms vsv-vconverse*)

lemma *v11-vconverse-app-in-vdomain*:
assumes $y \in_\circ \mathcal{R}_\circ\ r$
shows $r^{-1}_\circ(\!|y|\!) \in_\circ \mathcal{D}_\circ\ r$
using *assms v11-vconverse*
unfolding *vrange-vconverse*[*symmetric*]
by (*auto simp*: *v11-def*)

lemma *v11-app-if-vconverse-app*:
assumes $y \in_\circ \mathcal{R}_\circ\ r$ **and** $r^{-1}_\circ(\!|y|\!) = x$
shows $r(\!|x|\!) = y$
using *assms vsv-vconverse* **by** *auto*

lemma *v11-app-vconverse-app*:
assumes $a \in_\circ \mathcal{R}_\circ\ r$
shows $r(\!|r^{-1}_\circ(\!|a|\!)|\!) = a$
using *assms* **by** (*meson v11-app-if-vconverse-app*)

lemma *v11-vconverse-app-app*:
assumes $a \in_\circ \mathcal{D}_\circ\ r$
shows $r^{-1}_\circ(\!|r(\!|a|\!)|\!) = a$
using *assms v11-vconverse-app* **by** *auto*

end

lemma *v11-vlrestriction-vsubset*:
assumes *v11* $(f \restriction^l_\circ A)$ **and** $B \subseteq_\circ A$
shows *v11* $(f \restriction^l_\circ B)$
proof–

from *assms* **have** *fB-def*: $f \restriction^l_\circ B = (f \restriction^l_\circ A) \restriction^l_\circ B$ **by** *auto*
show *?thesis* **unfolding** *fB-def* **by** (*intro v11.v11-vlrestriction assms*(1))
qed

lemma *v11-vlrestriction-vrange*:
assumes *v11 s* **and** *v11* $(r \restriction^l_\circ \mathcal{R}_\circ s)$
shows *v11* $(r \circ_\circ s)$
proof(*intro v11I*)
interpret *v11 s* **by** (*rule assms*(1))
from *assms vsv-vlrestriction-vrange* **show** *vsv* $(r \circ_\circ s)$
by (*simp add*: *v11.axioms*(1))
show *vsv* $((r \circ_\circ s)^{-1}_\circ)$
unfolding *vconverse-vcomp*
proof(*rule vsvI*)
fix *a c c'* **assume** $\langle a, c\rangle \in_\circ s^{-1}_\circ \circ_\circ r^{-1}_\circ \langle a, c'\rangle \in_\circ s^{-1}_\circ \circ_\circ r^{-1}_\circ$
then obtain *b* **and** *b'*
where $\langle b, a\rangle \in_\circ r$
and *bc*: $\langle c, b\rangle \in_\circ s$
and $\langle b', a\rangle \in_\circ r$
and *b'c'*: $\langle c', b'\rangle \in_\circ s$
by *auto*
moreover then have $b \in_\circ \mathcal{R}_\circ s$ **and** $b' \in_\circ \mathcal{R}_\circ s$ **by** *auto*
ultimately have $\langle b, a\rangle \in_\circ (r \restriction^l_\circ \mathcal{R}_\circ s)$ **and** $\langle b', a\rangle \in_\circ (r \restriction^l_\circ \mathcal{R}_\circ s)$ **by** *auto*
with *assms*(2) **have** *bb'*: $b = b'$ **by** *auto*
from *assms bc*[*unfolded bb'*] *b'c'* **show** $c = c'$ **by** *auto*
qed *auto*
qed

lemma *v11-vlrestriction-vcomp*:
assumes *v11* $(f \restriction^l_\circ A)$ **and** *v11* $(g \restriction^l_\circ (f `_\circ A))$
shows *v11* $((g \circ_\circ f) \restriction^l_\circ A)$
using *assms v11-vlrestriction-vrange* **by** (*auto simp*: *assms*(2) *app-vimage-def*)

Alternative forms of existing results.

lemmas [*intro*, *simp*] = *v11.v11-vinsert*
and [*intro*, *simp*] = *v11.v11-vintersection*
and [*intro*, *simp*] = *v11.v11-vdiff*
and [*intro*, *simp*] = *v11.v11-vrrestriction*
and [*intro*, *simp*] = *v11.v11-vlrestriction*
and [*intro*, *simp*] = *v11.v11-vrestriction*
and [*intro*] = *v11.v11-vimage-vpsubset-neq*

2.4.5 Tools: *mk-VLambda*

ML<

```
(* low level unfold *)
(*Designed based on an algorithm from HOL-Types_To_Sets/unoverload_def.ML*)
fun pure_unfold ctxt thms = ctxt
  |>
    (
      thms
      |> Conv.rewrs_conv
      |> Conv.try_conv
      |> K
      |> Conv.top_conv
    )
  |> Conv.fconv_rule;
```

```
val msg_args = "mk_VLambda: invalid arguments"

val vsv_VLambda_thm = @{thm vsv_VLambda};
val vsv_VLambda_match = vsv_VLambda_thm
  |> Thm.full_prop_of
  |> HOLogic.dest_Trueprop
  |> dest_comb
  |> #2;

val vdomain_VLambda_thm = @{thm vdomain_VLambda};
val vdomain_VLambda_match = vdomain_VLambda_thm
  |> Thm.full_prop_of
  |> HOLogic.dest_Trueprop
  |> HOLogic.dest_eq
  |> #1
  |> dest_comb
  |> #2;

val app_VLambda_thm = @{thm ZFC_Cardinals.beta};
val app_VLambda_match = app_VLambda_thm
  |> Thm.concl_of
  |> HOLogic.dest_Trueprop
  |> HOLogic.dest_eq
  |> #1
  |> strip_comb
  |> #2
  |> hd;

fun mk_VLabmda_thm match_t match_thm ctxt thm =
  let
    val thm_ct = Thm.cprop_of thm
    val (_, rhs_ct) = Thm.dest_equals thm_ct
      handle TERM ("dest_equals", _) => error msg_args
    val insts = Thm.match (Thm.cterm_of ctxt match_t, rhs_ct)
      handle Pattern.MATCH => error msg_args
  in
    match_thm
    |> Drule.instantiate_normalize insts
    |> pure_unfold ctxt (thm |> Thm.symmetric |> single)
  end;

val mk_VLambda_vsv =
  mk_VLabmda_thm vsv_VLambda_match vsv_VLambda_thm;
val mk_VLambda_vdomain =
  mk_VLabmda_thm vdomain_VLambda_match vdomain_VLambda_thm;
val mk_VLambda_app =
  mk_VLabmda_thm app_VLambda_match app_VLambda_thm;

val mk_VLambda_parser = Parse.thm --
  (
    Scan.repeat
      (
        (keyword<|vsv> -- Parse_Spec.opt_thm_name "|") ||
        (keyword<|app> -- Parse_Spec.opt_thm_name "|") ||
        (keyword<|vdomain> -- Parse_Spec.opt_thm_name "|")
      )
  );
```

```
fun process_mk_VLambda_thm mk_VLambda_thm (b, thm) ctxt =
  let
    val thm' = mk_VLambda_thm ctxt thm
    val (res, ctxt') = ctxt
      |> Local_Theory.note (b ||> map (Attrib.check_src ctxt), single thm')
    val _ = Proof_Display.print_theorem (Position.thread_data ()) ctxt' res
  in ctxt' end;

fun folder_mk_VLambda (("|vsv", b), thm) ctxt =
      process_mk_VLambda_thm mk_VLambda_vsv (b, thm) ctxt
  | folder_mk_VLambda (("|app", b), thm) ctxt =
      process_mk_VLambda_thm mk_VLambda_app (b, thm) ctxt
  | folder_mk_VLambda (("|vdomain", b), thm) ctxt =
      process_mk_VLambda_thm mk_VLambda_vdomain (b, thm) ctxt
  | folder_mk_VLambda _ _ = error msg_args

fun process_mk_VLambda (thm, ins) ctxt =
  let
    val _ = ins |> map fst |> has_duplicates op= |> not orelse error msg_args
    val thm' = thm
      |> singleton (Attrib.eval_thms ctxt)
      |> Local_Defs.meta_rewrite_rule ctxt;
  in fold folder_mk_VLambda (map (fn x => (x, thm')) ins) ctxt end;

val _ =
  Outer_Syntax.local_theory
    command_keyword<mk_VLambda>
    "VLambda"
    (mk_VLambda_parser >> process_mk_VLambda);

>
```

2.5 Operations on indexed families of sets

2.5.1 Background

This section presents results about the fundamental operations on the indexed families of sets, such as unions and intersections of the indexed families of sets, disjoint unions and infinite Cartesian products.

Certain elements of the content of this section were inspired by elements of the content of [50]. However, as previously, many other results were ported (with amendments) from the main library of Isabelle/HOL.

abbreviation (*input*) *imVLambda* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  where *imVLambda A f* $\equiv (\lambda a{\in_\circ}A.\ f\ a)$ `‘∘ A`

2.5.2 Intersection of an indexed family of sets

syntax -*VIFINTER* :: *pttrn* $\Rightarrow V \Rightarrow V \Rightarrow V$ (‹(3$\bigcap_\circ$-$\in_\circ$-./ -)› [0, 0, 10] 10)

syntax-consts -*VIFINTER* $\rightleftharpoons$ *VInter*

translations $\bigcap_\circ x{\in_\circ}A.\ f \rightleftharpoons$ *CONST VInter* (*CONST imVLambda A* $(\lambda x.\ f)$)

Rules.

lemma *vifintersectionI*[*intro*]:
  assumes $I \neq 0$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow a \in_\circ f\ i$
  shows $a \in_\circ (\bigcap_\circ i{\in_\circ}I.\ f\ i)$
  using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vifintersectionD*[*dest*]:
  assumes $a \in_\circ (\bigcap_\circ i{\in_\circ}I.\ f\ i)$ **and** $i \in_\circ I$
  shows $a \in_\circ f\ i$
  using *assms* **by** *blast*

lemma *vifintersectionE*1[*elim*]:
  assumes $a \in_\circ (\bigcap_\circ i{\in_\circ}I.\ f\ i)$ **and** $a \in_\circ f\ i \Longrightarrow P$ **and** $i \notin_\circ I \Longrightarrow P$
  shows *P*
  using *assms* **by** *blast*

lemma *vifintersectionE*3[*elim*]:
  assumes $a \in_\circ (\bigcap_\circ i{\in_\circ}I.\ f\ i)$
  obtains $\bigwedge i.\ i{\in_\circ}I \Longrightarrow a \in_\circ f\ i$
  using *assms* **by** *blast*

lemma *vifintersectionE*2[*elim*]:
  assumes $a \in_\circ (\bigcap_\circ i{\in_\circ}I.\ f\ i)$
  obtains *i* **where** $i \in_\circ I$ **and** $a \in_\circ f\ i$
  using *assms* **by** (*elim vifintersectionE*3) (*meson assms VInterE*2 *app-vimageE*)

Set operations.

lemma *vifintersection-vempty-is*[*simp*]: $(\bigcap_\circ i{\in_\circ}0.\ f\ i) = 0$ **by** *auto*

lemma *vifintersection-vsingleton-is*[*simp*]: $(\bigcap_\circ i{\in_\circ}set\{i\}.\ f\ i) = f\ i$
  using *elts*-0 **by** *blast*

lemma *vifintersection-vdoubleton-is*[*simp*]: $(\bigcap_\circ i{\in_\circ}set\ \{i, j\}.\ f\ i) = f\ i \cap_\circ f\ j$
  by
    (
      intro vsubset-antisym vsubsetI;

(*elim vifintersectionE*3 | *intro vifintersectionI*)
)
auto

lemma *vifintersection-antimono*1:
assumes $I \neq 0$ **and** $I \subseteq_\circ J$
shows $(\bigcap_\circ j \in_\circ J.\ f\ j) \subseteq_\circ (\bigcap_\circ i \in_\circ I.\ f\ i)$
using *assms* **by** *blast*

lemma *vifintersection-antimono*2:
assumes $I \neq 0$ **and** $I \subseteq_\circ J$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow f\ i \subseteq_\circ g\ i$
shows $(\bigcap_\circ j \in_\circ J.\ f\ j) \subseteq_\circ (\bigcap_\circ i \in_\circ I.\ g\ i)$
using *assms* **by** *blast*

lemma *vifintersection-vintersection*:
assumes $I \neq 0$ **and** $J \neq 0$
shows $(\bigcap_\circ i \in_\circ I.\ f\ i) \cap_\circ (\bigcap_\circ i \in_\circ J.\ f\ i) = (\bigcap_\circ i \in_\circ I \cup_\circ J.\ f\ i)$
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

lemma *vifintersection-vintersection-family*:
assumes $I \neq 0$
shows $(\bigcap_\circ i \in_\circ I.\ A\ i) \cap_\circ (\bigcap_\circ i \in_\circ I.\ B\ i) = (\bigcap_\circ i \in_\circ I.\ A\ i \cap_\circ B\ i)$
using *assms*
by (*intro vsubset-antisym vsubsetI*, *intro vifintersectionI* | *tactic*‹*all-tac*›)
blast+

lemma *vifintersection-vunion*:
assumes $I \neq 0$ **and** $J \neq 0$
shows $(\bigcap_\circ i \in_\circ I.\ f\ i) \cup_\circ (\bigcap_\circ j \in_\circ J.\ g\ j) = (\bigcap_\circ i \in_\circ I.\ \bigcap_\circ j \in_\circ J.\ f\ i \cup_\circ g\ j)$
using *assms* **by** (*blast intro*!: *vsubset-antisym*)

lemma *vifintersection-vinsert-is*[*intro*, *simp*]:
assumes $I \neq 0$
shows $(\bigcap_\circ i \in_\circ vinsert\ j\ I.\ f\ i) = f\ j \cap_\circ (\bigcap_\circ i \in_\circ I.\ f\ i)$
apply(*insert assms*, *intro vsubset-antisym vsubsetI*)
subgoal for *b* **by** (*subgoal-tac* ‹$b \in_\circ f\ j$› ‹$b \in_\circ (\bigcap_\circ i \in_\circ I.\ f\ i)$›) *blast*+
subgoal for *b*
by (*subgoal-tac* ‹$b \in_\circ f\ j$› ‹$b \in_\circ (\bigcap_\circ i \in_\circ I.\ f\ i)$›)
(*blast intro*!: *vsubset-antisym*)+
done

lemma *vifintersection-VPow*:
assumes $I \neq 0$
shows $VPow\ (\bigcap_\circ i \in_\circ I.\ f\ i) = (\bigcap_\circ i \in_\circ I.\ VPow\ (f\ i))$
using *assms* **by** (*auto intro*!: *vsubset-antisym*)

Elementary properties.

lemma *vifintersection-constant*[*intro*, *simp*]:
assumes $I \neq 0$
shows $(\bigcap_\circ y \in_\circ I.\ c) = c$
using *assms* **by** *auto*

lemma *vifintersection-vsubset-iff*:
assumes $I \neq 0$
shows $A \subseteq_\circ (\bigcap_\circ i \in_\circ I.\ f\ i) \longleftrightarrow (\forall\, i \in_\circ I.\ A \subseteq_\circ f\ i)$
using *assms* **by** *blast*

lemma *vifintersection-vsubset-lower*:

assumes $i \in_\circ I$
shows $(\bigcap_\circ i \in_\circ I.\ f\ i) \subseteq_\circ f\ i$
using *assms* **by** *blast*

lemma *vifintersection-vsubset-greatest*:
assumes $I \neq 0$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow A \subseteq_\circ f\ i$
shows $A \subseteq_\circ (\bigcap_\circ i \in_\circ I.\ f\ i)$
using *assms* **by** (*intro vsubsetI vifintersectionI*) *auto*

lemma *vifintersection-vintersection-value*:
assumes $i \in_\circ I$
shows $f\ i \cap_\circ (\bigcap_\circ i \in_\circ I.\ f\ i) = (\bigcap_\circ i \in_\circ I.\ f\ i)$
using *assms* **by** *blast*

lemma *vifintersection-vintersection-single*:
assumes $I \neq 0$
shows $B \cup_\circ (\bigcap_\circ i \in_\circ I.\ A\ i) = (\bigcap_\circ i \in_\circ I.\ B \cup_\circ A\ i)$
by (*insert assms*, *intro vsubset-antisym vsubsetI vifintersectionI*)
blast+

Connections.

lemma *vifintersection-vrange-VLambda*: $(\bigcap_\circ i \in_\circ I.\ f\ i) = \bigcap_\circ (\mathcal{R}_\circ\ (\lambda a \in_\circ I.\ f\ a))$
by (*simp add*: *vimage-VLambda-vrange-rep*)

2.5.3 Union of an indexed family of sets

syntax -*VIFUNION* :: *pttrn* ⇒ *V* ⇒ *V* ⇒ *V* (‹(3⋃∘-∈∘-./ -)› [0, 0, 10] 10)

syntax-consts -*VIFUNION* ⇌ *VUnion*

translations $\bigcup_\circ x \in_\circ A.\ f \rightleftharpoons$ *CONST VUnion* (*CONST imVLambda A* ($\lambda x.\ f$))

Rules.

lemma *vifunion-iff*: $b \in_\circ (\bigcup_\circ i \in_\circ I.\ f\ i) \longleftrightarrow (\exists\, i \in_\circ I.\ b \in_\circ f\ i)$ **by** *force*

lemma *vifunionI*[*intro*]:
assumes $i \in_\circ I$ **and** $a \in_\circ f\ i$
shows $a \in_\circ (\bigcup_\circ i \in_\circ I.\ f\ i)$
using *assms* **by** *force*

lemma *vifunionD*[*dest*]:
assumes $a \in_\circ (\bigcup_\circ i \in_\circ I.\ f\ i)$
shows $\exists\, i \in_\circ I.\ a \in_\circ f\ i$
using *assms* **by** *auto*

lemma *vifunionE*[*elim*!]:
assumes $a \in_\circ (\bigcup_\circ i \in_\circ I.\ f\ i)$ **and** $\bigwedge i.\ [\![\ i \in_\circ I;\ a \in_\circ f\ i\ ]\!] \Longrightarrow R$
shows R
using *assms* **by** *auto*

Set operations.

lemma *vifunion-vempty-family*[*simp*]: $(\bigcup_\circ i \in_\circ I.\ 0) = 0$ **by** *auto*

lemma *vifunion-vsingleton-is*[*simp*]: $(\bigcup_\circ i \in_\circ set\ \{i\}.\ f\ i) = f\ i$ **by** *force*

lemma *vifunion-vsingleton-family*[*simp*]: $(\bigcup_\circ i \in_\circ I.\ set\ \{i\}) = I$ **by** *force*

lemma *vifunion-vdoubleton*: $(\bigcup_\circ i \in_\circ set\ \{i, j\}.\ f\ i) = f\ i \cup_\circ f\ j$

using *VLambda-vinsert vimage-vunion-left*
by (*force simp*: *VLambda-vsingleton simp*: *vinsert-set-insert-eq*)

lemma *vifunion-mono*:
assumes $I \subseteq_\circ J$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow f\ i \subseteq_\circ g\ i$
shows $(\bigcup_\circ i{\in_\circ}I.\ f\ i) \subseteq_\circ (\bigcup_\circ j{\in_\circ}J.\ g\ j)$
using *assms* **by** *force*

lemma *vifunion-vunion-is*: $(\bigcup_\circ i{\in_\circ}I.\ f\ i) \cup_\circ (\bigcup_\circ j{\in_\circ}J.\ f\ j) = (\bigcup_\circ i{\in_\circ}I \cup_\circ J.\ f\ i)$
by *force*

lemma *vifunion-vunion-family*:
$(\bigcup_\circ i{\in_\circ}I.\ f\ i) \cup_\circ (\bigcup_\circ i{\in_\circ}I.\ g\ i) = (\bigcup_\circ i{\in_\circ}I.\ f\ i \cup_\circ g\ i)$
by (*intro vsubset-antisym vsubsetI*; *elim vunionE vifunionE*) *force+*

lemma *vifunion-vintersection*:
$(\bigcup_\circ i{\in_\circ}I.\ f\ i) \cap_\circ (\bigcup_\circ j{\in_\circ}J.\ g\ j) = (\bigcup_\circ i{\in_\circ}I.\ \bigcup_\circ j{\in_\circ}J.\ f\ i \cap_\circ g\ j)$
by (*force simp*: *vrange-VLambda vimage-VLambda-vrange-rep*)

lemma *vifunion-vinsert-is*:
$(\bigcup_\circ i{\in_\circ}vinsert\ j\ I.\ f\ i) = f\ j \cup_\circ (\bigcup_\circ i{\in_\circ}I.\ f\ i)$
by (*force simp*: *vimage-VLambda-vrange-rep*)

lemma *vifunion-VPow*: $(\bigcup_\circ i{\in_\circ}I.\ VPow\ (f\ i)) \subseteq_\circ VPow\ (\bigcup_\circ i{\in_\circ}I.\ f\ i)$ **by** *force*

Elementary properties.

lemma *vifunion-vempty-conv*:
$0 = (\bigcup_\circ i{\in_\circ}I.\ f\ i) \longleftrightarrow (\forall\ i{\in_\circ}I.\ f\ i = 0)$
$(\bigcup_\circ i{\in_\circ}I.\ f\ i) = 0 \longleftrightarrow (\forall\ i{\in_\circ}I.\ f\ i = 0)$
by (*auto simp*: *vrange-VLambda vimage-VLambda-vrange-rep*)

lemma *vifunion-constant*[*simp*]: $(\bigcup_\circ i{\in_\circ}I.\ c) = (if\ I = 0\ then\ 0\ else\ c)$
proof(*intro vsubset-antisym*)
show $(if\ I = 0\ then\ 0\ else\ c) \subseteq_\circ (\bigcup_\circ i{\in_\circ}I.\ c)$
by (*cases* ‹*vdisjnt I I*›) (*auto simp*: *VLambda-vconst-on*)
qed *auto*

lemma *vifunion-upper*:
assumes $i \in_\circ I$
shows $f\ i \subseteq_\circ (\bigcup_\circ i{\in_\circ}I.\ f\ i)$
using *assms* **by** *force*

lemma *vifunion-least*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow f\ i \subseteq_\circ C$
shows $(\bigcup_\circ i{\in_\circ}I.\ f\ i) \subseteq_\circ C$
using *assms* **by** *auto*

lemma *vifunion-absorb*:
assumes $j \in_\circ I$
shows $f\ j \cup_\circ (\bigcup_\circ i{\in_\circ}I.\ f\ i) = (\bigcup_\circ i{\in_\circ}I.\ f\ i)$
using *assms* **by** *force*

lemma *vifunion-vifunion-flatten*:
$(\bigcup_\circ j{\in_\circ}(\bigcup_\circ i{\in_\circ}I.\ f\ i).\ g\ j) = (\bigcup_\circ i{\in_\circ}I.\ \bigcup_\circ j{\in_\circ}f\ i.\ g\ j)$
by (*force simp*: *vrange-VLambda vimage-VLambda-vrange-rep*)

lemma *vifunion-vsubset-iff*: $((\bigcup_\circ i{\in_\circ}I.\ f\ i) \subseteq_\circ B) = (\forall\ i{\in_\circ}I.\ f\ i \subseteq_\circ B)$ **by** *force*

lemma *vifunion-vsingleton-eq-vrange*: $(\bigcup_\circ i \in_\circ I.\ set\ \{f\ i\}) = \mathcal{R}_\circ\ (\lambda a \in_\circ I.\ f\ a)$
by *force*

lemma *vball-vifunion*[*simp*]: $(\forall\, z \in_\circ (\bigcup_\circ i \in_\circ I.\ f\ i).\ P\ z) \longleftrightarrow (\forall\, x \in_\circ I.\ \forall\, z \in_\circ f\ x.\ P\ z)$
by *force*

lemma *vbex-vifunion*[*simp*]: $(\exists\, z \in_\circ (\bigcup_\circ i \in_\circ I.\ f\ i).\ P\ z) \longleftrightarrow (\exists\, x \in_\circ I.\ \exists\, z \in_\circ f\ x.\ P\ z)$
by *force*

lemma *vifunion-vintersection-index-right*[*simp*]: $(\bigcup_\circ C \in_\circ B.\ A \cap_\circ C) = A \cap_\circ \bigcup_\circ B$
by (*force simp*: *vimage-VLambda-vrange-rep*)

lemma *vifunion-vintersection-index-left*[*simp*]: $(\bigcup_\circ C \in_\circ B.\ C \cap_\circ A) = \bigcup_\circ B \cap_\circ A$
by (*force simp*: *vimage-VLambda-vrange-rep*)

lemma *vifunion-vunion-index*[*intro*, *simp*]:
assumes $B \neq 0$
shows $(\bigcap_\circ C \in_\circ B.\ A \cup_\circ C) = A \cup_\circ \bigcap_\circ B$
using *assms*
by
(
(*intro vsubset-antisym vsubsetI*);
(*intro vifintersectionI* | *tactic*‹*all-tac*›)
)
blast+

lemma *vifunion-vintersection-single*: $B \cap_\circ (\bigcup_\circ i \in_\circ I.\ f\ i) = (\bigcup_\circ i \in_\circ I.\ B \cap_\circ f\ i)$
by (*force simp*: *vrange-VLambda vimage-VLambda-vrange-rep*)

lemma *vifunion-vifunion-flip*:
$(\bigcup_\circ b \in_\circ B.\ \bigcup_\circ a \in_\circ A.\ f\ b\ a) = (\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ f\ b\ a)$
proof–
have $x \in_\circ (\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ f\ b\ a)$ **if** $x \in_\circ (\bigcup_\circ b \in_\circ B.\ \bigcup_\circ a \in_\circ A.\ f\ b\ a)$
for $x\ f\ A\ B$
proof–
from *that* **obtain** b **where** b: $b \in_\circ B$ **and** $x\text{-}b$: $x \in_\circ (\bigcup_\circ a \in_\circ A.\ f\ b\ a)$
by *fastforce*
then obtain a **where** a: $a \in_\circ A$ **and** $x\text{-}fba$: $x \in_\circ f\ b\ a$ **by** *fastforce*
show $x \in_\circ (\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ f\ b\ a)$
unfolding *vifunion-iff* **by** (*auto intro*: *a b x-fba*)
qed
then show *?thesis* **by** (*intro vsubset-antisym vsubsetI*) *auto*
qed

Connections.

lemma *vifunion-disjoint*: $(\bigcup_\circ C \cap_\circ A = 0) \longleftrightarrow (\forall\, B \in_\circ C.\ vdisjnt\ B\ A)$
by (*intro iffI*)
(*auto intro*!: *vsubset-antisym simp*: *Sup-upper vdisjnt-vsubset-left*)

lemma *vdisjnt-vifunion-iff*:
vdisjnt A $(\bigcup_\circ i \in_\circ I.\ f\ i) \longleftrightarrow (\forall\, i \in_\circ I.\ vdisjnt\ A\ (f\ i))$
by (*force intro*!: *vsubset-antisym simp*: *vdisjnt-iff*)+

lemma *vifunion-VLambda*: $(\bigcup_\circ i \in_\circ A.\ set\ \{\langle i, f\ i\rangle\}) = (\lambda a \in_\circ A.\ f\ a)$
using *vifunionI* **by** (*intro vsubset-antisym vsubsetI*) *auto*

lemma *vifunion-vrange-VLambda*: $(\bigcup_\circ i \in_\circ I.\ f\ i) = \bigcup_\circ(\mathcal{R}_\circ\ (\lambda a \in_\circ I.\ f\ a))$
using *vimage-VLambda-vrange-rep* **by** *auto*

lemma (in *vsv*) *vsv-vrange-vsubset-vifunion-app*:
assumes $\mathcal{D}_\circ\ r = I$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow r(\!|i|\!) \in_\circ A\ i$
shows $\mathcal{R}_\circ\ r \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$
proof(*intro vsubsetI*)
fix x **assume** $x \in_\circ \mathcal{R}_\circ\ r$
with *assms*(1) **obtain** i **where** *x-def*: $x = r(\!|i|\!)$ **and** *i*: $i \in_\circ I$
by (*metis vrange-atE*)
from *i assms*(2)[*rule-format*, *OF i*] **show** $x \in_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$
unfolding *x-def* **by** (*intro vifunionI*) *auto*
qed

lemma *v11-vlrestriction-vifintersection*:
assumes $I \neq 0$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow v11\ (f \restriction^l_\circ (A\ i))$
shows $v11\ (f \restriction^l_\circ (\bigcap_\circ i \in_\circ I.\ A\ i))$
proof(*intro v11I*)
show *vsv* $(f \restriction^l_\circ \bigcap_\circ ((\lambda a \in_\circ I.\ A\ a)\ `_\circ\ I))$

apply(*subgoal-tac* $\langle \bigwedge i.\ i \in_\circ I \Longrightarrow vsv\ (f \restriction^l_\circ (A\ i))\rangle$)
subgoal by (*insert assms*(1), *intro vsvI*) (*blast intro*!: *vsubset-antisym*)+
subgoal using *assms* **by** *blast*
done
show *vsv* $((f \restriction^l_\circ \bigcap_\circ ((\lambda a \in_\circ I.\ A\ a)\ `_\circ\ I))^{-1}{}_\circ)$
proof(*intro vsvI*)
fix *a b c*
assume *ab*: $\langle a, b\rangle \in_\circ (f \restriction^l_\circ \bigcap_\circ ((\lambda a \in_\circ I.\ A\ a)\ `_\circ\ I))^{-1}{}_\circ$
and *ac*: $\langle a, c\rangle \in_\circ (f \restriction^l_\circ \bigcap_\circ ((\lambda a \in_\circ I.\ A\ a)\ `_\circ\ I))^{-1}{}_\circ$
from *assms*(2) **have** *hyp*: $\bigwedge i.\ i \in_\circ I \Longrightarrow vsv\ ((f \restriction^l_\circ (A\ i))^{-1}{}_\circ)$ **by** *blast*
from *assms*(1) **obtain** i **where** $i \in_\circ I$ **and** $\bigcap_\circ ((\lambda a \in_\circ I.\ A\ a)\ `_\circ\ I) \subseteq_\circ A\ i$
by (*auto intro*!: *vsubset-antisym*)
with *ab ac hyp* $\langle i \in_\circ I\rangle$ **show** $b = c$ **by** *auto*
qed *auto*
qed

2.5.4 Additional simplification rules for indexed families of sets.

Union.

lemma *vifunion-simps*[*simp*]:
$\bigwedge a\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ vinsert\ a\ (B\ i)) =$
$(if\ I{=}0\ then\ 0\ else\ vinsert\ a\ (\bigcup_\circ i \in_\circ I.\ B\ i))$
$\bigwedge A\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ A\ i \cup_\circ B) = ((if\ I{=}0\ then\ 0\ else\ (\bigcup_\circ i \in_\circ I.\ A\ i) \cup_\circ B))$
$\bigwedge A\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ A \cup_\circ B\ i) = ((if\ I{=}0\ then\ 0\ else\ A \cup_\circ (\bigcup_\circ i \in_\circ I.\ B\ i)))$
$\bigwedge A\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ A\ i \cap_\circ B) = ((\bigcup_\circ i \in_\circ I.\ A\ i) \cap_\circ B)$
$\bigwedge A\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ A \cap_\circ B\ i) = (A \cap_\circ (\bigcup_\circ i \in_\circ I.\ B\ i))$
$\bigwedge A\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ A\ i -_\circ B) = ((\bigcup_\circ i \in_\circ I.\ A\ i) -_\circ B)$
$\bigwedge A\ B.\ (\bigcup_\circ i \in_\circ \bigcup_\circ A.\ B\ i) = (\bigcup_\circ y \in_\circ A.\ \bigcup_\circ i \in_\circ y.\ B\ i)$
by
(
force
simp: *vrange-VLambda vimage-VLambda-vrange-rep*
intro!: *vsubset-antisym*
)+

lemma *vifunion-simps-ext*:
$\bigwedge a\ B\ I.\ vinsert\ a\ (\bigcup_\circ i \in_\circ I.\ B\ i) =$
$(if\ I{=}0\ then\ set\ \{a\}\ else\ (\bigcup_\circ i \in_\circ I.\ vinsert\ a\ (B\ i)))$
$\bigwedge A\ B\ I.\ (\bigcup_\circ i \in_\circ I.\ A\ i) \cup_\circ B = (if\ I{=}0\ then\ B\ else\ (\bigcup_\circ i \in_\circ I.\ A\ i \cup_\circ B))$

$\bigwedge A\ B\ I.\ A \cup_\circ (\bigcup_\circ i \in_\circ I.\ B\ i) = (\mathit{if}\ I{=}0\ \mathit{then}\ A\ \mathit{else}\ (\bigcup_\circ i \in_\circ I.\ A \cup_\circ B\ i))$
$\bigwedge A\ B\ I.\ ((\bigcup_\circ i \in_\circ I.\ A\ i) \cap_\circ B) = (\bigcup_\circ i \in_\circ I.\ A\ i \cap_\circ B)$
$\bigwedge A\ B\ I.\ ((\bigcup_\circ i \in_\circ I.\ A\ i) -_\circ B) = (\bigcup_\circ i \in_\circ I.\ A\ i -_\circ B)$
$\bigwedge A\ B.\ (\bigcup_\circ y \in_\circ A.\ \bigcup_\circ i \in_\circ y.\ B\ i) = (\bigcup_\circ i \in_\circ \bigcup_\circ A.\ B\ i)$
by (*auto simp*: *vrange-VLambda*)

lemma *vifunion-vball-vbex-simps*[*simp*]:
$\bigwedge A\ P.\ (\forall\, a \in_\circ \bigcup_\circ A.\ P\ a) \longleftrightarrow (\forall\, y \in_\circ A.\ \forall\, a \in_\circ y.\ P\ a)$
$\bigwedge A\ P.\ (\exists\, a \in_\circ \bigcup_\circ A.\ P\ a) \longleftrightarrow (\exists\, y \in_\circ A.\ \exists\, a \in_\circ y.\ P\ a)$
using *vball-vifunion vbex-vifunion* **by** *auto*

Intersection.

lemma *vifintersection-simps*[*simp*]:
$\bigwedge I\ A\ B.\ (\bigcap_\circ i \in_\circ I.\ A\ i \cap_\circ B) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A\ i) \cap_\circ B)$
$\bigwedge I\ A\ B.\ (\bigcap_\circ i \in_\circ I.\ A \cap_\circ B\ i) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ A \cap_\circ (\bigcap_\circ i \in_\circ I.\ B\ i))$
$\bigwedge I\ A\ B.\ (\bigcap_\circ i \in_\circ I.\ A\ i -_\circ B) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A\ i) -_\circ B)$
$\bigwedge I\ A\ B.\ (\bigcap_\circ i \in_\circ I.\ A -_\circ B\ i) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ A -_\circ (\bigcup_\circ i \in_\circ I.\ B\ i))$
$\bigwedge I\ a\ B.$
$(\bigcap_\circ i \in_\circ I.\ \mathit{vinsert}\ a\ (B\ i)) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ \mathit{vinsert}\ a\ (\bigcap_\circ i \in_\circ I.\ B\ i))$
$\bigwedge I\ A\ B.\ (\bigcap_\circ i \in_\circ I.\ A\ i \cup_\circ B) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ ((\bigcap_\circ i \in_\circ I.\ A\ i) \cup_\circ B))$
$\bigwedge I\ A\ B.\ (\bigcap_\circ i \in_\circ I.\ A \cup_\circ B\ i) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ (A \cup_\circ (\bigcap_\circ i \in_\circ I.\ B\ i)))$
by *force+*

lemma *vifintersection-simps-ext*:
$\bigwedge A\ B\ I.\ (\bigcap_\circ i \in_\circ I.\ A\ i) \cap_\circ B = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A\ i \cap_\circ B))$
$\bigwedge A\ B\ I.\ A \cap_\circ (\bigcap_\circ i \in_\circ I.\ B\ i) = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A \cap_\circ B\ i))$
$\bigwedge A\ B\ I.\ (\bigcap_\circ i \in_\circ I.\ A\ i) -_\circ B = (\mathit{if}\ I = 0\ \mathit{then}\ 0\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A\ i -_\circ B))$
$\bigwedge A\ B\ I.\ A -_\circ (\bigcup_\circ i \in_\circ I.\ B\ i) = (\mathit{if}\ I = 0\ \mathit{then}\ A\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A -_\circ B\ i))$
$\bigwedge a\ B\ I.\ \mathit{vinsert}\ a\ (\bigcap_\circ i \in_\circ I.\ B\ i) =$
$(\mathit{if}\ I = 0\ \mathit{then}\ \mathit{set}\ \{a\}\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ \mathit{vinsert}\ a\ (B\ i)))$
$\bigwedge A\ B\ I.\ ((\bigcap_\circ i \in_\circ I.\ A\ i) \cup_\circ B) = (\mathit{if}\ I = 0\ \mathit{then}\ B\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A\ i \cup_\circ B))$
$\bigwedge A\ B\ I.\ A \cup_\circ (\bigcap_\circ i \in_\circ I.\ B\ i) = (\mathit{if}\ I = 0\ \mathit{then}\ A\ \mathit{else}\ (\bigcap_\circ i \in_\circ I.\ A \cup_\circ B\ i))$
using *vifintersection-simps* **by** *auto*

2.5.5 Knowledge transfer: union and intersection of indexed families

lemma *SUP-vifunion*: $(\mathit{SUP}\ \xi \in \mathit{elts}\ \alpha.\ A\ \xi) = (\bigcup_\circ \xi \in_\circ \alpha.\ A\ \xi)$
by (*simp add*: *vimage-VLambda-vrange-rep vrange-VLambda*)

lemma *INF-vifintersection*: $(\mathit{INF}\ \xi \in \mathit{elts}\ \alpha.\ A\ \xi) = (\bigcap_\circ \xi \in_\circ \alpha.\ A\ \xi)$
by (*simp add*: *vimage-VLambda-vrange-rep vrange-VLambda*)

lemmas *Ord-induct3'*[*consumes* 1, *case-names* 0 *succ Limit*, *induct type*: V] =
Ord-induct3[*unfolded SUP-vifunion*]

lemma *Limit-vifunion-def*[*simp*]:
assumes *Limit* α
shows $(\bigcup_\circ \xi \in_\circ \alpha.\ \xi) = \alpha$
using *assms* **unfolding** *SUP-vifunion*[*symmetric*] **by** *simp*

lemmas-with[*unfolded SUP-vifunion INF-vifintersection*]:
TC = *ZFC-Cardinals.TC*
and *rank-Sup* = *ZFC-Cardinals.rank-Sup*
and *TC-def* = *ZFC-Cardinals.TC-def*
and *Ord-equality* = *ZFC-in-HOL.Ord-equality*
and *Aleph-Limit* = *ZFC-Cardinals.Aleph-Limit*
and *rank* = *ZFC-Cardinals.rank*
and *Vset* = *ZFC-in-HOL.Vset*

and *mult = Kirby.mult*
and *Aleph-def = ZFC-Cardinals.Aleph-def*
and *times-V-def = Kirby.times-V-def*
and *mult-Limit = Kirby.mult-Limit*
and *Vfrom = ZFC-in-HOL.Vfrom*
and *Vfrom-def = ZFC-in-HOL.Vfrom-def*
and *rank-def = ZFC-Cardinals.rank-def*
and *add-Limit = Kirby.add-Limit*
and *Limit-Vfrom-eq = ZFC-in-HOL.Limit-Vfrom-eq*
and *VSigma-def = ZFC-Cardinals.VSigma-def*
and *add-Sup-distrib-id = Kirby.add-Sup-distrib-id*
and *Limit-add-Sup-distrib = Kirby.Limit-add-Sup-distrib*
and *TC-mult = Kirby.TC-mult*
and *add-Sup-distrib = Kirby.add-Sup-distrib*

2.5.6 Disjoint union

See the main library of *ZFC in HOL* for further information and elementary properties.

syntax *-VSIGMA* :: *pttrn* ⇒ *V* ⇒ *V* ⇒ *V* (‹(3∐∘-∈∘-./ -)› [0, 0, 10] 10)

syntax-consts *-VSIGMA* ⇌ *VSigma*

translations ∐∘*i*∈∘*I*. *A* ⇌ *CONST VSigma I* (λ*i*. *A*)

Further rules.

lemma *vdunion-def*: (∐∘*i*∈∘*I*. *A i*) = (⋃∘*i*∈∘*I*. ⋃∘*x*∈∘*A i*. *set* {⟨*i*, *x*⟩})
by (*auto simp*: *vrange-VLambda vimage-VLambda-vrange-rep*)

lemma *vdunionI*:
assumes *ix* = ⟨*i*, *x*⟩ **and** *i* ∈∘ *I* **and** *x* ∈∘ *A i*
shows *ix* ∈∘ (∐∘*i*∈∘*I*. *A i*)
using *assms*(2,3) **unfolding** *assms*(1) *vdunion-def* **by** (*intro vifunionI*) *auto*

lemmas *vdunionD* = *VSigmaD*1 *VSigmaD*2
and *vdunionE* = *VSigmaE*

Set operations.

lemma *vdunion-vsingleton*: (∐∘*i*∈∘*set*{*c*}. *A i*) = *set* {*c*} ×∘ *A c* **by** *auto*

lemma *vdunion-vdoubleton*:
(∐∘*i*∈∘*set*{*a*, *b*}. *A i*) = *set* {*a*} ×∘ *A a* ∪∘ *set* {*b*} ×∘ *A b*
by *auto*

Connections.

lemma *vdunion-vsum*: (∐∘*i*∈∘*set*{0, 1}. *if i*=0 *then A else B*) = *A* ⊎ *B*
unfolding *vdunion-vdoubleton vsum-def* **by** *simp*

2.5.7 Canonical injection

definition *vcinjection* :: (*V* ⇒ *V*) ⇒ *V* ⇒ *V*
where *vcinjection A i* = (λ*x*∈∘*A i*. ⟨*i*, *x*⟩)

Rules.

mk-VLambda *vcinjection-def*
|*vsv vcinjection-vsv*[*intro*]|
|*vdomain vcinjection-vdomain*[*simp*]|

|*app vcinjection-app*[*simp*, *intro*]|

Elementary results.

lemma *vcinjection-vrange-vsubset*:
assumes $i \in_\circ I$
shows $\mathcal{R}_\circ\ (\textit{vcinjection}\ A\ i) \subseteq_\circ (\coprod_\circ i \in_\circ I.\ A\ i)$
unfolding *vcinjection-def*
proof(*intro vrange-VLambda-vsubset*)
fix x **assume** *prems*: $x \in_\circ A\ i$
show $\langle i, x\rangle \in_\circ (\coprod_\circ i \in_\circ I.\ A\ i)$
by (*intro vdunionI*[**where** $A{=}A$, *OF - assms prems*]) *simp*
qed

lemma *vcinjection-vrange*:
assumes $i \in_\circ I$ **and** $\bigwedge j.\ j \in_\circ I \Longrightarrow A\ j \neq 0$
shows $\mathcal{R}_\circ\ (\textit{vcinjection}\ A\ i) = (\bigcup_\circ x \in_\circ A\ i.\ \textit{set}\ \{\langle i, x\rangle\})$
proof(*intro vsubset-antisym*)
interpret *vsv* ‹*vcinjection A i*› **by** (*rule vcinjection-vsv*)
show $\mathcal{R}_\circ\ (\textit{vcinjection}\ A\ i) \subseteq_\circ (\bigcup_\circ x \in_\circ A\ i.\ \textit{set}\ \{\langle i, x\rangle\})$
unfolding *vcinjection-def*
proof(*intro vrange-VLambda-vsubset*)
fix x **assume** *prems*: $x \in_\circ A\ i$
show $\langle i, x\rangle \in_\circ (\bigcup_\circ x \in_\circ A\ i.\ \textit{set}\ \{\langle i, x\rangle\})$
by (*intro vifunionI*, *rule prems*) *simp*
qed
show $(\bigcup_\circ x \in_\circ A\ i.\ \textit{set}\ \{\langle i, x\rangle\}) \subseteq_\circ \mathcal{R}_\circ\ (\textit{vcinjection}\ A\ i)$
proof(*rule vsubsetI*)
fix *ix* **assume** $ix \in_\circ (\bigcup_\circ x \in_\circ A\ i.\ \textit{set}\ \{\langle i, x\rangle\})$
then obtain x **where** x: $x \in_\circ A\ i$ **and** *ix-def*: $ix = \langle i, x\rangle$
by (*elim vifunionE*) *auto*
with x **show** $ix \in_\circ \mathcal{R}_\circ\ (\textit{vcinjection}\ A\ i)$
unfolding *ix-def* **by** (*intro vsv-vimageI2′*) *auto*
qed
qed

2.5.8 Infinite Cartesian product

definition *vproduct* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where $\textit{vproduct}\ I\ A = \textit{set}\ \{f.\ \textit{vsv}\ f \wedge \mathcal{D}_\circ\ f = I \wedge (\forall\, i \in_\circ I.\ f(\!|i|\!) \in_\circ A\ i)\}$

syntax -*VPRODUCT* :: *pttrn* ⇒ $V \Rightarrow V \Rightarrow V$ (‹(3$\prod_\circ$-$\in_\circ$-./ -)› [0, 0, 10] 10)

syntax-consts -*VPRODUCT* ⇌ *vproduct*

translations $\prod_\circ i \in_\circ I.\ A \rightleftharpoons \textit{CONST vproduct}\ I\ (\lambda i.\ A)$

lemma *small-vproduct*[*simp*]:
small $\{f.\ \textit{vsv}\ f \wedge \mathcal{D}_\circ\ f = I \wedge (\forall\, i \in_\circ I.\ f(\!|i|\!) \in_\circ A\ i)\}$
(**is** ‹*small ?A*›)
proof–
from *small-vsv*[*of I* ‹$(\bigcup_\circ i \in_\circ I.\ A\ i)$›] **have**
small $\{f.\ \textit{vsv}\ f \wedge \mathcal{D}_\circ\ f = I \wedge \mathcal{R}_\circ\ f \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)\}$
by *simp*
moreover have $?A \subseteq \{f.\ \textit{vsv}\ f \wedge \mathcal{D}_\circ\ f = I \wedge \mathcal{R}_\circ\ f \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)\}$
proof(*intro subsetI*, *unfold mem-Collect-eq*, *elim conjE*, *intro conjI*)
fix f **assume** *prems*: *vsv f* $\mathcal{D}_\circ\ f = I$ $\forall\, i \in_\circ \textit{elts}\ I.\ f(\!|i|\!) \in_\circ A\ i$
interpret *vsv f* **by** (*rule prems*(1))
show $\mathcal{R}_\circ\ f \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$

```
    proof(intro vsubsetI)
      fix y assume y ∈ₒ ℛₒ f
      with prems(2) obtain i where y-def: y = f(|i|) and i: i ∈ₒ I
        by (blast dest: vrange-atD)
      from i prems(3) vifunionI show y ∈ₒ (⋃ₒ i∈ₒI. A i)
        unfolding y-def by auto
    qed
  qed
  ultimately show ?thesis by (metis (lifting) smaller-than-small)
qed
```

Rules.

```
lemma vproductI[intro!]:
  assumes vsv f and 𝒟ₒ f = I and ∀ i∈ₒI. f(|i|) ∈ₒ A i
  shows f ∈ₒ (∏ₒ i∈ₒI. A i)
  using assms small-vproduct unfolding vproduct-def by auto

lemma vproductD[dest]:
  assumes f ∈ₒ (∏ₒ i∈ₒI. A i)
  shows vsv f
    and 𝒟ₒ f = I
    and ∀ i∈ₒI. f(|i|) ∈ₒ A i
  using assms unfolding vproduct-def by auto

lemma vproductE[elim!]:
  assumes f ∈ₒ (∏ₒ i∈ₒI. A i)
  obtains vsv f and 𝒟ₒ f = I and ∀ i∈ₒI. f(|i|) ∈ₒ A i
  using assms unfolding vproduct-def by auto
```

Set operations.

```
lemma vproduct-index-vempty[simp]: (∏ₒ i∈ₒ0. A i) = set {0}
proof–
  have {f. vsv f ∧ 𝒟ₒ f = 0 ∧ (∀ i∈ₒ0. f(|i|) ∈ₒ A i)} = {0}
    using vbrelation.vlrestriction-vdomain vsv-eqI by fastforce
  then show ?thesis unfolding vproduct-def by simp
qed

lemma vproduct-vsingletonI:
  assumes f(|c|) ∈ₒ A c and f = set {⟨c, f(|c|)⟩}
  shows f ∈ₒ (∏ₒ i∈ₒset{c}. A i)
  using assms
  apply(intro vproductI)
  subgoal by (metis rel-vsingleton.vsv-axioms)
  subgoal by (force intro!: vsubset-antisym)
  subgoal by auto
  done

lemma vproduct-vsingletonD:
  assumes f ∈ₒ (∏ₒ i∈ₒset{c}. A i)
  shows vsv f and f(|c|) ∈ₒ A c and f = set {⟨c, f(|c|)⟩}
proof–
  from assms show f = set {⟨c, f(|c|)⟩}
    by (elim vproductE) (metis VLambda-vsingleton vsv.vsv-is-VLambda)
qed (use assms in auto)

lemma vproduct-vsingletonE:
  assumes f ∈ₒ (∏ₒ i∈ₒset{c}. A i)
  obtains vsv f and f(|c|) ∈ₒ A c and f = set {⟨c, f(|c|)⟩}
```

using *assms vproduct-vsingletonD that* **by** *auto*

lemma *vproduct-vsingleton-iff*:
$f \in_\circ (\prod_\circ i \in_\circ set\{c\}.\ A\ i) \longleftrightarrow f(\!|c|\!) \in_\circ A\ c \wedge f = set\ \{\langle c, f(\!|c|\!)\rangle\}$
by (*rule iffI*) (*auto simp*: *vproduct-vsingletonD intro*!: *vproduct-vsingletonI*)

lemma *vproduct-vdoubletonI*[*intro*]:
assumes *vsv f*
and $f(\!|a|\!) \in_\circ A\ a$
and $f(\!|b|\!) \in_\circ A\ b$
and $\mathcal{D}_\circ\ f = set\ \{a, b\}$
and $\mathcal{R}_\circ\ f \subseteq_\circ A\ a \cup_\circ A\ b$
shows $f \in_\circ (\prod_\circ i \in_\circ set\ \{a, b\}.\ A\ i)$
using *assms vifunion-vdoubleton* **by** (*intro vproductI*) *auto*

lemma *vproduct-vdoubletonD*[*dest*]:
assumes $f \in_\circ (\prod_\circ i \in_\circ set\{a, b\}.\ A\ i)$
shows *vsv f*
and $f(\!|a|\!) \in_\circ A\ a$
and $f(\!|b|\!) \in_\circ A\ b$
and $\mathcal{D}_\circ\ f = set\ \{a, b\}$
and $f = set\ \{\langle a, f(\!|a|\!)\rangle, \langle b, f(\!|b|\!)\rangle\}$
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms vifunion-vdoubleton* **by** *fastforce*
subgoal by (*metis assms VLambda-vdoubleton vproductE vsv.vsv-is-VLambda*)
done

lemma *vproduct-vdoubletonE*:
assumes $f \in_\circ (\prod_\circ i \in_\circ set\{a, b\}.\ A\ i)$
obtains *vsv f*
and $f(\!|a|\!) \in_\circ A\ a$
and $f(\!|b|\!) \in_\circ A\ b$
and $\mathcal{D}_\circ\ f = set\ \{a, b\}$
and $f = set\ \{\langle a, f(\!|a|\!)\rangle, \langle b, f(\!|b|\!)\rangle\}$
using *assms vproduct-vdoubletonD that* **by** *simp*

lemma *vproduct-vdoubleton-iff*:
$f \in_\circ (\prod_\circ i \in_\circ set\ \{a, b\}.\ A\ i) \longleftrightarrow$
vsv f $\wedge$
$f(\!|a|\!) \in_\circ A\ a \wedge$
$f(\!|b|\!) \in_\circ A\ b \wedge$
$\mathcal{D}_\circ\ f = set\ \{a, b\} \wedge$
$f = set\ \{\langle a, f(\!|a|\!)\rangle, \langle b, f(\!|b|\!)\rangle\}$
by (*force elim*!: *vproduct-vdoubletonE*)+

Elementary properties.

lemma *vproduct-eq-vemptyI*:
assumes $i \in_\circ I$ **and** $A\ i = 0$
shows $(\prod_\circ i \in_\circ I.\ A\ i) = 0$
proof(*intro vsubset-antisym vsubsetI*)
fix *x* **assume** *prems*: $x \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$
from *assms vproductD*(3)[*OF prems*] **show** $x \in_\circ 0$ **by** *auto*
qed *auto*

lemma *vproduct-eq-vemptyD*:
assumes $(\prod_\circ i \in_\circ I.\ A\ i) \neq 0$

shows $\bigwedge i.\ i \in_\circ I \Longrightarrow A\ i \neq 0$
proof(*rule ccontr*, *unfold not-not*)
fix i **assume** *prems*: $i \in_\circ I\ A\ i = 0$
with *vproduct-eq-vemptyI*[**where** *A=A*, *OF prems*] *assms* **show** *False* **by** *simp*
qed

lemma *vproduct-vrange*:
assumes $f \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$
shows $\mathcal{R}_\circ\ f \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$
proof(*intro vsubsetI*)
fix x **assume** *prems*: $x \in_\circ \mathcal{R}_\circ\ f$
have *vsv-f*: *vsv f*
and *dom-f*: $\mathcal{D}_\circ\ f = I$
and *fi*: $\bigwedge i.\ i \in_\circ I \Longrightarrow f(\!|i|\!) \in_\circ A\ i$
by (*simp-all add*: *vproductD*[*OF assms*, *rule-format*])
interpret *f*: *vsv f* **by** (*rule vsv-f*)
from *prems dom-f* **obtain** i **where** *x-def*: $x = f(\!|i|\!)$ **and** *i*: $i \in_\circ I$
by (*auto elim*: *f.vrange-atE*)
from *i fi* **show** $x \in_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$ **unfolding** *x-def* **by** (*intro vifunionI*) *auto*
qed

lemma *vproduct-vsubset-VPow*: $(\prod_\circ i \in_\circ I.\ A\ i) \subseteq_\circ VPow\ (I \times_\circ (\bigcup_\circ i \in_\circ I.\ A\ i))$
proof(*intro vsubsetI*)
fix f **assume** $f \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$
then have *vsv*: *vsv f*
and *domain*: $\mathcal{D}_\circ\ f = I$
and *range*: $\forall\ i \in elts\ I.\ f(\!|i|\!) \in_\circ A\ i$
by *auto*
interpret *f*: *vsv f* **by** (*rule vsv*)
have $f \subseteq_\circ I \times_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$
proof(*intro vsubsetI*)
fix x **assume** *prems*: $x \in_\circ f$
then obtain $a\ b$ **where** *x-def*: $x = \langle a, b \rangle$ **by** (*elim f.vbrelation-vinE*)
with *prems* **have** $a \in_\circ \mathcal{D}_\circ\ f$ **and** $b \in_\circ \mathcal{R}_\circ\ f$ **by** *auto*
with *range domain prems* **show** $x \in_\circ I \times_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)$
by (*fastforce simp*: *x-def*)
qed
then show $f \in_\circ VPow\ (I \times_\circ (\bigcup_\circ i \in_\circ I.\ A\ i))$ **by** *simp*
qed

lemma *VLambda-in-vproduct*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow f\ i \in_\circ A\ i$
shows $(\lambda i \in_\circ I.\ f\ i) \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$
using *assms* **by** (*simp add*: *vproductI vsv.vsv-vrange-vsubset-vifunion-app*)

lemma *vproduct-cong*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow f\ i = g\ i$
shows $(\prod_\circ i \in_\circ I.\ f\ i) = (\prod_\circ i \in_\circ I.\ g\ i)$
proof–
have $(\prod_\circ i \in_\circ I.\ f\ i) \subseteq_\circ (\prod_\circ i \in_\circ I.\ g\ i)$ **if** $\bigwedge i.\ i \in_\circ I \Longrightarrow f\ i = g\ i$ **for** $f\ g$
proof(*intro vsubsetI*)
fix x **assume** $x \in_\circ (\prod_\circ i \in_\circ I.\ f\ i)$
note *xD* = *vproductD*[*OF this*]
interpret *vsv x* **by** (*rule xD*(1))
show $x \in_\circ (\prod_\circ i \in_\circ I.\ g\ i)$
by (*metis xD*(2,3) *that VLambda-in-vproduct vsv-is-VLambda*)
qed
with *assms* **show** *?thesis* **by** (*intro vsubset-antisym*) *auto*

qed

lemma *vproduct-ex-in-vproduct*:
assumes $x \in_\circ (\prod_\circ i \in_\circ J.\ A\ i)$ **and** $J \subseteq_\circ I$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow A\ i \neq 0$
obtains X **where** $X \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$ **and** $x = X \restriction^l_\circ J$
proof–
define X **where** $X = (\lambda i \in_\circ I.\ \textit{if } i \in_\circ J \textit{ then } x(\!|i|\!) \textit{ else } (\textit{SOME } x.\ x \in_\circ A\ i))$
have X: $X \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$
by (*intro vproductI*) (*use assms* **in** ‹*auto simp*: *X-def*›)
moreover have $x = X \restriction^l_\circ J$
proof(*rule vsv-eqI*)
from *assms*(1) **have** [*simp*]: $\mathcal{D}_\circ\ x = J$ **by** *clarsimp*
moreover from *assms*(2) **have** $\mathcal{D}_\circ\ (X \restriction^l_\circ J) = J$ **unfolding** *X-def* **by** *fastforce*
ultimately show $\mathcal{D}_\circ\ x = \mathcal{D}_\circ\ (X \restriction^l_\circ J)$ **by** *simp*
show $x(\!|i|\!) = (X \restriction^l_\circ J)(\!|i|\!)$ **if** $i \in_\circ \mathcal{D}_\circ\ x$ **for** i
using *that assms*(2) **unfolding** *X-def* **by** *auto*
qed (*use assms X* **in** *auto*)
ultimately show *?thesis* **using** *that* **by** *simp*
qed

lemma *vproduct-vsingleton-def*: $(\prod_\circ i \in_\circ set\ \{j\}.\ A\ i) = (\prod_\circ i \in_\circ set\ \{j\}.\ A\ j)$
by *auto*

lemma *vprojection-in-VUnionI*:
assumes $A \subseteq_\circ (\prod_\circ i \in_\circ I.\ F\ i)$ **and** $f \in_\circ A$ **and** $i \in_\circ I$
shows $f(\!|i|\!) \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ A))$
proof(*intro VUnionI*)
show $f \in_\circ A$ **by** (*rule assms*(2))
from *assms*(1,2) **have** $f \in_\circ (\prod_\circ i \in_\circ I.\ F\ i)$ **by** *auto*
note f = *vproductD*[*OF this*, *rule-format*]
interpret *vsv f* **rewrites** $\mathcal{D}_\circ\ f = I$ **by** (*auto intro*: $f(1)$ *simp*: $f(2)$)
show $\langle i, f(\!|i|\!)\rangle \in_\circ f$ **by** (*meson assms*(3) *vsv-appE*)
show *set* $\{i, f(\!|i|\!)\} \in_\circ \langle i, f(\!|i|\!)\rangle$ **unfolding** *vpair-def* **by** *simp*
qed *simp*

2.5.9 Projection

definition *vprojection* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow V$
where *vprojection I A i* = $(\lambda f \in_\circ (\prod_\circ i \in_\circ I.\ A\ i).\ f(\!|i|\!))$

Rules.

mk-VLambda *vprojection-def*
|*vsv vprojection-vsv*[*intro*]|
|*vdomain vprojection-vdomain*[*simp*]|
|*app vprojection-app*[*simp*, *intro*]|

Elementary results.

lemma *vprojection-vrange-vsubset*:
assumes $i \in_\circ I$
shows $\mathcal{R}_\circ\ (\textit{vprojection } I\ A\ i) \subseteq_\circ A\ i$
unfolding *vprojection-def*
proof(*intro vrange-VLambda-vsubset*)
fix f **assume** *prems*: $f \in_\circ (\prod_\circ i \in_\circ I.\ A\ i)$
show $f(\!|i|\!) \in_\circ A\ i$ **by** (*intro vproductD*(3)[*OF prems*, *rule-format*] *assms*)
qed

lemma *vprojection-vrange*:
assumes $i \in_\circ I$ **and** $\bigwedge j.\ j \in_\circ I \Longrightarrow A\ j \neq 0$

shows $\mathcal{R}_\circ\ (\mathit{vprojection}\ I\ A\ i) = A\ i$
proof
(
intro
vsubset-antisym vprojection-vrange-vsubset vrange-VLambda-vsubset assms(1)
)
show $A\ i \subseteq_\circ \mathcal{R}_\circ\ (\mathit{vprojection}\ I\ A\ i)$
proof(*intro vsubsetI*)
fix x **assume** *prems*: $x \in_\circ A\ i$
obtain f
where f: $\bigwedge x.\ x \in_\circ \mathit{set}\ \{A\ i \mid i.\ i \in_\circ I\} \Longrightarrow x \neq 0 \Longrightarrow f(\!|x|\!) \in_\circ x$
and *vsv f*
using *that* **by** (*rule Axiom-of-Choice*)
define f' **where** $f' = (\lambda j \in_\circ I.\ \mathit{if}\ j = i\ \mathit{then}\ x\ \mathit{else}\ f(\!|A\ j|\!))$
show $x \in_\circ \mathcal{R}_\circ\ (\mathit{vprojection}\ I\ A\ i)$
unfolding *vprojection-def*
proof(*rule rel-VLambda.vsv-vimageI2'*)
show $f' \in_\circ \mathcal{D}_\circ\ (\lambda f \in_\circ \mathit{vproduct}\ I\ A.\ f(\!|i|\!))$
unfolding *vdomain-VLambda*
proof(*intro vproductI*, *unfold Ball-def*; (*intro allI conjI impI*)*?*)
fix j **assume** $j \in_\circ I$
with *prems assms*(2) **show** $f'(\!|j|\!) \in_\circ A\ j$
unfolding *f'-def* **by** (*cases* ‹$j = i$›) (*auto intro*!: *f*)
qed (*simp-all add*: *f'-def*)
with *assms*(1) **show** $x = (\lambda f \in_\circ \mathit{vproduct}\ I\ A.\ f(\!|i|\!))(\!|f'|\!)$
unfolding *f'-def* **by** *simp*
qed
qed
qed

2.5.10 Cartesian power of a set

definition $\mathit{vcpower} :: V \Rightarrow V \Rightarrow V$ (**infixr** ‹$\widehat{}_\times$› 80)
where $A\ \widehat{}_\times\ n = (\prod_\circ i \in_\circ n.\ A)$

Rules.

lemma *vcpowerI*[*intro*]:
assumes $f \in_\circ (\prod_\circ i \in_\circ n.\ A)$
shows $f \in_\circ (A\ \widehat{}_\times\ n)$
using *assms* **unfolding** *vcpower-def* **by** *auto*

lemma *vcpowerD*[*dest*]:
assumes $f \in_\circ (A\ \widehat{}_\times\ n)$
shows $f \in_\circ (\prod_\circ i \in_\circ n.\ A)$
using *assms* **unfolding** *vcpower-def* **by** *auto*

lemma *vcpowerE*[*elim*]:
assumes $f \in_\circ (A\ \widehat{}_\times\ n)$ **and** $f \in_\circ (\prod_\circ i \in_\circ n.\ A) \Longrightarrow P$
shows P
using *assms* **unfolding** *vcpower-def* **by** *auto*

Set operations.

lemma *vcpower-index-vempty*[*simp*]: $A\ \widehat{}_\times\ 0 = \mathit{set}\ \{0\}$
unfolding *vcpower-def* **by** (*rule vproduct-index-vempty*)

lemma *vcpower-of-vempty*:
assumes $n \neq 0$
shows $0\ \widehat{}_\times\ n = 0$

using *assms* **unfolding** *vcpower-def vproduct-def* **by** *simp*

lemma *vcpower-vsubset-mono*:
assumes $A \subseteq_\circ B$
shows $A \mathbin{\widehat{}}_\times n \subseteq_\circ B \mathbin{\widehat{}}_\times n$
using *assms*
by (*intro vsubsetI vcpowerI vproductI*)
(*auto intro*: *vproductD*[*OF vcpowerD*, *rule-format*])

Connections.

lemma *vcpower-vdomain*:
assumes $f \in_\circ (A \mathbin{\widehat{}}_\times n)$
shows $\mathcal{D}_\circ f = n$
using *assms* **by** *auto*

lemma *vcpower-vrange*:
assumes $f \in_\circ (A \mathbin{\widehat{}}_\times n)$
shows $\mathcal{R}_\circ f \subseteq_\circ A$
using *assms* **by** (*intro vsubsetI*; *elim vcpowerE vproductE*) *auto*

2.6 Equipollence

2.6.1 Background

The section presents an adaption of the existing framework *Equipollence* in the main library of Isabelle/HOL to the type *V*.

Some of content of this theory was ported directly (with amendments) from the theory *HOL–Library.Equipollence* in the main library of Isabelle/HOL.

2.6.2 *veqpoll*

abbreviation *veqpoll* :: $V \Rightarrow V \Rightarrow$ *bool* (**infixl** ‹$\approx_\circ$› 50)
  where $A \approx_\circ B \equiv$ *elts* $A \approx$ *elts* B

Rules

lemma (**in** *v*11) *v*11-*veqpollI*[*intro*]:
  assumes $\mathcal{D}_\circ\ r = A$ **and** $\mathcal{R}_\circ\ r = B$
  shows $A \approx_\circ B$
  unfolding *eqpoll-def*
proof(*intro exI*[*of* - ‹$\lambda x.\ r(\!|x|\!)$›] *bij-betw-imageI*)
  from *v*11.*v*11-*injective* *v*11-*axioms* **show** *inj-on* (*app r*) (*elts A*)
    unfolding *assms*[*symmetric*] **by** (*intro inj-onI*) *blast*
  show *app r* ‘ *elts A* = *elts B* **unfolding** *assms*[*symmetric*] **by** *force*+
qed

lemmas *v*11-*veqpollI*[*intro*] = *v*11.*v*11-*veqpollI*

lemma *v*11-*veqpollE*[*elim*]:
  assumes $A \approx_\circ B$
  obtains *f* **where** *v*11 *f* **and** $\mathcal{D}_\circ\ f = A$ **and** $\mathcal{R}_\circ\ f = B$
proof–
  from *assms* **obtain** *f* **where** *bij-f*: *bij-betw f* (*elts A*) (*elts B*)
    unfolding *eqpoll-def* **by** *auto*
  then have *v*11 ($\lambda a \in_\circ A.\ f\ a$)
    and $\mathcal{D}_\circ$ ($\lambda a \in_\circ A.\ f\ a$) = A
    and $\mathcal{R}_\circ$ ($\lambda a \in_\circ A.\ f\ a$) = B
    by (*auto simp add*: *in-mono vrange-VLambda*)
  then show *?thesis* **using** *that* **by** *simp*
qed

Set operations.

lemma *veqpoll-vsingleton*: *set* {*x*} $\approx_\circ$ *set* {*y*}
  by (*simp add*: *singleton-eqpoll*)

lemma *veqpoll-vinsert*:
  assumes $A \approx_\circ B$ **and** $a \notin_\circ A$ **and** $b \notin_\circ B$
  shows *vinsert a A* $\approx_\circ$ *vinsert b B*
  using *assms* **by** (*simp add*: *insert-eqpoll-insert-iff*)

lemma *veqpoll-pair*:
  assumes $a \neq b$ **and** $c \neq d$
  shows *set* {*a*, *b*} $\approx_\circ$ *set* {*c*, *d*}
  using *assms* **by** (*simp add*: *insert-eqpoll-cong*)

lemma *veqpoll-vpair*:
  assumes $a \neq b$ **and** $c \neq d$
  shows ⟨*a*, *b*⟩ $\approx_\circ$ ⟨*c*, *d*⟩

using *assms*
unfolding *vpair-def*
by (*metis doubleton-eq-iff insert-absorb2 veqpoll-pair*)

2.6.3 *vlepoll*

abbreviation *vlepoll* :: $V \Rightarrow V \Rightarrow$ *bool* (**infixl** ‹$\lesssim_\circ$› 50)
where $A \lesssim_\circ B \equiv elts\ A \lesssim elts\ B$

Set operations.

lemma *vlepoll-vsubset*:
assumes $A \subseteq_\circ B$
shows $A \lesssim_\circ B$
using *assms* **by** (*simp add*: *less-eq-V-def subset-imp-lepoll*)

Special properties.

lemma *vlepoll-singleton-vinsert*: *set* $\{x\} \lesssim_\circ$ *vinsert y A*
by (*simp add*: *singleton-lepoll*)

lemma *vlepoll-vempty-iff*[*simp*]: $A \lesssim_\circ 0 \longleftrightarrow A = 0$ **by** (*rule iffI*) *fastforce+*

2.6.4 *vlespoll*

abbreviation *vlesspoll* :: $V \Rightarrow V \Rightarrow$ *bool* (**infixl** ‹$\prec_\circ$› 50)
where $A \prec_\circ B \equiv elts\ A \prec elts\ B$

lemma *vlesspoll-def*: $A \prec_\circ B = (A \lesssim_\circ B \wedge \tilde{}(A \approx_\circ B))$ **by** (*simp add*: *lesspoll-def*)

Rules.

lemmas *vlesspollI*[*intro*] = *vlesspoll-def*[*THEN iffD2*]

lemmas *vlesspollD*[*dest*] = *vlesspoll-def*[*THEN iffD1*]

lemma *vlesspollE*[*elim*]:
assumes $A \prec_\circ B$ **and** $A \lesssim_\circ B \Longrightarrow \tilde{}(A \approx_\circ B) \Longrightarrow P$
shows *P*
using *assms* **by** (*simp add*: *vlesspoll-def*)

lemma (**in** *v*11) *v*11-*vlepollI*[*intro*]:
assumes $\mathcal{D}_\circ\ r = A$ **and** $\mathcal{R}_\circ\ r \subseteq_\circ B$
shows $A \lesssim_\circ B$
unfolding *lepoll-def*
proof(*intro exI*[*of* - ‹$\lambda x.\ r(\lfloor x \rfloor)$›] *conjI*)
show *inj-on* (*app r*) (*elts A*)
using *assms*(1) *v*11.*v*11-*injective v*11-*axioms* **by** (*intro inj-onI*) *blast*
show *app r* ‘ *elts A* ⊆ *elts B*
by (*intro subsetI*) (*metis assms*(1,2) *imageE rev-vsubsetD vdomain-atD*)
qed

lemmas *v*11-*vlepollI*[*intro*] = *v*11.*v*11-*vlepollI*

lemma *v*11-*vlepollE*[*elim*]:
assumes $A \lesssim_\circ B$
obtains *f* **where** *v*11 *f* **and** $\mathcal{D}_\circ\ f = A$ **and** $\mathcal{R}_\circ\ f \subseteq_\circ B$
proof–
from *assms* **obtain** *f* **where** *inj-on f* (*elts A*) **and** *f* ‘ *elts A* ⊆ *elts B*
unfolding *lepoll-def* **by** *auto*
then have *v*11 ($\lambda a \in_\circ A.\ f\ a$)

```
    and 𝒟∘ (λa∈∘A. f a) = A
    and ℛ∘ (λa∈∘A. f a) ⊆∘ B
    by (auto simp: in-mono vrange-VLambda)
  then show ?thesis using that by simp
qed
```

2.7 Cardinality

2.7.1 Background

The section presents further results about the cardinality of terms of the type V. The emphasis of this work, however, is on the development of a theory of finite sets internalized in the type V.

Many of the results that are presented in this section were carried over (with amendments) from the theory *Finite* in the main library of Isabelle/HOL.

declare *One-nat-def*[*simp del*]

2.7.2 Cardinality of an arbitrary set

Elementary properties.

lemma *vcard-veqpoll*: *vcard A = vcard B* $\longleftrightarrow$ $A \approx_\circ B$
by (*metis cardinal-cong cardinal-eqpoll eqpoll-sym eqpoll-trans*)

lemma *vcard-vlepoll*: *vcard A* $\leq$ *vcard B* $\longleftrightarrow$ $A \lesssim_\circ B$
proof
assume *vcard A* $\leq$ *vcard B*
moreover have *vcard A* $\approx_\circ A$ **by** (*rule cardinal-eqpoll*)
moreover have *vcard B* $\approx_\circ B$ **by** (*rule cardinal-eqpoll*)
ultimately show $A \lesssim_\circ B$
by (*meson eqpoll-sym lepoll-trans*1 *lepoll-trans*2 *vlepoll-vsubset*)
qed (*simp add*: *lepoll-imp-Card-le*)

lemma *vcard-vempty*: *vcard A* = 0 $\longleftrightarrow$ $A = 0$
proof–
have *vcard-A*: *vcard A* $\approx_\circ A$ **by** (*simp add*: *cardinal-eqpoll*)
then show *?thesis* **using** *eq*0-*iff eqpoll-iff-bijections* **by** *metis*
qed

lemmas *vcard-vemptyD* = *vcard-vempty*[*THEN iffD*1]
and *vcard-vemptyI* = *vcard-vempty*[*THEN iffD*2]

lemma *vcard-neq-vempty*: *vcard A* $\neq 0_{\mathbf{N}}$ $\longleftrightarrow$ $A \neq 0_{\mathbf{N}}$
using *vcard-vempty* **by** *auto*

lemmas *vcard-neq-vemptyD* = *vcard-neq-vempty*[*THEN iffD*1]
and *vcard-neq-vemptyI* = *vcard-neq-vempty*[*THEN iffD*2]

Set operations.

lemma *vcard-mono*:
assumes $A \subseteq_\circ B$
shows *vcard A* $\leq$ *vcard B*
using *assms* **by** (*simp add*: *lepoll-imp-Card-le vlepoll-vsubset*)

lemma *vcard-vinsert-in*[*simp*]:
assumes $a \in_\circ A$
shows *vcard* (*vinsert a A*) = *vcard A*
using *assms* **by** (*simp add*: *cardinal-cong insert-absorb*)

lemma *vcard-vintersection-left*: *vcard* $(A \cap_\circ B) \leq$ *vcard A*
by (*simp add*: *vcard-mono*)

lemma *vcard-vintersection-right*: *vcard* $(A \cap_\circ B) \leq$ *vcard B*

by (*simp add*: *vcard-mono*)

lemma *vcard-vunion*:
assumes *vdisjnt A B*
shows *vcard* $(A \cup_\circ B)$ = *vcard A* $\oplus$ *vcard B*
using *assms* **by** (*rule vcard-disjoint-sup*)

lemma *vcard-vdiff*[*simp*]: *vcard* $(A -_\circ B) \oplus$ *vcard* $(A \cap_\circ B)$ = *vcard A*
proof–
have *ABB*: *vdisjnt* $(A -_\circ B)$ $(A \cap_\circ B)$ **by** *auto*
have $A -_\circ B \cup_\circ A \cap_\circ B = A$ **by** *auto*
from *vcard-vunion*[*OF ABB*, *unfolded this*] **show** *?thesis* **..**
qed

lemma *vcard-vdiff-vsubset*:
assumes $B \subseteq_\circ A$
shows *vcard* $(A -_\circ B) \oplus$ *vcard B* = *vcard A*
by (*metis assms inf.absorb-iff2 vcard-vdiff*)

Connections.

lemma (in *vsv*) *vsv-vcard-vdomain*: *vcard* $(\mathcal{D}_\circ\ r)$ = *vcard r*
unfolding *vcard-veqpoll*
proof–
define *f* **where** $f\ x = \langle x, r(\!|x|\!)\rangle$ **for** *x*
have *bij-betw f* (*elts* $(\mathcal{D}_\circ\ r)$) (*elts r*)
unfolding *f-def bij-betw-def*
proof(*intro conjI inj-onI subset-antisym subsetI*)
from *vlrestriction-vdomain* **show**
$x \in_\circ r \Longrightarrow x \in (\lambda x.\ \langle x, r(\!|x|\!)\rangle)$ ‘ *elts* $(\mathcal{D}_\circ\ r)$
for *x*
unfolding *mem-Collect-eq* **by** *blast*
qed (*auto simp*: *image-def*)
then show $\mathcal{D}_\circ\ r \approx_\circ r$ **unfolding** *eqpoll-def* **by** *auto*
qed

Special properties.

lemma *vcard-vunion-vintersection*:
vcard $(A \cup_\circ B) \oplus$ *vcard* $(A \cap_\circ B)$ = *vcard A* $\oplus$ *vcard B*
proof–
have *AB-ABB*: $A \cup_\circ B = B \cup_\circ (A -_\circ B)$ **by** *auto*
have *ABB*: *vdisjnt B* $(A -_\circ B)$ **by** *auto*
show *?thesis*
unfolding *vcard-vunion*[*OF ABB*, *folded AB-ABB*] *cadd-assoc vcard-vdiff*
by (*simp add*: *cadd-commute*)
qed

2.7.3 Finite sets

abbreviation *vfinite* :: $V \Rightarrow$ *bool*
where *vfinite A* ≡ *finite* (*elts A*)

lemma *vfinite-def*: *vfinite A* $\longleftrightarrow$ $(\exists\, n{\in_\circ}\omega.\ n \approx_\circ A)$
proof
assume *finite* (*elts A*)
then obtain *n* :: *nat* **where** *eltsA*: *elts A* ≈ {..<*n*}
by (*simp add*: *eqpoll-iff-card*)
have *on*: *ord-of-nat n* = *set* (*ord-of-nat* ‘ {..<*n*})
by (*simp add*: *ord-of-nat-eq-initial*[*symmetric*])

from *eltsA* **have** *elts A* ≈ *elts* (*ord-of-nat n*)
unfolding *on* **by** (*simp add*: *inj-on-def*)
moreover have *ord-of-nat n* ∈$_\circ$ *ω* **by** (*simp add*: *ω-def*)
ultimately show ∃ *n*∈$_\circ$*ω*. *n* ≈$_\circ$ *A* **by** (*auto intro*: *eqpoll-sym*)
next
assume ∃ *n*∈$_\circ$*ω*. *n* ≈$_\circ$ *A*
then obtain *n* **where** *n* ∈$_\circ$ *ω* **and** *n* ≈$_\circ$ *A* **by** *auto*
with *eqpoll-finite-iff* **show** *finite* (*elts A*)
by (*auto intro*: *finite-Ord-omega*)
qed

Rules.

lemmas *vfiniteI*[*intro*!] = *vfinite-def*[*THEN iffD*2]

lemmas *vfiniteD*[*dest*!] = *vfinite-def*[*THEN iffD*1]

lemma *vfiniteE*1[*elim*!]:
assumes *vfinite A* **and** ∃ *n*∈$_\circ$*ω*. *n* ≈$_\circ$ *A* ⟹ *P*
shows *P*
using *assms* **by** *auto*

lemma *vfiniteE*2[*elim*]:
assumes *vfinite A*
obtains *n* **where** *n* ∈$_\circ$ *ω* **and** *n* ≈$_\circ$ *A*
using *assms* **by** *auto*

Elementary properties.

lemma *veqpoll-omega-vcard*[*intro*, *simp*]:
assumes *n* ∈$_\circ$ *ω* **and** *n* ≈$_\circ$ *A*
shows *vcard A* = *n*
using
nat-into-Card[*OF assms*(1), *unfolded Card-def*]
cardinal-cong[*OF assms*(2)]
by *simp*

lemma (**in** *vsv*) *vfinite-vimage*[*intro*]:
assumes *vfinite A*
shows *vfinite* (*r* ‘$_\circ$ *A*)
proof–
have *rA*: *r* ‘$_\circ$ *A* = *r* ‘$_\circ$ (*𝒟*$_\circ$ *r* ∩$_\circ$ *A*) **by** *fast*
have *DrA*: *𝒟*$_\circ$ *r* ∩$_\circ$ *A* ⊆$_\circ$ *𝒟*$_\circ$ *r* **by** *simp*
show *?thesis* **by** (*simp add*: *inf-V-def assms vimage-image*[*OF DrA*, *folded rA*])
qed

lemmas [*intro*] = *vsv.vfinite-vimage*

lemma *vfinite-veqpoll-trans*:
assumes *vfinite A* **and** *A* ≈$_\circ$ *B*
shows *vfinite B*
using *assms* **by** (*simp add*: *eqpoll-finite-iff*)

lemma *vfinite-vlepoll-trans*:
assumes *vfinite A* **and** *B* ≲$_\circ$ *A*
shows *vfinite B*
by (*meson assms eqpoll-finite-iff finite-lepoll-infinite lepoll-antisym*)

lemma *vfinite-vlesspoll-trans*:
assumes *vfinite A* **and** *B* ≺$_\circ$ *A*

shows *vfinite B*
using *assms* **by** (*auto simp*: *vlesspoll-def vfinite-vlepoll-trans*)

Induction.

lemma *vfinite-induct*[*consumes* 1, *case-names vempty vinsert*]:
assumes *vfinite F*
and *P* 0
and $\bigwedge x\ F.\ [\![\mathit{vfinite}\ F;\ x \notin_\circ F;\ P\ F]\!] \Longrightarrow P\ (\mathit{vinsert}\ x\ F)$
shows *P F*
proof–

from *assms*(1) **obtain** *n* **where** *n*: $n \in_\circ \omega$ **and** $n \approx_\circ F$ **by** *clarsimp*
then obtain *f''* **where** *bij*: *bij-betw f''* (*elts n*) (*elts F*)
unfolding *eqpoll-def* **by** *clarsimp*
define *f* **where** $f = (\lambda a{\in_\circ} n.\ f''\ a)$
interpret *v11 f*
unfolding *f-def*
proof(*intro v11I*)
show *vsv* $((\lambda a{\in_\circ} n.\ f''\ a)^{-1}{}_\circ)$
proof(*intro vsvI*)
fix *a b c*
assume $\langle a,\ b\rangle \in_\circ (\lambda a{\in_\circ} n.\ f''\ a)^{-1}{}_\circ$ **and** $\langle a,\ c\rangle \in_\circ (\lambda a{\in_\circ} n.\ f''\ a)^{-1}{}_\circ$
then have $\langle b,\ a\rangle \in_\circ (\lambda a{\in_\circ} n.\ f''\ a)$
and $\langle c,\ a\rangle \in_\circ (\lambda a{\in_\circ} n.\ f''\ a)$
and $b \in_\circ n$
and $c \in_\circ n$
by *auto*
moreover then have $f''\ b = f''\ c$ **by** *auto*
ultimately show $b = c$ **using** *bij* **by** (*metis bij-betw-iff-bijections*)
qed *auto*
qed *auto*
have *dom-f*: $\mathcal{D}_\circ\ f = n$ **unfolding** *f-def* **by** *clarsimp*
have *ran-f*: $\mathcal{R}_\circ\ f = F$
proof(*intro vsubset-antisym vsubsetI*)
fix *b* **assume** $b \in_\circ \mathcal{R}_\circ\ f$
then obtain *a* **where** $a \in_\circ n$ **and** $b = f''\ a$ **unfolding** *f-def* **by** *auto*
then show $b \in_\circ F$ **by** (*meson bij bij-betw-iff-bijections*)
next
fix *b* **assume** $b \in_\circ F$
then obtain *a* **where** $a \in_\circ n$ **and** $b = f''\ a$
by (*metis bij bij-betw-iff-bijections*)
then show $b \in_\circ \mathcal{R}_\circ\ f$ **unfolding** *f-def* **by** *auto*
qed

define *f'* **where** $f'\ n = f\ `_\circ\ n$ **for** *n*
have *F-def*: $F = f'\ n$
unfolding *f'-def* **using** *dom-f ran-f vimage-vdomain* **by** *clarsimp*
have *v11* $(\lambda a{\in_\circ} n.\ f'\ a)$
proof(*intro vsv.vsv-valneq-v11I*, *unfold vdomain-VLambda*)
show *vsv* $(\lambda a{\in_\circ} n.\ f'\ a)$ **by** *simp*
fix *x y* **assume** *xD*: $x \in_\circ n$ **and** *yD*: $y \in_\circ n$ **and** *xy*: $x \neq y$
from $\langle x \in_\circ n\rangle$ $\langle y \in_\circ n\rangle$ $\langle n \in_\circ \omega\rangle$ **have** *xn*: $x \subseteq_\circ n$ **and** *yn*: $y \subseteq_\circ n$
by (*simp-all add*: *OrdmemD order.strict-implies-order*)
show $(\lambda a{\in_\circ} n.\ f'\ a)(\lfloor x\rfloor) \neq (\lambda a{\in_\circ} n.\ f'\ a)(\lfloor y\rfloor)$
unfolding *beta*[*OF xD*] *beta*[*OF yD*] *f'-def*
using *xn yn xy*
by (*simp add*: *dom-f v11-vimage-vpsubset-neq*)
qed

```
  define P' where P' n' = (if n' ≤ n then P (f' n') else True) for n'
  from n have P' n
  proof(induct rule: omega-induct)
    case 0 then show ?case
      unfolding P'-def f'-def using assms(2) by auto
  next
    case (succ k) show ?case
    proof(cases ‹succ k ≤ n›)
      case True
      then obtain x where xF: vinsert x (f' k) = (f' (succ k))
        by (simp add: f'-def succ-def vsubsetD dom-f vsv-vimage-vinsert)
      from True have k ≤ n by auto
      with ‹P' k› have P (f' k) unfolding P'-def by simp
      then have f' k ≠ f' (succ k)
        by (simp add: True f'-def ‹k ≤ n› dom-f v11-vimage-vpsubset-neq)
      with xF have x ∉₀ f' k by auto
      have vfinite (f' k)
        by (simp add: ‹k ∈₀ ω› f'-def finite-Ord-omega vfinite-vimage)
      from assms(3)[OF ‹vfinite (f' k)› ‹x ∉₀ f' k› ‹P (f' k)›] show ?thesis
        unfolding xF P'-def by simp
    qed (unfold P'-def, auto)
  qed

  then show ?thesis unfolding P'-def F-def by simp

qed
```

Set operations.

```
lemma vfinite-vempty[simp]: vfinite (0ℕ) by simp

lemma vfinite-vsingleton[simp]: vfinite (set {x}) by simp

lemma vfinite-vdoubleton[simp]: vfinite (set {x, y}) by simp

lemma vfinite-vinsert:
  assumes vfinite F
  shows vfinite (vinsert x F)
  using assms by simp

lemma vfinite-vinsertD:
  assumes vfinite (vinsert x F)
  shows vfinite F
  using assms by simp

lemma vfinite-vsubset:
  assumes vfinite B and A ⊆₀ B
  shows vfinite A
  using assms
  by (induct arbitrary: A rule: vfinite-induct)
    (simp-all add: less-eq-V-def finite-subset)

lemma vfinite-vunion: vfinite (A ∪₀ B) ⟷ vfinite A ∧ vfinite B
  by (auto simp: elts-sup-iff)

lemma vfinite-vunionI:
  assumes vfinite A and vfinite B
  shows vfinite (A ∪₀ B)
```

using *assms* **by** (*simp add*: *elts-sup-iff*)

lemma *vfinite-vunionD*:
assumes *vfinite* ($A \cup_\circ B$)
shows *vfinite A* **and** *vfinite B*
using *assms* **by** (*auto simp*: *elts-sup-iff*)

lemma *vfinite-vintersectionI*:
assumes *vfinite A* **and** *vfinite B*
shows *vfinite* ($A \cap_\circ B$)
using *assms* **by** (*simp add*: *vfinite-vsubset*)

lemma *vfinite-VPowI*:
assumes *vfinite A*
shows *vfinite* (*VPow A*)
using *assms*
proof(*induct rule*: *vfinite-induct*)
case *vempty* **then show** *?case* **by** *simp*
next
case (*vinsert x F*)
then show *?case*
unfolding *VPow-vinsert*
using *rel-VLambda.vfinite-vimage*
by (*intro vfinite-vunionI*) *metis+*
qed

Connections.

lemma *vfinite-vcard-vfinite*: *vfinite* (*vcard A*) = *vfinite A*
by (*simp add*: *cardinal-eqpoll eqpoll-finite-iff*)

lemma *vfinite-vcard-omega-iff*: *vfinite A* $\longleftrightarrow$ *vcard A* $\in_\circ \omega$
using *vfinite-vcard-vfinite* **by** *auto*

lemmas *vcard-vfinite-omega* = *vfinite-vcard-omega-iff*[*THEN iffD*2]
and *vfinite-vcard-omega* = *vfinite-vcard-omega-iff*[*THEN iffD*1]

lemma *vfinite-csucc*[*intro*, *simp*]:
assumes *vfinite A*
shows *csucc* (*vcard A*) = *succ* (*vcard A*)
using *assms* **by** (*force simp*: *finite-csucc*)

lemmas [*intro*, *simp*] = *finite-csucc*

Previous connections.

lemma *vcard-vsingleton*[*simp*]: *vcard* (*set* $\{a\}$) = $1_{\mathbb{N}}$ **by** *auto*

lemma *vfinite-vcard-vinsert-nin*[*simp*]:
assumes *vfinite A* **and** $a \notin_\circ A$
shows *vcard* (*vinsert a A*) = *csucc* (*vcard A*)
using *assms* **by** (*simp add*: *ZFC-in-HOL.vinsert-def*)

2.8 Further results about ordinal numbers

2.8.1 Background

The subsection presents several results about ordinal numbers. The primary general reference for this section is [59].

lemmas [*intro*] = *Limit-is-Ord Ord-in-Ord*

2.8.2 Further ordinal arithmetic and inequalities

lemma *Ord-succ-mono*:
assumes *Ord* β **and** $\alpha \in_\circ \beta$
shows *succ* $\alpha \in_\circ$ *succ* β
proof–
from *assms* **have** *Ord* α **by** *blast*
from *assms* ‹*Ord* α› **have** $\alpha < \beta$ **by** (*auto dest*: *Ord-mem-iff-lt*)
from *assms*(1,2) *this* **have** *succ* $\alpha <$ *succ* β
by (*meson assms* ‹*Ord* α› *Ord-linear*2 *Ord-succ leD le-succ-iff*)
with *assms*(1) ‹*Ord* α› *Ord-mem-iff-lt* **show** *succ* $\alpha \in_\circ$ *succ* β **by** *blast*
qed

lemma *Limit-right-Limit-mult*:
— Based on Theorem 8.23 in [59].
assumes *Ord* α **and** *Limit* β **and** $0 \in_\circ \alpha$
shows *Limit* $(\alpha * \beta)$
proof–
have $\alpha\beta$: $\alpha * \beta = (\bigcup_\circ \xi \in_\circ \beta.\ \alpha * \xi)$ **by** (*rule mult-Limit*[*OF assms*(2), *of* α])
from *assms*(1,2) *Ord-mult* **have** *Ord* $(\alpha * \beta)$ **by** *blast*
then show *?thesis*
proof(*cases rule*: *Ord-cases*)
case (*succ* γ)
from *succ*(1) **have** $\gamma \in_\circ \alpha * \beta$ **unfolding** *succ*(2)[*symmetric*] **by** *simp*
then obtain ξ **where** $\xi \in_\circ \beta$ **and** $\gamma \in_\circ \alpha * \xi$ **unfolding** $\alpha\beta$ **by** *auto*
moreover with *assms*(2) **have** *Ord* ξ **by** *auto*
ultimately have $s\gamma$-$s\alpha\xi$: *succ* $\gamma \in_\circ$ *succ* $(\alpha * \xi)$
using *assms*(1) *Ord-succ-mono* **by** *simp*
from *assms*(2,3) **have** *succ* $(\alpha * \xi) \subseteq_\circ \alpha * \xi + \alpha$
unfolding *succ-eq-add*1 **by** *force*
with $s\gamma$-$s\alpha\xi$ **have** *succ* $\gamma \in_\circ \alpha *$ *succ* ξ
unfolding *mult-succ*[*symmetric*] **by** *auto*
moreover have *succ* $\xi \in_\circ \beta$
by (*simp add*: *succ-in-Limit-iff* ‹$\xi \in_\circ \beta$› *assms*(2))
ultimately have *succ* $\gamma \in_\circ \alpha * \beta$ **unfolding** $\alpha\beta$ **by** *force*
with *succ*(2) **show** *?thesis* **by** *simp*
qed (*use assms*(2,3) **in** *auto*)
qed

lemma *Limit-left-Limit-mult*:
assumes *Limit* α **and** *Ord* β **and** $0 \in_\circ \beta$
shows *Limit* $(\alpha * \beta)$
proof(*cases* ‹*Limit* β›)
case *False*
then obtain β' **where** *Ord* β' **and** β-*def*: $\beta =$ *succ* β'
by (*metis Ord-cases assms*(2,3) *eq*0-*iff*)
have α-$s\beta'$: $\alpha *$ *succ* $\beta' = \alpha * \beta' + \alpha$ **by** (*simp add*: *mult-succ*)
from *assms*(1) **have** *Limit* $(\alpha * \beta' + \alpha)$ **by** (*simp add*: *Limit-is-Ord* ‹*Ord* β'›)
then show *Limit* $(\alpha * \beta)$ **unfolding** β-*def* α-$s\beta'$ **by** *simp*
qed (*use assms* **in** ‹*auto simp*: *Limit-def dest*: *Limit-right-Limit-mult*›)

lemma *zero-if-Limit-eq-Limit-plus-vnat*:
assumes *Limit* α **and** *Limit* β **and** $\alpha = \beta + n$ **and** $n \in_\circ \omega$
shows $n = 0$
proof(*rule ccontr*)
assume *prems*: $n \neq 0$
from *assms*(1,2,4) **have** *Ord* α **and** *Ord* β **and** *Ord* 0 **and** *Ord* n **by** *auto*
have $0 \in_\circ n$ **by** (*simp add*: *mem-0-Ord prems assms*(4))
with *assms*(4) **obtain** m **where** *n-def*: $n = succ\ m$ **by** (*auto elim*: *omega-prev*)
from *assms*(1,3) **show** *False* **by** (*simp add*: *n-def plus-V-succ-right*)
qed

lemma *Ord-vsubset-closed*:
assumes *Ord* α **and** *Ord* γ **and** $\alpha \subseteq_\circ \beta$ **and** $\beta \in_\circ \gamma$
shows $\alpha \in_\circ \gamma$
proof–
from *assms* **have** *Ord* β **by** *auto*
with *assms* **show** *?thesis* **by** (*simp add*: *Ord-mem-iff-lt*)
qed

lemma
— Based on Exercise 1, page 53 in [59].
assumes *Ord* α **and** *Ord* γ **and** $\alpha + \beta \in_\circ \gamma$
shows *Ord-plus-Ord-closed-augend*: $\alpha \in_\circ \gamma$
and *Ord-plus-Ord-closed-addend*: $\beta \in_\circ \gamma$
proof–
from *assms* **have** $\alpha + \beta \in_\circ \alpha + \gamma$ **by** (*meson vsubsetD add-le-left*)
from *add-mem-right-cancel*[*THEN iffD*1, *OF this*] **show** $\beta \in_\circ \gamma$ **.**
from *assms* **have** $\alpha \subseteq_\circ \alpha + \beta$ **by** *simp*
from *Ord-vsubset-closed*[*OF assms*(1,2) *this assms*(3)] **show** $\alpha \in_\circ \gamma$ **.**
qed

lemma *Ord-ex*1-*Limit-plus-in-omega*:
— Based on Theorem 8.13 in [59].
assumes *Ord* α **and** $\omega \subseteq_\circ \alpha$
shows $\exists!\beta.\ \exists!n.\ n \in_\circ \omega \wedge Limit\ \beta \wedge \alpha = \beta + n$
proof–
let *?A* = ‹*set* $\{\gamma.\ Limit\ \gamma \wedge \gamma \subseteq_\circ \alpha\}$›
have *small*[*simp*]: *small* $\{\gamma.\ Limit\ \gamma \wedge \gamma \subseteq_\circ \alpha\}$
proof–
from *Ord-mem-iff-lt* **have** $\{\gamma.\ Limit\ \gamma \wedge \gamma \subseteq_\circ \alpha\} \subseteq elts\ (succ\ \alpha)$
by (*auto dest*: *order.not-eq-order-implies-strict intro*: *assms*(1))
then show *small* $\{\gamma.\ Limit\ \gamma \wedge \gamma \subseteq_\circ \alpha\}$ **by** (*meson down*)
qed
let *?β* = ‹$\bigcup_\circ$ *?A*›
have *?β* $\subseteq_\circ \alpha$ **by** *auto*
moreover have *L-β*: *Limit ?β*
proof(*subst Limit-def*, *intro conjI allI impI*)
show *Ord ?β* **by** (*fastforce intro*: *Ord-Sup*)
from *assms*(2) **show** $0 \in_\circ$ *?β* **by** *auto*
fix *y* **assume** *y* $\in_\circ$ *?β*
then obtain γ **where** *y* $\in_\circ \gamma$ **and** $\gamma \in_\circ$ *?A* **by** *clarsimp*
then show *succ y* $\in_\circ$ *?β* **by** (*auto simp*: *succ-in-Limit-iff*)
qed
ultimately obtain γ **where** *Ord* γ **and** α-*def*: $\alpha =$ *?β* $+ \gamma$
by (*metis assms*(1) *le-Ord-diff Limit-is-Ord*)
from *L-β* **have** *L-βω*: *Limit* (*?β* $+ \omega$) **by** (*blast intro*: *Limit-add-Limit*)
have $\gamma \subset_\circ \omega$

```
  proof(rule ccontr)
    assume ~γ ⊂∘ ω
    with ‹Ord γ› Ord-linear2 have ω ⊆∘ γ by auto
    then obtain δ where γ-def: γ = ω + δ
      by (blast dest: Ord-odiff-eq intro: ‹Ord γ›)
    from α-def have α = (?β + ω) + δ by (simp add: add.assoc γ-def)
    then have ?β + ω ⊆∘ α by (metis add-le-cancel-left0)
    with L-βω have ?β + ω ⊆∘ ?β by auto
    with add-le-cancel-left[of ?β ω 0, THEN iffD1] show False by simp
  qed
  with α-def have γ ∈∘ ω by (auto simp: Ord-mem-iff-lt ‹Ord γ›)
  show ?thesis
  proof
    (
      intro ex1I conjI;
      (elim conjE ex1E allE conjE impE | tactic‹all-tac›);
      (intro conjI | tactic‹all-tac›)
    )
    show γ ∈∘ ω by (rule ‹γ ∈∘ ω›)
    show Limit ?β by (rule ‹Limit ?β›)
    show α = ?β + γ by (rule α-def)
    from α-def show α = ?β + n ⟹ n = γ for n by auto
    show n ∈∘ ω ⟹ Limit β ⟹ α = β + n ⟹ β = ?β for n β
    proof–
      assume prems: n ∈∘ ω Limit β α = β + n
      from L-β prems(2,3) have β ⊆∘ ?β by auto
      then obtain η where β-def: ?β = β + η and Ord η
        by (metis (lifting) L-β Limit-is-Ord le-Ord-diff prems(2))
      moreover have η ∈∘ ω
      proof–
        from α-def β-def have β + η + γ = β + n by (simp add: prems(3))
        then have η + γ = n by (simp add: add.assoc)
        with ‹γ ∈∘ ω› ‹n ∈∘ ω› ‹Ord γ› show η ∈∘ ω
          by (blast intro: calculation(2) Ord-plus-Ord-closed-augend)
      qed
      ultimately show ?thesis
        using prems(2) L-β by (force dest: zero-if-Limit-eq-Limit-plus-vnat)
    qed
  qed
qed

lemma not-Limit-if-in-Limit-plus-omega:
  assumes Limit α and α ∈∘ β and β ∈∘ α + ω
  shows ~Limit β
proof–
  from assms Ord-add have Ord β by blast
  show ?thesis
    using assms(3)
  proof(cases rule: mem-plus-V-E)
    case 1 with mem-not-sym show ?thesis by (auto simp: assms(2,3))
  next
    case (2 z)
    from zero-if-Limit-eq-Limit-plus-vnat[OF - assms(1) 2(2) 2(1)] 2(2) assms(2)
    show ?thesis
      by force
  qed
qed
```

lemma *Limit-plus-omega-vsubset-Limit*:
assumes *Limit* α **and** *Limit* β **and** $\alpha \in_\circ \beta$
shows $\alpha + \omega \subseteq_\circ \beta$
proof–
from *assms*(1) **have** *Lαω*: *Limit* $(\alpha + \omega)$ **by** (*simp add*: *Limit-is-Ord*)
from *not-Limit-if-in-Limit-plus-omega*[*OF assms*(1,3)] *assms*(2) **have**
$\beta \notin_\circ \alpha + \omega$
by *clarsimp*
with *assms*(2) **have** $^\sim\beta \subset_\circ \alpha + \omega$
by (*blast intro*: *Lαω dest*: *Ord-mem-iff-lt Limit-is-Ord*)
then show $\alpha + \omega \subseteq_\circ \beta$ **by** (*meson assms Lαω Limit-is-Ord Ord-linear*2)
qed

lemma *Limit-plus-nat-in-Limit*:
assumes *Limit* α **and** *Limit* β **and** $\alpha \in_\circ \beta$
shows $\alpha + a_{\mathbf{N}} \in_\circ \beta$
using *assms Limit-plus-omega-vsubset-Limit*[*OF assms*] **by** *auto*

lemma *omega*2*-vsubset-Limit*:
assumes *Limit* α **and** $\omega \in_\circ \alpha$
shows $\omega + \omega \subseteq_\circ \alpha$
using *assms* **by** (*simp add*: *Limit-plus-omega-vsubset-Limit*)

2.9 Finite sequences

2.9.1 Background

The section presents a theory of finite sequences internalized in the type V. The content of this subsection was inspired by and draws on many ideas from the content of the theory *List* in the main library of Isabelle/HOL.

2.9.2 Definition and common properties

A finite sequence is defined as a single-valued binary relation whose domain is an initial segment of the set of natural numbers.

locale *vfsequence* = *vsv xs* **for** *xs* +
  assumes *vfsequence-vdomain-in-omega*: $\mathcal{D}_\circ\ xs \in_\circ \omega$

locale *vfsequence-pair* = r_1: *vfsequence* xs_1 + r_2: *vfsequence* xs_2 **for** xs_1 xs_2

Rules.

lemmas [*intro*] = *vfsequence.axioms*(1)

lemma *vfsequenceI*[*intro*]:
  assumes *vsv xs* **and** $\mathcal{D}_\circ\ xs \in_\circ \omega$
  shows *vfsequence xs*
  using *assms* **by** (*simp add*: *vfsequence.intro vfsequence-axioms-def*)

lemma *vfsequenceD*[*dest*]:
  assumes *vfsequence xs*
  shows $\mathcal{D}_\circ\ xs \in_\circ \omega$
  using *assms vfsequence.vfsequence-vdomain-in-omega* **by** *simp*

lemma *vfsequenceE*[*elim*]:
  assumes *vfsequence xs* **and** $\mathcal{D}_\circ\ xs \in_\circ \omega \Longrightarrow P$
  shows *P*
  using *assms* **by** *auto*

lemma *vfsequence-iff*: *vfsequence xs* $\longleftrightarrow$ *vsv xs* $\wedge$ $\mathcal{D}_\circ\ xs \in_\circ \omega$
  using *vfsequence-def* **by** *auto*

Elementary properties.

lemma (**in** *vfsequence*) *vfsequence-vdomain*: $\mathcal{D}_\circ\ xs$ = *vcard xs*
  unfolding *vsv-vcard-vdomain*[*symmetric*] **using** *vfsequence-vdomain-in-omega* **by** *simp*

lemma (**in** *vfsequence*) *vfsequence-vcard-in-omega*[*simp*]: *vcard xs* $\in_\circ \omega$
  using *vfsequence-vdomain-in-omega* **by** (*simp add*: *vfsequence-vdomain*)

Set operations.

lemma *vfsequence-vempty*[*intro*, *simp*]: *vfsequence* 0 **by** (*simp add*: *vfsequenceI*)

lemma *vfsequence-vsingleton*[*intro*, *simp*]: *vfsequence* (*set* $\{\langle 0, a\rangle\}$)
  using *vone-in-omega*
  unfolding *one-V-def*
  by (*intro vfsequenceI*) (*auto simp*: *set-vzero-eq-ord-of-nat-vone*)

lemma (**in** *vfsequence*) *vfsequence-vinsert*:
  vfsequence (*vinsert* $\langle$*vcard xs*, *a*$\rangle$ *xs*)
  using *succ-def succ-in-omega* **by** (*auto simp*: *vfsequence-vdomain*)

Connections.

lemma (in *vfsequence*) *vfsequence-vfinite*[*simp*]: *vfinite xs*
by (*simp add*: *vfinite-vcard-omega-iff*)

lemma (in *vfsequence*) *vfsequence-vlrestriction*[*intro*, *simp*]:
assumes $k \in_\circ \omega$
shows *vfsequence* $(xs \restriction_\circ^l k)$
using *assms* **by** (*force simp*: *vfsequence-vdomain vdomain-vlrestriction*)

lemma *vfsequence-vproduct*:
assumes $n \in_\circ \omega$ **and** $xs \in_\circ (\prod_\circ i \in_\circ n.\ A\ i)$
shows *vfsequence xs*
using *assms* **by** *auto*

lemma *vfsequence-vcpower*:
assumes $n \in_\circ \omega$ **and** $xs \in_\circ A \mathbin{\widehat{}}_\times n$
shows *vfsequence xs*
using *assms vfsequence-vproduct* **by** *auto*

lemma *vfsequence-vcomp-vsv-vfsequence*:
assumes *vsv f* **and** *vfsequence xs* **and** $\mathcal{R}_\circ\ xs \subseteq_\circ \mathcal{D}_\circ\ f$
shows *vfsequence* $(f \circ_\circ xs)$
proof(*intro vfsequenceI vsv-vcomp*)
interpret *xs*: *vfsequence xs* **by** (*rule assms*(2))
show $\mathcal{D}_\circ\ (f \circ_\circ xs) \in_\circ \omega$
unfolding *vdomain-vcomp-vsubset*[*OF assms*(3)]
by (*force simp*: *xs.vfsequence-vdomain-in-omega*)
qed (*auto intro*: *assms*)

Special properties.

lemma (in *vfsequence*) *vfsequence-vdomain-vlrestriction*[*intro*, *simp*]:
assumes $k \in_\circ$ *vcard xs*
shows $\mathcal{D}_\circ\ (xs \restriction_\circ^l k) = k$
using *assms*
by
(
simp add:
OrdmemD
inf-absorb2
order.strict-implies-order
vdomain-vlrestriction
vfsequence-vdomain
)

lemma (in *vfsequence*) *vfsequence-vlrestriction-vcard*[*simp*]:
$xs \restriction_\circ^l$ (*vcard xs*) = *xs*
by (*rule vlrestriction-vdomain*[*unfolded vfsequence-vdomain*])

lemma *vfsequence-vfinite-vcardI*:
assumes *vsv xs* **and** *vfinite xs* **and** $\mathcal{D}_\circ\ xs$ = *vcard xs*
shows *vfsequence xs*
using *assms* **by** (*intro vfsequenceI*) (*auto simp*: *vfinite-vcard-omega*)

lemma (in *vfsequence*) *vfsequence-vrangeE*:
assumes $a \in_\circ \mathcal{R}_\circ\ xs$
obtains *n* **where** $n \in_\circ$ *vcard xs* **and** $xs(\lfloor n \rfloor) = a$
using *assms vfsequence-vdomain* **by** *auto*

lemma (in *vfsequence*) *vfsequence-vrange-vproduct*:
assumes $\bigwedge i.\ i \in_\circ vcard\ xs \Longrightarrow xs(\!|i|\!) \in_\circ A\ i$
shows $xs \in_\circ (\prod_\circ i \in_\circ vcard\ xs.\ A\ i)$
using *vfsequence-vdomain vsv-axioms assms*
by
(
intro vproductI;
(*intro vsv.vsv-vrange-vsubset-vifunion-app* | *tactic*‹*all-tac*›)
) *auto*

lemma (in *vfsequence*) *vfsequence-vrange-vcpower*:
assumes $\mathcal{R}_\circ\ xs \subseteq_\circ A$
shows $xs \in_\circ A\ ^{\frown}_\times\ (vcard\ xs)$
using *assms*
proof(*elim vsubsetE*; *intro vcpowerI*)
assume *hyp*: $x \in_\circ \mathcal{R}_\circ\ xs \Longrightarrow x \in_\circ A$ **for** x
from *vfsequence-vdomain* **show** $xs \in_\circ (\prod_\circ i \in_\circ vcard\ xs.\ A)$
by (*intro vproductI*) (*blast intro*: *hyp elim*: *vdomain-atE*)+
qed

Alternative forms of existing results.

lemmas [*intro*, *simp*] = *vfsequence.vfsequence-vcard-in-omega*
and [*intro*, *simp*] = *vfsequence.vfsequence-vfinite*
and [*intro*, *simp*] = *vfsequence.vfsequence-vlrestriction*
and [*intro*, *simp*] = *vfsequence.vfsequence-vdomain-vlrestriction*
and [*intro*, *simp*] = *vfsequence.vfsequence-vlrestriction-vcard*

2.9.3 Appending an element to a finite sequence: *vcons*

Definition and common properties

definition $vcons :: V \Rightarrow V \Rightarrow V$ (**infixr** ‹$\#_\circ$› 65)
where $xs\ \#_\circ\ x = vinsert\ \langle vcard\ xs, x\rangle\ xs$

Syntax.

abbreviation *vempty-vfsequence* (‹$[]_\circ$›) **where**
$vempty\text{-}vfsequence \equiv 0 :: V$

notation *vempty-vfsequence* (‹$[]_\circ$›)

nonterminal *fsfields*
nonterminal *vlist*

syntax
$:: V \Rightarrow fsfields$ (‹-›)
-fsfields :: $fsfields \Rightarrow V \Rightarrow fsfields$ (‹-,/ -›)
-vlist :: $fsfields \Rightarrow V$ (‹$[(-)]_\circ$›)
-vapp :: $V \Rightarrow fsfields \Rightarrow V$ (‹- $(\!|(-)|\!)_\bullet$› [100, 100] 100)

syntax-consts
-vlist == *vcons* **and**
-vapp == *app*

translations
$[xs, x]_\circ$ == $[xs]_\circ\ \#_\circ\ x$
$[x]_\circ$ == $[]_\circ\ \#_\circ\ x$

translations

$f(\!|xs,\ x|\!)_\bullet = f(\!|[xs,\ x]_\circ|\!)$
$f(\!|x|\!)_\bullet = f(\!|[x]_\circ|\!)$

Rules.

lemma *vconsI*[*intro*!]:
assumes $a \in_\circ$ *vinsert* $\langle$*vcard xs*, *x*$\rangle$ *xs*
shows $a \in_\circ$ *xs* $\#_\circ$ *x*
using *assms* **unfolding** *vcons-def* **by** *clarsimp*

lemma *vconsD*[*dest*!]:
assumes $a \in_\circ$ *xs* $\#_\circ$ *x*
shows $a \in_\circ$ *vinsert* $\langle$*vcard xs*, *x*$\rangle$ *xs*
using *assms* **unfolding** *vcons-def* **by** *clarsimp*

lemma *vconsE*[*elim*!]:
assumes $a \in_\circ$ *xs* $\#_\circ$ *x*
obtains *a* **where** $a \in_\circ$ *vinsert* $\langle$*vcard xs*, *x*$\rangle$ *xs*
using *assms* **unfolding** *vcons-def* **by** *clarsimp*

Elementary properties.

lemma *vcons-neq-vempty*[*simp*]: *ys* $\#_\circ$ *y* $\neq$ $[]_\circ$ **by** *auto*

Set operations.

lemma *vcons-vsingleton*: $[a]_\circ$ = *set* $\{\langle 0_{\mathbb{N}},\ a\rangle\}$ **unfolding** *vcons-def* **by** *simp*

lemma *vcons-vdoubleton*: $[a,\ b]_\circ$ = *set* $\{\langle 0_{\mathbb{N}},\ a\rangle, \langle 1_{\mathbb{N}},\ b\rangle\}$
unfolding *vcons-def*
using *vinsert-vsingleton*
by (*force simp*: *vinsert-set-insert-eq*)

lemma *vcons-vsubset*: *xs* $\subseteq_\circ$ *xs* $\#_\circ$ *x* **by** *clarsimp*

lemma *vcons-vsubset′*:
assumes *vcons xs x* $\subseteq_\circ$ *ys*
shows *vcons xs x* $\subseteq_\circ$ *vcons ys y*
using *assms* **unfolding** *vcons-def* **by** *auto*

Connections.

lemma (**in** *vfsequence*) *vfsequence-vcons*[*intro*, *simp*]: *vfsequence* (*xs* $\#_\circ$ *x*)
proof(*intro vfsequenceI*)
from *vfsequence-vdomain-in-omega vsv-vcard-vdomain* **have** *vcard xs* = $\mathcal{D}_\circ$ *xs*
by (*simp add*: *vcard-veqpoll*)
show *vsv* (*xs* $\#_\circ$ *x*)
proof(*intro vsvI*)
fix *a b c* **assume** *ab*: $\langle a,\ b\rangle \in_\circ$ *xs* $\#_\circ$ *x* **and** *ac*: $\langle a,\ c\rangle \in_\circ$ *xs* $\#_\circ$ *x*
then consider (*dom*) *a* $\in_\circ$ $\mathcal{D}_\circ$ *xs* | (*ndom*) *a* = *vcard xs*
unfolding *vcons-def* **by** *auto*
then show *b* = *c*
proof *cases*
case *dom*
with *ab* **have** $\langle a,\ b\rangle \in_\circ$ *xs*
unfolding *vcons-def* **by** (*auto simp*: ‹*vcard xs* = $\mathcal{D}_\circ$ *xs*›)
moreover from *dom ac* **have** $\langle a,\ c\rangle \in_\circ$ *xs*
unfolding *vcons-def* **by** (*auto simp*: ‹*vcard xs* = $\mathcal{D}_\circ$ *xs*›)
ultimately show *?thesis* **using** *vsv* **by** *simp*
next
case *ndom*

```
    from ab have ⟨a, b⟩ = ⟨vcard xs, x⟩
      unfolding ndom vcons-def using ‹vcard xs = 𝒟₀ xs› mem-not-refl by blast
    moreover from ac have ⟨a, c⟩ = ⟨vcard xs, x⟩
      unfolding ndom vcons-def using ‹vcard xs = 𝒟₀ xs› mem-not-refl by blast
    ultimately show ?thesis by simp
  qed
 next
  show vbrelation (xs #₀ x) unfolding vcons-def
    using vbrelation-vinsertI by auto
 qed
 show 𝒟₀ (xs #₀ x) ∈₀ ω
  unfolding vcons-def
  using succ-in-omega
  by (auto simp: vfsequence-vdomain-in-omega succ-def ‹vcard xs = 𝒟₀ xs›)
qed

lemma (in vfsequence) vfsequence-vcons-vdomain[simp]:
 𝒟₀ (xs #₀ x) = succ (vcard xs)
 by (simp add: succ-def vcons-def vfsequence-vdomain)

lemma (in vfsequence) vfsequence-vcons-vrange[simp]:
 ℛ₀ (xs #₀ x) = vinsert x (ℛ₀ xs)
 by (simp add: vcons-def)

lemma (in vfsequence) vfsequence-vrange-vconsI:
 assumes ℛ₀ xs ⊆₀ X and x ∈₀ X
 shows ℛ₀ (xs #₀ x) ⊆₀ X
 using assms unfolding vcons-def by auto

lemmas vfsequence-vrange-vconsI = vfsequence.vfsequence-vrange-vconsI[rotated 1]
```

Special properties.

```
lemma vcons-vrange-mono:
 assumes xs ⊆₀ ys
 shows ℛ₀ (xs #₀ x) ⊆₀ ℛ₀ (ys #₀ x)
 using assms
 unfolding vcons-def
 by (simp add: vrange-mono vsubset-vinsert-leftI vsubset-vinsert-rightI)

lemma (in vfsequence) vfsequence-vlrestriction-succ:
 assumes [simp]: k ∈₀ vcard xs
 shows xs ↾ᵥ₀ succ k = xs ↾ᵥ₀ k #₀ (xs(|k|))
proof–
 interpret vlr: vfsequence ‹xs ↾ᵥ₀ k›
  using assms by (blast intro: vfsequence-vcard-in-omega Ord-trans)
 from vlr.vfsequence-vdomain[symmetric, simplified] show ?thesis
  by
   (
    simp add:
     vcons-def succ-def vfsequence-vdomain vsv-vlrestriction-vinsert
   )
qed

lemma (in vfsequence) vfsequence-vremove-vcons-vfsequence:
 assumes xs = xs' #₀ x
 shows vfsequence xs'
proof(cases‹⟨vcard xs', x⟩ ∈₀ xs'›)
 case True
```

```
  with assms[unfolded vcons-def] have xs = xs' by auto
  then show ?thesis using vfsequence-axioms by simp
next
  case False
  note x-def[simp] = assms[unfolded vcons-def]
  interpret xs': vsv xs' using vsv-axioms by (auto intro: vsv-vinsertD)
  have fin: vfinite xs' using vfsequence-vfinite by auto
  have vcard-xs: vcard xs = succ (vcard xs') by (simp add: fin False)
  have [simp]: vcard xs' ∉₀ 𝒟₀ xs' using False vsv-axioms by auto
  have vcard xs' ∈₀ ω using fin vfinite-vcard-omega by auto
  have xs'-def: xs' = xs ↾ˡ₀ (vcard xs')
    using vcard-xs fin vfsequence-vdomain
    by (auto simp: vinsert-ident succ-def)
  from vfsequence-vlrestriction[OF ‹vcard xs' ∈₀ ω›] show ?thesis
    unfolding xs'-def[symmetric] .
qed

lemma (in vfsequence) vfsequence-vcons-ex:
  assumes xs ≠ []₀
  obtains xs' x where xs = xs' #₀ x and vfsequence xs'
proof–
  from vcard-vempty have 0 ∈₀ vcard xs by (simp add: assms mem-0-Ord)
  then obtain k where succk: succ k = vcard xs
    by (metis omega-prev vfsequence-vcard-in-omega)
  then have k ∈₀ vcard xs using elts-succ by blast
  from vfsequence-vlrestriction-succ[OF this, unfolded succk] show ?thesis
    by (simp add: vfsequence-vremove-vcons-vfsequence that)
qed
```

Induction and case analysis

```
lemma vfsequence-induct[consumes 1, case-names 0 vcons]:
  assumes vfsequence xs
    and P []₀
    and ⋀xs x. [[vfsequence xs; P xs]] ⟹ P (xs #₀ x)
  shows P xs
proof–
  interpret vfsequence xs by (rule assms(1))
  from assms(1) obtain n where n ∈₀ ω and 𝒟₀ xs = n by auto
  then have n ≤ 𝒟₀ xs by auto
  define P' where P' k = P (xs ↾ˡ₀ k) for k
  from ‹n ∈₀ ω› and ‹n ≤ 𝒟₀ xs› have P' n
  proof(induction rule: omega-induct)
    case (succ n') then show ?case
    proof–
      interpret vlr: vfsequence ‹xs ↾ˡ₀ n'› by (simp add: succ.hyps)
      have P' n' using succ.prems by (force intro: succ.IH)
      then have P (xs ↾ˡ₀ n') unfolding P'-def by assumption
      have n' ∈₀ vcard xs
        using succ.prems by (auto simp: vsubset-iff vfsequence-vdomain)
      from vfsequence-vlrestriction-succ[OF ‹n' ∈₀ vcard xs›]
      show P' (succ n')
        by (simp add: P'-def ‹P (xs ↾ˡ₀ n')› assms(3) vlr.vfsequence-axioms)
    qed
  qed (simp add: P'-def assms(2))
  then show ?thesis unfolding P'-def ‹𝒟₀ xs = n›[symmetric] by simp
qed
```

lemma *vfsequence-cases*[*consumes* 1, *case-names* 0 *vcons*]:
  assumes *vfsequence xs*
    and $xs = []_\circ \Longrightarrow P$
    and $\bigwedge xs'\ x.\ [\![xs = xs' \#_\circ x;\ vfsequence\ xs']\!] \Longrightarrow P$
  shows *P*
proof–
  interpret *vfsequence xs* **by** (*rule assms*(1))
  show *?thesis*
  proof(*cases* ‹*xs* = 0›)
    case *False*
    then obtain *xs*′ *x* **where** $xs = xs' \#_\circ x$
      by (*blast intro*: *vfsequence-vcons-ex*)
    then show *?thesis* **by** (*auto simp*: *assms*(3) *intro*: *vfsequence-vcons-ex*)
  qed (*use assms*(2) **in** *auto*)
qed

Evaluation

lemma (**in** *vfsequence*) *vfsequence-vcard-vcons*[*simp*]:
  $vcard\ (xs \#_\circ x) = succ\ (vcard\ xs)$
proof–
  interpret *xsx*: *vfsequence* ‹*xs* $\#_\circ$ *x*› **by** *simp*
  have $vcard\ (xs \#_\circ x) = \mathcal{D}_\circ\ (xs \#_\circ x)$
    by (*rule xsx.vfsequence-vdomain*[*symmetric*])
  then show *?thesis*
    by (*subst vcons-def*) (*simp add*: *succ-def vcons-def vfsequence-vdomain*)
qed

lemma (**in** *vfsequence*) *vfsequence-at-last*[*intro*, *simp*]:
  assumes $i = vcard\ xs$
  shows $(xs \#_\circ x)(\!|i|\!) = x$
  by (*simp add*: *vfsequence-vdomain vcons-def assms*)

lemma (**in** *vfsequence*) *vfsequence-at-not-last*[*intro*, *simp*]:
  assumes $i \in_\circ vcard\ xs$
  shows $(xs \#_\circ x)(\!|i|\!) = xs(\!|i|\!)$
proof–
  from *assms* **have** [*simp*]: $\mathcal{D}_\circ\ xs = vcard\ xs$ **by** (*auto simp*: *vfsequence-vdomain*)
  from *assms* **have** $i \in_\circ \mathcal{D}_\circ\ xs$ **by** *simp*
  moreover have $i \neq vcard\ xs$ **using** *assms mem-not-refl* **by** *blast*
  ultimately show *?thesis*
    unfolding *vcons-def* **using** *vsv.vsv-vinsert vsvE vsv-axioms* **by** *auto*
qed

Alternative forms of existing results.

lemmas [*intro*, *simp*] = *vfsequence.vfsequence-vcons*
  and [*intro*, *simp*] = *vfsequence.vfsequence-vcard-vcons*
  and [*intro*, *simp*] = *vfsequence.vfsequence-at-last*
  and [*intro*, *simp*] = *vfsequence.vfsequence-at-not-last*
  and [*intro*, *simp*] = *vfsequence.vfsequence-vcons-vdomain*
  and [*intro*, *simp*] = *vfsequence.vfsequence-vcons-vrange*

Congruence-like properties

context *vfsequence-pair*
begin

lemma *vcons-eq-vcard-eq*:

```
  assumes xs₁ #∘ x₁ = xs₂ #∘ x₂
  shows vcard xs₁ = vcard xs₂
  by
    (
      metis
        assms
        succ-inject-iff
        vfsequence.vfsequence-vcons-vdomain
        r₁.vfsequence-axioms
        r₂.vfsequence-axioms
    )

lemma vcons-eqD[dest]:
  assumes xs₁ #∘ x₁ = xs₂ #∘ x₂
  shows xs₁ = xs₂ and x₁ = x₂
proof–

  have xsx1-last: (xs₁ #∘ x₁)(|vcard xs₁|) = x₁ by simp
  have xsx2-last: (xs₂ #∘ x₂)(|vcard xs₂|) = x₂ by simp

  from assms have vcard: vcard xs₁ = vcard xs₂ by (rule vcons-eq-vcard-eq)
  from trans[OF xsx1-last xsx1-last[unfolded vcard assms, symmetric]]

  show x₁ = x₂ unfolding xsx1-last xsx2-last .

  have nxs1: ⟨vcard xs₁, x₁⟩ ∉∘ xs₁
    using mem-not-refl r₁.vfsequence-vdomain by blast
  have nxs2: ⟨vcard xs₂, x₂⟩ ∉∘ xs₂
    using mem-not-refl r₂.vfsequence-vdomain by blast
  have xsx1-xsx2: ⟨vcard xs₁, x₁⟩ = ⟨vcard xs₂, x₂⟩
    unfolding vcons-eq-vcard-eq[OF assms(1)] ‹x₁ = x₂› by simp

  show xs₁ = xs₂
  proof(rule vinsert-identD[OF - nxs1])
    from assms(1)[unfolded vcons-def] show
      vinsert ⟨vcard xs₁, x₁⟩ xs₁ = vinsert ⟨vcard xs₁, x₁⟩ xs₂
      by (auto simp: xsx1-xsx2)
    show ⟨vcard xs₁, x₁⟩ ∉∘ xs₂
      by (rule nxs2[folded ‹x₁ = x₂› vcons-eq-vcard-eq[OF assms(1)]])
  qed

qed

lemma vcons-eqI:
  assumes xs₁ = xs₂ and x₁ = x₂
  shows xs₁ #∘ x₁ = xs₂ #∘ x₂
  using assms by (rule arg-cong2)

lemma vcons-eq-iff[simp]: (xs₁ #∘ x₁ = xs₂ #∘ x₂) ⟷ (xs₁ = xs₂ ∧ x₁ = x₂)
  by auto

end
```

Alternative forms of existing results.

```
context
  fixes xs₁ xs₂
  assumes xs₁: vfsequence xs₁
    and xs₂: vfsequence xs₂
```

begin

lemmas-with[*OF vfsequence-pair.intro*[*OF* xs_1 xs_2]]:
vcons-eqD′ = vfsequence-pair.vcons-eqD
and *vcons-eq-iff*[*intro*, *simp*] = *vfsequence-pair.vcons-eq-iff*

end

lemmas *vcons-eqD*[*dest*] = *vcons-eqD′*[*rotated* −1]

2.9.4 Transfer between the type *V list* and finite sequences

Initialization

primrec *vfsequence-of-vlist* :: *V list* ⇒ *V*
where
vfsequence-of-vlist [] = 0
| *vfsequence-of-vlist* (*x* # *xs*) = *vfsequence-of-vlist xs* #∘ *x*

definition *vlist-of-vfsequence* :: *V* ⇒ *V list*
where *vlist-of-vfsequence* = *inv-into UNIV vfsequence-of-vlist*

lemma *vfsequence-vfsequence-of-vlist*: *vfsequence* (*vfsequence-of-vlist xs*)
by (*induction xs*) *auto*

lemma *inj-vfsequence-of-vlist*: *inj vfsequence-of-vlist*
proof
show *vfsequence-of-vlist x* = *vfsequence-of-vlist y* ⟹ *x* = *y*
for *x y*
proof(*induction y arbitrary*: *x*)
case *Nil* **then show** *?case* **by** (*cases x*) *auto*
next
case (*Cons a ys*)
note *Cons′* = *Cons*
show *?case*
proof(*cases x*)
case *Nil* **with** *Cons* **show** *?thesis* **by** *auto*
next
case (*Cons b zs*)
from *Cons′*[*unfolded Cons vfsequence-of-vlist.simps*] **have**
vfsequence-of-vlist zs #∘ *b* = *vfsequence-of-vlist ys* #∘ *a*
by *simp*
then have *vfsequence-of-vlist zs* = *vfsequence-of-vlist ys* **and** *b* = *a*
by (*auto simp*: *vfsequence-vfsequence-of-vlist*)
from *Cons′*(1)[*OF this*(1)] *this*(2) **show** *?thesis* **unfolding** *Cons* **by** *auto*
qed
qed
qed

lemma *range-vfsequence-of-vlist*:
range vfsequence-of-vlist = {*xs*. *vfsequence xs*}
proof(*intro subset-antisym subsetI*; *unfold mem-Collect-eq*)
show *xs* ∈ *range vfsequence-of-vlist* ⟹ *vfsequence xs* **for** *xs*
by (*clarsimp simp*: *vfsequence-vfsequence-of-vlist*)
fix *xs* **assume** *vfsequence xs*
then show *xs* ∈ *range vfsequence-of-vlist*
proof(*induction rule*: *vfsequence-induct*)
case 0 **then show** *?case*

```
    by (metis image-iff iso-tuple-UNIV-I vfsequence-of-vlist.simps(1))
  next
    case (vcons xs x) then show ?case
    by (metis rangeE rangeI vfsequence-of-vlist.simps(2))
  qed
qed

lemma vlist-of-vfsequence-vfsequence-of-vlist[simp]:
  vlist-of-vfsequence (vfsequence-of-vlist xs) = xs
  by (simp add: inj-vfsequence-of-vlist vlist-of-vfsequence-def)

lemma (in vfsequence) vfsequence-of-vlist-vlist-of-vfsequence[simp]:
  vfsequence-of-vlist (vlist-of-vfsequence xs) = xs
  using vfsequence-axioms range-vfsequence-of-vlist inj-vfsequence-of-vlist
  by (simp add: f-inv-into-f vlist-of-vfsequence-def)

lemmas vfsequence-of-vlist-vlist-of-vfsequence[intro, simp] =
  vfsequence.vfsequence-of-vlist-vlist-of-vfsequence

lemma vlist-of-vfsequence-vempty[simp]: vlist-of-vfsequence []ₒ = []
  by
    (
      metis
        vfsequence-of-vlist.simps(1)
        vlist-of-vfsequence-vfsequence-of-vlist
    )
```

Transfer relation 1.

```
definition cr-vfsequence :: V ⇒ V list ⇒ bool
  where cr-vfsequence a b ⟷ (a = vfsequence-of-vlist b)

lemma cr-vfsequence-right-total[transfer-rule]: right-total cr-vfsequence
  unfolding cr-vfsequence-def right-total-def by simp

lemma cr-vfsequence-bi-unqie[transfer-rule]: bi-unique cr-vfsequence
  unfolding cr-vfsequence-def bi-unique-def
  by (simp add: inj-eq inj-vfsequence-of-vlist)

lemma cr-vfsequence-transfer-domain-rule[transfer-domain-rule]:
  Domainp cr-vfsequence = (λxs. vfsequence xs)
  unfolding cr-vfsequence-def
proof(intro HOL.ext, rule iffI)
  fix xs assume prems: vfsequence xs
  interpret vfsequence xs by (rule prems)
  have ∃ys. xs = vfsequence-of-vlist ys
    using prems
  proof(induction rule: vfsequence-induct)
    show ⟦ vfsequence xs; ∃ys. xs = vfsequence-of-vlist ys ⟧ ⟹
      ∃ys. xs #ₒ x = vfsequence-of-vlist ys
      for xs x
      unfolding vfsequence-of-vlist-def by (metis list.simps(7))
  qed auto
  then show Domainp (λa b. a = vfsequence-of-vlist b) xs by auto
qed (clarsimp simp: vfsequence-vfsequence-of-vlist)

lemma cr-vfsequence-vconsD:
  assumes cr-vfsequence (xs #ₒ x) (y # ys)
  shows cr-vfsequence xs ys and x = y
```

proof–
from *assms*[*unfolded cr-vfsequence-def*] **have** *xs-x-def*:
xs #$_\circ$ *x* = *vfsequence-of-vlist* (*y* # *ys*) **.**
then have *xs-x*: *vfsequence* (*xs* #$_\circ$ *x*)
by (*simp add*: *vfsequence-vfsequence-of-vlist*)
interpret *vfsequence xs*
by (*blast intro*: *vfsequence.vfsequence-vremove-vcons-vfsequence xs-x*)
from
assms[*unfolded cr-vfsequence-def vfsequence-of-vlist.simps*(2)]
vfsequence-axioms
show *cr-vfsequence xs ys* **and** *x* = *y*
unfolding *cr-vfsequence-def* **by** (*auto simp*: *vfsequence-vfsequence-of-vlist*)
qed

Transfer relation 2.

definition *cr-cr-vfsequence* :: *V* ⇒ *V list list* ⇒ *bool*
where *cr-cr-vfsequence a b* ⟷
(*a* = *vfsequence-of-vlist* (*map vfsequence-of-vlist b*))

lemma *cr-cr-vfsequence-right-total*[*transfer-rule*]:
right-total cr-cr-vfsequence
unfolding *cr-cr-vfsequence-def right-total-def* **by** *simp*

lemma *cr-cr-vfsequence-bi-unqie*[*transfer-rule*]: *bi-unique cr-cr-vfsequence*
unfolding *cr-cr-vfsequence-def bi-unique-def*
by (*simp add*: *inj-eq inj-vfsequence-of-vlist*)

Transfer relation for scalars.

definition *cr-scalar* :: (*V* ⇒ *'a* ⇒ *bool*) ⇒ *V* ⇒ *'a* ⇒ *bool*
where *cr-scalar R x y* = (∃ *a*. *x* = [*a*]$_\circ$ ∧ *R a y*)

lemma *cr-scalar-bi-unique*[*transfer-rule*]:
assumes *bi-unique R*
shows *bi-unique* (*cr-scalar R*)
using *assms* **unfolding** *cr-scalar-def bi-unique-def* **by** *auto*

lemma *cr-scalar-right-total*[*transfer-rule*]:
assumes *right-total R*
shows *right-total* (*cr-scalar R*)
using *assms* **unfolding** *cr-scalar-def right-total-def* **by** *simp*

lemma *cr-scalar-transfer-domain-rule*[*transfer-domain-rule*]:
Domainp (*cr-scalar R*) = (λ*x*. ∃ *a*. *x* = [*a*]$_\circ$ ∧ *Domainp R a*)
unfolding *cr-scalar-def* **by** *auto*

Transfer rules for previously defined entities

context
includes *lifting-syntax*
begin

lemma *vfsequence-vempty-transfer*[*transfer-rule*]: *cr-vfsequence* []$_\circ$ []
unfolding *cr-vfsequence-def* **by** *simp*

lemma *vfsequence-vempty-ll-transfer*[*transfer-rule*]:
cr-cr-vfsequence [[]$_\circ$]$_\circ$ [[]]
unfolding *cr-cr-vfsequence-def* **by** *simp*

```
lemma vcons-transfer[transfer-rule]:
  ((=) ===> cr-vfsequence ===> cr-vfsequence) (λx xs. xs #∘ x) (λx xs. x # xs)
  by (intro rel-funI) (simp add: cr-vfsequence-def)

lemma vcons-ll-transfer[transfer-rule]:
  (cr-vfsequence ===> cr-cr-vfsequence ===> cr-cr-vfsequence)
    (λx xs. xs #∘ x) (λx xs. x # xs)
  by (intro rel-funI) (simp add: cr-vfsequence-def cr-cr-vfsequence-def)

lemma vfsequence-vrange-transfer[transfer-rule]:
  (cr-vfsequence ===> (=)) (λxs. elts (ℛ∘ xs)) list.set
proof(intro rel-funI)
  fix xs ys assume prems: cr-vfsequence xs ys
  then have xs = vfsequence-of-vlist ys unfolding cr-vfsequence-def by simp
  then have vfsequence xs by (simp add: vfsequence-vfsequence-of-vlist)
  from this prems show elts (ℛ∘ xs) = list.set ys
  proof(induction ys arbitrary: xs)
    case (Cons a ys)
    from Cons(2) show ?case
    proof(cases xs rule: vfsequence-cases)
      case 0 with Cons show ?thesis by (simp add: Cons.IH cr-vfsequence-def)
    next
      case (vcons xs' x)
      interpret vfsequence xs' by (rule vcons(2))
      note vcons-transfer = cr-vfsequence-vconsD[OF Cons(3)[unfolded vcons(1)]]
      have a-ys: list.set (a # ys) = insert a (list.set ys) by simp
      from vcons(2) have R-xs'x: ℛ∘ (xs' #∘ x) = vinsert x (ℛ∘ xs') by simp
      show elts (ℛ∘ xs) = (list.set (a # ys))
        unfolding vcons(1) R-xs'x a-ys
        by
          (
            auto simp:
              vcons-transfer(2) Cons(1)[OF vfsequence-axioms vcons-transfer(1)]
          )
    qed
  qed (auto simp: cr-vfsequence-def)
qed

lemma vcard-transfer[transfer-rule]:
  (cr-vfsequence ===> cr-omega) vcard length
proof(intro rel-funI)
  fix xs ys assume prems: cr-vfsequence xs ys
  then have xs = vfsequence-of-vlist ys unfolding cr-vfsequence-def by simp
  then have vfsequence xs by (simp add: vfsequence-vfsequence-of-vlist)
  from this prems show cr-omega (vcard xs) (length ys)
  proof(induction ys arbitrary: xs)
    case (Cons y ys)
    from Cons(2) show ?case
    proof(cases xs rule: vfsequence-cases)
      case 0 with Cons show ?thesis by (simp add: Cons.IH cr-vfsequence-def)
    next
      case (vcons xs' x)
      interpret vfsequence xs' by (rule vcons(2))
      note vcons-transfer = cr-vfsequence-vconsD[OF Cons(3)[unfolded vcons(1)]]
      have vcard-xs-x: vcard (xs' #∘ x) = succ (vcard xs') by simp
      have vcard-y-ys: length (y # ys) = Suc (length ys) by simp
      from vfsequence-axioms have [transfer-rule]:
        cr-omega (vcard xs') (length ys)
```

```
      by (simp add: vcons-transfer(1) Cons.IH)
    show ?thesis unfolding vcons(1) vcard-xs-x vcard-y-ys by transfer-prover
  qed
 qed (auto simp: cr-omega-def cr-vfsequence-def)
qed

lemma vcard-ll-transfer[transfer-rule]:
 (cr-cr-vfsequence ===> cr-omega) vcard length
 unfolding cr-cr-vfsequence-def
 by (intro rel-funI)
   (metis cr-vfsequence-def length-map rel-funD vcard-transfer)

end
```

Corollaries.

```
lemma vdomain-vfsequence-of-vlist: 𝒟∘ (vfsequence-of-vlist xs) = length xs
proof–
 define ys where ys = vfsequence-of-vlist xs
 interpret vfsequence ys
   unfolding ys-def by (rule vfsequence-vfsequence-of-vlist)
 have [transfer-rule]: cr-vfsequence ys xs
   unfolding ys-def cr-vfsequence-def by simp-all
 show ?thesis
   by (fold ys-def, unfold vfsequence-vdomain, transfer) simp
qed

lemma vrange-vfsequence-of-vlist:
 ℛ∘ (vfsequence-of-vlist xs) = set (list.set xs)
proof(intro vsubset-antisym vsubsetI)
 fix x assume prems: x ∈∘ ℛ∘ (vfsequence-of-vlist xs)
 define ys where ys = vfsequence-of-vlist xs
 have [transfer-rule]: cr-vfsequence ys xs x = x
   unfolding ys-def cr-vfsequence-def by simp-all
 show x ∈∘ set (list.set xs) by transfer (simp add: prems[folded ys-def])
next
 fix x assume prems: x ∈∘ set (list.set xs)
 define ys where ys = vfsequence-of-vlist xs
 have [transfer-rule]: cr-vfsequence ys xs x = x
   unfolding ys-def cr-vfsequence-def by simp-all
 from prems[untransferred] show x ∈∘ ℛ∘ (vfsequence-of-vlist xs)
   unfolding ys-def by simp
qed

lemma cr-cr-vfsequence-transfer-domain-rule[transfer-domain-rule]:
 Domainp cr-cr-vfsequence =
   (λxss. vfsequence xss ∧ (∀ xs∈∘ℛ∘ xss. vfsequence xs))
proof(intro HOL.ext, rule iffI; (elim conjE | intro conjI ballI))
 fix xss assume prems: Domainp cr-cr-vfsequence xss
 with vfsequence-vfsequence-of-vlist show xss: vfsequence xss
   unfolding cr-cr-vfsequence-def by clarsimp
 interpret vfsequence xss by (rule xss)
 fix xs assume prems': xs ∈∘ ℛ∘ xss
 from prems obtain yss where xss-def:
   xss = vfsequence-of-vlist (map vfsequence-of-vlist yss)
   unfolding cr-cr-vfsequence-def by clarsimp
 from prems' have xs ∈∘ set (list.set (map vfsequence-of-vlist yss))
   unfolding xss-def vrange-vfsequence-of-vlist by simp
 then obtain ys where xs-def: xs = vfsequence-of-vlist ys by clarsimp
```

```
  show vfsequence xs
    unfolding xs-def by (simp add: vfsequence-vfsequence-of-vlist)
next
  fix xss assume prems: vfsequence xss ∀ xs∈ₒ 𝒟ₒ xss. vfsequence xs
  have ∃ yss. xss = vfsequence-of-vlist (map vfsequence-of-vlist yss)
    using prems
  proof(induction rule: vfsequence-induct)
    case (vcons xss x)
    let ?y = ‹vlist-of-vfsequence x›
    from vcons(2,3) obtain yss where xss-def:
      xss = vfsequence-of-vlist (map vfsequence-of-vlist yss)
      by auto
    from vcons(3) have vfsequence x by auto
    then have x-def: x = vfsequence-of-vlist (vlist-of-vfsequence x) by simp
    then have
      xss ##ₒ x = vfsequence-of-vlist (map vfsequence-of-vlist (?y # yss))
      unfolding xss-def by simp
    then show ?case by blast
  qed (auto intro: exI[of - ‹[]›])
  then show Domainp cr-cr-vfsequence xss
    unfolding cr-cr-vfsequence-def by blast
qed
```

Appending elements

```
definition vappend :: V ⇒ V ⇒ V (infixr ‹@ₒ› 65)
  where xs @ₒ ys =
    vfsequence-of-vlist (vlist-of-vfsequence ys @ vlist-of-vfsequence xs)
```

Transfer.

```
lemma vappend-transfer[transfer-rule]:
  includes lifting-syntax
  shows (cr-vfsequence ===> cr-vfsequence ===> cr-vfsequence)
    (λxs ys. vappend ys xs) append
  by (intro rel-funI, unfold cr-vfsequence-def) (simp add: vappend-def)

lemma vappend-ll-transfer[transfer-rule]:
  includes lifting-syntax
  shows (cr-cr-vfsequence ===> cr-cr-vfsequence ===> cr-cr-vfsequence)
    (λxs ys. vappend ys xs) append
  by (intro rel-funI, unfold cr-cr-vfsequence-def) (simp add: vappend-def)
```

Elementary properties.

```
lemma (in vfsequence) vfsequence-vappend-vempty-vfsequence[simp]:
  []ₒ @ₒ xs = xs
  unfolding vappend-def by auto

lemmas vfsequence-vappend-vempty-vfsequence[simp] =
  vfsequence.vfsequence-vappend-vempty-vfsequence

lemma (in vfsequence) vfsequence-vappend-vfsequence-vempty[simp]:
  xs @ₒ []ₒ = xs
  unfolding vappend-def by auto

lemmas vfsequence-vappend-vfsequence-vempty[simp] =
  vfsequence.vfsequence-vappend-vfsequence-vempty

lemma vappend-vcons[simp]:
```

assumes *vfsequence xs* **and** *vfsequence ys*
shows *xs* $@_\circ$ (*ys* $\#_\circ$ *y*) = (*xs* $@_\circ$ *ys*) $\#_\circ$ *y*
using *append-Cons*[**where** *'a=V*, *untransferred*, *OF assms*(2,1)] **by** *simp*

Distinct elements

definition *vdistinct* :: *V* ⇒ *bool*
where *vdistinct xs* = *distinct* (*vlist-of-vfsequence xs*)

Transfer.

lemma *vdistinct-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vfsequence* ===> (=)) *vdistinct distinct*
by (*intro rel-funI*, *unfold cr-vfsequence-def*) (*simp add*: *vdistinct-def*)

lemma *vdistinct-ll-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-cr-vfsequence* ===> (=)) *vdistinct distinct*
by (*intro rel-funI*, *unfold cr-cr-vfsequence-def*)
(
metis
vdistinct-def
distinct-map
inj-onI
vlist-of-vfsequence-vfsequence-of-vlist
)

Elementary properties.

lemma (**in** *vfsequence*) *vfsequence-vdistinct-if-vcard-vrange-eq-vcard*:
assumes *vcard* ($\mathcal{R}_\circ$ *xs*) = *vcard xs*
shows *vdistinct xs*
proof–
have *finite* (*elts* ($\mathcal{R}_\circ$ *xs*)) **by** (*simp add*: *assms vcard-vfinite-omega*)
from *vcard-finite-set*[*OF this*] *assms* **have** *card* (*elts* ($\mathcal{R}_\circ$ *xs*))$_\mathbb{N}$ = *vcard xs*
by *simp*
from *card-distinct*[**where** *?'a=V*, *untransferred*, *OF vfsequence-axioms this*]
show *?thesis*.
qed

lemma *vdistinct-vempty*[*intro*, *simp*]: *vdistinct* []$_\circ$
proof–
have *t*: *distinct* ([]::*V list*) **by** *simp*
show *?thesis* **by** (*rule t*[*untransferred*])
qed

lemma (**in** *vfsequence*) *vfsequence-vcons-vdistinct*:
assumes *vdistinct* (*xs* $\#_\circ$ *x*)
shows *vdistinct xs*
proof–
from *distinct.simps*(2)[**where** *'a=V*, *THEN iffD1*, *THEN conjunct2*, *untransferred*]
show *?thesis*
using *vfsequence-axioms assms* **by** *simp*
qed

lemma (**in** *vfsequence*) *vfsequence-vcons-nin-vrange*:
assumes *vdistinct* (*xs* $\#_\circ$ *x*)
shows *x* $\notin_\circ$ $\mathcal{R}_\circ$ *xs*
proof–

```
  from distinct.simps(2)[where 'a=V, THEN iffD1, THEN conjunct1, untransferred]
  show ?thesis
    using vfsequence-axioms assms by simp
qed

lemma (in vfsequence) vfsequence-v11I[intro]:
  assumes vdistinct xs
  shows v11 xs
  using vfsequence-axioms assms
proof(induction xs rule: vfsequence-induct)
  case (vcons xs x)
  interpret vfsequence xs by (rule vcons(1))
  from vcons(3) have dxs: vdistinct xs by (rule vfsequence-vcons-vdistinct)
  interpret v11 xs using dxs by (rule vcons(2))
  from vfsequence-vcons-nin-vrange[OF vcons(3)] have x ∉ₒ ℛₒ xs .
  show v11 (xs #ₒ x)
    by
      (
        simp-all add:
          vcons-def vfsequence-vdomain vfsequence-vcons-nin-vrange[OF vcons(3)]
      )
qed simp

lemma (in vfsequence) vfsequence-vcons-vdistinctI:
  assumes vdistinct xs and x ∉ₒ ℛₒ xs
  shows vdistinct (xs #ₒ x)
proof–
  have t: distinct xs ⟹ x ∉ list.set xs ⟹ distinct (x # xs)
    for x :: V and xs
    by simp
  from vfsequence-axioms assms show ?thesis by (rule t[untransferred])
qed

lemmas vfsequence-vcons-vdistinctI[intro] =
  vfsequence.vfsequence-vcons-vdistinctI

lemma (in vfsequence) vfsequence-nin-vrange-vcons:
  assumes y ∉ₒ ℛₒ xs and y ≠ x
  shows y ∉ₒ ℛₒ (xs #ₒ x)
proof–
  have t: y ∉ list.set xs ⟹ y ≠ x ⟹ y ∉ list.set (x # xs)
    for x y :: V and xs
    by simp
  from vfsequence-axioms assms show ?thesis by (rule t[untransferred])
qed

lemmas vfsequence-nin-vrange-vcons[intro] =
  vfsequence.vfsequence-nin-vrange-vcons
```

Concatenation of sequences

```
definition vconcat :: V ⇒ V
  where vconcat xss =
    vfsequence-of-vlist(
      concat (map vlist-of-vfsequence (vlist-of-vfsequence xss))
    )
```

Transfer.

lemma *vconcat-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-cr-vfsequence* ===> *cr-vfsequence*) *vconcat concat*
proof(*intro rel-funI*)
fix *xs ys* **assume** *cr-cr-vfsequence xs ys*
then have *xs-def*: *xs* = *vfsequence-of-vlist* (*map vfsequence-of-vlist ys*)
unfolding *cr-cr-vfsequence-def* **by** *simp*
have *main-eq*: *map vlist-of-vfsequence* (*vlist-of-vfsequence xs*) = *ys*
unfolding *xs-def* **by** (*simp add*: *map-idI*)
show *cr-vfsequence* (*vconcat xs*) (*concat ys*)
unfolding *cr-vfsequence-def vconcat-def main-eq* **..**
qed

Elementary properties.

lemma *vconcat-vempty*[*simp*]: *vconcat* []$_\circ$ = []$_\circ$
unfolding *vconcat-def* **by** *simp*

lemma *vconcat-append*[*simp*]:
assumes *vfsequence xss*
and $\forall$ *xs*$\in_\circ\mathcal{R}_\circ$ *xss*. *vfsequence xs*
and *vfsequence yss*
and $\forall$ *xs*$\in_\circ\mathcal{R}_\circ$ *yss*. *vfsequence xs*
shows *vconcat* (*xss* @$_\circ$ *yss*) = *vconcat xss* @$_\circ$ *vconcat yss*
using *assms concat-append*[**where** *'a*=*V*, *untransferred*] **by** *simp*

lemma *vconcat-vcons*[*simp*]:
assumes *vfsequence xs* **and** *vfsequence xss* **and** $\forall$ *xs*$\in_\circ\mathcal{R}_\circ$ *xss*. *vfsequence xs*
shows *vconcat* (*xss* #$_\circ$ *xs*) = *vconcat xss* @$_\circ$ *xs*
using *assms concat.simps*(2)[**where** *'a*=*V*, *untransferred*] **by** *simp*

lemma (**in** *vfsequence*) *vfsequence-vconcat-fsingleton*[*simp*]: *vconcat* [*xs*]$_\circ$ = *xs*
using *vfsequence-axioms*
by
(
metis
vfsequence-vappend-vempty-vfsequence
vconcat-vcons
vconcat-vempty
vempty-nin
vfsequence-vempty
vrange-vempty
)

lemmas *vfsequence-vconcat-fsingleton*[*simp*] =
vfsequence.vfsequence-vconcat-fsingleton

2.9.5 Finite sequences and the Cartesian product

lemma *vfsequence-vcons-vproductI*[*intro*!]:
assumes *n* $\in_\circ$ ω
and *xs* $\in_\circ$ ($\prod_\circ i\in_\circ$*vcard xs*. *A i*)
and *x* $\in_\circ$ *A* (*vcard xs*)
and *n* = *vcard* (*xs* #$_\circ$ *x*)
shows *xs* #$_\circ$ *x* $\in_\circ$ ($\prod_\circ i\in_\circ n$. *A i*)
proof
interpret *xs*: *vfsequence xs*
using *assms*
apply(*intro vfsequenceI*)

```
    subgoal by auto
    subgoal
      by
        (
          metis
            vcard-vfinite-omega
            vcons-vsubset
            vfinite-vcard-omega
            vfinite-vsubset vproductD(2)
        )
    done
  interpret xsx: vfsequence ‹xs #∘ x› by auto
  show vsv (xs #∘ x) by (simp add: xsx.vsv-axioms)
  show D: 𝒟∘ (xs #∘ x) = n unfolding assms(4) xsx.vfsequence-vdomain by auto
  from vproductD[OF assms(2)] have elem: i ∈∘ vcard xs ⟹ xs(|i|) ∈∘ A i for i
    by auto
  show ∀ i∈∘n. (xs #∘ x)(|i|) ∈∘ A i by (auto simp: elem assms(3,4))
qed

lemma vfsequence-vcons-vproductD[dest]:
  assumes xs #∘ x ∈∘ (∏∘i∈∘n. A i) and n ∈∘ ω
  shows xs ∈∘ (∏∘i∈∘vcard xs. A i)
    and x ∈∘ A (vcard xs)
    and n = vcard (xs #∘ x)
proof–

  interpret xsx: vfsequence ‹xs #∘ x›
    by (meson assms succ-in-omega vfsequence-vproduct)
  interpret xs: vfsequence xs
    by (blast intro: xsx.vfsequence-vremove-vcons-vfsequence)

  show n-def: n = vcard (xs #∘ x)
    using assms using xsx.vfsequence-vdomain by blast
  from vproductD[OF assms(1), unfolded n-def]
  have elem-xs-x: i ∈∘ vcard (xs #∘ x) ⟹ (xs #∘ x)(|i|) ∈∘ A i
    for i
    by auto
  then have elem-xs[simp]: i ∈∘ vcard xs ⟹ xs(|i|) ∈∘ A i for i
    by (metis rev-vsubsetD vcard-mono vcons-vsubset xs.vfsequence-at-not-last)
  show xs ∈∘ (∏∘i∈∘vcard xs. A i)
    by (auto simp: xs.vsv-axioms xs.vfsequence-vdomain)
  from elem-xs-x show x ∈∘ A (vcard xs) by fastforce

qed

lemma vfsequence-vcons-vproductE[elim!]:
  assumes xs #∘ x ∈∘ (∏∘i∈∘n. A i) and n ∈∘ ω
  obtains xs ∈∘ (∏∘i∈∘vcard xs. A i)
    and x ∈∘ A (vcard xs)
    and n = vcard (xs #∘ x)
  using assms by (auto simp: vfsequence-vcons-vproductD)
```

2.9.6 Binary Cartesian product based on finite sequences: *ftimes*

```
definition ftimes :: V ⇒ V ⇒ V (infixr ‹×•› 80)
  where ftimes a b ≡ (∏∘i∈∘2ℕ. if i = 0 then a else b)

lemma small-fpairs[simp]: small {[a, b]∘ | a b. [a, b]∘ ∈∘ r}
```

by (*rule down*[*of - r*]) *clarsimp*

Rules.

lemma *ftimesI*1[*intro*]:
assumes $x = [a, b]_\circ$ **and** $a \in_\circ A$ **and** $b \in_\circ B$
shows $x \in_\circ A \times_\bullet B$
unfolding *ftimes-def*
proof
show *vsv*: *vsv x* **by** (*simp add*: *assms*(1) *vfsequence.axioms*(1))
then interpret *vsv x* **.**
from *assms* **show** *D*: $\mathcal{D}_\circ\ x = 2_\mathbb{N}$
unfolding *nat-omega-simps two One-nat-def* **by** *auto*
from *assms*(2,3) **have** *i*: $i \in_\circ 2_\mathbb{N} \Longrightarrow x(\!|i|\!) \in_\circ (\textit{if } i = 0_\mathbb{N} \textit{ then } A \textit{ else } B)$
for *i*
unfolding *assms*(1) *two nat-omega-simps One-nat-def* **by** *auto*
from *i* **show** $\forall\, i \in_\circ 2_\mathbb{N}.\ x(\!|i|\!) \in_\circ (\textit{if } i = 0 \textit{ then } A \textit{ else } B)$ **by** *auto*
qed

lemma *ftimesI*2[*intro*!]:
assumes $a \in_\circ A$ **and** $b \in_\circ B$
shows $[a, b]_\circ \in_\circ A \times_\bullet B$
using *assms ftimesI*1 **by** *auto*

lemma *fproductE*1[*elim*!]:
assumes $x \in_\circ A \times_\bullet B$
obtains *a b* **where** $x = [a, b]_\circ$ **and** $a \in_\circ A$ **and** $b \in_\circ B$
proof–
from *vproduct-vdoubletonD*[*OF assms*[*unfolded two ftimes-def*]]
have *x-def*: $x = set\ \{\langle 0_\mathbb{N}, x(\!|0_\mathbb{N}|\!)\rangle, \langle 1_\mathbb{N}, x(\!|1_\mathbb{N}|\!)\rangle\}$
and $x(\!|0_\mathbb{N}|\!) \in_\circ A$
and $x(\!|1_\mathbb{N}|\!) \in_\circ B$
by *auto*
then show *?thesis* **using** *that* **using** *vcons-vdoubleton* **by** *simp*
qed

lemma *fproductE*2[*elim*!]:
assumes $[a, b]_\circ \in_\circ A \times_\bullet B$ **obtains** $a \in_\circ A$ **and** $b \in_\circ B$
using *assms* **by** *blast*

Set operations.

lemma *vfinite-0-left*[*simp*]: $0 \times_\bullet b = 0$
by (*meson eq0-iff fproductE*1)

lemma *vfinite-0-right*[*simp*]: $a \times_\bullet 0 = 0$
by (*meson eq0-iff fproductE*1)

lemma *fproduct-vintersection*: $(a \cap_\circ b) \times_\bullet (c \cap_\circ d) = (a \times_\bullet c) \cap_\circ (b \times_\bullet d)$
by *auto*

lemma *fproduct-vdiff*: $a \times_\bullet (b -_\circ c) = (a \times_\bullet b) -_\circ (a \times_\bullet c)$ **by** *auto*

lemma *vfinite-ftimesI*[*intro*!]:
assumes *vfinite a* **and** *vfinite b*
shows *vfinite* $(a \times_\bullet b)$
using *assms*(1,2)
proof(*induction arbitrary*: *b rule*: *vfinite-induct*)
case (*vinsert x a'*)
from *vinsert*(4) **have** *vfinite* $(set\ \{x\} \times_\bullet b)$

proof(*induction rule*: *vfinite-induct*)
case (*vinsert y b′*)
have *set* {*x*} ×• *vinsert y b′* = *vinsert* [*x*, *y*]∘ (*set* {*x*} ×• *b′*) **by** *auto*
with *vinsert*(3) **show** *?case* **by** *simp*
qed *simp*
moreover have *vinsert x a′* ×• *b* = (*set* {*x*} ×• *b*) ∪∘ (*a′* ×• *b*) **by** *auto*
ultimately show *?case* **using** *vinsert* **by** (*auto simp*: *vfinite-vunionI*)
qed *simp*

ftimes and *vcpower*

lemma *vproduct-vpair*: [*a*, *b*]∘ ∈∘ (∏∘*i*∈∘2ℕ. *f i*) ⟷ ⟨*a*, *b*⟩ ∈∘ *f* (0ℕ) ×∘ *f* (1ℕ)
proof
interpret *vfsequence* ‹[*a*, *b*]∘› **by** *simp*
show [*a*, *b*]∘ ∈∘ (∏∘*i*∈∘2ℕ. *f i*) ⟹ ⟨*a*, *b*⟩ ∈∘ *f* (0ℕ) ×∘ *f* (1ℕ)
unfolding *vcons-vdoubleton two* **by** (*elim vproduct-vdoubletonE*) *auto*
assume *hyp*: ⟨*a*, *b*⟩ ∈∘ *f* (0ℕ) ×∘ *f* (1ℕ)
then have *af*: *a* ∈∘ *f* (0ℕ) **and** *bf*: *b* ∈∘ *f* (1ℕ) **by** *auto*
have *dom*: 𝒟∘ [*a*, *b*]∘ = *set* {0ℕ, 1ℕ} **by** (*auto intro*!: *vsubset-antisym*)
have *ran*: ℛ∘ [*a*, *b*]∘ ⊆∘ (⋃∘*i*∈∘2ℕ. *f i*)
unfolding *two* **using** *af bf vifunion-vdoubleton* **by** *auto*
show [*a*, *b*]∘ ∈∘ (∏∘*i*∈∘2ℕ. *f i*)
apply(*intro vproductI*)
subgoal using *dom ran vsv-axioms* **unfolding** *two* **by** *auto*
subgoal using *af bf* **unfolding** *two* **by** (*auto intro*!: *vsubset-antisym*)
subgoal
unfolding *two*
using *hyp VSigmaE*2 *small-empty vcons-vdoubleton*
by (*auto simp*: *vinsert-set-insert-eq*)
done
qed

Connections.

lemma *vcpower-two-ftimes*: *A* ^× 2ℕ = *A* ×• *A*
unfolding *vcpower-def ftimes-def two* **by** *simp*

lemma *vcpower-two-ftimesI*[*intro*]:
assumes *x* ∈∘ *A* ×• *A*
shows *x* ∈∘ *A* ^× 2ℕ
using *assms* **unfolding** *ftimes-def two* **by** *auto*

lemma *vcpower-two-ftimesD*[*dest*]:
assumes *x* ∈∘ *A* ^× 2ℕ
shows *x* ∈∘ *A* ×• *A*
using *assms* **unfolding** *vcpower-def ftimes-def two* **by** *simp*

lemma *vcpower-two-ftimesE*[*elim*]:
assumes *x* ∈∘ *A* ^× 2ℕ **and** *x* ∈∘ *A* ×• *A* ⟹ *P*
shows *P*
using *assms* **unfolding** *vcpower-def ftimes-def two* **by** *simp*

lemma *vfsequence-vcpower-two-vpair*: [*a*, *b*]∘ ∈∘ *A* ^× 2ℕ ⟷ ⟨*a*, *b*⟩ ∈∘ *A* ×∘ *A*
proof(*rule iffI*)
show [*a*, *b*]∘ ∈∘ *A* ^× 2ℕ ⟹ ⟨*a*, *b*⟩ ∈∘ *A* ×∘ *A*
by (*elim vcpowerE*, *unfold vproduct-vpair*)
qed (*intro vcpowerI*, *unfold vproduct-vpair*)

lemma *vsv-vfsequence-two*:
assumes *vsv gf* **and** 𝒟∘ *gf* = 2ℕ

```
  shows [vpfst gf, vpsnd gf]∘ = gf
proof–
  interpret gf: vsv gf by (auto intro: assms(1))
  show ?thesis
    by
      (
        rule sym,
        rule vsv-eqI,
        blast,
        blast,
        simp add: assms(2) nat-omega-simps,
        unfold assms(2),
        elim-in-numeral,
        all‹simp add: nat-omega-simps›
      )
qed

lemma vsv-vfsequence-three:
  assumes vsv hgf and 𝒟∘ hgf = 3ℕ
  shows [vpfst hgf, vpsnd hgf, vpthrd hgf]∘ = hgf
proof–
  interpret hgf: vsv hgf by (auto intro: assms(1))
  show ?thesis
    by
      (
        rule sym,
        rule vsv-eqI,
        blast,
        blast,
        simp add: assms(2) nat-omega-simps,
        unfold assms(2),
        elim-in-numeral,
        all‹simp add: nat-omega-simps›
      )
qed
```

2.9.7 Sequence as an element of a Cartesian power of a set

```
lemma vcons-in-vcpowerI[intro!]:
  assumes n ∈∘ ω
    and xs ∈∘ A ^× vcard xs
    and x ∈∘ A
    and n = vcard (xs #∘ x)
  shows xs #∘ x ∈∘ A ^× n
proof–
  interpret vfsequence xs
    using assms
    by
      (
        meson
          vcons-vsubset
          vfinite-vcard-omega-iff
          vfinite-vsubset
          vfsequence-vcpower
      )
  show ?thesis
    by
      (
```

```
      metis
        assms(2,3,4)
        vcpower-vrange
        vfsequence-vcons
        vfsequence-vcons-vrange
        vfsequence.vfsequence-vrange-vcpower
        vsubset-vinsert-leftI
    )
qed

lemma vcons-in-vcpowerD[dest]:
  assumes xs #∘ x ∈∘ A ^× n and n ∈∘ ω
  shows xs ∈∘ A ^× vcard xs
    and x ∈∘ A
    and n = vcard (xs #∘ x)
proof–
  interpret vfsequence xs
    by
      (
      meson
        assms
        vfsequence.vfsequence-vremove-vcons-vfsequence
        vfsequence-vcpower
      )
  from assms vfsequence-vcard-vcons show n = vcard (xs #∘ x) by auto
  then show xs ∈∘ A ^× vcard xs
    by
      (
      metis
        assms(1)
        vcpower-vrange
        vfsequence-vcons-vrange
        vfsequence-vrange-vcpower
        vsubset-vinsert-leftD
      )
  show x ∈∘ A
    by
      (
      metis
        assms(1)
        vcpower-vrange
        vfsequence.vfsequence-vcons-vrange
        vfsequence-axioms
        vinsertI1
        vsubsetE
      )
qed

lemma vcons-in-vcpowerE1[elim!]:
  assumes xs #∘ x ∈∘ A ^× n and n ∈∘ ω
  obtains xs ∈∘ A ^× vcard xs and x ∈∘ A and n = vcard (xs #∘ x)
  using assms by blast

lemma vcons-in-vcpowerE2:
  assumes xs ∈∘ A ^× n and n ∈∘ ω and 0 ∈∘ n
  obtains x xs′ where xs = xs′ #∘ x
    and xs′ ∈∘ A ^× vcard xs′
    and x ∈∘ A
```

```
    and n = vcard (xs′ #∘ x)
proof–
  interpret vfsequence xs using assms(1,2) by auto
  from assms obtain x xs′ where xs-def: xs = xs′ #∘ x
    by
      (
        metis
          eq0-iff vcard-0 vcpower-vdomain vfsequence-vcons-ex vfsequence-vdomain
      )
  from vcons-in-vcpowerE1[OF assms(1)[unfolded xs-def] assms(2)] have
    xs′ ∈∘ A ^× vcard xs′ and x ∈∘ A and n = vcard (xs′ #∘ x)
    by blast+
  from xs-def this show ?thesis by (clarsimp simp: that)
qed

lemma vcons-vcpower1E:
  assumes xs ∈∘ A ^× 1ℕ
  obtains x where xs = [x]∘ and x ∈∘ A
proof–
  have 01: 0 ∈∘ 1ℕ by simp
  from vcons-in-vcpowerE2[OF assms ord-of-nat-ω 01] obtain x xs′
    where xs-def: xs = xs′ #∘ x
      and xs′: xs′ ∈∘ A ^× vcard xs′
      and x: x ∈∘ A
      and one: 1ℕ = vcard (xs′ #∘ x)
    by metis
  interpret xs: vfsequence xs using assms by (auto intro: vfsequence-vcpower)
  interpret xs′: vfsequence xs′
    using xs′ xs-def xs.vfsequence-vremove-vcons-vfsequence by blast
  from one have vcard xs′ = 0
    by (metis ord-of-nat-succ-vempty succ-inject-iff xs′.vfsequence-vcard-vcons)
  then have xs = [x]∘ unfolding xs-def by (simp add: vcard-vempty)
  with x that show ?thesis by simp
qed
```

2.9.8 The set of all finite sequences on a set

Definition and elementary properties

```
definition vfsequences-on :: V ⇒ V
  where vfsequences-on X = set {x. vfsequence x ∧ (∀ i∈∘𝒟∘ x. x(|i|) ∈∘ X)}

lemma vfsequences-on-subset-ω-set:
  {x. vfsequence x ∧ (∀ i∈elts (𝒟∘ x). x(|i|) ∈∘ X)} ⊆ elts (VPow (ω ×∘ X))
proof
  (
    intro subsetI,
    unfold mem-Collect-eq VPow-iff,
    elim conjE,
    intro vsubsetI
  )
  fix xs nx
  assume prems[rule-format]:
    vfsequence xs
    ∀ i∈∘𝒟∘ xs. xs(|i|) ∈∘ X
    nx ∈∘ xs
  interpret vfsequence xs by (rule prems(1))
  from prems(3) vbrelation obtain n x where nx-def: nx = ⟨n, x⟩ by auto
```

from *vsv-appI*[*OF prems*(3)[*unfolded this*]] **have** *xsn*: $xs(\!|n|\!) = x$ **.**
from *prems*(3) *nx-def* **have** $n \in_\circ \mathcal{D}_\circ\ xs$ **by** *auto*
with *prems*(2) **show** $nx \in_\circ \omega \times_\circ X$
by (*auto simp*: *nx-def xsn*[*symmetric*] *Ord-trans vfsequence-vdomain-in-omega*)
qed

lemma *small-vfsequences-on*[*simp*]:
small $\{x.\ vfsequence\ x \wedge (\forall i \in_\circ \mathcal{D}_\circ\ x.\ x(\!|i|\!) \in_\circ X)\}$
by (*rule down*, *rule vfsequences-on-subset-ω-set*)

Rules.

lemma *vfsequences-onI*:
assumes *vfsequence xs* **and** $\bigwedge i.\ i \in_\circ \mathcal{D}_\circ\ xs \Longrightarrow xs(\!|i|\!) \in_\circ X$
shows $xs \in_\circ$ *vfsequences-on X*
using *assms* **unfolding** *vfsequences-on-def* **by** *simp*

lemma *vfsequences-onD*[*dest*]:
assumes $xs \in_\circ$ *vfsequences-on X*
shows *vfsequence xs* **and** $\bigwedge i.\ i \in_\circ \mathcal{D}_\circ\ xs \Longrightarrow xs(\!|i|\!) \in_\circ X$
using *assms* **unfolding** *vfsequences-on-def* **by** *auto*

lemma *vfsequences-onE*[*elim*]:
assumes $xs \in_\circ$ *vfsequences-on X*
obtains *vfsequence xs* **and** $\bigwedge i.\ i \in_\circ \mathcal{D}_\circ\ xs \Longrightarrow xs(\!|i|\!) \in_\circ X$
using *assms* **unfolding** *vfsequences-on-def* **by** *auto*

Further properties

lemma *vfsequences-on-vsubset-mono*:
assumes $A \subseteq_\circ B$
shows *vfsequences-on* $A \subseteq_\circ$ *vfsequences-on B*
proof(*intro vsubsetI vfsequences-onI*; *elim vfsequences-onE*)
fix *i xs* **assume** *prems*: $i \in_\circ \mathcal{D}_\circ\ xs$ $\bigwedge i.\ i \in_\circ \mathcal{D}_\circ\ xs \Longrightarrow xs(\!|i|\!) \in_\circ A$
from *assms prems*(2)[*OF prems*(1)] **show** $xs(\!|i|\!) \in_\circ B$ **by** *auto*
qed

2.10 Binary relation as a finite sequence

2.10.1 Background

This section exposes the theory of binary relations that are represented by a two element finite sequence $[a,\ b]_\circ$ (as opposed to a pair $\langle a,\ b\rangle$). Many results were adapted from the theory *CZH-Sets-BRelations*.

As previously, many of the results that are presented in this section can be assumed to have been adapted (with amendments) from the theory *Relation* in the main library.

lemma *fpair-iff*[*simp*]: $([a,\ b]_\circ = [a',\ b']_\circ) = (a = a' \wedge b = b')$ **by** *simp*

lemmas *fpair-inject*[*elim*!] = *fpair-iff*[*THEN iffD*1, *THEN conjE*]

2.10.2 *fpairs*

definition *fpairs* :: $V \Rightarrow V$ **where**
fpairs r = *set* $\{x.\ x \in_\circ r \wedge (\exists\, a\ b.\ x = [a,\ b]_\circ)\}$

lemma *small-fpairs*[*simp*]: *small* $\{x.\ x \in_\circ r \wedge (\exists\, a\ b.\ x = [a,\ b]_\circ)\}$
by (*rule down*[*of* - *r*]) *clarsimp*

Rules.

lemma *fpairsI*[*intro*]:
assumes $x \in_\circ r$ **and** $x = [a,\ b]_\circ$
shows $x \in_\circ$ *fpairs r*
using *assms* **unfolding** *fpairs-def* **by** *auto*

lemma *fpairsD*[*dest*]:
assumes $x \in_\circ$ *fpairs r*
shows $x \in_\circ r$ **and** $\exists\, a\ b.\ x = [a,\ b]_\circ$
using *assms* **unfolding** *fpairs-def* **by** *auto*

lemma *fpairsE*[*elim*]:
assumes $x \in_\circ$ *fpairs r*
obtains *a b* **where** $x = [a,\ b]_\circ$ **and** $[a,\ b]_\circ \in_\circ r$
using *assms* **unfolding** *fpairs-def* **by** *auto*

lemma *fpairs-iff*: $x \in_\circ$ *fpairs* $r \longleftrightarrow x \in_\circ r \wedge (\exists\, a\ b.\ x = [a,\ b]_\circ)$ **by** *auto*

Elementary properties.

lemma *fpairs-iff-elts*: $[a,\ b]_\circ \in_\circ$ *fpairs* $r \longleftrightarrow [a,\ b]_\circ \in_\circ r$ **by** *auto*

Set operations.

lemma *fpairs-vempty*[*simp*]: *fpairs* $0 = 0$ **by** *auto*

lemma *fpairs-vsingleton*[*simp*]: *fpairs* (*set* $\{[a,\ b]_\circ\}$) = *set* $\{[a,\ b]_\circ\}$ **by** *auto*

lemma *fpairs-vinsert*: *fpairs* (*vinsert* $[a,\ b]_\circ$ *A*) = *set* $\{[a,\ b]_\circ\} \cup_\circ$ *fpairs A*
by *auto*

lemma *fpairs-mono*:
assumes $r \subseteq_\circ s$
shows *fpairs r* $\subseteq_\circ$ *fpairs s*
using *assms* **by** *blast*

lemma *fpairs-vunion*: *fpairs* $(A \cup_\circ B)$ = *fpairs A* $\cup_\circ$ *fpairs B* **by** *auto*

lemma *fpairs-vintersection*: *fpairs* $(A \cap_\circ B)$ = *fpairs* $A \cap_\circ$ *fpairs* B **by** *auto*

lemma *fpairs-vdiff*: *fpairs* $(A -_\circ B)$ = *fpairs* $A -_\circ$ *fpairs* B **by** *auto*

Special properties.

lemma *fpairs-ex-vfst*:
assumes $x \in_\circ$ *fpairs* r
shows $\exists\, b.\ [x(\!|0_{\mathbb{N}}|\!),\ b]_\circ \in_\circ r$
proof–
from *assms* **have** *xr*: $x \in_\circ r$ **by** *auto*
moreover from *assms* **obtain** b **where** *x-def*: $x = [x(\!|0_{\mathbb{N}}|\!),\ b]_\circ$ **by** *auto*
ultimately have $[x(\!|0_{\mathbb{N}}|\!),\ b]_\circ \in_\circ r$ **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *fpairs-ex-vsnd*:
assumes $x \in_\circ$ *fpairs* r
shows $\exists\, a.\ [a,\ x(\!|1_{\mathbb{N}}|\!)]_\circ \in_\circ r$
proof–
from *assms* **have** *xr*: $x \in_\circ r$ **by** *auto*
moreover from *assms* **obtain** a **where** *x-def*: $x = [a,\ x(\!|1_{\mathbb{N}}|\!)]_\circ$
by (*auto simp*: *nat-omega-simps*)
ultimately have $[a,\ x(\!|1_{\mathbb{N}}|\!)]_\circ \in_\circ r$ **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *fpair-vcpower2I*[*intro*]:
assumes $a \in_\circ A \mathbin{\widehat{\ }}_\times 1_{\mathbb{N}}$ **and** $b \in_\circ A \mathbin{\widehat{\ }}_\times 1_{\mathbb{N}}$
shows *vconcat* $[a,\ b]_\circ \in_\circ A \mathbin{\widehat{\ }}_\times 2_{\mathbb{N}}$
proof–
from *assms* **obtain** a' b'
where *a-def*: $a = [a']_\circ$ **and** *b-def*: $b = [b']_\circ$ **and** $a' \in_\circ A$ **and** $b' \in_\circ A$
by (*force elim*: *vcons-vcpower1E*)
then show *?thesis* **by** (*auto simp*: *nat-omega-simps*)
qed

2.10.3 Constructors

Identity relation

definition *fid-on* :: $V \Rightarrow V$
where *fid-on* A = *set* $\{[a,\ a]_\circ \mid a.\ a \in_\circ A\}$

lemma *fid-on-small*[*simp*]: *small* $\{[a,\ a]_\circ \mid a.\ a \in_\circ A\}$
proof(*rule down*[*of* - $\langle A \mathbin{\widehat{\ }}_\times (2_{\mathbb{N}})\rangle$], *intro subsetI*)
fix x **assume** $x \in \{[a,\ a]_\circ \mid a.\ a \in_\circ A\}$
then obtain a **where** *x-def*: $x = [a,\ a]_\circ$ **and** $a \in_\circ A$ **by** *clarsimp*
interpret *vfsequence* $\langle [a,\ a]_\circ \rangle$ **by** *simp*
have *vcard-aa*: $2_{\mathbb{N}}$ = *vcard* $[a,\ a]_\circ$ **by** (*simp add*: *nat-omega-simps*)
from $\langle a \in_\circ A \rangle$ **show** $x \in_\circ A \mathbin{\widehat{\ }}_\times 2_{\mathbb{N}}$
unfolding *x-def vcard-aa* **by** (*intro vfsequence-vrange-vcpower*) *auto*
qed

Rules.

lemma *fid-on-eqI*:
assumes $a = b$ **and** $a \in_\circ A$
shows $[a,\ b]_\circ \in_\circ$ *fid-on* A

using *assms* **by** (*simp add*: *fid-on-def*)

lemma *fid-onI*[*intro*!]:
assumes $a \in_\circ A$
shows $[a, a]_\circ \in_\circ$ *fid-on* A
by (*rule fid-on-eqI*) (*simp-all add*: *assms*)

lemma *fid-onD*[*dest*!]:
assumes $[a, a]_\circ \in_\circ$ *fid-on* A
shows $a \in_\circ A$
using *assms* **unfolding** *fid-on-def* **by** *auto*

lemma *fid-onE*[*elim*!]:
assumes $x \in_\circ$ *fid-on* A **and** $\exists\, a{\in_\circ}A.\ x = [a, a]_\circ \Longrightarrow P$
shows P
using *assms* **unfolding** *fid-on-def* **by** *auto*

lemma *fid-on-iff*: $[a, b]_\circ \in_\circ$ *fid-on* $A \longleftrightarrow a = b \wedge a \in_\circ A$ **by** *auto*

Set operations.

lemma *fid-on-vempty*[*simp*]: *fid-on* $0 = 0$ **by** *auto*

lemma *fid-on-vsingleton*[*simp*]: *fid-on* (*set* $\{a\}$) = *set* $\{[a, a]_\circ\}$ **by** *auto*

lemma *fid-on-vdoubleton*: *fid-on* (*set* $\{a, b\}$) = *set* $\{[a, a]_\circ, [b, b]_\circ\}$ **by** *force*

lemma *fid-on-mono*:
assumes $A \subseteq_\circ B$
shows *fid-on* $A \subseteq_\circ$ *fid-on* B
using *assms* **by** *auto*

lemma *fid-on-vinsert*: *vinsert* $[a, a]_\circ$ (*fid-on* A) = *fid-on* (*vinsert* a A)
by *auto*

lemma *fid-on-vintersection*: *fid-on* $(A \cap_\circ B)$ = *fid-on* $A \cap_\circ$ *fid-on* B **by** *auto*

lemma *fid-on-vunion*: *fid-on* $(A \cup_\circ B)$ = *fid-on* $A \cup_\circ$ *fid-on* B **by** *auto*

lemma *fid-on-vdiff*: *fid-on* $(A -_\circ B)$ = *fid-on* $A -_\circ$ *fid-on* B **by** *auto*

Special properties.

lemma *fid-on-vsubset-vcpower*: *fid-on* $A \subseteq_\circ A\ \widehat{}_\times\ 2_{\mathbf{N}}$ **by** *force*

Constant function

definition *fconst-on* :: $V \Rightarrow V \Rightarrow V$
where *fconst-on* A c = *set* $\{[a, c]_\circ \mid a.\ a \in_\circ A\}$

lemma *small-fconst-on*[*simp*]: *small* $\{[a, c]_\circ \mid a.\ a \in_\circ A\}$
by (*rule down*[*of* - ‹$A \times_\bullet$ *set* $\{c\}$›]) *blast*

Rules.

lemma *fconst-onI*[*intro*!]:
assumes $a \in_\circ A$
shows $[a, c]_\circ \in_\circ$ *fconst-on* A c
using *assms* **unfolding** *fconst-on-def* **by** *simp*

lemma *fconst-onD*[*dest*!]:

assumes $[a,\ c]_\circ \in_\circ$ *fconst-on* $A\ c$
shows $a \in_\circ A$
using *assms* **unfolding** *fconst-on-def* **by** *simp*

lemma *fconst-onE*[*elim*!]:
assumes $x \in_\circ$ *fconst-on* $A\ c$
obtains a **where** $a \in_\circ A$ **and** $x = [a,\ c]_\circ$
using *assms* **unfolding** *fconst-on-def* **by** *auto*

lemma *fconst-on-iff*: $[a,\ c]_\circ \in_\circ$ *fconst-on* $A\ c \longleftrightarrow a \in_\circ A$ **by** *auto*

Set operations.

lemma *fconst-on-vempty*[*simp*]: *fconst-on* $0\ c = 0$
unfolding *fconst-on-def* **by** *auto*

lemma *fconst-on-vsingleton*[*simp*]: *fconst-on* $(set\ \{a\})\ c = set\ \{[a,\ c]_\circ\}$
by *auto*

lemma *fconst-on-vdoubleton*: *fconst-on* $(set\ \{a,\ b\})\ c = set\ \{[a,\ c]_\circ,\ [b,\ c]_\circ\}$
by *force*

lemma *fconst-on-mono*:
assumes $A \subseteq_\circ B$
shows *fconst-on* $A\ c \subseteq_\circ$ *fconst-on* $B\ c$
using *assms* **by** *auto*

lemma *fconst-on-vinsert*:
$(vinsert\ [a,\ c]_\circ\ (fconst\text{-}on\ A\ c)) = (fconst\text{-}on\ (vinsert\ a\ A)\ c)$
by *auto*

lemma *fconst-on-vintersection*:
fconst-on $(A \cap_\circ B)\ c =$ *fconst-on* $A\ c \cap_\circ$ *fconst-on* $B\ c$
by *auto*

lemma *fconst-on-vunion*: *fconst-on* $(A \cup_\circ B)\ c =$ *fconst-on* $A\ c \cup_\circ$ *fconst-on* $B\ c$
by *auto*

lemma *fconst-on-vdiff*: *fconst-on* $(A -_\circ B)\ c =$ *fconst-on* $A\ c -_\circ$ *fconst-on* $B\ c$
by *auto*

Special properties.

lemma *fconst-on-eq-ftimes*: *fconst-on* $A\ c = A \times_\bullet set\ \{c\}$ **by** *blast*

Composition

definition *fcomp* :: $V \Rightarrow V \Rightarrow V$ (**infixr** ‹$\circ_\bullet$› 75)
where $r \circ_\bullet s = set\ \{[a,\ c]_\circ \mid a\ c.\ \exists b.\ [a,\ b]_\circ \in_\circ s \wedge [b,\ c]_\circ \in_\circ r\}$
notation *fcomp* (**infixr** ‹$\circ_\bullet$› 75)

lemma *fcomp-small*[*simp*]: *small* $\{[a,\ c]_\circ \mid a\ c.\ \exists b.\ [a,\ b]_\circ \in_\circ s \wedge [b,\ c]_\circ \in_\circ r\}$
(**is** ‹*small* *?s*›)
proof–
define *comp′* **where** $comp' = (\lambda\langle ab,\ cd\rangle.\ [ab(\!|0_{\mathbb{N}}|\!),\ cd(\!|1_{\mathbb{N}}|\!)]_\circ)$
have *small* $(elts\ (vpairs\ (s \times_\circ r)))$ **by** *simp*
then have *small-comp*: *small* $(comp'\ `\ elts\ (vpairs\ (s \times_\circ r)))$ **by** *simp*
have *ss*: $?s \subseteq (comp'\ `\ elts\ (vpairs\ (s \times_\circ r)))$
proof
fix x **assume** $x \in ?s$

```
    then obtain a b c where x-def: x = [a, c]∘
      and [a, b]∘ ∈∘ s
      and [b, c]∘ ∈∘ r
      by auto
    then have abbc: ⟨[a, b]∘, [b, c]∘⟩ ∈∘ vpairs (s ×∘ r)
      by (simp add: vpairs-iff-elts)
    have x-def': x = comp' ⟨[a, b]∘, [b, c]∘⟩
      unfolding comp'-def x-def by (auto simp: nat-omega-simps)
    then show x ∈ comp' ‘ elts (vpairs (s ×∘ r))
      unfolding x-def' using abbc by auto
  qed
  with small-comp show ?thesis by (meson smaller-than-small)
qed
```

Rules.

```
lemma fcompI[intro]:
  assumes [b, c]∘ ∈∘ r and [a, b]∘ ∈∘ s
  shows [a, c]∘ ∈∘ r ∘• s
  using assms unfolding fcomp-def by auto

lemma fcompD[dest]:
  assumes [a, c]∘ ∈∘ r ∘• s
  shows ∃ b. [b, c]∘ ∈∘ r ∧ [a, b]∘ ∈∘ s
  using assms unfolding fcomp-def by auto

lemma fcompE[elim]:
  assumes ac ∈∘ r ∘• s
  obtains a b c where ac = [a, c]∘ and [a, b]∘ ∈∘ s and [b, c]∘ ∈∘ r
  using assms unfolding fcomp-def by clarsimp
```

Elementary properties.

```
lemma fcomp-assoc: (r ∘• s) ∘• t = r ∘• (s ∘• t) by fast
```

Set operations.

```
lemma fcomp-vempty-left[simp]: 0 ∘• r = 0 unfolding vcomp-def by force

lemma fcomp-vempty-right[simp]: r ∘• 0 = 0 unfolding vcomp-def by force

lemma fcomp-mono:
  assumes r' ⊆∘ r and s' ⊆∘ s
  shows r' ∘• s' ⊆∘ r ∘• s
  using assms by force

lemma fcomp-vinsert-left[simp]:
  vinsert ([a, b]∘) s ∘• r = (set {[a, b]∘} ∘• r) ∪∘ (s ∘• r)
  by auto

lemma fcomp-vinsert-right[simp]:
  r ∘• vinsert [a, b]∘ s = (r ∘• set {[a, b]∘}) ∪∘ (r ∘• s)
  by auto

lemma fcomp-vunion-left[simp]: (s ∪∘ t) ∘• r = (s ∘• r) ∪∘ (t ∘• r) by auto

lemma fcomp-vunion-right[simp]: r ∘• (s ∪∘ t) = (r ∘• s) ∪∘ (r ∘• t) by auto
```

Connections.

```
lemma fcomp-fid-on-idem[simp]: fid-on A ∘• fid-on A = fid-on A by force
```

lemma *fcomp-fid-on*[*simp*]: *fid-on A* $\circ_\bullet$ *fid-on B* = *fid-on* (*A* $\cap_\circ$ *B*) **by** *force*

lemma *fcomp-fconst-on-fid-on*[*simp*]: *fconst-on A c* $\circ_\bullet$ *fid-on A* = *fconst-on A c*
by *auto*

Special properties.

lemma *fcomp-vsubset-vtimes*:
assumes *r* $\subseteq_\circ$ *B* $\times_\bullet$ *C* **and** *s* $\subseteq_\circ$ *A* $\times_\bullet$ *B*
shows *r* $\circ_\bullet$ *s* $\subseteq_\circ$ *A* $\times_\bullet$ *C*
using *assms* **by** *blast*

lemma *fcomp-obtain-middle*[*elim*]:
assumes [*a*, *c*]$_\circ$ $\in_\circ$ *f* $\circ_\bullet$ *g*
obtains *b* **where** [*a*, *b*]$_\circ$ $\in_\circ$ *g* **and** [*b*, *c*]$_\circ$ $\in_\circ$ *f*
using *assms* **by** *auto*

Converse relation

definition *fconverse* :: *V* $\Rightarrow$ *V* (‹(-$^{-1}{}_\bullet$)› [1000] 999)
where $r^{-1}{}_\bullet$ = *set* {[*b*, *a*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ *r*}

lemma *fconverse-small*[*simp*]: *small* {[*b*, *a*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ *r*}
proof–
have *eq*:
{[*b*, *a*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ *r*} = (λ*x*. [*x*(|1$_\mathbb{N}$|), *x*(|0$_\mathbb{N}$|)]$_\circ$) ‘ *elts* (*fpairs r*)
proof(*rule subset-antisym*; *rule subsetI*, *unfold mem-Collect-eq*)
fix *x* **assume** *x* ∈ (λ*x*. [*x*(|1$_\mathbb{N}$|), *x*(|0$_\mathbb{N}$|)]$_\circ$) ‘ *elts* (*fpairs r*)
then obtain *a b* **where** [*a*, *b*]$_\circ$ $\in_\circ$ *fpairs r*
and *x* = (λ*x*. [*x*(|1$_\mathbb{N}$|), *x*(|0$_\mathbb{N}$|)]$_\circ$) [*a*, *b*]$_\circ$
by *blast*
then show ∃ *a b*. *x* = [*b*, *a*]$_\circ$ ∧ [*a*, *b*]$_\circ$ $\in_\circ$ *r* **by** (*auto simp*: *nat-omega-simps*)
qed (*use image-iff fpairs-iff-elts* **in** ‹*fastforce simp*: *nat-omega-simps*›)
show *?thesis* **unfolding** *eq* **by** (*rule replacement*) *auto*
qed

Rules.

lemma *fconverseI*[*sym*, *intro*!]:
assumes [*a*, *b*]$_\circ$ $\in_\circ$ *r*
shows [*b*, *a*]$_\circ$ $\in_\circ$ $r^{-1}{}_\bullet$
using *assms* **unfolding** *fconverse-def* **by** *simp*

lemma *fconverseD*[*sym*, *dest*]:
assumes [*a*, *b*]$_\circ$ $\in_\circ$ $r^{-1}{}_\bullet$
shows [*b*, *a*]$_\circ$ $\in_\circ$ *r*
using *assms* **unfolding** *fconverse-def* **by** *simp*

lemma *fconverseE*[*elim*!]:
assumes *x* $\in_\circ$ $r^{-1}{}_\bullet$
obtains *a b* **where** *x* = [*b*, *a*]$_\circ$ **and** [*a*, *b*]$_\circ$ $\in_\circ$ *r*
using *assms* **unfolding** *fconverse-def* **by** *auto*

lemma *fconverse-iff*: [*b*, *a*]$_\circ$ $\in_\circ$ $r^{-1}{}_\bullet$ $\longleftrightarrow$ [*a*, *b*]$_\circ$ $\in_\circ$ *r* **by** *auto*

Set operations.

lemma *fconverse-vempty*[*simp*]: $0^{-1}{}_\bullet$ = 0 **by** *auto*

lemma *fconverse-vsingleton*: (*set* {[*a*, *b*]$_\circ$})$^{-1}{}_\bullet$ = *set* {[*b*, *a*]$_\circ$} **by** *auto*

lemma *fconverse-vdoubleton*: $(set\ \{[a,\ b]_\circ,\ [c,\ d]_\circ\})^{-1}_\bullet = set\ \{[b,\ a]_\circ,\ [d,\ c]_\circ\}$
by *force*

lemma *fconverse-vinsert*: $(vinsert\ [a,\ b]_\circ\ r)^{-1}_\bullet = vinsert\ [b,\ a]_\circ\ (r^{-1}_\bullet)$ **by** *auto*

lemma *fconverse-vintersection*: $(r \cap_\circ s)^{-1}_\bullet = r^{-1}_\bullet \cap_\circ s^{-1}_\bullet$ **by** *auto*

lemma *fconverse-vunion*: $(r \cup_\circ s)^{-1}_\bullet = r^{-1}_\bullet \cup_\circ s^{-1}_\bullet$ **by** *auto*

Connections.

lemma *fconverse-fid-on*[*simp*]: $(fid\text{-}on\ A)^{-1}_\bullet = fid\text{-}on\ A$ **by** *auto*

lemma *fconverse-fconst-on*[*simp*]: $(fconst\text{-}on\ A\ c)^{-1}_\bullet = set\ \{c\} \times_\bullet A$ **by** *blast*

lemma *fconverse-fcomp*: $(r \circ_\bullet s)^{-1}_\bullet = s^{-1}_\bullet \circ_\bullet r^{-1}_\bullet$ **by** *auto*

lemma *fconverse-ftimes*: $(A \times_\bullet B)^{-1}_\bullet = (B \times_\bullet A)$ **by** *auto*

Special properties.

lemma *fconverse-pred*:
assumes $small\ \{[a,\ b]_\circ \mid a\ b.\ P\ a\ b\}$
shows $(set\ \{[a,\ b]_\circ \mid a\ b.\ P\ a\ b\})^{-1}_\bullet = set\ \{[b,\ a]_\circ \mid a\ b.\ P\ a\ b\}$
using *assms* **unfolding** *fconverse-def* **by** *simp*

Left restriction

definition *flrestriction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\restriction^l_\bullet\rangle$ 80)
where $r \restriction^l_\bullet A = set\ \{[a,\ b]_\circ \mid a\ b.\ a \in_\circ A \wedge [a,\ b]_\circ \in_\circ r\}$

lemma *flrestriction-small*[*simp*]: $small\ \{[a,\ b]_\circ \mid a\ b.\ a \in_\circ A \wedge [a,\ b]_\circ \in_\circ r\}$
by (*rule down*[*of* - *r*]) *auto*

Rules.

lemma *flrestrictionI*[*intro*!]:
assumes $a \in_\circ A$ **and** $[a,\ b]_\circ \in_\circ r$
shows $[a,\ b]_\circ \in_\circ r \restriction^l_\bullet A$
using *assms* **unfolding** *flrestriction-def* **by** *simp*

lemma *flrestrictionD*[*dest*]:
assumes $[a,\ b]_\circ \in_\circ r \restriction^l_\bullet A$
shows $a \in_\circ A$ **and** $[a,\ b]_\circ \in_\circ r$
using *assms* **unfolding** *flrestriction-def* **by** *auto*

lemma *flrestrictionE*[*elim*!]:
assumes $x \in_\circ r \restriction^l_\bullet A$
obtains *a b* **where** $x = [a,\ b]_\circ$ **and** $a \in_\circ A$ **and** $[a,\ b]_\circ \in_\circ r$
using *assms* **unfolding** *flrestriction-def* **by** *auto*

Set operations.

lemma *flrestriction-on-vempty*[*simp*]: $r \restriction^l_\bullet 0 = 0$ **by** *auto*

lemma *flrestriction-vempty*[*simp*]: $0 \restriction^l_\bullet A = 0$ **by** *auto*

lemma *flrestriction-vsingleton-in*[*simp*]:
assumes $a \in_\circ A$
shows $set\ \{[a,\ b]_\circ\} \restriction^l_\bullet A = set\ \{[a,\ b]_\circ\}$

using *assms* **by** *auto*

lemma *flrestriction-vsingleton-nin*[*simp*]:
assumes $a \notin_\circ A$
shows *set* $\{[a, b]_\circ\} \restriction^l_\bullet A = 0$
using *assms* **by** *auto*

lemma *flrestriction-mono*:
assumes $A \subseteq_\circ B$
shows $r \restriction^l_\bullet A \subseteq_\circ r \restriction^l_\bullet B$
using *assms* **by** *auto*

lemma *flrestriction-vinsert-nin*[*simp*]:
assumes $a \notin_\circ A$
shows (*vinsert* $[a, b]_\circ$ r) $\restriction^l_\bullet A = r \restriction^l_\bullet A$
using *assms* **by** *auto*

lemma *flrestriction-vinsert-in*:
assumes $a \in_\circ A$
shows (*vinsert* $[a, b]_\circ$ r) $\restriction^l_\bullet A =$ *vinsert* $[a, b]_\circ$ $(r \restriction^l_\bullet A)$
using *assms* **by** *auto*

lemma *flrestriction-vintersection*: $(r \cap_\circ s) \restriction^l_\bullet A = r \restriction^l_\bullet A \cap_\circ s \restriction^l_\bullet A$ **by** *auto*

lemma *flrestriction-vunion*: $(r \cup_\circ s) \restriction^l_\bullet A = r \restriction^l_\bullet A \cup_\circ s \restriction^l_\bullet A$ **by** *auto*

lemma *flrestriction-vdiff*: $(r -_\circ s) \restriction^l_\bullet A = r \restriction^l_\bullet A -_\circ s \restriction^l_\bullet A$ **by** *auto*

Connections.

lemma *flrestriction-fid-on*[*simp*]: (*fid-on* A) $\restriction^l_\bullet B$ = *fid-on* $(A \cap_\circ B)$ **by** *auto*

lemma *flrestriction-fconst-on*: (*fconst-on* A c) $\restriction^l_\bullet B$ = (*fconst-on* B c) $\restriction^l_\bullet A$
by *auto*

lemma *flrestriction-fconst-on-commute*:
assumes $x \in_\circ$ *fconst-on* A $c \restriction^l_\bullet B$
shows $x \in_\circ$ *fconst-on* B $c \restriction^l_\bullet A$
using *assms* **by** *auto*

lemma *flrestriction-fcomp*[*simp*]: $(r \circ_\bullet s) \restriction^l_\bullet A = r \circ_\bullet (s \restriction^l_\bullet A)$ **by** *auto*

Previous connections.

lemma *fcomp-rel-fid-on*[*simp*]: $r \circ_\bullet$ *fid-on* $A = r \restriction^l_\bullet A$ **by** *auto*

lemma *fcomp-fconst-on*:
$r \circ_\bullet$ (*fconst-on* A c) = $(r \restriction^l_\bullet$ *set* $\{c\}) \circ_\bullet$ (*fconst-on* A c)
by *auto*

Special properties.

lemma *flrestriction-vsubset-fpairs*: $r \restriction^l_\bullet A \subseteq_\circ$ *fpairs* r
by (*rule vsubsetI*) (*metis fpairs-iff-elts flrestrictionE*)

lemma *flrestriction-vsubset-frel*: $r \restriction^l_\bullet A \subseteq_\circ r$ **by** *auto*

Right restriction

definition *frrestriction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\restriction^r_\bullet\rangle$ 80)
where $r \restriction^r_\bullet A$ = *set* $\{[a, b]_\circ \mid a\ b.\ b \in_\circ A \wedge [a, b]_\circ \in_\circ r\}$

lemma *frrestriction-small*[*simp*]: *small* $\{[a,\ b]_\circ \mid a\ b.\ b \in_\circ A \wedge [a,\ b]_\circ \in_\circ r\}$
by (*rule down*[*of* - *r*]) *auto*

Rules.

lemma *frrestrictionI*[*intro*!]:
assumes $b \in_\circ A$ **and** $[a,\ b]_\circ \in_\circ r$
shows $[a,\ b]_\circ \in_\circ r \upharpoonright^r_\bullet A$
using *assms* **unfolding** *frrestriction-def* **by** *simp*

lemma *frrestrictionD*[*dest*]:
assumes $[a,\ b]_\circ \in_\circ r \upharpoonright^r_\bullet A$
shows $b \in_\circ A$ **and** $[a,\ b]_\circ \in_\circ r$
using *assms* **unfolding** *frrestriction-def* **by** *auto*

lemma *frrestrictionE*[*elim*!]:
assumes $x \in_\circ r \upharpoonright^r_\bullet A$
obtains *a b* **where** $x = [a,\ b]_\circ$ **and** $b \in_\circ A$ **and** $[a,\ b]_\circ \in_\circ r$
using *assms* **unfolding** *frrestriction-def* **by** *auto*

Set operations.

lemma *frrestriction-on-vempty*[*simp*]: $r \upharpoonright^r_\bullet 0 = 0$ **by** *auto*

lemma *frrestriction-vempty*[*simp*]: $0 \upharpoonright^r_\bullet A = 0$ **by** *auto*

lemma *frrestriction-vsingleton-in*[*simp*]:
assumes $b \in_\circ A$
shows *set* $\{[a,\ b]_\circ\} \upharpoonright^r_\bullet A =$ *set* $\{[a,\ b]_\circ\}$
using *assms* **by** *auto*

lemma *frrestriction-vsingleton-nin*[*simp*]:
assumes $b \notin_\circ A$
shows *set* $\{[a,\ b]_\circ\} \upharpoonright^r_\bullet A = 0$
using *assms* **by** *auto*

lemma *frrestriction-mono*:
assumes $A \subseteq_\circ B$
shows $r \upharpoonright^r_\bullet A \subseteq_\circ r \upharpoonright^r_\bullet B$
using *assms* **by** *auto*

lemma *frrestriction-vinsert-nin*[*simp*]:
assumes $b \notin_\circ A$
shows (*vinsert* $[a,\ b]_\circ\ r) \upharpoonright^r_\bullet A = r \upharpoonright^r_\bullet A$
using *assms* **by** *auto*

lemma *frrestriction-vinsert-in*:
assumes $b \in_\circ A$
shows (*vinsert* $[a,\ b]_\circ\ r) \upharpoonright^r_\bullet A =$ *vinsert* $[a,\ b]_\circ\ (r \upharpoonright^r_\bullet A)$
using *assms* **by** *auto*

lemma *frrestriction-vintersection*: $(r \cap_\circ s) \upharpoonright^r_\bullet A = r \upharpoonright^r_\bullet A \cap_\circ s \upharpoonright^r_\bullet A$ **by** *auto*

lemma *frrestriction-vunion*: $(r \cup_\circ s) \upharpoonright^r_\bullet A = r \upharpoonright^r_\bullet A \cup_\circ s \upharpoonright^r_\bullet A$ **by** *auto*

lemma *frrestriction-vdiff*: $(r -_\circ s) \upharpoonright^r_\bullet A = r \upharpoonright^r_\bullet A -_\circ s \upharpoonright^r_\bullet A$ **by** *auto*

Connections.

lemma *frrestriction-fid-on*[*simp*]: (*fid-on A*) $\upharpoonright^r_\bullet B =$ *fid-on* $(A \cap_\circ B)$ **by** *auto*

lemma *frrestriction-fconst-on*:
assumes $c \in_\circ B$
shows $(\mathit{fconst\text{-}on}\ A\ c) \restriction^r_\bullet B = \mathit{fconst\text{-}on}\ A\ c$
using *assms* **by** *auto*

lemma *frrestriction-fcomp*[*simp*]: $(r \circ_\bullet s) \restriction^r_\bullet A = (r \restriction^r_\bullet A) \circ_\bullet s$ **by** *auto*

Previous connections.

lemma *fcomp-fid-on-rel*[*simp*]: $\mathit{fid\text{-}on}\ A \circ_\bullet r = r \restriction^r_\bullet A$ **by** *force*

lemma *fcomp-fconst-on-rel*: $(\mathit{fconst\text{-}on}\ A\ c) \circ_\bullet r = (\mathit{fconst\text{-}on}\ A\ c) \circ_\bullet (r \restriction^r_\bullet A)$
by *auto*

lemma *ftrestriction-fconverse*: $r^{-1}_\bullet \restriction^l_\bullet A = (r \restriction^r_\bullet A)^{-1}_\bullet$ **by** *auto*

lemma *frrestriction-fconverse*: $r^{-1}_\bullet \restriction^r_\bullet A = (r \restriction^l_\bullet A)^{-1}_\bullet$ **by** *auto*

Special properties.

lemma *frrestriction-vsubset-rel*: $r \restriction^r_\bullet A \subseteq_\circ r$ **by** *auto*

lemma *frrestriction-vsubset-vpairs*: $r \restriction^r_\bullet A \subseteq_\circ \mathit{fpairs}\ r$ **by** *auto*

Restriction

definition *frestriction* :: $V \Rightarrow V \Rightarrow V$ (**infixr** $\langle\restriction_\bullet\rangle$ 80)
where $r \restriction_\bullet A = \mathit{set}\ \{[a, b]_\circ \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge [a, b]_\circ \in_\circ r\}$

lemma *frestriction-small*[*simp*]:
$\mathit{small}\ \{[a, b]_\circ \mid a\ b.\ a \in_\circ A \wedge b \in_\circ A \wedge [a, b]_\circ \in_\circ r\}$
by (*rule down*[*of* - *r*]) *auto*

Rules.

lemma *frestrictionI*[*intro*!]:
assumes $a \in_\circ A$ **and** $b \in_\circ A$ **and** $[a, b]_\circ \in_\circ r$
shows $[a, b]_\circ \in_\circ r \restriction_\bullet A$
using *assms* **unfolding** *frestriction-def* **by** *simp*

lemma *frestrictionD*[*dest*]:
assumes $[a, b]_\circ \in_\circ r \restriction_\bullet A$
shows $a \in_\circ A$ **and** $b \in_\circ A$ **and** $[a, b]_\circ \in_\circ r$
using *assms* **unfolding** *frestriction-def* **by** *auto*

lemma *frestrictionE*[*elim*!]:
assumes $x \in_\circ r \restriction_\bullet A$
obtains $a\ b$ **where** $x = [a, b]_\circ$ **and** $a \in_\circ A$ **and** $b \in_\circ A$ **and** $[a, b]_\circ \in_\circ r$
using *assms* **unfolding** *frestriction-def* **by** *clarsimp*

Set operations.

lemma *frestriction-on-vempty*[*simp*]: $r \restriction_\bullet 0 = 0$ **by** *auto*

lemma *frestriction-vempty*[*simp*]: $0 \restriction_\bullet A = 0$ **by** *auto*

lemma *frestriction-vsingleton-in*[*simp*]:
assumes $a \in_\circ A$ **and** $b \in_\circ A$
shows $\mathit{set}\ \{[a, b]_\circ\} \restriction_\bullet A = \mathit{set}\ \{[a, b]_\circ\}$
using *assms* **by** *auto*

lemma *frestriction-vsingleton-nin-left*[*simp*]:
assumes $a \notin_\circ A$
shows *set* $\{[a, b]_\circ\} \upharpoonright_\bullet A = 0$
using *assms* **by** *auto*

lemma *frestriction-vsingleton-nin-right*[*simp*]:
assumes $b \notin_\circ A$
shows *set* $\{[a, b]_\circ\} \upharpoonright_\bullet A = 0$
using *assms* **by** *auto*

lemma *frestriction-mono*:
assumes $A \subseteq_\circ B$
shows $r \upharpoonright_\bullet A \subseteq_\circ r \upharpoonright_\bullet B$
using *assms* **by** *auto*

lemma *frestriction-vinsert-nin*[*simp*]:
assumes $a \notin_\circ A$ **and** $b \notin_\circ A$
shows $(\mathit{vinsert}\ [a, b]_\circ\ r) \upharpoonright_\bullet A = r \upharpoonright_\bullet A$
using *assms* **by** *auto*

lemma *frestriction-vinsert-in*:
assumes $a \in_\circ A$ **and** $b \in_\circ A$
shows $(\mathit{vinsert}\ [a, b]_\circ\ r) \upharpoonright_\bullet A = \mathit{vinsert}\ [a, b]_\circ\ (r \upharpoonright_\bullet A)$
using *assms* **by** *auto*

lemma *frestriction-vintersection*: $(r \cap_\circ s) \upharpoonright_\bullet A = r \upharpoonright_\bullet A \cap_\circ s \upharpoonright_\bullet A$ **by** *auto*

lemma *frestriction-vunion*: $(r \cup_\circ s) \upharpoonright_\bullet A = r \upharpoonright_\bullet A \cup_\circ s \upharpoonright_\bullet A$ **by** *auto*

lemma *frestriction-vdiff*: $(r -_\circ s) \upharpoonright_\bullet A = r \upharpoonright_\bullet A -_\circ s \upharpoonright_\bullet A$ **by** *auto*

Connections.

lemma *fid-on-frestriction*[*simp*]: $(\mathit{fid\text{-}on}\ A) \upharpoonright_\bullet B = \mathit{fid\text{-}on}\ (A \cap_\circ B)$ **by** *auto*

lemma *frestriction-fconst-on-ex*:
assumes $c \in_\circ B$
shows $(\mathit{fconst\text{-}on}\ A\ c) \upharpoonright_\bullet B = \mathit{fconst\text{-}on}\ (A \cap_\circ B)\ c$
using *assms* **by** *auto*

lemma *frestriction-fconst-on-nex*:
assumes $c \notin_\circ B$
shows $(\mathit{fconst\text{-}on}\ A\ c) \upharpoonright_\bullet B = 0$
using *assms* **by** *auto*

lemma *frestriction-fcomp*[*simp*]: $(r \circ_\bullet s) \upharpoonright_\bullet A = (r \upharpoonright^r_\bullet A) \circ_\bullet (s \upharpoonright^l_\bullet A)$ **by** *auto*

lemma *frestriction-fconverse*: $r^{-1}_\bullet \upharpoonright_\bullet A = (r \upharpoonright_\bullet A)^{-1}_\bullet$ **by** *auto*

Previous connections.

lemma *frrestriction-flrestriction*[*simp*]: $(r \upharpoonright^r_\bullet A) \upharpoonright^l_\bullet A = r \upharpoonright_\bullet A$ **by** *auto*

lemma *flrestriction-frrestriction*[*simp*]: $(r \upharpoonright^l_\bullet A) \upharpoonright^r_\bullet A = r \upharpoonright_\bullet A$ **by** *auto*

lemma *frestriction-flrestriction*[*simp*]: $(r \upharpoonright_\bullet A) \upharpoonright^l_\bullet A = r \upharpoonright_\bullet A$ **by** *auto*

lemma *frestriction-frrestriction*[*simp*]: $(r \upharpoonright_\bullet A) \upharpoonright^r_\bullet A = r \upharpoonright_\bullet A$ **by** *auto*

Special properties.

lemma *frestriction-vsubset-fpairs*: $r \upharpoonright_\bullet A \subseteq_\circ \mathit{fpairs}\ r$ **by** *auto*

lemma *frestriction-vsubset-ftimes*: $r \upharpoonright_\bullet A \subseteq_\circ A \mathbin{\hat{\times}} 2_\mathbb{N}$ **by** *force*

lemma *frestriction-vsubset-rel*: $r \upharpoonright_\bullet A \subseteq_\circ r$ **by** *auto*

2.10.4 Properties

Domain

definition *fdomain* :: $V \Rightarrow V$ $(\langle \mathcal{D}_\bullet \rangle)$
  where $\mathcal{D}_\bullet\ r = \mathit{set}\ \{a.\ \exists\, b.\ [a,\ b]_\circ \in_\circ r\}$
notation *fdomain* $(\langle \mathcal{D}_\bullet \rangle)$

lemma *fdomain-small*[*simp*]: *small* $\{a.\ \exists\, b.\ [a,\ b]_\circ \in_\circ r\}$
proof–
  have *ss*: $\{a.\ \exists\, b.\ [a,\ b]_\circ \in_\circ r\} \subseteq (\lambda x.\ x(\!|0_\mathbb{N}|\!))\ `\ \mathit{elts}\ r$
    using *image-iff* **by** *force*
  have *small*: *small* $((\lambda x.\ x(\!|0_\mathbb{N}|\!))\ `\ \mathit{elts}\ r)$ **by** (*rule replacement*) *simp*
  show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

Rules.

lemma *fdomainI*[*intro*]:
  assumes $[a,\ b]_\circ \in_\circ r$
  shows $a \in_\circ \mathcal{D}_\bullet\ r$
  using *assms* **unfolding** *fdomain-def* **by** *auto*

lemma *fdomainD*[*dest*]:
  assumes $a \in_\circ \mathcal{D}_\bullet\ r$
  shows $\exists\, b.\ [a,\ b]_\circ \in_\circ r$
  using *assms* **unfolding** *fdomain-def* **by** *auto*

lemma *fdomainE*[*elim*]:
  assumes $a \in_\circ \mathcal{D}_\bullet\ r$
  obtains *b* **where** $[a,\ b]_\circ \in_\circ r$
  using *assms* **unfolding** *fdomain-def* **by** *clarsimp*

lemma *fdomain-iff*: $a \in_\circ \mathcal{D}_\bullet\ r \longleftrightarrow (\exists\, y.\ [a,\ y]_\circ \in_\circ r)$ **by** *auto*

Set operations.

lemma *fdomain-vempty*[*simp*]: $\mathcal{D}_\bullet\ 0 = 0$ **by** *force*

lemma *fdomain-vsingleton*[*simp*]: $\mathcal{D}_\bullet\ (\mathit{set}\ \{[a,\ b]_\circ\}) = \mathit{set}\ \{a\}$ **by** *auto*

lemma *fdomain-vdoubleton*[*simp*]: $\mathcal{D}_\bullet\ (\mathit{set}\ \{[a,\ b]_\circ,\ [c,\ d]_\circ\}) = \mathit{set}\ \{a,\ c\}$
  by *force*

lemma *fdomain-mono*:
  assumes $r \subseteq_\circ s$
  shows $\mathcal{D}_\bullet\ r \subseteq_\circ \mathcal{D}_\bullet\ s$
  using *assms* **by** *blast*

lemma *fdomain-vinsert*[*simp*]: $\mathcal{D}_\bullet\ (\mathit{vinsert}\ [a,\ b]_\circ\ r) = \mathit{vinsert}\ a\ (\mathcal{D}_\bullet\ r)$
  by *force*

lemma *fdomain-vunion*: $\mathcal{D}_\bullet\ (A \cup_\circ B) = \mathcal{D}_\bullet\ A \cup_\circ \mathcal{D}_\bullet\ B$ **by** *force*

lemma *fdomain-vintersection-vsubset*: $\mathcal{D}_\bullet\ (A \cap_\circ B) \subseteq_\circ \mathcal{D}_\bullet\ A \cap_\circ \mathcal{D}_\bullet\ B$ **by** *auto*

lemma *fdomain-vdiff-vsubset*: $\mathcal{D}_\bullet\ A -_\circ \mathcal{D}_\bullet\ B \subseteq_\circ \mathcal{D}_\bullet\ (A -_\circ B)$ **by** *auto*

Connections.

lemma *fdomain-fid-on*[*simp*]: $\mathcal{D}_\bullet\ (\mathit{fid\text{-}on}\ A) = A$ **by** *force*

lemma *fdomain-fconst-on*[*simp*]: $\mathcal{D}_\bullet\ (\mathit{fconst\text{-}on}\ A\ c) = A$ **by** *force*

lemma *fdomain-flrestriction*: $\mathcal{D}_\bullet\ (r \upharpoonright^l_\bullet A) = \mathcal{D}_\bullet\ r \cap_\circ A$ **by** *auto*

Special properties.

lemma *fdomain-vsubset-ftimes*:
assumes *fpairs* $r \subseteq_\circ A \times_\bullet B$
shows $\mathcal{D}_\bullet\ r \subseteq_\circ A$
using *assms* **by** *blast*

lemma *fdomain-vsubset-VUnion2*: $\mathcal{D}_\bullet\ r \subseteq_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ r))$
proof(*intro vsubsetI*)
fix x **assume** $x \in_\circ \mathcal{D}_\bullet\ r$
then obtain y **where** $[x, y]_\circ \in_\circ r$ **by** *auto*
then have *set* $\{\langle 0_\mathbb{N}, x\rangle, \langle 1_\mathbb{N}, y\rangle\} \in_\circ r$ **unfolding** *vcons-vdoubleton* **by** *simp*
with *insert-commute* **have** $\langle 0_\mathbb{N}, x\rangle \in_\circ \bigcup_\circ r$ **by** *auto*
then show $x \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ r))$
unfolding *vpair-def*
by (*metis* (*full-types*) *VUnion-iff insert-commute vintersection-vdoubleton*)
qed

Range

definition *frange* :: $V \Rightarrow V$ ($\langle\mathcal{R}_\bullet\rangle$)
where *frange* r = *set* $\{b.\ \exists a.\ [a, b]_\circ \in_\circ r\}$
notation *frange* ($\langle\mathcal{R}_\bullet\rangle$)

lemma *frange-small*[*simp*]: *small* $\{b.\ \exists a.\ [a, b]_\circ \in_\circ r\}$
proof–
have *ss*: $\{b.\ \exists a.\ [a, b]_\circ \in_\circ r\} \subseteq (\lambda x.\ x(\lfloor 1_\mathbb{N}\rfloor))$ ‘ *elts* r
using *image-iff* **by** (*fastforce simp*: *nat-omega-simps*)
have *small*: *small* $((\lambda x.\ x(\lfloor 1_\mathbb{N}\rfloor))$ ‘ *elts* $r)$ **by** (*rule replacement*) *simp*
show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

Rules.

lemma *frangeI*[*intro*]:
assumes $[a, b]_\circ \in_\circ r$
shows $b \in_\circ \mathcal{R}_\bullet\ r$
using *assms* **unfolding** *frange-def* **by** *auto*

lemma *frangeD*[*dest*]:
assumes $b \in_\circ \mathcal{R}_\bullet\ r$
shows $\exists a.\ [a, b]_\circ \in_\circ r$
using *assms* **unfolding** *frange-def* **by** *simp*

lemma *frangeE*[*elim*!]:
assumes $b \in_\circ \mathcal{R}_\bullet\ r$
obtains a **where** $[a, b]_\circ \in_\circ r$
using *assms* **unfolding** *frange-def* **by** *clarsimp*

lemma *frange-iff*: $b \in_\circ \mathcal{R}_\bullet\ r \longleftrightarrow (\exists\, a.\ [a,\ b]_\circ \in_\circ r)$ **by** *auto*

Set operations.

lemma *frange-vempty*[*simp*]: $\mathcal{R}_\bullet\ 0 = 0$ **by** *auto*

lemma *frange-vsingleton*[*simp*]: $\mathcal{R}_\bullet\ (set\ \{[a,\ b]_\circ\}) = set\ \{b\}$ **by** *auto*

lemma *frange-vdoubleton*[*simp*]: $\mathcal{R}_\bullet\ (set\ \{[a,\ b]_\circ,\ [c,\ d]_\circ\}) = set\ \{b,\ d\}$
by *force*

lemma *frange-mono*:
assumes $r \subseteq_\circ s$
shows $\mathcal{R}_\bullet\ r \subseteq_\circ \mathcal{R}_\bullet\ s$
using *assms* **by** *force*

lemma *frange-vinsert*[*simp*]: $\mathcal{R}_\bullet\ (vinsert\ [a,\ b]_\circ\ r) = vinsert\ b\ (\mathcal{R}_\bullet\ r)$ **by** *auto*

lemma *frange-vunion*: $\mathcal{R}_\bullet\ (r \cup_\circ s) = \mathcal{R}_\bullet\ r \cup_\circ \mathcal{R}_\bullet\ s$ **by** *auto*

lemma *frange-vintersection-vsubset*: $\mathcal{R}_\bullet\ (r \cap_\circ s) \subseteq_\circ \mathcal{R}_\bullet\ r \cap_\circ \mathcal{R}_\bullet\ s$ **by** *auto*

lemma *frange-vdiff-vsubset*: $\mathcal{R}_\bullet\ r -_\circ \mathcal{R}_\bullet\ s \subseteq_\circ \mathcal{R}_\bullet\ (r -_\circ s)$ **by** *auto*

Connections.

lemma *frange-fid-on*[*simp*]: $\mathcal{R}_\bullet\ (fid\text{-}on\ A) = A$ **by** *force*

lemma *frange-fconst-on-vempty*[*simp*]: $\mathcal{R}_\bullet\ (fconst\text{-}on\ 0\ c) = 0$ **by** *auto*

lemma *frange-fconst-on-ne*[*simp*]:
assumes $A \neq 0$
shows $\mathcal{R}_\bullet\ (fconst\text{-}on\ A\ c) = set\ \{c\}$
using *assms* **by** *force*

lemma *frange-vrrestriction*: $\mathcal{R}_\bullet\ (r \restriction^r_\bullet A) = \mathcal{R}_\bullet\ r \cap_\circ A$ **by** *auto*

Previous connections

lemma *fdomain-fconverse*[*simp*]: $\mathcal{D}_\bullet\ (r^{-1}{}_\bullet) = \mathcal{R}_\bullet\ r$ **by** *auto*

lemma *frange-fconverse*[*simp*]: $\mathcal{R}_\bullet\ (r^{-1}{}_\bullet) = \mathcal{D}_\bullet\ r$ **by** *force*

Special properties.

lemma *frange-iff-vdomain*: $b \in_\circ \mathcal{R}_\bullet\ r \longleftrightarrow (\exists\, a{\in_\circ}\mathcal{D}_\bullet\ r.\ [a,\ b]_\circ \in_\circ r)$ **by** *auto*

lemma *frange-vsubset-ftimes*:
assumes *fpairs* $r \subseteq_\circ A \times_\bullet B$
shows $\mathcal{R}_\bullet\ r \subseteq_\circ B$
using *assms* **by** *blast*

lemma *fpairs-vsubset-fdomain-frange*[*simp*]: *fpairs* $r \subseteq_\circ (\mathcal{D}_\bullet\ r) \times_\bullet (\mathcal{R}_\bullet\ r)$
by *blast*

lemma *frange-vsubset-VUnion*2: $\mathcal{R}_\bullet\ r \subseteq_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ r))$
proof(*intro vsubsetI*)
fix y **assume** $y \in_\circ \mathcal{R}_\bullet\ r$
then obtain x **where** $[x,\ y]_\circ \in_\circ r$ **by** *auto*
then have $set\ \{\langle 0_{\mathbb{N}},\ x\rangle,\ \langle 1_{\mathbb{N}},\ y\rangle\} \in_\circ r$ **unfolding** *vcons-vdoubleton* **by** *simp*

with *insert-commute* **have** $\langle 1_{\mathbb{N}},\ y\rangle \in_\circ \bigcup_\circ r$ **by** *auto*
then show $y \in_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ r))$
unfolding *vpair-def*
by (*metis* (*full-types*) *VUnion-iff insert-commute vintersection-vdoubleton*)
qed

Field

definition *ffield* :: $V \Rightarrow V$
where *ffield* $r = \mathcal{D}_\bullet\ r \cup_\circ \mathcal{R}_\bullet\ r$

abbreviation *app-ffield* :: $V \Rightarrow V$ $(\langle\mathcal{F}_\bullet\rangle)$
where $\mathcal{F}_\bullet\ r \equiv$ *ffield* r

Rules.

lemma *ffieldI1*[*intro*]:
assumes $a \in_\circ \mathcal{D}_\bullet\ r \cup_\circ \mathcal{R}_\bullet\ r$
shows $a \in_\circ$ *ffield* r
using *assms* **unfolding** *ffield-def* **by** *simp*

lemma *ffieldI2*[*intro*]:
assumes $[a,\ b]_\circ \in_\circ r$
shows $a \in_\circ$ *ffield* r
using *assms* **by** *auto*

lemma *ffieldI3*[*intro*]:
assumes $[a,\ b]_\circ \in_\circ r$
shows $b \in_\circ$ *ffield* r
using *assms* **by** *auto*

lemma *ffieldD*[*intro*]:
assumes $a \in_\circ$ *ffield* r
shows $a \in_\circ \mathcal{D}_\bullet\ r \cup_\circ \mathcal{R}_\bullet\ r$
using *assms* **unfolding** *ffield-def* **by** *simp*

lemma *ffieldE*[*elim*]:
assumes $a \in_\circ$ *ffield* r **and** $a \in_\circ \mathcal{D}_\bullet\ r \cup_\circ \mathcal{R}_\bullet\ r \Longrightarrow P$
shows P
using *assms* **by** (*auto dest*: *ffieldD*)

lemma *ffield-pair*[*elim*]:
assumes $a \in_\circ$ *ffield* r
obtains b **where** $[a,\ b]_\circ \in_\circ r \vee [b,\ a]_\circ \in_\circ r$
using *assms* **by** *auto*

lemma *ffield-iff*: $a \in_\circ$ *ffield* $r \longleftrightarrow (\exists\, b.\ [a,\ b]_\circ \in_\circ r \vee [b,\ a]_\circ \in_\circ r)$ **by** *auto*

Set operations.

lemma *ffield-vempty*[*simp*]: *ffield* $0 = 0$ **by** *force*

lemma *ffield-vsingleton*[*simp*]: *ffield* $(set\ \{[a,\ b]_\circ\}) = set\ \{a,\ b\}$ **by** *force*

lemma *ffield-vdoubleton*[*simp*]:
ffield $(set\ \{[a,\ b]_\circ,\ [c,\ d]_\circ\}) = set\ \{a,\ b,\ c,\ d\}$
by *force*

lemma *ffield-mono*:
assumes $r \subseteq_\circ s$

shows *ffield* $r \subseteq_\circ$ *ffield* s
using *assms* **by** *fastforce*

lemma *ffield-vinsert*[*simp*]:
ffield (*vinsert* $[a, b]_\circ$ r) = *set* $\{a, b\} \cup_\circ$ (*ffield* r)
apply (*intro vsubset-antisym*; *intro vsubsetI*)
subgoal by *auto*
subgoal by (*metis ffield-iff vinsert-iff vinsert-vinsert*)
done

lemma *ffield-vunion*[*simp*]: *ffield* ($r \cup_\circ s$) = *ffield* $r \cup_\circ$ *ffield* s
unfolding *ffield-def* **by** *auto*

Connections.

lemma *fid-on-ffield*[*simp*]: *ffield* (*fid-on* A) = A **by** *force*

lemma *fconst-on-ffield-ne*[*intro*, *simp*]:
assumes $A \neq 0$
shows *ffield* (*fconst-on* A c) = *vinsert* c A
using *assms* **by** *force*

lemma *fconst-on-ffield-vempty*[*simp*]: *ffield* (*fconst-on* 0 c) = 0 **by** *auto*

lemma *ffield-fconverse*[*simp*]: *ffield* ($r^{-1}{}_\bullet$) = *ffield* r **by** *force*

Special properties.

lemma *ffield-vsubset-VUnion*2: $\mathcal{F}_\bullet\ r \subseteq_\circ \bigcup_\circ(\bigcup_\circ(\bigcup_\circ r))$
using *fdomain-vsubset-VUnion*2 *frange-vsubset-VUnion*2 **by** (*auto simp*: *ffield-def*)

Image

definition *fimage* :: $V \Rightarrow V \Rightarrow V$ (**infixr** ‹‘$_\bullet$› 90)
where r ‘$_\bullet$ A = $\mathcal{R}_\bullet$ (r $\restriction^l_\bullet$ A)
notation *fimage* (**infixr** ‹‘$_\bullet$› 90)

lemma *fimage-small*[*simp*]: *small* $\{b.\ \exists\, a{\in_\circ}A.\ [a, b]_\circ \in_\circ r\}$
proof–
from *image-iff ord-of-nat-succ-vempty* **have** *ss*:
$\{b.\ \exists\, a{\in_\circ}A.\ [a, b]_\circ \in_\circ r\} \subseteq (\lambda x.\ x(\lfloor 1_{\mathbb{N}} \rfloor))$ ‘ *elts* r
by *fastforce*
have *small*: *small* $((\lambda x.\ x(\lfloor 1_{\mathbb{N}} \rfloor))$ ‘ *elts* $r)$ **by** (*rule replacement*) *simp*
show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)
qed

Rules.

lemma *fimageI*1:
assumes $x \in_\circ \mathcal{R}_\bullet$ (r $\restriction^l_\bullet$ A)
shows $x \in_\circ r$ ‘$_\bullet$ A
using *assms* **unfolding** *fimage-def* **by** *simp*

lemma *fimageI*2[*intro*]:
assumes $[a, b]_\circ \in_\circ r$ **and** $a \in_\circ A$
shows $b \in_\circ r$ ‘$_\bullet$ A
using *assms fimageI*1 **by** *auto*

lemma *fimageD*[*dest*]:
assumes $x \in_\circ r$ ‘$_\bullet$ A
shows $x \in_\circ \mathcal{R}_\bullet$ (r $\restriction^l_\bullet$ A)

using *assms* **unfolding** *fimage-def* **by** *simp*

lemma *fimageE*[*elim*]:
assumes $b \in_\circ r \mathbin{`\!\bullet} A$
obtains a **where** $[a, b]_\circ \in_\circ r$ **and** $a \in_\circ A$
using *assms* **unfolding** *fimage-def* **by** *auto*

lemma *fimage-iff*: $b \in_\circ r \mathbin{`\!\bullet} A \longleftrightarrow (\exists a \in_\circ A.\ [a, b]_\circ \in_\circ r)$ **by** *auto*

Set operations.

lemma *fimage-vempty*[*simp*]: $0 \mathbin{`\!\bullet} A = 0$ **by** *force*

lemma *fimage-of-vempty*[*simp*]: $r \mathbin{`\!\bullet} 0 = 0$ **by** *force*

lemma *fimage-vsingleton-in*[*intro*, *simp*]:
assumes $a \in_\circ A$
shows *set* $\{[a, b]_\circ\} \mathbin{`\!\bullet} A = set\ \{b\}$
using *assms* **by** *auto*

lemma *fimage-vsingleton-nin*[*intro*, *simp*]:
assumes $a \notin_\circ A$
shows *set* $\{[a, b]_\circ\} \mathbin{`\!\bullet} A = 0$
using *assms* **by** *auto*

lemma *fimage-vsingleton-vinsert*[*intro*, *simp*]:
set $\{[a, b]_\circ\} \mathbin{`\!\bullet} vinsert\ a\ A = set\ \{b\}$
by *auto*

lemma *fimage-mono*:
assumes $r' \subseteq_\circ r$ **and** $A' \subseteq_\circ A$
shows $(r' \mathbin{`\!\bullet} A') \subseteq_\circ (r \mathbin{`\!\bullet} A)$
using *assms* **by** *fastforce*

lemma *fimage-vinsert*: $r \mathbin{`\!\bullet} (vinsert\ a\ A) = r \mathbin{`\!\bullet} set\ \{a\} \cup_\circ r \mathbin{`\!\bullet} A$ **by** *auto*

lemma *fimage-vunion-left*: $(r \cup_\circ s) \mathbin{`\!\bullet} A = r \mathbin{`\!\bullet} A \cup_\circ s \mathbin{`\!\bullet} A$ **by** *auto*

lemma *fimage-vunion-right*: $r \mathbin{`\!\bullet} (A \cup_\circ B) = r \mathbin{`\!\bullet} A \cup_\circ r \mathbin{`\!\bullet} B$ **by** *auto*

lemma *fimage-vintersection*: $r \mathbin{`\!\bullet} (A \cap_\circ B) \subseteq_\circ r \mathbin{`\!\bullet} A \cap_\circ r \mathbin{`\!\bullet} B$ **by** *auto*

lemma *fimage-vdiff*: $r \mathbin{`\!\bullet} A -_\circ r \mathbin{`\!\bullet} B \subseteq_\circ r \mathbin{`\!\bullet} (A -_\circ B)$ **by** *auto*

Special properties.

lemma *fimage-vsingleton-iff*[*iff*]: $b \in_\circ r \mathbin{`\!\bullet} set\ \{a\} \longleftrightarrow [a, b]_\circ \in_\circ r$ **by** *auto*

lemma *fimage-is-vempty*[*iff*]: $r \mathbin{`\!\bullet} A = 0 \longleftrightarrow vdisjnt\ (\mathcal{D}_\bullet\ r)\ A$ **by** *fastforce*

Connections.

lemma *fid-on-fimage*[*simp*]: $(fid\text{-}on\ A) \mathbin{`\!\bullet} B = A \cap_\circ B$ **by** *force*

lemma *fimage-fconst-on-ne*[*simp*]:
assumes $B \cap_\circ A \neq 0$
shows $(fconst\text{-}on\ A\ c) \mathbin{`\!\bullet} B = set\ \{c\}$
using *assms* **by** *auto*

lemma *fimage-fconst-on-vempty*[*simp*]:

assumes *vdisjnt A B*
shows (*fconst-on A c*) ‘• *B* = 0
using *assms* **by** *auto*

lemma *fimage-fconst-on-vsubset-const*[*simp*]: (*fconst-on A c*) ‘• *B* ⊆∘ *set* {*c*}
by *auto*

lemma *fcomp-frange*: $\mathcal{R}_\bullet$ (*r* ∘• *s*) = *r* ‘• ($\mathcal{R}_\bullet$ *s*) **by** *blast*

lemma *fcomp-fimage*: (*r* ∘• *s*) ‘• *A* = *r* ‘• (*s* ‘• *A*) **by** *blast*

lemma *fimage-fltrestriction*[*simp*]: (*r* ↾$^l_\bullet$ *A*) ‘• *B* = *r* ‘• (*A* ∩∘ *B*) **by** *auto*

lemma *fimage-frrestriction*[*simp*]: (*r* ↾$^r_\bullet$ *A*) ‘• *B* = *A* ∩∘ *r* ‘• *B* **by** *auto*

lemma *fimage-frestriction*[*simp*]: (*r* ↾• *A*) ‘• *B* = *A* ∩∘ (*r* ‘• (*A* ∩∘ *B*)) **by** *auto*

lemma *fimage-fdomain*: *r* ‘• $\mathcal{D}_\bullet$ *r* = $\mathcal{R}_\bullet$ *r* **by** *auto*

lemma *fimage-eq-imp-fcomp*:
assumes *f* ‘• *A* = *g* ‘• *B*
shows (*h* ∘• *f*) ‘• *A* = (*h* ∘• *g*) ‘• *B*
using *assms* **by** (*metis fcomp-fimage*)

Previous connections.

lemma *fcomp-rel-fconst-on-ftimes*: *r* ∘• (*fconst-on A c*) = *A* ×• (*r* ‘• *set* {*c*})
by *blast*

Further special properties.

lemma *fimage-vsubset*:
assumes *r* ⊆∘ *A* ×• *B*
shows *r* ‘• *C* ⊆∘ *B*
using *assms* **by** *blast*

lemma *fimage-set-def*: *r* ‘• *A* = *set* {*b*. ∃ *a*∈∘*A*. [*a*, *b*]∘ ∈∘ *r*}
unfolding *fimage-def frange-def* **by** *auto*

lemma *fimage-vsingleton*: *r* ‘• *set* {*a*} = *set* {*b*. [*a*, *b*]∘ ∈∘ *r*}
proof–
have {*b*. [*a*, *b*]∘ ∈∘ *r*} ⊆ {*b*. ∃ *a*. [*a*, *b*]∘ ∈∘ *r*} **by** *auto*
then have [*simp*]: *small* {*b*. [*a*, *b*]∘ ∈∘ *r*}
by (*rule smaller-than-small*[*OF frange-small*[*of r*]])
show *?thesis* **by** *auto*
qed

lemma *fimage-strict-vsubset*: *f* ‘• *A* ⊆∘ *f* ‘• $\mathcal{D}_\bullet$ *f* **by** *auto*

Inverse image

definition *finvimage* :: *V* ⇒ *V* ⇒ *V* (**infixr** ‹−‘•› 90)
where *r* −‘• *A* = *r*$^{-1}_\bullet$ ‘• *A*

lemma *finvimage-small*[*simp*]: *small* {*a*. ∃ *b*∈∘*A*. [*a*, *b*]∘ ∈∘ *r*}
proof–
have *ss*: {*a*. ∃ *b*∈∘*A*. [*a*, *b*]∘ ∈∘ *r*} ⊆ (λ*x*. *x*(|$0_\mathbb{N}$|)) ‘ *elts r*
using *image-iff* **by** *fastforce*
have *small*: *small* ((λ*x*. *x*(|$0_\mathbb{N}$|)) ‘ *elts r*) **by** (*rule replacement*) *simp*
show *?thesis* **by** (*rule smaller-than-small*, *rule small*, *rule ss*)

qed

Rules.

lemma *finvimageI*[*intro*]:
assumes $[a,\ b]_\circ \in_\circ r$ **and** $b \in_\circ A$
shows $a \in_\circ r -`\!`\!\bullet\ A$
using *assms finvimage-def* **by** *auto*

lemma *finvimageD*[*dest*]:
assumes $a \in_\circ r -`\!`\!\bullet\ A$
shows $a \in_\circ \mathcal{D}_\bullet\ (r \restriction^r_\bullet A)$
using *assms* **using** *finvimage-def* **by** *auto*

lemma *finvimageE*[*elim*]:
assumes $a \in_\circ r -`\!`\!\bullet\ A$
obtains b **where** $[a,\ b]_\circ \in_\circ r$ **and** $b \in_\circ A$
using *assms* **unfolding** *finvimage-def* **by** *auto*

lemma *finvimageI*1:
assumes $a \in_\circ \mathcal{D}_\bullet\ (r \restriction^r_\bullet A)$
shows $a \in_\circ r -`\!`\!\bullet\ A$
using *assms* **unfolding** *fimage-def*
by (*simp add*: *finvimage-def fimageI*1 *frestriction-fconverse*)

lemma *finvimageD*1:
assumes $a \in_\circ r -`\!`\!\bullet\ A$
shows $a \in_\circ \mathcal{D}_\bullet\ (r \restriction^r_\bullet A)$
using *assms* **by** *fastforce*

lemma *finvimageE*1:
assumes $a \in_\circ r -`\!`\!\bullet\ A$ **and** $a \in_\circ \mathcal{D}_\bullet\ (r \restriction^r_\bullet A) \Longrightarrow P$
shows P
using *assms* **by** *auto*

lemma *finvimageI*2:
assumes $a \in_\circ r^{-1}_\bullet\ `\!`\!\bullet\ A$
shows $a \in_\circ r -`\!`\!\bullet\ A$
using *assms* **unfolding** *finvimage-def* **by** *simp*

lemma *finvimageD*2:
assumes $a \in_\circ r -`\!`\!\bullet\ A$
shows $a \in_\circ r^{-1}_\bullet\ `\!`\!\bullet\ A$
using *assms* **unfolding** *finvimage-def* **by** *simp*

lemma *finvimageE*2:
assumes $a \in_\circ r -`\!`\!\circ\ A$ **and** $a \in_\circ r^{-1}_\circ\ `\!`\!\circ\ A \Longrightarrow P$
shows P
unfolding *vimage-def* **using** *assms* **by** *blast*

lemma *finvimage-iff*: $a \in_\circ r -`\!`\!\bullet\ A \longleftrightarrow (\exists\, b{\in_\circ}A.\ [a,\ b]_\circ \in_\circ r)$ **by** *auto*

lemma *finvimage-iff*1: $a \in_\circ r -`\!`\!\bullet\ A \longleftrightarrow a \in_\circ \mathcal{D}_\bullet\ (r \restriction^r_\bullet A)$ **by** *auto*

lemma *finvimage-iff*2: $a \in_\circ r -`\!`\!\bullet\ A \longleftrightarrow a \in_\circ r^{-1}_\bullet\ `\!`\!\bullet\ A$ **by** *auto*

Set operations.

lemma *finvimage-vempty*[*simp*]: $0 -`\!`\!\bullet\ A = 0$ **by** *force*

lemma *finvimage-of-vempty*[*simp*]: $r -`_\bullet 0 = 0$ **by** *force*

lemma *finvimage-vsingleton-in*[*intro*, *simp*]:
assumes $b \in_\circ A$
shows $set\ \{[a, b]_\circ\} -`_\bullet A = set\ \{a\}$
using *assms* **by** *auto*

lemma *finvimage-vsingleton-nin*[*intro*, *simp*]:
assumes $b \notin_\circ A$
shows $set\ \{[a, b]_\circ\} -`_\bullet A = 0$
using *assms* **by** *auto*

lemma *finvimage-vsingleton-vinsert*[*intro*, *simp*]:
$set\ \{[a, b]_\circ\} -`_\bullet vinsert\ b\ A = set\ \{a\}$
by *auto*

lemma *finvimage-mono*:
assumes $r' \subseteq_\circ r$ **and** $A' \subseteq_\circ A$
shows $(r' -`_\bullet A') \subseteq_\circ (r -`_\bullet A)$
using *assms* **by** *fastforce*

lemma *finvimage-vinsert*: $r -`_\bullet (vinsert\ a\ A) = r -`_\bullet set\ \{a\} \cup_\circ r -`_\bullet A$ **by** *auto*

lemma *finvimage-vunion-left*: $(r \cup_\circ s) -`_\bullet A = r -`_\bullet A \cup_\circ s -`_\bullet A$ **by** *auto*

lemma *finvimage-vunion-right*: $r -`_\bullet (A \cup_\circ B) = r -`_\bullet A \cup_\circ r -`_\bullet B$ **by** *auto*

lemma *finvimage-vintersection*: $r -`_\bullet (A \cap_\circ B) \subseteq_\circ r -`_\bullet A \cap_\circ r -`_\bullet B$ **by** *auto*

lemma *finvimage-vdiff*: $r -`_\bullet A -_\circ r -`_\bullet B \subseteq_\circ r -`_\bullet (A -_\circ B)$ **by** *auto*

Special properties.

lemma *finvimage-set-def*: $r -`_\bullet A = set\ \{a.\ \exists b{\in_\circ}A.\ [a, b]_\circ \in_\circ r\}$ **by** *fastforce*

lemma *finvimage-eq-fdomain-frestriction*: $r -`_\bullet A = \mathcal{D}_\bullet (r \restriction^r_\bullet A)$ **by** *fastforce*

lemma *finvimage-frange*[*simp*]: $r -`_\bullet \mathcal{R}_\bullet r = \mathcal{D}_\bullet r$
unfolding *invimage-def* **by** *force*

lemma *finvimage-frange-vsubset*[*simp*]:
assumes $\mathcal{R}_\bullet r \subseteq_\circ B$
shows $r -`_\bullet B = \mathcal{D}_\bullet r$
using *assms* **unfolding** *finvimage-def* **by** *force*

Connections.

lemma *finvimage-fid-on*[*simp*]: $(fid\text{-}on\ A) -`_\bullet B = A \cap_\circ B$ **by** *force*

lemma *finvimage-fconst-on-vsubset-fdomain*[*simp*]: $(fconst\text{-}on\ A\ c) -`_\bullet B \subseteq_\circ A$
unfolding *finvimage-def* **by** *blast*

lemma *finvimage-fconst-on-ne*[*simp*]:
assumes $c \in_\circ B$
shows $(fconst\text{-}on\ A\ c) -`_\bullet B = A$
by (*simp add*: *assms finvimage-eq-fdomain-frestriction frrestriction-fconst-on*)

lemma *finvimage-fconst-on-vempty*[*simp*]:
assumes $c \notin_\circ B$
shows $(fconst\text{-}on\ A\ c) -`_\bullet B = 0$

using *assms* **by** *auto*

lemma *finvimage-fcomp*: $(g \circ_\bullet f) -`_\bullet x = f -`_\bullet (g -`_\bullet x)$
by (*simp add*: *finvimage-def fconverse-fcomp fcomp-fimage*)

lemma *finvimage-fconverse*[*simp*]: $r^{-1}{}_\bullet -`_\bullet A = r `_\bullet A$ **by** *auto*

lemma *finvimage-flrestriction*[*simp*]: $(r \restriction^l_\bullet A) -`_\bullet B = A \cap_\circ r -`_\bullet B$ **by** *auto*

lemma *finvimage-frrestriction*[*simp*]: $(r \restriction^r_\bullet A) -`_\bullet B = (r -`_\bullet (A \cap_\circ B))$ **by** *auto*

lemma *finvimage-frestriction*[*simp*]: $(r \restriction_\bullet A) -`_\bullet B = A \cap_\circ (r -`_\bullet (A \cap_\circ B))$
by *blast*

Previous connections.

lemma *fdomain-fcomp*[*simp*]: $\mathcal{D}_\bullet (r \circ_\bullet s) = s -`_\bullet \mathcal{D}_\bullet r$ **by** *force*

2.10.5 Classification of relations

Binary relation

locale *fbrelation* =
fixes $r :: V$
assumes *fbrelation*[*simp*]: *fpairs* $r = r$

locale *fbrelation-pair* = r_1: *fbrelation* r_1 + r_2: *fbrelation* r_2 **for** r_1 r_2

Rules.

lemma *fpairs-eqI*[*intro*!]:
assumes $\bigwedge x.\ x \in_\circ r \Longrightarrow \exists\, a\ b.\ x = [a,\ b]_\circ$
shows *fpairs* $r = r$
using *assms* **by** *auto*

lemma *fpairs-eqD*[*dest*]:
assumes *fpairs* $r = r$
shows $\bigwedge x.\ x \in_\circ r \Longrightarrow \exists\, a\ b.\ x = [a,\ b]_\circ$
using *assms* **by** *auto*

lemma *fpairs-eqE*[*elim*!]:
assumes *fpairs* $r = r$ **and** $(\bigwedge x.\ x \in_\circ r \Longrightarrow \exists\, a\ b.\ x = [a,\ b]_\circ) \Longrightarrow P$
shows P
using *assms* **by** *auto*

lemmas *fbrelationI*[*intro*!] = *fbrelation.intro*
lemmas *fbrelationD*[*dest*!] = *fbrelation.fbrelation*

lemma *fbrelationE*[*elim*!]:
assumes *fbrelation* r **and** $(\mathit{fpairs}\ r = r) \Longrightarrow P$
shows P
using *assms* **unfolding** *fbrelation-def* **by** *auto*

lemma *fbrelationE*1:
assumes *fbrelation* r **and** $x \in_\circ r$
obtains $a\ b$ **where** $x = [a,\ b]_\circ$
using *assms* **by** *auto*

lemma *fbrelationD*1[*dest*]:
assumes *fbrelation* r **and** $x \in_\circ r$

shows $\exists\, a\ b.\ x = [a,\ b]_\circ$
using *assms* **by** *auto*

Set operations.

lemma *fbrelation-vsubset*:
assumes *fbrelation s* **and** $r \subseteq_\circ s$
shows *fbrelation r*
using *assms* **by** *auto*

lemma *fbrelation-vinsert*: *fbrelation* (*vinsert* $[a,\ b]_\circ$ *r*) $\longleftrightarrow$ *fbrelation r*
by *auto*

lemma (**in** *fbrelation*) *fbrelation-vinsertI*: *fbrelation* (*vinsert* $[a,\ b]_\circ$ *r*)
using *fbrelation-axioms* **by** *auto*

lemma *fbrelation-vinsertD*[*dest*]:
assumes *fbrelation* (*vinsert* $\langle a,\ b\rangle$ *r*)
shows *fbrelation r*
using *assms* **by** *auto*

lemma *fbrelation-vunion*: *fbrelation* ($r \cup_\circ s$) $\longleftrightarrow$ *fbrelation r* $\wedge$ *fbrelation s*
by *auto*

lemma (**in** *fbrelation-pair*) *fbrelation-vunionI*: *fbrelation* ($r_1 \cup_\circ r_2$)
using r_1.*fbrelation-axioms* r_2.*fbrelation-axioms* **by** *auto*

lemma *fbrelation-vunionD*[*dest*]:
assumes *fbrelation* ($r \cup_\circ s$)
shows *fbrelation r* **and** *fbrelation s*
using *assms* **by** *auto*

lemma (**in** *fbrelation*) *fbrelation-vintersectionI*: *fbrelation* ($r \cap_\circ s$)
using *fbrelation-axioms* **by** *auto*

lemma (**in** *fbrelation*) *fbrelation-vdiffI*: *fbrelation* ($r -_\circ s$)
using *fbrelation-axioms* **by** *auto*

Connections.

lemma *fbrelation-vempty*: *fbrelation* 0 **by** *auto*

lemma *fbrelation-vsingleton*: *fbrelation* (*set* $\{[a,\ b]_\circ\}$) **by** *auto*

global-interpretation *frel-vsingleton*: *fbrelation* ‹*set* $\{[a,\ b]_\circ\}$›
by (*rule fbrelation-vsingleton*)

lemma *fbrelation-vdoubleton*: *fbrelation* (*set* $\{[a,\ b]_\circ, [c,\ d]_\circ\}$) **by** *auto*

lemma *fbrelation-sid-on*[*simp*]: *fbrelation* (*fid-on A*) **by** *auto*

lemma *fbrelation-fconst-on*[*simp*]: *fbrelation* (*fconst-on A c*) **by** *auto*

lemma (**in** *fbrelation-pair*) *fbrelation-fcomp*: *fbrelation* ($r_1 \circ_\bullet r_2$)
using r_1.*fbrelation-axioms* r_2.*fbrelation-axioms* **by** *auto*

sublocale *fbrelation-pair* $\subseteq$ *fcomp*$_{21}$: *fbrelation* ‹$r_2 \circ_\bullet r_1$›
by
(
simp add:

fbrelation-pair.fbrelation-fcomp
fbrelation-pair-def
r_1.*fbrelation-axioms*
r_2.*fbrelation-axioms*
)

sublocale *fbrelation-pair* ⊆ *fcomp*$_{12}$: *fbrelation* ‹r_1 ∘$_\bullet$ r_2›
by (*rule fbrelation-fcomp*)

lemma (**in** *fbrelation*) *fbrelation-fconverse*: *fbrelation* ($r^{-1}{}_\bullet$)
using *fbrelation-axioms* **by** *clarsimp*

lemma *fbrelation-flrestriction*[*intro*, *simp*]: *fbrelation* (r ↾$^l_\bullet$ A) **by** *auto*

lemma *fbrelation-frrestriction*[*intro*, *simp*]: *fbrelation* (r ↾$^r_\bullet$ A) **by** *auto*

lemma *fbrelation-frestriction*[*intro*, *simp*]: *fbrelation* (r ↾$_\bullet$ A) **by** *auto*

Previous connections.

lemma (**in** *fbrelation*) *fconverse-fconverse*[*simp*]: ($r^{-1}{}_\bullet$)$^{-1}{}_\bullet$ = r
using *fbrelation-axioms* **by** *auto*

lemma (**in** *fbrelation-pair*) *fconverse-mono*[*simp*]: $r_1{}^{-1}{}_\bullet$ ⊆$_\circ$ $r_2{}^{-1}{}_\bullet$ ⟷ r_1 ⊆$_\circ$ r_2
using r_1.*fbrelation-axioms* r_2.*fbrelation-axioms*
by (*force intro*: *fconverse-vunion*)+

lemma (**in** *fbrelation-pair*) *fconverse-inject*[*simp*]: $r_1{}^{-1}{}_\bullet$ = $r_2{}^{-1}{}_\bullet$ ⟷ r_1 = r_2
using r_1.*fbrelation-axioms* r_2.*fbrelation-axioms* **by** *fast*

lemma (**in** *fbrelation*) *fconverse-vsubset-swap*-2:
assumes $r^{-1}{}_\bullet$ ⊆$_\circ$ s
shows r ⊆$_\circ$ $s^{-1}{}_\bullet$
using *assms fbrelation-axioms* **by** *auto*

lemma (**in** *fbrelation*) *flrestriction-fdomain*[*simp*]: r ↾$^l_\bullet$ $\mathcal{D}_\bullet$ r = r
using *fbrelation-axioms* **by** (*elim fbrelationE*) *blast*

lemma (**in** *fbrelation*) *frrestriction-frange*[*simp*]: r ↾$^r_\bullet$ $\mathcal{R}_\bullet$ r = r
using *fbrelation-axioms* **by** (*elim fbrelationE*) *blast*

Special properties.

lemma *vsubset-vtimes-fbrelation*:
assumes r ⊆$_\circ$ A ×$_\bullet$ B
shows *fbrelation* r
using *assms* **by** *blast*

lemma (**in** *fbrelation*) *fbrelation-vintersection-vdomain*:
assumes *vdisjnt* ($\mathcal{D}_\bullet$ r) ($\mathcal{D}_\bullet$ s)
shows *vdisjnt* r s
proof(*rule vsubset-antisym*; *rule vsubsetI*)
fix x **assume** x ∈$_\circ$ r ∩$_\circ$ s
then obtain a b **where** [a, b]$_\circ$ ∈$_\circ$ r ∩$_\circ$ s
by (*metis fbrelationE*1 *fbrelation-vintersectionI*)
with *assms* **show** x ∈$_\circ$ 0 **by** *auto*
qed *simp*

lemma (**in** *fbrelation*) *fbrelation-vintersection-vrange*:
assumes *vdisjnt* ($\mathcal{R}_\bullet$ r) ($\mathcal{R}_\bullet$ s)

shows *vdisjnt r s*
proof(*rule vsubset-antisym*; *rule vsubsetI*)
fix *x* **assume** $x \in_\circ r \cap_\circ s$
then obtain *a b* **where** $[a, b]_\circ \in_\circ r \cap_\circ s$
by (*metis fbrelationE*1 *fbrelation-vintersectionI*)
with *assms* **show** $x \in_\circ 0$ **by** *auto*
qed *simp*

lemma (**in** *fbrelation*) *fbrelation-vintersection-vfield*:
assumes *vdisjnt* (*ffield r*) (*ffield s*)
shows *vdisjnt r s*
proof(*rule vsubset-antisym*; *rule vsubsetI*)
fix *x* **assume** $x \in_\circ r \cap_\circ s$
then obtain *a b* **where** $[a, b]_\circ \in_\circ r \cap_\circ s$
by (*metis fbrelationE*1 *fbrelation-vintersectionI*)
with *assms* **show** $x \in_\circ 0$ **by** *auto*
qed *auto*

lemma (**in** *fbrelation*) *vdomain-vrange-vtimes*: $r \subseteq_\circ \mathcal{D}_\bullet\ r \times_\bullet \mathcal{R}_\bullet\ r$
using *fbrelation* **by** *blast*

lemma (**in** *fbrelation*) *fconverse-eq-frel*[*intro*, *simp*]:
assumes $\bigwedge a\ b.\ [a, b]_\circ \in_\circ r \Longrightarrow [b, a]_\circ \in_\circ r$
shows $r^{-1}{}_\bullet = r$
using *assms*
apply (*intro vsubset-antisym*; *intro vsubsetI*)
subgoal by *blast*
subgoal by (*metis fconverseE fconverseI fconverse-fconverse*)
done

lemma *fcomp-fconverse-frel-eq-frel-fbrelationI*:
assumes $r^{-1}{}_\bullet \circ_\bullet r = r$
shows *fbrelation r*
using *assms* **by** (*intro fbrelationI*, *elim vequalityE vsubsetE*) *force*

Alternative forms of existing results.

lemmas [*intro*, *simp*] = *fbrelation.fconverse-fconverse*
and *fconverse-eq-frel*[*intro*, *simp*] = *fbrelation.fconverse-eq-frel*

context
fixes r_1 r_2
assumes r_1: *fbrelation* r_1
and r_2: *fbrelation* r_2
begin

lemmas-with[*OF fbrelation-pair.intro*[*OF* r_1 r_2]] :
fbrelation-fconverse-mono[*intro*, *simp*] = *fbrelation-pair.fconverse-mono*
and *fbrelation-frrestriction-srange*[*intro*, *simp*] =
fbrelation-pair.fconverse-inject

end

2.11 Further results related to the von Neumann hierarchy of sets

2.11.1 Background

The subsection presents several further auxiliary results about the von Neumann hierarchy of sets. The primary general reference for this section is [59].

2.11.2 Further properties of *Vfrom*

Reusable patterns.

lemma *Vfrom-Ord-bundle*:
assumes $A = A$ **and** $i = i$
shows *Vfrom A i* = *Vfrom A* (*rank i*) **and** *Ord* (*rank i*)
by (*simp-all add*: *Vfrom-rank-eq*)

lemma *Vfrom-in-bundle*:
assumes $i \in_\circ j$ **and** $A = A$ **and** $B = B$
shows *Vfrom A i* = *Vfrom A* (*rank i*)
and *Ord* (*rank i*)
and *Vfrom B j* = *Vfrom B* (*rank j*)
and *Ord* (*rank j*)
and *rank i* $\in_\circ$ *rank j*
by (*simp-all add*: *assms*(1) *Vfrom-rank-eq Ord-mem-iff-lt rank-lt*)

Elementary corollaries.

lemma *Ord-Vset-in-Vset-succI*[*intro*]:
assumes *Ord* α
shows *Vset* $\alpha \in_\circ$ *Vset* (*succ* α)
by (*simp add*: *Vset-succ assms*)

lemma *Ord-in-in-VsetI*[*intro*]:
assumes *Ord* α **and** $a \in_\circ \alpha$
shows $a \in_\circ$ *Vset* α
by (*metis assms Ord-VsetI Ord-iff-rank rank-lt*)

Transitivity of the constant *Vfrom*.

lemma *Vfrom-trans*[*intro*]:
assumes *Transset A* **and** $x \in_\circ X$ **and** $X \in_\circ$ *Vfrom A i*
shows $x \in_\circ$ *Vfrom A i*
using *Transset-def* **by** (*blast intro*: *assms Transset-Vfrom*)

lemma *Vset-trans*[*intro*]:
assumes $x \in_\circ X$ **and** $X \in_\circ$ *Vset i*
shows $x \in_\circ$ *Vset i*
by (*auto intro*: *assms*)

Monotonicity of the constant *Vfrom*.

lemma *Vfrom-in-mono*:
assumes $A \subseteq_\circ B$ **and** $i \in_\circ j$
shows *Vfrom A i* $\in_\circ$ *Vfrom B j*
proof–
define i' **where** $i' =$ *rank i*
define j' **where** $j' =$ *rank j*
note *rank-conv* =
Vfrom-in-bundle[

```
    OF assms(2) HOL.refl[of A] HOL.refl[of B], folded i'-def j'-def
    ]
  show ?thesis
    unfolding rank-conv using rank-conv(4,5)
  proof induction
    case (succ j')
    from succ have Ord (succ j') by auto
    from succ(3) succ.hyps have i' ⊆∘ j' by (auto simp: Ord-def Transset-def)
    from Vfrom-mono[OF ‹Ord i'› assms(1) this] show ?case
      unfolding Vfrom-succ-Ord[OF ‹Ord j'›, of B] by simp
  next
    case (Limit j')
    from Limit(3) obtain ξ where i' ∈∘ ξ and ξ ∈∘ j' by auto
    with vifunionI have Vfrom A i' ∈∘ (⋃∘ξ∈∘j'. Vfrom B ξ)
      by (auto simp: Limit.IH)
    then show Vfrom A i' ∈∘ Vfrom B (⋃∘ξ∈∘j'. ξ)
      unfolding Limit-Vfrom-eq[symmetric, OF Limit(1)]
      by (simp add: SUP-vifunion[symmetric] Limit.hyps)
  qed auto
qed

lemmas Vset-in-mono = Vfrom-in-mono[OF order-refl, of - - 0]

lemma Vfrom-vsubset-mono:
  assumes A ⊆∘ B and i ⊆∘ j
  shows Vfrom A i ⊆∘ Vfrom B j
  by (metis assms Vfrom-Ord-bundle(1,2) Vfrom-mono rank-mono)

lemmas Vset-vsubset-mono = Vfrom-vsubset-mono[OF order-refl, of - - 0]

lemma arg1-vsubset-Vfrom: a ⊆∘ Vfrom a i using Vfrom by blast

lemma VPow-vsubset-Vset:
  — Based on Theorem 9.10 from [59]
  assumes X ∈∘ Vset i
  shows VPow X ⊆∘ Vset i
proof–
  define i' where i' = rank i
  note rank-conv = Vfrom-Ord-bundle[OF refl[of 0] refl[of i], folded i'-def]
  show ?thesis
    using rank-conv(2) assms unfolding rank-conv
  proof induction
    case (Limit α)
    from Limit have X ∈∘ (⋃∘i∈∘α. Vset i)
      by (simp add: SUP-vifunion[symmetric] Limit-Vfrom-eq)
    then have VPow X ⊆∘ (⋃∘i∈∘α. Vset i)
      by (intro vsubsetI) (metis Limit.IH vifunionE vifunionI vsubsetE)
    then show ?case
      by (simp add: SUP-vifunion[symmetric] Limit.hyps Limit-Vfrom-eq)
  qed (simp-all add: Vset-succ)
qed

lemma Vfrom-vsubset-VPow-Vfrom:
  assumes Transset A
  shows Vfrom A i ⊆∘ VPow (Vfrom A i)
  using assms Transset-VPow Transset-Vfrom by (auto simp: Transset-def)

lemma arg1-vsubset-VPow-Vfrom:
```

assumes *Transset A*
shows $A \subseteq_\circ$ *VPow* (*Vfrom A i*)
by (*meson assms Vfrom-vsubset-VPow-Vfrom arg*1*-vsubset-Vfrom dual-order.trans*)

2.11.3 Operations closed with respect to *Vset*

Empty set.

lemma *Limit-vempty-in-VsetI*:
assumes *Limit* α
shows $0 \in_\circ$ *Vset* α
using *assms* **by** (*auto simp*: *Limit-def*)

Subset.

lemma *vsubset-in-VsetI*[*intro*]:
assumes $a \subseteq_\circ A$ **and** $A \in_\circ$ *Vset i*
shows $a \in_\circ$ *Vset i*
using *assms* **by** (*auto dest*: *VPow-vsubset-Vset*)

lemma *Ord-vsubset-in-Vset-succI*:
assumes *Ord* α **and** $A \subseteq_\circ$ *Vset* α
shows $A \in_\circ$ *Vset* (*succ* α)
using *assms Ord-Vset-in-Vset-succI* **by** *auto*

Power set.

lemma *Limit-VPow-in-VsetI*[*intro*]:
assumes *Limit* α **and** $A \in_\circ$ *Vset* α
shows *VPow A* $\in_\circ$ *Vset* α
proof–
from *assms*(1) **have** *Ord* α **by** *auto*
with *assms* **obtain** *i* **where** $A \in_\circ$ *Vset i* **and** $i \in_\circ \alpha$ **and** *Ord i*
by (*fastforce simp*: *Ord-in-Ord Limit-Vfrom-eq*)
have *Vset i* $\in_\circ$ *Vset* α **by** (*rule Vset-in-mono*) (*auto intro*: ‹$i \in_\circ \alpha$›)
from *VPow-vsubset-Vset*[*OF* ‹$A \in_\circ$ *Vset i*›] *this* **show** *?thesis*
by (*rule vsubset-in-VsetI*)
qed

lemma *VPow-in-Vset-revD*:
assumes *VPow A* $\in_\circ$ *Vset i*
shows $A \in_\circ$ *Vset i*
using *assms Vset-trans* **by** *blast*

lemma *Ord-VPow-in-Vset-succI*:
assumes *Ord* α **and** $a \in_\circ$ *Vset* α
shows *VPow a* $\in_\circ$ *Vset* (*succ* α)
using *VPow-vsubset-Vset*[*OF assms*(2)]
by (*auto intro*: *Ord-Vset-in-Vset-succI*[*OF assms*(1)])

lemma *Ord-VPow-in-Vset-succD*:
assumes *Ord* α **and** *VPow a* $\in_\circ$ *Vset* (*succ* α)
shows $a \in_\circ$ *Vset* α
using *assms* **by** (*fastforce dest*: *Vset-succ*)

Union of elements.

lemma *VUnion-in-VsetI*[*intro*]:
assumes $A \in_\circ$ *Vset i*
shows $\bigcup_\circ A \in_\circ$ *Vset i*
proof–

```
  define i' where i' = rank i
  note rank-conv = Vfrom-Ord-bundle[OF refl[of 0] refl[of i], folded i'-def]
  from rank-conv(2) assms show ?thesis
    unfolding rank-conv
  proof induction
    case (succ α)
    show ⋃₀A ∈₀ Vset (succ α)
      by (metis succ(1,3) VPow-iff VUnion-least Vset-trans Vset-succ)
  qed (auto simp: vrange-VLambda vimage-VLambda-vrange-rep Limit-Vfrom-eq)
qed

lemma Limit-VUnion-in-VsetD:
  assumes Limit α and ⋃₀A ∈₀ Vset α
  shows A ∈₀ Vset α
proof–
  have A ⊆₀ VPow (⋃₀A) by auto
  moreover from assms have VPow (⋃₀A) ∈₀ Vset α by (rule Limit-VPow-in-VsetI)
  ultimately show ?thesis using assms(1) by auto
qed
```

Intersection of elements.

```
lemma VInter-in-VsetI[intro]:
  assumes A ∈₀ Vset α
  shows ⋂₀A ∈₀ Vset α
proof–
  have subset: ⋂₀A ⊆₀ ⋃₀A by auto
  moreover from assms have ⋃₀A ∈₀ Vset α by (rule VUnion-in-VsetI)
  ultimately show ?thesis by (rule vsubset-in-VsetI)
qed
```

Singleton.

```
lemma Limit-vsingleton-in-VsetI[intro]:
  assumes Limit α and a ∈₀ Vset α
  shows set {a} ∈₀ Vset α
proof–
  have aa: set {a} ⊆₀ VPow a by auto
  from assms(1) have Ord α by auto
  from vsubset-in-VsetI[OF aa Limit-VPow-in-VsetI[OF assms(1)]] show ?thesis
    by (simp add: Limit-is-Ord assms(2))
qed

lemma Limit-vsingleton-in-VsetD:
  assumes set {a} ∈₀ Vset α
  shows a ∈₀ Vset α
  using assms by auto

lemma Ord-vsingleton-in-Vset-succI:
  assumes Ord α and a ∈₀ Vset α
  shows set {a} ∈₀ Vset (succ α)
  using assms by (simp add: Vset-succ vsubset-vsingleton-leftI)
```

Doubleton.

```
lemma Limit-vdoubleton-in-VsetI[intro]:
  assumes Limit α and a ∈₀ Vset α and b ∈₀ Vset α
  shows set {a, b} ∈₀ Vset α
proof–
  from assms(1) have Ord α by auto
```

```
  from assms have a ∈∘ (⋃∘ξ∈∘α. Vset ξ) and b ∈∘ (⋃∘ξ∈∘α. Vset ξ)
    by (simp-all add: SUP-vifunion[symmetric] Limit-Vfrom-eq)
  then obtain A B
    where a: a ∈∘ Vset A and A ∈∘ α and b: b ∈∘ Vset B and B ∈∘ α
    by blast
  moreover with assms have Ord A and Ord B by auto
  ultimately have A ∪∘ B ∈∘ α
    by (metis Ord-linear-le le-iff-sup sup.order-iff)
  then have Vset (A ∪∘ B) ∈∘ Vset α
    by (simp add: assms Limit-is-Ord Vset-in-mono)
  moreover from a b have set {a, b} ⊆∘ Vset (A ∪∘ B)
    by (simp add: Vfrom-sup vsubset-vdoubleton-leftI)
  ultimately show set {a, b} ∈∘ Vset α by (rule vsubset-in-VsetI[rotated 1])
qed

lemma vdoubleton-in-VsetD:
  assumes set {a, b} ∈∘ Vset α
  shows a ∈∘ Vset α and b ∈∘ Vset α
  using assms by (auto intro!: Vset-trans[of - ‹set {a, b}›])

lemma Ord-vdoubleton-in-Vset-succI:
  assumes Ord α and a ∈∘ Vset α and b ∈∘ Vset α
  shows set {a, b} ∈∘ Vset (succ α)
  by
    (
      meson
        assms Ord-Vset-in-Vset-succI vsubset-in-VsetI vsubset-vdoubleton-leftI
    )
```

Pairwise union.

```
lemma vunion-in-VsetI[intro]:
  assumes a ∈∘ Vset i and b ∈∘ Vset i
  shows a ∪∘ b ∈∘ Vset i
proof–
  define i′ where i′ = rank i
  note rank-conv = Vfrom-Ord-bundle[OF refl[of 0] refl[of i], folded i′-def]
  show ?thesis
    using rank-conv(2) assms unfolding rank-conv
  proof induction
    case (Limit α)
    from Limit have set {a, b} ∈∘ Vset α
      by (intro Limit-vdoubleton-in-VsetI; unfold SUP-vifunion[symmetric])
        simp-all
    then have ⋃∘(set {a, b}) ∈∘ Vset α by (blast intro: Limit.hyps)
    with Limit.hyps VUnion-vdoubleton have a ∪∘ b ∈∘ (⋃∘ξ∈∘α. Vset ξ)
      by (auto simp: Limit-Vfrom-eq)
    then show a ∪∘ b ∈∘ Vset (⋃∘ξ∈∘α. ξ)
      by (simp add: ‹Limit α› Limit-Vfrom-eq)
  qed (auto simp add: Vset-succ)
qed

lemma vunion-in-VsetD:
  assumes a ∪∘ b ∈∘ Vset α
  shows a ∈∘ Vset α and b ∈∘ Vset α
  using assms by (meson vsubset-in-VsetI inf-sup-ord(3,4))+
```

Pairwise intersection.

```
lemma vintersection-in-VsetI[intro]:
```

assumes $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
shows $a \cap_\circ b \in_\circ Vset\ \alpha$
using *assms* **by** (*meson vsubset-in-VsetI inf-sup-ord*(2))

Set difference.

lemma *vdiff-in-VsetI*[*intro*]:
assumes $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
shows $a -_\circ b \in_\circ Vset\ \alpha$
using *assms* **by** *auto*

vinsert.

lemma *vinsert-in-VsetI*[*intro*]:
assumes *Limit* α **and** $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
shows *vinsert a b* $\in_\circ Vset\ \alpha$
proof–
have *ab*: *vinsert a b* = *set* $\{a\} \cup_\circ b$ **by** *simp*
from *assms*(2) **have** *set* $\{a\} \in_\circ Vset\ \alpha$
by (*simp add*: *Limit-vsingleton-in-VsetI assms*(1))
from *this assms*(1,3) **show** *vinsert a b* $\in_\circ Vset\ \alpha$
unfolding *ab* **by** *blast*
qed

lemma *vinsert-in-Vset-succI*[*intro*]:
assumes *Ord* α **and** $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
shows *vinsert a b* $\in_\circ Vset\ (succ\ \alpha)$
using *assms* **by** *blast*

lemma *vinsert-in-Vset-succI'*[*intro*]:
assumes *Ord* α **and** $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ (succ\ \alpha)$
shows *vinsert a b* $\in_\circ Vset\ (succ\ \alpha)$
proof–
have *ab*: *vinsert a b* = *set* $\{a\} \cup_\circ b$ **by** *simp*
show *?thesis*
unfolding *ab* **by** (*intro vunion-in-VsetI Ord-vsingleton-in-Vset-succI assms*)
qed

lemma *vinsert-in-VsetD*:
assumes *vinsert a b* $\in_\circ Vset\ \alpha$
shows $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
using *assms Vset-trans* **by** *blast*+

lemma *Limit-insert-in-VsetI*:
assumes [*intro*]: *Limit* α
and [*simp*]: *small x*
and *set x* $\in_\circ Vset\ \alpha$
and [*intro*]: $a \in_\circ Vset\ \alpha$
shows *set* (*insert a x*) $\in_\circ Vset\ \alpha$
proof–
have *ax*: *set* (*insert a x*) = *vinsert a* (*set x*) **by** *auto*
from *assms* **show** *?thesis* **unfolding** *ax* **by** *auto*
qed

Pair.

lemma *Limit-vpair-in-VsetI*[*intro*]:
assumes *Limit* α **and** $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
shows $\langle a, b \rangle \in_\circ Vset\ \alpha$
using *assms Limit-vdoubleton-in-VsetI Limit-vsingleton-in-VsetI*

unfolding *vpair-def*
by *simp*

lemma *vpair-in-VsetD*[*intro*]:
assumes $\langle a,\ b\rangle \in_\circ Vset\ \alpha$
shows $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$
using *assms* **unfolding** *vpair-def* **by** (*meson vdoubleton-in-VsetD*)+

Cartesian product.

lemma *Limit-vtimes-in-VsetI*[*intro*]:
assumes *Limit* α **and** $A \in_\circ Vset\ \alpha$ **and** $B \in_\circ Vset\ \alpha$
shows $A \times_\circ B \in_\circ Vset\ \alpha$
proof–
from *assms*(1) **have** *Ord* α **by** *auto*
have $VPow\ (VPow\ (A \cup_\circ B)) \in_\circ Vset\ \alpha$
by (*simp add*: *assms Limit-VPow-in-VsetI Limit-is-Ord vunion-in-VsetI*)
from *assms*(1) *vsubset-in-VsetI*[*OF vtimes-vsubset-VPowVPow this*] **show** *?thesis*
by *auto*
qed

Binary relations.

lemma (**in** *vbrelation*) *vbrelation-Limit-in-VsetI*[*intro*]:
assumes *Limit* α **and** $\mathcal{D}_\circ\ r \in_\circ Vset\ \alpha$ **and** $\mathcal{R}_\circ\ r \in_\circ Vset\ \alpha$
shows $r \in_\circ Vset\ \alpha$
using *assms vdomain-vrange-vtimes* **by** *auto*

lemma
assumes $r \in_\circ Vset\ \alpha$
shows *vdomain-in-VsetI*: $\mathcal{D}_\circ\ r \in_\circ Vset\ \alpha$
and *vrange-in-VsetI*: $\mathcal{R}_\circ\ r \in_\circ Vset\ \alpha$
and *vfield-in-VsetI*: $\mathcal{F}_\circ\ r \in_\circ Vset\ \alpha$
proof–
from *assms* **have** $\bigcup_\circ r \in_\circ Vset\ \alpha$ **by** *auto*
with *assms*(1) **have** *r*: $\bigcup_\circ(\bigcup_\circ r) \in_\circ Vset\ \alpha$ **by** *blast*
from *r assms*(1) *vfield-vsubset-VUnion*2 **show** $\mathcal{F}_\circ\ r \in_\circ Vset\ \alpha$ **by** *auto*
from *r assms*(1) *vdomain-vsubset-VUnion*2 *vrange-vsubset-VUnion*2 **show**
$\mathcal{D}_\circ\ r \in_\circ Vset\ \alpha\ \mathcal{R}_\circ\ r \in_\circ Vset\ \alpha$
by *auto*
qed

lemma (**in** *vbrelation*) *vbrelation-Limit-vsubset-VsetI*:
assumes *Limit* α **and** $\mathcal{D}_\circ\ r \subseteq_\circ Vset\ \alpha$ **and** $\mathcal{R}_\circ\ r \subseteq_\circ Vset\ \alpha$
shows $r \subseteq_\circ Vset\ \alpha$
proof(*intro vsubsetI*)
fix *x* **assume** $x \in_\circ r$
moreover then obtain *a b* **where** *x-def*: $x = \langle a,\ b\rangle$ **by** (*elim vbrelation-vinE*)
ultimately have $a \in_\circ \mathcal{D}_\circ\ r$ **and** $b \in_\circ \mathcal{R}_\circ\ r$ **by** *auto*
with *assms* **show** $x \in_\circ Vset\ \alpha$ **unfolding** *x-def* **by** *auto*
qed

lemma
assumes $r \in_\circ Vset\ \alpha$
shows *fdomain-in-VsetI*: $\mathcal{D}_\bullet\ r \in_\circ Vset\ \alpha$
and *frange-in-VsetI*: $\mathcal{R}_\bullet\ r \in_\circ Vset\ \alpha$
and *ffield-in-VsetI*: $\mathcal{F}_\bullet\ r \in_\circ Vset\ \alpha$
proof–
from *assms* **have** $\bigcup_\circ r \in_\circ Vset\ \alpha$ **by** *auto*
with *assms* **have** *r*: $\bigcup_\circ(\bigcup_\circ(\bigcup_\circ r)) \in_\circ Vset\ \alpha$ **by** *blast*

```
  from r assms(1) fdomain-vsubset-VUnion2 frange-vsubset-VUnion2 show
    𝒟• r ∈∘ Vset α ℛ• r ∈∘ Vset α
    by auto
  from r assms(1) ffield-vsubset-VUnion2 show ℱ• r ∈∘ Vset α by auto
qed

lemma (in vsv) vsv-Limit-vrange-in-VsetI[intro]:
  assumes Limit α and ℛ∘ r ⊆∘ Vset α and vfinite (𝒟∘ r)
  shows ℛ∘ r ∈∘ Vset α
  using assms(3,1,2) vsv-axioms
proof(induction ‹𝒟∘ r› arbitrary: r rule: vfinite-induct)
  case vempty
  interpret r': vsv r by (rule vempty(4))
  from vempty(1) r'.vlrestriction-vdomain have r = 0 by simp
  from Vset-in-mono vempty.prems(1) show ?case
    unfolding ‹r = 0› by (auto simp: Limit-def)
next
  case (vinsert x F)
  interpret r': vsv r by (rule vinsert(7))
  have RrF-Rr: ℛ∘ (r ↾ₗ∘ F) ⊆∘ ℛ∘ r by auto
  have F-DrF: F = 𝒟∘ (r ↾ₗ∘ F)
    unfolding vdomain-vlrestriction vinsert(4)[symmetric] by auto
  moreover note assms(1)
  moreover from RrF-Rr vinsert(6) have ℛ∘ (r ↾ₗ∘ F) ⊆∘ Vset α by auto
  moreover have vsv (r ↾ₗ∘ F) by simp
  ultimately have RrF-Vα: ℛ∘ (r ↾ₗ∘ F) ∈∘ Vset α by (rule vinsert(3))
  have ℛ∘ r = vinsert (r(|x|)) (ℛ∘ (r ↾ₗ∘ F))
  proof(intro vsubset-antisym vsubsetI)
    fix b assume b ∈∘ ℛ∘ r
    then obtain a where a ∈∘ 𝒟∘ r and b-def: b = r(|a|) by force
    with vinsert.hyps(4) have a = x ∨ a ∈∘ F by auto
    with ‹a ∈∘ 𝒟∘ r› show b ∈∘ vinsert (r(|x|)) (ℛ∘ (r ↾ₗ∘ F))
      unfolding b-def by (blast dest: r'.vsv-vimageI1)
  next
    fix b assume b ∈∘ vinsert (r(|x|)) (ℛ∘ (r ↾ₗ∘ F))
    with RrF-Rr r'.vsv-axioms vinsert.hyps(4) show b ∈∘ ℛ∘ r by auto
  qed
  moreover with vinsert.prems(2) have r(|x|) ∈∘ Vset α by auto
  moreover have ℛ∘ (r ↾ₗ∘ F) ∈∘ Vset α by (blast intro: RrF-Vα)
  ultimately show ℛ∘ r ∈∘ Vset α
    by (simp add: vinsert.prems(1) vinsert-in-VsetI)
qed

lemma (in vsv) vsv-Limit-vsv-in-VsetI[intro]:
  assumes Limit α
    and 𝒟∘ r ∈∘ Vset α
    and ℛ∘ r ⊆∘ Vset α
    and vfinite (𝒟∘ r)
  shows r ∈∘ Vset α
  by (simp add: assms vsv-Limit-vrange-in-VsetI vbrelation-Limit-in-VsetI)

lemma Limit-vcomp-in-VsetI:
  assumes Limit α and r ∈∘ Vset α and s ∈∘ Vset α
  shows r ∘∘ s ∈∘ Vset α
proof(rule vbrelation.vbrelation-Limit-in-VsetI; (intro assms(1))?)
  show vbrelation (r ∘∘ s) by auto
  have 𝒟∘ (r ∘∘ s) ⊆∘ 𝒟∘ s by auto
  with assms(3) show 𝒟∘ (r ∘∘ s) ∈∘ Vset α
```

by (*auto simp*: *vdomain-in-VsetI vsubset-in-VsetI*)
have $\mathcal{R}_\circ\ (r \circ_\circ s) \subseteq_\circ \mathcal{R}_\circ\ r$ **by** *auto*
with *assms*(2) **show** $\mathcal{R}_\circ\ (r \circ_\circ s) \in_\circ Vset\ \alpha$
by (*auto simp*: *vrange-in-VsetI vsubset-in-VsetI*)
qed

Operations on indexed families of sets.

lemma *Limit-vifintersection-in-VsetI*:
assumes *Limit* α **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow A\ i \in_\circ Vset\ \alpha$ **and** *vfinite I*
shows $(\bigcap_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
proof–
from *assms*(2) **have** *range*: $\mathcal{R}_\circ\ (\lambda i \in_\circ I.\ A\ i) \subseteq_\circ Vset\ \alpha$ **by** *auto*
from *assms*(1) *range assms*(3) **have** $\mathcal{R}_\circ\ (\lambda i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
by (*rule rel-VLambda.vsv-Limit-vrange-in-VsetI*[*unfolded vdomain-VLambda*])
then have $(\lambda i \in_\circ I.\ A\ i)$ ‘$_\circ$ $I \in_\circ Vset\ \alpha$
by (*simp add*: *vimage-VLambda-vrange-rep*)
then show $(\bigcap_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$ **by** *auto*
qed

lemma *Limit-vifunion-in-VsetI*:
assumes *Limit* α **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow A\ i \in_\circ Vset\ \alpha$ **and** *vfinite I*
shows $(\bigcup_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
proof–
from *assms*(2) **have** *range*: $\mathcal{R}_\circ\ (\lambda i \in_\circ I.\ A\ i) \subseteq_\circ Vset\ \alpha$ **by** *auto*
from *assms*(1) *range assms*(3) **have** $\mathcal{R}_\circ\ (\lambda i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
by (*rule rel-VLambda.vsv-Limit-vrange-in-VsetI*[*unfolded vdomain-VLambda*])
then have $(\lambda i \in_\circ I.\ A\ i)$ ‘$_\circ$ $I \in_\circ Vset\ \alpha$
by (*simp add*: *vimage-VLambda-vrange-rep*)
then show $(\bigcup_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$ **by** *auto*
qed

lemma *Limit-vifunion-in-Vset-if-VLambda-in-VsetI*:
assumes *Limit* α **and** *VLambda I A* $\in_\circ Vset\ \alpha$
shows $(\bigcup_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
proof–
from *assms*(2) **have** $\mathcal{R}_\circ\ (\lambda i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
by (*simp add*: *vrange-in-VsetI*)
then have $(\lambda i \in_\circ I.\ A\ i)$ ‘$_\circ$ $I \in_\circ Vset\ \alpha$
by (*simp add*: *vimage-VLambda-vrange-rep*)
then show $(\bigcup_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$ **by** *auto*
qed

lemma *Limit-vproduct-in-VsetI*:
assumes *Limit* α
and $I \in_\circ Vset\ \alpha$
and $\bigwedge i.\ i \in_\circ I \Longrightarrow A\ i \in_\circ Vset\ \alpha$
and *vfinite I*
shows $(\prod_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
proof–
have $(\bigcup_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$
by (*rule Limit-vifunion-in-VsetI*) (*simp-all add*: *assms*(1,3,4))
with *assms* **have** $I \times_\circ (\bigcup_\circ i \in_\circ I.\ A\ i) \in_\circ Vset\ \alpha$ **by** *auto*
with *assms*(1) **have** $VPow\ (I \times_\circ (\bigcup_\circ i \in_\circ I.\ A\ i)) \in_\circ Vset\ \alpha$ **by** *auto*
from *vsubset-in-VsetI*[*OF vproduct-vsubset-VPow*[*of I A*] *this*] **show** *?thesis*
by *simp*
qed

lemma *Limit-vproduct-in-Vset-if-VLambda-in-VsetI*:

```
  assumes Limit α and VLambda I A ∈∘ Vset α
  shows (∏∘ i∈∘ I. A i) ∈∘ Vset α
proof–
  have (⋃∘ i∈∘ I. A i) ∈∘ Vset α
    by (rule Limit-vifunion-in-Vset-if-VLambda-in-VsetI)
      (simp-all add: assms)
  moreover from assms(2) have I ∈∘ Vset α
    by (metis vdomain-VLambda vdomain-in-VsetI)
  ultimately have I ×∘ (⋃∘ i∈∘ I. A i) ∈∘ Vset α
    using assms by auto
  with assms(1) have VPow (I ×∘ (⋃∘ i∈∘ I. A i)) ∈∘ Vset α by auto
  from vsubset-in-VsetI[OF vproduct-vsubset-VPow[of I A] this] show ?thesis
    by simp
qed

lemma Limit-vdunion-in-Vset-if-VLambda-in-VsetI:
  assumes Limit α and VLambda I A ∈∘ Vset α
  shows (∐∘ i∈∘ I. A i) ∈∘ Vset α
proof–
  interpret vsv ‹VLambda I A› by auto
  from assms have 𝒟∘ (VLambda I A) ∈∘ Vset α
    by (fastforce intro!: vdomain-in-VsetI)
  then have I: I ∈∘ Vset α by simp
  have (∐∘ i∈∘ I. A i) ⊆∘ I ×∘ (⋃∘(ℛ∘ (VLambda I A))) by force
  moreover have I ×∘ (⋃∘(ℛ∘ (VLambda I A))) ∈∘ Vset α
    by (intro Limit-vtimes-in-VsetI assms I VUnion-in-VsetI vrange-in-VsetI)
  ultimately show (∐∘ i∈∘ I. A i) ∈∘ Vset α by auto
qed

lemma vrange-vprojection-in-VsetI:
  assumes Limit α
    and A ∈∘ Vset α
    and ⋀f. f ∈∘ A ⟹ vsv f
    and ⋀f. f ∈∘ A ⟹ x ∈∘ 𝒟∘ f
  shows ℛ∘ (λf∈∘A. f(|x|)) ∈∘ Vset α
proof–
  have ℛ∘ (λf∈∘A. f(|x|)) ⊆∘ ⋃∘(⋃∘(⋃∘A))
  proof(intro vsubsetI)
    fix y assume y ∈∘ ℛ∘ (λf∈∘A. f(|x|))
    then obtain f where f: f ∈∘ A and y-def: y = f(|x|) by auto
    from f have vsv f and x ∈∘ 𝒟∘ f by (auto intro: assms(3,4))+
    with y-def have xy: ⟨x, y⟩ ∈∘ f by auto
    show y ∈∘ ⋃∘(⋃∘(⋃∘A))
    proof(intro VUnionI)
      show f ∈∘ A by (rule f)
      show ⟨x, y⟩ ∈∘ f by (rule xy)
      show set {x, y} ∈∘ ⟨x, y⟩ unfolding vpair-def by simp
    qed auto
  qed
  moreover from assms(1,2) have ⋃∘(⋃∘(⋃∘A)) ∈∘ Vset α
    by (intro VUnion-in-VsetI)
  ultimately show ?thesis by auto
qed

lemma Limit-vcpower-in-VsetI:
  assumes Limit α and n ∈∘ Vset α and A ∈∘ Vset α and vfinite n
  shows A ^× n ∈∘ Vset α
  using assms Limit-vproduct-in-VsetI unfolding vcpower-def by auto
```

Finite sets.

lemma *Limit-vfinite-in-VsetI*:
assumes *Limit* α **and** $A \subseteq_\circ$ *Vset* α **and** *vfinite A*
shows $A \in_\circ$ *Vset* α
proof–
from *assms*(3) **obtain** n **where** n: $n \in_\circ \omega$ **and** $n \approx_\circ A$ **by** *clarsimp*
then obtain f **where** f: *v11* f **and** dr: $\mathcal{D}_\circ\ f = n\ \mathcal{R}_\circ\ f = A$ **by** *auto*
interpret f: *v11* f **by** (*rule f*)
from n **have** n: *vfinite n* **by** *auto*
show *?thesis*
by (*rule f.vsv-Limit-vrange-in-VsetI*[*simplified dr*, *OF assms*(1,2) n])
qed

Ordinal numbers.

lemma *Limit-omega-in-VsetI*:
assumes *Limit* α
shows $a_\mathbb{N} \in_\circ$ *Vset* α
proof–
from *assms* **have** $\alpha \subseteq_\circ$ *Vset* α **by** *force*
moreover have $\omega \subseteq_\circ \alpha$ **by** (*simp add*: *assms omega-le-Limit*)
moreover have $a_\mathbb{N} \in_\circ \omega$ **by** *simp*
ultimately show $a_\mathbb{N} \in_\circ$ *Vset* α **by** *auto*
qed

lemma *Limit-succ-in-VsetI*:
assumes *Limit* α **and** $a \in_\circ$ *Vset* α
shows *succ a* $\in_\circ$ *Vset* α
by (*simp add*: *assms succ-def vinsert-in-VsetI*)

Sequences.

lemma (**in** *vfsequence*) *vfsequence-Limit-vcons-in-VsetI*:
assumes *Limit* α **and** $x \in_\circ$ *Vset* α **and** $xs \in_\circ$ *Vset* α
shows *vcons xs x* $\in_\circ$ *Vset* α
unfolding *vcons-def*
proof(*intro vinsert-in-VsetI Limit-vpair-in-VsetI assms*)
show *vcard xs* $\in_\circ$ *Vset* α
by (*metis assms*(3) *vdomain-in-VsetI vfsequence-vdomain*)
qed

ftimes.

lemma *Limit-ftimes-in-VsetI*:
assumes *Limit* α **and** $A \in_\circ$ *Vset* α **and** $B \in_\circ$ *Vset* α
shows $A \times_\bullet B \in_\circ$ *Vset* α
unfolding *ftimes-def*
proof(*rule Limit-vproduct-in-VsetI*)
from *assms*(1) **show** $2_\mathbb{N} \in_\circ$ *Vset* α **by** (*meson Limit-omega-in-VsetI*)
fix i **assume** $i \in_\circ 2_\mathbb{N}$
with *assms*(2,3) **show** ($i = 0$ *?* $A : B$) $\in_\circ$ *Vset* α **by** *simp*
qed (*auto simp*: *assms*(1))

Auxiliary results.

lemma *vempty-in-Vset-succ*[*simp*, *intro*]: $0 \in_\circ$ *Vfrom a* (*succ b*)
unfolding *Vfrom-succ* **by** *force*

lemma *Limit-vid-on-in-Vset*:
assumes *Limit* α **and** $A \in_\circ$ *Vset* α
shows *vid-on A* $\in_\circ$ *Vset* α

```
  by
    (
      rule vbrelation.vbrelation-Limit-in-VsetI
        [
          OF vbrelation-vid-on assms(1) ,
          unfolded vdomain-vid-on vrange-vid-on, OF assms(2,2)
        ]
    )

lemma Ord-vpair-in-Vset-succI[intro]:
  assumes Ord α and a ∈ₒ Vset α and b ∈ₒ Vset α
  shows ⟨a, b⟩ ∈ₒ Vset (succ (succ α))
  unfolding vpair-def
proof–
  have aab: set {set {a}, set {a, b}} = vinsert (set {a}) (set {set {a}, set {a, b}})
    by auto
  show set {set {a}, set {a, b}} ∈ₒ Vset (succ (succ α))
    unfolding aab
    by
      (
        intro
          assms
          vinsert-in-Vset-succI′
          Ord-vsingleton-in-Vset-succI
          Ord-vdoubleton-in-Vset-succI
          Ord-succ
      )
qed

lemma Limit-vifunion-vsubset-VsetI:
  assumes Limit α and ⋀i. i ∈ₒ I ⟹ A i ∈ₒ Vset α
  shows (⋃ₒi∈ₒI. A i) ⊆ₒ Vset α
proof(intro vsubsetI)
  fix x assume x ∈ₒ (⋃ₒi∈ₒI. A i)
  then obtain i where i: i ∈ₒ I and x ∈ₒ A i by auto
  with assms(1) assms(2)[OF i] show x ∈ₒ Vset α by auto
qed

lemma Limit-vproduct-vsubset-Vset-succI:
  assumes Limit α and I ∈ₒ Vset α and ⋀i. i ∈ₒ I ⟹ A i ⊆ₒ Vset α
  shows (∏ₒi∈ₒI. A i) ⊆ₒ Vset (succ α)
proof(intro vsubsetI)
  fix a assume prems: a ∈ₒ (∏ₒi∈ₒI. A i)
  note a = vproductD[OF prems]
  interpret vsv a by (rule a(1))
  from prems have ℛₒ a ⊆ₒ (⋃ₒi∈ₒI. A i) by (rule vproduct-vrange)
  moreover have (⋃ₒi∈ₒI. A i) ⊆ₒ Vset α by (intro vifunion-least assms(3))
  ultimately have ℛₒ a ⊆ₒ Vset α by auto
  moreover from assms(2) prems have 𝒟ₒ a ⊆ₒ Vset α unfolding a(2) by auto
  ultimately have a ⊆ₒ Vset α
    by (intro assms(1) vbrelation-Limit-vsubset-VsetI)
  with assms(1) show a ∈ₒ Vset (succ α)
    by (simp add: Limit-is-Ord Ord-vsubset-in-Vset-succI)
qed

lemma Limit-vproduct-vsubset-Vset-succI′:
  assumes Limit α and I ∈ₒ Vset α and ⋀i. i ∈ₒ I ⟹ A i ∈ₒ Vset α
  shows (∏ₒi∈ₒI. A i) ⊆ₒ Vset (succ α)
```

```
proof–
  have A i ⊆∘ Vset α if i ∈∘ I for i
    by (simp add: Vset-trans vsubsetI assms(3) that)
  from assms(1,2) this show ?thesis by (rule Limit-vproduct-vsubset-Vset-succI)
qed

lemma (in vfsequence) vfsequence-Ord-vcons-in-Vset-succI:
  assumes Ord α
    and ω ∈∘ α
    and x ∈∘ Vset α
    and xs ∈∘ Vset (succ (succ (succ α)))
  shows vcons xs x ∈∘ Vset (succ (succ (succ α)))
  unfolding vcons-def
proof(intro vinsert-in-Vset-succI' Ord-succ Ord-vpair-in-Vset-succI assms)
  have vcard xs = 𝒟∘ xs by (simp add: vfsequence-vdomain)
  from assms(1,2) vfsequence-vdomain-in-omega show vcard xs ∈∘ Vset α
    unfolding vfsequence-vdomain[symmetric]
    by (meson Ord-in-in-VsetI Vset-trans)
qed

lemma Limit-VUnion-vdomain-in-VsetI:
  assumes Limit α and Q ∈∘ Vset α
  shows (⋃∘r∈∘Q. 𝒟∘ r) ∈∘ Vset α
proof–
  have (⋃∘r∈∘Q. 𝒟∘ r) ⊆∘ ⋃∘(⋃∘(⋃∘Q))
  proof(intro vsubsetI)
    fix a assume a ∈∘ (⋃∘r∈∘Q. 𝒟∘ r)
    then obtain r where r: r ∈∘ Q and a ∈∘ 𝒟∘ r by auto
    with assms obtain b where ab: ⟨a, b⟩ ∈∘ r by auto
    show a ∈∘ ⋃∘(⋃∘(⋃∘Q))
    proof(intro VUnionI)
      show r ∈∘ Q by (rule r)
      show ⟨a, b⟩ ∈∘ r by (rule ab)
      show set {a, b} ∈∘ ⟨a, b⟩ unfolding vpair-def by simp
    qed auto
  qed
  moreover from assms(2) have ⋃∘(⋃∘(⋃∘Q)) ∈∘ Vset α
    by (blast dest!: VUnion-in-VsetI)
  ultimately show ?thesis using assms(1) by (auto simp: vsubset-in-VsetI)
qed

lemma Limit-VUnion-vrange-in-VsetI:
  assumes Limit α and Q ∈∘ Vset α
  shows (⋃∘r∈∘Q. ℛ∘ r) ∈∘ Vset α
proof–
  have (⋃∘r∈∘Q. ℛ∘ r) ⊆∘ ⋃∘(⋃∘(⋃∘Q))
  proof(intro vsubsetI)
    fix b assume b ∈∘ (⋃∘r∈∘Q. ℛ∘ r)
    then obtain r where r: r ∈∘ Q and b ∈∘ ℛ∘ r by auto
    with assms obtain a where ab: ⟨a, b⟩ ∈∘ r by auto
    show b ∈∘ ⋃∘(⋃∘(⋃∘Q))
    proof(intro VUnionI)
      show r ∈∘ Q by (rule r)
      show ⟨a, b⟩ ∈∘ r by (rule ab)
      show set {a, b} ∈∘ ⟨a, b⟩ unfolding vpair-def by simp
    qed auto
  qed
  moreover from assms(2) have ⋃∘(⋃∘(⋃∘Q)) ∈∘ Vset α
```

```
    by (blast dest!: VUnion-in-VsetI)
  ultimately show ?thesis using assms(1) by (auto simp: vsubset-in-VsetI)
qed
```

2.11.4 Axioms for *Vset* α

The subsection demonstrates that the axioms of ZFC except for the Axiom Schema of Replacement hold in *Vset* α for any limit ordinal α such that $\omega \in_\circ \alpha$[1].

```
locale 𝒵 =
  fixes α
  assumes Limit-α[intro, simp]: Limit α
    and omega-in-α[intro, simp]: ω ∈∘ α
begin

lemmas [intro] = 𝒵-axioms

lemma vempty-Z-def: 0 = set {x. x ≠ x} by auto

lemma vempty-is-zet[intro, simp]: 0 ∈∘ Vset α
  using Vset-in-mono omega-in-α by auto

lemma Axiom-of-Extensionality:
  assumes a ∈∘ Vset α and x = y and x ∈∘ a
  shows y ∈∘ a and x ∈∘ Vset α and y ∈∘ Vset α
  using assms by (simp-all add: Vset-trans)

lemma Axiom-of-Pairing:
  assumes a ∈∘ Vset α and b ∈∘ Vset α
  shows set {a, b} ∈∘ Vset α
  using assms by (simp add: Limit-vdoubleton-in-VsetI)

lemma Axiom-of-Unions:
  assumes a ∈∘ Vset α
  shows ⋃∘a ∈∘ Vset α
  using assms by (simp add: VUnion-in-VsetI)

lemma Axiom-of-Powers:
  assumes a ∈∘ Vset α
  shows VPow a ∈∘ Vset α
  using assms by (simp add: Limit-VPow-in-VsetI)

lemma Axiom-of-Regularity:
  assumes a ≠ 0 and a ∈∘ Vset α
  obtains x where x ∈∘ a and x ∩∘ a = 0
  using assms by (auto dest: trad-foundation)

lemma Axiom-of-Infinity: ω ∈∘ Vset α
  using Limit-is-Ord by (auto simp: Ord-iff-rank Ord-VsetI OrdmemD)

lemma Axiom-of-Choice:
  assumes A ∈∘ Vset α
  obtains f where f ∈∘ Vset α and ⋀x. x ∈∘ A ⟹ x ≠ 0 ⟹ f(|x|) ∈∘ x
proof–
  define f where f = (λx∈∘A. (SOME a. a ∈∘ x ∨ (x = 0 ∧ a = 0)))
  interpret vsv f unfolding f-def by auto
```

[1]The presentation of the axioms is loosely based on the statement of the axioms of ZFC in Chapters 1-11 in [59].

```
  have A-def: A = 𝒟∘ f unfolding f-def by simp
  have Rf: ℛ∘ f ⊆∘ vinsert 0 (⋃∘A)
  proof(rule vsubsetI)
    fix y assume y ∈∘ ℛ∘ f
    then obtain x where x ∈∘ A and y = f(|x|)
      unfolding A-def by (blast dest: vrange-atD)
    then have y-def: y = (SOME a. a ∈∘ x ∨ x = 0 ∧ a = 0)
      unfolding f-def unfolding A-def by simp
    have y = 0 ∨ y ∈∘ x
    proof(cases ‹x = 0›)
      case False then show ?thesis
        unfolding y-def by (metis (mono-tags, lifting) verit-sko-ex′ vemptyE)
    qed (simp add: y-def)
    with ‹x ∈∘ A› show y ∈∘ vinsert 0 (⋃∘A) by clarsimp
  qed
  from assms have ⋃∘A ∈∘ Vset α by (simp add: Axiom-of-Unions)
  with vempty-is-zet Limit-α have vinsert 0 (⋃∘A) ∈∘ Vset α by auto
  with Rf have ℛ∘ f ∈∘ Vset α by auto
  with Limit-α assms[unfolded A-def] have f ∈∘ Vset α by auto
  moreover have x ∈∘ A ⟹ x ≠ 0 ⟹ f(|x|) ∈∘ x for x
  proof–
    assume prems: x ∈∘ A x ≠ 0
    then have f(|x|) = (SOME a. a ∈∘ x ∨ (x = 0 ∧ a = 0))
      unfolding f-def by simp
    with prems(2) show f(|x|) ∈∘ x
      by (metis (mono-tags, lifting) someI-ex vemptyE)
  qed
  ultimately show ?thesis by (simp add: that)
qed

end
```

Trivial corollaries.

```
lemma (in 𝒵) Ord-α: Ord α by auto

lemma (in 𝒵) 𝒵-Vset-ω2-vsubset-Vset: Vset (ω + ω) ⊆∘ Vset α
  by (simp add: Vset-vsubset-mono omega2-vsubset-Limit)

lemma (in 𝒵) 𝒵-Limit-αω: Limit (α + ω) by (simp add: Limit-is-Ord)

lemma (in 𝒵) 𝒵-α-αω: α ∈∘ α + ω
  by (simp add: Limit-is-Ord Ord-mem-iff-lt)

lemma (in 𝒵) 𝒵-ω-αω: ω ∈∘ α + ω
  using add-le-cancel-left0 by blast

lemma 𝒵-ωω: 𝒵 (ω + ω)
  using ω-gt0 by (auto intro: 𝒵.intro simp: Ord-mem-iff-lt)

lemma (in 𝒵) in-omega-in-omega-plus[intro]:
  assumes a ∈∘ ω
  shows a ∈∘ Vset (α + ω)
proof–
  from assms have a ∈∘ Vset ω by auto
  moreover have Vset ω ∈∘ Vset (α + ω) by (simp add: Vset-in-mono 𝒵-ω-αω)
  ultimately show a ∈∘ Vset (α + ω) by auto
qed
```

lemma (in $\mathcal{Z}$**)** *ord-of-nat-in-Vset*[*simp*]: $a_{\mathbb{N}} \in_\circ Vset\ \alpha$ **by** *force*

vfsequences-on.

lemma (in $\mathcal{Z}$**)** *vfsequences-on-in-VsetI*:
assumes $X \in_\circ Vset\ \alpha$
shows *vfsequences-on* $X \in_\circ Vset\ \alpha$
proof–
from *vfsequences-on-subset-ω-set* **have** *vfsequences-on* $X \subseteq_\circ VPow\ (\omega \times_\circ X)$
by (*auto simp*: *less-eq-V-def*)
moreover have $VPow\ (\omega \times_\circ X) \in_\circ Vset\ \alpha$
by (*intro Limit-VPow-in-VsetI Limit-vtimes-in-VsetI assms Axiom-of-Infinity*)
auto
ultimately show *?thesis* **by** *auto*
qed

2.11.5 Existence of a disjoint subset in *Vset* α

definition *mk-doubleton* :: $V \Rightarrow V \Rightarrow V$
where *mk-doubleton* $X\ a$ = *set* $\{a, X\}$

definition *mk-doubleton-image* :: $V \Rightarrow V \Rightarrow V$
where *mk-doubleton-image* $X\ Y$ = *set* (*mk-doubleton* Y ‘ *elts* X)

lemma *inj-on-mk-doubleton*: *inj-on* (*mk-doubleton* X) (*elts* X)
proof
fix $a\ b$ **assume** *mk-doubleton* $X\ a$ = *mk-doubleton* $X\ b$
then have $\{a, X\} = \{b, X\}$ **unfolding** *mk-doubleton-def* **by** *auto*
then show $a = b$ **by** (*metis doubleton-eq-iff*)
qed

lemma *mk-doubleton-image-vsubset-veqpoll*:
assumes $X \subseteq_\circ Y$
shows *mk-doubleton-image* $X\ X \approx_\circ$ *mk-doubleton-image* $X\ Y$
unfolding *eqpoll-def*
proof(*intro exI*[*of* - ‹$\lambda A.$ *vinsert* Y ($A -_\circ$ *set* $\{X\}$)›] *bij-betw-imageI*)
show *inj-on* ($\lambda A.$ *vinsert* Y ($A -_\circ$ *set* $\{X\}$)) (*elts* (*mk-doubleton-image* $X\ X$))
unfolding *mk-doubleton-image-def*
proof(*intro inj-onI*)
fix $y\ y'$ **assume** *prems*:
$y \in_\circ$ *set* (*mk-doubleton* X ‘ *elts* X)
$y' \in_\circ$ *set* (*mk-doubleton* X ‘ *elts* X)
vinsert Y ($y -_\circ$ *set* $\{X\}$) = *vinsert* Y ($y' -_\circ$ *set* $\{X\}$)
then obtain $x\ x'$
where $x \in_\circ X$
and $x' \in_\circ X$
and *y-def*: y = *set* $\{x, X\}$
and *y′-def*: y' = *set* $\{x', X\}$
by (*clarsimp simp*: *mk-doubleton-def*)
with *assms* **have** *xX-X*: *set* $\{x, X\} -_\circ$ *set* $\{X\}$ = *set* $\{x\}$
and *x′X-X*: *set* $\{x', X\} -_\circ$ *set* $\{X\}$ = *set* $\{x'\}$
by *fastforce+*
from *prems*(3)[*unfolded y-def y′-def*] **have** *set* $\{x, Y\}$ = *set* $\{x', Y\}$
unfolding *xX-X x′X-X* **by** *auto*
then have $x = x'$ **by** (*auto simp*: *doubleton-eq-iff*)
then show $y = y'$ **unfolding** *y-def y′-def* **by** *simp*
qed
show
($\lambda A.$ *vinsert* Y ($A -_\circ$ *set* $\{X\}$)) ‘ (*elts* (*mk-doubleton-image* $X\ X$)) =

```
    (elts (mk-doubleton-image X Y))
  proof(intro subset-antisym subsetI)
    fix z
    assume prems:
      z ∈ (λA. vinsert Y (A −∘ set {X})) ‘ (elts (mk-doubleton-image X X))
    then obtain y
      where y ∈∘ set (mk-doubleton X ‘ elts X)
        and z-def: z = vinsert Y (y −∘ set {X})
      unfolding mk-doubleton-image-def by auto
    then obtain x where xX: x ∈∘ X and y-def: y = set {x, X}
      unfolding mk-doubleton-def by clarsimp
    from xX have y-X: y −∘ set {X} = set {x} unfolding y-def by fastforce
    from z-def have z-def′: z = set {x, Y}
      unfolding y-X by (simp add: doubleton-eq-iff vinsert-vsingleton)
    from xX show z ∈∘ mk-doubleton-image X Y
      unfolding z-def′ mk-doubleton-def mk-doubleton-image-def by simp
  next
    fix z assume prems: z ∈∘ mk-doubleton-image X Y
    then obtain x where xX: x ∈∘ X and z-def: z = set {x, Y}
      unfolding mk-doubleton-def mk-doubleton-image-def by clarsimp
    from xX have xX-XX: set {x, X} ∈∘ set (mk-doubleton X ‘ elts X)
      unfolding mk-doubleton-def by simp
    from xX have xX-X: set {x, X} −∘ set {X} = set {x} by fastforce
    have z-def′: z = vinsert Y (set {x, X} −∘ set {X})
      unfolding xX-X z-def by auto
    with xX-XX show
      z ∈ (λA. vinsert Y (A −∘ set {X})) ‘ (elts (mk-doubleton-image X X))
      unfolding z-def′ mk-doubleton-image-def by simp
  qed
qed

lemma mk-doubleton-image-veqpoll:
  assumes X ⊆∘ Y
  shows X ≈∘ mk-doubleton-image X Y
proof–
  have X ≈∘ mk-doubleton-image X X
    unfolding mk-doubleton-image-def by (auto simp: inj-on-mk-doubleton)
  also have ... ≈ elts (mk-doubleton-image X Y)
    by (rule mk-doubleton-image-vsubset-veqpoll[OF assms])
  finally show X ≈∘ mk-doubleton-image X Y.
qed

lemma vdisjnt-mk-doubleton-image: vdisjnt (mk-doubleton-image X Y) Y
proof
  fix b assume prems: b ∈∘ Y b ∈∘ mk-doubleton-image X Y
  then obtain a where a ∈∘ X and set {a, Y} = b
    unfolding mk-doubleton-def mk-doubleton-image-def by clarsimp
  then have Y ∈∘ b by clarsimp
  with mem-not-sym show False by (simp add: prems)
qed

lemma Limit-mk-doubleton-image-vsubset-Vset:
  assumes Limit α and X ⊆∘ Y and Y ∈∘ Vset α
  shows mk-doubleton-image X Y ⊆∘ Vset α
proof(intro vsubsetI)
  fix b assume b ∈∘ mk-doubleton-image X Y
  then obtain a where b = mk-doubleton Y a and a ∈∘ X
    unfolding mk-doubleton-image-def by clarsimp
```

```
  with assms have b-def: b = set {a, Y} and aα: a ∈∘ Vset α
    by (auto simp: mk-doubleton-def)
  from this(2) assms show b ∈∘ Vset α
    unfolding b-def by (simp add: Limit-vdoubleton-in-VsetI)
qed

lemma Ord-mk-doubleton-image-vsubset-Vset-succ:
  assumes Ord α and X ⊆∘ Y and Y ∈∘ Vset α
  shows mk-doubleton-image X Y ⊆∘ Vset (succ α)
proof(intro vsubsetI)
  fix b assume b ∈∘ mk-doubleton-image X Y
  then obtain a where b = mk-doubleton Y a and a ∈∘ X
    unfolding mk-doubleton-image-def by clarsimp
  with assms have b-def: b = set {a, Y} and aα: a ∈∘ Vset α
    by (auto simp: mk-doubleton-def)
  from this(2) assms show b ∈∘ Vset (succ α)
    unfolding b-def by (simp add: Ord-vdoubleton-in-Vset-succI)
qed

lemma Limit-ex-eqpoll-vdisjnt:
  assumes Limit α and X ⊆∘ Y and Y ∈∘ Vset α
  obtains Z where X ≈∘ Z and vdisjnt Z Y and Z ⊆∘ Vset α
  using assms
  by (intro that[of ‹mk-doubleton-image X Y›])
    (
      simp-all add:
        mk-doubleton-image-veqpoll
        vdisjnt-mk-doubleton-image
        Limit-mk-doubleton-image-vsubset-Vset
    )

lemma Ord-ex-eqpoll-vdisjnt:
  assumes Ord α and X ⊆∘ Y and Y ∈∘ Vset α
  obtains Z where X ≈∘ Z and vdisjnt Z Y and Z ⊆∘ Vset (succ α)
  using assms
  by (intro that[of ‹mk-doubleton-image X Y›])
    (
      simp-all add:
        mk-doubleton-image-veqpoll
        vdisjnt-mk-doubleton-image
        Ord-mk-doubleton-image-vsubset-Vset-succ
    )
```

2.12 n-ary operation

2.12.1 Partial n-ary operation

locale *pnop* = *vsv f* **for** *A n f* :: *V* +
  assumes *pnop-n*: $n \in_\circ \omega$
    and *pnop-vdomain*: $\mathcal{D}_\circ(f) \subseteq_\circ A \mathbin{\widehat{\times}} n$
    and *pnop-vrange*: $\mathcal{R}_\circ(f) \subseteq_\circ A$

Rules.

lemma *pnopI*[*intro*]:
  assumes *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ(f) \subseteq_\circ A \mathbin{\widehat{\times}} n$
    and $\mathcal{R}_\circ(f) \subseteq_\circ A$
  shows *pnop A n f*
  using *assms* **unfolding** *pnop-def pnop-axioms-def* **by** *blast*

lemma *pnopD*[*dest*]:
  assumes *pnop A n f*
  shows *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ(f) \subseteq_\circ A \mathbin{\widehat{\times}} n$
    and $\mathcal{R}_\circ(f) \subseteq_\circ A$
  using *assms* **unfolding** *pnop-def pnop-axioms-def* **by** *blast*+

lemma *pnopE*[*elim*]:
  assumes *pnop A n f*
  obtains *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ(f) \subseteq_\circ A \mathbin{\widehat{\times}} n$
    and $\mathcal{R}_\circ(f) \subseteq_\circ A$
  using *assms* **by** *force*

2.12.2 Total n-ary operation

locale *nop* = *vsv f* **for** *A n f* :: *V* +
  assumes *nop-n*: $n \in_\circ \omega$
    and *nop-vdomain*: $\mathcal{D}_\circ(f) = A \mathbin{\widehat{\times}} n$
    and *nop-vrange*: $\mathcal{R}_\circ(f) \subseteq_\circ A$

sublocale *nop* $\subseteq$ *pnop A n f*
proof(*intro pnopI*)
  show *vsv f* **by** (*rule vsv-axioms*)
  show $n \in_\circ \omega$ **by** (*rule nop-n*)
  from *nop-vdomain* **show** $\mathcal{D}_\circ(f) \subseteq_\circ A \mathbin{\widehat{\times}} n$ **by** *simp*
  show $\mathcal{R}_\circ(f) \subseteq_\circ A$ **by** (*rule nop-vrange*)
qed

Rules.

lemma *nopI*[*intro*]:
  assumes *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ(f) = A \mathbin{\widehat{\times}} n$
    and $\mathcal{R}_\circ(f) \subseteq_\circ A$
  shows *nop A n f*
  using *assms* **unfolding** *nop-def nop-axioms-def* **by** *blast*

lemma *nopD*[*dest*]:
assumes *nop A n f*
shows *vsv f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$
and $\mathcal{R}_\circ\ f \subseteq_\circ A$
using *assms* **unfolding** *nop-def nop-axioms-def* **by** *blast+*

lemma *nopE*[*elim*]:
assumes *nop A n f*
obtains *vsv f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$
and $\mathcal{R}_\circ\ f \subseteq_\circ A$
using *assms* **by** *force*

2.12.3 Injective n-ary operation

locale *nop-v11* = *v11 f* **for** *A n f* :: *V* +
assumes *nop-v11-n*: $n \in_\circ \omega$
and *nop-v11-vdomain*: $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$
and *nop-v11-vrange*: $\mathcal{R}_\circ\ f \subseteq_\circ A$

sublocale *nop-v11* ⊆ *nop*
proof
show *vsv f* **by** (*rule vsv-axioms*)
show $n \in_\circ \omega$ **by** (*rule nop-v11-n*)
show $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$ **by** (*rule nop-v11-vdomain*)
show $\mathcal{R}_\circ\ f \subseteq_\circ A$ **by** (*rule nop-v11-vrange*)
qed

Rules.

lemma *nop-v11I*[*intro*]:
assumes *v11 f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$
and $\mathcal{R}_\circ\ f \subseteq_\circ A$
shows *nop-v11 A n f*
using *assms* **unfolding** *nop-v11-def nop-v11-axioms-def* **by** *blast*

lemma *nop-v11D*[*dest*]:
assumes *nop-v11 A n f*
shows *v11 f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$
and $\mathcal{R}_\circ\ f \subseteq_\circ A$
using *assms* **unfolding** *nop-v11-def nop-v11-axioms-def* **by** *blast+*

lemma *nop-v11E*[*elim*]:
assumes *nop-v11 A n f*
obtains *v11 f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A \mathbin{\widehat{\times}} n$
and $\mathcal{R}_\circ\ f \subseteq_\circ A$
using *assms* **by** *force*

2.12.4 Surjective n-ary operation

locale *nop-onto* = *vsv f* **for** *A n f* :: *V* +
  assumes *nop-onto-n*: $n \in_\circ \omega$
    and *nop-onto-vdomain*: $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$
    and *nop-onto-vrange*: $\mathcal{R}_\circ\ f = A$

sublocale *nop-onto* ⊆ *nop*
proof
  show *vsv f* **by** (*rule vsv-axioms*)
  show $n \in_\circ \omega$ **by** (*rule nop-onto-n*)
  show $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$ **by** (*rule nop-onto-vdomain*)
  show $\mathcal{R}_\circ\ f \subseteq_\circ A$ **by** (*simp add*: *nop-onto-vrange*)
qed

Rules.

lemma *nop-ontoI*[*intro*]:
  assumes *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$
    and $\mathcal{R}_\circ\ f = A$
  shows *nop-onto A n f*
  using *assms* **unfolding** *nop-onto-def nop-onto-axioms-def* **by** *blast*

lemma *nop-ontoD*[*dest*]:
  assumes *nop-onto A n f*
  shows *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$
    and $\mathcal{R}_\circ\ f = A$
  using *assms* **unfolding** *nop-onto-def nop-onto-axioms-def* **by** *auto*

lemma *nop-ontoE*[*elim*]:
  assumes *nop-onto A n f*
  obtains *vsv f*
    and $n \in_\circ \omega$
    and $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$
    and $\mathcal{R}_\circ\ f = A$
  using *assms* **by** *force*

2.12.5 Bijective n-ary operation

locale *nop-bij* = *v*11 *f* **for** *A n f* :: *V* +
  assumes *nop-bij-n*: $n \in_\circ \omega$
    and *nop-bij-vdomain*: $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$
    and *nop-bij-vrange*: $\mathcal{R}_\circ\ f = A$

sublocale *nop-bij* ⊆ *nop-v*11
proof
  show *v*11 *f* **by** (*rule v*11*-axioms*)
  show $n \in_\circ \omega$ **by** (*rule nop-bij-n*)
  show $\mathcal{D}_\circ\ f = A\ ^\frown_\times\ n$ **by** (*rule nop-bij-vdomain*)
  show $\mathcal{R}_\circ\ f \subseteq_\circ A$ **by** (*simp add*: *nop-bij-vrange*)
qed

sublocale *nop-bij* ⊆ *nop-onto*
proof
  show *vsv f* **by** (*rule vsv-axioms*)
  show $n \in_\circ \omega$ **by** (*rule nop-bij-n*)

show $\mathcal{D}_\circ\ f = A\ \widehat{\ }_\times\ n$ **by** (*rule nop-bij-vdomain*)
show $\mathcal{R}_\circ\ f = A$ **by** (*rule nop-bij-vrange*)
qed

Rules.

lemma *nop-bijI*[*intro*]:
assumes *v*11 *f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A\ \widehat{\ }_\times\ n$
and $\mathcal{R}_\circ\ f = A$
shows *nop-bij A n f*
using *assms* **unfolding** *nop-bij-def nop-bij-axioms-def* **by** *blast*

lemma *nop-bijD*[*dest*]:
assumes *nop-bij A n f*
shows *v*11 *f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A\ \widehat{\ }_\times\ n$
and $\mathcal{R}_\circ\ f = A$
using *assms* **unfolding** *nop-bij-def nop-bij-axioms-def* **by** *auto*

lemma *nop-bijE*[*elim*]:
assumes *nop-bij A n f*
obtains *v*11 *f*
and $n \in_\circ \omega$
and $\mathcal{D}_\circ\ f = A\ \widehat{\ }_\times\ n$
and $\mathcal{R}_\circ\ f = A$
using *assms* **by** *force*

2.12.6 Scalar

locale *scalar* =
fixes *A f*
assumes *scalar-nop*: *nop A* 0 *f*

sublocale *scalar* ⊆ *nop A* 0 *f*
rewrites *scalar-vdomain*[*simp*]: $A\ \widehat{\ }_\times\ 0 = set\ \{0\}$
by (*auto simp*: *scalar-nop*)

Rules.

lemmas *scalarI*[*intro*] = *scalar.intro*

lemma *scalarD*[*dest*]:
assumes *scalar A f*
shows *nop A* 0 *f*
using *assms* **unfolding** *scalar-def* **by** *auto*

lemma *scalarE*[*elim*]:
assumes *scalar A f*
obtains *nop A* 0 *f*
using *assms* **by** *auto*

2.12.7 Unary operation

locale *unop* = *nop A* ‹$1_\mathbb{N}$› *f* **for** *A f*

Rules.

lemmas *unopI*[*intro*] = *unop.intro*

lemma *unopD*[*dest*]:
assumes *unop A f*
shows *nop A* ($1_{\mathbb{N}}$) *f*
using *assms* **unfolding** *unop-def* **by** *auto*

lemma *unopE*[*elim*]:
assumes *unop A f*
obtains *nop A* ($1_{\mathbb{N}}$) *f*
using *assms* **by** *blast*

2.12.8 Injective unary operation

locale *unop-v*11 = *nop-v*11 *A* ‹$1_{\mathbb{N}}$› *f* **for** *A f*

sublocale *unop-v*11 ⊆ *unop A f* **by** (*intro unopI*) (*simp add*: *nop-axioms*)

Rules.

lemma *unop-v*11*I*[*intro*]:
assumes *nop-v*11 *A* ($1_{\mathbb{N}}$) *f*
shows *unop-v*11 *A f*
using *assms* **by** (*rule unop-v*11.*intro*)

lemma *unop-v*11*D*[*dest*]:
assumes *unop-v*11 *A f*
shows *nop-v*11 *A* ($1_{\mathbb{N}}$) *f*
using *assms* **by** (*rule unop-v*11.*axioms*)

lemma *unop-v*11*E*[*elim*]:
assumes *unop-v*11 *A f*
obtains *nop-v*11 *A* ($1_{\mathbb{N}}$) *f*
using *assms* **by** *blast*

2.12.9 Surjective unary operation

locale *unop-onto* = *nop-onto A* ‹$1_{\mathbb{N}}$› *f* **for** *A f*

sublocale *unop-onto* ⊆ *unop A f* **by** (*intro unopI*) (*simp add*: *nop-axioms*)

Rules.

lemma *unop-ontoI*[*intro*]:
assumes *nop-onto A* ($1_{\mathbb{N}}$) *f*
shows *unop-onto A f*
using *assms* **by** (*rule unop-onto.intro*)

lemma *unop-ontoD*[*dest*]:
assumes *unop-onto A f*
shows *nop-onto A* ($1_{\mathbb{N}}$) *f*
using *assms* **by** (*rule unop-onto.axioms*)

lemma *unop-ontoE*[*elim*]:
assumes *unop-onto A f*
obtains *nop-onto A* ($1_{\mathbb{N}}$) *f*
using *assms* **by** *blast*

lemma *unop-ontoI′*[*intro*]:
assumes *unop A f* **and** $A \subseteq_\circ \mathcal{R}_\circ f$
shows *unop-onto A f*

proof–
interpret *unop A f* **by** (*rule assms*(1))
from *assms*(2) *nop-vrange* **have** $A = \mathcal{R}_\circ\ f$ **by** *simp*
with *assms*(1) **show** *unop-onto A f* **by** *auto*
qed

2.12.10 Bijective unary operation

locale *unop-bij* = *nop-bij A* ‹$1_\mathbb{N}$› *f* **for** *A f*

sublocale *unop-bij* ⊆ *unop-v*11 *A f*
by (*intro unop-v*11*I*) (*simp add*: *nop-v*11*-axioms*)

sublocale *unop-bij* ⊆ *unop-onto A f*
by (*intro unop-ontoI*) (*simp add*: *nop-onto-axioms*)

Rules.

lemma *unop-bijI*[*intro*]:
assumes *nop-bij A* ($1_\mathbb{N}$) *f*
shows *unop-bij A f*
using *assms* **by** (*rule unop-bij.intro*)

lemma *unop-bijD*[*dest*]:
assumes *unop-bij A f*
shows *nop-bij A* ($1_\mathbb{N}$) *f*
using *assms* **by** (*rule unop-bij.axioms*)

lemma *unop-bijE*[*elim*]:
assumes *unop-bij A f*
obtains *nop-bij A* ($1_\mathbb{N}$) *f*
using *assms* **by** *blast*

lemma *unop-bijI'*[*intro*]:
assumes *unop-v*11 *A f* **and** $A \subseteq_\circ \mathcal{R}_\circ\ f$
shows *unop-bij A f*
proof–
interpret *unop-v*11 *A f* **by** (*rule assms*(1))
from *assms*(2) *nop-vrange* **have** $A = \mathcal{R}_\circ\ f$ **by** *simp*
with *assms*(1) **show** *unop-bij A f* **by** *auto*
qed

2.12.11 Partial binary operation

locale *pbinop* = *pnop A* ‹$2_\mathbb{N}$› *f* **for** *A f*

sublocale *pbinop* ⊆ *dom*: *fbrelation* ‹$\mathcal{D}_\circ\ f$›
proof
from *pnop-vdomain* **show** *fpairs* ($\mathcal{D}_\circ\ f$) = $\mathcal{D}_\circ\ f$
by (*intro vsubset-antisym vsubsetI*) *auto*
qed

Rules.

lemmas *pbinopI*[*intro*] = *pbinop.intro*

lemma *pbinopD*[*dest*]:
assumes *pbinop A f*
shows *pnop A* ($2_\mathbb{N}$) *f*
using *assms* **unfolding** *pbinop-def* **by** *auto*

lemma *pbinopE*[*elim*]:
assumes *pbinop A f*
obtains *pnop A* ($2_{\mathbb{N}}$) *f*
using *assms* **by** *auto*

Elementary properties.

lemma (**in** *pbinop*) *fbinop-vcard*:
assumes $x \in_{\circ} \mathcal{D}_{\circ}\ f$
shows *vcard x* = $2_{\mathbb{N}}$
proof–
from *assms dom.fbrelation-axioms* **obtain** *a b* **where** *x-def*: $x = [a, b]_{\circ}$ **by** *blast*
show *?thesis* **by** (*auto simp*: *x-def nat-omega-simps*)
qed

2.12.12 Total binary operation

locale *binop* = *nop A* ‹$2_{\mathbb{N}}$› *f* **for** *A f*

sublocale *binop* ⊆ *pbinop* **by** *unfold-locales*

Rules.

lemmas *binopI*[*intro*] = *binop.intro*

lemma *binopD*[*dest*]:
assumes *binop A f*
shows *nop A* ($2_{\mathbb{N}}$) *f*
using *assms* **unfolding** *binop-def* **by** *auto*

lemma *binopE*[*elim*]:
assumes *binop A f*
obtains *nop A* ($2_{\mathbb{N}}$) *f*
using *assms* **by** *auto*

Elementary properties.

lemma *binop-eqI*:
assumes *binop A g*
and *binop A f*
and $\bigwedge a\ b.\ [\![\ a \in_{\circ} A;\ b \in_{\circ} A\ ]\!] \Longrightarrow g(\!|a, b|\!)_{\bullet} = f(\!|a, b|\!)_{\bullet}$
shows *g* = *f*
proof–
interpret *g*: *binop A g* **by** (*rule assms*(1))
interpret *f*: *binop A f* **by** (*rule assms*(2))
show *?thesis*
proof
(
rule vsv-eqI;
(*intro g.vsv-axioms f.vsv-axioms*) *?*;
(*unfold g.nop-vdomain f.nop-vdomain*)
)
fix *ab* **assume** $ab \in_{\circ} A \mathbin{\widehat{\times}} 2_{\mathbb{N}}$
then obtain *a b* **where** *ab-def*: $ab = [a, b]_{\circ}$
and *a*: $a \in_{\circ} A$
and *b*: $b \in_{\circ} A$
by *auto*
show $g(\!|ab|\!) = f(\!|ab|\!)$ **unfolding** *ab-def* **by** (*rule assms*(3)[*OF a b*])
qed *simp*
qed

lemma (in *binop*) *binop-app-in-vrange*[*intro*]:
assumes $a \in_\circ A$ **and** $b \in_\circ A$
shows $f(\!|a, b|\!)_\bullet \in_\circ \mathcal{R}_\circ f$
proof–
from *assms* **have** $[a, b]_\circ \in_\circ A \mathbin{\widehat{\times}} 2_\mathbb{N}$ **by** (*auto simp*: *nat-omega-simps*)
then show *?thesis* **by** (*simp add*: *nop-vdomain vsv-vimageI2*)
qed

2.12.13 Injective binary operation

locale *binop-v11* = *nop-v11* A ‹$2_\mathbb{N}$› f **for** A f

sublocale *binop-v11* ⊆ *binop* A f **by** (*intro binopI*) (*simp add*: *nop-axioms*)

Rules.

lemma *binop-v11I*[*intro*]:
assumes *nop-v11* A ($2_\mathbb{N}$) f
shows *binop-v11* A f
using *assms* **by** (*rule binop-v11.intro*)

lemma *binop-v11D*[*dest*]:
assumes *binop-v11* A f
shows *nop-v11* A ($2_\mathbb{N}$) f
using *assms* **by** (*rule binop-v11.axioms*)

lemma *binop-v11E*[*elim*]:
assumes *binop-v11* A f
obtains *nop-v11* A ($2_\mathbb{N}$) f
using *assms* **by** *blast*

2.12.14 Surjective binary operation

locale *binop-onto* = *nop-onto* A ‹$2_\mathbb{N}$› f **for** A f

sublocale *binop-onto* ⊆ *binop* A f **by** (*intro binopI*) (*simp add*: *nop-axioms*)

Rules.

lemma *binop-ontoI*[*intro*]:
assumes *nop-onto* A ($2_\mathbb{N}$) f
shows *binop-onto* A f
using *assms* **by** (*rule binop-onto.intro*)

lemma *binop-ontoD*[*dest*]:
assumes *binop-onto* A f
shows *nop-onto* A ($2_\mathbb{N}$) f
using *assms* **by** (*rule binop-onto.axioms*)

lemma *binop-ontoE*[*elim*]:
assumes *binop-onto* A f
obtains *nop-onto* A ($2_\mathbb{N}$) f
using *assms* **by** *blast*

lemma *binop-ontoI′*[*intro*]:
assumes *binop* A f **and** $A \subseteq_\circ \mathcal{R}_\circ f$
shows *binop-onto* A f
proof–
interpret *binop* A f **by** (*rule assms*(1))

from *assms*(2) *nop-vrange* **have** $A = \mathcal{R}_\circ\ f$ **by** *simp*
with *assms*(1) **show** *binop-onto A f* **by** *auto*
qed

2.12.15 Bijective binary operation

locale *binop-bij* = *nop-bij A* ‹$2_\mathbb{N}$› *f* **for** *A f*

sublocale *binop-bij* ⊆ *binop-v*11 *A f*
by (*intro binop-v*11*I*) (*simp add*: *nop-v*11*-axioms*)

sublocale *binop-bij* ⊆ *binop-onto A f*
by (*intro binop-ontoI*) (*simp add*: *nop-onto-axioms*)

Rules.

lemma *binop-bijI*[*intro*]:
assumes *nop-bij A* ($2_\mathbb{N}$) *f*
shows *binop-bij A f*
using *assms* **by** (*rule binop-bij.intro*)

lemma *binop-bijD*[*dest*]:
assumes *binop-bij A f*
shows *nop-bij A* ($2_\mathbb{N}$) *f*
using *assms* **by** (*rule binop-bij.axioms*)

lemma *binop-bijE*[*elim*]:
assumes *binop-bij A f*
obtains *nop-bij A* ($2_\mathbb{N}$) *f*
using *assms* **by** *blast*

lemma *binop-bijI′*[*intro*]:
assumes *binop-v*11 *A f* **and** $A \subseteq_\circ \mathcal{R}_\circ\ f$
shows *binop-bij A f*
proof–
interpret *binop-v*11 *A f* **by** (*rule assms*(1))
from *assms*(2) *nop-vrange* **have** $A = \mathcal{R}_\circ\ f$ **by** *simp*
with *assms*(1) **show** *binop-bij A f* **by** *auto*
qed

2.12.16 Flip

definition *fflip* :: $V \Rightarrow V$
where *fflip f* = $(\lambda ab{\in_\circ}(\mathcal{D}_\circ\ f)^{-1}_\bullet.\ f(\!|ab(\!|1_\mathbb{N}|\!),\ ab(\!|0|\!)|\!)_\bullet)$

Elementary properties.

lemma *fflip-vempty*[*simp*]: *fflip* 0 = 0 **unfolding** *fflip-def* **by** *auto*

lemma *fflip-vsv*: *vsv* (*fflip f*)
by (*intro vsvI*) (*auto simp*: *fflip-def*)

lemma *vdomain-fflip*[*simp*]: $\mathcal{D}_\circ$ (*fflip f*) = $(\mathcal{D}_\circ\ f)^{-1}_\bullet$
unfolding *fflip-def* **by** *simp*

lemma (**in** *pbinop*) *vrange-fflip*: $\mathcal{R}_\circ$ (*fflip f*) = $\mathcal{R}_\circ\ f$
unfolding *fflip-def*
proof(*intro vsubset-antisym vsubsetI*)
fix *y* **assume** *y* $\in_\circ \mathcal{R}_\circ\ ((\lambda x{\in_\circ}(\mathcal{D}_\circ\ f)^{-1}_\bullet.\ f(\!|x(\!|1_\mathbb{N}|\!),\ x(\!|0|\!)|\!)_\bullet))$
then obtain *x* **where** *x* $\in_\circ (\mathcal{D}_\circ\ f)^{-1}_\bullet$ **and** *y-def*: *y* = $f(\!|x(\!|1_\mathbb{N}|\!),\ x(\!|0|\!)|\!)_\bullet$ **by** *fast*

```
  then obtain a b where x-def: x = [b, a]∘ by clarsimp
  have y-def': y = f(|a, b|)•
    unfolding y-def x-def by (simp add: nat-omega-simps)
  from x-def ‹x ∈∘ (𝒟∘ f)⁻¹•› have [a, b]∘ ∈∘ 𝒟∘ f by clarsimp
  then show y ∈∘ ℛ∘ f unfolding y-def' by (simp add: vsv-vimageI2)
next
  fix y assume y ∈∘ ℛ∘ f
  with vrange-atD obtain x where x: x ∈∘ 𝒟∘ f and y-def: y = f(|x|) by blast
  with dom.fbrelation obtain a b where x-def: x = [a, b]∘ by blast
  from x have ba: [b, a]∘ ∈∘ (𝒟∘ f)⁻¹• unfolding x-def by clarsimp
  then have y-def': y = f([[b, a]∘(|1ℕ|), [b, a]∘(|0|)]•
    unfolding y-def x-def by (auto simp: nat-omega-simps)
  then show y ∈∘ ℛ∘ ((λab∈∘(𝒟∘ f)⁻¹•. f(|ab(|1ℕ|), ab(|0|)|)•))
    unfolding y-def'
    by (metis (lifting) ba beta rel-VLambda.vsv-vimageI2 vdomain-VLambda)
qed

lemma fflip-app[simp]:
  assumes [a, b]∘ ∈∘ 𝒟∘ f
  shows fflip f(|b, a|)• = f(|a, b|)•
proof–
  from assms have [b, a]∘ ∈∘ 𝒟∘ (fflip f) by clarsimp
  then show fflip f(|b, a|)• = f(|a, b|)•
    by (simp add: fflip-def ord-of-nat-succ-vempty)
qed

lemma (in pbinop) pbinop-fflip-fflip: fflip (fflip f) = f
proof(rule vsv-eqI)
  show vsv (fflip (fflip f)) by (simp add: fflip-vsv)
  show vsv f by (rule vsv-axioms)
  show dom: 𝒟∘ (fflip (fflip f)) = 𝒟∘ f by simp
  fix x assume prems: x ∈∘ 𝒟∘ (fflip (fflip f))
  with dom dom.fbrelation-axioms obtain a b where x-def: x = [a, b]∘ by auto
  from prems show fflip (fflip f)(|x|) = f(|x|)
    unfolding x-def by (auto simp: fconverseI)
qed

lemma (in binop) pbinop-fflip-app[simp]:
  assumes a ∈∘ A and b ∈∘ A
  shows fflip f(|b, a|)• = f(|a, b|)•
proof–
  from assms have [a, b]∘ ∈∘ 𝒟∘ f
    unfolding nop-vdomain by (auto simp: nat-omega-simps)
  then show ?thesis by auto
qed

lemma fflip-vsingleton: fflip (set {⟨[a, b]∘, c⟩}) = set {⟨[b, a]∘, c⟩}
proof–
  have dom-lhs: 𝒟∘ (fflip (set {⟨[a, b]∘, c⟩})) = set {[b, a]∘}
    unfolding fflip-def by auto
  have dom-rhs: 𝒟∘ (set {⟨[a, b]∘, c⟩}) = set {[b, a]∘} by simp
  show ?thesis
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix q assume q ∈∘ set {[b, a]∘}
    then have q-def: q = [b, a]∘ by simp
    show fflip (set {⟨[a, b]∘, c⟩})(|q|) = set {⟨[b, a]∘, c⟩}(|q|)
      unfolding q-def by auto
  qed (auto simp: fflip-def)
```

qed

2.13 Construction of integer numbers, rational numbers and real numbers

2.13.1 Background

The set of real numbers $\mathbb{R}_\circ$ is defined in a way such that it agrees with the set of natural numbers ω. However, otherwise, real numbers are allowed to be arbitrary sets in $Vset\ (\omega + \omega)$.[2] Integer and rational numbers are exposed via canonical injections into the set of real numbers from the types *int* and *rat*, respectively. Lastly, common operations on the real, integer and rational numbers are defined and some of their main properties are exposed.

The primary reference for this section is the textbook *The Real Numbers and Real Analysis* by E. Bloch [14]. Nonetheless, it is not claimed that the exposition of the subject presented in this section is entirely congruent with the exposition in the aforementioned reference.

declare *One-nat-def*[*simp del*]

named-theorems *vnumber-simps*

lemmas [*vnumber-simps*] =
Collect-mem-eq Ball-def[*symmetric*] *Bex-def*[*symmetric*] *vsubset-eq*[*symmetric*]

Supplementary material for the evaluation of the upper bound of the cardinality of the continuum.

lemma *inj-image-ord-of-nat*: *inj* (*image ord-of-nat*)
by (*intro injI*) (*simp add*: *inj-image-eq-iff inj-ord-of-nat*)

lemma *vlepoll-VPow-omega-if-vreal-lepoll-real*:
assumes $x \lesssim$ (*UNIV*::*real set*)
shows *set* $x \lesssim_\circ$ *VPow* ω
proof–
note *assms*
also from *eqpoll-UNIV-real* **have** ... $\lesssim$ (*UNIV*::*nat set set*)
unfolding *Pow-UNIV* **by** (*simp add*: *eqpoll-imp-lepoll*)
also from *inj-image-ord-of-nat* **have** ... $\lesssim$ *Pow* (*elts* ω)
unfolding *lepoll-def* **by** *auto*
also from *down* **have** ... $\lesssim$ *elts* (*VPow* ω)
unfolding *lepoll-def*
by (*intro exI*[*of - set*] *conjI inj-onI*) (*auto simp*: *elts-VPow*)
finally show *set* $x \lesssim_\circ$ *VPow* ω **by** *simp*
qed

2.13.2 Real numbers

Definition

abbreviation *real* :: *nat* ⇒ *real*
where *real* ≡ *of-nat*

definition *nat-of-real* :: *real* ⇒ *nat*
where *nat-of-real* = *inv-into UNIV real*

definition *vreal-of-real-impl* :: *real* ⇒ *V*
where *vreal-of-real-impl* = (*SOME V-of*::*real*⇒*V*. *inj V-of*)

lemma *inj-vreal-of-real-impl*: *inj vreal-of-real-impl*
unfolding *vreal-of-real-impl-def*

[2] The idea itself is not new, e.g., see [26].

by (*metis embeddable-class.ex-inj verit-sko-ex'*)

lemma *inj-on-inv-vreal-of-real-impl*:
inj-on (*inv vreal-of-real-impl*) (*range vreal-of-real-impl*)
by (*intro inj-onI*) (*fastforce intro*: *inv-into-injective*)

lemma *range-vreal-of-real-impl-vlepoll-VPow-omega*:
set (*range vreal-of-real-impl*) $\lesssim_\circ$ *VPow* ω
proof–
have *range vreal-of-real-impl* $\lesssim$ (*UNIV*::*real set*)
unfolding *lepoll-def* **by** (*auto intro*: *inj-on-inv-vreal-of-real-impl*)
from *vlepoll-VPow-omega-if-vreal-lepoll-real*[*OF this*] **show** *?thesis* **.**
qed

definition *vreal-impl* :: *V*
where *vreal-impl* =
(
SOME y.
range vreal-of-real-impl $\approx$ *elts y* $\wedge$
vdisjnt y ω $\wedge$
$y \in_\circ$ *Vset* $(\omega + \omega)$
)

lemma *vreal-impl-eqpoll*: *range vreal-of-real-impl* $\approx$ *elts vreal-impl*
and *vreal-impl-vdisjnt*: *vdisjnt vreal-impl* ω
and *vreal-impl-in-Vset-ss-omega*: *vreal-impl* $\in_\circ$ *Vset* $(\omega + \omega)$
proof–
from *Ord-*ω **have** *VPow-in-Vset*: *VPow* $\omega \in_\circ$ *Vset* (*succ* (*succ* ω))
by (*intro Ord-VPow-in-Vset-succI*)
(*auto simp*: *less-TC-succ Ord-iff-rank VsetI*)
have [*simp*]: *small* (*range vreal-of-real-impl*) **by** *simp*
then obtain *x* **where** *x*: *range vreal-of-real-impl* = *elts x*
unfolding *small-iff* **by** *clarsimp*
from *range-vreal-of-real-impl-vlepoll-VPow-omega*[*unfolded x*] **have**
$x \lesssim_\circ$ *VPow* ω
by *simp*
then obtain *f* **where** *v11 f* **and** $\mathcal{D}_\circ\ f = x$ **and** $\mathcal{R}_\circ\ f \subseteq_\circ$ *VPow* ω **by** *auto*
moreover have *O*ω2: *Ord* (*succ* (*succ* ω)) **by** *auto*
ultimately have *x-Rf*: $x \approx_\circ \mathcal{R}_\circ\ f$ **and** $\mathcal{R}_\circ\ f \in_\circ$ *Vset* (*succ* (*succ* ω))
by (*auto intro*: *VPow-in-Vset*)
then have $\omega \cup_\circ \mathcal{R}_\circ\ f \in_\circ$ *Vset* (*succ* (*succ* ω)) **and** $\mathcal{R}_\circ\ f \subseteq_\circ \omega \cup_\circ \mathcal{R}_\circ\ f$
by (*auto simp*: *VPow-in-Vset VPow-in-Vset-revD vunion-in-VsetI*)
from *Ord-ex-eqpoll-vdisjnt*[*OF O*ω2 *this*(2,1)] **obtain** *z*
where *Rf-z*: $\mathcal{R}_\circ\ f \approx_\circ z$
and *vdisjnt z* ($\omega \cup_\circ \mathcal{R}_\circ\ f$)
and *z*: $z \subseteq_\circ$ *Vset* (*succ* (*succ* (*succ* ω)))
by *auto*
then have *vdisjnt-z*ω: *vdisjnt z* ω
and *z-ssss*ω: $z \in_\circ$ *Vset* (*succ* (*succ* (*succ* (*succ* ω))))
by
(
auto simp:
vdisjnt-vunion-right vsubset-in-VsetI Ord-succ Ord-Vset-in-Vset-succI
)
have *Limit* $(\omega + \omega)$ **by** *simp*
then have *succ* (*succ* (*succ* (*succ* ω))) $\in_\circ$ $\omega + \omega$
by (*metis Limit-def add.right-neutral add-mem-right-cancel Limit-omega*)
then have *Vset* (*succ* (*succ* (*succ* (*succ* ω)))) $\in_\circ$ *Vset* $(\omega + \omega)$

```
    by (simp add: Vset-in-mono)
  with z z-ssssω have z ∈∘ Vset (ω + ω) by auto
  moreover from x-Rf Rf-z have range vreal-of-real-impl ≈ elts z
    unfolding x by (auto intro: eqpoll-trans)
  ultimately show range vreal-of-real-impl ≈ elts vreal-impl
    and vdisjnt vreal-impl ω
    and vreal-impl ∈∘ Vset (ω + ω)
    using vdisjnt-zω
    unfolding vreal-impl-def
    by (metis (mono-tags, lifting) verit-sko-ex')+
qed

definition vreal-of-real-impl' :: V ⇒ V
  where vreal-of-real-impl' =
    (SOME f. bij-betw f (range vreal-of-real-impl) (elts vreal-impl))

lemma vreal-of-real-impl'-bij-betw:
  bij-betw vreal-of-real-impl' (range vreal-of-real-impl) (elts vreal-impl)
proof–
  from eqpoll-def obtain f where f:
    bij-betw f (range vreal-of-real-impl) (elts vreal-impl)
    by (auto intro: vreal-impl-eqpoll)
  then show ?thesis unfolding vreal-of-real-impl'-def by (metis verit-sko-ex')
qed

definition vreal-of-real-impl'' :: real ⇒ V
  where vreal-of-real-impl'' = vreal-of-real-impl' ∘ vreal-of-real-impl

lemma vreal-of-real-impl'': disjnt (range vreal-of-real-impl'') (elts ω)
proof–
  from comp-apply vreal-impl-vdisjnt vreal-of-real-impl'-bij-betw have
    vreal-of-real-impl'' y ∉∘ ω for y
    unfolding vreal-of-real-impl''-def by fastforce
  then show ?thesis unfolding disjnt-iff by clarsimp
qed

lemma inj-vreal-of-real-impl'': inj vreal-of-real-impl''
  unfolding vreal-of-real-impl''-def
  by
    (
      meson
        bij-betwE
        comp-inj-on
        inj-vreal-of-real-impl
        vreal-of-real-impl'-bij-betw
    )
```

Main definitions.

```
definition vreal-of-real :: real ⇒ V
  where vreal-of-real x =
    (if x ∈ ℕ then (nat-of-real x)ℕ else vreal-of-real-impl'' x)

notation vreal-of-real (‹-ℝ› [1000] 999)

declare [[coercion vreal-of-real :: real ⇒ V]]

definition vreal :: V (‹ℝ∘›)
  where vreal = set (range vreal-of-real)
```

definition *real-of-vreal* :: $V \Rightarrow$ *real*
where *real-of-vreal* = *inv-into UNIV vreal-of-real*

Rules.

lemma *vreal-of-real-in-vrealI*[*intro*, *simp*]: $a_{\mathbb{R}} \in_\circ \mathbb{R}_\circ$
by (*simp add*: *vreal-def*)

lemma *vreal-of-real-in-vrealE*[*elim*]:
assumes $a \in_\circ \mathbb{R}_\circ$
obtains b **where** $b_{\mathbb{R}} = a$
using *assms* **unfolding** *vreal-def* **by** *auto*

Elementary properties.

lemma *vnat-eq-vreal*: $x_{\mathbb{N}} = x_{\mathbb{R}}$ **by** (*simp add*: *nat-of-real-def vreal-of-real-def*)

lemma *omega-vsubset-vreal*: $\omega \subseteq_\circ \mathbb{R}_\circ$
proof
fix x **assume** $x \in_\circ \omega$
with *nat-of-omega* **obtain** y **where** *x-def*: $x = y_{\mathbb{N}}$ **by** *auto*
then have *vreal-of-real* (*real* y) = (*nat-of-real* (*real* y))$_{\mathbb{N}}$
unfolding *vreal-of-real-def* **by** *simp*
moreover have (*nat-of-real* (*real* y))$_{\mathbb{N}}$ = x
by (*simp add*: *nat-of-real-def x-def*)
ultimately show $x \in_\circ \mathbb{R}_\circ$ **unfolding** *vreal-def* **by** *clarsimp*
qed

lemma *inj-vreal-of-real*: *inj vreal-of-real*
proof
fix x y **assume** *prems*: *vreal-of-real* x = *vreal-of-real* y
consider
(*xy*) ‹$x \in \mathbb{N} \wedge y \in \mathbb{N}$› |
(*x-ny*) ‹$x \in \mathbb{N} \wedge y \notin \mathbb{N}$› |
(*nx-y*) ‹$x \notin \mathbb{N} \wedge y \in \mathbb{N}$› |
(*nxy*) ‹$x \notin \mathbb{N} \wedge y \notin \mathbb{N}$›
by *auto*
then show $x = y$
proof *cases*
case *xy*
then have (*nat-of-real* x)$_{\mathbb{N}}$ = (*nat-of-real* y)$_{\mathbb{N}}$
using *vreal-of-real-def prems* **by** *simp*
then show *?thesis*
by (*metis Nats-def f-inv-into-f nat-of-real-def ord-of-nat-inject xy*)
next
case *x-ny*
with *prems* **have** *eq*: (*nat-of-real* x)$_{\mathbb{N}}$ = *vreal-of-real-impl''* y
unfolding *vreal-of-real-def* **by** *simp*
have *vreal-of-real-impl''* $y \notin_\circ \omega$
by (*meson disjnt-iff rangeI vreal-of-real-impl''*)
then show *?thesis* **unfolding** *eq*[*symmetric*] **by** *auto*
next
case *nx-y*
with *prems* **have** *eq*: (*nat-of-real* y)$_{\mathbb{N}}$ = *vreal-of-real-impl''* x
unfolding *vreal-of-real-def* **by** *simp*
have *vreal-of-real-impl''* $x \notin_\circ \omega$
by (*meson disjnt-iff rangeI vreal-of-real-impl''*)
then show *?thesis* **unfolding** *eq*[*symmetric*] **by** *auto*
next

```
    case nxy
    then have x ∉ ℕ and y ∉ ℕ by auto
    with prems
    have vreal-of-real-impl'' x = vreal-of-real-impl'' y
      unfolding vreal-of-real-def by simp
    then show ?thesis by (meson inj-def inj-vreal-of-real-impl'')
  qed
qed

lemma vreal-in-Vset-ω2: ℝ∘ ∈∘ Vset (ω + ω)
  unfolding vreal-def
proof–
  have set (range vreal-of-real) ⊆∘ set (range vreal-of-real-impl'') ∪∘ ω
    unfolding vreal-of-real-def by auto
  moreover from vreal-of-real-impl'-bij-betw have
    set (range vreal-of-real-impl'') ⊆∘ vreal-impl
    unfolding vreal-of-real-impl''-def by fastforce
  ultimately show set (range vreal-of-real) ∈∘ Vset (ω + ω)
    using Ord-ω Ord-add
    by
      (
        auto simp:
          Ord-iff-rank
          Ord-VsetI
          vreal-impl-in-Vset-ss-omega
          vsubset-in-VsetI
          vunion-in-VsetI
      )
qed

lemma real-of-vreal-vreal-of-real[simp]: real-of-vreal (a_ℝ) = a
  by (simp add: inj-vreal-of-real real-of-vreal-def)
```

Transfer rules

```
definition cr-vreal :: V ⇒ real ⇒ bool
  where cr-vreal a b ⟷ (a = vreal-of-real b)

lemma cr-vreal-right-total[transfer-rule]: right-total cr-vreal
  unfolding cr-vreal-def right-total-def by simp

lemma cr-vreal-bi-uniqie[transfer-rule]: bi-unique cr-vreal
  unfolding cr-vreal-def bi-unique-def
  by (simp add: inj-eq inj-vreal-of-real)

lemma cr-vreal-transfer-domain-rule[transfer-domain-rule]:
  Domainp cr-vreal = (λx. x ∈∘ ℝ∘)
  unfolding cr-vreal-def by force

lemma vreal-transfer[transfer-rule]:
  (rel-set cr-vreal) (elts ℝ∘) (UNIV::real set)
  unfolding cr-vreal-def rel-set-def by auto

lemma vreal-of-real-transfer[transfer-rule]: cr-vreal (vreal-of-real a) a
  unfolding cr-vreal-def by auto
```

Constants and operations

Auxiliary.

lemma *vreal-fsingleton-in-fproduct-vreal*: $[a_{\mathbb{R}}]_{\circ} \in_{\circ} \mathbb{R}_{\circ} \mathbin{\widehat{\times}} 1_{\mathbb{N}}$ **by** *auto*

lemma *vreal-fpair-in-fproduct-vreal*: $[a_{\mathbb{R}}, b_{\mathbb{R}}]_{\circ} \in_{\circ} \mathbb{R}_{\circ} \mathbin{\widehat{\times}} 2_{\mathbb{N}}$ **by** *force*

Zero.

lemma *vreal-zero*: $0_{\mathbb{R}} = (0 :: V)$
by (*simp add*: *ord-of-nat-vempty vnat-eq-vreal*)

One.

lemma *vreal-one*: $1_{\mathbb{R}} = (1 :: V)$
by (*simp add*: *ord-of-nat-vone vnat-eq-vreal*)

Addition.

definition *vreal-plus* :: *V*
where *vreal-plus* =
$(\lambda x \in_{\circ} \mathbb{R}_{\circ} \mathbin{\widehat{\times}} 2_{\mathbb{N}}.$ (*real-of-vreal* ($x(\!|0_{\mathbb{N}}|\!)$) + *real-of-vreal* ($x(\!|1_{\mathbb{N}}|\!)$))$_{\mathbb{R}}$)

abbreviation *vreal-plus-app* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$+_{\mathbb{R}}$› 65)
where *vreal-plus-app a b* ≡ *vreal-plus*$(\!|a, b|\!)_{\bullet}$
notation *vreal-plus-app* (**infixl** ‹$+_{\mathbb{R}}$› 65)

lemma *vreal-plus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal* ===> *cr-vreal*)
($+_{\mathbb{R}}$) (+)
using *vreal-fpair-in-fproduct-vreal*
by (*intro rel-funI*, *unfold vreal-plus-def cr-vreal-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Multiplication.

definition *vreal-mult* :: *V*
where *vreal-mult* =
$(\lambda x \in_{\circ} \mathbb{R}_{\circ} \mathbin{\widehat{\times}} 2_{\mathbb{N}}.$ (*real-of-vreal* ($x(\!|0_{\mathbb{N}}|\!)$) * *real-of-vreal* ($x(\!|1_{\mathbb{N}}|\!)$))$_{\mathbb{R}}$)

abbreviation *vreal-mult-app* (**infixl** ‹$*_{\mathbb{R}}$› 70)
where *vreal-mult-app a b* ≡ *vreal-mult*$(\!|a, b|\!)_{\bullet}$
notation *vreal-mult-app* (**infixl** ‹$*_{\mathbb{R}}$› 70)

lemma *vreal-mult-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal* ===> *cr-vreal*) ($*_{\mathbb{R}}$) (*)
using *vreal-fpair-in-fproduct-vreal*
by (*intro rel-funI*, *unfold vreal-mult-def cr-vreal-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Unary minus.

definition *vreal-uminus* :: *V*
where *vreal-uminus* = $(\lambda x \in_{\circ} \mathbb{R}_{\circ}.$ (*uminus* (*real-of-vreal x*))$_{\mathbb{R}}$)

abbreviation *vreal-uminus-app* (‹$-_{\mathbb{R}}$ -› [81] 80)
where $-_{\mathbb{R}}$ *a* ≡ *vreal-uminus*$(\!|a|\!)$

lemma *vreal-uminus-transfer*[*transfer-rule*]:

includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal*) (*vreal-uminus-app*) (*uminus*)
using *vreal-fsingleton-in-fproduct-vreal*
by (*intro rel-funI*, *unfold vreal-uminus-def cr-vreal-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Multiplicative inverse.

definition *vreal-inverse* :: *V*
where *vreal-inverse* = ($\lambda x \in_\circ \mathbb{R}_\circ$. (*inverse* (*real-of-vreal x*))$_{\mathbb{R}}$)

abbreviation *vreal-inverse-app* (‹(-$^{-1}{}_{\mathbb{R}}$)› [1000] 999)
where $a^{-1}{}_{\mathbb{R}} \equiv$ *vreal-inverse*(|*a*|)

lemma *vreal-inverse-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal*) (*vreal-inverse-app*) (*inverse*)
using *vreal-fsingleton-in-fproduct-vreal*
by (*intro rel-funI*, *unfold vreal-inverse-def cr-vreal-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Order.

definition *vreal-le* :: *V*
where *vreal-le* =
set {$[a, b]_\circ$ | *a b*. $[a, b]_\circ \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}} \wedge$ *real-of-vreal a* $\leq$ *real-of-vreal b*}

abbreviation *vreal-le'* (‹(-/ $\leq_{\mathbb{R}}$ -)› [51, 51] 50)
where $a \leq_{\mathbb{R}} b \equiv [a, b]_\circ \in_\circ$ *vreal-le*

lemma *small-vreal-le*[*simp*]:
small
{$[a, b]_\circ$ | *a b*. $[a, b]_\circ \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}} \wedge$ *real-of-vreal a* $\leq$ *real-of-vreal b*}
proof–
have *small*: *small* {$[a, b]_\circ$ | *a b*. $[a, b]_\circ \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}$} **by** *simp*
show *?thesis* **by** (*rule smaller-than-small*[*OF small*]) *auto*
qed

lemma *vreal-le-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal* ===> (=)) *vreal-le'* ($\leq$)
using *vreal-fsingleton-in-fproduct-vreal*
by (*intro rel-funI*, *unfold cr-scalar-def cr-vreal-def vreal-le-def*)
(*auto simp*: *nat-omega-simps*)

Strict order.

definition *vreal-ls* :: *V*
where *vreal-ls* =
set {$[a, b]_\circ$ | *a b*. $[a, b]_\circ \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}} \wedge$ *real-of-vreal a* $<$ *real-of-vreal b*}

abbreviation *vreal-ls'* (‹(-/ $<_{\mathbb{R}}$ -)› [51, 51] 50)
where $a <_{\mathbb{R}} b \equiv [a, b]_\circ \in_\circ$ *vreal-ls*

lemma *small-vreal-ls*[*simp*]:
small
{$[a, b]_\circ$ | *a b*. $[a, b]_\circ \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}} \wedge$ *real-of-vreal a* $<$ *real-of-vreal b*}
proof–
have *small*: *small* {$[a, b]_\circ$ | *a b*. $[a, b]_\circ \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}$} **by** *simp*
show *?thesis* **by** (*rule smaller-than-small*[*OF small*]) *auto*

qed

lemma *vreal-ls-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal* ===> (=)) *vreal-ls'* (<)
by (*intro rel-funI*, *unfold cr-scalar-def cr-vreal-def vreal-ls-def*)
(*auto simp*: *nat-omega-simps*)

Subtraction.

definition *vreal-minus* :: *V*
where *vreal-minus* =
($\lambda x \in_\circ \mathbb{R}_\circ \mathbin{\widehat{}}_\times 2_\mathbb{N}$. (*real-of-vreal* ($x(\!|0_\mathbb{N}|\!)$) − *real-of-vreal* ($x(\!|1_\mathbb{N}|\!)$))$_\mathbb{R}$)

abbreviation *vreal-minus-app* (**infixl** $\langle -_\mathbb{R} \rangle$ 65)
where *vreal-minus-app a b* ≡ *vreal-minus*$(\!|a, b|\!)_\bullet$

lemma *vreal-minus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vreal* ===> *cr-vreal* ===> *cr-vreal*) ($-_\mathbb{R}$) (−)
using *vreal-fpair-in-fproduct-vreal*
by (*intro rel-funI*, *unfold vreal-minus-def cr-vreal-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Axioms of an ordered field with the least upper bound property.

The exposition follows the Definitions 2.2.1 and 2.2.3 from the textbook *The Real Numbers and Real Analysis* by E. Bloch [14].

lemma *vreal-zero-closed*: $0_\mathbb{R} \in_\circ \mathbb{R}_\circ$
proof–
have (0::*real*) ∈ *UNIV* **by** *simp*
from *this*[*untransferred*] **show** *?thesis*.
qed

lemma *vreal-one-closed*: $1_\mathbb{R} \in_\circ \mathbb{R}_\circ$
proof–
have (1::*real*) ∈ *UNIV* **by** *simp*
from *this*[*untransferred*] **show** *?thesis*.
qed

lemma *vreal-plus-closed*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$
shows $x +_\mathbb{R} y \in_\circ \mathbb{R}_\circ$
proof–
have $x' + y' \in$ *UNIV* **for** x' y' :: *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vreal-uminus-closed*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $-_\mathbb{R} x \in_\circ \mathbb{R}_\circ$
proof–
have $-x' \in$ *UNIV* **for** x' :: *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vreal-mult-closed*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$

shows $x *_{\mathbb{R}} y \in_\circ \mathbb{R}_\circ$
proof–
have $x' * y' \in UNIV$ **for** $x'\ y' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vreal-inverse-closed*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $x^{-1}_{\mathbb{R}} \in_\circ \mathbb{R}_\circ$
proof–
have *inverse* $x' \in UNIV$ **for** $x' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Associative Law for Addition: Definition 2.2.1.a.

lemma *vreal-assoc-law-addition*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$ **and** $z \in_\circ \mathbb{R}_\circ$
shows $(x +_{\mathbb{R}} y) +_{\mathbb{R}} z = x +_{\mathbb{R}} (y +_{\mathbb{R}} z)$
proof–
have $(x' + y') + z' = x' + (y' + z')$ **for** $x'\ y'\ z' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Commutative Law for Addition: Definition 2.2.1.b.

lemma *vreal-commutative-law-addition*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$
shows $x +_{\mathbb{R}} y = y +_{\mathbb{R}} x$
proof–
have $(x' + y') = y' + x'$ **for** $x'\ y' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Identity Law for Addition: Definition 2.2.1.c.

lemma *vreal-identity-law-addition*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $x +_{\mathbb{R}} 0_{\mathbb{R}} = x$
proof–
have $x' + 0 = x'$ **for** $x' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Inverses Law for Addition: Definition 2.2.1.d.

lemma *vreal-inverses-law-addition*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $x +_{\mathbb{R}} (-_{\mathbb{R}} x) = 0_{\mathbb{R}}$
proof–
have $x' + (-x') = 0$ **for** $x' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Associative Law for Multiplication: Definition 2.2.1.e.

lemma *vreal-assoc-law-multiplication*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$ **and** $z \in_\circ \mathbb{R}_\circ$
shows $(x *_{\mathbb{R}} y) *_{\mathbb{R}} z = x *_{\mathbb{R}} (y *_{\mathbb{R}} z)$
proof–
have $(x' * y') * z' = x' * (y' * z')$ **for** $x'\ y'\ z' :: real$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.

qed

Commutative Law for Multiplication: Definition 2.2.1.f.

lemma *vreal-commutative-law-multiplication*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$
shows $x *_{\mathbb{R}} y = y *_{\mathbb{R}} x$
proof–
have $(x' * y') = y' * x'$ **for** $x'\ y' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Identity Law for Multiplication: Definition 2.2.1.g.

lemma *vreal-identity-law-multiplication*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $x *_{\mathbb{R}} 1_{\mathbb{R}} = x$
proof–
have $x' * 1 = x'$ **for** $x' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Inverses Law for Multiplication: Definition 2.2.1.h.

lemma *vreal-inverses-law-multiplication*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $x \neq 0_{\mathbb{R}}$
shows $x *_{\mathbb{R}} {x^{-1}}_{\mathbb{R}} = 1_{\mathbb{R}}$
proof–
have $x' \neq 0 \Longrightarrow x' * \mathit{inverse}\ x' = 1$ **for** $x' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Distributive Law: Definition 2.2.1.i.

lemma *vreal-distributive-law*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$ **and** $z \in_\circ \mathbb{R}_\circ$
shows $x *_{\mathbb{R}} (y +_{\mathbb{R}} z) = x *_{\mathbb{R}} y +_{\mathbb{R}} x *_{\mathbb{R}} z$
proof–
have $x' * (y' + z') = (x' * y') + (x' * z')$ **for** $x'\ y'\ z' ::$ *real*
by (*simp add*: *field-simps*)
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Trichotomy Law: Definition 2.2.1.j.

lemma *vreal-trichotomy-law*:
assumes $x \in_\circ \mathbb{R}_\circ$ $y \in_\circ \mathbb{R}_\circ$
shows
$(x <_{\mathbb{R}} y \wedge \sim(x = y) \wedge \sim(y <_{\mathbb{R}} x)) \vee$
$(\sim(x <_{\mathbb{R}} y) \wedge x = y \wedge \sim(y <_{\mathbb{R}} x)) \vee$
$(\sim(x <_{\mathbb{R}} y) \wedge \sim(x = y) \wedge y <_{\mathbb{R}} x)$
proof–
have $(x' < y' \wedge \sim(x' = y') \wedge \sim(y' < x')) \vee$
$(\sim(x' < y') \wedge x' = y' \wedge \sim(y' < x')) \vee$
$(\sim(x' < y') \wedge \sim(x' = y') \wedge y' < x')$
for $x'\ y'\ z' ::$ *real*
by *auto*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Transitive Law: Definition 2.2.1.k.

lemma *vreal-transitive-law*:

```
  assumes x ∈∘ ℝ∘
    and y ∈∘ ℝ∘
    and z ∈∘ ℝ∘
    and x <ℝ y and y <ℝ z
  shows x <ℝ z
proof–
  have x' < y' ⟹ y' < z' ⟹ x' < z' for x' y' z' :: real by simp
  from this[untransferred, OF assms] show ?thesis.
qed
```

Addition Law of Order: Definition 2.2.1.l.

```
lemma vreal-addition-law-of-order:
  assumes x ∈∘ ℝ∘ and y ∈∘ ℝ∘ and z ∈∘ ℝ∘ and x <ℝ y
  shows x +ℝ z <ℝ y +ℝ z
proof–
  have x' < y' ⟹ x' + z' < y' + z' for x' y' z' :: real by simp
  from this[untransferred, OF assms] show ?thesis.
qed
```

Multiplication Law of Order: Definition 2.2.1.m.

```
lemma vreal-multiplication-law-of-order:
  assumes x ∈∘ ℝ∘
    and y ∈∘ ℝ∘
    and z ∈∘ ℝ∘
    and x <ℝ y
    and 0ℝ <ℝ z
  shows x *ℝ z <ℝ y *ℝ z
proof–
  have x' < y' ⟹ 0 < z' ⟹ x' * z' < y' * z' for x' y' z' :: real by simp
  from this[untransferred, OF assms] show ?thesis.
qed
```

Non-Triviality: Definition 2.2.1.n.

```
lemma vreal-non-triviality: 0ℝ ≠ 1ℝ
proof–
  have 0 ≠ (1::real) by simp
  from this[untransferred] show ?thesis.
qed
```

Least upper bound property: Definition 2.2.3.

```
lemma least-upper-bound-property:
  defines vreal-ub S M ≡ (S ⊆∘ ℝ∘ ∧ M ∈∘ ℝ∘ ∧ (∀ x∈∘S. x ≤ℝ M))
  assumes A ⊆∘ ℝ∘ and A ≠ 0 and ∃ M. vreal-ub A M
  obtains M where vreal-ub A M and ⋀T. vreal-ub A T ⟹ M ≤ℝ T
proof–
  note complete-real =
    complete-real[
      untransferred, of ‹elts A›, unfolded vnumber-simps, OF assms(2)
      ]
  from assms obtain x where x ∈∘ A by force
  moreover with assms have x ∈∘ ℝ∘ by auto
  ultimately have 1: ∃ x∈∘ℝ∘. x ∈∘ A by auto
  from assms have 2: ∃ x∈∘ℝ∘. ∀ y∈∘A. y ≤ℝ x by auto
  from complete-real[OF 1 2]
    obtain M
      where M ∈∘ ℝ∘
        and ⋀x. x ∈∘ A ⟹ x ≤ℝ M
```

and [*simp*]: $\bigwedge T.\ T \in_\circ \mathbb{R}_\circ \Longrightarrow (\bigwedge x.\ x \in_\circ A \Longrightarrow x \leq_{\mathbb{R}} T) \Longrightarrow M \leq_{\mathbb{R}} T$
by *force*
with *assms*(2) **have** *vreal-ub A M* **unfolding** *vreal-ub-def* **by** *simp*
moreover have *vreal-ub A T* $\Longrightarrow$ $M \leq_{\mathbb{R}} T$ **for** *T* **unfolding** *vreal-ub-def* **by** *simp*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

Fundamental properties of other operations

Minus.

lemma *vreal-minus-closed*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$
shows $x -_{\mathbb{R}} y \in_\circ \mathbb{R}_\circ$
proof–
have $x' - y' \in$ *UNIV* **for** $x'\ y' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vreal-minus-eq-plus-uminus*:
assumes $x \in_\circ \mathbb{R}_\circ$ **and** $y \in_\circ \mathbb{R}_\circ$
shows $x -_{\mathbb{R}} y = x +_{\mathbb{R}} (-_{\mathbb{R}} y)$
proof–
have $x' - y' = x' + (-y')$ **for** $x'\ y' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Unary minus.

lemma *vreal-uminus-uminus*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $x = -_{\mathbb{R}} (-_{\mathbb{R}} x)$
proof–
have $x' = -(-x')$ **for** $x' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Multiplicative inverse.

lemma *vreal-inverse-inverse*:
assumes $x \in_\circ \mathbb{R}_\circ$
shows $x = (x^{-1}{}_{\mathbb{R}})^{-1}{}_{\mathbb{R}}$
proof–
have $x' =$ *inverse* (*inverse* x') **for** $x' ::$ *real* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Further properties

Addition.

global-interpretation *vreal-plus*: *binop-onto* ‹$\mathbb{R}_\circ$› *vreal-plus*
proof–
have *binop*: *binop* $\mathbb{R}_\circ$ *vreal-plus*
proof(*intro binopI nopI*)
show *vsv*: *vsv vreal-plus* **unfolding** *vreal-plus-def* **by** *auto*
interpret *vsv vreal-plus* **by** (*rule vsv*)
show $2_{\mathbb{N}} \in_\circ \omega$ **by** *simp*
show *dom*: $\mathcal{D}_\circ$ *vreal-plus* = $\mathbb{R}_\circ \mathbin{\widehat{\ }}_\times 2_{\mathbb{N}}$ **unfolding** *vreal-plus-def* **by** *simp*
show $\mathcal{R}_\circ$ *vreal-plus* $\subseteq_\circ \mathbb{R}_\circ$

```
    proof(intro vsubsetI)
      fix y assume y ∈∘ ℛ∘ vreal-plus
      then obtain ab where ab ∈∘ ℝ∘ ^× 2ℕ and y-def: y = vreal-plus(|ab|)
        unfolding dom[symmetric] by force
      then obtain a b
        where ab-def: ab = [a, b]∘ and a: a ∈∘ ℝ∘ and b: b ∈∘ ℝ∘
        by blast
      then show y ∈∘ ℝ∘ by (simp add: vreal-plus-closed y-def)
    qed
  qed
  interpret binop ‹ℝ∘› vreal-plus by (rule binop)
  show binop-onto ℝ∘ vreal-plus
  proof(intro binop-ontoI′)
    show binop ℝ∘ vreal-plus by (rule binop-axioms)
    show ℝ∘ ⊆∘ ℛ∘ vreal-plus
    proof(intro vsubsetI)
      fix y assume prems: y ∈∘ ℝ∘
      moreover from vreal-zero vreal-zero-closed have 0 ∈∘ ℝ∘ by auto
      ultimately have y +ℝ 0 ∈∘ ℛ∘ vreal-plus by auto
      moreover from prems vreal-identity-law-addition have y = y +ℝ 0
        by (simp add: vreal-zero)
      ultimately show y ∈∘ ℛ∘ vreal-plus by simp
    qed
  qed
qed
```

Unary minus.

```
global-interpretation vreal-uminus: v11 vreal-uminus
  rewrites vreal-uminus-vdomain[simp]: 𝒟∘ vreal-uminus = ℝ∘
    and vreal-uminus-vrange[simp]: ℛ∘ vreal-uminus = ℝ∘
proof–
  show v11: v11 vreal-uminus
  proof(intro v11I)
    show vsv: vsv vreal-uminus unfolding vreal-uminus-def by simp
    interpret vsv vreal-uminus by (rule vsv)
    show vsv (vreal-uminus⁻¹∘)
    proof(intro vsvI)
      show vbrelation (vreal-uminus⁻¹∘) by clarsimp
      fix a b c
      assume prems: ⟨a, b⟩ ∈∘ vreal-uminus⁻¹∘ ⟨a, c⟩ ∈∘ vreal-uminus⁻¹∘
      then have ba: ⟨b, a⟩ ∈∘ vreal-uminus and ca: ⟨c, a⟩ ∈∘ vreal-uminus
        by auto
      then have b: b ∈∘ ℝ∘ and c: c ∈∘ ℝ∘
        by (simp-all add: VLambda-iff2 vreal-uminus-def)
      from ba ca have a = −ℝ b a = −ℝ c by simp-all
      with ba ca b c show b = c  by (metis vreal-uminus-uminus)
    qed
  qed
  interpret v11 vreal-uminus by (rule v11)
  show dom: 𝒟∘ vreal-uminus = ℝ∘ unfolding vreal-uminus-def by simp
  have ℛ∘ vreal-uminus ⊆∘ ℝ∘
  proof(intro vsubsetI)
    fix y assume y ∈∘ ℛ∘ vreal-uminus
    then obtain x where x ∈∘ ℝ∘ and y-def: y = −ℝ x
      unfolding dom[symmetric] by force
    then show y ∈∘ ℝ∘ by (simp add: vreal-uminus-closed)
  qed
  moreover have ℝ∘ ⊆∘ ℛ∘ vreal-uminus
```

```
    by (intro vsubsetI)
      (metis dom vdomain-atD vreal-uminus-closed vreal-uminus-uminus)
  ultimately show ℛ∘ vreal-uminus = ℝ∘ by simp
qed
```

Multiplication.

```
global-interpretation vreal-mult: binop-onto ‹ℝ∘› vreal-mult
proof–
  have binop: binop ℝ∘ vreal-mult
  proof(intro binopI nopI)
    show vsv: vsv vreal-mult unfolding vreal-mult-def by auto
    interpret vsv vreal-mult by (rule vsv)
    show 2ℕ ∈∘ ω by simp
    show dom: 𝒟∘ vreal-mult = ℝ∘ ^× 2ℕ unfolding vreal-mult-def by simp
    show ℛ∘ vreal-mult ⊆∘ ℝ∘
    proof(intro vsubsetI)
      fix y assume y ∈∘ ℛ∘ vreal-mult
      then obtain ab where ab ∈∘ ℝ∘ ^× 2ℕ and y-def: y = vreal-mult(|ab|)
        unfolding dom[symmetric] by force
      then obtain a b
        where ab-def: ab = [a, b]∘ and a: a ∈∘ ℝ∘ and b: b ∈∘ ℝ∘
        by blast
      then show y ∈∘ ℝ∘ by (simp add: vreal-mult-closed y-def)
    qed
  qed
  interpret binop ‹ℝ∘› vreal-mult by (rule binop)
  show binop-onto ℝ∘ vreal-mult
  proof(intro binop-ontoI′)
    show binop ℝ∘ vreal-mult by (rule binop-axioms)
    show ℝ∘ ⊆∘ ℝ∘ vreal-mult
    proof(intro vsubsetI)
      fix y assume prems: y ∈∘ ℝ∘
      moreover from vreal-one vreal-one-closed have 1 ∈∘ ℝ∘ by auto
      ultimately have y *ℝ 1 ∈∘ ℛ∘ vreal-mult by auto
      moreover from prems vreal-identity-law-multiplication have y = y *ℝ 1
        by (simp add: vreal-one)
      ultimately show y ∈∘ ℛ∘ vreal-mult by simp
    qed
  qed
qed
```

Multiplicative inverse.

```
global-interpretation vreal-inverse: v11 vreal-inverse
  rewrites vreal-inverse-vdomain[simp]: 𝒟∘ vreal-inverse = ℝ∘
    and vreal-inverse-vrange[simp]: ℛ∘ vreal-inverse = ℝ∘
proof–
  show v11: v11 vreal-inverse
  proof(intro v11I)
    show vsv: vsv vreal-inverse unfolding vreal-inverse-def by simp
    interpret vsv vreal-inverse by (rule vsv)
    show vsv (vreal-inverse⁻¹∘)
    proof(intro vsvI)
      show vbrelation (vreal-inverse⁻¹∘) by clarsimp
      fix a b c
      assume prems: ⟨a, b⟩ ∈∘ vreal-inverse⁻¹∘ ⟨a, c⟩ ∈∘ vreal-inverse⁻¹∘
      then have ba: ⟨b, a⟩ ∈∘ vreal-inverse and ca: ⟨c, a⟩ ∈∘ vreal-inverse
        by auto
      then have b: b ∈∘ ℝ∘ and c: c ∈∘ ℝ∘
```

```
      by (simp-all add: VLambda-iff2 vreal-inverse-def)
    from ba ca have a = b⁻¹ℝ a = c⁻¹ℝ by simp-all
    with ba ca b c show b = c  by (metis vreal-inverse-inverse)
  qed
 qed
 interpret v11 vreal-inverse by (rule v11)
 show dom: 𝒟∘ vreal-inverse = ℝ∘ unfolding vreal-inverse-def by simp
 have ℛ∘ vreal-inverse ⊆∘ ℝ∘
 proof(intro vsubsetI)
  fix y assume y ∈∘ ℛ∘ vreal-inverse
  then obtain x where x ∈∘ ℝ∘ and y-def: y = x⁻¹ℝ
   unfolding dom[symmetric] by force
  then show y ∈∘ ℝ∘ by (simp add: vreal-inverse-closed)
 qed
 moreover have ℝ∘ ⊆∘ ℛ∘ vreal-inverse
  by (intro vsubsetI)
   (metis dom vdomain-atD vreal-inverse-closed vreal-inverse-inverse)
 ultimately show ℛ∘ vreal-inverse = ℝ∘ by simp
qed
```

2.13.3 Integer numbers

Definition

```
definition vint-of-int :: int ⇒ V
 where vint-of-int = vreal-of-real

notation vint-of-int (‹-ℤ› [999] 999)

declare [[coercion vint-of-int :: int ⇒ V]]

definition vint :: V (‹ℤ∘›)
 where vint = set (range vint-of-int)

definition int-of-vint :: V ⇒ int
 where int-of-vint = inv-into UNIV vint-of-int
```

Rules.

```
lemma vint-of-int-in-vintI[intro, simp]: a_ℤ ∈∘ ℤ∘ by (simp add: vint-def)

lemma vint-of-int-in-vintE[elim]:
 assumes a ∈∘ ℤ∘
 obtains b where b_ℤ = a
 using assms unfolding vint-def by auto
```

Elementary properties

```
lemma vint-vsubset-vreal: ℤ∘ ⊆∘ ℝ∘
 unfolding vint-def vint-of-int-def vreal-def using image-cong by auto

lemma inj-vint-of-int: inj vint-of-int
 using inj-vreal-of-real
 unfolding vint-of-int-def inj-def of-int-eq-iff
 by force

lemma vint-in-Vset-ω2: ℤ∘ ∈∘ Vset (ω + ω)
 using vint-vsubset-vreal vreal-in-Vset-ω2 by auto

lemma int-of-vint-vint-of-int[simp]: int-of-vint (a_ℤ) = a
```

by (*simp add*: *inj-vint-of-int int-of-vint-def*)

Transfer rules.

definition *cr-vint* :: $V \Rightarrow int \Rightarrow bool$
where *cr-vint a b* $\longleftrightarrow$ (*a* = *vint-of-int b*)

lemma *cr-vint-right-total*[*transfer-rule*]: *right-total cr-vint*
unfolding *cr-vint-def right-total-def* **by** *simp*

lemma *cr-vint-bi-unqie*[*transfer-rule*]: *bi-unique cr-vint*
unfolding *cr-vint-def bi-unique-def*
by (*simp add*: *inj-eq inj-vint-of-int*)

lemma *cr-vint-transfer-domain-rule*[*transfer-domain-rule*]:
Domainp cr-vint = ($\lambda x.\ x \in_\circ \mathbb{Z}_\circ$)
unfolding *cr-vint-def* **by** *force*

lemma *vint-transfer*[*transfer-rule*]:
(*rel-set cr-vint*) (*elts* $\mathbb{Z}_\circ$) (*UNIV*::*int set*)
unfolding *cr-vint-def rel-set-def* **by** *auto*

lemma *vint-of-int-transfer*[*transfer-rule*]: *cr-vint* (*vint-of-int a*) *a*
unfolding *cr-vint-def* **by** *auto*

Constants and operations

Auxiliary.

lemma *vint-fsingleton-in-fproduct-vint*: $[a_\mathbb{Z}]_\circ \in_\circ \mathbb{Z}_\circ \mathbin{\widehat{}}_\times 1_\mathbb{N}$ **by** *auto*

lemma *vint-fpair-in-fproduct-vint*: $[a_\mathbb{Z},\ b_\mathbb{Z}]_\circ \in_\circ \mathbb{Z}_\circ \mathbin{\widehat{}}_\times 2_\mathbb{N}$ **by** *force*

Zero.

lemma *vint-zero*: $0_\mathbb{Z}$ = (0::*V*) **by** (*simp add*: *vint-of-int-def vreal-zero*)

One.

lemma *vint-one*: $1_\mathbb{Z}$ = (1::*V*) **by** (*simp add*: *vreal-one vint-of-int-def*)

Addition.

definition *vint-plus* :: *V*
where *vint-plus* =
($\lambda x \in_\circ \mathbb{Z}_\circ \mathbin{\widehat{}}_\times 2_\mathbb{N}$. (*int-of-vint* ($x(\!|0_\mathbb{N}|\!)$) + *int-of-vint* ($x(\!|1_\mathbb{N}|\!)$))$_\mathbb{Z}$)

abbreviation *vint-plus-app* (**infixl** ‹+$_\mathbb{Z}$› 65)
where *vint-plus-app a b* ≡ *vint-plus*$(\!|a,\ b|\!)_\bullet$

lemma *vint-plus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vint* ===> *cr-vint* ===> *cr-vint*) (+$_\mathbb{Z}$) (+)
using *vint-fpair-in-fproduct-vint*
by (*intro rel-funI*, *unfold vint-plus-def cr-vint-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Multiplication.

definition *vint-mult* :: *V*
where *vint-mult* =
($\lambda x \in_\circ \mathbb{Z}_\circ \mathbin{\widehat{}}_\times 2_\mathbb{N}$. (*int-of-vint* ($x(\!|0_\mathbb{N}|\!)$) * *int-of-vint* ($x(\!|1_\mathbb{N}|\!)$))$_\mathbb{Z}$)

abbreviation *vint-mult-app* (**infixl** ‹$*_{\mathbb{Z}}$› 65)
  where *vint-mult-app a b* ≡ *vint-mult*(⌊*a*, *b*⌋)$_\bullet$

lemma *vint-mult-transfer*[*transfer-rule*]:
  includes *lifting-syntax*
  shows (*cr-vint* ===> *cr-vint* ===> *cr-vint*) ($*_{\mathbb{Z}}$) (*)
  using *vint-fpair-in-fproduct-vint*
  by (*intro rel-funI*, *unfold vint-mult-def cr-vint-def cr-scalar-def*)
    (*simp add*: *nat-omega-simps*)

Unary minus.

definition *vint-uminus* :: *V*
  where *vint-uminus* = ($\lambda x \in_\circ \mathbb{Z}_\circ$. (*uminus* (*int-of-vint x*))$_{\mathbb{Z}}$)

abbreviation *vint-uminus-app* (‹$-_{\mathbb{Z}}$ -› [81] 80)
  where $-_{\mathbb{Z}}$ *a* ≡ *vint-uminus*(⌊*a*⌋)

lemma *vint-uminus-transfer*[*transfer-rule*]:
  includes *lifting-syntax*
  shows (*cr-vint* ===> *cr-vint*) (*vint-uminus-app*) (*uminus*)
  using *vint-fsingleton-in-fproduct-vint*
  by (*intro rel-funI*, *unfold vint-uminus-def cr-vint-def cr-scalar-def*)
    (*simp add*: *nat-omega-simps*)

Order.

definition *vint-le* :: *V*
  where *vint-le* =
    set {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Z}_\circ$ $\widehat{}_\times$ $2_{\mathbb{N}}$ ∧ *int-of-vint a* ≤ *int-of-vint b*}

abbreviation *vint-le′* (‹(-/ $\leq_{\mathbb{Z}}$ -)› [51, 51] 50)
  where *a* $\leq_{\mathbb{Z}}$ *b* ≡ [*a*, *b*]$_\circ$ $\in_\circ$ *vint-le*

lemma *small-vint-le*[*simp*]:
  small {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Z}_\circ$ $\widehat{}_\times$ $2_{\mathbb{N}}$ ∧ *int-of-vint a* ≤ *int-of-vint b*}
proof–
  have *small*: *small* {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Z}_\circ$ $\widehat{}_\times$ $2_{\mathbb{N}}$} **by** *simp*
  show *?thesis* **by** (*rule smaller-than-small*[*OF small*]) *auto*
qed

lemma *vint-le-transfer*[*transfer-rule*]:
  includes *lifting-syntax*
  shows (*cr-vint* ===> *cr-vint* ===> (=)) *vint-le′* (≤)
  using *vint-fsingleton-in-fproduct-vint*
  by (*intro rel-funI*, *unfold cr-scalar-def cr-vint-def vint-le-def*)
    (*auto simp*: *nat-omega-simps*)

Strict order.

definition *vint-ls* :: *V*
  where *vint-ls* =
    set {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Z}_\circ$ $\widehat{}_\times$ $2_{\mathbb{N}}$ ∧ *int-of-vint a* < *int-of-vint b*}

abbreviation *vint-ls′* (‹(-/ $<_{\mathbb{Z}}$ -)› [51, 51] 50)
  where *a* $<_{\mathbb{Z}}$ *b* ≡ [*a*, *b*]$_\circ$ $\in_\circ$ *vint-ls*

lemma *small-vint-ls*[*simp*]:
  small {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Z}_\circ$ $\widehat{}_\times$ $2_{\mathbb{N}}$ ∧ *int-of-vint a* < *int-of-vint b*}

proof–
have *small*: *small* $\{[a,\ b]_\circ \mid a\ b.\ [a,\ b]_\circ \in_\circ \mathbb{Z}_\circ\ \widehat{}_\times\ 2_{\mathbb{N}}\}$ **by** *simp*
show *?thesis* **by** (*rule smaller-than-small*[*OF small*]) *auto*
qed

lemma *vint-ls-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vint* ===> *cr-vint* ===> (=)) *vint-ls'* (<)
using *vint-fsingleton-in-fproduct-vint*
by (*intro rel-funI*, *unfold cr-scalar-def cr-vint-def vint-ls-def*)
(*auto simp*: *nat-omega-simps*)

Subtraction.

definition *vint-minus* :: *V*
where *vint-minus* =
$(\lambda x \in_\circ \mathbb{Z}_\circ\ \widehat{}_\times\ 2_{\mathbb{N}}.\ (\textit{int-of-vint}\ (x(\!|0_{\mathbb{N}}|\!)) - \textit{int-of-vint}\ (x(\!|1_{\mathbb{N}}|\!)))_{\mathbb{Z}})$

abbreviation *vint-minus-app* (**infixl** $\langle -_{\mathbb{Z}} \rangle$ 65)
where *vint-minus-app a b* ≡ *vint-minus*$(\!|a,\ b|\!)_\bullet$

lemma *vint-minus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vint* ===> *cr-vint* ===> *cr-vint*) $(-_{\mathbb{Z}})$ (−)
using *vint-fpair-in-fproduct-vint*
by (*intro rel-funI*, *unfold vint-minus-def cr-vint-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Axioms of a well ordered integral domain

The exposition follows Definition 1.4.1 from the textbook *The Real Numbers and Real Analysis* by E. Bloch [14].

lemma *vint-zero-closed*: $0_{\mathbb{Z}} \in_\circ \mathbb{Z}_\circ$ **by** *auto*

lemma *vint-one-closed*: $1_{\mathbb{Z}} \in_\circ \mathbb{Z}_\circ$ **by** *auto*

lemma *vint-plus-closed*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows $x +_{\mathbb{Z}} y \in_\circ \mathbb{Z}_\circ$
proof–
have $x' + y' \in \textit{UNIV}$ **for** $x'\ y'$ :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vint-mult-closed*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows $x *_{\mathbb{Z}} y \in_\circ \mathbb{Z}_\circ$
proof–
have $(x'::\textit{int}) * y' \in \textit{UNIV}$ **for** $x'\ y'$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vint-uminus-closed*:
assumes $x \in_\circ \mathbb{Z}_\circ$
shows $-_{\mathbb{Z}} x \in_\circ \mathbb{Z}_\circ$
proof–
have $(-x'::\textit{int}) \in \textit{UNIV}$ **for** x' **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.

qed

Associative Law for Addition: Definition 1.4.1.a.

lemma *vint-assoc-law-addition*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$ **and** $z \in_\circ \mathbb{Z}_\circ$
shows $(x +_\mathbb{Z} y) +_\mathbb{Z} z = x +_\mathbb{Z} (y +_\mathbb{Z} z)$
proof–
have $(x' + y') + z' = x' + (y' + z')$ **for** x' y' z' :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Commutative Law for Addition: Definition 1.4.1.b.

lemma *vint-commutative-law-addition*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows $x +_\mathbb{Z} y = y +_\mathbb{Z} x$
proof–
have $x' + y' = y' + x'$ **for** x' y' :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Identity Law for Addition: Definition 1.4.1.c.

lemma *vint-identity-law-addition*:
assumes [*simp*]: $x \in_\circ \mathbb{Z}_\circ$
shows $x +_\mathbb{Z} 0_\mathbb{Z} = x$
proof–
have $x' + 0 = x'$ **for** x' :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Inverses Law for Addition: Definition 1.4.1.d.

lemma *vint-inverses-law-addition*:
assumes [*simp*]: $x \in_\circ \mathbb{Z}_\circ$
shows $x +_\mathbb{Z} (-_\mathbb{Z} x) = 0_\mathbb{Z}$
proof–
have $x' + (-x') = 0$ **for** x' :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Associative Law for Multiplication: Definition 1.4.1.e.

lemma *vint-assoc-law-multiplication*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$ **and** $z \in_\circ \mathbb{Z}_\circ$
shows $(x *_\mathbb{Z} y) *_\mathbb{Z} z = x *_\mathbb{Z} (y *_\mathbb{Z} z)$
proof–
have $(x' * y') * z' = x' * (y' * z')$ **for** x' y' z' :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Commutative Law for Multiplication: Definition 1.4.1.f.

lemma *vint-commutative-law-multiplication*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows $x *_\mathbb{Z} y = y *_\mathbb{Z} x$
proof–
have $x' * y' = y' * x'$ **for** x' y' :: *int* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Identity Law for multiplication: Definition 1.4.1.g.

lemma *vint-identity-law-multiplication*:
assumes $x \in_\circ \mathbb{Z}_\circ$
shows $x *_{\mathbb{Z}} 1_{\mathbb{Z}} = x$
proof–
have $x' * 1 = x'$ **for** $x' :: int$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Distributive Law for Multiplication: Definition 1.4.1.h.

lemma *vint-distributive-law*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$ **and** $z \in_\circ \mathbb{Z}_\circ$
shows $x *_{\mathbb{Z}} (y +_{\mathbb{Z}} z) = (x *_{\mathbb{Z}} y) +_{\mathbb{Z}} (x *_{\mathbb{Z}} z)$
proof–
have $x' * (y' + z') = (x' * y') + (x' * z')$ **for** $x'\ y'\ z' :: int$
by (*simp add*: *algebra-simps*)
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

No Zero Divisors Law: Definition 1.4.1.i.

lemma *vint-no-zero-divisors-law*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$ **and** $x *_{\mathbb{Z}} y = 0_{\mathbb{Z}}$
shows $x = 0_{\mathbb{Z}} \vee y = 0_{\mathbb{Z}}$
proof–
have $x' * y' = 0 \Longrightarrow x' = 0 \vee y' = 0$ **for** $x'\ y'\ z' :: int$
by (*simp add*: *algebra-simps*)
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Trichotomy Law: Definition 1.4.1.j

lemma *vint-trichotomy-law*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows
$(x <_{\mathbb{Z}} y \wedge \sim(x = y) \wedge \sim(y <_{\mathbb{Z}} x)) \vee$
$(\sim(x <_{\mathbb{Z}} y) \wedge x = y \wedge \sim(y <_{\mathbb{Z}} x)) \vee$
$(\sim(x <_{\mathbb{Z}} y) \wedge \sim(x = y) \wedge y <_{\mathbb{Z}} x)$
proof–
have
$(x' < y' \wedge \sim(x' = y') \wedge \sim(y' < x')) \vee$
$(\sim(x' < y') \wedge x' = y' \wedge \sim(y' < x')) \vee$
$(\sim(x' < y') \wedge \sim(x' = y') \wedge y' < x')$
for $x'\ y'\ z' :: int$
by *auto*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Transitive Law: Definition 1.4.1.k

lemma *vint-transitive-law*:
assumes $x \in_\circ \mathbb{Z}_\circ$
and $y \in_\circ \mathbb{Z}_\circ$
and $z \in_\circ \mathbb{Z}_\circ$
and $x <_{\mathbb{Z}} y$
and $y <_{\mathbb{Z}} z$
shows $x <_{\mathbb{Z}} z$
proof–
have $x' < y' \Longrightarrow y' < z' \Longrightarrow x' < z'$ **for** $x'\ y'\ z' :: int$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Addition Law of Order: Definition 1.4.1.l

lemma *vint-addition-law-of-order*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$ **and** $z \in_\circ \mathbb{Z}_\circ$ **and** $x <_\mathbb{Z} y$
shows $x +_\mathbb{Z} z <_\mathbb{Z} y +_\mathbb{Z} z$
proof–
have $x' < y' \Longrightarrow x' + z' < y' + z'$ **for** $x'\ y'\ z' :: int$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Multiplication Law of Order: Definition 1.4.1.m

lemma *vint-multiplication-law-of-order*:
assumes $x \in_\circ \mathbb{Z}_\circ$
and $y \in_\circ \mathbb{Z}_\circ$
and $z \in_\circ \mathbb{Z}_\circ$
and $x <_\mathbb{Z} y$
and $0_\mathbb{Z} <_\mathbb{Z} z$
shows $x *_\mathbb{Z} z <_\mathbb{Z} y *_\mathbb{Z} z$
proof–
have $x' < y' \Longrightarrow 0 < z' \Longrightarrow x' * z' < y' * z'$ **for** $x'\ y'\ z' :: int$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Non-Triviality: Definition 1.4.1.n

lemma *vint-non-triviality*: $0_\mathbb{Z} \neq 1_\mathbb{Z}$
proof–
have $0 \neq (1::int)$ **by** *simp*
from *this*[*untransferred*] **show** *?thesis*.
qed

Well-Ordering Principle.

lemma *well-ordering-principle*:
assumes $A \subseteq_\circ \mathbb{Z}_\circ$
and $a \in_\circ \mathbb{Z}_\circ$
and $A \neq 0$
and $\bigwedge x.\ x \in_\circ A \Longrightarrow a <_\mathbb{Z} x$
obtains b **where** $b \in_\circ A$ **and** $\bigwedge x.\ x \in_\circ A \Longrightarrow b \leq_\mathbb{Z} x$
proof–
{
fix A' **and** $a' :: int$ **assume** *prems*: $A' \neq \{\}\ x \in A' \Longrightarrow a' < x$ **for** x
then obtain a'' **where** a'': $a'' \in A'$ **by** *auto*
from *wfE-min*[*OF wf-int-ge-less-than*[*of* a'], *OF* a''] **obtain** b'
where b'-A': $b' \in A'$
and yb': $(y,\ b') \in$ *int-ge-less-than* $a' \Longrightarrow y \notin A'$
for y
by *auto*
moreover from *prems* b'-A' yb' **have** $\bigwedge x.\ x \in A' \Longrightarrow b' \leq x$
unfolding *int-ge-less-than-def* **by** *fastforce*
with b'-A' **have** $\exists\, b.\ b \in A' \wedge (\forall\, x.\ x \in A' \longrightarrow b \leq x)$ **by** *blast*
}
note *real-wo* = *this*
from *real-wo*[
untransferred, *of* ‹*elts A*›, *unfolded vnumber-simps*, *OF assms*(1,2)
]
obtain b
where $b \in_\circ \mathbb{Z}_\circ$
and $b \in_\circ A$
and $\bigwedge x.\ x \in_\circ \mathbb{Z}_\circ \Longrightarrow x \in_\circ A \Longrightarrow b \leq_\mathbb{Z} x$

by (*auto simp*: *assms*(3,4))
with *assms that* **show** *?thesis* **unfolding** *vsubset-iff* **by** *simp*
qed

Fundamental properties of other operations

Minus.

lemma *vint-minus-closed*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows $x -_{\mathbb{Z}} y \in_\circ \mathbb{Z}_\circ$
proof–
have $x' - y' \in \mathit{UNIV}$ **for** $x'\ y' :: \mathit{int}$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vint-minus-eq-plus-uminus*:
assumes $x \in_\circ \mathbb{Z}_\circ$ **and** $y \in_\circ \mathbb{Z}_\circ$
shows $x -_{\mathbb{Z}} y = x +_{\mathbb{Z}} (-_{\mathbb{Z}} y)$
proof–
have $x' - y' = x' + (-y')$ **for** $x'\ y' :: \mathit{int}$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Unary minus.

lemma *vint-uminus-uminus*:
assumes $x \in_\circ \mathbb{Z}_\circ$
shows $x = -_{\mathbb{Z}} (-_{\mathbb{Z}} x)$
proof–
have $x' = -(-x')$ **for** $x' :: \mathit{int}$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Further properties

Addition.

global-interpretation *vint-plus*: *binop-onto* ‹$\mathbb{Z}_\circ$› *vint-plus*
proof–
have *binop*: *binop* $\mathbb{Z}_\circ$ *vint-plus*
proof(*intro binopI nopI*)
show *vsv*: *vsv vint-plus* **unfolding** *vint-plus-def* **by** *auto*
interpret *vsv vint-plus* **by** (*rule vsv*)
show $2_{\mathbb{N}} \in_\circ \omega$ **by** *simp*
show *dom*: $\mathcal{D}_\circ$ *vint-plus* = $\mathbb{Z}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}$ **unfolding** *vint-plus-def* **by** *simp*
show $\mathcal{R}_\circ$ *vint-plus* $\subseteq_\circ \mathbb{Z}_\circ$
proof(*intro vsubsetI*)
fix *y* **assume** $y \in_\circ \mathcal{R}_\circ$ *vint-plus*
then obtain *ab* **where** $ab \in_\circ \mathbb{Z}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}$ **and** *y-def*: y = *vint-plus*(‖*ab*‖)
unfolding *dom*[*symmetric*] **by** *force*
then obtain *a b*
where *ab-def*: $ab = [a, b]_\circ$ **and** *a*: $a \in_\circ \mathbb{Z}_\circ$ **and** *b*: $b \in_\circ \mathbb{Z}_\circ$
by *blast*
then show $y \in_\circ \mathbb{Z}_\circ$ **by** (*simp add*: *vint-plus-closed y-def*)
qed
qed
interpret *binop* ‹$\mathbb{Z}_\circ$› *vint-plus* **by** (*rule binop*)
show *binop-onto* $\mathbb{Z}_\circ$ *vint-plus*
proof(*intro binop-ontoI′*)

```
    show binop ℤ∘ vint-plus by (rule binop-axioms)
    show ℤ∘ ⊆∘ ℛ∘ vint-plus
    proof(intro vsubsetI)
      fix y assume prems: y ∈∘ ℤ∘
      moreover from vint-zero vint-zero-closed have 0 ∈∘ ℤ∘ by auto
      ultimately have y +ℤ 0 ∈∘ ℛ∘ vint-plus by auto
      moreover from prems vint-identity-law-addition have y = y +ℤ 0
        by (simp add: vint-zero)
      ultimately show y ∈∘ ℛ∘ vint-plus by simp
    qed
  qed
qed
```

Unary minus.

```
global-interpretation vint-uminus: v11 vint-uminus
  rewrites vint-uminus-vdomain[simp]: 𝒟∘ vint-uminus = ℤ∘
    and vint-uminus-vrange[simp]: ℛ∘ vint-uminus = ℤ∘
proof–
  show v11: v11 vint-uminus
  proof(intro v11I)
    show vsv: vsv vint-uminus unfolding vint-uminus-def by simp
    interpret vsv vint-uminus by (rule vsv)
    show vsv (vint-uminus⁻¹∘)
    proof(intro vsvI)
      show vbrelation (vint-uminus⁻¹∘) by clarsimp
      fix a b c
      assume prems: ⟨a, b⟩ ∈∘ vint-uminus⁻¹∘ ⟨a, c⟩ ∈∘ vint-uminus⁻¹∘
      then have ba: ⟨b, a⟩ ∈∘ vint-uminus and ca: ⟨c, a⟩ ∈∘ vint-uminus
        by auto
      then have b: b ∈∘ ℤ∘ and c: c ∈∘ ℤ∘
        by (simp-all add: VLambda-iff2 vint-uminus-def)
      from ba ca have a = −ℤ b a = −ℤ c by simp-all
      with ba ca b c show b = c  by (metis vint-uminus-uminus)
    qed
  qed
  interpret v11 vint-uminus by (rule v11)
  show dom: 𝒟∘ vint-uminus = ℤ∘ unfolding vint-uminus-def by simp
  have ℛ∘ vint-uminus ⊆∘ ℤ∘
  proof(intro vsubsetI)
    fix y assume y ∈∘ ℛ∘ vint-uminus
    then obtain x where x ∈∘ ℤ∘ and y-def: y = −ℤ x
      unfolding dom[symmetric] by force
    then show y ∈∘ ℤ∘ by (simp add: vint-uminus-closed)
  qed
  moreover have ℤ∘ ⊆∘ ℛ∘ vint-uminus
    by (intro vsubsetI)
      (metis dom vdomain-atD vint-uminus-closed vint-uminus-uminus)
  ultimately show ℛ∘ vint-uminus = ℤ∘ by simp
qed
```

Multiplication.

```
global-interpretation vint-mult: binop-onto ‹ℤ∘› vint-mult
proof–
  have binop: binop ℤ∘ vint-mult
  proof(intro binopI nopI)
    show vsv: vsv vint-mult unfolding vint-mult-def by auto
    interpret vsv vint-mult by (rule vsv)
    show 2ℕ ∈∘ ω by simp
```

```
    show dom: 𝒟∘ vint-mult = ℤ∘ ^× 2ℕ unfolding vint-mult-def by simp
    show ℛ∘ vint-mult ⊆∘ ℤ∘
    proof(intro vsubsetI)
      fix y assume y ∈∘ ℛ∘ vint-mult
      then obtain ab where ab ∈∘ ℤ∘ ^× 2ℕ and y-def: y = vint-mult⦇ab⦈
        unfolding dom[symmetric] by force
      then obtain a b
        where ab-def: ab = [a, b]∘ and a: a ∈∘ ℤ∘ and b: b ∈∘ ℤ∘
        by blast
      then show y ∈∘ ℤ∘ by (simp add: vint-mult-closed y-def)
    qed
  qed
  interpret binop ‹ℤ∘› vint-mult by (rule binop)
  show binop-onto ℤ∘ vint-mult
  proof(intro binop-ontoI′)
    show binop ℤ∘ vint-mult by (rule binop-axioms)
    show ℤ∘ ⊆∘ ℛ∘ vint-mult
    proof(intro vsubsetI)
      fix y assume prems: y ∈∘ ℤ∘
      moreover from vint-one vint-one-closed have 0: 1 ∈∘ ℤ∘ by auto
      ultimately have y *ℤ 1 ∈∘ ℛ∘ vint-mult by auto
      moreover from prems vint-identity-law-multiplication have y = y *ℤ 1
        by (simp add: vint-one)
      ultimately show y ∈∘ ℛ∘ vint-mult by simp
    qed
  qed
qed
```

Misc.

```
lemma (in 𝒵) vint-in-Vset[intro]: ℤ∘ ∈∘ Vset α
  using vint-in-Vset-ω2 vsubsetD by (auto intro!: 𝒵-Vset-ω2-vsubset-Vset)
```

2.13.4 Rational numbers

Definition

```
definition vrat-of-rat :: rat ⇒ V
  where vrat-of-rat x = vreal-of-real (real-of-rat x)

notation vrat-of-rat (‹-ℚ› [999] 999)

declare [[coercion vrat-of-rat :: rat ⇒ V]]

definition vrat :: V (‹ℚ∘›)
  where vrat = set (range vrat-of-rat)

definition rat-of-vrat :: V ⇒ rat
  where rat-of-vrat = inv-into UNIV vrat-of-rat
```

Rules.

```
lemma vrat-of-rat-in-vratI[intro, simp]: a_ℚ ∈∘ ℚ∘ by (simp add: vrat-def)

lemma vrat-of-rat-in-vratE[elim]:
  assumes a ∈∘ ℚ∘
  obtains b where b_ℚ = a
  using assms unfolding vrat-def by auto
```

Elementary properties

lemma *vrat-vsubset-vreal*: $\mathbb{Q}_\circ \subseteq_\circ \mathbb{R}_\circ$
unfolding *vrat-def vrat-of-rat-def vreal-def* **using** *image-cong* **by** *auto*

lemma *vrat-in-Vset-ω2*: $\mathbb{Q}_\circ \in_\circ Vset\ (\omega + \omega)$
using *vrat-vsubset-vreal vreal-in-Vset-ω2* **by** *auto*

lemma *inj-vrat-of-rat*: *inj vrat-of-rat*
using *inj-vreal-of-real*
unfolding *vrat-of-rat-def inj-def of-rat-eq-iff*
by *force*

lemma *rat-of-vrat-vrat-of-rat*[*simp*]: *rat-of-vrat* $(a_{\mathbb{Q}})$ = *a*
by (*simp add*: *inj-vrat-of-rat rat-of-vrat-def*)

Transfer rules.

definition *cr-vrat* :: *V* ⇒ *rat* ⇒ *bool*
where *cr-vrat a b* ⟷ (*a* = *vrat-of-rat b*)

lemma *cr-vrat-right-total*[*transfer-rule*]: *right-total cr-vrat*
unfolding *cr-vrat-def right-total-def* **by** *simp*

lemma *cr-vrat-bi-unqie*[*transfer-rule*]: *bi-unique cr-vrat*
unfolding *cr-vrat-def bi-unique-def*
by (*simp add*: *inj-eq inj-vrat-of-rat*)

lemma *cr-vrat-transfer-domain-rule*[*transfer-domain-rule*]:
Domainp cr-vrat = (λ*x*. *x* $\in_\circ \mathbb{Q}_\circ$)
unfolding *cr-vrat-def* **by** *force*

lemma *vrat-transfer*[*transfer-rule*]:
(*rel-set cr-vrat*) (*elts* $\mathbb{Q}_\circ$) (*UNIV*::*rat set*)
unfolding *cr-vrat-def rel-set-def* **by** *auto*

lemma *vrat-of-rat-transfer*[*transfer-rule*]: *cr-vrat* (*vrat-of-rat a*) *a*
unfolding *cr-vrat-def* **by** *auto*

Operations

lemma *vrat-fsingleton-in-fproduct-vrat*: $[a_{\mathbb{Q}}]_\circ \in_\circ \mathbb{Q}_\circ \mathbin{\widehat{}}_\times 1_{\mathbb{N}}$ **by** *auto*

lemma *vrat-fpair-in-fproduct-vrat*: $[a_{\mathbb{Q}}, b_{\mathbb{Q}}]_\circ \in_\circ \mathbb{Q}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}$ **by** *force*

Zero.

lemma *vrat-zero*: $0_{\mathbb{Q}}$ = (0::*V*) **by** (*simp add*: *vrat-of-rat-def vreal-zero*)

One.

lemma *vrat-one*: $1_{\mathbb{Q}}$ = (1::*V*) **by** (*simp add*: *vreal-one vrat-of-rat-def*)

Addition.

definition *vrat-plus* :: *V*
where *vrat-plus* =
($\lambda x \in_\circ \mathbb{Q}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}$. (*rat-of-vrat* ($x(\!|0_{\mathbb{N}}|\!)$) + *rat-of-vrat* ($x(\!|1_{\mathbb{N}}|\!)$))$_{\mathbb{Q}}$)

abbreviation *vrat-plus-app* (**infixl** ‹+$_{\mathbb{Q}}$› 65)
where *vrat-plus-app a b* ≡ *vrat-plus*$(\!|a, b|\!)_\bullet$

lemma *vrat-plus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat* ===> *cr-vrat*) $(+_{\mathbb{Q}})$ $(+)$
using *vrat-fpair-in-fproduct-vrat*
by (*intro rel-funI*, *unfold vrat-plus-def cr-vrat-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Multiplication.

definition *vrat-mult* :: *V*
where *vrat-mult* =
$(\lambda x \in_\circ \mathbb{Q}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}}.\ (\mathit{rat\text{-}of\text{-}vrat}\ (x(\!|0_{\mathbb{N}}|\!)) * \mathit{rat\text{-}of\text{-}vrat}\ (x(\!|1_{\mathbb{N}}|\!)))_{\mathbb{Q}})$

abbreviation *vrat-mult-app* (**infixl** $\langle *_{\mathbb{Q}} \rangle$ 65)
where *vrat-mult-app a b* ≡ *vrat-mult*$(\!|a, b|\!)_\bullet$

lemma *vrat-mult-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat* ===> *cr-vrat*) $(*_{\mathbb{Q}})$ $(*)$
using *vrat-fpair-in-fproduct-vrat*
by (*intro rel-funI*, *unfold vrat-mult-def cr-vrat-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Unary minus.

definition *vrat-uminus* :: *V*
where *vrat-uminus* = $(\lambda x \in_\circ \mathbb{Q}_\circ.\ (\mathit{uminus}\ (\mathit{rat\text{-}of\text{-}vrat}\ x))_{\mathbb{Q}})$

abbreviation *vrat-uminus-app* $(\langle -_{\mathbb{Q}}\ \text{-}\rangle\ [81]\ 80)$
where $-_{\mathbb{Q}}\ a \equiv \mathit{vrat\text{-}uminus}(\!|a|\!)$

lemma *vrat-uminus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat*) (*vrat-uminus-app*) (*uminus*)
using *vrat-fsingleton-in-fproduct-vrat*
by (*intro rel-funI*, *unfold vrat-uminus-def cr-vrat-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Multiplicative inverse.

definition *vrat-inverse* :: *V*
where *vrat-inverse* = $(\lambda x \in_\circ \mathbb{Q}_\circ.\ (\mathit{inverse}\ (\mathit{rat\text{-}of\text{-}vrat}\ x))_{\mathbb{Q}})$

abbreviation *vrat-inverse-app* $(\langle (\text{-}^{-1}{}_{\mathbb{Q}}) \rangle\ [1000]\ 999)$
where $a^{-1}{}_{\mathbb{Q}} \equiv \mathit{vrat\text{-}inverse}(\!|a|\!)$

lemma *vrat-inverse-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat*) (*vrat-inverse-app*) (*inverse*)
using *vrat-fsingleton-in-fproduct-vrat*
by (*intro rel-funI*, *unfold vrat-inverse-def cr-vrat-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Order.

definition *vrat-le* :: *V*
where *vrat-le* =
set $\{[a, b]_\circ \mid a\ b.\ [a, b]_\circ \in_\circ \mathbb{Q}_\circ \mathbin{\widehat{}}_\times 2_{\mathbb{N}} \wedge \mathit{rat\text{-}of\text{-}vrat}\ a \leq \mathit{rat\text{-}of\text{-}vrat}\ b\}$

abbreviation *vrat-le'* $(\langle (\text{-}/ \leq_{\mathbb{Q}}\ \text{-}) \rangle\ [51, 51]\ 50)$
where $a \leq_{\mathbb{Q}} b \equiv [a, b]_\circ \in_\circ \mathit{vrat\text{-}le}$

lemma *small-vrat-le*[*simp*]:
small {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Q}_\circ$ $\widehat{\times}$ $2_\mathbb{N}$ $\wedge$ *rat-of-vrat a* $\leq$ *rat-of-vrat b*}
proof–
have *small*: *small* {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Q}_\circ$ $\widehat{\times}$ $2_\mathbb{N}$} **by** *simp*
show *?thesis* **by** (*rule smaller-than-small*[*OF small*]) *auto*
qed

lemma *vrat-le-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat* ===> (=)) *vrat-le′* (≤)
using *vrat-fsingleton-in-fproduct-vrat*
by (*intro rel-funI*, *unfold cr-scalar-def cr-vrat-def vrat-le-def*)
(*auto simp*: *nat-omega-simps*)

Strict order.

definition *vrat-ls* :: *V*
where *vrat-ls* =
set {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Q}_\circ$ $\widehat{\times}$ $2_\mathbb{N}$ $\wedge$ *rat-of-vrat a* < *rat-of-vrat b*}

abbreviation *vrat-ls′* (‹(-/ <$_\mathbb{Q}$ -)› [51, 51] 50)
where *a* <$_\mathbb{Q}$ *b* ≡ [*a*, *b*]$_\circ$ $\in_\circ$ *vrat-ls*

lemma *small-vrat-ls*[*simp*]:
small {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Q}_\circ$ $\widehat{\times}$ $2_\mathbb{N}$ $\wedge$ *rat-of-vrat a* < *rat-of-vrat b*}
proof–
have *small*: *small* {[*a*, *b*]$_\circ$ | *a b*. [*a*, *b*]$_\circ$ $\in_\circ$ $\mathbb{Q}_\circ$ $\widehat{\times}$ $2_\mathbb{N}$} **by** *simp*
show *?thesis* **by** (*rule smaller-than-small*[*OF small*]) *auto*
qed

lemma *vrat-ls-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat* ===> (=)) *vrat-ls′* (<)
by (*intro rel-funI*, *unfold cr-scalar-def cr-vrat-def vrat-ls-def*)
(*auto simp*: *nat-omega-simps*)

Subtraction.

definition *vrat-minus* :: *V*
where *vrat-minus* =
(λ*x*∈$_\circ$$\mathbb{Q}_\circ$ $\widehat{\times}$ $2_\mathbb{N}$. (*rat-of-vrat* (*x*⦇$0_\mathbb{N}$⦈) − *rat-of-vrat* (*x*⦇$1_\mathbb{N}$⦈))$_\mathbb{Q}$)

abbreviation *vrat-minus-app* (**infixl** ‹−$_\mathbb{Q}$› 65)
where *vrat-minus-app a b* ≡ *vrat-minus*⦇*a*, *b*⦈.

lemma *vrat-minus-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows (*cr-vrat* ===> *cr-vrat* ===> *cr-vrat*)
(−$_\mathbb{Q}$) (−)
using *vrat-fpair-in-fproduct-vrat*
by (*intro rel-funI*, *unfold vrat-minus-def cr-vrat-def cr-scalar-def*)
(*simp add*: *nat-omega-simps*)

Axioms of an ordered field

The exposition follows Theorem 1.5.5 from the textbook *The Real Numbers and Real Analysis* by E. Bloch [14].

lemma *vrat-zero-closed*: $0_\mathbb{Q}$ $\in_\circ$ $\mathbb{Q}_\circ$ **by** *auto*

lemma *vrat-one-closed*: $1_{\mathbb{Q}} \in_\circ \mathbb{Q}_\circ$ **by** *auto*

lemma *vrat-plus-closed*:
assumes $x \in_\circ \mathbb{Q}_\circ$ $y \in_\circ \mathbb{Q}_\circ$
shows $x +_{\mathbb{Q}} y \in_\circ \mathbb{Q}_\circ$
proof–
have $x' + y' \in$ *UNIV* **for** x' y' :: *rat* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vrat-mult-closed*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$
shows $x *_{\mathbb{Q}} y \in_\circ \mathbb{Q}_\circ$
proof–
have $(x'::rat) * y' \in$ *UNIV* **for** x' y' **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vrat-uminus-closed*:
assumes $x \in_\circ \mathbb{Q}_\circ$
shows $-_{\mathbb{Q}} x \in_\circ \mathbb{Q}_\circ$
proof–
have $(-x'::rat) \in$ *UNIV* **for** x' **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vrat-inverse-closed*:
assumes $x \in_\circ \mathbb{Q}_\circ$
shows ${x^{-1}}_{\mathbb{Q}} \in_\circ \mathbb{Q}_\circ$
proof–
have *inverse* $(x'::rat) \in$ *UNIV* **for** x' **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Associative Law for Addition: Theorem 1.5.5.1.

lemma *vrat-assoc-law-addition*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$ **and** $z \in_\circ \mathbb{Q}_\circ$
shows $(x +_{\mathbb{Q}} y) +_{\mathbb{Q}} z = x +_{\mathbb{Q}} (y +_{\mathbb{Q}} z)$
proof–
have $(x' + y') + z' = x' + (y' + z')$ **for** x' y' z' :: *rat* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Commutative Law for Addition: Theorem 1.5.5.2.

lemma *vrat-commutative-law-addition*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$
shows $x +_{\mathbb{Q}} y = y +_{\mathbb{Q}} x$
proof–
have $x' + y' = y' + x'$ **for** x' y' :: *rat* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Identity Law for Addition: Theorem 1.5.5.3.

lemma *vrat-identity-law-addition*:
assumes [*simp*]: $x \in_\circ \mathbb{Q}_\circ$
shows $x +_{\mathbb{Q}} 0_{\mathbb{Q}} = x$

proof–
have $x' + 0 = x'$ **for** $x' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Inverses Law for Addition: Theorem 1.5.5.4.

lemma *vrat-inverses-law-addition*:
assumes [*simp*]: $x \in_\circ \mathbb{Q}_\circ$
shows $x +_\mathbb{Q} (-_\mathbb{Q} x) = 0_\mathbb{Q}$
proof–
have $x' + (-x') = 0$ **for** $x' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Associative Law for Multiplication: Theorem 1.5.5.5.

lemma *vrat-assoc-law-multiplication*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$ **and** $z \in_\circ \mathbb{Q}_\circ$
shows $(x *_\mathbb{Q} y) *_\mathbb{Q} z = x *_\mathbb{Q} (y *_\mathbb{Q} z)$
proof–
have $(x' * y') * z' = x' * (y' * z')$ **for** $x' y' z' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Commutative Law for Multiplication: Theorem 1.5.5.6.

lemma *vrat-commutative-law-multiplication*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$
shows $x *_\mathbb{Q} y = y *_\mathbb{Q} x$
proof–
have $x' * y' = y' * x'$ **for** $x' y' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Identity Law for multiplication: Theorem 1.5.5.7.

lemma *vrat-identity-law-multiplication*:
assumes $x \in_\circ \mathbb{Q}_\circ$
shows $x *_\mathbb{Q} 1_\mathbb{Q} = x$
proof–
have $x' * 1 = x'$ **for** $x' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Inverses Law for Multiplication: Definition 2.2.1.8.

lemma *vrat-inverses-law-multiplication*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $x \neq 0_\mathbb{Q}$
shows $x *_\mathbb{Q} x^{-1}{}_\mathbb{Q} = 1_\mathbb{Q}$
proof–
have $x' \neq 0 \Longrightarrow x' * inverse\ x' = 1$ **for** $x' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Distributive Law for Multiplication: Theorem 1.5.5.9.

lemma *vrat-distributive-law*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$ **and** $z \in_\circ \mathbb{Q}_\circ$
shows $x *_\mathbb{Q} (y +_\mathbb{Q} z) = (x *_\mathbb{Q} y) +_\mathbb{Q} (x *_\mathbb{Q} z)$
proof–
have $x' * (y' + z') = (x' * y') + (x' * z')$ **for** $x' y' z' :: rat$

by (*simp add*: *algebra-simps*)
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Trichotomy Law: Theorem 1.5.5.10.

lemma *vrat-trichotomy-law*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$
shows
$(x <_{\mathbb{Q}} y \wedge \sim(x = y) \wedge \sim(y <_{\mathbb{Q}} x)) \vee$
$(\sim(x <_{\mathbb{Q}} y) \wedge x = y \wedge \sim(y <_{\mathbb{Q}} x)) \vee$
$(\sim(x <_{\mathbb{Q}} y) \wedge \sim(x = y) \wedge y <_{\mathbb{Q}} x)$
proof–
have
$(x' < y' \wedge \sim(x' = y') \wedge \sim(y' < x')) \vee$
$(\sim(x' < y') \wedge x' = y' \wedge \sim(y' < x')) \vee$
$(\sim(x' < y') \wedge \sim(x' = y') \wedge y' < x')$
for x' y' z' :: *rat*
by *auto*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Transitive Law: Theorem 1.5.5.11.

lemma *vrat-transitive-law*:
assumes $x \in_\circ \mathbb{Q}_\circ$
and $y \in_\circ \mathbb{Q}_\circ$
and $z \in_\circ \mathbb{Q}_\circ$
and $x <_{\mathbb{Q}} y$
and $y <_{\mathbb{Q}} z$
shows $x <_{\mathbb{Q}} z$
proof–
have $x' < y' \Longrightarrow y' < z' \Longrightarrow x' < z'$ **for** x' y' z' :: *rat* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Addition Law of Order: Theorem 1.5.5.12.

lemma *vrat-addition-law-of-order*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$ **and** $z \in_\circ \mathbb{Q}_\circ$ **and** $x <_{\mathbb{Q}} y$
shows $x +_{\mathbb{Q}} z <_{\mathbb{Q}} y +_{\mathbb{Q}} z$
proof–
have $x' < y' \Longrightarrow x' + z' < y' + z'$ **for** x' y' z' :: *rat* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Multiplication Law of Order: Theorem 1.5.5.13.

lemma *vrat-multiplication-law-of-order*:
assumes $x \in_\circ \mathbb{Q}_\circ$
and $y \in_\circ \mathbb{Q}_\circ$
and $z \in_\circ \mathbb{Q}_\circ$
and $x <_{\mathbb{Q}} y$
and $0_{\mathbb{Q}} <_{\mathbb{Q}} z$
shows $x *_{\mathbb{Q}} z <_{\mathbb{Q}} y *_{\mathbb{Q}} z$
proof–
have $x' < y' \Longrightarrow 0 < z' \Longrightarrow x' * z' < y' * z'$ **for** x' y' z' :: *rat* **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Non-Triviality: Theorem 1.5.5.14.

lemma *vrat-non-triviality*: $0_{\mathbb{Q}} \neq 1_{\mathbb{Q}}$
proof–
have $0 \neq (1{::}rat)$ **by** *simp*
from *this*[*untransferred*] **show** *?thesis*.
qed

Fundamental properties of other operations

Minus.

lemma *vrat-minus-closed*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$
shows $x -_{\mathbb{Q}} y \in_\circ \mathbb{Q}_\circ$
proof–
have $x' - y' \in UNIV$ **for** $x'\ y' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

lemma *vrat-minus-eq-plus-uminus*:
assumes $x \in_\circ \mathbb{Q}_\circ$ **and** $y \in_\circ \mathbb{Q}_\circ$
shows $x -_{\mathbb{Q}} y = x +_{\mathbb{Q}} (-_{\mathbb{Q}}\ y)$
proof–
have $x' - y' = x' + (-y')$ **for** $x'\ y' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Unary minus.

lemma *vrat-uminus-uminus*:
assumes $x \in_\circ \mathbb{Q}_\circ$
shows $x = -_{\mathbb{Q}} (-_{\mathbb{Q}}\ x)$
proof–
have $x' = -(-x')$ **for** $x' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Multiplicative inverse.

lemma *vrat-inverse-inverse*:
assumes $x \in_\circ \mathbb{Q}_\circ$
shows $x = (x^{-1}{}_{\mathbb{Q}})^{-1}{}_{\mathbb{Q}}$
proof–
have $x' = inverse\ (inverse\ x')$ **for** $x' :: rat$ **by** *simp*
from *this*[*untransferred*, *OF assms*] **show** *?thesis*.
qed

Further properties

Addition.

global-interpretation *vrat-plus*: *binop-onto* ‹$\mathbb{Q}_\circ$› *vrat-plus*
proof–
have *binop*: *binop* $\mathbb{Q}_\circ$ *vrat-plus*
proof(*intro binopI nopI*)
show *vsv*: *vsv vrat-plus* **unfolding** *vrat-plus-def* **by** *auto*
interpret *vsv vrat-plus* **by** (*rule vsv*)
show $2_{\mathbb{N}} \in_\circ \omega$ **by** *simp*
show *dom*: $\mathcal{D}_\circ$ *vrat-plus* = $\mathbb{Q}_\circ \mathbin{\widehat{}_\times} 2_{\mathbb{N}}$ **unfolding** *vrat-plus-def* **by** *simp*
show $\mathcal{R}_\circ$ *vrat-plus* $\subseteq_\circ \mathbb{Q}_\circ$
proof(*intro vsubsetI*)

```
    fix y assume y ∈∘ ℛ∘ vrat-plus
    then obtain ab where ab ∈∘ ℚ∘ ^× 2ℕ and y-def: y = vrat-plus(|ab|)
      unfolding dom[symmetric] by force
    then obtain a b
      where ab-def: ab = [a, b]∘ and a: a ∈∘ ℚ∘ and b: b ∈∘ ℚ∘
      by blast
    then show y ∈∘ ℚ∘ by (simp add: vrat-plus-closed y-def)
  qed
 qed
 interpret binop ‹ℚ∘› vrat-plus by (rule binop)
 show binop-onto ℚ∘ vrat-plus
 proof(intro binop-ontoI′)
  show binop ℚ∘ vrat-plus by (rule binop-axioms)
  show ℚ∘ ⊆∘ ℛ∘ vrat-plus
  proof(intro vsubsetI)
    fix y assume prems: y ∈∘ ℚ∘
    moreover from vrat-zero vrat-zero-closed have 0: 0 ∈∘ ℚ∘
      by auto
    ultimately have y +ℚ 0 ∈∘ ℛ∘ vrat-plus by auto
    moreover from prems vrat-identity-law-addition have y = y +ℚ 0
      by (simp add: vrat-zero)
    ultimately show y ∈∘ ℛ∘ vrat-plus by simp
  qed
 qed
qed
```

Unary minus.

```
global-interpretation vrat-uminus: v11 vrat-uminus
 rewrites vrat-uminus-vdomain[simp]: 𝒟∘ vrat-uminus = ℚ∘
  and vrat-uminus-vrange[simp]: ℛ∘ vrat-uminus = ℚ∘
proof–
 show v11: v11 vrat-uminus
 proof(intro v11I)
  show vsv: vsv vrat-uminus unfolding vrat-uminus-def by simp
  interpret vsv vrat-uminus by (rule vsv)
  show vsv (vrat-uminus⁻¹∘)
  proof(intro vsvI)
    show vbrelation (vrat-uminus⁻¹∘) by clarsimp
    fix a b c
    assume prems: ⟨a, b⟩ ∈∘ vrat-uminus⁻¹∘ ⟨a, c⟩ ∈∘ vrat-uminus⁻¹∘
    then have ba: ⟨b, a⟩ ∈∘ vrat-uminus and ca: ⟨c, a⟩ ∈∘ vrat-uminus
      by auto
    then have b: b ∈∘ ℚ∘ and c: c ∈∘ ℚ∘
      by (simp-all add: VLambda-iff2 vrat-uminus-def)
    from ba ca have a = −ℚ b a = −ℚ c by simp-all
    with ba ca b c show b = c  by (metis vrat-uminus-uminus)
  qed
 qed
 interpret v11 vrat-uminus by (rule v11)
 show dom: 𝒟∘ vrat-uminus = ℚ∘ unfolding vrat-uminus-def by simp
 have ℛ∘ vrat-uminus ⊆∘ ℚ∘
 proof(intro vsubsetI)
  fix y assume y ∈∘ ℛ∘ vrat-uminus
  then obtain x where x ∈∘ ℚ∘ and y-def: y = −ℚ x
    unfolding dom[symmetric] by force
  then show y ∈∘ ℚ∘ by (simp add: vrat-uminus-closed)
 qed
 moreover have ℚ∘ ⊆∘ ℛ∘ vrat-uminus
```

```
    by (intro vsubsetI)
      (metis dom vdomain-atD vrat-uminus-closed vrat-uminus-uminus)
  ultimately show ℛ∘ vrat-uminus = ℚ∘ by simp
qed
```

Multiplication.

```
global-interpretation vrat-mult: binop-onto ‹ℚ∘› vrat-mult
proof–
  have binop: binop ℚ∘ vrat-mult
  proof(intro binopI nopI)
    show vsv: vsv vrat-mult unfolding vrat-mult-def by auto
    interpret vsv vrat-mult by (rule vsv)
    show 2ℕ ∈∘ ω by simp
    show dom: 𝒟∘ vrat-mult = ℚ∘ ^× 2ℕ unfolding vrat-mult-def by simp
    show ℛ∘ vrat-mult ⊆∘ ℚ∘
    proof(intro vsubsetI)
      fix y assume y ∈∘ ℛ∘ vrat-mult
      then obtain ab where ab ∈∘ ℚ∘ ^× 2ℕ and y-def: y = vrat-mult(|ab|)
        unfolding dom[symmetric] by force
      then obtain a b
        where ab-def: ab = [a, b]∘ and a: a ∈∘ ℚ∘ and b: b ∈∘ ℚ∘
        by blast
      then show y ∈∘ ℚ∘ by (simp add: vrat-mult-closed y-def)
    qed
  qed
  interpret binop ‹ℚ∘› vrat-mult by (rule binop)
  show binop-onto ℚ∘ vrat-mult
  proof(intro binop-ontoI′)
    show binop ℚ∘ vrat-mult by (rule binop-axioms)
    show ℚ∘ ⊆∘ ℛ∘ vrat-mult
    proof(intro vsubsetI)
      fix y assume prems: y ∈∘ ℚ∘
      moreover from vrat-one vrat-one-closed have 1 ∈∘ ℚ∘ by auto
      ultimately have y *ℚ 1 ∈∘ ℛ∘ vrat-mult by auto
      moreover from prems vrat-identity-law-multiplication have y = y *ℚ 1
        by (simp add: vrat-one)
      ultimately show y ∈∘ ℛ∘ vrat-mult by simp
    qed
  qed
qed
```

Multiplicative inverse.

```
global-interpretation vrat-inverse: v11 vrat-inverse
  rewrites vrat-inverse-vdomain[simp]: 𝒟∘ vrat-inverse = ℚ∘
    and vrat-inverse-vrange[simp]: ℛ∘ vrat-inverse = ℚ∘
proof–
  show v11: v11 vrat-inverse
  proof(intro v11I)
    show vsv: vsv vrat-inverse unfolding vrat-inverse-def by simp
    interpret vsv vrat-inverse by (rule vsv)
    show vsv (vrat-inverse⁻¹∘)
    proof(intro vsvI)
      show vbrelation (vrat-inverse⁻¹∘) by clarsimp
      fix a b c
      assume prems: ⟨a, b⟩ ∈∘ vrat-inverse⁻¹∘ ⟨a, c⟩ ∈∘ vrat-inverse⁻¹∘
      then have ba: ⟨b, a⟩ ∈∘ vrat-inverse and ca: ⟨c, a⟩ ∈∘ vrat-inverse
        by auto
      then have b: b ∈∘ ℚ∘ and c: c ∈∘ ℚ∘
```

```
      by (simp-all add: VLambda-iff2 vrat-inverse-def)
    from ba ca have a = b⁻¹_Q a = c⁻¹_Q by simp-all
    with ba ca b c show b = c  by (metis vrat-inverse-inverse)
  qed
 qed
 interpret v11 vrat-inverse by (rule v11)
 show dom: 𝒟₀ vrat-inverse = ℚ₀ unfolding vrat-inverse-def by simp
 have ℛ₀ vrat-inverse ⊆₀ ℚ₀
 proof(intro vsubsetI)
  fix y assume y ∈₀ ℛ₀ vrat-inverse
  then obtain x where x ∈₀ ℚ₀ and y-def: y = x⁻¹_Q
    unfolding dom[symmetric] by force
  then show y ∈₀ ℚ₀ by (simp add: vrat-inverse-closed)
 qed
 moreover have ℚ₀ ⊆₀ ℛ₀ vrat-inverse
  by (intro vsubsetI)
   (metis dom vdomain-atD vrat-inverse-closed vrat-inverse-inverse)
 ultimately show ℛ₀ vrat-inverse = ℚ₀ by simp
qed
```

Misc.

```
lemma (in 𝒵) vrat-in-Vset[intro]: ℚ₀ ∈₀ Vset α
 using vrat-in-Vset-ω2 vsubsetD by (auto intro!: 𝒵-Vset-ω2-vsubset-Vset)
```

2.13.5 Upper bound on the cardinality of the continuum for V

```
lemma inj-on-inv-vreal-of-real: inj-on (inv vreal-of-real) (elts ℝ₀)
 by (intro inj-onI) (fastforce intro: inv-into-injective)

lemma vreal-vlepoll-VPow-omega: ℝ₀ ≲₀ VPow ω
proof–
 have elts ℝ₀ ≲ (UNIV::real set)
  unfolding lepoll-def by (auto intro: inj-on-inv-vreal-of-real)
 from vlepoll-VPow-omega-if-vreal-lepoll-real[OF this] show ?thesis by simp
qed

lemma (in 𝒵) vreal-in-Vset[intro]: ℝ₀ ∈₀ Vset α
 using vreal-in-Vset-ω2 vsubsetD by (auto intro!: 𝒵-Vset-ω2-vsubset-Vset)
```

2.14 Example I: absence of replacement in $V_{\omega+\omega}$

The statement of the main result presented in this subsection can be found in [5][3]

definition *repl-ex-fun* :: *V*
where *repl-ex-fun* = (λ*i*∈∘ω. *Vfrom* ω *i*)

mk-VLambda *repl-ex-fun-def*
|*vsv repl-ex-fun-vsv*|
|*vdomain repl-ex-fun-vdomain*|
|*app repl-ex-fun-app*|

lemma *repl-ex-fun-vrange*: $\mathcal{R}_\circ$ *repl-ex-fun* ⊆∘ *Vset* (ω + ω)
proof(*intro vsv.vsv-vrange-vsubset*, *unfold repl-ex-fun-vdomain*)
fix *x* **assume** *prems*: *x* ∈∘ ω
then show *repl-ex-fun*(|*x*|) ∈∘ *Vset* (ω + ω)
proof(*induction rule*: *omega-induct*)
case 0 **then show** *?case*
by
(
auto
simp: *repl-ex-fun-app intro*!: *vreal-in-Vset-ω2 omega-vsubset-vreal*
)
next
case (*succ n*)
then have *Ord-n*: *Ord n* **by** *auto*
have *Limit-ωω*: *Limit* (ω + ω) **by** *auto*
from *succ* **show** *?case*
by
(
auto
simp: *Vfrom-succ-Ord*[*OF Ord-n*, *of* ω] *repl-ex-fun-app*
intro: *Limit-ωω*
intro!: *omega-vsubset-vreal vreal-in-Vset-ω2*
)
qed
qed (*unfold repl-ex-fun-def*, *auto*)

lemma *Limit-vsv-not-in-Vset-if-vrange-not-in-Vset*:
assumes *Limit* α **and** $\mathcal{R}_\circ$ *f* ∉∘ *Vset* α
shows *f* ∉∘ *Vset* α
proof(*rule ccontr*, *unfold not-not*)
assume *f* ∈∘ *Vset* α
with *assms*(1) **have** $\mathcal{R}_\circ$ *f* ∈∘ *Vset* α **by** (*simp add*: *vrange-in-VsetI*)
with *assms*(2) **show** *False* **by** *simp*
qed

lemma *Ord-not-in-Vset*:
assumes *Ord* α
shows α ∉∘ *Vset* α
using *assms*
proof(*induction rule*: *Ord-induct3′*)
case (*succ* α)
then have *succα*: *Vset* (*succ* α) = *VPow* (*Vset* α) **by** (*simp add*: *Vset-succ*)
show *?case*
proof(*rule ccontr*, *unfold not-not*)
assume *succ* α ∈∘ *Vset* (*succ* α)

[3] https://en.wikipedia.org/wiki/Zermelo_set_theory

```
    then have vinsert α α ∈∘ VPow (Vset α)
      unfolding succα by (simp add: succ-def)
    with succ(2) show False by auto
  qed
next
  case (Limit α) show ?case
  proof(rule ccontr, unfold not-not)
    assume (⋃∘ξ∈∘α. ξ) ∈∘ Vset (⋃∘ξ∈∘α. ξ)
    with Limit(1) have α ∈∘ Vset α by auto
    with Limit(1) obtain i where i: i ∈∘ α and (⋃∘ξ∈∘α. ξ) ∈∘ Vset i
      by (metis Limit-Vfrom-eq Limit-vifunion-def vifunion-iff)
    moreover with Limit(1) have α ∈∘ Vset i by auto
    ultimately have i ∈∘ Vset i by auto
    with Limit(2)[OF i] show False by auto
  qed
qed simp

lemma Ord-succ-vsusbset-Vfrom-succ:
  assumes Transset A and Ord a and a ∈∘ Vfrom A i
  shows succ a ⊆∘ Vfrom A (succ i)
proof(intro vsubsetI)
  from Vfrom-in-mono[OF vsubset-reflexive] have i-succi:
    Vfrom A i ∈∘ Vfrom A (succ i)
    by simp
  fix x assume prems: x ∈∘ succ a
  then consider ‹x ∈∘ a› | ‹x = a› unfolding succ-def by auto
  then show x ∈∘ Vfrom A (succ i)
  proof cases
    case 1
    have x ∈∘ Vfrom A i by (rule Vfrom-trans[OF assms(1) 1 assms(3)])
    then show x ∈∘ Vfrom A (succ i) by (rule Vfrom-trans[OF assms(1) - i-succi])
  next
    case 2 from assms(3) show ?thesis
      unfolding 2 by (intro Vfrom-trans[OF assms(1) - i-succi])
  qed
qed

lemma Ord-succ-in-Vfrom-succ:
  assumes Transset A and Ord a and a ∈∘ Vfrom A i
  shows succ a ∈∘ Vfrom A (succ (succ i))
  using Ord-succ-vsusbset-Vfrom-succ[OF assms] by (simp add: Vfrom-succ)

lemma ω-vplus-in-Vfrom-ω:
  assumes j ∈∘ ω
  shows ω + j ∈∘ Vfrom ω (succ (2ₙ * j))
  using assms
proof(induction rule: omega-induct)
  case 0
  have ω ∈∘ Vfrom ω (succ 0)
    unfolding Vfrom-succ-Ord[where i=0, simplified] by auto
  then show ?case by simp
next
  case (succ n)
  from succ(1) obtain m where n-def: n = mₙ by (auto elim: nat-of-omega)
  from succ(1) have ω-succn: ω + succ n = succ (ω + n) by (simp add: plus-V-succ-right)
  from succ(1) have succ-2succn: succ (2ₙ * succ n) = succ (succ (succ (2ₙ * n)))
    unfolding n-def by (cs-concl-step nat-omega-simps)+ auto
  show ?case
```

```
    unfolding ω-succn succ-2succn
    by (intro Ord-succ-in-Vfrom-succ succ)
      (auto simp: succ(1) intro: Ord-is-Transset)
qed

lemma repl-ex-fun-vrange-not-in-Vset: ℛ∘ repl-ex-fun ∉∘ Vset (ω + ω)
proof(rule ccontr, unfold not-not)
  assume prems: ℛ∘ repl-ex-fun ∈∘ Vset (ω + ω)
  then have ⋃∘(ℛ∘ repl-ex-fun) ∈∘ Vset (ω + ω) by (simp add: VUnion-in-VsetI)
  moreover have ω + ω ⊆∘ ⋃∘(ℛ∘ repl-ex-fun)
  proof(intro vsubsetI)
    fix x assume prems: x ∈∘ ω + ω
    from prems consider ‹x ∈∘ ω› | ‹x ∉∘ ω› by auto
    then show x ∈∘ ⋃∘(ℛ∘ repl-ex-fun)
    proof cases
      case 1
      show ?thesis
      proof(rule VUnionI)
        show Vfrom ω 0 ∈∘ ℛ∘ repl-ex-fun
          unfolding repl-ex-fun-def by blast
        from 1 show x ∈∘ Vfrom ω 0 by auto
      qed
    next
      case 2
      with prems obtain j where x-def: x = ω + j and j: j ∈∘ ω
        by (auto elim: mem-plus-V-E)
      show ?thesis
      proof(rule VUnionI)
        from j show Vfrom ω (succ (2ℕ ∗ j)) ∈∘ ℛ∘ repl-ex-fun
          unfolding repl-ex-fun-def by blast
        show x ∈∘ Vfrom ω (succ (2ℕ ∗ j))
          by (rule ω-vplus-in-Vfrom-ω[OF j, folded x-def])
      qed
    qed
  qed
  ultimately have ω + ω ∈∘ Vset (ω + ω) by auto
  with Ord-not-in-Vset show False by auto
qed

lemma repl-ex-fun-not-in-Vset: repl-ex-fun ∉∘ Vset (ω + ω)
  by (rule Limit-vsv-not-in-Vset-if-vrange-not-in-Vset)
    (auto simp: repl-ex-fun-vrange-not-in-Vset)
```

2.15 Example II: topological spaces

2.15.1 Background

The section presents elements of the foundations of the theory of topological spaces formalized in *ZFC in HOL*. The definitions were adopted (with amendments) from the main library of Isabelle/HOL and [35].

named-theorems *ts-struct-field-simps*

2.15.2 $\mathcal{Z}$-sequence

locale $\mathcal{Z}$*-vfsequence* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{S}$ **for** α $\mathfrak{S}$ +
  assumes *vrange-vsubset-Vset*: $\mathcal{R}_\circ$ $\mathfrak{S}$ $\subseteq_\circ$ *Vset* α

Rules.

lemma $\mathcal{Z}$*-vfsequenceI*[*intro*]:
  assumes $\mathcal{Z}$ α **and** *vfsequence* $\mathfrak{S}$ **and** $\mathcal{R}_\circ$ $\mathfrak{S}$ $\subseteq_\circ$ *Vset* α
  shows $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$
  using *assms* **unfolding** $\mathcal{Z}$*-vfsequence-def* $\mathcal{Z}$*-vfsequence-axioms-def* **by** *simp*

lemmas $\mathcal{Z}$*-vfsequenceD*[*dest*] = $\mathcal{Z}$*-vfsequence.axioms*

lemma $\mathcal{Z}$*-vfsequenceE*[*elim*]:
  assumes $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$
  obtains $\mathcal{Z}$ α **and** *vfsequence* $\mathfrak{S}$ **and** $\mathcal{R}_\circ$ $\mathfrak{S}$ $\subseteq_\circ$ *Vset* α
  using *assms* **by** (*simp add*: $\mathcal{Z}$*-vfsequence.axioms*(1,2) $\mathcal{Z}$*-vfsequence.vrange-vsubset-Vset*)

Elementary properties.

context $\mathcal{Z}$*-vfsequence*
begin

lemma (**in** $\mathcal{Z}$*-vfsequence*) $\mathcal{Z}$*-vfsequence-vdomain-in-Vset*[*intro*, *simp*]:
  $\mathcal{D}_\circ$ $\mathfrak{S}$ $\in_\circ$ *Vset* α
  using *Axiom-of-Infinity vfsequence-vdomain-in-omega* **by** *auto*

lemma (**in** $\mathcal{Z}$*-vfsequence*) $\mathcal{Z}$*-vfsequence-vrange-in-Vset*[*intro*, *simp*]:
  $\mathcal{R}_\circ$ $\mathfrak{S}$ $\in_\circ$ *Vset* α
  using *vrange-vsubset-Vset vfsequence-vdomain-in-omega* **by** *auto*

lemma (**in** $\mathcal{Z}$*-vfsequence*) $\mathcal{Z}$*-vfsequence-struct-in-Vset*: $\mathfrak{S}$ $\in_\circ$ *Vset* α
  by (*auto simp*: *vrange-vsubset-Vset vsv-Limit-vsv-in-VsetI*)

end

2.15.3 Topological space

definition $\mathcal{A}$ **where** [*ts-struct-field-simps*]: $\mathcal{A}$ = 0
definition $\mathcal{T}$ **where** [*ts-struct-field-simps*]: $\mathcal{T}$ = $1_{\mathbb{N}}$

locale $\mathcal{Z}$*-ts* = $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$ **for** α $\mathfrak{S}$ +
  assumes $\mathcal{Z}$*-ts-length*: $2_{\mathbb{N}}$ $\leq$ *vcard* $\mathfrak{S}$
    and $\mathcal{Z}$*-ts-closed*[*intro*]: A $\in_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$ $\Longrightarrow$ A $\subseteq_\circ$ $\mathfrak{S}(\lfloor\mathcal{A}\rfloor)$
    and $\mathcal{Z}$*-ts-domain*[*intro*, *simp*]: $\mathfrak{S}(\lfloor\mathcal{A}\rfloor)$ $\in_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$
    and $\mathcal{Z}$*-ts-vintersection*[*intro*]:
      A $\in_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$ $\Longrightarrow$ B $\in_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$ $\Longrightarrow$ A $\cap_\circ$ B $\in_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$
    and $\mathcal{Z}$*-ts-VUnion*[*intro*]: X $\subseteq_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$ $\Longrightarrow$ $\bigcup_\circ X$ $\in_\circ$ $\mathfrak{S}(\lfloor\mathcal{T}\rfloor)$

Rules.

lemma *Z-tsI*[*intro*]:
assumes *Z-vfsequence* α $\mathfrak{S}$
and $2_{\mathbb{N}} \leq$ *vcard* $\mathfrak{S}$
and $\bigwedge A.\ A \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow A \subseteq_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$
and $\mathfrak{S}(\!|\mathcal{A}|\!) \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
and $\bigwedge A\ B.\ A \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow B \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow A \cap_{\circ} B \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
and $\bigwedge X.\ X \subseteq_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow \bigcup_{\circ} X \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
shows *Z-ts* α $\mathfrak{S}$
using *assms* **unfolding** *Z-ts-def Z-ts-axioms-def* **by** *simp*

lemma *Z-tsD*[*dest*]:
assumes *Z-ts* α $\mathfrak{S}$
shows *Z-vfsequence* α $\mathfrak{S}$
and $2_{\mathbb{N}} \leq$ *vcard* $\mathfrak{S}$
and $\bigwedge A.\ A \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow A \subseteq_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$
and $\mathfrak{S}(\!|\mathcal{A}|\!) \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
and $\bigwedge A\ B.\ A \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow B \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow A \cap_{\circ} B \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
and $\bigwedge X.\ X \subseteq_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow \bigcup_{\circ} X \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
using *assms* **unfolding** *Z-ts-def Z-ts-axioms-def* **by** *auto*

lemma *Z-tsE*[*elim*]:
assumes *Z-ts* α $\mathfrak{S}$
obtains *Z-vfsequence* α $\mathfrak{S}$
and $2_{\mathbb{N}} \leq$ *vcard* $\mathfrak{S}$
and $\bigwedge A.\ A \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow A \subseteq_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$
and $\mathfrak{S}(\!|\mathcal{A}|\!) \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
and $\bigwedge A\ B.\ A \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow B \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow A \cap_{\circ} B \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
and $\bigwedge X.\ X \subseteq_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!) \Longrightarrow \bigcup_{\circ} X \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
using *assms* **by** *auto*

Elementary properties.

lemma (**in** *Z-ts*) *Z-ts-vempty-in-ts*: $0 \in_{\circ} \mathfrak{S}(\!|\mathcal{T}|\!)$
using *Z-ts-VUnion*[*of* 0] **by** *simp*

2.15.4 Indiscrete topology

definition *ts-indiscrete* :: $V \Rightarrow V$
where *ts-indiscrete* A = $[A,\ set\ \{0,\ A\}]_{\circ}$

named-theorems *ts-indiscrete-simps*

lemma *ts-indiscrete-*$\mathcal{A}$[*ts-indiscrete-simps*]: *ts-indiscrete* $A(\!|\mathcal{A}|\!)$ = A
unfolding *ts-indiscrete-def* **by** (*auto simp*: *ts-struct-field-simps*)

lemma *ts-indiscrete-*$\mathcal{T}$[*ts-indiscrete-simps*]: *ts-indiscrete* $A(\!|\mathcal{T}|\!)$ = *set* $\{0,\ A\}$
unfolding *ts-indiscrete-def*
by (*simp add*: *ts-struct-field-simps nat-omega-simps*)

lemma (**in** *Z*) *Z-ts-ts-indiscrete*:
assumes $A \in_{\circ}$ *Vset* α
shows *Z-ts* α (*ts-indiscrete* A)
proof(*intro Z-tsI*)

show *struct*: *Z-vfsequence* α (*ts-indiscrete* A)
proof(*intro Z-vfsequenceI*)
show *vfsequence* (*ts-indiscrete* A) **unfolding** *ts-indiscrete-def* **by** *auto*
show $\mathcal{R}_{\circ}$ (*ts-indiscrete* A) $\subseteq_{\circ}$ *Vset* α
proof(*intro vsubsetI*)

```
    fix x assume x ∈∘ ℛ∘ (ts-indiscrete A)
    then consider ‹x = A› | ‹x = set {0, A}›
      unfolding ts-indiscrete-def by auto
    then show x ∈∘ Vset α by cases (simp-all add: Axiom-of-Pairing assms)
  qed
qed (simp-all add: 𝒵-axioms)

interpret struct: 𝒵-vfsequence α ‹ts-indiscrete A› by (rule struct)

show X ⊆∘ ts-indiscrete A(|𝒯|) ⟹ ⋃∘X ∈∘ ts-indiscrete A(|𝒯|) for X
  unfolding ts-indiscrete-simps
proof–
  assume X ⊆∘ set {0, A}
  then have X ∈∘ VPow (set {0, A}) by force
  then consider ‹X = 0› | ‹X = set {0}› | ‹X = set {A}› | ‹X = set {0, A}›
    by auto
  then show ⋃∘X ∈∘ set {0, A} by cases auto
qed

show 2_ℕ ⊆∘ vcard (ts-indiscrete A) unfolding ts-indiscrete-def by fastforce

qed (auto simp: ts-indiscrete-simps)
```

2.16 Example III: abstract algebra

2.16.1 Background

The section presents several examples of algebraic structures formalized in *ZFC in HOL*. The definitions were adopted (with amendments) from the main library of Isabelle/HOL.

named-theorems *sgrp-struct-field-simps*

lemmas [*sgrp-struct-field-simps*] = $\mathcal{A}$-*def*

2.16.2 Semigroup

Foundations

definition *mbinop* **where** [*sgrp-struct-field-simps*]: *mbinop* = $1_{\mathbf{N}}$

locale $\mathcal{Z}$-*sgrp-basis* = $\mathcal{Z}$-*vfsequence* α $\mathfrak{S}$ + *op*: *binop* ‹$\mathfrak{S}(\!|\mathcal{A}|\!)$› ‹$\mathfrak{S}(\!|$*mbinop*$|\!)$›
  for α $\mathfrak{S}$ +
  assumes $\mathcal{Z}$-*sgrp-length*: *vcard* $\mathfrak{S}$ = $2_{\mathbf{N}}$

abbreviation *sgrp-app* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\odot_{\circ 1}$› 70)
  where *sgrp-app* $\mathfrak{S}$ *a b* ≡ $\mathfrak{S}(\!|$*mbinop*$|\!)(\!|a, b|\!)_{\bullet}$
notation *sgrp-app* (**infixl** ‹$\odot_{\circ}$› 70)

Rules.

lemma $\mathcal{Z}$-*sgrp-basisI*[*intro*]:
  assumes $\mathcal{Z}$-*vfsequence* α $\mathfrak{S}$
    and *vcard* $\mathfrak{S}$ = $2_{\mathbf{N}}$
    and *binop* ($\mathfrak{S}(\!|\mathcal{A}|\!)$) ($\mathfrak{S}(\!|$*mbinop*$|\!)$)
  shows $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$
  using *assms* **unfolding** $\mathcal{Z}$-*sgrp-basis-def* $\mathcal{Z}$-*sgrp-basis-axioms-def* **by** *simp*

lemma $\mathcal{Z}$-*sgrp-basisD*[*dest*]:
  assumes $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$
  shows $\mathcal{Z}$-*vfsequence* α $\mathfrak{S}$
    and *vcard* $\mathfrak{S}$ = $2_{\mathbf{N}}$
    and *binop* ($\mathfrak{S}(\!|\mathcal{A}|\!)$) ($\mathfrak{S}(\!|$*mbinop*$|\!)$)
  using *assms* **unfolding** $\mathcal{Z}$-*sgrp-basis-def* $\mathcal{Z}$-*sgrp-basis-axioms-def* **by** *auto*

lemma $\mathcal{Z}$-*sgrp-basisE*[*elim*]:
  assumes $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$
  shows $\mathcal{Z}$-*vfsequence* α $\mathfrak{S}$
    and *vcard* $\mathfrak{S}$ = $2_{\mathbf{N}}$
    and *binop* ($\mathfrak{S}(\!|\mathcal{A}|\!)$) ($\mathfrak{S}(\!|$*mbinop*$|\!)$)
  using *assms* **unfolding** $\mathcal{Z}$-*sgrp-basis-def* $\mathcal{Z}$-*sgrp-basis-axioms-def* **by** *auto*

Simple semigroup

locale $\mathcal{Z}$-*sgrp* = $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$ **for** α $\mathfrak{S}$ +
  assumes $\mathcal{Z}$-*sgrp-assoc*:
    [[$a \in_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$; $b \in_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$; $c \in_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$]] ⟹
      $(a \odot_{\circ\mathfrak{S}} b) \odot_{\circ\mathfrak{S}} c = a \odot_{\circ\mathfrak{S}} (b \odot_{\circ\mathfrak{S}} c)$

Rules.

lemma $\mathcal{Z}$-*sgrpI*[*intro*]:
  assumes $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$
    and $\bigwedge a\ b\ c$. [[$a \in_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$; $b \in_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$; $c \in_{\circ} \mathfrak{S}(\!|\mathcal{A}|\!)$]] ⟹
      $(a \odot_{\circ\mathfrak{S}} b) \odot_{\circ\mathfrak{S}} c = a \odot_{\circ\mathfrak{S}} (b \odot_{\circ\mathfrak{S}} c)$

shows $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$
using *assms* **unfolding** $\mathcal{Z}$-*sgrp-def* $\mathcal{Z}$-*sgrp-axioms-def* **by** *simp*

lemma $\mathcal{Z}$-*sgrpD*[*dest*]:
assumes $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$
shows $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$
and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ c \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow$
$(a \odot_{\circ\mathfrak{S}} b) \odot_{\circ\mathfrak{S}} c = a \odot_{\circ\mathfrak{S}} (b \odot_{\circ\mathfrak{S}} c)$
using *assms* **unfolding** $\mathcal{Z}$-*sgrp-def* $\mathcal{Z}$-*sgrp-axioms-def* **by** *simp-all*

lemma $\mathcal{Z}$-*sgrpE*[*elim*]:
assumes $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$
obtains $\mathcal{Z}$-*sgrp-basis* α $\mathfrak{S}$
and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ c \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow$
$(a \odot_{\circ\mathfrak{S}} b) \odot_{\circ\mathfrak{S}} c = a \odot_{\circ\mathfrak{S}} (b \odot_{\circ\mathfrak{S}} c)$
using *assms* **by** *auto*

2.16.3 Commutative semigroup

locale $\mathcal{Z}$-*csgrp* = $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$ **for** α $\mathfrak{S}$ +
assumes $\mathcal{Z}$-*csgrp-commutative*:
$[\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow a \odot_{\circ\mathfrak{S}} b = b \odot_{\circ\mathfrak{S}} a$

Rules.

lemma $\mathcal{Z}$-*csgrpI*[*intro*]:
assumes $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow a \odot_{\circ\mathfrak{S}} b = b \odot_{\circ\mathfrak{S}} a$
shows $\mathcal{Z}$-*csgrp* α $\mathfrak{S}$
using *assms* **unfolding** $\mathcal{Z}$-*csgrp-def* $\mathcal{Z}$-*csgrp-axioms-def* **by** *simp*

lemma $\mathcal{Z}$-*csgrpD*[*dest*]:
assumes $\mathcal{Z}$-*csgrp* α $\mathfrak{S}$
shows $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow a \odot_{\circ\mathfrak{S}} b = b \odot_{\circ\mathfrak{S}} a$
using *assms* **unfolding** $\mathcal{Z}$-*csgrp-def* $\mathcal{Z}$-*csgrp-axioms-def* **by** *simp-all*

lemma $\mathcal{Z}$-*csgrpE*[*elim*]:
assumes $\mathcal{Z}$-*csgrp* α $\mathfrak{S}$
obtains $\mathcal{Z}$-*sgrp* α $\mathfrak{S}$
and $\bigwedge a\ b.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow a \odot_{\circ\mathfrak{S}} b = b \odot_{\circ\mathfrak{S}} a$
using *assms* **by** *auto*

2.16.4 Semiring

Foundations

definition *vplus* :: V **where** [*sgrp-struct-field-simps*]: *vplus* = $1_{\mathbf{N}}$
definition *vmult* :: V **where** [*sgrp-struct-field-simps*]: *vmult* = $2_{\mathbf{N}}$

abbreviation *vplus-app* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (**infixl** ‹$+_{\circ 1}$› 65)
where $a +_{\circ\mathfrak{S}} b \equiv \mathfrak{S}(\!|vplus|\!)(\!|a,b|\!)_\bullet$
notation *vplus-app* (**infixl** ‹$+_{\circ 1}$› 65)

abbreviation *vmult-app* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (**infixl** ‹$*_{\circ 1}$› 70)
where $a *_{\circ\mathfrak{S}} b \equiv \mathfrak{S}(\!|vmult|\!)(\!|a,b|\!)_\bullet$
notation *vmult-app* (**infixl** ‹$*_{\circ 1}$› 70)

Simple semiring

locale $\mathcal{Z}$*-srng* = $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$ **for** α $\mathfrak{S}$ +
  assumes $\mathcal{Z}$*-srng-length*: *vcard* $\mathfrak{S}$ = $3_{\mathbb{N}}$
    and $\mathcal{Z}$*-srng-*$\mathcal{Z}$*-csgrp-vplus*: $\mathcal{Z}$*-csgrp* α $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ$
    and $\mathcal{Z}$*-srng-*$\mathcal{Z}$*-sgrp-vmult*: $\mathcal{Z}$*-sgrp* α $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ$
    and $\mathcal{Z}$*-srng-distrib-right*:
      $[\![\ a \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ b \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ c \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor)\ ]\!] \Longrightarrow$
        $(a +_{\circ\mathfrak{S}} b) *_{\circ\mathfrak{S}} c = (a *_{\circ\mathfrak{S}} c) +_{\circ\mathfrak{S}} (b *_{\circ\mathfrak{S}} c)$
    and $\mathcal{Z}$*-srng-distrib-left*:
      $[\![\ a \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ b \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ c \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor)\ ]\!] \Longrightarrow$
        $a *_{\circ\mathfrak{S}} (b +_{\circ\mathfrak{S}} c) = (a *_{\circ\mathfrak{S}} b) +_{\circ\mathfrak{S}} (a *_{\circ\mathfrak{S}} c)$
begin

sublocale *vplus*: $\mathcal{Z}$*-csgrp* α $\langle[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ\rangle$
  rewrites $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ(\lfloor\mathcal{A}\rfloor) = \mathfrak{S}(\lfloor\mathcal{A}\rfloor)$
    and $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ(\lfloor mbinop\rfloor) = \mathfrak{S}(\lfloor vplus\rfloor)$
    and *sgrp-app* $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ$ = *vplus-app* $\mathfrak{S}$
proof(*rule* $\mathcal{Z}$*-srng-*$\mathcal{Z}$*-csgrp-vplus*)
  show $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ(\lfloor\mathcal{A}\rfloor) = \mathfrak{S}(\lfloor\mathcal{A}\rfloor)$
    and [*simp*]: $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ(\lfloor mbinop\rfloor) = \mathfrak{S}(\lfloor vplus\rfloor)$
    by (*auto simp*: $\mathcal{A}$*-def mbinop-def nat-omega-simps*)
  show $(\odot_{\circ[\mathfrak{S}(\lfloor\mathcal{A}\rfloor),\ \mathfrak{S}(\lfloor vplus\rfloor)]_\circ}) = (+_{\circ\mathfrak{S}})$ **by** *simp*
qed

sublocale *vmult*: $\mathcal{Z}$*-sgrp* α $\langle[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ\rangle$
  rewrites $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ(\lfloor\mathcal{A}\rfloor) = \mathfrak{S}(\lfloor\mathcal{A}\rfloor)$
    and $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ(\lfloor mbinop\rfloor) = \mathfrak{S}(\lfloor vmult\rfloor)$
    and *sgrp-app* $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ$ = *vmult-app* $\mathfrak{S}$
proof(*rule* $\mathcal{Z}$*-srng-*$\mathcal{Z}$*-sgrp-vmult*)
  show $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ(\lfloor\mathcal{A}\rfloor) = \mathfrak{S}(\lfloor\mathcal{A}\rfloor)$
    and [*simp*]: $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ(\lfloor mbinop\rfloor) = \mathfrak{S}(\lfloor vmult\rfloor)$
    by (*auto simp*: $\mathcal{A}$*-def mbinop-def nat-omega-simps*)
  show $(\odot_{\circ[\mathfrak{S}(\lfloor\mathcal{A}\rfloor),\ \mathfrak{S}(\lfloor vmult\rfloor)]_\circ}) = (*_{\circ\mathfrak{S}})$ **by** *simp*
qed

end

Rules.

lemma $\mathcal{Z}$*-srngI*[*intro*]:
  assumes $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$
    and *vcard* $\mathfrak{S}$ = $3_{\mathbb{N}}$
    and $\mathcal{Z}$*-csgrp* α $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ$
    and $\mathcal{Z}$*-sgrp* α $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ$
    and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ b \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ c \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor)\ ]\!] \Longrightarrow$
      $(a +_{\circ\mathfrak{S}} b) *_{\circ\mathfrak{S}} c = (a *_{\circ\mathfrak{S}} c) +_{\circ\mathfrak{S}} (b *_{\circ\mathfrak{S}} c)$
    and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ b \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ c \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor)\ ]\!] \Longrightarrow$
      $a *_{\circ\mathfrak{S}} (b +_{\circ\mathfrak{S}} c) = (a *_{\circ\mathfrak{S}} b) +_{\circ\mathfrak{S}} (a *_{\circ\mathfrak{S}} c)$
  shows $\mathcal{Z}$*-srng* α $\mathfrak{S}$
  using *assms* **unfolding** $\mathcal{Z}$*-srng-def* $\mathcal{Z}$*-srng-axioms-def* **by** *simp*

lemma $\mathcal{Z}$*-srngD*[*dest*]:
  assumes $\mathcal{Z}$*-srng* α $\mathfrak{S}$
  shows $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$
    and *vcard* $\mathfrak{S}$ = $3_{\mathbb{N}}$
    and $\mathcal{Z}$*-csgrp* α $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vplus\rfloor)]_\circ$
    and $\mathcal{Z}$*-sgrp* α $[\mathfrak{S}(\lfloor\mathcal{A}\rfloor), \mathfrak{S}(\lfloor vmult\rfloor)]_\circ$
    and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ b \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor);\ c \in_\circ \mathfrak{S}(\lfloor\mathcal{A}\rfloor)\ ]\!] \Longrightarrow$

$(a +_{\circ\mathfrak{S}} b) *_{\circ\mathfrak{S}} c = (a *_{\circ\mathfrak{S}} c) +_{\circ\mathfrak{S}} (b *_{\circ\mathfrak{S}} c)$
and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ c \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow$
$a *_{\circ\mathfrak{S}} (b +_{\circ\mathfrak{S}} c) = (a *_{\circ\mathfrak{S}} b) +_{\circ\mathfrak{S}} (a *_{\circ\mathfrak{S}} c)$
using *assms* **unfolding** $\mathcal{Z}$*-srng-def* $\mathcal{Z}$*-srng-axioms-def* **by** *simp-all*

lemma $\mathcal{Z}$*-srngE*[*elim*]:
assumes $\mathcal{Z}$*-srng* α $\mathfrak{S}$
obtains $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}$
and *vcard* $\mathfrak{S} = 3_{\mathbb{N}}$
and $\mathcal{Z}$*-csgrp* α $[\mathfrak{S}(\!|\mathcal{A}|\!), \mathfrak{S}(\!|vplus|\!)]_\circ$
and $\mathcal{Z}$*-sgrp* α $[\mathfrak{S}(\!|\mathcal{A}|\!), \mathfrak{S}(\!|vmult|\!)]_\circ$
and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ c \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow$
$(a +_{\circ\mathfrak{S}} b) *_{\circ\mathfrak{S}} c = (a *_{\circ\mathfrak{S}} c) +_{\circ\mathfrak{S}} (b *_{\circ\mathfrak{S}} c)$
and $\bigwedge a\ b\ c.\ [\![\ a \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ b \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!);\ c \in_\circ \mathfrak{S}(\!|\mathcal{A}|\!)\ ]\!] \Longrightarrow$
$a *_{\circ\mathfrak{S}} (b +_{\circ\mathfrak{S}} c) = (a *_{\circ\mathfrak{S}} b) +_{\circ\mathfrak{S}} (a *_{\circ\mathfrak{S}} c)$
using *assms* **unfolding** $\mathcal{Z}$*-srng-def* $\mathcal{Z}$*-srng-axioms-def* **by** *auto*

2.16.5 Integer numbers form a semiring

definition *vint-struct* :: *V* ($\langle\mathfrak{S}_{\mathbb{Z}}\rangle$)
where *vint-struct* = $[\mathbb{Z}_\circ, vint\text{-}plus, vint\text{-}mult]_\circ$

named-theorems *vint-struct-simps*

lemma *vint-struct-*$\mathcal{A}$[*vint-struct-simps*]: $\mathfrak{S}_{\mathbb{Z}}(\!|\mathcal{A}|\!) = \mathbb{Z}_\circ$
unfolding *vint-struct-def* **by** (*auto simp*: *sgrp-struct-field-simps*)

lemma *vint-struct-vplus*[*vint-struct-simps*]: $\mathfrak{S}_{\mathbb{Z}}(\!|vplus|\!) = vint\text{-}plus$
unfolding *vint-struct-def*
by (*simp add*: *sgrp-struct-field-simps nat-omega-simps*)

lemma *vint-struct-vmult*[*vint-struct-simps*]: $\mathfrak{S}_{\mathbb{Z}}(\!|vmult|\!) = vint\text{-}mult$
unfolding *vint-struct-def*
by (*simp add*: *sgrp-struct-field-simps nat-omega-simps*)

context $\mathcal{Z}$
begin

lemma $\mathcal{Z}$*-srng-vint*: $\mathcal{Z}$*-srng* α $\mathfrak{S}_{\mathbb{Z}}$
proof(*intro* $\mathcal{Z}$*-srngI*, *unfold vint-struct-simps*)

interpret $\mathfrak{S}$: *vfsequence* $\langle\mathfrak{S}_{\mathbb{Z}}\rangle$ **unfolding** *vint-struct-def* **by** *simp*

show *vint-struct*: $\mathcal{Z}$*-vfsequence* α $\mathfrak{S}_{\mathbb{Z}}$
proof(*intro* $\mathcal{Z}$*-vfsequenceI*)
show *vfsequence* $\mathfrak{S}_{\mathbb{Z}}$ **unfolding** *vint-struct-def* **by** *simp*
show $\mathcal{R}_\circ\ \mathfrak{S}_{\mathbb{Z}} \subseteq_\circ Vset\ \alpha$
proof(*intro vsubsetI*)
fix *x* **assume** $x \in_\circ \mathcal{R}_\circ\ \mathfrak{S}_{\mathbb{Z}}$
then consider $\langle x = \mathbb{Z}_\circ\rangle \mid \langle x = vint\text{-}plus\rangle \mid \langle x = vint\text{-}mult\rangle$
unfolding *vint-struct-def* **by** *fastforce*
then show $x \in_\circ Vset\ \alpha$
proof *cases*
case 1 **with** $\mathcal{Z}$*-Vset-*ω2*-vsubset-Vset vint-in-Vset-*ω2 **show** *?thesis* **by** *auto*
next
case 2
have $\mathcal{D}_\circ\ vint\text{-}plus \in_\circ Vset\ \alpha$
unfolding *vint-plus.nop-vdomain*

```
    proof(rule Limit-vcpower-in-VsetI)
      from Axiom-of-Infinity show 2ℕ ∈∘ Vset α by auto
      from ℤ-Vset-ω2-vsubset-Vset show ℤ∘ ∈∘ Vset α
        by (auto intro: vint-in-Vset-ω2)
    qed auto
    moreover from ℤ-Vset-ω2-vsubset-Vset have ℛ∘ vint-plus ∈∘ Vset α
      unfolding vint-plus.nop-onto-vrange by (auto intro: vint-in-Vset-ω2)
    ultimately show x ∈∘ Vset α
      unfolding 2
      by (simp add: rel-VLambda.vbrelation-Limit-in-VsetI vint-plus-def)
  next
    case 3
    have 𝒟∘ vint-mult ∈∘ Vset α
      unfolding vint-mult.nop-vdomain
    proof(rule Limit-vcpower-in-VsetI)
      from Axiom-of-Infinity show 2ℕ ∈∘ Vset α by auto
      from ℤ-Vset-ω2-vsubset-Vset show ℤ∘ ∈∘ Vset α
        by (auto intro: vint-in-Vset-ω2)
    qed auto
    moreover from ℤ-Vset-ω2-vsubset-Vset Axiom-of-Infinity have
      ℛ∘ vint-mult ∈∘ Vset α
      unfolding vint-mult.nop-onto-vrange by (auto intro: vint-in-Vset-ω2)
    ultimately show x ∈∘ Vset α
      unfolding 3
      by (simp add: rel-VLambda.vbrelation-Limit-in-VsetI vint-mult-def)
  qed
 qed
qed (simp add: ℤ-axioms)

interpret vint-struct: ℤ-vfsequence α ‹𝔖ℤ› by (rule vint-struct)

show vcard 𝔖ℤ = 3ℕ
  unfolding vint-struct-def by (simp add: nat-omega-simps)

have [vint-struct-simps]:
  [ℤ∘, vint-plus]∘(|𝒜|) = ℤ∘ [ℤ∘, vint-plus]∘(|mbinop|) = vint-plus
  [ℤ∘, vint-mult]∘(|𝒜|) = ℤ∘ [ℤ∘, vint-mult]∘(|mbinop|) = vint-mult
  by (auto simp: sgrp-struct-field-simps nat-omega-simps)

have [vint-struct-simps]:
  sgrp-app [ℤ∘, vint-plus]∘ = (+ℤ)
  sgrp-app [ℤ∘, vint-mult]∘ = (*ℤ)
  unfolding vint-struct-simps by simp-all

show ℤ-csgrp α [ℤ∘, vint-plus]∘
proof(intro ℤ-csgrpI, unfold vint-struct-simps)
  show ℤ-sgrp α [ℤ∘, vint-plus]∘
  proof(intro ℤ-sgrpI ℤ-sgrp-basisI, unfold vint-struct-simps)
    show ℤ-vfsequence α [ℤ∘, vint-plus]∘
    proof(intro ℤ-vfsequenceI)
      show ℛ∘ [ℤ∘, vint-plus]∘ ⊆∘ Vset α
      proof(intro vfsequence-vrange-vconsI)
        from ℤ-Vset-ω2-vsubset-Vset show [simp]: ℤ∘ ∈∘ Vset α
          by (auto intro: vint-in-Vset-ω2)
        show vint-plus ∈∘ Vset α
        proof(rule vbrelation.vbrelation-Limit-in-VsetI)
          from Axiom-of-Infinity show 𝒟∘ vint-plus ∈∘ Vset α
            unfolding vint-plus.nop-vdomain
```

```
          by (intro Limit-vcpower-in-VsetI) auto
        from Axiom-of-Infinity show ℛ∘ vint-plus ∈∘ Vset α
          unfolding vint-plus.nop-onto-vrange by auto
      qed (simp-all add: vint-plus-def)
    qed simp-all
  qed (simp-all add: ℤ-axioms)
 qed
  (
    auto simp:
     nat-omega-simps
     vint-plus.binop-axioms
     vint-assoc-law-addition
  )
qed (simp add: vint-commutative-law-addition)

show ℤ-sgrp α [ℤ∘, vint-mult]∘
proof
  (
    intro ℤ-sgrpI ℤ-sgrp-basisI;
    (unfold vint-struct-simps | tactic‹all-tac›)
  )
  show ℤ-vfsequence α [ℤ∘, vint-mult]∘
  proof(intro ℤ-vfsequenceI; (unfold vint-struct-simps | tactic‹all-tac›))
    from ℤ-axioms show ℤ α by simp
    show ℛ∘ [ℤ∘, vint-mult]∘ ⊆∘ Vset α
    proof(intro vfsequence-vrange-vconsI)
      from ℤ-Vset-ω2-vsubset-Vset show [simp]: ℤ∘ ∈∘ Vset α
        by (auto intro: vint-in-Vset-ω2)
      show vint-mult ∈∘ Vset α
      proof(rule vbrelation.vbrelation-Limit-in-VsetI)
        from Axiom-of-Infinity show 𝒟∘ vint-mult ∈∘ Vset α
          unfolding vint-mult.nop-vdomain
          by (intro Limit-vcpower-in-VsetI) auto
        from Axiom-of-Infinity show ℛ∘ vint-mult ∈∘ Vset α
          unfolding vint-mult.nop-onto-vrange by auto
      qed (simp-all add: vint-mult-def)
    qed simp-all
  qed auto
qed
  (
    auto simp:
      nat-omega-simps
      vint-mult.binop-axioms
      vint-assoc-law-multiplication
  )

qed
  (
    auto simp:
      vint-commutative-law-multiplication
      vint-plus-closed
      vint-distributive-law
  )
```

Interpretation.

```
interpretation vint: ℤ-srng α ‹𝔖ℤ›
  rewrites 𝔖ℤ(|𝒜|) = ℤ∘
    and 𝔖ℤ(|vplus|) = vint-plus
```

and $\mathfrak{S}_{\mathbb{Z}}(\!|vmult|\!)$ = *vint-mult*
and *vplus-app* $(\mathfrak{S}_{\mathbb{Z}})$ = *vint-plus-app*
and *vmult-app* $(\mathfrak{S}_{\mathbb{Z}})$ = *vint-mult-app*
unfolding *vint-struct-simps* **by** (*rule* $\mathcal{Z}$*-srng-vint*) *simp-all*

thm *vint.vmult.*$\mathcal{Z}$*-sgrp-assoc*
thm *vint.vplus.*$\mathcal{Z}$*-sgrp-assoc*
thm *vint.*$\mathcal{Z}$*-srng-distrib-left*

end

Digraphs

3.1 Introduction

3.1.1 Background

Many concepts that are normally associated with category theory can be generalized to directed graphs. It is the goal of this chapter to expose these generalized concepts and provide the relevant foundations for the development of the notion of a semicategory in the next chapter. It is important to note, however, that it is not the goal of this chapter to present a comprehensive canonical theory of directed graphs. Nonetheless, there is little that could prevent one from extending this body of work by providing canonical results from the theory of directed graphs.

3.1.2 Preliminaries

declare *One-nat-def*[*simp del*]

named-theorems *slicing-simps*
named-theorems *slicing-commute*
named-theorems *slicing-intros*

named-theorems *dg-op-simps*
named-theorems *dg-op-intros*

named-theorems *dg-cs-simps*
named-theorems *dg-cs-intros*

named-theorems *dg-shared-cs-simps*
named-theorems *dg-shared-cs-intros*

3.1.3 CS setup for foundations

named-theorems *V-cs-simps*
named-theorems *V-cs-intros*

named-theorems *Ord-cs-simps*
named-theorems *Ord-cs-intros*

Basic *HOL*

lemma (**in** *semilattice-sup*) *sup-commute'*:
  shows $b' = b \Longrightarrow a' = a \Longrightarrow a \sqcup b = b' \sqcup a'$
    and $b' = b \Longrightarrow a' = a \Longrightarrow a \sqcup b' = b \sqcup a'$
    and $b' = b \Longrightarrow a' = a \Longrightarrow a' \sqcup b = b' \sqcup a$
    and $b' = b \Longrightarrow a' = a \Longrightarrow a \sqcup b' = b \sqcup a'$
    and $b' = b \Longrightarrow a' = a \Longrightarrow a' \sqcup b' = b \sqcup a$
  by (*auto simp*: *sup.commute*)

lemma (**in** *semilattice-inf*) *inf-commute'*:

shows $b' = b \Longrightarrow a' = a \Longrightarrow a \sqcap b = b' \sqcap a'$
and $b' = b \Longrightarrow a' = a \Longrightarrow a \sqcap b' = b \sqcap a'$
and $b' = b \Longrightarrow a' = a \Longrightarrow a' \sqcap b = b' \sqcap a$
and $b' = b \Longrightarrow a' = a \Longrightarrow a \sqcap b' = b \sqcap a'$
and $b' = b \Longrightarrow a' = a \Longrightarrow a' \sqcap b' = b \sqcap a$
by (*auto simp*: *inf.commute*)

lemmas [*V-cs-simps*] =
if-P
if-not-P
*inf.absorb*1
*inf.absorb*2
*sup.absorb*1
*sup.absorb*2
add-0-*right*
add-0

lemmas [*V-cs-intros*] =
conjI
sup-commute'
inf-commute'
sup.commute
inf.commute

Lists for *HOL*

lemma *list-all-singleton*: *list-all P* [*x*] = *P x* **by** *simp*

lemma *replicate-one*: *replicate* 1 *x* = [*x*]
by (*simp add*: *One-nat-def*)

lemma *list-all-mono*:
assumes *list-all P xs* **and** $P \leq Q$
shows *list-all Q xs*
using *assms* **by** (*metis list.pred-mono-strong rev-predicate*1*D*)

lemma *pred-in-set-mono*:
assumes $S \subseteq T$
shows $(\lambda x.\ x \in S) \leq (\lambda x.\ x \in T)$
using *assms* **by** *auto*

lemma *elts-subset-mono*:
assumes $S \subseteq_\circ T$
shows *elts S* ⊆ *elts T*
using *assms* **by** *auto*

lemma *list-all-replicate*:
assumes *P x*
shows *list-all P* (*replicate n x*)
using *assms* **by** (*metis Ball-set in-set-replicate*)

lemma *list-all-set*:
assumes *list-all P xs* **and** *x* ∈ *list.set xs*
shows *P x*
using *assms* **by** (*induct xs*) *auto*

lemma *list-map-id*:
assumes *list-all* (λ*x*. *f x* = *x*) *xs*

shows *map f xs = xs*
using *assms* **by** (*induct xs*) *auto*

lemmas [*V-cs-simps*] =
List.append.append-Nil
List.append-Nil2
List.append.append-Cons
List.rev.simps(1)
list.map(1,2)
rev.simps(2)
List.map-append
list-all-append
replicate.replicate-0
rev-replicate
semiring-1-*class.of-nat*-0
group-add-class.minus-zero
group-add-class.minus-minus
replicate.replicate-Suc
replicate-one
list-all-singleton

lemmas [*V-cs-intros*] =
exI
pred-in-set-mono
elts-subset-mono
list-all-replicate

Foundations

abbreviation (*input*) *if3* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *if3 a b c* ≡
(
$\lambda i.$ *if* $i = 0 \Rightarrow a$
| $i = 1_{\mathbb{N}} \Rightarrow b$
| *otherwise* ⇒ *c*
)
lemma *if3*-0[*V-cs-simps*]: *if3 a b c* 0 = *a* **by** *auto*
lemma *if3*-1[*V-cs-simps*]: *if3 a b c* ($1_{\mathbb{N}}$) = *b* **by** *auto*
lemma *if3*-2[*V-cs-simps*]: *if3 a b c* ($2_{\mathbb{N}}$) = *c* **by** *auto*

lemma *vinsertI1′*:
assumes $x' = x$
shows $x \in_\circ$ *vinsert* x' *A*
unfolding *assms* **by** (*rule vinsertI1*)

lemma *in-vsingleton*[*V-cs-intros*]:
assumes *f* = *a*
shows $f \in_\circ$ *set* {*a*}
unfolding *assms* **by** *simp*

lemma *a-in-succ-a*: $a \in_\circ$ *succ a* **by** *simp*

lemma *a-in-succ-xI*:
assumes $a \in_\circ x$
shows $a \in_\circ$ *succ x*
using *assms* **by** *simp*

lemma *vone-ne*[*V-cs-intros*]: $1_{\mathbb{N}} \neq 0$ **by** *clarsimp*

lemmas [*V-cs-simps*] =
vinsert-set-insert-eq
beta
set-empty
vcard-0

lemmas [*V-cs-intros*] =
mem-not-refl
succ-notin-self
vset-neq-1
vset-neq-2
nin-vinsertI
*vinsertI*1′
*vinsertI*2
vfinite-vinsert
vfinite-vsingleton
vdisjnt-nin-right
vdisjnt-nin-left
*vunionI*1
*vunionI*2
vunion-in-VsetI
vintersection-in-VsetI
vsubset-reflexive
vsingletonI
small-insert small-empty
Limit-vtimes-in-VsetI
Limit-VPow-in-VsetI
a-in-succ-a
vsubset-vempty

Binary relations

lemma *vtimesI*′[*V-cs-intros*]:
assumes $ab = \langle a, b \rangle$ **and** $a \in_\circ A$ **and** $b \in_\circ B$
shows $ab \in_\circ A \times_\circ B$
using *assms* **by** *simp*

lemma *vrange-vcomp-vsubset*[*V-cs-intros*]:
assumes $\mathcal{R}_\circ\ r \subseteq_\circ B$
shows $\mathcal{R}_\circ\ (r \circ_\circ s) \subseteq_\circ B$
using *assms* **by** *auto*

lemma *vrange-vconst-on-vsubset*[*V-cs-intros*]:
assumes $a \in_\circ R$
shows $\mathcal{R}_\circ\ (\mathit{vconst\text{-}on}\ A\ a) \subseteq_\circ R$
using *assms* **by** *auto*

lemma *vrange-vcomp-eq-vrange*[*V-cs-simps*]:
assumes $\mathcal{D}_\circ\ r = \mathcal{R}_\circ\ s$
shows $\mathcal{R}_\circ\ (r \circ_\circ s) = \mathcal{R}_\circ\ r$
using *assms* **by** (*metis vimage-vdomain vrange-vcomp*)

lemmas [*V-cs-simps*] =
vdomain-vsingleton
vdomain-vlrestriction
vdomain-vlrestriction-vsubset
vdomain-vcomp-vsubset

vdomain-vconverse
vrange-vconverse
vdomain-vconst-on
vconverse-vtimes
vdomain-VLambda

lemmas [*V-cs-intros*] = *vcpower-vsubset-mono*

Single-valued functions

lemmas (**in** *vsv*) [*V-cs-intros*] = *vsv-axioms*

lemma *vpair-app*:
assumes $j = a$
shows *set* $\{\langle a, b\rangle\}(\!|j|\!) = b$
unfolding *assms* **by** *simp*

lemmas [*V-cs-simps*] =
vpair-app
vsv.vlrestriction-app
vsv-vcomp-at
vid-on-atI

lemmas (**in** *vsv*) [*V-cs-intros*] = *vsv-vimageI2′*

lemmas [*V-cs-intros*] =
vsv-vsingleton
vsv.vsv-vimageI2′
vsv-vcomp

Injective single-valued functions

lemmas (**in** *v11*) [*V-cs-intros*] = *v11-axioms*

lemma (**in** *v11*) *v11-vconverse-app-in-vdomain′*:
assumes $y \in_\circ \mathcal{R}_\circ\ r$ **and** $A = \mathcal{D}_\circ\ r$
shows $r^{-1}{}_\circ(\!|y|\!) \in_\circ A$
using *assms*(1) **unfolding** *assms*(2) **by** (*rule v11-vconverse-app-in-vdomain*)

lemmas (**in** *v11*) [*V-cs-intros*] = *v11-vconverse-app-in-vdomain′*
lemmas [*V-cs-intros*] = *v11.v11-vconverse-app-in-vdomain′*

lemmas (**in** *v11*) [*V-cs-simps*] =
v11-app-if-vconverse-app[*rotated* −2]
v11-app-vconverse-app
v11-vconverse-app-app

lemmas [*V-cs-simps*] =
v11.v11-vconverse-app[*rotated* −1]
v11.v11-app-vconverse-app
v11.v11-vconverse-app-app

lemmas [*V-cs-intros*] =
v11D(1)
v11.v11-vconverse
v11-vcomp

Operations on indexed families of sets

lemmas [*V-cs-simps*] =
vprojection-app
vprojection-vdomain

lemmas [*V-cs-intros*] = *vprojection-vsv*

Finite sequences

lemmas (**in** *vfsequence*) [*V-cs-intros*] = *vfsequence-axioms*

lemmas (**in** *vfsequence*) [*V-cs-simps*] = *vfsequence-vdomain*
lemmas [*V-cs-simps*] = *vfsequence.vfsequence-vdomain*

lemmas [*V-cs-intros*] =
vfsequence.vfsequence-vcons
vfsequence-vempty

lemmas [*V-cs-simps*] =
vfinite-0-*left*
vfinite-0-*right*

Binary relation as a finite sequence

lemmas [*V-cs-simps*] =
fconverse-vunion
fconverse-ftimes
vdomain-fflip

lemmas [*V-cs-intros*] =
*ftimesI*2
vcpower-two-ftimesI

Ordinals

lemmas [*Ord-cs-intros*] =
Limit-right-Limit-mult
Limit-left-Limit-mult
Ord-succ-mono
Limit-plus-omega-vsubset-Limit
Limit-plus-nat-in-Limit

von Neumann hierarchy

lemma (**in** $\mathcal{Z}$) *omega-in-any*[*V-cs-intros*]:
assumes $\alpha \subseteq_\circ \beta$
shows $\omega \in_\circ \beta$
using *assms* **by** *auto*

lemma *Ord-vsubset-succ*[*V-cs-intros*]:
assumes *Ord* α **and** *Ord* β **and** $\alpha \subseteq_\circ \beta$
shows $\alpha \subseteq_\circ$ *succ* β
by (*metis Ord-linear-le Ord-succ assms*(1) *assms*(2) *assms*(3) *leD succ-le-iff*)

lemma *Ord-in-Vset-succ*[*V-cs-intros*]:
assumes *Ord* α **and** $a \in_\circ$ *Vset* α
shows $a \in_\circ$ *Vset* (*succ* α)
using *assms* **by** (*auto simp*: *Ord-Vset-in-Vset-succI*)

lemma *Ord-vsubset-Vset-succ*[*V-cs-intros*]:
assumes *Ord* α **and** $B \subseteq_\circ$ *Vset* α
shows $B \subseteq_\circ$ *Vset* (*succ* α)
by (*intro vsubsetI*)
(*auto simp*: *assms Vset-trans Ord-vsubset-in-Vset-succI*)

lemmas (**in** $\mathcal{Z}$) [*V-cs-intros*] =
omega-in-α
Ord-α
Limit-α

lemmas [*V-cs-intros*] =
vempty-in-Vset-succ
$\mathcal{Z}$.*ord-of-nat-in-Vset*
Vset-in-mono
Limit-vpair-in-VsetI
Vset-vsubset-mono
Ord-succ
Limit-vempty-in-VsetI
Limit-insert-in-VsetI
vfsequence.vfsequence-Limit-vcons-in-VsetI
vfsequence.vfsequence-Ord-vcons-in-Vset-succI
Limit-vdoubleton-in-VsetI
Limit-omega-in-VsetI
Limit-ftimes-in-VsetI

n-ary operations

lemmas [*V-cs-simps*] =
fflip-app
vdomain-fflip

Countable ordinals as a set

named-theorems *omega-of-set*
named-theorems *nat-omega-simps-extra*

lemmas [*nat-omega-simps-extra*] =
add-num-simps
Suc-numeral
Suc-1
le-num-simps
less-numeral-simps(1,2)
less-num-simps
less-one
nat-omega-simps

lemmas [*omega-of-set*] = *nat-omega-simps-extra*

lemma *set-insert-succ*[*omega-of-set*]:
assumes [*simp*]: *small b* **and** *set b* = $a_{\mathbb{N}}$
shows *set* (*insert* ($a_{\mathbb{N}}$) *b*) = *succ* ($a_{\mathbb{N}}$)
unfolding *assms*(2)[*symmetric*] **by** *auto*

lemma *set-0*[*omega-of-set*]: *set* {0} = *succ* 0 **by** *auto*

Sequences

named-theorems *vfsequence-simps*
named-theorems *vfsequence-intros*

lemmas [*vfsequence-simps*] =
vfsequence.vfsequence-at-last[*rotated*]
vfsequence.vfsequence-vcard-vcons[*rotated*]
vfsequence.vfsequence-at-not-last[*rotated*]

lemmas [*vfsequence-intros*] =
vfsequence.vfsequence-vcons
vfsequence-vempty

Further numerals

named-theorems *nat-omega-intros*

lemma [*nat-omega-intros*]:
assumes $a < b$
shows $a_{\mathbb{N}} \in_{\circ} b_{\mathbb{N}}$
using *assms* **by** *simp*

lemma [*nat-omega-intros*]:
assumes $0 < b$
shows $0 \in_{\circ} b_{\mathbb{N}}$
using *assms* **by** *auto*

lemma [*nat-omega-intros*]:
assumes $a = numeral\ b$
shows $(0{::}nat) < a$
using *assms* **by** *auto*

lemma *nat-le-if-in*[*nat-omega-intros*]:
assumes $x_{\mathbb{N}} \in_{\circ} y_{\mathbb{N}}$
shows $x_{\mathbb{N}} \leq y_{\mathbb{N}}$
using *assms* **by** *auto*

lemma *vempty-le-nat*[*nat-omega-intros*]: $0 \leq y_{\mathbb{N}}$ **by** *auto*

lemmas [*nat-omega-intros*] =
preorder-class.order-refl
preorder-class.eq-refl

Generally available foundational results

lemma (in $\mathcal{Z}$**)** $\mathcal{Z}$*-*β:
assumes $\beta = \alpha$
shows $\mathcal{Z}\ \beta$
unfolding *assms* **by** *auto*

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

3.2 Digraph

3.2.1 Background

named-theorems *dg-field-simps*

definition *Obj* :: *V* **where** [*dg-field-simps*]: *Obj* = 0
definition *Arr* :: *V* **where** [*dg-field-simps*]: *Arr* = $1_{\mathbb{N}}$
definition *Dom* :: *V* **where** [*dg-field-simps*]: *Dom* = $2_{\mathbb{N}}$
definition *Cod* :: *V* **where** [*dg-field-simps*]: *Cod* = $3_{\mathbb{N}}$

3.2.2 Arrow with a domain and a codomain

The definition of and notation for an arrow with a domain and codomain is adapted from Chapter I-1 in [39]. The definition is applicable to digraphs and all other relevant derived entities, such as semicategories and categories, that are presented in the subsequent chapters.

In this work, by convention, the definition of an arrow with a domain and a codomain is nearly always preferred to the explicit use of the domain and codomain functions for the specification of the fundamental properties of arrows. Thus, to say that f is an arrow with the domain a, it is preferable to write $f : a \mapsto_{\mathfrak{C}} b$ (b can be assumed to be arbitrary) instead of $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)$ and $\mathfrak{C}(\!|Dom|\!)(\!|f|\!) = a$.

definition *is-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  where *is-arr* $\mathfrak{C}$ *a b f* $\longleftrightarrow$ $f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge \mathfrak{C}(\!|Dom|\!)(\!|f|\!) = a \wedge \mathfrak{C}(\!|Cod|\!)(\!|f|\!) = b$

syntax *-is-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool* ($\langle$- : - $\mapsto_1$ -$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-arr* $\rightleftharpoons$ *is-arr*
translations $f : a \mapsto_{\mathfrak{C}} b \rightleftharpoons$ *CONST is-arr* $\mathfrak{C}$ *a b f*

Rules.

mk-ide *is-arr-def*
  |*intro is-arrI*|
  |*dest is-arrD*[*dest*]|
  |*elim is-arrE*[*elim*]|

lemmas [*dg-shared-cs-intros*, *dg-cs-intros*] = *is-arrD*(1)
lemmas [*dg-shared-cs-simps*, *dg-cs-simps*] = *is-arrD*(2,3)

3.2.3 *Hom*-set

See Chapter I-8 in [39].

abbreviation *Hom* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *Hom* $\mathfrak{C}$ *a b* $\equiv$ *set* $\{f.\ f : a \mapsto_{\mathfrak{C}} b\}$

lemma *small-Hom*[*simp*]: *small* $\{f.\ f : a \mapsto_{\mathfrak{C}} b\}$ **unfolding** *is-arr-def* **by** *simp*

Rules.

lemma *HomI*[*dg-shared-cs-intros*, *dg-cs-intros*]:
  assumes $f : a \mapsto_{\mathfrak{C}} b$
  shows $f \in_{\circ}$ *Hom* $\mathfrak{C}$ *a b*
  using *assms* **by** *auto*

lemma *in-Hom-iff*[*dg-shared-cs-simps*, *dg-cs-simps*]:
  $f \in_{\circ}$ *Hom* $\mathfrak{C}$ *a b* $\longleftrightarrow$ $f : a \mapsto_{\mathfrak{C}} b$
  by *simp*

The *Hom*-sets in a given digraph are pairwise disjoint. This property was exposed as Axiom (v) in an alternative definition of a category presented in Chapter I-8 in [39]. Within the scope of the definitional framework employed in this study, this property holds unconditionally.

lemma *Hom-vdisjnt*:
assumes $a \neq a' \vee b \neq b'$
and $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $a' \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
and $b' \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows *vdisjnt* (*Hom* $\mathfrak{C}$ a b) (*Hom* $\mathfrak{C}$ a' b')
proof(*intro vdisjntI*, *unfold in-Hom-iff*)
fix g f **assume** $g : a \mapsto_{\mathfrak{C}} b$ **and** $f : a' \mapsto_{\mathfrak{C}} b'$
then have $g \in_\circ \mathfrak{C}(\!|Arr|\!)$
and $f \in_\circ \mathfrak{C}(\!|Arr|\!)$
and $\mathfrak{C}(\!|Dom|\!)(\!|g|\!) = a$
and $\mathfrak{C}(\!|Cod|\!)(\!|g|\!) = b$
and $\mathfrak{C}(\!|Dom|\!)(\!|f|\!) = a'$
and $\mathfrak{C}(\!|Cod|\!)(\!|f|\!) = b'$
by (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)+
with *assms*(1) **have** $\mathfrak{C}(\!|Dom|\!)(\!|g|\!) \neq \mathfrak{C}(\!|Dom|\!)(\!|f|\!) \vee \mathfrak{C}(\!|Cod|\!)(\!|g|\!) \neq \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$ **by** *auto*
then show $g \neq f$ **by** *clarsimp*
qed

3.2.4 Digraph: background information

The definition of a digraph that is employed in this work is similar to the definition of a *directed graph* presented in Chapter I-2 in [39]. However, there are notable differences. More specifically, the definition is parameterized by a limit ordinal α, such that $\omega < \alpha$; the set of objects is assumed to be a subset of the set V_α in the von Neumann hierarchy of sets (e.g., see [59]). Such digraphs are called *α-digraphs* to make the dependence on the parameter α explicit.[1] This definition was inspired by the ideas expressed in [28], [52] and [57].

In ZFC in HOL, the predicate *small* is used for distinguishing the terms of any type of the form *'a set* that are isomorphic to elements of a term of the type V (the elements can be exposed via the predicate *elts*). Thus, the collection of the elements associated with any term of the type V (e.g., *elts a*) is always small (see the theorem *small-elts* in [51]). Therefore, in this study, in an attempt to avoid confusion, the term "small" is never used to refer to digraphs. Instead, a new terminology is introduced in this body of work.

Thus, in this work, an α-digraph is a tiny α-digraph if and only if the set of its objects and the set of its arrows both belong to the set V_α. This notion is similar to the notion of a small category in the sense of the definition employed in Chapter I-6 in [39], if it is assumed that the "smallness" is determined with respect to the set V_α instead of the universe U. Also, in what follows, any member of the set V_α will be referred to as an α-tiny set.

All of the large (i.e. non-tiny) digraphs that are considered within the scope of this work have a slightly unconventional condition associated with the size of their *Hom*-sets. This condition implies that all *Hom*-sets of a digraph are tiny, but it is not equivalent to all *Hom*-sets being tiny. The condition was introduced in an attempt to resolve some of the issues related to the lack of an analogue of the Axiom Schema of Replacement closed with respect to V_α.

3.2.5 Digraph: definition and elementary properties

locale *digraph* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ + *Dom*: *vsv* ‹$\mathfrak{C}(\!|Dom|\!)$› + *Cod*: *vsv* ‹$\mathfrak{C}(\!|Cod|\!)$›

[1] The prefix "α-" may be omitted whenever it is possible to infer the value of α from the context. This applies not only to the digraphs, but all other entities that are parameterized by a limit ordinal α such that $\omega < \alpha$.

for α $\mathfrak{C}$ +
assumes *dg-length*[*dg-cs-simps*]: *vcard* $\mathfrak{C} = 4_{\mathbb{N}}$
and *dg-Dom-vdomain*[*dg-cs-simps*]: $\mathcal{D}_{\circ}$ ($\mathfrak{C}(\!|Dom|\!)$) = $\mathfrak{C}(\!|Arr|\!)$
and *dg-Dom-vrange*: $\mathcal{R}_{\circ}$ ($\mathfrak{C}(\!|Dom|\!)$) $\subseteq_{\circ}$ $\mathfrak{C}(\!|Obj|\!)$
and *dg-Cod-vdomain*[*dg-cs-simps*]: $\mathcal{D}_{\circ}$ ($\mathfrak{C}(\!|Cod|\!)$) = $\mathfrak{C}(\!|Arr|\!)$
and *dg-Cod-vrange*: $\mathcal{R}_{\circ}$ ($\mathfrak{C}(\!|Cod|\!)$) $\subseteq_{\circ}$ $\mathfrak{C}(\!|Obj|\!)$
and *dg-Obj-vsubset-Vset*: $\mathfrak{C}(\!|Obj|\!)$ $\subseteq_{\circ}$ *Vset* α
and *dg-Hom-vifunion-in-Vset*[*dg-cs-intros*]:
$[\![A \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!);\ B \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!);\ A \in_{\circ} Vset\ \alpha;\ B \in_{\circ} Vset\ \alpha\ ]\!] \Longrightarrow$
$(\bigcup_{\circ} a \in_{\circ} A.\ \bigcup_{\circ} b \in_{\circ} B.\ Hom\ \mathfrak{C}\ a\ b) \in_{\circ} Vset\ \alpha$

lemmas [*dg-cs-simps*] =
digraph.dg-length
digraph.dg-Dom-vdomain
digraph.dg-Cod-vdomain

lemmas [*dg-cs-intros*] =
digraph.dg-Hom-vifunion-in-Vset

Rules.

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

mk-ide rf *digraph-def*[*unfolded digraph-axioms-def*]
|*intro digraphI*|
|*dest digraphD*[*dest*]|
|*elim digraphE*[*elim*]|

Elementary properties.

lemma *dg-eqI*:
assumes *digraph* α $\mathfrak{A}$
and *digraph* α $\mathfrak{B}$
and $\mathfrak{A}(\!|Obj|\!) = \mathfrak{B}(\!|Obj|\!)$
and $\mathfrak{A}(\!|Arr|\!) = \mathfrak{B}(\!|Arr|\!)$
and $\mathfrak{A}(\!|Dom|\!) = \mathfrak{B}(\!|Dom|\!)$
and $\mathfrak{A}(\!|Cod|\!) = \mathfrak{B}(\!|Cod|\!)$
shows $\mathfrak{A} = \mathfrak{B}$
proof–
interpret $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **by** (*rule assms*(1))
interpret $\mathfrak{B}$: *digraph* α $\mathfrak{B}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_{\circ}$ $\mathfrak{A}$ = $4_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *V-cs-simps dg-cs-simps*)
show $a \in_{\circ} \mathcal{D}_{\circ}\ \mathfrak{A} \Longrightarrow \mathfrak{A}(\!|a|\!) = \mathfrak{B}(\!|a|\!)$ **for** a
by (*unfold dom-lhs*, *elim-in-numeral*, *insert assms*)
(*auto simp*: *dg-field-simps*)
qed
(
cs-concl **cs-shallow**
cs-simp: *V-cs-simps dg-cs-simps* **cs-intro**: *V-cs-intros*
)+
qed

lemma (**in** *digraph*) *dg-def*: $\mathfrak{C} = [\mathfrak{C}(\!|Obj|\!), \mathfrak{C}(\!|Arr|\!), \mathfrak{C}(\!|Dom|\!), \mathfrak{C}(\!|Cod|\!)]_{\circ}$
proof(*rule vsv-eqI*)

```
  have dom-lhs: 𝒟∘ 𝔊 = 4_ℕ
    by (cs-concl cs-shallow cs-simp: V-cs-simps dg-cs-simps)
  have dom-rhs: 𝒟∘ [𝔊(|Obj|), 𝔊(|Arr|), 𝔊(|Dom|), 𝔊(|Cod|)]∘ = 4_ℕ
    by (simp add: nat-omega-simps)
  then show 𝒟∘ 𝔊 = 𝒟∘ [𝔊(|Obj|), 𝔊(|Arr|), 𝔊(|Dom|), 𝔊(|Cod|)]∘
    unfolding dom-lhs dom-rhs by simp
  show a ∈∘ 𝒟∘ 𝔊 ⟹ 𝔊(|a|) = [𝔊(|Obj|), 𝔊(|Arr|), 𝔊(|Dom|), 𝔊(|Cod|)]∘(|a|) for a
    by (unfold dom-lhs, elim-in-numeral, unfold dg-field-simps)
      (simp-all add: nat-omega-simps)
qed (auto simp: vsv-axioms)

lemma (in digraph) dg-Obj-if-Dom-vrange:
  assumes a ∈∘ ℛ∘ (𝔊(|Dom|))
  shows a ∈∘ 𝔊(|Obj|)
  using assms dg-Dom-vrange by auto

lemma (in digraph) dg-Obj-if-Cod-vrange:
  assumes a ∈∘ ℛ∘ (𝔊(|Cod|))
  shows a ∈∘ 𝔊(|Obj|)
  using assms dg-Cod-vrange by auto

lemma (in digraph) dg-is-arrD:
  assumes f : a ↦_𝔊 b
  shows f ∈∘ 𝔊(|Arr|)
    and a ∈∘ 𝔊(|Obj|)
    and b ∈∘ 𝔊(|Obj|)
    and 𝔊(|Dom|)(|f|) = a
    and 𝔊(|Cod|)(|f|) = b
proof–
  from assms show prems: f ∈∘ 𝔊(|Arr|)
    and fa[symmetric]: 𝔊(|Dom|)(|f|) = a
    and fb[symmetric]: 𝔊(|Cod|)(|f|) = b
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)+
  from digraph-axioms prems have f ∈∘ 𝒟∘ (𝔊(|Dom|)) f ∈∘ 𝒟∘ (𝔊(|Cod|))
    by (cs-concl cs-shallow cs-simp: dg-cs-simps)+
  with assms show a ∈∘ 𝔊(|Obj|) b ∈∘ 𝔊(|Obj|)
    by
      (
        cs-concl
          cs-intro: dg-Obj-if-Dom-vrange dg-Obj-if-Cod-vrange V-cs-intros
          cs-simp: fa fb
      )+
qed

lemmas [dg-cs-intros] = digraph.dg-is-arrD(1–3)

lemma (in digraph) dg-is-arrE[elim]:
  assumes f : a ↦_𝔊 b
  obtains f ∈∘ 𝔊(|Arr|)
    and a ∈∘ 𝔊(|Obj|)
    and b ∈∘ 𝔊(|Obj|)
    and 𝔊(|Dom|)(|f|) = a
    and 𝔊(|Cod|)(|f|) = b
  using assms by (blast dest: dg-is-arrD)

lemma (in digraph) dg-in-ArrE[elim]:
  assumes f ∈∘ 𝔊(|Arr|)
  obtains a b where f : a ↦_𝔊 b and a ∈∘ 𝔊(|Obj|) and b ∈∘ 𝔊(|Obj|)
```

using *assms* **by** (*auto dest*: *dg-is-arrD*(2,3) *is-arrI*)

lemma (**in** *digraph*) *dg-Hom-in-Vset*[*dg-cs-intros*]:
assumes $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows $Hom\ \mathfrak{C}\ a\ b \in_\circ Vset\ \alpha$
proof–
let $?A = \langle set\ \{a\}\rangle$ **and** $?B = \langle set\ \{b\}\rangle$
from *assms* **have** *A*: $?A \subseteq_\circ \mathfrak{C}(\!|Obj|\!)$ **and** *B*: $?B \subseteq_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *auto*
from *assms dg-Obj-vsubset-Vset* **have** $a \in_\circ Vset\ \alpha$ **and** $b \in_\circ Vset\ \alpha$ **by** *auto*
then have *a*: $set\ \{a\} \in_\circ Vset\ \alpha$ **and** *b*: $set\ \{b\} \in_\circ Vset\ \alpha$
by (*metis Axiom-of-Pairing insert-absorb*2)+
from *dg-Hom-vifunion-in-Vset*[*OF A B a b*] **show** $Hom\ \mathfrak{C}\ a\ b \in_\circ Vset\ \alpha$ **by** *simp*
qed

lemmas [*dg-cs-intros*] = *digraph.dg-Hom-in-Vset*

Size.

lemma (**in** *digraph*) *dg-Arr-vsubset-Vset*: $\mathfrak{C}(\!|Arr|\!) \subseteq_\circ Vset\ \alpha$
proof(*intro vsubsetI*)
fix *f* **assume** $f \in_\circ \mathfrak{C}(\!|Arr|\!)$
then obtain *a b*
where *f*: $f : a \mapsto_{\mathfrak{C}} b$ **and** *a*: $a \in_\circ \mathfrak{C}(\!|Obj|\!)$ **and** *b*: $b \in_\circ \mathfrak{C}(\!|Obj|\!)$
by *blast*
show $f \in_\circ Vset\ \alpha$
by (*rule Vset-trans*, *rule HomI*[*OF f*], *rule dg-Hom-in-Vset*[*OF a b*])
qed

lemma (**in** *digraph*) *dg-Dom-vsubset-Vset*: $\mathfrak{C}(\!|Dom|\!) \subseteq_\circ Vset\ \alpha$
by
(
rule Dom.vbrelation-Limit-vsubset-VsetI,
unfold dg-cs-simps,
insert dg-Dom-vrange dg-Obj-vsubset-Vset
)
(*auto intro*!: *dg-Arr-vsubset-Vset*)

lemma (**in** *digraph*) *dg-Cod-vsubset-Vset*: $\mathfrak{C}(\!|Cod|\!) \subseteq_\circ Vset\ \alpha$
by
(
rule Cod.vbrelation-Limit-vsubset-VsetI,
unfold dg-cs-simps,
insert dg-Cod-vrange dg-Obj-vsubset-Vset
)
(*auto intro*!: *dg-Arr-vsubset-Vset*)

lemma (**in** *digraph*) *dg-digraph-in-Vset-4*: $\mathfrak{C} \in_\circ Vset\ (\alpha + 4_{\mathbb{N}})$
proof–
note [*folded VPow-iff*, *folded Vset-succ*[*OF Ord-α*], *dg-cs-intros*] =
dg-Obj-vsubset-Vset
dg-Arr-vsubset-Vset
dg-Dom-vsubset-Vset
dg-Cod-vsubset-Vset
show *?thesis*
by (*subst dg-def*, *succ-of-numeral*)
(
cs-concl
cs-simp: *plus-V-succ-right V-cs-simps*
cs-intro: *dg-cs-intros V-cs-intros*

```
    )
qed

lemma (in digraph) dg-Obj-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔇(|Obj|) ∈∘ Vset β
  using assms dg-Obj-vsubset-Vset Vset-in-mono by auto

lemma (in digraph) dg-in-Obj-in-Vset[dg-cs-intros]:
  assumes a ∈∘ 𝔇(|Obj|)
  shows a ∈∘ Vset α
  using assms dg-Obj-vsubset-Vset by auto

lemma (in digraph) dg-Arr-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔇(|Arr|) ∈∘ Vset β
  using assms dg-Arr-vsubset-Vset Vset-in-mono by auto

lemma (in digraph) dg-in-Arr-in-Vset[dg-cs-intros]:
  assumes a ∈∘ 𝔇(|Arr|)
  shows a ∈∘ Vset α
  using assms dg-Arr-vsubset-Vset by auto

lemma (in digraph) dg-Dom-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔇(|Dom|) ∈∘ Vset β
  by (meson assms dg-Dom-vsubset-Vset Vset-in-mono vsubset-in-VsetI)

lemma (in digraph) dg-Cod-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔇(|Cod|) ∈∘ Vset β
  by (meson assms dg-Cod-vsubset-Vset Vset-in-mono vsubset-in-VsetI)

lemma (in digraph) dg-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔇 ∈∘ Vset β
proof–
  interpret β: 𝒵 β by (rule assms(1))
  note [dg-cs-intros] =
    dg-Obj-in-Vset dg-Arr-in-Vset dg-Dom-in-Vset dg-Cod-in-Vset
  from assms(2) show ?thesis
    by (subst dg-def)
      (cs-concl cs-shallow cs-intro: dg-cs-intros V-cs-intros)
qed

lemma (in digraph) dg-digraph-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows digraph β 𝔇
proof(rule digraphI)
  show vfsequence 𝔇 by (simp add: vfsequence-axioms)
  show 𝔇(|Obj|) ⊆∘ Vset β
    by (rule vsubsetI)
      (meson Vset-in-mono Vset-trans assms(2) dg-Obj-vsubset-Vset vsubsetE)
  fix A B assume A ⊆∘ 𝔇(|Obj|) B ⊆∘ 𝔇(|Obj|) A ∈∘ Vset β B ∈∘ Vset β
  then have (⋃∘a∈∘A. ⋃∘b∈∘B. Hom 𝔇 a b) ⊆∘ 𝔇(|Arr|) by auto
  moreover note dg-Arr-vsubset-Vset
  moreover have Vset α ∈∘ Vset β by (simp add: Vset-in-mono assms(2))
  ultimately show (⋃∘a∈∘A. ⋃∘b∈∘B. Hom 𝔇 a b) ∈∘ Vset β by auto
```

qed (*auto simp*: *assms*(1) *dg-Dom-vrange dg-Cod-vrange dg-cs-simps*)

lemma *small-digraph*[*simp*]: *small* {𝔊. *digraph* α 𝔊}
proof(*cases* ‹$\mathcal{Z}$ α›)
case *True*
with *digraph.dg-in-Vset* **show** *?thesis*
by (*intro down*[*of* - ‹*Vset* (α + ω)›] *subsetI*)
(*auto simp*: $\mathcal{Z}$.*Limit-αω* $\mathcal{Z}$.*ω-αω* $\mathcal{Z}$.*intro* $\mathcal{Z}$.*α-αω*)
next
case *False*
then have {𝔊. *digraph* α 𝔊} = {} **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma (**in** $\mathcal{Z}$) *digraphs-in-Vset*:
assumes $\mathcal{Z}$ β **and** α $\in_\circ$ β
shows *set* {𝔊. *digraph* α 𝔊} $\in_\circ$ *Vset* β
proof(*rule vsubset-in-VsetI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(1))
show *set* {𝔊. *digraph* α 𝔊} $\subseteq_\circ$ *Vset* (α + $4_{\mathbb{N}}$)
proof(*intro vsubsetI*)
fix 𝔊 **assume** 𝔊 $\in_\circ$ *set* {𝔊. *digraph* α 𝔊}
then interpret *digraph* α 𝔊 **by** *simp*
show 𝔊 $\in_\circ$ *Vset* (α + $4_{\mathbb{N}}$)
unfolding *VPow-iff* **by** (*rule dg-digraph-in-Vset*-4)
qed
from *assms*(2) **show** *Vset* (α + $4_{\mathbb{N}}$) $\in_\circ$ *Vset* β
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros Ord-cs-intros*)
qed

lemma *digraph-if-digraph*:
assumes *digraph* β 𝔊
and $\mathcal{Z}$ α
and 𝔊(|*Obj*|) $\subseteq_\circ$ *Vset* α
and $\bigwedge A\ B$. [[A $\subseteq_\circ$ 𝔊(|*Obj*|); B $\subseteq_\circ$ 𝔊(|*Obj*|); A $\in_\circ$ *Vset* α; B $\in_\circ$ *Vset* α]] $\Longrightarrow$
($\bigcup_\circ a \in_\circ A$. $\bigcup_\circ b \in_\circ B$. *Hom* 𝔊 a b) $\in_\circ$ *Vset* α
shows *digraph* α 𝔊
proof–
interpret *digraph* β 𝔊 **by** (*rule assms*(1))
interpret α: $\mathcal{Z}$ α **by** (*rule assms*(2))
show *?thesis*
proof(*intro digraphI*)
show *vfsequence* 𝔊 **by** (*simp add*: *vfsequence-axioms*)
show ($\bigcup_\circ a \in_\circ A$. $\bigcup_\circ b \in_\circ B$. *Hom* 𝔊 a b) $\in_\circ$ *Vset* α
if A $\subseteq_\circ$ 𝔊(|*Obj*|) B $\subseteq_\circ$ 𝔊(|*Obj*|) A $\in_\circ$ *Vset* α B $\in_\circ$ *Vset* α **for** A B
by (*rule assms*(4)[*OF that*])
qed (*auto simp*: *assms*(3) *dg-Cod-vrange dg-cs-simps intro*!: *dg-Dom-vrange*)
qed

Further properties.

lemma (**in** *digraph*) *dg-Dom-app-in-Obj*:
assumes f $\in_\circ$ 𝔊(|*Arr*|)
shows 𝔊(|*Dom*|)(|f|) $\in_\circ$ 𝔊(|*Obj*|)
using *assms dg-Dom-vrange* **by** (*auto simp*: *Dom.vsv-vimageI*2)

lemma (**in** *digraph*) *dg-Cod-app-in-Obj*:
assumes f $\in_\circ$ 𝔊(|*Arr*|)
shows 𝔊(|*Cod*|)(|f|) $\in_\circ$ 𝔊(|*Obj*|)

using *assms dg-Cod-vrange* **by** (*auto simp*: *Cod.vsv-vimageI2*)

lemma (**in** *digraph*) *dg-Arr-vempty-if-Obj-vempty*:
assumes $\mathfrak{C}(\!|Obj|\!) = 0$
shows $\mathfrak{C}(\!|Arr|\!) = 0$
by (*metis assms eq0-iff dg-Cod-app-in-Obj*)

lemma (**in** *digraph*) *dg-Dom-vempty-if-Arr-vempty*:
assumes $\mathfrak{C}(\!|Arr|\!) = 0$
shows $\mathfrak{C}(\!|Dom|\!) = 0$
using *assms Dom.vdomain-vrange-is-vempty*
by (*auto intro*: *Dom.vsv-vrange-vempty simp*: *dg-cs-simps*)

lemma (**in** *digraph*) *dg-Cod-vempty-if-Arr-vempty*:
assumes $\mathfrak{C}(\!|Arr|\!) = 0$
shows $\mathfrak{C}(\!|Cod|\!) = 0$
using *assms Cod.vdomain-vrange-is-vempty*
by (*auto intro*: *Cod.vsv-vrange-vempty simp*: *dg-cs-simps*)

3.2.6 Opposite digraph

Definition and elementary properties

See Chapter II-2 in [39].

definition *op-dg* :: $V \Rightarrow V$
where *op-dg* $\mathfrak{C}$ = $[\mathfrak{C}(\!|Obj|\!), \mathfrak{C}(\!|Arr|\!), \mathfrak{C}(\!|Cod|\!), \mathfrak{C}(\!|Dom|\!)]_{\circ}$

Components.

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

lemma *op-dg-component-intros*[*dg-op-intros*]:
shows $a \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow a \in_{\circ}$ *op-dg* $\mathfrak{C}(\!|Obj|\!)$
and $f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \Longrightarrow f \in_{\circ}$ *op-dg* $\mathfrak{C}(\!|Arr|\!)$
unfolding *dg-op-simps* **by** *simp-all*

Elementary properties.

lemma *op-dg-is-arr*[*dg-op-simps*]: $f : b \mapsto_{op\text{-}dg\ \mathfrak{C}} a \longleftrightarrow f : a \mapsto_{\mathfrak{C}} b$
unfolding *dg-op-simps is-arr-def* **by** *auto*

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

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

Further properties

lemma (**in** *digraph*) *digraph-op*[*dg-op-intros*]: *digraph* α (*op-dg* $\mathfrak{C}$)
proof(*intro digraphI*, *unfold op-dg-components dg-op-simps*)
show *vfsequence* (*op-dg* $\mathfrak{C}$) **unfolding** *op-dg-def* **by** *simp*
show *vcard* (*op-dg* $\mathfrak{C}$) = $4_{\mathbb{N}}$
unfolding *op-dg-def* **by** (*simp add*: *nat-omega-simps*)
fix *A B* **assume** $A \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$ $B \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$ $A \in_{\circ}$ *Vset* α $B \in_{\circ}$ *Vset* α

then show $\bigcup_\circ((\lambda a \in_\circ A.\ \bigcup_\circ((\lambda aa \in_\circ B.\ \mathit{Hom}\ \mathfrak{C}\ aa\ a)\ `_\circ\ B))\ `_\circ\ A) \in_\circ \mathit{Vset}\ \alpha$
by (*subst vifunion-vifunion-flip*) (*intro dg-Hom-vifunion-in-Vset*)
qed (*auto simp*: *dg-Dom-vrange dg-Cod-vrange dg-Obj-vsubset-Vset dg-cs-simps*)

lemmas *digraph-op*[*dg-op-intros*] = *digraph.digraph-op*

lemma (**in** *digraph*) *dg-op-dg-op-dg*[*dg-op-simps*]: *op-dg* (*op-dg* $\mathfrak{C}$) = $\mathfrak{C}$
by (*rule dg-eqI*[*of* α], *unfold dg-op-simps*)
(*simp-all add*: *digraph-axioms digraph.digraph-op digraph-op*)

lemmas *dg-op-dg-op-dg*[*dg-op-simps*] = *digraph.dg-op-dg-op-dg*

lemma *eq-op-dg-iff*[*dg-op-simps*]:
assumes *digraph* α $\mathfrak{A}$ **and** *digraph* α $\mathfrak{B}$
shows *op-dg* $\mathfrak{A}$ = *op-dg* $\mathfrak{B}$ $\longleftrightarrow$ $\mathfrak{A}$ = $\mathfrak{B}$
proof
interpret $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **by** (*rule assms*(1))
interpret $\mathfrak{B}$: *digraph* α $\mathfrak{B}$ **by** (*rule assms*(2))
assume *prems*: *op-dg* $\mathfrak{A}$ = *op-dg* $\mathfrak{B}$
show $\mathfrak{A}$ = $\mathfrak{B}$
proof(*rule dg-eqI*[*of* α])
from *prems* **show**
$\mathfrak{A}(\!|Obj|\!) = \mathfrak{B}(\!|Obj|\!)$ $\mathfrak{A}(\!|Arr|\!) = \mathfrak{B}(\!|Arr|\!)$ $\mathfrak{A}(\!|Dom|\!) = \mathfrak{B}(\!|Dom|\!)$ $\mathfrak{A}(\!|Cod|\!) = \mathfrak{B}(\!|Cod|\!)$
by (*metis prems* $\mathfrak{A}$.*dg-op-dg-op-dg* $\mathfrak{B}$.*dg-op-dg-op-dg*)+
qed (*simp-all add*: *assms*)
qed *auto*

3.3 Smallness for digraphs

3.3.1 Background

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

3.3.2 Tiny digraph

Definition and elementary properties

locale *tiny-digraph* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ + *Dom*: *vsv* ‹$\mathfrak{C}(\!|Dom|\!)$› + *Cod*: *vsv* ‹$\mathfrak{C}(\!|Cod|\!)$›
for α $\mathfrak{C}$ +
assumes *tiny-dg-length*[*dg-cs-simps*]: *vcard* $\mathfrak{C}$ = $4_{\mathbb{N}}$
and *tiny-dg-Dom-vdomain*[*dg-cs-simps*]: $\mathcal{D}_\circ$ ($\mathfrak{C}(\!|Dom|\!)$) = $\mathfrak{C}(\!|Arr|\!)$
and *tiny-dg-Dom-vrange*: $\mathcal{R}_\circ$ ($\mathfrak{C}(\!|Dom|\!)$) $\subseteq_\circ$ $\mathfrak{C}(\!|Obj|\!)$
and *tiny-dg-Cod-vdomain*[*dg-cs-simps*]: $\mathcal{D}_\circ$ ($\mathfrak{C}(\!|Cod|\!)$) = $\mathfrak{C}(\!|Arr|\!)$
and *tiny-dg-Cod-vrange*: $\mathcal{R}_\circ$ ($\mathfrak{C}(\!|Cod|\!)$) $\subseteq_\circ$ $\mathfrak{C}(\!|Obj|\!)$
and *tiny-dg-Obj-in-Vset*[*dg-small-cs-intros*]: $\mathfrak{C}(\!|Obj|\!)$ $\in_\circ$ *Vset* α
and *tiny-dg-Arr-in-Vset*[*dg-small-cs-intros*]: $\mathfrak{C}(\!|Arr|\!)$ $\in_\circ$ *Vset* α

lemmas [*dg-small-cs-intros*] =
tiny-digraph.tiny-dg-Obj-in-Vset
tiny-digraph.tiny-dg-Arr-in-Vset

Rules.

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

mk-ide rf *tiny-digraph-def*[*unfolded tiny-digraph-axioms-def*]
|*intro tiny-digraphI*|
|*dest tiny-digraphD*[*dest*]|
|*elim tiny-digraphE*[*elim*]|

lemma *tiny-digraphI'*:
assumes *digraph* α $\mathfrak{C}$ **and** $\mathfrak{C}(\!|Obj|\!)$ $\in_\circ$ *Vset* α **and** $\mathfrak{C}(\!|Arr|\!)$ $\in_\circ$ *Vset* α
shows *tiny-digraph* α $\mathfrak{C}$
using *assms* **by** (*meson digraphD*(5,6,7,8,9) *digraph-def tiny-digraphI*)

Elementary properties.

sublocale *tiny-digraph* $\subseteq$ *digraph*
proof(*rule digraphI*)
from *tiny-dg-Obj-in-Vset* **show** $\mathfrak{C}(\!|Obj|\!)$ $\subseteq_\circ$ *Vset* α **by** *auto*
fix *A B* **assume** *A* $\subseteq_\circ$ $\mathfrak{C}(\!|Obj|\!)$ *B* $\subseteq_\circ$ $\mathfrak{C}(\!|Obj|\!)$ *A* $\in_\circ$ *Vset* α *B* $\in_\circ$ *Vset* α
then have ($\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.$ *Hom* $\mathfrak{C}$ *a b*) $\subseteq_\circ$ $\mathfrak{C}(\!|Arr|\!)$ **by** *auto*
with *tiny-dg-Arr-in-Vset* **show** ($\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.$ *Hom* $\mathfrak{C}$ *a b*) $\in_\circ$ *Vset* α **by** *blast*
qed
(
cs-concl **cs-shallow**
cs-simp: *dg-cs-simps*
cs-intro: *tiny-dg-Cod-vrange tiny-dg-Dom-vrange dg-cs-intros V-cs-intros*
)+

lemmas (**in** *tiny-digraph*) *tiny-dg-digraph* = *digraph-axioms*

lemmas [*dg-small-cs-intros*] = *tiny-digraph.tiny-dg-digraph*

Size.

lemma (**in** *tiny-digraph*) *tiny-dg-Dom-in-Vset*: $\mathfrak{C}(\!|Dom|\!) \in_\circ Vset\ \alpha$
proof–
  from $\mathcal{Z}$*-Limit-*$\alpha\omega$ **have** $\mathcal{D}_\circ\ (\mathfrak{C}(\!|Dom|\!)) \in_\circ Vset\ \alpha$
    by (*simp add*: *tiny-dg-Arr-in-Vset dg-cs-simps*)
  moreover from *tiny-dg-Dom-vrange* **have** $\mathcal{R}_\circ\ (\mathfrak{C}(\!|Dom|\!)) \in_\circ Vset\ \alpha$
    by (*auto intro*: *tiny-dg-Obj-in-Vset*)
  ultimately show *?thesis*
    by (*simp add*: *Dom.vbrelation-Limit-in-VsetI* $\mathcal{Z}$*-Limit-*$\alpha\omega$)
qed

lemma (**in** *tiny-digraph*) *tiny-dg-Cod-in-Vset*: $\mathfrak{C}(\!|Cod|\!) \in_\circ Vset\ \alpha$
proof–
  from $\mathcal{Z}$*-Limit-*$\alpha\omega$ **have** $\mathcal{D}_\circ\ (\mathfrak{C}(\!|Cod|\!)) \in_\circ Vset\ \alpha$
    by (*simp add*: *tiny-dg-Arr-in-Vset dg-cs-simps*)
  moreover from *tiny-dg-Cod-vrange* **have** $\mathcal{R}_\circ\ (\mathfrak{C}(\!|Cod|\!)) \in_\circ Vset\ \alpha$
    by (*auto intro*: *tiny-dg-Obj-in-Vset*)
  ultimately show *?thesis*
    by (*simp add*: *Cod.vbrelation-Limit-in-VsetI* $\mathcal{Z}$*-Limit-*$\alpha\omega$)
qed

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

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

lemma *tiny-digraphs-vsubset-Vset*: *set* $\{\mathfrak{C}.\ tiny\text{-}digraph\ \alpha\ \mathfrak{C}\} \subseteq_\circ Vset\ \alpha$
  by (*rule vsubsetI*) (*simp add*: *tiny-digraph.tiny-dg-in-Vset*)

lemma (**in** *digraph*) *dg-tiny-digraph-if-ge-Limit*:
  assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
  shows *tiny-digraph* $\beta\ \mathfrak{C}$
proof(*intro tiny-digraphI'*)
  interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(1))
  show *digraph* $\beta\ \mathfrak{C}$
    by (*intro dg-digraph-if-ge-Limit*)
      (*use assms*(2) **in** ‹*cs-concl cs-shallow cs-intro*: *dg-cs-intros*›)+
  show $\mathfrak{C}(\!|Obj|\!) \in_\circ Vset\ \beta\ \mathfrak{C}(\!|Arr|\!) \in_\circ Vset\ \beta$
    by (*auto simp*: $\beta.\mathcal{Z}$*-*β *assms*(2) *dg-Obj-in-Vset dg-Arr-in-Vset*)
qed

Opposite tiny digraph

lemma (**in** *tiny-digraph*) *tiny-digraph-op*: *tiny-digraph* α (*op-dg* $\mathfrak{C}$)
  by (*intro tiny-digraphI'*, *unfold dg-op-simps*)
    (*auto simp*: *tiny-dg-Obj-in-Vset tiny-dg-Arr-in-Vset dg-cs-simps dg-op-intros*)

lemmas *tiny-digraph-op*[*dg-op-intros*] = *tiny-digraph.tiny-digraph-op*

3.3.3 Finite digraph

Definition and elementary properties

A finite digraph is a generalization of the concept of a finite category, as presented in nLab [3][2].

locale *finite-digraph* = *digraph* α $\mathfrak{C}$ **for** α $\mathfrak{C}$ +
  assumes *fin-dg-Obj-vfinite*[*dg-small-cs-intros*]: *vfinite* ($\mathfrak{C}$⦇*Obj*⦈)
    and *fin-dg-Arr-vfinite*[*dg-small-cs-intros*]: *vfinite* ($\mathfrak{C}$⦇*Arr*⦈)

lemmas [*dg-small-cs-intros*] =
  finite-digraph.fin-dg-Obj-vfinite
  finite-digraph.fin-dg-Arr-vfinite

Rules.

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

mk-ide rf *finite-digraph-def*[*unfolded finite-digraph-axioms-def*]
  |*intro finite-digraphI*|
  |*dest finite-digraphD*[*dest*]|
  |*elim finite-digraphE*[*elim*]|

Elementary properties.

sublocale *finite-digraph* ⊆ *tiny-digraph*
proof(*rule tiny-digraphI'*)
  show $\mathfrak{C}$⦇*Obj*⦈ ∈$_\circ$ *Vset* α
    by
      (
        cs-concl **cs-shallow cs-intro**:
          dg-small-cs-intros V-cs-intros
          dg-Obj-vsubset-Vset Limit-vfinite-in-VsetI
)
  show $\mathfrak{C}$⦇*Arr*⦈ ∈$_\circ$ *Vset* α
    by
      (
        cs-concl **cs-shallow cs-intro**:
          dg-small-cs-intros V-cs-intros
          dg-Arr-vsubset-Vset Limit-vfinite-in-VsetI
)
qed (*auto intro*: *dg-cs-intros*)

lemmas (**in** *finite-digraph*) *fin-dg-tiny-digraph* = *tiny-digraph-axioms*

lemmas [*dg-small-cs-intros*] = *finite-digraph.fin-dg-tiny-digraph*

Size.

lemma *small-finite-digraphs*[*simp*]: *small* {$\mathfrak{C}$. *finite-digraph* α $\mathfrak{C}$}
proof(*rule down*)
  show {$\mathfrak{C}$. *finite-digraph* α $\mathfrak{C}$} ⊆ *elts* (*set* {$\mathfrak{C}$. *digraph* α $\mathfrak{C}$})
    by (*auto intro*: *dg-cs-intros*)

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

qed

lemma *finite-digraphs-vsubset-Vset*: *set* {𝔇. *finite-digraph* α 𝔇} $\subseteq_\circ$ *Vset* α
by
(
force simp:
tiny-digraph.tiny-dg-in-Vset finite-digraph.fin-dg-tiny-digraph
)

Opposite finite digraph

lemma (**in** *finite-digraph*) *fininte-digraph-op*: *finite-digraph* α (*op-dg* 𝔇)
by (*intro finite-digraphI*, *unfold dg-op-simps*)
(*auto simp*: *dg-small-cs-intros dg-op-intros*)

lemmas *fininte-digraph-op*[*dg-op-intros*] = *finite-digraph.fininte-digraph-op*

3.4 Homomorphism of digraphs

3.4.1 Background

named-theorems *dghm-cs-simps*
named-theorems *dghm-cs-intros*

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

named-theorems *dghm-field-simps*

definition *ObjMap* :: *V* **where** [*dghm-field-simps*]: *ObjMap* = 0
definition *ArrMap* :: *V* **where** [*dghm-field-simps*]: *ArrMap* = $1_{\mathbb{N}}$
definition *HomDom* :: *V* **where** [*dghm-field-simps*]: *HomDom* = $2_{\mathbb{N}}$
definition *HomCod* :: *V* **where** [*dghm-field-simps*]: *HomCod* = $3_{\mathbb{N}}$

3.4.2 Definition and elementary properties

A homomorphism of digraphs, as presented in this work, can be seen as a generalization of the concept of a functor between categories, as presented in Chapter I-3 in [39], to digraphs. The generalization is performed by removing the axioms (1) from the definition. It is expected that the resulting definition is consistent with the conventional notion of a homomorphism of digraphs in graph theory, but further details are considered to be outside of the scope of this work.

The definition of a digraph homomorphism is parameterized by a limit ordinal α such that $\omega < \alpha$. Such digraph homomorphisms are referred to either as α-digraph homomorphisms or homomorphisms of α-digraphs.

Following [39], all digraph homomorphisms are covariant (see Chapter II-2). However, a special notation is adapted for the digraph homomorphisms from an opposite digraph. Normally, such digraph homomorphisms will be referred to as the contravariant digraph homomorphisms, but this convention will not be enforced.

locale *is-dghm* =
  $\mathcal{Z}$ α + *vfsequence* $\mathfrak{F}$ + *HomDom*: *digraph* α $\mathfrak{A}$ + *HomCod*: *digraph* α $\mathfrak{B}$
  for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *dghm-length*[*dg-cs-simps*]: *vcard* $\mathfrak{F}$ = $4_{\mathbb{N}}$
    and *dghm-HomDom*[*dg-cs-simps*]: $\mathfrak{F}(\lfloor HomDom \rfloor) = \mathfrak{A}$
    and *dghm-HomCod*[*dg-cs-simps*]: $\mathfrak{F}(\lfloor HomCod \rfloor) = \mathfrak{B}$
    and *dghm-ObjMap-vsv*: *vsv* $(\mathfrak{F}(\lfloor ObjMap \rfloor))$
    and *dghm-ArrMap-vsv*: *vsv* $(\mathfrak{F}(\lfloor ArrMap \rfloor))$
    and *dghm-ObjMap-vdomain*[*dg-cs-simps*]: $\mathcal{D}_\circ\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
    and *dghm-ObjMap-vrange*: $\mathcal{R}_\circ\ (\mathfrak{F}(\lfloor ObjMap \rfloor)) \subseteq_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
    and *dghm-ArrMap-vdomain*[*dg-cs-simps*]: $\mathcal{D}_\circ\ (\mathfrak{F}(\lfloor ArrMap \rfloor)) = \mathfrak{A}(\lfloor Arr \rfloor)$
    and *dghm-ArrMap-is-arr*:
      $f : a \mapsto_{\mathfrak{A}} b \Longrightarrow \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$

syntax *-is-dghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\mapsto\!\!\mapsto_{DG 1}$ -)$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-dghm* $\rightleftharpoons$ *is-dghm*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{DG\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

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

syntax *-is-cn-dghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $_{DG}\!\mapsto\!\!\mapsto_1$ -)$\rangle$ [51, 51, 51] 51)

syntax-consts *-is-cn-dghm* ⇌ *is-cn-dghm*
translations $\mathfrak{F} : \mathfrak{A}\ {}_{DG}{\mapsto\!\mapsto}_{\alpha}\ \mathfrak{B} \rightharpoonup CONST\ is\text{-}cn\text{-}dghm\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$

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

abbreviation *dghms* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where $dghms\ \alpha\ \mathfrak{A}\ \mathfrak{B} \equiv set\ \{\mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\!\mapsto_{DG\alpha} \mathfrak{B}\}$

sublocale *is-dghm* ⊆ *ObjMap*: *vsv* ‹$\mathfrak{F}(\!|ObjMap|\!)$›
  rewrites $\mathcal{D}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
  by (*rule dghm-ObjMap-vsv*) (*simp add*: *dg-cs-simps*)

sublocale *is-dghm* ⊆ *ArrMap*: *vsv* ‹$\mathfrak{F}(\!|ArrMap|\!)$›
  rewrites $\mathcal{D}_{\circ}\ (\mathfrak{F}(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$
  by (*rule dghm-ArrMap-vsv*) (*simp add*: *dg-cs-simps*)

lemmas [*dg-cs-simps*] =
  is-dghm.dghm-HomDom
  is-dghm.dghm-HomCod
  is-dghm.dghm-ObjMap-vdomain
  is-dghm.dghm-ArrMap-vdomain

lemma (**in** *is-dghm*) *dghm-ArrMap-is-arr''*[*dg-cs-intros*]:
  assumes $f : a \mapsto_{\mathfrak{A}} b$ **and** $\mathfrak{F}f = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
  shows $\mathfrak{F}f : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
  using *assms*(1) **unfolding** *assms*(2) **by** (*rule dghm-ArrMap-is-arr*)

lemma (**in** *is-dghm*) *dghm-ArrMap-is-arr'*[*dg-cs-intros*]:
  assumes $f : a \mapsto_{\mathfrak{A}} b$
    and $A = \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
    and $B = \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
  shows $\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) : A \mapsto_{\mathfrak{B}} B$
  using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule dghm-ArrMap-is-arr*)

lemmas [*dg-cs-intros*] = *is-dghm.dghm-ArrMap-is-arr'*

Rules.

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

mk-ide rf *is-dghm-def*[*unfolded is-dghm-axioms-def*]
  |*intro is-dghmI*|
  |*dest is-dghmD*[*dest*]|
  |*elim is-dghmE*[*elim*]|

lemmas [*dg-cs-intros*] = *is-dghmD*(3,4)

Elementary properties.

lemma *dghm-eqI*:
  assumes $\mathfrak{G} : \mathfrak{A} \mapsto\!\mapsto_{DG\alpha} \mathfrak{B}$
    and $\mathfrak{F} : \mathfrak{C} \mapsto\!\mapsto_{DG\alpha} \mathfrak{D}$
    and $\mathfrak{G}(\!|ObjMap|\!) = \mathfrak{F}(\!|ObjMap|\!)$
    and $\mathfrak{G}(\!|ArrMap|\!) = \mathfrak{F}(\!|ArrMap|\!)$
    and $\mathfrak{A} = \mathfrak{C}$
    and $\mathfrak{B} = \mathfrak{D}$

shows $\mathfrak{G} = \mathfrak{F}$
proof–
interpret *L*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom*: $\mathcal{D}_\circ\ \mathfrak{G} = 4_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps V-cs-simps*)
from *assms*(5,6) **have** *sup*: $\mathfrak{G}(\!|HomDom|\!) = \mathfrak{F}(\!|HomDom|\!)\ \mathfrak{G}(\!|HomCod|\!) = \mathfrak{F}(\!|HomCod|\!)$
by (*simp-all add*: *dg-cs-simps*)
show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{G} \Longrightarrow \mathfrak{G}(\!|a|\!) = \mathfrak{F}(\!|a|\!)$ **for** *a*
by (*unfold dom*, *elim-in-numeral*, *insert assms*(3,4) *sup*)
(*auto simp*: *dghm-field-simps*)
qed (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps V-cs-simps* **cs-intro**: *V-cs-intros*)+
qed

lemma (**in** *is-dghm*) *dghm-def*: $\mathfrak{F} = [\mathfrak{F}(\!|ObjMap|\!), \mathfrak{F}(\!|ArrMap|\!), \mathfrak{F}(\!|HomDom|\!), \mathfrak{F}(\!|HomCod|\!)]_\circ$
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ\ \mathfrak{F} = 4_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps V-cs-simps*)
have *dom-rhs*: $\mathcal{D}_\circ\ [\mathfrak{F}(\!|ObjMap|\!), \mathfrak{F}(\!|ArrMap|\!), \mathfrak{F}(\!|HomDom|\!), \mathfrak{F}(\!|HomCod|\!)]_\circ = 4_{\mathbb{N}}$
by (*simp add*: *nat-omega-simps*)
then show $\mathcal{D}_\circ\ \mathfrak{F} = \mathcal{D}_\circ\ [\mathfrak{F}(\!|ObjMap|\!), \mathfrak{F}(\!|ArrMap|\!), \mathfrak{F}(\!|HomDom|\!), \mathfrak{F}(\!|HomCod|\!)]_\circ$
unfolding *dom-lhs dom-rhs* **by** (*simp add*: *nat-omega-simps*)
show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{F} \Longrightarrow \mathfrak{F}(\!|a|\!) = [\mathfrak{F}(\!|ObjMap|\!), \mathfrak{F}(\!|ArrMap|\!), \mathfrak{F}(\!|HomDom|\!), \mathfrak{F}(\!|HomCod|\!)]_\circ(\!|a|\!)$
for *a*
by (*unfold dom-lhs*, *elim-in-numeral*, *unfold dghm-field-simps*)
(*simp-all add*: *nat-omega-simps*)
qed (*auto simp*: *vsv-axioms*)

lemma (**in** *is-dghm*) *dghm-ObjMap-app-in-HomCod-Obj*[*dg-cs-intros*]:
assumes $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
shows $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \in_\circ \mathfrak{B}(\!|Obj|\!)$
using *assms dghm-ObjMap-vrange* **by** (*blast dest*: *ObjMap.vsv-vimageI2*)

lemmas [*dg-cs-intros*] = *is-dghm.dghm-ObjMap-app-in-HomCod-Obj*

lemma (**in** *is-dghm*) *dghm-ArrMap-vrange*: $\mathcal{R}_\circ\ (\mathfrak{F}(\!|ArrMap|\!)) \subseteq_\circ \mathfrak{B}(\!|Arr|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold dg-cs-simps*)
fix *f* **assume** $f \in_\circ \mathfrak{A}(\!|Arr|\!)$
then obtain *a b* **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 (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
then show $\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) \in_\circ \mathfrak{B}(\!|Arr|\!)$ **by** *auto*
qed *auto*

lemma (**in** *is-dghm*) *dghm-ArrMap-app-in-HomCod-Arr*[*dg-cs-intros*]:
assumes $a \in_\circ \mathfrak{A}(\!|Arr|\!)$
shows $\mathfrak{F}(\!|ArrMap|\!)(\!|a|\!) \in_\circ \mathfrak{B}(\!|Arr|\!)$
using *assms dghm-ArrMap-vrange* **by** (*blast dest*: *ArrMap.vsv-vimageI2*)

lemmas [*dg-cs-intros*] = *is-dghm.dghm-ArrMap-app-in-HomCod-Arr*

Size.

lemma (**in** *is-dghm*) *dghm-ObjMap-vsubset-Vset*: $\mathfrak{F}(\!|ObjMap|\!) \subseteq_\circ Vset\ \alpha$
by
(
rule ObjMap.vbrelation-Limit-vsubset-VsetI,

```
    insert dghm-ObjMap-vrange HomCod.dg-Obj-vsubset-Vset
  )
  (auto intro!: HomDom.dg-Obj-vsubset-Vset)

lemma (in is-dghm) dghm-ArrMap-vsubset-Vset: 𝔉(‖ArrMap‖) ⊆∘ Vset α
  by
  (
    rule ArrMap.vbrelation-Limit-vsubset-VsetI,
    insert dghm-ArrMap-vrange HomCod.dg-Arr-vsubset-Vset
  )
  (auto intro!: HomDom.dg-Arr-vsubset-Vset)

lemma (in is-dghm) dghm-ObjMap-in-Vset:
  assumes α ∈∘ β
  shows 𝔉(‖ObjMap‖) ∈∘ Vset β
  by (meson assms dghm-ObjMap-vsubset-Vset Vset-in-mono vsubset-in-VsetI)

lemma (in is-dghm) dghm-ArrMap-in-Vset:
  assumes α ∈∘ β
  shows 𝔉(‖ArrMap‖) ∈∘ Vset β
  by (meson assms dghm-ArrMap-vsubset-Vset Vset-in-mono vsubset-in-VsetI)

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

lemma (in is-dghm) dghm-is-dghm-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔉 : 𝔄 ↦↦DGβ 𝔅
proof(rule is-dghmI)
  from is-dghm-axioms assms show digraph β 𝔄
    by (cs-concl cs-intro: digraph.dg-digraph-if-ge-Limit dg-cs-intros)
  from is-dghm-axioms assms show digraph β 𝔅
    by (cs-concl cs-intro: digraph.dg-digraph-if-ge-Limit dg-cs-intros)
qed
  (
    cs-concl
      cs-simp: dg-cs-simps
      cs-intro: assms(1) dg-cs-intros V-cs-intros dghm-ObjMap-vrange
  )+

lemma small-all-dghms[simp]: small {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦DGα 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-dghm.dghm-in-Vset show ?thesis
    by (intro down[of - ‹Vset (α + ω)›] subsetI)
      (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 ↦↦_{DGα} 𝔅} = {} by auto
  then show ?thesis by simp
qed

lemma (in is-dghm) dghm-in-Vset-7: 𝔉 ∈∘ Vset (α + 7_ℕ)
proof–
  note [folded VPow-iff, folded Vset-succ[OF Ord-α], dg-cs-intros] =
    dghm-ObjMap-vsubset-Vset
    dghm-ArrMap-vsubset-Vset
  from HomDom.dg-digraph-in-Vset-4 have [dg-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.dg-digraph-in-Vset-4 have [dg-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 dghm-def, succ-of-numeral)
      (
        cs-concl
          cs-simp: plus-V-succ-right V-cs-simps dg-cs-simps
          cs-intro: dg-cs-intros V-cs-intros
      )
qed

lemma (in 𝒵) all-dghms-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows all-dghms α ∈∘ Vset β
proof(rule vsubset-in-VsetI)
  interpret β: 𝒵 β by (rule assms(1))
  show all-dghms α ⊆∘ Vset (α + 7_ℕ)
  proof(intro vsubsetI)
    fix 𝔉 assume 𝔉 ∈∘ all-dghms α
    then obtain 𝔄 𝔅 where 𝔉: 𝔉 : 𝔄 ↦↦_{DGα} 𝔅 by clarsimp
    interpret is-dghm α 𝔄 𝔅 𝔉 using 𝔉 by simp
    show 𝔉 ∈∘ Vset (α + 7_ℕ) by (rule dghm-in-Vset-7)
  qed
  from assms(2) show Vset (α + 7_ℕ) ∈∘ Vset β
    by (cs-concl cs-shallow cs-intro: V-cs-intros Ord-cs-intros)
qed

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

Further properties.

```
lemma (in is-dghm) dghm-is-arr-HomCod:
  assumes f : a ↦_𝔄 b
  shows 𝔉(|ArrMap|)(|f|) ∈∘ 𝔅(|Arr|) 𝔉(|ObjMap|)(|a|) ∈∘ 𝔅(|Obj|) 𝔉(|ObjMap|)(|b|) ∈∘ 𝔅(|Obj|)
  using assms by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)+

lemma (in is-dghm) dghm-vimage-dghm-ArrMap-vsubset-Hom:
  assumes a ∈∘ 𝔄(|Obj|) and b ∈∘ 𝔄(|Obj|)
  shows 𝔉(|ArrMap|) `∘ Hom 𝔄 a b ⊆∘ Hom 𝔅 (𝔉(|ObjMap|)(|a|)) (𝔉(|ObjMap|)(|b|))
proof(intro vsubsetI)
  fix g assume g ∈∘ 𝔉(|ArrMap|) `∘ Hom 𝔄 a b
```

then obtain f **where** $f \in_\circ Hom\ (\mathfrak{F}(\!|HomDom|\!))\ a\ b$ **and** $g = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
by (*auto simp*: *dg-cs-simps*)
then show $g \in_\circ Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
by (*simp add*: *dghm-ArrMap-is-arr dg-cs-simps*)
qed

3.4.3 Opposite digraph homomorphism

Definition and elementary properties

See Chapter II-2 in [39].

definition *op-dghm* :: $V \Rightarrow V$
where *op-dghm* $\mathfrak{F}$ =
$[\mathfrak{F}(\!|ObjMap|\!), \mathfrak{F}(\!|ArrMap|\!), op\text{-}dg\ (\mathfrak{F}(\!|HomDom|\!)), op\text{-}dg\ (\mathfrak{F}(\!|HomCod|\!))]_\circ$

Components.

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

Further properties

lemma (**in** *is-dghm*) *is-dghm-op*: *op-dghm* $\mathfrak{F} : op\text{-}dg\ \mathfrak{A} \mapsto\!\!\mapsto_{DG\alpha} op\text{-}dg\ \mathfrak{B}$
proof(*intro is-dghmI*, *unfold dg-op-simps*)
show *vfsequence* (*op-dghm* $\mathfrak{F}$) **unfolding** *op-dghm-def* **by** *simp*
show *vcard* (*op-dghm* $\mathfrak{F}$) = $4_{\mathbb{N}}$
unfolding *op-dghm-def* **by** (*auto simp*: *nat-omega-simps*)
qed
(
cs-concl **cs-shallow**
cs-intro: *dghm-ObjMap-vrange dg-cs-intros dg-op-intros V-cs-intros*
cs-simp: *dg-cs-simps dg-op-simps*
)+

lemma (**in** *is-dghm*) *is-dghm-op'*[*dg-op-intros*]:
assumes $\mathfrak{A}'$ = *op-dg* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-dg* $\mathfrak{B}$ **and** $\alpha' = \alpha$
shows *op-dghm* $\mathfrak{F} : \mathfrak{A}' \mapsto\!\!\mapsto_{DG_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-dghm-op*)

lemmas *is-dghm-op*[*dg-op-intros*] = *is-dghm.is-dghm-op'*

lemma (**in** *is-dghm*) *dghm-op-dghm-op-dghm*[*dg-op-simps*]: *op-dghm* (*op-dghm* $\mathfrak{F}$) = $\mathfrak{F}$
using *is-dghm-axioms*
by
(
cs-concl **cs-shallow**
cs-simp: *dg-op-simps*
cs-intro: *dg-op-intros dghm-eqI*[**where** $\mathfrak{F}$=$\mathfrak{F}$]
)

lemmas *dghm-op-dghm-op-dghm*[*dg-op-simps*] = *is-dghm.dghm-op-dghm-op-dghm*

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

proof
interpret *L*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}$ **by** (*rule assms*(2))
assume *prems*: *op-dghm* $\mathfrak{G}$ = *op-dghm* $\mathfrak{F}$
show $\mathfrak{G}$ = $\mathfrak{F}$
proof(*rule dghm-eqI*[*OF assms*])
from *prems R.dghm-op-dghm-op-dghm L.dghm-op-dghm-op-dghm* **show**
$\mathfrak{G}(\!|ObjMap|\!)$ = $\mathfrak{F}(\!|ObjMap|\!)$ **and** $\mathfrak{G}(\!|ArrMap|\!)$ = $\mathfrak{F}(\!|ArrMap|\!)$
by *metis+*
from *prems R.dghm-op-dghm-op-dghm L.dghm-op-dghm-op-dghm* **have**
$\mathfrak{G}(\!|HomDom|\!)$ = $\mathfrak{F}(\!|HomDom|\!)$ $\mathfrak{G}(\!|HomCod|\!)$ = $\mathfrak{F}(\!|HomCod|\!)$
by *auto*
then show $\mathfrak{A}$ = $\mathfrak{C}$ $\mathfrak{B}$ = $\mathfrak{D}$ **by** (*auto simp*: *dg-cs-simps*)
qed
qed *auto*

3.4.4 Composition of covariant digraph homomorphisms

Definition and elementary properties

See Chapter I-3 in [39].

definition *dghm-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** $\langle\circ_{DGHM}\rangle$ 55)
where $\mathfrak{G} \circ_{DGHM} \mathfrak{F}$ =
$[\mathfrak{G}(\!|ObjMap|\!) \circ_\circ \mathfrak{F}(\!|ObjMap|\!), \mathfrak{G}(\!|ArrMap|\!) \circ_\circ \mathfrak{F}(\!|ArrMap|\!), \mathfrak{F}(\!|HomDom|\!), \mathfrak{G}(\!|HomCod|\!)]_\circ$

Components.

lemma *dghm-comp-components*:
shows $(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)$ = $\mathfrak{G}(\!|ObjMap|\!) \circ_\circ \mathfrak{F}(\!|ObjMap|\!)$
and $(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ArrMap|\!)$ = $\mathfrak{G}(\!|ArrMap|\!) \circ_\circ \mathfrak{F}(\!|ArrMap|\!)$
and [*dg-shared-cs-simps*, *dg-cs-simps*]: $(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|HomDom|\!)$ = $\mathfrak{F}(\!|HomDom|\!)$
and [*dg-shared-cs-simps*, *dg-cs-simps*]: $(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|HomCod|\!)$ = $\mathfrak{G}(\!|HomCod|\!)$
unfolding *dghm-comp-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object map

lemma *dghm-comp-ObjMap-vsv*[*dg-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows *vsv* $((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!))$
proof–
interpret *L*: *is-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
show *?thesis* **by** (*cs-concl* **cs-simp**: *dghm-comp-components* **cs-intro**: *V-cs-intros*)
qed

lemma *dghm-comp-ObjMap-vdomain*[*dg-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ$ $((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!))$ = $\mathfrak{A}(\!|Obj|\!)$
using *assms*
by
(
cs-concl
cs-simp: *dghm-comp-components dg-cs-simps V-cs-simps*
cs-intro: *is-dghm.dghm-ObjMap-vrange*
)

lemma *dghm-comp-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ$ $((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!))$ $\subseteq_\circ$ $\mathfrak{C}(\!|Obj|\!)$

```
  using assms
  by
    (
      cs-concl cs-shallow
        cs-simp: dghm-comp-components
        cs-intro: is-dghm.dghm-ObjMap-vrange V-cs-intros
    )

lemma dghm-comp-ObjMap-app[dg-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦DGα ℭ and 𝔉 : 𝔄 ↦↦DGα 𝔅 and a ∈∘ 𝔄(|Obj|)
  shows (𝔊 ∘DGHM 𝔉)(|ObjMap|)(|a|) = 𝔊(|ObjMap|)(|𝔉(|ObjMap|)(|a|)|)
proof–
  interpret L: is-dghm α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  from assms(3) show (𝔊 ∘DGHM 𝔉)(|ObjMap|)(|a|) = 𝔊(|ObjMap|)(|𝔉(|ObjMap|)(|a|)|)
    by
      (
        cs-concl
          cs-simp: dghm-comp-components dg-cs-simps V-cs-simps
          cs-intro: V-cs-intros dg-cs-intros
      )
qed
```

Arrow map

```
lemma dghm-comp-ArrMap-vsv[dg-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦DGα ℭ and 𝔉 : 𝔄 ↦↦DGα 𝔅
  shows vsv ((𝔊 ∘DGHM 𝔉)(|ArrMap|))
proof–
  interpret L: is-dghm α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by (cs-concl cs-simp: dghm-comp-components cs-intro: V-cs-intros)
qed

lemma dghm-comp-ArrMap-vdomain[dg-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦DGα ℭ and 𝔉 : 𝔄 ↦↦DGα 𝔅
  shows 𝒟∘ ((𝔊 ∘DGHM 𝔉)(|ArrMap|)) = 𝔄(|Arr|)
  using assms
  by
    (
      cs-concl
        cs-simp: dghm-comp-components dg-cs-simps V-cs-simps
        cs-intro: is-dghm.dghm-ArrMap-vrange
    )

lemma dghm-comp-ArrMap-vrange[dg-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦DGα ℭ and 𝔉 : 𝔄 ↦↦DGα 𝔅
  shows ℛ∘ ((𝔊 ∘DGHM 𝔉)(|ArrMap|)) ⊆∘ ℭ(|Arr|)
  using assms
  by
    (
      cs-concl cs-shallow
        cs-simp: dghm-comp-components
        cs-intro: is-dghm.dghm-ArrMap-vrange V-cs-intros
    )

lemma dghm-comp-ArrMap-app[dg-cs-simps]:
```

```
  assumes 𝔊 : 𝔅 ↦↦_{DGα} ℭ and 𝔉 : 𝔄 ↦↦_{DGα} 𝔅 and f ∈∘ 𝔄(|Arr|)
  shows (𝔊 ∘_{DGHM} 𝔉)(|ArrMap|)(|f|) = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|f|)|)
proof–
  interpret L: is-dghm α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  from assms(3) show (𝔊 ∘_{DGHM} 𝔉)(|ArrMap|)(|f|) = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|f|)|)
    by
      (
        cs-concl
          cs-simp: dghm-comp-components dg-cs-simps V-cs-simps
          cs-intro: V-cs-intros dg-cs-intros
      )
qed
```

Opposite of the composition of covariant digraph homomorphisms

```
lemma op-dghm-dghm-comp[dg-op-simps]:
  op-dghm (𝔊 ∘_{DGHM} 𝔉) = op-dghm 𝔊 ∘_{DGHM} op-dghm 𝔉
  unfolding dghm-comp-def op-dghm-def dghm-field-simps
  by (simp add: nat-omega-simps)
```

Further properties

```
lemma dghm-comp-is-dghm[dg-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦_{DGα} ℭ and 𝔉 : 𝔄 ↦↦_{DGα} 𝔅
  shows 𝔊 ∘_{DGHM} 𝔉 : 𝔄 ↦↦_{DGα} ℭ
proof–
  interpret L: is-dghm α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
  proof(intro is-dghmI is-dghmI, unfold dg-cs-simps)
    show vfsequence (𝔊 ∘_{DGHM} 𝔉) unfolding dghm-comp-def by simp
    show vcard (𝔊 ∘_{DGHM} 𝔉) = 4_ℕ
      unfolding dghm-comp-def by (simp add: nat-omega-simps)
    fix f a b assume f : a ↦_{ℜ(𝔄)} b
    with assms show (𝔊 ∘_{DGHM} 𝔉)(|ArrMap|)(|f|) :
      (𝔊 ∘_{DGHM} 𝔉)(|ObjMap|)(|a|) ↦_ℭ (𝔊 ∘_{DGHM} 𝔉)(|ObjMap|)(|b|)
      by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  qed
    (
      use assms in
        ‹
          cs-concl cs-shallow
            cs-intro: dg-cs-intros dghm-comp-ObjMap-vrange
            cs-simp: dg-cs-simps
        ›
    )+
qed

lemma dghm-comp-assoc[dg-cs-simps]:
  assumes ℌ : ℭ ↦↦_{DGα} 𝔇 and 𝔊 : 𝔅 ↦↦_{DGα} ℭ and 𝔉 : 𝔄 ↦↦_{DGα} 𝔅
  shows (ℌ ∘_{DGHM} 𝔊) ∘_{DGHM} 𝔉 = ℌ ∘_{DGHM} (𝔊 ∘_{DGHM} 𝔉)
proof(rule dghm-eqI [of α 𝔄 𝔇 - 𝔄 𝔇])
  show (ℌ ∘_{DGHM} 𝔊 ∘_{DGHM} 𝔉)(|ObjMap|) = (ℌ ∘_{DGHM} (𝔊 ∘_{DGHM} 𝔉))(|ObjMap|)
  proof(rule vsv-eqI)
    show (ℌ ∘_{DGHM} 𝔊 ∘_{DGHM} 𝔉)(|ObjMap|)(|a|) = (ℌ ∘_{DGHM} (𝔊 ∘_{DGHM} 𝔉))(|ObjMap|)(|a|)
      if a ∈∘ 𝒟∘ ((ℌ ∘_{DGHM} 𝔊 ∘_{DGHM} 𝔉)(|ObjMap|)) for a
      using that assms
```

```
    by
      (cs-prems cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
      (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  qed (use assms in ‹cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros›)+
  show (ℌ ∘_DGHM 𝔊 ∘_DGHM 𝔉)(|ArrMap|) = (ℌ ∘_DGHM (𝔊 ∘_DGHM 𝔉))(|ArrMap|)
  proof(rule vsv-eqI)
    show (ℌ ∘_DGHM 𝔊 ∘_DGHM 𝔉)(|ArrMap|)(|a|) = (ℌ ∘_DGHM (𝔊 ∘_DGHM 𝔉))(|ArrMap|)(|a|)
      if a ∈_∘ 𝒟_∘ ((ℌ ∘_DGHM 𝔊 ∘_DGHM 𝔉)(|ArrMap|)) for a
      using that assms
      by
        (cs-prems cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
        (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  qed
    (
      use assms in
        ‹cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros›
    )+
qed (use assms in ‹cs-concl cs-shallow cs-intro: dg-cs-intros›)+
```

3.4.5 Composition of contravariant digraph homomorphisms

Definition and elementary properties

See section 1.2 in [15].

```
definition dghm-cn-comp :: V ⇒ V ⇒ V (infixl ‹_DGHM∘› 55)
  where 𝔊 _DGHM∘ 𝔉 =
    [
      𝔊(|ObjMap|) ∘_∘ 𝔉(|ObjMap|),
      𝔊(|ArrMap|) ∘_∘ 𝔉(|ArrMap|),
      op-dg (𝔉(|HomDom|)),
      𝔊(|HomCod|)
    ]_∘
```

Components.

```
lemma dghm-cn-comp-components:
  shows (𝔊 _DGHM∘ 𝔉)(|ObjMap|) = 𝔊(|ObjMap|) ∘_∘ 𝔉(|ObjMap|)
    and (𝔊 _DGHM∘ 𝔉)(|ArrMap|) = 𝔊(|ArrMap|) ∘_∘ 𝔉(|ArrMap|)
    and [dg-cn-cs-simps]: (𝔊 _DGHM∘ 𝔉)(|HomDom|) = op-dg (𝔉(|HomDom|))
    and [dg-cn-cs-simps]: (𝔊 _DGHM∘ 𝔉)(|HomCod|) = 𝔊(|HomCod|)
  unfolding dghm-cn-comp-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

Object map: two contravariant digraph homomorphisms

```
lemma dghm-cn-comp-ObjMap-vsv[dg-cn-cs-intros]:
  assumes 𝔊 : 𝔅 _DG↦↦_α ℭ and 𝔉 : 𝔄 _DG↦↦_α 𝔅
  shows vsv ((𝔊 _DGHM∘ 𝔉)(|ObjMap|))
proof–
  interpret L: is-dghm α ‹op-dg 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-dghm α ‹op-dg 𝔄› 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by (cs-concl cs-simp: dghm-cn-comp-components cs-intro: V-cs-intros)
qed

lemma dghm-cn-comp-ObjMap-vdomain[dg-cn-cs-simps]:
  assumes 𝔊 : 𝔅 _DG↦↦_α ℭ and 𝔉 : 𝔄 _DG↦↦_α 𝔅
  shows 𝒟_∘ ((𝔊 _DGHM∘ 𝔉)(|ObjMap|)) = 𝔄(|Obj|)
  using assms
  by
```

(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps dg-op-simps V-cs-simps*
cs-intro: *is-dghm.dghm-ObjMap-vrange*
)

lemma *dghm-cn-comp-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\lfloor ObjMap \rfloor)) \subseteq_{\circ} \mathfrak{C}(\lfloor Obj \rfloor)$
using *assms*
by
(
cs-concl **cs-shallow**
cs-simp: *dghm-cn-comp-components*
cs-intro: *is-dghm.dghm-ObjMap-vrange V-cs-intros*
)

lemma *dghm-cn-comp-ObjMap-app*[*dg-cn-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$ **and** $a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor)$
shows $(\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α ‹*op-dg* $\mathfrak{A}$› $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
from *assms*(3) **show** $(\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)$
by
(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps dg-op-simps V-cs-simps*
cs-intro: *V-cs-intros dg-cs-intros*
)
qed

Arrow map: two contravariant digraph homomorphisms

lemma *dghm-cn-comp-ArrMap-vsv*[*dg-cn-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
shows *vsv* $((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\lfloor ArrMap \rfloor))$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α ‹*op-dg* $\mathfrak{A}$› $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
show *?thesis*
by (*cs-concl* **cs-simp**: *dghm-cn-comp-components* **cs-intro**: *V-cs-intros*)
qed

lemma *dghm-cn-comp-ArrMap-vdomain*[*dg-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
shows $\mathcal{D}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\lfloor ArrMap \rfloor)) = \mathfrak{A}(\lfloor Arr \rfloor)$
using *assms*
by
(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps dg-op-simps V-cs-simps*
cs-intro: *is-dghm.dghm-ArrMap-vrange*
)

lemma *dghm-cn-comp-ArrMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\lfloor ArrMap \rfloor)) \subseteq_{\circ} \mathfrak{C}(\lfloor Arr \rfloor)$

using *assms*
by
(
cs-concl **cs-shallow**
cs-simp: *dghm-cn-comp-components*
cs-intro: *is-dghm.dghm-ArrMap-vrange V-cs-intros*
)

lemma *dghm-cn-comp-ArrMap-app*[*dg-cn-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{B}$ **and** $a \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
shows $(\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\!|ArrMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|a|\!)|\!)$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α ‹*op-dg* $\mathfrak{A}$› $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
from *assms*(3) **show** $(\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\!|ArrMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|a|\!)|\!)$
by
(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps dg-op-simps V-cs-simps*
cs-intro: *V-cs-intros dg-cs-intros*
)
qed

Object map: contravariant and covariant digraph homomorphisms

lemma *dghm-cn-cov-comp-ObjMap-vsv*[*dg-cn-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows *vsv* $((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\!|ObjMap|\!))$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
show *?thesis*
by (*cs-concl* **cs-simp**: *dghm-cn-comp-components* **cs-intro**: *V-cs-intros*)
qed

lemma *dghm-cn-cov-comp-ObjMap-vdomain*[*dg-cn-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathcal{D}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\!|ObjMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
using *assms*
by
(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps dg-op-simps V-cs-simps*
cs-intro: *is-dghm.dghm-ObjMap-vrange*
)

lemma *dghm-cn-cov-comp-ObjMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\mapsto\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\circ\ \mathfrak{F})(\!|ObjMap|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Obj|\!)$
using *assms*
by
(
cs-concl **cs-shallow**
cs-simp: *dghm-cn-comp-components*
cs-intro: *is-dghm.dghm-ObjMap-vrange V-cs-intros*
)

lemma *dghm-cn-cov-comp-ObjMap-app*[*dg-cn-cs-simps*]:

assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\!\mapsto\!\!\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ \mapsto\!\!\mapsto_{DG\alpha}\ \mathfrak{B}$ **and** $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
shows $(\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)|\!)$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
from *assms* **show** $(\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)|\!)$
by
(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps V-cs-simps*
cs-intro: *V-cs-intros dg-op-intros dg-cs-intros*
)
qed

Arrow map: contravariant and covariant digraph homomorphisms

lemma *dghm-cn-cov-comp-ArrMap-vsv*[*dg-cn-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\!\mapsto\!\!\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ \mapsto\!\!\mapsto_{DG\alpha}\ \mathfrak{B}$
shows *vsv* $((\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ArrMap|\!))$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
show *?thesis*
by (*cs-concl* **cs-simp**: *dghm-cn-comp-components* **cs-intro**: *V-cs-intros*)
qed

lemma *dghm-cn-cov-comp-ArrMap-vdomain*[*dg-cn-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\!\mapsto\!\!\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ \mapsto\!\!\mapsto_{DG\alpha}\ \mathfrak{B}$
shows $\mathcal{D}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$
using *assms*
by
(
cs-concl
cs-simp: *dghm-cn-comp-components dg-cs-simps dg-op-simps V-cs-simps*
cs-intro: *is-dghm.dghm-ArrMap-vrange*
)

lemma *dghm-cn-cov-comp-ArrMap-vrange*:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\!\mapsto\!\!\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ \mapsto\!\!\mapsto_{DG\alpha}\ \mathfrak{B}$
shows $\mathcal{R}_{\circ}\ ((\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ArrMap|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!)$
using *assms*
by
(
cs-concl **cs-shallow**
cs-simp: *dghm-cn-comp-components*
cs-intro: *is-dghm.dghm-ArrMap-vrange V-cs-intros*
)

lemma *dghm-cn-cov-comp-ArrMap-app*[*dg-cn-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B}\ _{DG}\!\mapsto\!\!\mapsto_{\alpha}\ \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}\ \mapsto\!\!\mapsto_{DG\alpha}\ \mathfrak{B}$ **and** $a \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
shows $(\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ArrMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|a|\!)|\!)$
proof–
interpret *L*: *is-dghm* α ‹*op-dg* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
from *assms*(3) **show** $(\mathfrak{G}\ _{DGHM}\!\circ\ \mathfrak{F})(\!|ArrMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ArrMap|\!)(\!|\mathfrak{F}(\!|ArrMap|\!)(\!|a|\!)|\!)$
by
(
cs-concl

```
      cs-simp: dghm-cn-comp-components dg-cs-simps V-cs-simps
      cs-intro: V-cs-intros dg-op-intros dg-cs-intros
  )
qed
```

Opposite of the contravariant composition of the digraph homomorphisms

```
lemma op-dghm-dghm-cn-comp[dg-op-simps]:
  op-dghm (𝔊 DGHM∘ 𝔉) = op-dghm 𝔊 DGHM∘ op-dghm 𝔉
  unfolding op-dghm-def dghm-cn-comp-def dghm-field-simps
  by (auto simp: nat-omega-simps)
```

Further properties

```
lemma dghm-cn-comp-is-dghm[dg-cn-cs-intros]:
  — See section 1.2 in [15].
  assumes digraph α 𝔄 and 𝔊 : 𝔅 DG↦↦α ℭ and 𝔉 : 𝔄 DG↦↦α 𝔅
  shows 𝔊 DGHM∘ 𝔉 : 𝔄 ↦↦DGα ℭ
proof–
  interpret 𝔄: digraph α 𝔄 by (rule assms(1))
  interpret L: is-dghm α ‹op-dg 𝔅› ℭ 𝔊 using assms(2) by auto
  interpret R: is-dghm α ‹op-dg 𝔄› 𝔅 𝔉 using assms(3) by auto
  show ?thesis
  proof(intro is-dghmI, unfold dg-op-simps dg-cs-simps dg-cn-cs-simps)
    show vfsequence (𝔊 DGHM∘ 𝔉) unfolding dghm-cn-comp-def by auto
    show vcard (𝔊 DGHM∘ 𝔉) = 4ℕ
      unfolding dghm-cn-comp-def by (simp add: nat-omega-simps)
    fix f a b assume f : a ↦₂₁ b
    with assms show (𝔊 DGHM∘ 𝔉)(|ArrMap|)(|f|) :
      (𝔊 DGHM∘ 𝔉)(|ObjMap|)(|a|) ↦ℭ (𝔊 DGHM∘ 𝔉)(|ObjMap|)(|b|)
      by
        (
          cs-concl
            cs-simp: dg-cn-cs-simps
            cs-intro: dg-cs-intros dg-op-intros
        )
  qed
    (
      cs-concl
        cs-simp: dg-cs-simps dg-cn-cs-simps
        cs-intro: dghm-cn-comp-ObjMap-vrange dg-cs-intros dg-cn-cs-intros
    )+
qed

lemma dghm-cn-cov-comp-is-dghm[dg-cn-cs-intros]:
  — See section 1.2 in [15].
  assumes 𝔊 : 𝔅 DG↦↦α ℭ and 𝔉 : 𝔄 ↦↦DGα 𝔅
  shows 𝔊 DGHM∘ 𝔉 : 𝔄 DG↦↦α ℭ
proof–
  interpret L: is-dghm α ‹op-dg 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
  proof(intro is-dghmI, unfold dg-op-simps dg-cs-simps)
    show vfsequence (𝔊 DGHM∘ 𝔉) unfolding dghm-cn-comp-def by simp
    show vcard (𝔊 DGHM∘ 𝔉) = 4ℕ
      unfolding dghm-cn-comp-def by (auto simp: nat-omega-simps)
    fix f b a assume f : b ↦₂₁ a
    with assms show (𝔊 DGHM∘ 𝔉)(|ArrMap|)(|f|) :
```

```
    (𝔊 _DGHM∘ 𝔉)(⌊ObjMap⌋)(⌊a⌋) ↦_𝔈 (𝔊 _DGHM∘ 𝔉)(⌊ObjMap⌋)(⌊b⌋)
    by (cs-concl cs-simp: dg-cn-cs-simps dg-op-simps cs-intro: dg-cs-intros)
  qed
  (
    cs-concl cs-shallow
      cs-simp: dg-cs-simps dg-cn-cs-simps
      cs-intro:
        dghm-cn-cov-comp-ObjMap-vrange
        dg-cs-intros dg-cn-cs-intros dg-op-intros
  )+
qed

lemma dghm-cov-cn-comp-is-dghm:
  — See section 1.2 in [15]
  assumes 𝔊 : 𝔅 ↦↦_DGα 𝔈 and 𝔉 : 𝔄 _DG↦↦_α 𝔅
  shows 𝔊 ∘_DGHM 𝔉 : 𝔄 _DG↦↦_α 𝔈
  using assms by (rule dghm-comp-is-dghm)
```

3.4.6 Identity digraph homomorphism

Definition and elementary properties

See Chapter I-3 in [39].

```
definition dghm-id :: V ⇒ V
  where dghm-id 𝔈 = [vid-on (𝔈(⌊Obj⌋)), vid-on (𝔈(⌊Arr⌋)), 𝔈, 𝔈]∘
```

Components.

```
lemma dghm-id-components:
  shows dghm-id 𝔈(⌊ObjMap⌋) = vid-on (𝔈(⌊Obj⌋))
    and dghm-id 𝔈(⌊ArrMap⌋) = vid-on (𝔈(⌊Arr⌋))
    and [dg-shared-cs-simps, dg-cs-simps]: dghm-id 𝔈(⌊HomDom⌋) = 𝔈
    and [dg-shared-cs-simps, dg-cs-simps]: dghm-id 𝔈(⌊HomCod⌋) = 𝔈
  unfolding dghm-id-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

Object map

```
mk-VLambda dghm-id-components(1)[folded VLambda-vid-on]
  |vsv dghm-id-ObjMap-vsv[dg-shared-cs-intros, dg-cs-intros]|
  |vdomain dghm-id-ObjMap-vdomain[dg-shared-cs-simps, dg-cs-simps]|
  |app dghm-id-ObjMap-app[dg-shared-cs-simps, dg-cs-simps]|

lemma dghm-id-ObjMap-vrange[dg-shared-cs-simps, dg-cs-simps]:
  ℛ∘ (dghm-id 𝔈(⌊ObjMap⌋)) = 𝔈(⌊Obj⌋)
  unfolding dghm-id-components by simp
```

Arrow map

```
mk-VLambda dghm-id-components(2)[folded VLambda-vid-on]
  |vsv dghm-id-ArrMap-vsv[dg-shared-cs-intros, dg-cs-intros]|
  |vdomain dghm-id-ArrMap-vdomain[dg-shared-cs-simps, dg-cs-simps]|
  |app dghm-id-ArrMap-app[dg-shared-cs-simps, dg-cs-simps]|

lemma dghm-id-ArrMap-vrange[dg-shared-cs-simps, dg-cs-simps]:
  ℛ∘ (dghm-id 𝔈(⌊ArrMap⌋)) = 𝔈(⌊Arr⌋)
  unfolding dghm-id-components by simp
```

Opposite identity digraph homomorphism

```
lemma op-dghm-dghm-id[dg-op-simps]: op-dghm (dghm-id 𝔈) = dghm-id (op-dg 𝔈)
```

unfolding *dghm-id-def op-dg-def op-dghm-def dghm-field-simps dg-field-simps*
by (*auto simp*: *nat-omega-simps*)

An identity digraph homomorphism is a digraph homomorphism

lemma (**in** *digraph*) *dg-dghm-id-is-dghm*: *dghm-id* $\mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$
proof(*intro is-dghmI*, *unfold dg-cs-simps*)
show *vfsequence* (*dghm-id* $\mathfrak{C}$) **unfolding** *dghm-id-def* **by** *simp*
show *vcard* (*dghm-id* $\mathfrak{C}$) = $4_{\mathbb{N}}$
unfolding *dghm-id-def* **by** (*simp add*: *nat-omega-simps*)
qed (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros V-cs-intros*)+

lemma (**in** *digraph*) *dg-dghm-id-is-dghm'*:
assumes $\mathfrak{A}$ = $\mathfrak{C}$ **and** $\mathfrak{B}$ = $\mathfrak{C}$
shows *dghm-id* $\mathfrak{C} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule dg-dghm-id-is-dghm*)

lemmas [*dg-cs-intros*] = *digraph.dg-dghm-id-is-dghm'*

Further properties

lemma (**in** *is-dghm*) *dghm-dghm-comp-dghm-id-left*: *dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$ = $\mathfrak{F}$
— See Chapter I-3 in [39]).
proof(*rule dghm-eqI* [*of* α $\mathfrak{A}$ $\mathfrak{B}$ - $\mathfrak{A}$ $\mathfrak{B}$])
show (*dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$)(|*ObjMap*|) = $\mathfrak{F}$(|*ObjMap*|)
proof(*rule vsv-eqI*)
show (*dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$)(|*ObjMap*|)(|*a*|) = $\mathfrak{F}$(|*ObjMap*|)(|*a*|)
if *a* $\in_{\circ}$ $\mathcal{D}_{\circ}$ ((*dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$)(|*ObjMap*|)) **for** *a*
using *that*
by
(*cs-prems* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
(*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
qed (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros V-cs-intros*)+
show (*dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$)(|*ArrMap*|) = $\mathfrak{F}$(|*ArrMap*|)
proof(*rule vsv-eqI*)
show (*dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$)(|*ArrMap*|)(|*a*|) = $\mathfrak{F}$(|*ArrMap*|)(|*a*|)
if *a* $\in_{\circ}$ $\mathcal{D}_{\circ}$ ((*dghm-id* $\mathfrak{B} \circ_{DGHM} \mathfrak{F}$)(|*ArrMap*|)) **for** *a*
using *that*
by
(*cs-prems* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
(*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
qed (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros V-cs-intros*)+
qed (*cs-concl* **cs-shallow cs-simp**: **cs-intro**: *dg-cs-intros*)+

lemmas [*dg-cs-simps*] = *is-dghm.dghm-dghm-comp-dghm-id-left*

lemma (**in** *is-dghm*) *dghm-dghm-comp-dghm-id-right*: $\mathfrak{F} \circ_{DGHM}$ *dghm-id* $\mathfrak{A}$ = $\mathfrak{F}$
— See Chapter I-3 in [39]).
proof(*rule dghm-eqI* [*of* α $\mathfrak{A}$ $\mathfrak{B}$ - $\mathfrak{A}$ $\mathfrak{B}$])
show ($\mathfrak{F} \circ_{DGHM}$ *dghm-id* $\mathfrak{A}$)(|*ObjMap*|) = $\mathfrak{F}$(|*ObjMap*|)
proof(*rule vsv-eqI*)
show ($\mathfrak{F} \circ_{DGHM}$ *dghm-id* $\mathfrak{A}$)(|*ObjMap*|)(|*a*|) = $\mathfrak{F}$(|*ObjMap*|)(|*a*|)
if *a* $\in_{\circ}$ $\mathcal{D}_{\circ}$ (($\mathfrak{F} \circ_{DGHM}$ *dghm-id* $\mathfrak{A}$)(|*ObjMap*|)) **for** *a*
using *that*
by
(*cs-prems* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
(*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
qed (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros V-cs-intros*)+

show $(\mathfrak{F} \circ_{DGHM} \mathit{dghm\text{-}id}\ \mathfrak{A})(\lfloor \mathit{ArrMap} \rfloor) = \mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor)$
proof(*rule vsv-eqI*)
show $(\mathfrak{F} \circ_{DGHM} \mathit{dghm\text{-}id}\ \mathfrak{A})(\lfloor \mathit{ArrMap} \rfloor)(\lfloor a \rfloor) = \mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor)(\lfloor a \rfloor)$
if $a \in_\circ \mathcal{D}_\circ\ ((\mathfrak{F} \circ_{DGHM} \mathit{dghm\text{-}id}\ \mathfrak{A})(\lfloor \mathit{ArrMap} \rfloor))$ **for** a
using *that*
by
(*cs-prems* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
(*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
qed (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros V-cs-intros*)+
qed (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)+

lemmas [*dg-cs-simps*] = *is-dghm.dghm-dghm-comp-dghm-id-right*

3.4.7 Constant digraph homomorphism

Definition and elementary properties

See Chapter III-3 in [39].

definition $\mathit{dghm\text{-}const} :: V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where $\mathit{dghm\text{-}const}\ \mathfrak{C}\ \mathfrak{D}\ a\ f =$
$[\mathit{vconst\text{-}on}\ (\mathfrak{C}(\lfloor \mathit{Obj} \rfloor))\ a,\ \mathit{vconst\text{-}on}\ (\mathfrak{C}(\lfloor \mathit{Arr} \rfloor))\ f,\ \mathfrak{C},\ \mathfrak{D}]_\circ$

Components.

lemma *dghm-const-components*:
shows $\mathit{dghm\text{-}const}\ \mathfrak{C}\ \mathfrak{D}\ a\ f(\lfloor \mathit{ObjMap} \rfloor) = \mathit{vconst\text{-}on}\ (\mathfrak{C}(\lfloor \mathit{Obj} \rfloor))\ a$
and $\mathit{dghm\text{-}const}\ \mathfrak{C}\ \mathfrak{D}\ a\ f(\lfloor \mathit{ArrMap} \rfloor) = \mathit{vconst\text{-}on}\ (\mathfrak{C}(\lfloor \mathit{Arr} \rfloor))\ f$
and [*dg-shared-cs-simps*, *dg-cs-simps*]: $\mathit{dghm\text{-}const}\ \mathfrak{C}\ \mathfrak{D}\ a\ f(\lfloor \mathit{HomDom} \rfloor) = \mathfrak{C}$
and [*dg-shared-cs-simps*, *dg-cs-simps*]: $\mathit{dghm\text{-}const}\ \mathfrak{C}\ \mathfrak{D}\ a\ f(\lfloor \mathit{HomCod} \rfloor) = \mathfrak{D}$
unfolding *dghm-const-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object map

mk-VLambda *dghm-const-components*(1)[*folded VLambda-vconst-on*]
|*vsv dghm-const-ObjMap-vsv*[*dg-shared-cs-intros*, *dg-cs-intros*]|
|*vdomain dghm-const-ObjMap-vdomain*[*dg-shared-cs-simps*, *dg-cs-simps*]|
|*app dghm-const-ObjMap-app*[*dg-shared-cs-simps*, *dg-cs-simps*]|

Arrow map

mk-VLambda *dghm-const-components*(2)[*folded VLambda-vconst-on*]
|*vsv dghm-const-ArrMap-vsv*[*dg-shared-cs-intros*, *dg-cs-intros*]|
|*vdomain dghm-const-ArrMap-vdomain*[*dg-shared-cs-simps*, *dg-cs-simps*]|
|*app dghm-const-ArrMap-app*[*dg-shared-cs-simps*, *dg-cs-simps*]|

Opposite constant digraph homomorphism

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

A constant digraph homomorphism is a digraph homomorphism

lemma *dghm-const-is-dghm*:
assumes *digraph* $\alpha\ \mathfrak{C}$ **and** *digraph* $\alpha\ \mathfrak{D}$ **and** $f : a \mapsto_{\mathfrak{D}} a$
shows $\mathit{dghm\text{-}const}\ \mathfrak{C}\ \mathfrak{D}\ a\ f : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{D}$
proof–
interpret $\mathfrak{D}$: *digraph* $\alpha\ \mathfrak{D}$ **by** (*rule assms*(2))

```
  show ?thesis
  proof(intro is-dghmI)
    show vfsequence (dghm-const 𝔈 𝔇 a f)
      unfolding dghm-const-def by simp
    show vcard (dghm-const 𝔈 𝔇 a f) = 4ℕ
      unfolding dghm-const-def by (simp add: nat-omega-simps)
  qed
    (
      use assms in
        ‹
          cs-concl
            cs-simp: dghm-const-components(1) dg-cs-simps
            cs-intro: V-cs-intros dg-cs-intros
        ›
    )+
qed

lemma dghm-const-is-dghm'[dg-cs-intros]:
  assumes digraph α 𝔈
    and digraph α 𝔇
    and f : a ↦_𝔇 a
    and 𝔄 = 𝔈
    and 𝔅 = 𝔇
  shows dghm-const 𝔈 𝔇 a f : 𝔄 ↦↦_DGα 𝔅
  using assms(1–3) unfolding assms(4,5) by (rule dghm-const-is-dghm)
```

Further properties

```
lemma (in is-dghm) dghm-dghm-comp-dghm-const[dg-cs-simps]:
  assumes digraph α 𝔈 and f : a ↦_𝔈 a
  shows dghm-const 𝔅 𝔈 a f ∘_DGHM 𝔉 = dghm-const 𝔄 𝔈 a f
proof(rule dghm-eqI)
  interpret 𝔈: digraph α 𝔈 by (rule assms(1))
  from assms(2) show dghm-const 𝔅 𝔈 a f ∘_DGHM 𝔉 : 𝔄 ↦↦_DGα 𝔈
    by (cs-concl cs-shallow cs-intro: dg-cs-intros)
  with assms(2) have ObjMap-dom-lhs:
    𝒟∘ ((dghm-const 𝔅 𝔈 a f ∘_DGHM 𝔉)(|ObjMap|)) = 𝔄(|Obj|)
    and ArrMap-dom-lhs: 𝒟∘ ((dghm-const 𝔅 𝔈 a f ∘_DGHM 𝔉)(|ArrMap|)) = 𝔄(|Arr|)
    by (cs-concl cs-simp: dg-cs-simps)+
  from assms(2) show dghm-const 𝔄 𝔈 a f : 𝔄 ↦↦_DGα 𝔈
    by (cs-concl cs-shallow cs-intro: dg-cs-intros)
  with assms(2) have ObjMap-dom-rhs: 𝒟∘ (dghm-const 𝔄 𝔈 a f(|ObjMap|)) = 𝔄(|Obj|)
    and ArrMap-dom-rhs: 𝒟∘ (dghm-const 𝔄 𝔈 a f(|ArrMap|)) = 𝔄(|Arr|)
    by (cs-concl cs-shallow cs-simp: dg-cs-simps)+
  show (dghm-const 𝔅 𝔈 a f ∘_DGHM 𝔉)(|ObjMap|) = dghm-const 𝔄 𝔈 a f(|ObjMap|)
    by (rule vsv-eqI, unfold ObjMap-dom-lhs ObjMap-dom-rhs)
      (
        use assms(2) in
          ‹cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros›
      )+
  show (dghm-const 𝔅 𝔈 a f ∘_DGHM 𝔉)(|ArrMap|) = dghm-const 𝔄 𝔈 a f(|ArrMap|)
    by (rule vsv-eqI, unfold ArrMap-dom-lhs ArrMap-dom-rhs)
      (
        use assms(2) in
          ‹cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros›
      )+
qed simp-all
```

lemmas [*dg-cs-simps*] = *is-dghm.dghm-dghm-comp-dghm-const*

3.4.8 Faithful digraph homomorphism

Definition and elementary properties

See Chapter I-3 in [39]).

locale *is-ft-dghm* = *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *ft-dghm-v11-on-Hom*:
    $[\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow v11\ (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$

syntax *-is-ft-dghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle (-:/\ -\mapsto\mapsto_{DG.faithful^1}\ -)\rangle\ [51, 51, 51]\ 51)$
syntax-consts *-is-ft-dghm* $\rightleftharpoons$ *is-ft-dghm*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.faithful\alpha} \mathfrak{B} \rightleftharpoons CONST\ is\text{-}ft\text{-}dghm\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$

Rules.

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

mk-ide rf *is-ft-dghm-def*[*unfolded is-ft-dghm-axioms-def*]
  |*intro is-ft-dghmI*|
  |*dest is-ft-dghmD*[*dest*]|
  |*elim is-ft-dghmE*[*elim*]|

lemmas [*dghm-cs-intros*] = *is-ft-dghmD*(1)

lemma *is-ft-dghmI''*:
  assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
    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 $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.faithful\alpha} \mathfrak{B}$
proof(*intro is-ft-dghmI assms*)
  interpret $\mathfrak{F}$: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(1))
  fix *a b* **assume** *prems*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ $b \in_\circ \mathfrak{A}(\!|Obj|\!)$
  have *dom-def*: $\mathcal{D}_\circ\ (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b) = Hom\ \mathfrak{A}\ a\ b$
    by (*intro vdomain-vlrestriction-vsubset vsubsetI*) (*auto simp*: *dg-cs-simps*)
  show $v11\ (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
  proof(*intro vsv.vsv-valeq-v11I*, *unfold dom-def dg-cs-simps*)
    from *prems* **show** *vsv* $(\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$ **by** *auto*
    fix *g f* **assume** *prems*:
      $g : a \mapsto_{\mathfrak{A}} b$
      $f : a \mapsto_{\mathfrak{A}} b$
      $(\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)(\!|g|\!) = (\mathfrak{F}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)(\!|f|\!)$
    from *prems*(3,1,2) **have** $\mathfrak{F}g\text{-}\mathfrak{F}f$: $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
      by
        (
          cs-prems
            cs-simp: *V-cs-simps dg-cs-simps*
            cs-intro: *V-cs-intros dg-cs-intros*
)
    show $g = f$ **by** (*rule assms*(2)[*OF prems*(1,2) $\mathfrak{F}g\text{-}\mathfrak{F}f$])
  qed
qed

Opposite faithful digraph homomorphism

lemma (in *is-ft-dghm*) *ft-dghm-op-dghm-is-ft-dghm*:
op-dghm $\mathfrak{F}$: *op-dg* $\mathfrak{A}$ $\mapsto\mapsto_{DG.faithful\alpha}$ *op-dg* $\mathfrak{B}$
by
(
rule is-ft-dghmI,
unfold dg-op-simps,
cs-concl **cs-shallow cs-intro**: *dg-cs-intros dg-op-intros*
)
(*auto simp*: *ft-dghm-v11-on-Hom*)

lemma (in *is-ft-dghm*) *ft-dghm-op-dghm-is-ft-dghm'*[*dg-op-intros*]:
assumes $\mathfrak{A}'$ = *op-dg* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-dg* $\mathfrak{B}$
shows *op-dghm* $\mathfrak{F}$: $\mathfrak{A}'$ $\mapsto\mapsto_{DG.faithful\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule ft-dghm-op-dghm-is-ft-dghm*)

lemmas *ft-dghm-op-dghm-is-ft-dghm*[*dg-op-intros*] =
is-ft-dghm.ft-dghm-op-dghm-is-ft-dghm'

The composition of faithful digraph homomorphisms is a faithful digraph homomorphism.

lemma *dghm-comp-is-ft-dghm*[*dghm-cs-intros*]:
— See Chapter I-3 in [39].
assumes $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\mapsto_{DG.faithful\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.faithful\alpha}$ $\mathfrak{B}$
shows $\mathfrak{G}$ $\circ_{DGHM}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.faithful\alpha}$ $\mathfrak{C}$
proof–
interpret *L*: *is-ft-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **using** *assms*(1) **by** *auto*
interpret *R*: *is-ft-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **using** *assms*(2) **by** *auto*
have *inj*:
$[\![a \in_\circ \mathfrak{A}(\!|Obj|\!) ; b \in_\circ \mathfrak{A}(\!|Obj|\!)]\!] \Longrightarrow v11\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
for *a b*
proof–
assume *prems*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ $b \in_\circ \mathfrak{A}(\!|Obj|\!)$
then have $\mathfrak{G}$-*hom*-$\mathfrak{B}$:
$v11\ (\mathfrak{G}(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)))$
by (*intro L.ft-dghm-v11-on-Hom*)
(*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)+
have $v11\ (\mathfrak{G}(\!|ArrMap|\!) \restriction^l_\circ (\mathfrak{F}(\!|ArrMap|\!)\ `_\circ\ Hom\ \mathfrak{A}\ a\ b))$
proof(*intro v11-vlrestriction-vsubset*[*OF* $\mathfrak{G}$-*hom*-$\mathfrak{B}$] *vsubsetI*)
fix *g* **assume** $g \in_\circ \mathfrak{F}(\!|ArrMap|\!)\ `_\circ\ Hom\ \mathfrak{A}\ a\ b$
then obtain *f* **where** *f*: $f : a \mapsto_{\mathfrak{A}} b$ **and** *g-def*: $g = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
by *auto*
from *f* **show** $g \in_\circ Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
by (*cs-concl* **cs-shallow cs-simp**: *g-def* **cs-intro**: *dg-cs-intros*)
qed
then show $v11\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ArrMap|\!) \restriction^l_\circ Hom\ \mathfrak{A}\ a\ b)$
unfolding *dghm-comp-components*
by (*intro v11-vlrestriction-vcomp*) (*auto simp*: *R.ft-dghm-v11-on-Hom prems*)
qed
then show $\mathfrak{G}$ $\circ_{DGHM}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.faithful\alpha}$ $\mathfrak{C}$
by (*intro is-ft-dghmI*, *cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*) *auto*
qed

Further properties

lemma (in *is-ft-dghm*) *ft-dghm-ArrMap-eqD*:
assumes $g : a \mapsto_{\mathfrak{A}} b$ **and** $f : a \mapsto_{\mathfrak{A}} b$ **and** $\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$

```
  shows g = f
proof–
  from assms(1) have a: a ∈₀ 𝔄(|Obj|) and b: b ∈₀ 𝔄(|Obj|) by auto
  interpret ArrMap: v11 ‹𝔉(|ArrMap|) ↾ᵗ Hom 𝔄 a b›
    by (rule ft-dghm-v11-on-Hom[OF a b])
  have dom-def: 𝒟₀ (𝔉(|ArrMap|) ↾ᵗ Hom 𝔄 a b) = Hom 𝔄 a b
    by (intro vdomain-vlrestriction-vsubset vsubsetI) (auto simp: dg-cs-simps)
  show ?thesis
  proof(rule ArrMap.v11-injective[unfolded dom-def dg-cs-simps, OF assms(1,2)])
    from assms(1,2) show
      (𝔉(|ArrMap|) ↾ᵗ Hom 𝔄 a b)(|g|) = (𝔉(|ArrMap|) ↾ᵗ Hom 𝔄 a b)(|f|)
      by
        (
          cs-concl
            cs-simp: dg-cs-simps assms(3) vsv.vlrestriction-atI
            cs-intro: V-cs-intros dg-cs-intros
        )
  qed
qed
```

3.4.9 Full digraph homomorphism

Definition and elementary properties

See Chapter I-3 in [39].

```
locale is-fl-dghm = is-dghm α 𝔄 𝔅 𝔉 for α 𝔄 𝔅 𝔉 +
  assumes fl-dghm-surj-on-Hom:
    [[ a ∈₀ 𝔄(|Obj|); b ∈₀ 𝔄(|Obj|) ]] ⟹
      𝔉(|ArrMap|) ‘₀ (Hom 𝔄 a b) = Hom 𝔅 (𝔉(|ObjMap|)(|a|)) (𝔉(|ObjMap|)(|b|))

syntax -is-fl-dghm :: V ⇒ V ⇒ V ⇒ V ⇒ bool
  (‹(- :/ - ↦↦_DG.full¹ -)› [51, 51, 51] 51)
translations 𝔉 : 𝔄 ↦↦_DG.full α 𝔅 ⇌ CONST is-fl-dghm α 𝔄 𝔅 𝔉
```

Rules.

```
lemma (in is-fl-dghm) is-fl-dghm-axioms'[dghm-cs-intros]:
  assumes α' = α and 𝔄' = 𝔄 and 𝔅' = 𝔅
  shows 𝔉 : 𝔄' ↦↦_DG.full_α' 𝔅'
  unfolding assms by (rule is-fl-dghm-axioms)

mk-ide rf is-fl-dghm-def[unfolded is-fl-dghm-axioms-def]
  |intro is-fl-dghmI|
  |dest is-fl-dghmD[dest]|
  |elim is-fl-dghmE[elim]|

lemmas [dghm-cs-intros] = is-fl-dghmD(1)
```

Opposite full digraph homomorphism

```
lemma (in is-fl-dghm) fl-dghm-op-dghm-is-fl-dghm:
  op-dghm 𝔉 : op-dg 𝔄 ↦↦_DG.full α op-dg 𝔅
  by
    (
      rule is-fl-dghmI,
      unfold dg-op-simps,
      cs-concl cs-shallow cs-intro: dg-cs-intros dg-op-intros
    )
    (auto simp: fl-dghm-surj-on-Hom)
```

lemma (in *is-fl-dghm*) *fl-dghm-op-dghm-is-fl-dghm'*[*dg-op-intros*]:
assumes $\mathfrak{A}' = op\text{-}dg\ \mathfrak{A}$ **and** $\mathfrak{B}' = op\text{-}dg\ \mathfrak{B}$
shows $op\text{-}dghm\ \mathfrak{F} : op\text{-}dg\ \mathfrak{A} \mapsto\mapsto_{DG.full\alpha} op\text{-}dg\ \mathfrak{B}$
unfolding *assms* **by** (*rule fl-dghm-op-dghm-is-fl-dghm*)

lemmas *fl-dghm-op-dghm-is-fl-dghm*[*dg-op-intros*] =
is-fl-dghm.fl-dghm-op-dghm-is-fl-dghm'

The composition of full digraph homomorphisms is a full digraph homomorphism

lemma *dghm-comp-is-fl-dghm*[*dghm-cs-intros*]:
— See Chapter I-3 in [39].
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG.full\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.full\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{DGHM} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.full\alpha} \mathfrak{C}$
proof–
interpret *L*: *is-fl-dghm* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-fl-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(2))
have *surj*:
$[\![\ a \in_\circ \mathfrak{A}(\!|Obj|\!);\ b \in_\circ \mathfrak{A}(\!|Obj|\!)\ ]\!] \Longrightarrow$
$(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ArrMap|\!)\ `_\circ\ (Hom\ \mathfrak{A}\ a\ b) =$
$Hom\ \mathfrak{C}\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|a|\!))\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|b|\!))$
for *a b*
proof–
assume *prems*: $a \in_\circ \mathfrak{A}(\!|Obj|\!)\ b \in_\circ \mathfrak{A}(\!|Obj|\!)$
have *surj-*$\mathfrak{F}$: $\mathfrak{F}(\!|ArrMap|\!)\ `_\circ\ Hom\ \mathfrak{A}\ a\ b = Hom\ \mathfrak{B}\ (\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))\ (\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
by (*rule R.fl-dghm-surj-on-Hom*[*OF prems*])
from *prems L.is-dghm-axioms R.is-dghm-axioms* **have** *comp-Obj*:
$(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{G}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)|\!)$
$(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|b|\!) = \mathfrak{G}(\!|ObjMap|\!)(\!|\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)|\!)$
by (*auto simp*: *dg-cs-simps*)
have $(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ArrMap|\!)\ `_\circ\ Hom\ \mathfrak{A}\ a\ b = \mathfrak{G}(\!|ArrMap|\!)\ `_\circ\ \mathfrak{F}(\!|ArrMap|\!)\ `_\circ\ Hom\ \mathfrak{A}\ a\ b$
unfolding *dghm-comp-components* **by** (*simp add*: *vcomp-vimage*)
also from *prems* **have**
$\ldots = Hom\ \mathfrak{C}\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|a|\!))\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|b|\!))$
unfolding *surj-*$\mathfrak{F}$ *comp-Obj*
by
(
simp add:
prems(2) *L.fl-dghm-surj-on-Hom R.dghm-ObjMap-app-in-HomCod-Obj*
)
finally show $(\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ArrMap|\!)\ `_\circ\ (Hom\ \mathfrak{A}\ a\ b) =$
$Hom\ \mathfrak{C}\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|a|\!))\ ((\mathfrak{G} \circ_{DGHM} \mathfrak{F})(\!|ObjMap|\!)(\!|b|\!))$
by *simp*
qed
show *?thesis*
by (*rule is-fl-dghmI*, *cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
(*auto simp*: *surj*)
qed

3.4.10 Fully faithful digraph homomorphism

Definition and elementary properties

See Chapter I-3 in [39].

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

syntax *-is-ff-dghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$

(‹(- :/ - ↦↦$_{DG.ff^1}$ -)› [51, 51, 51] 51)
syntax-consts *-is-ff-dghm* ⇌ *is-ff-dghm*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.ff\alpha} \mathfrak{B}$ ⇌ *CONST is-ff-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

Rules.

lemma (in *is-ff-dghm*) *is-ff-dghm-axioms'*[*dghm-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{DG.ff_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-ff-dghm-axioms*)

mk-ide rf *is-ff-dghm-def*
|*intro is-ff-dghmI*|
|*dest is-ff-dghmD*[*dest*]|
|*elim is-ff-dghmE*[*elim*]|

lemmas [*dghm-cs-intros*] = *is-ff-dghmD*

Opposite fully faithful digraph homomorphism.

lemma (in *is-ff-dghm*) *ff-dghm-op-dghm-is-ff-dghm*:
op-dghm $\mathfrak{F}$: *op-dg* $\mathfrak{A}$ ↦↦$_{DG.ff\alpha}$ *op-dg* $\mathfrak{B}$
by (*rule is-ff-dghmI*) (*cs-concl* **cs-shallow cs-intro**: *dg-op-intros*)+

lemma (in *is-ff-dghm*) *ff-dghm-op-dghm-is-ff-dghm'*[*dg-op-intros*]:
assumes $\mathfrak{A}'$ = *op-dg* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *op-dg* $\mathfrak{B}$
shows *op-dghm* $\mathfrak{F}$: $\mathfrak{A}'$ ↦↦$_{DG.ff\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule ff-dghm-op-dghm-is-ff-dghm*)

lemmas *ff-dghm-op-dghm-is-ff-dghm*[*dg-op-intros*] =
is-ff-dghm.ff-dghm-op-dghm-is-ff-dghm

The composition of fully faithful digraph homomorphisms is a fully faithful digraph homomorphism.

lemma *dghm-comp-is-ff-dghm*[*dghm-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG.ff\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.ff\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{DGHM} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.ff\alpha} \mathfrak{C}$
using *assms*
by (*intro is-ff-dghmI*, *elim is-ff-dghmE*) (*cs-concl* **cs-intro**: *dghm-cs-intros*)

3.4.11 Isomorphism of digraphs

Definition and elementary properties

See Chapter I-3 in [39].

locale *is-iso-dghm* = *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *iso-dghm-ObjMap-v11*: *v11* (ℑ(|*ObjMap*|))
and *iso-dghm-ArrMap-v11*: *v11* (ℑ(|*ArrMap*|))
and *iso-dghm-ObjMap-vrange*[*dghm-cs-simps*]: $\mathcal{R}_\circ$ (ℑ(|*ObjMap*|)) = $\mathfrak{B}$(|*Obj*|)
and *iso-dghm-ArrMap-vrange*[*dghm-cs-simps*]: $\mathcal{R}_\circ$ (ℑ(|*ArrMap*|)) = $\mathfrak{B}$(|*Arr*|)

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

sublocale *is-iso-dghm* ⊆ *ObjMap*: *v11* ‹ℑ(|*ObjMap*|)›
rewrites $\mathcal{D}_\circ$ (ℑ(|*ObjMap*|)) = $\mathfrak{A}$(|*Obj*|) **and** $\mathcal{R}_\circ$ (ℑ(|*ObjMap*|)) = $\mathfrak{B}$(|*Obj*|)

by
(
cs-concl **cs-shallow**
cs-simp: *dghm-cs-simps dg-cs-simps* **cs-intro**: *iso-dghm-ObjMap-v*11
)+

sublocale *is-iso-dghm* ⊆ *ArrMap*: *v*11 ‹$\mathfrak{F}(\!|ArrMap|\!)$›
rewrites $\mathcal{D}_{\circ}\ (\mathfrak{F}(\!|ArrMap|\!)) = \mathfrak{A}(\!|Arr|\!)$ **and** $\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ArrMap|\!)) = \mathfrak{B}(\!|Arr|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *dghm-cs-simps dg-cs-simps* **cs-intro**: *iso-dghm-ArrMap-v*11
)+

lemmas [*dghm-cs-simps*] =
is-iso-dghm.iso-dghm-ObjMap-vrange
is-iso-dghm.iso-dghm-ArrMap-vrange

Rules.

lemma (**in** *is-iso-dghm*) *is-iso-dghm-axioms'*[*dghm-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{DG.iso_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-iso-dghm-axioms*)

mk-ide rf *is-iso-dghm-def*[*unfolded is-iso-dghm-axioms-def*]
|*intro is-iso-dghmI*|
|*dest is-iso-dghmD*[*dest*]|
|*elim is-iso-dghmE*[*elim*]|

Elementary properties.

lemma (**in** *is-iso-dghm*) *iso-dghm-Obj-HomDom-if-Obj-HomCod*[*elim*]:
assumes $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$
obtains a **where** $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ **and** $b = \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
using *assms ObjMap.vrange-atD iso-dghm-ObjMap-vrange* **by** *blast*

lemma (**in** *is-iso-dghm*) *iso-dghm-Arr-HomDom-if-Arr-HomCod*[*elim*]:
assumes $g \in_{\circ} \mathfrak{B}(\!|Arr|\!)$
obtains f **where** $f \in_{\circ} \mathfrak{A}(\!|Arr|\!)$ **and** $g = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
using *assms ArrMap.vrange-atD iso-dghm-ArrMap-vrange* **by** *blast*

lemma (**in** *is-iso-dghm*) *iso-dghm-ObjMap-eqE*[*elim*]:
assumes $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
and $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $b \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $a = b \Longrightarrow P$
shows P
using *assms ObjMap.v*11*-eq-iff* **by** *auto*

lemma (**in** *is-iso-dghm*) *iso-dghm-ArrMap-eqE*[*elim*]:
assumes $\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!)$
and $f \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
and $g \in_{\circ} \mathfrak{A}(\!|Arr|\!)$
and $f = g \Longrightarrow P$
shows P
using *assms ArrMap.v*11*-eq-iff* **by** *auto*

sublocale *is-iso-dghm* ⊆ *is-ft-dghm*
by (*intro is-ft-dghmI*, *cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*) *auto*

sublocale *is-iso-dghm* ⊆ *is-fl-dghm*
proof
fix *a b* **assume** [*intro*]: $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$ $b \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
show $\mathfrak{F}(\!|ArrMap|\!)$ '$_{\circ}$ *Hom* $\mathfrak{A}$ *a b* = *Hom* $\mathfrak{B}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
proof(*intro vsubset-antisym vsubsetI*)
fix *g* **assume** *prems*: $g \in_{\circ}$ *Hom* $\mathfrak{B}$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!))$ $(\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!))$
then have *g*: $g : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$ **by** *auto*
then have *dom-g*: $\mathfrak{B}(\!|Dom|\!)(\!|g|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
and *cod-g*: $\mathfrak{B}(\!|Cod|\!)(\!|g|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|b|\!)$
by *blast+*
from *prems* **obtain** *f*
where [*intro*, *simp*]: $f \in_{\circ} \mathfrak{A}(\!|Arr|\!)$ **and** *g-def*: $g = \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
by *auto*
then obtain a' b' **where** *f*: $f : a' \mapsto_{\mathfrak{A}} b'$ **by** *blast*
then have $g : \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|b'|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *g-def dg-cs-simps* **cs-intro**: *dg-cs-intros*)
with *g* **have** $\mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!)$ **and** $\mathfrak{F}(\!|ObjMap|\!)(\!|b|\!) = \mathfrak{F}(\!|ObjMap|\!)(\!|b'|\!)$
by (*metis HomCod.dg-is-arrE cod-g*)+
with *f* **have** *a* = $\mathfrak{A}(\!|Dom|\!)(\!|f|\!)$ *b* = $\mathfrak{A}(\!|Cod|\!)(\!|f|\!)$ **by** *auto*
with *f* **show** $g \in_{\circ} \mathfrak{F}(\!|ArrMap|\!)$ '$_{\circ}$ *Hom* $\mathfrak{A}$ *a b*
by (*auto simp*: *g-def HomDom.dg-is-arrD*(4,5) *ArrMap.vsv-vimageI*1)
qed (*metis ArrMap.vsv-vimageE dghm-ArrMap-is-arr' in-Hom-iff*)
qed

sublocale *is-iso-dghm* ⊆ *is-ff-dghm* **by** *unfold-locales*

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

Opposite digraph isomorphisms

lemma (**in** *is-iso-dghm*) *is-iso-dghm-op*:
op-dghm $\mathfrak{F}$: *op-dg* $\mathfrak{A}$ $\mapsto\mapsto_{DG.iso\alpha}$ *op-dg* $\mathfrak{B}$
by (*intro is-iso-dghmI*, *unfold dg-op-simps*)
(
cs-concl **cs-shallow**
cs-simp: *dghm-cs-simps* **cs-intro**: *V-cs-intros dg-cs-intros dg-op-intros*
)+

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

lemmas *is-iso-dghm-op*[*dg-op-intros*] = *is-iso-dghm.is-iso-dghm-op'*

The composition of isomorphisms of digraphs is an isomorphism of digraphs

lemma *dghm-comp-is-iso-dghm*[*dghm-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{DGHM} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{F}$: *is-iso-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **using** *assms* **by** *auto*
interpret $\mathfrak{G}$: *is-iso-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **using** *assms* **by** *auto*
show *?thesis*
by (*intro is-iso-dghmI*, *unfold dghm-comp-components*)
(

cs-concl
cs-simp: *V-cs-simps dg-cs-simps dghm-cs-simps*
cs-intro: *dg-cs-intros V-cs-intros*
)+
qed

An identity digraph homomorphism is an isomorphism of digraphs.

lemma (in *digraph*) *dg-dghm-id-is-iso-dghm*: *dghm-id* $\mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{C}$
by (*rule is-iso-dghmI*) (*simp-all add*: *dg-dghm-id-is-dghm dghm-id-components*)

lemma (in *digraph*) *dg-dghm-id-is-iso-dghm'*[*dghm-cs-intros*]:
assumes $\mathfrak{A}' = \mathfrak{C}$ **and** $\mathfrak{B}' = \mathfrak{C}$
shows *dghm-id* $\mathfrak{C} : \mathfrak{A}' \mapsto\mapsto_{DG.iso\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule dg-dghm-id-is-iso-dghm*)

lemmas [*dghm-cs-intros*] = *digraph.dg-dghm-id-is-iso-dghm'*

3.4.12 Inverse digraph homomorphism

Definition and elementary properties

definition *inv-dghm* :: $V \Rightarrow V$
where *inv-dghm* $\mathfrak{F}$ = $[(\mathfrak{F}(\!|ObjMap|\!))^{-1}_{\circ}, (\mathfrak{F}(\!|ArrMap|\!))^{-1}_{\circ}, \mathfrak{F}(\!|HomCod|\!), \mathfrak{F}(\!|HomDom|\!)]_{\circ}$

Components.

lemma *inv-dghm-components*:
shows *inv-dghm* $\mathfrak{F}(\!|ObjMap|\!) = (\mathfrak{F}(\!|ObjMap|\!))^{-1}_{\circ}$
and *inv-dghm* $\mathfrak{F}(\!|ArrMap|\!) = (\mathfrak{F}(\!|ArrMap|\!))^{-1}_{\circ}$
and [*dghm-cs-simps*]: *inv-dghm* $\mathfrak{F}(\!|HomDom|\!) = \mathfrak{F}(\!|HomCod|\!)$
and [*dghm-cs-simps*]: *inv-dghm* $\mathfrak{F}(\!|HomCod|\!) = \mathfrak{F}(\!|HomDom|\!)$
unfolding *inv-dghm-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object map

lemma (in *is-iso-dghm*) *inv-dghm-ObjMap-v11*[*dghm-cs-intros*]:
v11 (*inv-dghm* $\mathfrak{F}(\!|ObjMap|\!)$)
unfolding *inv-dghm-components* **by** (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros*)

lemmas [*dghm-cs-intros*] = *is-iso-dghm.inv-dghm-ObjMap-v11*

lemma (in *is-iso-dghm*) *inv-dghm-ObjMap-vdomain*[*dghm-cs-simps*]:
$\mathcal{D}_{\circ}$ (*inv-dghm* $\mathfrak{F}(\!|ObjMap|\!)$) = $\mathfrak{B}(\!|Obj|\!)$
unfolding *inv-dghm-components* **by** (*cs-concl* **cs-simp**: *dghm-cs-simps V-cs-simps*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.inv-dghm-ObjMap-vdomain*

lemma (in *is-iso-dghm*) *inv-dghm-ObjMap-app*[*dghm-cs-simps*]:
assumes $a' = \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$ **and** $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
shows *inv-dghm* $\mathfrak{F}(\!|ObjMap|\!)(\!|a'|\!) = a$
unfolding *inv-dghm-components*
by (*metis assms ObjMap.vconverse-atI ObjMap.vsv-vconverse vsv.vsv-appI*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.inv-dghm-ObjMap-app*

lemma (in *is-iso-dghm*) *inv-dghm-ObjMap-vrange*[*dghm-cs-simps*]:
$\mathcal{R}_{\circ}$ (*inv-dghm* $\mathfrak{F}(\!|ObjMap|\!)$) = $\mathfrak{A}(\!|Obj|\!)$
unfolding *inv-dghm-components* **by** (*cs-concl* **cs-simp**: *dg-cs-simps V-cs-simps*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.inv-dghm-ObjMap-vrange*

Arrow map

lemma (**in** *is-iso-dghm*) *inv-dghm-ArrMap-v11*[*dghm-cs-intros*]:
v11 (*inv-dghm* $\mathfrak{F}(\lfloor ArrMap \rfloor)$)
unfolding *inv-dghm-components* **by** (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros*)

lemmas [*dghm-cs-intros*] = *is-iso-dghm.inv-dghm-ArrMap-v11*

lemma (**in** *is-iso-dghm*) *inv-dghm-ArrMap-vdomain*[*dghm-cs-simps*]:
$\mathcal{D}_\circ$ (*inv-dghm* $\mathfrak{F}(\lfloor ArrMap \rfloor)$) = $\mathfrak{B}(\lfloor Arr \rfloor)$
unfolding *inv-dghm-components* **by** (*cs-concl* **cs-simp**: *dghm-cs-simps V-cs-simps*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.inv-dghm-ArrMap-vdomain*

lemma (**in** *is-iso-dghm*) *inv-dghm-ArrMap-app*[*dghm-cs-simps*]:
assumes $a' = \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor a \rfloor)$ **and** $a \in_\circ \mathfrak{A}(\lfloor Arr \rfloor)$
shows *inv-dghm* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor a' \rfloor) = a$
unfolding *inv-dghm-components*
by (*metis assms ArrMap.vconverse-atI ArrMap.vsv-vconverse vsv.vsv-appI*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.inv-dghm-ArrMap-app*

lemma (**in** *is-iso-dghm*) *inv-dghm-ArrMap-vrange*[*dghm-cs-simps*]:
$\mathcal{R}_\circ$ (*inv-dghm* $\mathfrak{F}(\lfloor ArrMap \rfloor)$) = $\mathfrak{A}(\lfloor Arr \rfloor)$
unfolding *inv-dghm-components* **by** (*cs-concl* **cs-simp**: *dg-cs-simps V-cs-simps*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.inv-dghm-ArrMap-vrange*

Further properties

lemma (**in** *is-iso-dghm*) *iso-dghm-ObjMap-inv-dghm-ObjMap-app*[*dghm-cs-simps*]:
assumes $a \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
shows $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor$*inv-dghm* $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)\rfloor) = a$
using *assms* **by** (*cs-concl* **cs-simp**: *inv-dghm-components V-cs-simps*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.iso-dghm-ObjMap-inv-dghm-ObjMap-app*

lemma (**in** *is-iso-dghm*) *iso-dghm-ArrMap-inv-dghm-ArrMap-app*[*dghm-cs-simps*]:
assumes $f : a \mapsto_{\mathfrak{B}} b$
shows $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor$*inv-dghm* $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)\rfloor) = f$
using *assms*
by (*cs-concl* **cs-simp**: *inv-dghm-components V-cs-simps* **cs-intro**: *dg-cs-intros*)

lemmas [*dghm-cs-simps*] = *is-iso-dghm.iso-dghm-ArrMap-inv-dghm-ArrMap-app*

lemma (**in** *is-iso-dghm*) *iso-dghm-HomDom-is-arr-conv*:
assumes $f \in_\circ \mathfrak{A}(\lfloor Arr \rfloor)$
and $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
and $b \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
and $\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) : \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{B}} \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor b \rfloor)$
shows $f : a \mapsto_{\mathfrak{A}} b$
by
(
metis
assms
HomDom.dg-is-arrE

```
    is-arr-def
    dghm-ArrMap-is-arr
    iso-dghm-ObjMap-eqE
  )

lemma (in is-iso-dghm) iso-dghm-HomCod-is-arr-conv:
  assumes f ∈∘ 𝔅(|Arr|)
    and a ∈∘ 𝔅(|Obj|)
    and b ∈∘ 𝔅(|Obj|)
    and inv-dghm 𝔉(|ArrMap|)(|f|) : inv-dghm 𝔉(|ObjMap|)(|a|) ↦₂ inv-dghm 𝔉(|ObjMap|)(|b|)
  shows f : a ↦₂ b
  by
    (
      metis
        assms
        dghm-ArrMap-is-arr'
        is-arrI
        iso-dghm-ArrMap-inv-dghm-ArrMap-app
        iso-dghm-ObjMap-inv-dghm-ObjMap-app
    )
```

3.4.13 An isomorphism of digraphs is an isomorphism in the category *GRPH*

See Chapter I-3 in [39]).

```
lemma is-iso-arr-is-iso-dghm:
  assumes 𝔉 : 𝔄 ↦↦_DGα 𝔅
    and 𝔊 : 𝔅 ↦↦_DGα 𝔄
    and 𝔊 ∘_DGHM 𝔉 = dghm-id 𝔄
    and 𝔉 ∘_DGHM 𝔊 = dghm-id 𝔅
  shows 𝔉 : 𝔄 ↦↦_DG.isoα 𝔅
proof(intro is-iso-dghmI)

  interpret L: is-dghm α 𝔅 𝔄 𝔊 by (rule assms(2))
  interpret R: is-dghm α 𝔄 𝔅 𝔉 by (rule assms(1))

  show 𝔉 : 𝔄 ↦↦_DGα 𝔅 by (cs-concl cs-shallow cs-intro: dg-cs-intros)

  show v11 (𝔉(|ObjMap|))
  proof(rule R.ObjMap.vsv-valeq-v11I)
    fix a b assume prems[simp]:
      a ∈∘ 𝔄(|Obj|) b ∈∘ 𝔄(|Obj|) 𝔉(|ObjMap|)(|a|) = 𝔉(|ObjMap|)(|b|)
    from assms(1,2) have (𝔊 ∘_DGHM 𝔉)(|ObjMap|)(|a|) = (𝔊 ∘_DGHM 𝔉)(|ObjMap|)(|b|)
      by (simp add: dg-cs-simps)
    then show a = b by (simp add: assms(3) dghm-id-components)
  qed

  show v11 (𝔉(|ArrMap|))
  proof(rule R.ArrMap.vsv-valeq-v11I)
    fix a b
    assume prems[simp]:
      a ∈∘ 𝔄(|Arr|) b ∈∘ 𝔄(|Arr|) 𝔉(|ArrMap|)(|a|) = 𝔉(|ArrMap|)(|b|)
    then have 𝔉(|ArrMap|)(|a|) ∈∘ 𝔅(|Arr|)
      by (cs-concl cs-shallow cs-intro: dg-cs-intros)
    with R.dghm-ArrMap-vsv L.dghm-ArrMap-vsv R.dghm-ArrMap-vrange have
      (𝔊 ∘_DGHM 𝔉)(|ArrMap|)(|a|) = (𝔊 ∘_DGHM 𝔉)(|ArrMap|)(|b|)
      unfolding dghm-comp-components by (simp add: dg-cs-simps)
    then show a = b by (simp add: assms(3) dghm-id-components)
```

```
  qed

  show ℛ∘ (𝔉(⁅ObjMap⁆)) = 𝔅(⁅Obj⁆)
  proof(intro vsubset-antisym vsubsetI)
    from R.dghm-ObjMap-vrange show a ∈∘ ℛ∘ (𝔉(⁅ObjMap⁆)) ⟹ a ∈∘ 𝔅(⁅Obj⁆) for a
      by auto
  next
    fix a assume prems: a ∈∘ 𝔅(⁅Obj⁆)
    then have a-def[symmetric]: (𝔉 ∘DGHM 𝔊)(⁅ObjMap⁆)(⁅a⁆) = a
      unfolding assms(4) dghm-id-components by simp
    from prems show a ∈∘ ℛ∘ (𝔉(⁅ObjMap⁆))
      by (subst a-def)
        (
          cs-concl cs-shallow
            cs-intro: V-cs-intros dg-cs-intros cs-simp: dg-cs-simps
        )
  qed

  show ℛ∘ (𝔉(⁅ArrMap⁆)) = 𝔅(⁅Arr⁆)
  proof(intro vsubset-antisym vsubsetI)
    from R.dghm-ArrMap-vrange show a ∈∘ ℛ∘ (𝔉(⁅ArrMap⁆)) ⟹ a ∈∘ 𝔅(⁅Arr⁆) for a
      by auto
  next
    fix a assume prems: a ∈∘ 𝔅(⁅Arr⁆)
    then have a-def[symmetric]: (𝔉 ∘DGHM 𝔊)(⁅ArrMap⁆)(⁅a⁆) = a
      unfolding assms(4) dghm-id-components by simp
    with prems show a ∈∘ ℛ∘ (𝔉(⁅ArrMap⁆))
      by (subst a-def)
        (
          cs-concl cs-shallow
            cs-intro: V-cs-intros dg-cs-intros cs-simp: dg-cs-simps
        )
  qed

qed

lemma is-iso-dghm-is-iso-arr:
  assumes 𝔉 : 𝔄 ↦↦DG.isoα 𝔅
  shows [dghm-cs-intros]: inv-dghm 𝔉 : 𝔅 ↦↦DG.isoα 𝔄
    and [dghm-cs-simps]: inv-dghm 𝔉 ∘DGHM 𝔉 = dghm-id 𝔄
    and [dghm-cs-simps]: 𝔉 ∘DGHM inv-dghm 𝔉 = dghm-id 𝔅
proof–

  let ?𝔊 = ‹inv-dghm 𝔉›

  interpret is-iso-dghm α 𝔄 𝔅 𝔉 by (rule assms(1))

  show 𝔊: ?𝔊 : 𝔅 ↦↦DG.isoα 𝔄
  proof(intro is-iso-dghmI is-dghmI, unfold dghm-cs-simps)
    show vfsequence (inv-dghm 𝔉) unfolding inv-dghm-def by auto
    show vcard (inv-dghm 𝔉) = 4ℕ
      unfolding inv-dghm-def by (simp add: nat-omega-simps)
    show inv-dghm 𝔉(⁅ArrMap⁆)(⁅f⁆) : inv-dghm 𝔉(⁅ObjMap⁆)(⁅a⁆) ↦𝔄 inv-dghm 𝔉(⁅ObjMap⁆)(⁅b⁆)
      if f : a ↦𝔅 b for a b f
      using that
      by
        (
          intro iso-dghm-HomDom-is-arr-conv,
```

```
      use nothing in ‹unfold inv-dghm-components›
    )
    (
      cs-concl
        cs-simp: V-cs-simps dghm-cs-simps dg-cs-simps
        cs-intro: dg-cs-intros V-cs-intros
    )+
qed
  (
    cs-concl cs-shallow
      cs-simp: dg-cs-simps
      cs-intro: dg-cs-intros V-cs-intros dghm-cs-intros
  )+

show inv-dghm 𝔉 ∘_DGHM 𝔉 = dghm-id 𝔄
proof(rule dghm-eqI[of α 𝔄 𝔄 - 𝔄 𝔄])
  show (inv-dghm 𝔉 ∘_DGHM 𝔉)(|ObjMap|) = dghm-id 𝔄(|ObjMap|)
    unfolding inv-dghm-components dghm-comp-components dghm-id-components
    by (rule ObjMap.v11-vcomp-vconverse)
  show (inv-dghm 𝔉 ∘_DGHM 𝔉)(|ArrMap|) = dghm-id 𝔄(|ArrMap|)
    unfolding inv-dghm-components dghm-comp-components dghm-id-components
    by (rule ArrMap.v11-vcomp-vconverse)
qed (use 𝔊 in ‹cs-concl cs-intro: dghm-cs-intros›)

show 𝔉 ∘_DGHM inv-dghm 𝔉 = dghm-id 𝔅
proof(rule dghm-eqI[of α 𝔅 𝔅 - 𝔅 𝔅])
  show (𝔉 ∘_DGHM inv-dghm 𝔉)(|ObjMap|) = dghm-id 𝔅(|ObjMap|)
    unfolding inv-dghm-components dghm-comp-components dghm-id-components
    by (rule ObjMap.v11-vcomp-vconverse')
  show (𝔉 ∘_DGHM inv-dghm 𝔉)(|ArrMap|) = dghm-id 𝔅(|ArrMap|)
    unfolding inv-dghm-components dghm-comp-components dghm-id-components
    by (rule ArrMap.v11-vcomp-vconverse')
qed (use 𝔊 in ‹cs-concl cs-intro: dghm-cs-intros›)

qed
```

Further properties

```
lemma (in is-iso-dghm) iso-inv-dghm-ObjMap-dghm-ObjMap-app[dghm-cs-simps]:
  assumes a ∈∘ 𝔄(|Obj|)
  shows inv-dghm 𝔉(|ObjMap|)(|𝔉(|ObjMap|)(|a|)|) = a
proof–
  from is-iso-dghm-is-iso-arr[OF is-iso-dghm-axioms] have
    (inv-dghm 𝔉 ∘_DGHM 𝔉)(|ObjMap|)(|a|) = dghm-id 𝔄(|ObjMap|)(|a|)
    by simp
  from this assms show ?thesis
    by
      (
        cs-prems cs-shallow
          cs-simp: dg-cs-simps cs-intro: dg-cs-intros dghm-cs-intros
      )
qed

lemmas [dghm-cs-simps] = is-iso-dghm.iso-inv-dghm-ObjMap-dghm-ObjMap-app

lemma (in is-iso-dghm) iso-inv-dghm-ArrMap-dghm-ArrMap-app[dghm-cs-simps]:
  assumes f : a ↦_𝔄 b
  shows inv-dghm 𝔉(|ArrMap|)(|𝔉(|ArrMap|)(|f|)|) = f
```

proof–
from *is-iso-dghm-is-iso-arr*[*OF is-iso-dghm-axioms*] **have**
(*inv-dghm* $\mathfrak{F}$ $\circ_{DGHM}$ $\mathfrak{F}$)(⌊*ArrMap*⌋)(⌊f⌋) = *dghm-id* $\mathfrak{A}$(⌊*ArrMap*⌋)(⌊f⌋)
by *simp*
from *this assms* **show** *?thesis*
by
(
cs-prems **cs-shallow**
cs-simp: *dg-cs-simps* **cs-intro**: *dg-cs-intros dghm-cs-intros*
)
qed

lemmas [*dghm-cs-simps*] = *is-iso-dghm.iso-inv-dghm-ArrMap-dghm-ArrMap-app*

3.4.14 Isomorphic digraphs

Definition and elementary properties

See Chapter I-3 in [39]).

locale *iso-digraph* =
fixes α $\mathfrak{A}$ $\mathfrak{B}$ :: V
assumes *iso-digraph-is-iso-dghm*: ∃$\mathfrak{F}$. $\mathfrak{F}$: $\mathfrak{A}$ ↦↦$_{DG.iso\alpha}$ $\mathfrak{B}$

notation *iso-digraph* (**infixl** ‹≈$_{DG^1}$› 50)

sublocale *iso-digraph* ⊆ *HomDom*: *digraph* α $\mathfrak{A}$ + *HomCod*: *digraph* α $\mathfrak{B}$
using *iso-digraph-is-iso-dghm* **by** *auto*

Rules.

lemma *iso-digraphI'*:
assumes ∃$\mathfrak{F}$. $\mathfrak{F}$: $\mathfrak{A}$ ↦↦$_{DG.iso\alpha}$ $\mathfrak{B}$
shows $\mathfrak{A}$ ≈$_{DG\alpha}$ $\mathfrak{B}$
using *assms iso-digraph.intro* **by** *auto*

lemma *iso-digraphI*:
assumes $\mathfrak{F}$: $\mathfrak{A}$ ↦↦$_{DG.iso\alpha}$ $\mathfrak{B}$
shows $\mathfrak{A}$ ≈$_{DG\alpha}$ $\mathfrak{B}$
using *assms* **unfolding** *iso-digraph-def* **by** *auto*

lemma *iso-digraphD*[*dest*]:
assumes $\mathfrak{A}$ ≈$_{DG\alpha}$ $\mathfrak{B}$
shows ∃$\mathfrak{F}$. $\mathfrak{F}$: $\mathfrak{A}$ ↦↦$_{DG.iso\alpha}$ $\mathfrak{B}$
using *assms* **unfolding** *iso-digraph-def* **by** *simp-all*

lemma *iso-digraphE*[*elim*]:
assumes $\mathfrak{A}$ ≈$_{DG\alpha}$ $\mathfrak{B}$
obtains $\mathfrak{F}$ **where** $\mathfrak{F}$: $\mathfrak{A}$ ↦↦$_{DG.iso\alpha}$ $\mathfrak{B}$
using *assms* **by** *auto*

A digraph isomorphism is an equivalence relation

lemma *iso-digraph-refl*:
assumes *digraph* α $\mathfrak{A}$
shows $\mathfrak{A}$ ≈$_{DG\alpha}$ $\mathfrak{A}$
proof(*rule iso-digraphI*[*of* - - - ‹*dghm-id* $\mathfrak{A}$›])
interpret *digraph* α $\mathfrak{A}$ **by** (*rule assms*)
show *dghm-id* $\mathfrak{A}$: $\mathfrak{A}$ ↦↦$_{DG.iso\alpha}$ $\mathfrak{A}$ **by** (*rule dg-dghm-id-is-iso-dghm*)
qed

lemma *iso-digraph-sym*[*sym*]:
assumes $\mathfrak{A} \approx_{DG\alpha} \mathfrak{B}$
shows $\mathfrak{B} \approx_{DG\alpha} \mathfrak{A}$
proof–
interpret *iso-digraph* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*)
from *iso-digraph-is-iso-dghm* **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{B}$
by *clarsimp*
then have *inv-dghm* $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{DG.iso\alpha} \mathfrak{A}$
by (*simp add*: *is-iso-dghm-is-iso-arr*(1))
then show *?thesis*
by (*cs-concl* **cs-shallow cs-intro**: *dghm-cs-intros iso-digraphI*)
qed

lemma *iso-digraph-trans*[*trans*]:
assumes $\mathfrak{A} \approx_{DG\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \approx_{DG\alpha} \mathfrak{C}$
shows $\mathfrak{A} \approx_{DG\alpha} \mathfrak{C}$
proof–
interpret *L*: *iso-digraph* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(1))
interpret *R*: *iso-digraph* α $\mathfrak{B}$ $\mathfrak{C}$ **by** (*rule assms*(2))
from *L.iso-digraph-is-iso-dghm R.iso-digraph-is-iso-dghm* **show** *?thesis*
by (*meson dghm-comp-is-iso-dghm iso-digraph.intro*)
qed

3.5 Smallness for digraph homomorphisms

3.5.1 Digraph homomorphism with tiny maps

Definition and elementary properties

The following construction is based on the concept of a *small functor* used in [57] in the context of the presentation of the set theory *ZFC/S*.

locale *is-tm-dghm* =
  is-dghm α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  HomDom: *digraph* α $\mathfrak{A}$ +
  HomCod: *digraph* α $\mathfrak{B}$
  for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *tm-dghm-ObjMap-in-Vset*[*dg-small-cs-intros*]: $\mathfrak{F}(\lfloor ObjMap \rfloor) \in_{\circ} Vset\ \alpha$
    and *tm-dghm-ArrMap-in-Vset*[*dg-small-cs-intros*]: $\mathfrak{F}(\lfloor ArrMap \rfloor) \in_{\circ} Vset\ \alpha$

syntax *-is-tm-dghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  ($\langle$(- :/ - $\mapsto\mapsto_{DG.tm^1}$ -)$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-tm-dghm* $\rightleftharpoons$ *is-tm-dghm*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B} \rightleftharpoons CONST\ is\text{-}tm\text{-}dghm\ \alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$

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

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

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

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

lemmas [*dg-small-cs-intros*] =
  is-tm-dghm.tm-dghm-ObjMap-in-Vset
  is-tm-dghm.tm-dghm-ArrMap-in-Vset

Rules.

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

mk-ide rf *is-tm-dghm-def*[*unfolded is-tm-dghm-axioms-def*]
  |*intro is-tm-dghmI*|
  |*dest is-tm-dghmD*[*dest*]|
  |*elim is-tm-dghmE*[*elim*]|

lemmas [*dg-small-cs-intros*] = *is-tm-dghmD*(1)

Elementary properties.

sublocale *is-tm-dghm* $\subseteq$ *HomDom*: *tiny-digraph* α $\mathfrak{A}$
proof(*rule tiny-digraphI'*)
  show $\mathfrak{A}(\lfloor Obj \rfloor) \in_{\circ} Vset\ \alpha$
    by (*rule vdomain-in-VsetI*[*OF tm-dghm-ObjMap-in-Vset*, *simplified dg-cs-simps*])
  show $\mathfrak{A}(\lfloor Arr \rfloor) \in_{\circ} Vset\ \alpha$

by (*rule vdomain-in-VsetI*[*OF tm-dghm-ArrMap-in-Vset*, *simplified dg-cs-simps*])
qed (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)

lemmas (**in** *is-tm-dghm*)
tm-dghm-HomDom-is-tiny-digraph = *HomDom.tiny-digraph-axioms*

lemmas [*dg-small-cs-intros*] = *is-tm-dghm.tm-dghm-HomDom-is-tiny-digraph*

Further rules.

lemma *is-tm-dghmI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
and [*simp*]: $\mathfrak{F}(\lfloor ObjMap \rfloor) \in_\circ Vset\ \alpha$
and [*simp*]: $\mathfrak{F}(\lfloor ArrMap \rfloor) \in_\circ Vset\ \alpha$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B}$
proof–
interpret *is-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(1))
from *assms* **show** *?thesis*
by (*intro is-tm-dghmI*) (*auto simp*: *vfsequence-axioms dghm-ObjMap-vrange*)
qed

Size.

lemma *small-all-tm-dghms*[*simp*]: *small* $\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B}\}$
proof(*rule down*)
show
$\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B}\} \subseteq$
elts (*set* $\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}\}$)
proof
(
simp only: *elts-of-set small-all-dghms if-True*,
rule subsetI,
unfold mem-Collect-eq
)
fix $\mathfrak{F}$ **assume** $\exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B}$
then obtain $\mathfrak{A}\ \mathfrak{B}$ **where** $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B}$ **by** *clarsimp*
interpret *is-tm-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
from *is-dghm-axioms'* **show** $\exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$ **by** *blast*
qed
qed

Opposite digraph homomorphism with tiny maps

lemma (**in** *is-tm-dghm*) *is-tm-dghm-op*: *op-dghm* $\mathfrak{F}$: *op-dg* $\mathfrak{A} \mapsto\mapsto_{DG.tm\alpha}$ *op-dg* $\mathfrak{B}$
by (*intro is-tm-dghmI'*, *unfold dg-op-simps*)
(*cs-concl* **cs-intro**: *dg-cs-intros dg-small-cs-intros dg-op-intros*)

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

lemmas *is-tm-dghm-op*[*dg-op-intros*] = *is-tm-dghm.is-tm-dghm-op'*

Composition of a digraph homomorphism with tiny maps

lemma *dghm-comp-is-tm-dghm*[*dg-small-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{DGHM} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{C}$
proof–

```
  interpret 𝔉: is-tm-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  interpret 𝔊: is-tm-dghm α 𝔅 ℭ 𝔊 by (rule assms(1))
  show ?thesis
  proof(intro is-tm-dghmI')
    from assms show (𝔊 ∘_DGHM 𝔉)(|ObjMap|) ∈∘ Vset α
      by
        (
          cs-concl cs-shallow
            cs-simp: dghm-comp-components
            cs-intro: dg-small-cs-intros Limit-vcomp-in-VsetI 𝔉.Limit-α
        )+
    from assms show (𝔊 ∘_DGHM 𝔉)(|ArrMap|) ∈∘ Vset α
      by
        (
          cs-concl cs-shallow
            cs-simp: dghm-comp-components
            cs-intro: dg-small-cs-intros Limit-vcomp-in-VsetI 𝔉.Limit-α
        )+
  qed (auto intro: dg-cs-intros)
qed
```

Finite digraphs and digraph homomorphisms with tiny maps

```
lemma (in is-dghm) dghm-is-tm-dghm-if-HomDom-finite-digraph:
  assumes finite-digraph α 𝔄
  shows 𝔉 : 𝔄 ↦↦_DG.tmα 𝔅
proof(intro is-tm-dghmI')
  interpret 𝔄: finite-digraph α 𝔄 by (rule assms(1))
  show 𝔉(|ObjMap|) ∈∘ Vset α
  proof(rule ObjMap.vsv-Limit-vsv-in-VsetI)
    show ℛ∘ (𝔉(|ObjMap|)) ⊆∘ Vset α
    proof–
      have ℛ∘ (𝔉(|ObjMap|)) ⊆∘ 𝔅(|Obj|) by (simp add: dghm-ObjMap-vrange)
      moreover have 𝔅(|Obj|) ⊆∘ Vset α
        by (simp add: HomCod.dg-Obj-vsubset-Vset)
      ultimately show ?thesis by auto
    qed
  qed (auto simp: dg-cs-simps dg-small-cs-intros)
  show 𝔉(|ArrMap|) ∈∘ Vset α
  proof(rule ArrMap.vsv-Limit-vsv-in-VsetI)
    show ℛ∘ (𝔉(|ArrMap|)) ⊆∘ Vset α
    proof–
      have ℛ∘ (𝔉(|ArrMap|)) ⊆∘ 𝔅(|Arr|) by (simp add: dghm-ArrMap-vrange)
      moreover have 𝔅(|Arr|) ⊆∘ Vset α
        by (simp add: HomCod.dg-Arr-vsubset-Vset)
      ultimately show ?thesis by auto
    qed
  qed (auto simp: dg-cs-simps dg-small-cs-intros)
qed (simp add: dg-cs-intros)
```

Constant digraph homomorphism

```
lemma dghm-const-is-tm-dghm:
  assumes tiny-digraph α ℭ and digraph α 𝔇 and f : a ↦_𝔇 a
  shows dghm-const ℭ 𝔇 a f : ℭ ↦↦_DG.tmα 𝔇
proof(intro is-tm-dghmI')
  interpret ℭ: tiny-digraph α ℭ by (rule assms(1))
  interpret 𝔇: digraph α 𝔇 by (rule assms(2))
```

```
  from assms show dghm-const 𝔈 𝔇 a f : 𝔈 ↦↦_DGα 𝔇
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  show dghm-const 𝔈 𝔇 a f(|ObjMap|) ∈₀ Vset α
    unfolding dghm-const-components
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    from assms(3) have a ∈₀ set {a}
      by (cs-concl cs-shallow cs-intro: V-cs-intros)
    with assms(3) show ℛ₀ (vconst-on (𝔈(|Obj|)) a) ∈₀ Vset α
      by
        (
          cs-concl cs-intro:
            dg-cs-intros
            V-cs-intros
            𝔇.dg-in-Obj-in-Vset
            vsubset-in-VsetI
            Limit-vsingleton-in-VsetI
        )
    show 𝒟₀ (vconst-on (𝔈(|Obj|)) a) ∈₀ Vset α
      by (cs-concl cs-simp: V-cs-simps cs-intro: V-cs-intros dg-small-cs-intros)
  qed simp-all
  show dghm-const 𝔈 𝔇 a f(|ArrMap|) ∈₀ Vset α
    unfolding dghm-const-components
  proof(rule vbrelation.vbrelation-Limit-in-VsetI)
    from assms(3) 𝔇.dg-Arr-vsubset-Vset show
      ℛ₀ (vconst-on (𝔈(|Arr|)) f) ∈₀ Vset α
      by (cases ‹𝔈(|Arr|)=0›)
        (
          auto
            simp: dg-cs-simps 𝔇.dg-is-arrD(1)
            intro!: Limit-vsingleton-in-VsetI
        )
  qed (auto simp: 𝔈.tiny-dg-Arr-in-Vset)
qed

lemma dghm-const-is-tm-dghm'[dg-small-cs-intros]:
  assumes tiny-digraph α 𝔈
    and digraph α 𝔇
    and f : a ↦_𝔇 a
    and 𝔈' = 𝔈
    and 𝔇' = 𝔇
  shows dghm-const 𝔈 𝔇 a f : 𝔈' ↦↦_DG.tmα 𝔇'
  using assms(1–3) unfolding assms(4,5) by (rule dghm-const-is-tm-dghm)
```

3.5.2 Tiny digraph homomorphism

Definition and elementary properties

```
locale is-tiny-dghm =
  is-dghm α 𝔄 𝔅 𝔉 +
  HomDom: tiny-digraph α 𝔄 +
  HomCod: tiny-digraph α 𝔅
  for α 𝔄 𝔅 𝔉

syntax -is-tiny-dghm :: V ⇒ V ⇒ V ⇒ V ⇒ bool
  (‹(- :/ - ↦↦_DG.tiny1 -)› [51, 51, 51] 51)
syntax-consts -is-tiny-dghm ⇌ is-tiny-dghm
translations 𝔉 : 𝔄 ↦↦_DG.tinyα 𝔅 ⇌ CONST is-tiny-dghm α 𝔄 𝔅 𝔉
```

abbreviation (*input*) *is-cn-tiny-dghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  where *is-cn-tiny-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F} \equiv \mathfrak{F} : \mathit{op\text{-}dg}\ \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}$

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

abbreviation *all-tiny-dghms* :: $V \Rightarrow V$
  where *all-tiny-dghms* $\alpha \equiv \mathit{set}\ \{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$

abbreviation *small-dghms* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *small-dghms* $\alpha\ \mathfrak{A}\ \mathfrak{B} \equiv \mathit{set}\ \{\mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$

Rules.

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

mk-ide rf *is-tiny-dghm-def*
  |*intro is-tiny-dghmI*|
  |*dest is-tiny-dghmD*[*dest*]|
  |*elim is-tiny-dghmE*[*elim*]|

lemmas [*dg-small-cs-intros*] = *is-tiny-dghmD*(2,3)

Size.

lemma (**in** *is-tiny-dghm*) *tiny-dghm-ObjMap-in-Vset*[*dg-small-cs-intros*]:
  $\mathfrak{F}(\lfloor \mathit{ObjMap} \rfloor) \in_\circ \mathit{Vset}\ \alpha$
proof–
  have $\mathcal{D}_\circ\ (\mathfrak{F}(\lfloor \mathit{ObjMap} \rfloor)) \in_\circ \mathit{Vset}\ \alpha$
    by (*simp add*: *dghm-ObjMap-vdomain HomDom.tiny-dg-Obj-in-Vset*)
  moreover from *dghm-ObjMap-vrange* **have** $\mathcal{R}_\circ\ (\mathfrak{F}(\lfloor \mathit{ObjMap} \rfloor)) \in_\circ \mathit{Vset}\ \alpha$
    by (*simp add*: *vsubset-in-VsetI HomCod.tiny-dg-Obj-in-Vset*)
  ultimately show $\mathfrak{F}(\lfloor \mathit{ObjMap} \rfloor) \in_\circ \mathit{Vset}\ \alpha$
    by
      (
        cs-concl **cs-shallow cs-intro**:
          V-cs-intros dg-small-cs-intros ObjMap.vbrelation-Limit-in-VsetI
)
qed

lemmas [*dg-small-cs-intros*] = *is-tiny-dghm.tiny-dghm-ObjMap-in-Vset*

lemma (**in** *is-tiny-dghm*) *tiny-dghm-ArrMap-in-Vset*[*dg-small-cs-intros*]:
  $\mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor) \in_\circ \mathit{Vset}\ \alpha$
proof–
  have $\mathcal{D}_\circ\ (\mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor)) \in_\circ \mathit{Vset}\ \alpha$
    by (*simp add*: *dghm-ArrMap-vdomain HomDom.tiny-dg-Arr-in-Vset*)
  moreover from *HomCod.tiny-dg-Arr-in-Vset dghm-ArrMap-vrange* **have**
    $\mathcal{R}_\circ\ (\mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor)) \in_\circ \mathit{Vset}\ \alpha$
    by *auto*
  ultimately show $\mathfrak{F}(\lfloor \mathit{ArrMap} \rfloor) \in_\circ \mathit{Vset}\ \alpha$
    by
      (
        cs-concl **cs-shallow cs-intro**:
          V-cs-intros dg-small-cs-intros ArrMap.vbrelation-Limit-in-VsetI

```
    )
qed

lemmas [dg-small-cs-intros] = is-tiny-dghm.tiny-dghm-ArrMap-in-Vset

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

sublocale is-tiny-dghm ⊆ is-tm-dghm
  by (intro is-tm-dghmI') (auto simp: dg-cs-intros dg-small-cs-intros)

lemmas (in is-tiny-dghm) tiny-dghm-is-tm-dghm = is-tm-dghm-axioms

lemmas [dg-small-cs-intros] = is-tiny-dghm.tiny-dghm-is-tm-dghm

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

lemma tiny-dghms-vsubset-Vset[simp]:
  set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{DG.tiny α} 𝔅} ⊆∘ Vset α
proof(rule vsubsetI)
  fix 𝔉 assume 𝔉 ∈∘ all-tiny-dghms α
  then obtain 𝔄 𝔅 where 𝔉: 𝔉 : 𝔄 ↦↦_{DG.tiny α} 𝔅 by clarsimp
  then show 𝔉 ∈∘ Vset α by (auto simp: is-tiny-dghm.tiny-dghm-in-Vset)
qed

lemma (in is-dghm) dghm-is-tiny-dghm-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔉 : 𝔄 ↦↦_{DG.tiny β} 𝔅
proof(intro is-tiny-dghmI)
  interpret β: 𝒵 β by (rule assms(1))
```

```
  show 𝔉 : 𝔄 ↦↦_{DG β} 𝔅
    by (intro dghm-is-dghm-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: dg-cs-intros›)+
  show tiny-digraph β 𝔄 tiny-digraph β 𝔅
    by
      (
        simp-all add:
          assms
          HomDom.dg-tiny-digraph-if-ge-Limit
          HomCod.dg-tiny-digraph-if-ge-Limit
      )
qed
```

Opposite tiny digraph homomorphism

```
lemma (in is-tiny-dghm) is-tiny-dghm-op:
  op-dghm 𝔉 : op-dg 𝔄 ↦↦_{DG.tiny α} op-dg 𝔅
  by (intro is-tiny-dghmI)
    (cs-concl cs-shallow cs-intro: dg-small-cs-intros dg-cs-intros dg-op-intros)+

lemma (in is-tiny-dghm) is-tiny-dghm-op'[dg-op-intros]:
  assumes 𝔄' = op-dg 𝔄 and 𝔅' = op-dg 𝔅 and α' = α
  shows op-dghm 𝔉 : 𝔄' ↦↦_{DG.tiny_α'} 𝔅'
  unfolding assms by (rule is-tiny-dghm-op)

lemmas is-tiny-dghm-op[dg-op-intros] = is-tiny-dghm.is-tiny-dghm-op'
```

Composition of tiny digraph homomorphisms

```
lemma dghm-comp-is-tiny-dghm[dg-small-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦_{DG.tiny α} ℭ and 𝔉 : 𝔄 ↦↦_{DG.tiny α} 𝔅
  shows 𝔊 ∘_{DGHM} 𝔉 : 𝔄 ↦↦_{DG.tiny α} ℭ
proof–
  interpret 𝔉: is-tiny-dghm α 𝔄 𝔅 𝔉 by (rule assms(2))
  interpret 𝔊: is-tiny-dghm α 𝔅 ℭ 𝔊 by (rule assms(1))
  show ?thesis
    by (intro is-tiny-dghmI)
      (auto simp: dg-small-cs-intros dg-cs-simps intro: dg-cs-intros)
qed
```

Tiny constant digraph homomorphism

```
lemma dghm-const-is-tiny-dghm:
  assumes tiny-digraph α ℭ and tiny-digraph α 𝔇 and f : a ↦_𝔇 a
  shows dghm-const ℭ 𝔇 a f : ℭ ↦↦_{DG.tiny α} 𝔇
proof(intro is-tiny-dghmI)
  from assms show dghm-const ℭ 𝔇 a f : ℭ ↦↦_{DG α} 𝔇
    by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros dg-small-cs-intros)
qed (auto simp: assms(1,2))

lemma dghm-const-is-tiny-dghm'[dg-small-cs-intros]:
  assumes tiny-digraph α ℭ
    and tiny-digraph α 𝔇
    and f : a ↦_𝔇 a
    and ℭ' = ℭ
    and 𝔇' = 𝔇
  shows dghm-const ℭ 𝔇 a f : ℭ' ↦↦_{DG.tiny α} 𝔇'
  using assms(1–3) unfolding assms(4,5) by (rule dghm-const-is-tiny-dghm)
```

3.6 Transformation of digraph homomorphisms

3.6.1 Background

named-theorems *tdghm-cs-simps*
named-theorems *tdghm-cs-intros*
named-theorems *nt-field-simps*

definition $NTMap :: V$ **where** [*nt-field-simps*]: $NTMap = 0$
definition $NTDom :: V$ **where** [*nt-field-simps*]: $NTDom = 1_{\mathbb{N}}$
definition $NTCod :: V$ **where** [*nt-field-simps*]: $NTCod = 2_{\mathbb{N}}$
definition $NTDGDom :: V$ **where** [*nt-field-simps*]: $NTDGDom = 3_{\mathbb{N}}$
definition $NTDGCod :: V$ **where** [*nt-field-simps*]: $NTDGCod = 4_{\mathbb{N}}$

3.6.2 Definition and elementary properties

A transformation of digraph homomorphisms, as presented in this work, is a generalization of the concept of a natural transformation, as presented in Chapter I-4 in [39], to digraphs and digraph homomorphisms. The generalization is performed by excluding the commutativity axiom from the definition.

The definition of a transformation of digraph homomorphisms is parameterized by a limit ordinal α such that $\omega < \alpha$. Such transformations of digraph homomorphisms are referred to either as α-transformations of digraph homomorphisms or transformations of α-digraph homomorphisms.

locale *is-tdghm* =
  $\mathcal{Z}\ \alpha$ +
  vfsequence $\mathfrak{N}$ +
  NTDom: *is-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ +
  NTCod: *is-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$
  for $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ +
  assumes *tdghm-length*[*dg-cs-simps*]: *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
    and *tdghm-NTMap-vsv*: *vsv* $(\mathfrak{N}(\!|NTMap|\!))$
    and *tdghm-NTMap-vdomain*[*dg-cs-simps*]: $\mathcal{D}_{\circ}\ (\mathfrak{N}(\!|NTMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
    and *tdghm-NTDom*[*dg-cs-simps*]: $\mathfrak{N}(\!|NTDom|\!) = \mathfrak{F}$
    and *tdghm-NTCod*[*dg-cs-simps*]: $\mathfrak{N}(\!|NTCod|\!) = \mathfrak{G}$
    and *tdghm-NTDGDom*[*dg-cs-simps*]: $\mathfrak{N}(\!|NTDGDom|\!) = \mathfrak{A}$
    and *tdghm-NTDGCod*[*dg-cs-simps*]: $\mathfrak{N}(\!|NTDGCod|\!) = \mathfrak{B}$
    and *tdghm-NTMap-is-arr*:
      $a \in_{\circ} \mathfrak{A}(\!|Obj|\!) \Longrightarrow \mathfrak{N}(\!|NTMap|\!)(\!|a|\!) : \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{B}} \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$

syntax *-is-tdghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} \mapsto_{DGHM} \text{-} :/ \text{-} \mapsto\mapsto_{DG^1} \text{-})\rangle\ [51, 51, 51, 51, 51]\ 51)$
syntax-consts *-is-tdghm* $\rightleftharpoons$ *is-tdghm*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B} \rightleftharpoons$
  CONST is-tdghm $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$

abbreviation *all-tdghms* :: $V \Rightarrow V$
  where *all-tdghms* $\alpha \equiv$ *set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}\}$

abbreviation *tdghms* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *tdghms* $\alpha\ \mathfrak{A}\ \mathfrak{B} \equiv$ *set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}\}$

abbreviation *these-tdghms* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *these-tdghms* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G} \equiv$ *set* $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}\}$

sublocale *is-tdghm* $\subseteq$ *NTMap*: *vsv* $\langle\mathfrak{N}(\!|NTMap|\!)\rangle$
  rewrites $\mathcal{D}_{\circ}\ (\mathfrak{N}(\!|NTMap|\!)) = \mathfrak{A}(\!|Obj|\!)$
  by (*rule tdghm-NTMap-vsv*) (*simp add*: *dg-cs-simps*)

lemmas [*dg-cs-simps*] =
is-tdghm.tdghm-length
is-tdghm.tdghm-NTMap-vdomain
is-tdghm.tdghm-NTDom
is-tdghm.tdghm-NTCod
is-tdghm.tdghm-NTDGDom
is-tdghm.tdghm-NTDGCod

lemma (**in** *is-tdghm*) *tdghm-NTMap-is-arr'*[*dg-cs-intros*]:
assumes $a \in_{\circ} \mathfrak{A}(\!|Obj|\!)$
and $A = \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
and $B = \mathfrak{G}(\!|ObjMap|\!)(\!|a|\!)$
shows $\mathfrak{N}(\!|NTMap|\!)(\!|a|\!) : A \mapsto_{\mathfrak{B}} B$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule tdghm-NTMap-is-arr*)

lemmas [*dg-cs-intros*] = *is-tdghm.tdghm-NTMap-is-arr'*

Rules.

lemma (**in** *is-tdghm*) *is-tdghm-axioms'*[*dg-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 $\mathfrak{N} : \mathfrak{F}' \mapsto_{DGHM} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{DG_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-tdghm-axioms*)

mk-ide rf *is-tdghm-def*[*unfolded is-tdghm-axioms-def*]
|*intro is-tdghmI*|
|*dest is-tdghmD*[*dest*]|
|*elim is-tdghmE*[*elim*]|

lemmas [*dg-cs-intros*] =
is-tdghmD(3,4)

Elementary properties.

lemma *tdghm-eqI*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
and $\mathfrak{N}' : \mathfrak{F}' \mapsto_{DGHM} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{DG\alpha} \mathfrak{B}'$
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-tdghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret *R*: *is-tdghm* α $\mathfrak{A}'$ $\mathfrak{B}'$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{N}'$ **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**: *dg-cs-simps V-cs-simps*)
show $\mathcal{D}_{\circ}\ \mathfrak{N} = \mathcal{D}_{\circ}\ \mathfrak{N}'$
by (*cs-concl* **cs-shallow cs-simp**: *dg-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*: *dg-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 simp*: *L.vsv-axioms R.vsv-axioms*)
qed

lemma (**in** *is-tdghm*) *tdghm-def*:
$\mathfrak{N}$ = [$\mathfrak{N}$(|*NTMap*|), $\mathfrak{N}$(|*NTDom*|), $\mathfrak{N}$(|*NTCod*|), $\mathfrak{N}$(|*NTDGDom*|), $\mathfrak{N}$(|*NTDGCod*|)]$_\circ$
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ$ $\mathfrak{N}$ = $5_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps V-cs-simps*)
have *dom-rhs*:
$\mathcal{D}_\circ$ [$\mathfrak{N}$(|*NTMap*|), $\mathfrak{N}$(|*NTDGDom*|), $\mathfrak{N}$(|*NTDGCod*|), $\mathfrak{N}$(|*NTDom*|), $\mathfrak{N}$(|*NTCod*|)]$_\circ$ = $5_{\mathbb{N}}$
by (*simp add*: *nat-omega-simps*)
then show
$\mathcal{D}_\circ$ $\mathfrak{N}$ = $\mathcal{D}_\circ$ [$\mathfrak{N}$(|*NTMap*|), $\mathfrak{N}$(|*NTDom*|), $\mathfrak{N}$(|*NTCod*|), $\mathfrak{N}$(|*NTDGDom*|), $\mathfrak{N}$(|*NTDGCod*|)]$_\circ$
unfolding *dom-lhs dom-rhs* **by** (*simp add*: *nat-omega-simps*)
show *a* $\in_\circ$ $\mathcal{D}_\circ$ $\mathfrak{N}$ $\Longrightarrow$
$\mathfrak{N}$(|*a*|) = [$\mathfrak{N}$(|*NTMap*|), $\mathfrak{N}$(|*NTDom*|), $\mathfrak{N}$(|*NTCod*|), $\mathfrak{N}$(|*NTDGDom*|), $\mathfrak{N}$(|*NTDGCod*|)]$_\circ$(|*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-tdghm*) *tdghm-NTMap-app-in-Arr*[*dg-cs-intros*]:
assumes *a* $\in_\circ$ $\mathfrak{A}$(|*Obj*|)
shows $\mathfrak{N}$(|*NTMap*|)(|*a*|) $\in_\circ$ $\mathfrak{B}$(|*Arr*|)
using *assms* **using** *tdghm-NTMap-is-arr* **by** *auto*

lemmas [*dg-cs-intros*] = *is-tdghm.tdghm-NTMap-app-in-Arr*

lemma (**in** *is-tdghm*) *tdghm-NTMap-vrange-vifunion*:
$\mathcal{R}_\circ$ ($\mathfrak{N}$(|*NTMap*|)) $\subseteq_\circ$ ($\bigcup_\circ a \in_\circ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|)). $\bigcup_\circ b \in_\circ \mathcal{R}_\circ$ ($\mathfrak{G}$(|*ObjMap*|)). *Hom* $\mathfrak{B}$ *a b*)
proof(*intro NTMap.vsv-vrange-vsubset*)
fix *x* **assume** *prems*: *x* $\in_\circ$ $\mathfrak{A}$(|*Obj*|)
note $\mathfrak{N}x$ = *tdghm-NTMap-is-arr*[*OF prems*]
from *prems* **show**
$\mathfrak{N}$(|*NTMap*|)(|*x*|) $\in_\circ$ ($\bigcup_\circ a \in_\circ \mathcal{R}_\circ$ ($\mathfrak{F}$(|*ObjMap*|)). $\bigcup_\circ b \in_\circ \mathcal{R}_\circ$ ($\mathfrak{G}$(|*ObjMap*|)). *Hom* $\mathfrak{B}$ *a b*)
by (*intro vifunionI*, *unfold in-Hom-iff*)
(
auto intro:
dg-cs-intros NTDom.ObjMap.vsv-vimageI2′ NTCod.ObjMap.vsv-vimageI2′
)
qed

lemma (**in** *is-tdghm*) *tdghm-NTMap-vrange*: $\mathcal{R}_\circ$ ($\mathfrak{N}$(|*NTMap*|)) $\subseteq_\circ$ $\mathfrak{B}$(|*Arr*|)
proof(*intro NTMap.vsv-vrange-vsubset*)
fix *x* **assume** *x* $\in_\circ$ $\mathfrak{A}$(|*Obj*|)
with *is-tdghm-axioms* **show** $\mathfrak{N}$(|*NTMap*|)(|*x*|) $\in_\circ$ $\mathfrak{B}$(|*Arr*|)
by (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
qed

Size.

lemma (**in** *is-tdghm*) *tdghm-NTMap-vsubset-Vset*: $\mathfrak{N}$(|*NTMap*|) $\subseteq_\circ$ *Vset* α
proof(*intro NTMap.vbrelation-Limit-vsubset-VsetI*)
show $\mathcal{R}_\circ$ ($\mathfrak{N}$(|*NTMap*|)) $\subseteq_\circ$ *Vset* α
by
(
rule vsubset-transitive,
rule tdghm-NTMap-vrange,
rule NTDom.HomCod.dg-Arr-vsubset-Vset

```
    )
qed (simp-all add: NTDom.HomDom.dg-Obj-vsubset-Vset)

lemma (in is-tdghm) tdghm-NTMap-in-Vset:
  assumes α ∈∘ β
  shows ℜ(|NTMap|) ∈∘ Vset β
  by (meson assms tdghm-NTMap-vsubset-Vset Vset-in-mono vsubset-in-VsetI)

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

lemma (in is-tdghm) tdghm-is-tdghm-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows ℜ : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGβ 𝔅
proof(rule is-tdghmI)
  show ℜ(|NTMap|)(|a|) : 𝔉(|ObjMap|)(|a|) ↦_𝔅 𝔊(|ObjMap|)(|a|) if a ∈∘ 𝔄(|Obj|) for a
    using that by (cs-concl cs-shallow cs-intro: dg-cs-intros)
qed
  (
    cs-concl cs-shallow
      cs-simp: dg-cs-simps
      cs-intro:
        V-cs-intros
        assms NTDom.dghm-is-dghm-if-ge-Limit NTCod.dghm-is-dghm-if-ge-Limit
   )+

lemma small-all-tdghms[simp]:
  small {ℜ. ∃𝔉 𝔊 𝔄 𝔅. ℜ : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-tdghm.tdghm-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 {ℜ. ∃𝔉 𝔊 𝔄 𝔅. ℜ : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅} = {} by auto
  then show ?thesis by simp
qed

lemma small-tdghms[simp]: small {ℜ. ∃𝔉 𝔊. ℜ : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅}
  by (rule down[of - ‹set {ℜ. ∃𝔉 𝔊 𝔄 𝔅. ℜ : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅}›])
    auto
```

lemma *small-these-tdghms*[*simp*]: *small* {𝔑. 𝔑 : 𝔉 ↦$_{DGHM}$ 𝔊 : 𝔄 ↦↦$_{DG\alpha}$ 𝔅}
by (*rule down*[*of* - ‹*set* {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦$_{DGHM}$ 𝔊 : 𝔄 ↦↦$_{DG\alpha}$ 𝔅}›])
auto

Further elementary results.

lemma *these-tdghms-iff*:
𝔑 ∈$_\circ$ *these-tdghms* α 𝔄 𝔅 𝔉 𝔊 ⟷ 𝔑 : 𝔉 ↦$_{DGHM}$ 𝔊 : 𝔄 ↦↦$_{DG\alpha}$ 𝔅
by *auto*

3.6.3 Opposite transformation of digraph homomorphisms

Definition and elementary properties

See section 1.5 in [15].

definition *op-tdghm* :: *V* ⇒ *V*
where *op-tdghm* 𝔑 =
[
𝔑(|*NTMap*|),
op-dghm (𝔑(|*NTCod*|)),
op-dghm (𝔑(|*NTDom*|)),
op-dg (𝔑(|*NTDGDom*|)),
op-dg (𝔑(|*NTDGCod*|))
]$_\circ$

Components.

lemma *op-tdghm-components*[*dg-op-simps*]:
shows *op-tdghm* 𝔑(|*NTMap*|) = 𝔑(|*NTMap*|)
and *op-tdghm* 𝔑(|*NTDom*|) = *op-dghm* (𝔑(|*NTCod*|))
and *op-tdghm* 𝔑(|*NTCod*|) = *op-dghm* (𝔑(|*NTDom*|))
and *op-tdghm* 𝔑(|*NTDGDom*|) = *op-dg* (𝔑(|*NTDGDom*|))
and *op-tdghm* 𝔑(|*NTDGCod*|) = *op-dg* (𝔑(|*NTDGCod*|))
unfolding *op-tdghm-def nt-field-simps* **by** (*auto simp*: *nat-omega-simps*)

Further properties

lemma (**in** *is-tdghm*) *is-tdghm-op*:
op-tdghm 𝔑 : *op-dghm* 𝔊 ↦$_{DGHM}$ *op-dghm* 𝔉 : *op-dg* 𝔄 ↦↦$_{DG\alpha}$ *op-dg* 𝔅
proof(*rule is-tdghmI*, *unfold dg-op-simps*)
show *vfsequence* (*op-tdghm* 𝔑) **by** (*simp add*: *op-tdghm-def*)
show *vcard* (*op-tdghm* 𝔑) = 5$_\mathbb{N}$ **by** (*simp add*: *op-tdghm-def nat-omega-simps*)
show 𝔑(|*NTMap*|)(|*a*|) : 𝔉(|*ObjMap*|)(|*a*|) ↦$_{\mathfrak{B}}$ 𝔊(|*ObjMap*|)(|*a*|) **if** *a* ∈$_\circ$ 𝔄(|*Obj*|) **for** *a*
using *that* **by** (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
qed
(
cs-concl **cs-shallow**
cs-simp: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-op-intros V-cs-intros*
)+

lemma (**in** *is-tdghm*) *is-tdghm-op'*[*dg-op-intros*]:
assumes 𝔊′ = *op-dghm* 𝔊
and 𝔉′ = *op-dghm* 𝔉
and 𝔄′ = *op-dg* 𝔄
and 𝔅′ = *op-dg* 𝔅
shows *op-tdghm* 𝔑 : 𝔊′ ↦$_{DGHM}$ 𝔉′ : 𝔄′ ↦↦$_{DG\alpha}$ 𝔅′
unfolding *assms* **by** (*rule is-tdghm-op*)

lemmas *is-tdghm-op*[*dg-op-intros*] = *is-tdghm.is-tdghm-op'*

lemma (**in** *is-tdghm*) *tdghm-op-tdghm-op-tdghm*[*dg-op-simps*]:
op-tdghm (*op-tdghm* $\mathfrak{N}$) = $\mathfrak{N}$
proof(*rule tdghm-eqI*[*of* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ - $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$], *unfold dg-op-simps*)
interpret *op*:
is-tdghm α ‹*op-dg* $\mathfrak{A}$› ‹*op-dg* $\mathfrak{B}$› ‹*op-dghm* $\mathfrak{G}$› ‹*op-dghm* $\mathfrak{F}$› ‹*op-tdghm* $\mathfrak{N}$›
by (*rule is-tdghm-op*)
from *op.is-tdghm-op* **show**
op-tdghm (*op-tdghm* $\mathfrak{N}$) : $\mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
by (*simp add*: *dg-op-simps*)
qed (*auto simp*: *dg-cs-intros*)

lemmas *tdghm-op-tdghm-op-tdghm*[*dg-op-simps*] =
is-tdghm.tdghm-op-tdghm-op-tdghm

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

3.6.4 Composition of a transformation of digraph homomorphisms and a digraph homomorphism

Definition and elementary properties

definition *tdghm-dghm-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\circ_{TDGHM-DGHM}$› 55)
where $\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H}$ =
[
$(\lambda a \in_\circ \mathfrak{H}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ \mathfrak{N}(\lfloor NTMap \rfloor)(\mathfrak{H}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)))$,
$\mathfrak{N}(\lfloor NTDom \rfloor) \circ_{DGHM} \mathfrak{H}$,
$\mathfrak{N}(\lfloor NTCod \rfloor) \circ_{DGHM} \mathfrak{H}$,
$\mathfrak{H}(\lfloor HomDom \rfloor)$,
$\mathfrak{N}(\lfloor NTDGCod \rfloor)$
]$_\circ$

Components.

lemma *tdghm-dghm-comp-components*:
shows $(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTMap \rfloor)$ =
$(\lambda a \in_\circ \mathfrak{H}(\lfloor HomDom \rfloor)(\lfloor Obj \rfloor).\ \mathfrak{N}(\lfloor NTMap \rfloor)(\mathfrak{H}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)))$

and [*dg-shared-cs-simps*, *dg-cs-simps*]:
$(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTDom \rfloor) = \mathfrak{N}(\lfloor NTDom \rfloor) \circ_{DGHM} \mathfrak{H}$
and [*dg-shared-cs-simps*, *dg-cs-simps*]:
$(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTCod \rfloor) = \mathfrak{N}(\lfloor NTCod \rfloor) \circ_{DGHM} \mathfrak{H}$
and [*dg-shared-cs-simps*, *dg-cs-simps*]:
$(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTDGDom \rfloor) = \mathfrak{H}(\lfloor HomDom \rfloor)$
and [*dg-shared-cs-simps*, *dg-cs-simps*]:
$(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTDGCod \rfloor) = \mathfrak{N}(\lfloor NTDGCod \rfloor)$
unfolding *tdghm-dghm-comp-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Transformation map

mk-VLambda *tdghm-dghm-comp-components*(1)
|*vsv tdghm-dghm-comp-NTMap-vsv*[*dg-shared-cs-intros*, *dg-cs-intros*]|

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

lemmas [*dg-cs-simps*] =
is-dghm.tdghm-dghm-comp-NTMap-vdomain
is-dghm.tdghm-dghm-comp-NTMap-app

lemma *tdghm-dghm-comp-NTMap-vrange*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ\ ((\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof–
interpret $\mathfrak{N}$: *is-tdghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{H}$: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{H}$ **by** (*rule assms*(2))
show *?thesis*
unfolding *tdghm-dghm-comp-components*
proof(*rule vrange-VLambda-vsubset*, *unfold dg-cs-simps*)
fix x **assume** $x \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
then show $\mathfrak{N}(\lfloor NTMap \rfloor)(\mathfrak{H}(\lfloor ObjMap \rfloor)(\lfloor x \rfloor)) \in_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
by (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
qed
qed

Opposite of the composition of a transformation of digraph homomorphisms and a digraph homomorphism

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

Composition of a transformation of digraph homomorphisms and a digraph homomorphism is a transformation of digraph homomorphisms

lemma *tdghm-dghm-comp-is-tdghm*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H} : \mathfrak{F} \circ_{DGHM} \mathfrak{H} \mapsto_{DGHM} \mathfrak{G} \circ_{DGHM} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-tdghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{H}$: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{H}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule is-tdghmI*)
show *vfsequence* $(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})$ **unfolding** *tdghm-dghm-comp-def* **by** *simp*
show *vcard* $(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H}) = 5_{\mathbb{N}}$
unfolding *tdghm-dghm-comp-def* **by** (*simp add*: *nat-omega-simps*)
show $(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H})(\lfloor NTMap \rfloor)(\lfloor a \rfloor) :$
$(\mathfrak{F} \circ_{DGHM} \mathfrak{H})(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \mapsto_{\mathfrak{C}} (\mathfrak{G} \circ_{DGHM} \mathfrak{H})(\lfloor ObjMap \rfloor)(\lfloor a \rfloor)$
if $a \in_{\circ} \mathfrak{A}(\lfloor Obj \rfloor)$ **for** a
by
(
use that **in**
‹*cs-concl cs-shallow cs-simp*: *dg-cs-simps cs-intro*: *dg-cs-intros*›
)
qed (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)+
qed

lemma *tdghm-dghm-comp-is-tdghm'*[*dg-cs-intros*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
and $\mathfrak{F}' = \mathfrak{F} \circ_{DGHM} \mathfrak{H}$
and $\mathfrak{G}' = \mathfrak{G} \circ_{DGHM} \mathfrak{H}$
shows $\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{H} : \mathfrak{F}' \mapsto_{DGHM} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$
using *assms*(1,2) **unfolding** *assms*(3,4) **by** (*rule tdghm-dghm-comp-is-tdghm*)

Further properties

lemma *tdghm-dghm-comp-tdghm-dghm-comp-assoc*:
assumes $\mathfrak{N} : \mathfrak{H} \mapsto_{DGHM} \mathfrak{H}' : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{D}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{G}) \circ_{TDGHM-DGHM} \mathfrak{F} = \mathfrak{N} \circ_{TDGHM-DGHM} (\mathfrak{G} \circ_{DGHM} \mathfrak{F})$
proof–
interpret $\mathfrak{N}$: *is-tdghm* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{H}$ $\mathfrak{H}'$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{G}$: *is-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(2))
interpret $\mathfrak{F}$: *is-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(3))
show *?thesis*
proof(*rule tdghm-eqI*)
from *assms* **show**
$(\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{G}) \circ_{TDGHM-DGHM} \mathfrak{F} :$
$\mathfrak{H} \circ_{DGHM} \mathfrak{G} \circ_{DGHM} \mathfrak{F} \mapsto_{DGHM} \mathfrak{H}' \circ_{DGHM} \mathfrak{G} \circ_{DGHM} \mathfrak{F} :$
$\mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{D}$
by (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
then have *dom-lhs*: $\mathcal{D}_{\circ}\ (((\mathfrak{N} \circ_{TDGHM-DGHM} \mathfrak{G}) \circ_{TDGHM-DGHM} \mathfrak{F})(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-simp**: *dg-cs-simps*)
show $\mathfrak{N} \circ_{TDGHM-DGHM} (\mathfrak{G} \circ_{DGHM} \mathfrak{F}) :$
$\mathfrak{H} \circ_{DGHM} \mathfrak{G} \circ_{DGHM} \mathfrak{F} \mapsto_{DGHM} \mathfrak{H}' \circ_{DGHM} \mathfrak{G} \circ_{DGHM} \mathfrak{F} :$
$\mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{D}$
by (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_{\circ}\ ((\mathfrak{N} \circ_{TDGHM-DGHM} (\mathfrak{G} \circ_{DGHM} \mathfrak{F}))(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
by (*cs-concl* **cs-simp**: *dg-cs-simps*)

```
    show
      ((𝔑 ∘_TDGHM-DGHM 𝔊) ∘_TDGHM-DGHM 𝔉)(⌊NTMap⌋) =
        (𝔑 ∘_TDGHM-DGHM (𝔊 ∘_DGHM 𝔉))(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈₀ 𝔄(⌊Obj⌋)
      with assms show
        ((𝔑 ∘_TDGHM-DGHM 𝔊) ∘_TDGHM-DGHM 𝔉)(⌊NTMap⌋)(⌊a⌋) =
          (𝔑 ∘_TDGHM-DGHM (𝔊 ∘_DGHM 𝔉))(⌊NTMap⌋)(⌊a⌋)
        by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
    qed (cs-concl cs-shallow cs-intro: dg-cs-intros)
  qed simp-all
qed

lemma (in is-tdghm) tdghm-tdghm-dghm-comp-dghm-id[dg-cs-simps]:
  𝔑 ∘_TDGHM-DGHM dghm-id 𝔄 = 𝔑
proof(rule tdghm-eqI)
  show 𝔑 ∘_TDGHM-DGHM dghm-id 𝔄 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅
    by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  show 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  have dom-lhs: 𝒟₀ ((𝔑 ∘_TDGHM-DGHM dghm-id 𝔄)(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
    by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  show (𝔑 ∘_TDGHM-DGHM dghm-id 𝔄)(⌊NTMap⌋) = 𝔑(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dg-cs-simps)
    fix a assume a ∈₀ 𝔄(⌊Obj⌋)
    then show (𝔑 ∘_TDGHM-DGHM dghm-id 𝔄)(⌊NTMap⌋)(⌊a⌋) = 𝔑(⌊NTMap⌋)(⌊a⌋)
      by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  qed (cs-concl cs-intro: dg-cs-intros V-cs-intros)+
qed simp-all

lemmas [dg-cs-simps] = is-tdghm.tdghm-tdghm-dghm-comp-dghm-id
```

3.6.5 Composition of a digraph homomorphism and a transformation of digraph homomorphisms

Definition and elementary properties

```
definition dghm-tdghm-comp :: V ⇒ V ⇒ V (infixl ‹∘_DGHM-TDGHM› 55)
  where ℌ ∘_DGHM-TDGHM 𝔑 =
    [
      (λa∈₀𝔑(⌊NTDGDom⌋)(⌊Obj⌋). ℌ(⌊ArrMap⌋)(𝔑(⌊NTMap⌋)(⌊a⌋))),
      ℌ ∘_DGHM 𝔑(⌊NTDom⌋),
      ℌ ∘_DGHM 𝔑(⌊NTCod⌋),
      𝔑(⌊NTDGDom⌋),
      ℌ(⌊HomCod⌋)
    ]₀
```

Components.

```
lemma dghm-tdghm-comp-components:
  shows (ℌ ∘_DGHM-TDGHM 𝔑)(⌊NTMap⌋) =
    (λa∈₀𝔑(⌊NTDGDom⌋)(⌊Obj⌋). ℌ(⌊ArrMap⌋)(𝔑(⌊NTMap⌋)(⌊a⌋)))
    and [dg-shared-cs-simps, dg-cs-simps]:
      (ℌ ∘_DGHM-TDGHM 𝔑)(⌊NTDom⌋) = ℌ ∘_DGHM 𝔑(⌊NTDom⌋)
    and [dg-shared-cs-simps, dg-cs-simps]:
      (ℌ ∘_DGHM-TDGHM 𝔑)(⌊NTCod⌋) = ℌ ∘_DGHM 𝔑(⌊NTCod⌋)
    and [dg-shared-cs-simps, dg-cs-simps]:
      (ℌ ∘_DGHM-TDGHM 𝔑)(⌊NTDGDom⌋) = 𝔑(⌊NTDGDom⌋)
    and [dg-shared-cs-simps, dg-cs-simps]:
```

$(\mathfrak{H} \circ_{DGHM-TDGHM} \mathfrak{N})(\lfloor NTDGCod \rfloor) = \mathfrak{H}(\lfloor HomCod \rfloor)$
unfolding *dghm-tdghm-comp-def nt-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Transformation map

mk-VLambda *dghm-tdghm-comp-components*(1)
|*vsv dghm-tdghm-comp-NTMap-vsv*[*dg-shared-cs-intros*, *dg-cs-intros*]|

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

lemmas [*dg-cs-simps*] =
is-tdghm.dghm-tdghm-comp-NTMap-vdomain
is-tdghm.dghm-tdghm-comp-NTMap-app

lemma *dghm-tdghm-comp-NTMap-vrange*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ\ ((\mathfrak{H} \circ_{DGHM-TDGHM} \mathfrak{N})(\lfloor NTMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof–
interpret $\mathfrak{H}$: *is-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-tdghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))
show *?thesis*
unfolding *dghm-tdghm-comp-components*
proof(*rule vrange-VLambda-vsubset*, *unfold dg-cs-simps*)
fix x **assume** $x \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
then show $\mathfrak{H}(\lfloor ArrMap \rfloor)(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor x \rfloor)) \in_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
by (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
qed
qed

Opposite of the composition of a digraph homomorphism and a transformation of digraph homomorphisms

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

Composition of a digraph homomorphism and a transformation of digraph homomorphisms is a transformation of digraph homomorphisms

lemma *dghm-tdghm-comp-is-tdghm*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathfrak{H} \circ_{DGHM-TDGHM} \mathfrak{N} : \mathfrak{H} \circ_{DGHM} \mathfrak{F} \mapsto_{DGHM} \mathfrak{H} \circ_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{H}$: *is-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-tdghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))

```
  show ?thesis
  proof(rule is-tdghmI)
    show vfsequence (ℌ ∘DGHM-TDGHM 𝔑)
      unfolding dghm-tdghm-comp-def by simp
    show vcard (ℌ ∘DGHM-TDGHM 𝔑) = 5ℕ
      unfolding dghm-tdghm-comp-def by (simp add: nat-omega-simps)
    show (ℌ ∘DGHM-TDGHM 𝔑)(|NTMap|)(|a|) :
      (ℌ ∘DGHM 𝔉)(|ObjMap|)(|a|) ↦ℭ (ℌ ∘DGHM 𝔊)(|ObjMap|)(|a|)
      if a ∈∘ 𝔄(|Obj|) for a
      by (use that in ‹cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros›)
  qed (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)+
qed

lemma dghm-tdghm-comp-is-tdghm'[dg-cs-intros]:
  assumes ℌ : 𝔅 ↦↦DGα ℭ
    and 𝔑 : 𝔉 ↦DGHM 𝔊 : 𝔄 ↦↦DGα 𝔅
    and 𝔉' = ℌ ∘DGHM 𝔉
    and 𝔊' = ℌ ∘DGHM 𝔊
  shows ℌ ∘DGHM-TDGHM 𝔑 : 𝔉' ↦DGHM 𝔊' : 𝔄 ↦↦DGα ℭ
  using assms(1,2) unfolding assms(3,4) by (rule dghm-tdghm-comp-is-tdghm)
```

Further properties

```
lemma dghm-comp-dghm-tdghm-comp-assoc:
  assumes 𝔑 : ℌ ↦DGHM ℌ' : 𝔄 ↦↦DGα 𝔅
    and 𝔉 : 𝔅 ↦↦DGα ℭ
    and 𝔊 : ℭ ↦↦DGα 𝔇
  shows (𝔊 ∘DGHM 𝔉) ∘DGHM-TDGHM 𝔑 = 𝔊 ∘DGHM-TDGHM (𝔉 ∘DGHM-TDGHM 𝔑)
proof(rule tdghm-eqI)
  interpret 𝔑: is-tdghm α 𝔄 𝔅 ℌ ℌ' 𝔑 by (rule assms(1))
  interpret 𝔉: is-dghm α 𝔅 ℭ 𝔉 by (rule assms(2))
  interpret 𝔊: is-dghm α ℭ 𝔇 𝔊 by (rule assms(3))
  from assms show (𝔊 ∘DGHM 𝔉) ∘DGHM-TDGHM 𝔑 :
    𝔊 ∘DGHM 𝔉 ∘DGHM ℌ ↦DGHM 𝔊 ∘DGHM 𝔉 ∘DGHM ℌ' : 𝔄 ↦↦DGα 𝔇
    by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  then have dom-lhs: 𝒟∘ ((𝔊 ∘DGHM 𝔉 ∘DGHM-TDGHM 𝔑)(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-simp: dg-cs-simps)
  from assms show 𝔊 ∘DGHM-TDGHM (𝔉 ∘DGHM-TDGHM 𝔑) :
    𝔊 ∘DGHM 𝔉 ∘DGHM ℌ ↦DGHM 𝔊 ∘DGHM 𝔉 ∘DGHM ℌ' : 𝔄 ↦↦DGα 𝔇
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  then have dom-rhs:
    𝒟∘ ((𝔊 ∘DGHM-TDGHM (𝔉 ∘DGHM-TDGHM 𝔑))(|NTMap|)) = 𝔄(|Obj|)
    by (cs-concl cs-simp: dg-cs-simps)
  show
    ((𝔊 ∘DGHM 𝔉) ∘DGHM-TDGHM 𝔑)(|NTMap|) =
      (𝔊 ∘DGHM-TDGHM (𝔉 ∘DGHM-TDGHM 𝔑))(|NTMap|)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume a ∈∘ 𝔄(|Obj|)
    then show
      (𝔊 ∘DGHM 𝔉 ∘DGHM-TDGHM 𝔑)(|NTMap|)(|a|) =
        (𝔊 ∘DGHM-TDGHM (𝔉 ∘DGHM-TDGHM 𝔑))(|NTMap|)(|a|)
      by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  qed (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)+
qed simp-all

lemma (in is-tdghm) tdghm-dghm-tdghm-comp-dghm-id[dg-cs-simps]:
  dghm-id 𝔅 ∘DGHM-TDGHM 𝔑 = 𝔑
proof(rule tdghm-eqI)
```

```
  show dghm-id 𝔅 ∘_DGHM-TDGHM 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅
    by (cs-concl cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  then have dom-lhs: 𝒟∘ ((dghm-id 𝔅 ∘_DGHM-TDGHM 𝔑)(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
    by (cs-concl cs-simp: dg-cs-simps)
  show 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  show (dghm-id 𝔅 ∘_DGHM-TDGHM 𝔑)(⌊NTMap⌋) = 𝔑(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dg-cs-simps)
    show vsv (𝔑(⌊NTMap⌋)) by auto
    fix a assume a ∈∘ 𝔄(⌊Obj⌋)
    then show (dghm-id 𝔅 ∘_DGHM-TDGHM 𝔑)(⌊NTMap⌋)(⌊a⌋) = 𝔑(⌊NTMap⌋)(⌊a⌋)
      by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
  qed (cs-concl cs-shallow cs-intro: dg-cs-intros)+
qed simp-all

lemmas [dg-cs-simps] = is-tdghm.tdghm-dghm-tdghm-comp-dghm-id

lemma dghm-tdghm-comp-tdghm-dghm-comp-assoc:
  assumes 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔅 ↦↦_DGα ℭ
    and ℌ : ℭ ↦↦_DGα 𝔇
    and 𝔎 : 𝔄 ↦↦_DGα 𝔅
  shows (ℌ ∘_DGHM-TDGHM 𝔑) ∘_TDGHM-DGHM 𝔎 = ℌ ∘_DGHM-TDGHM (𝔑 ∘_TDGHM-DGHM 𝔎)
proof–
  interpret 𝔑: is-tdghm α 𝔅 ℭ 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret ℌ: is-dghm α ℭ 𝔇 ℌ by (rule assms(2))
  interpret 𝔎: is-dghm α 𝔄 𝔅 𝔎 by (rule assms(3))
  show ?thesis
  proof(rule tdghm-eqI)
    from assms have dom-lhs:
      𝒟∘ (((ℌ ∘_DGHM-TDGHM 𝔑) ∘_TDGHM-DGHM 𝔎)(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
      by (cs-concl cs-shallow cs-simp: dg-cs-simps)
    from assms have dom-rhs:
      𝒟∘ ((ℌ ∘_DGHM-TDGHM (𝔑 ∘_TDGHM-DGHM 𝔎))(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
      by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
    show
      ((ℌ ∘_DGHM-TDGHM 𝔑) ∘_TDGHM-DGHM 𝔎)(⌊NTMap⌋) =
        (ℌ ∘_DGHM-TDGHM (𝔑 ∘_TDGHM-DGHM 𝔎))(⌊NTMap⌋)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
      fix a assume a ∈∘ 𝔄(⌊Obj⌋)
      then show
        ((ℌ ∘_DGHM-TDGHM 𝔑) ∘_TDGHM-DGHM 𝔎)(⌊NTMap⌋)(⌊a⌋) =
          ((ℌ ∘_DGHM-TDGHM (𝔑 ∘_TDGHM-DGHM 𝔎)))(⌊NTMap⌋)(⌊a⌋)
        by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
    qed (cs-concl cs-shallow cs-intro: dg-cs-intros)
  qed (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)+
qed
```

3.7 Smallness for transformations of digraph homomorphisms

3.7.1 Transformation of digraph homomorphisms with tiny maps

Definition and elementary properties

locale *is-tm-tdghm* =
  $\mathcal{Z}$ α +
  NTDom: *is-tm-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  NTCod: *is-tm-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ +
  is-tdghm 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-tdghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\mapsto_{DGHM.tm}$ - :/ - $\mapsto\mapsto_{DG.tm1}$ -)$\rangle$ [51, 51, 51, 51, 51] 51)
syntax-consts *-is-tm-tdghm* $\rightleftharpoons$ *is-tm-tdghm*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tm\alpha} \mathfrak{B} \rightleftharpoons$
  CONST is-tm-tdghm α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

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

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

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

Rules.

lemma (**in** *is-tm-tdghm*) *is-tm-tdghm-axioms'*[*dg-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 $\mathfrak{N} : \mathfrak{F}' \mapsto_{DGHM.tm} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{DG.tm_{\alpha'}} \mathfrak{B}'$
  unfolding *assms* **by** (*rule is-tm-tdghm-axioms*)

mk-ide rf *is-tm-tdghm-def*
  |*intro is-tm-tdghmI*|
  |*dest is-tm-tdghmD*[*dest*]|
  |*elim is-tm-tdghmE*[*elim*]|

lemmas [*dg-small-cs-intros*] = *is-tm-tdghmD*(2,3,4)

Size.

lemma (**in** *is-tm-tdghm*) *tm-tdghm-NTMap-in-Vset*: $\mathfrak{N}(\!|NTMap|\!) \in_\circ Vset\ \alpha$
proof–
  show *?thesis*
  proof(*rule vbrelation.vbrelation-Limit-in-VsetI*)
    have $(\bigcup_\circ a \in_\circ \mathcal{R}_\circ\ (\mathfrak{F}(\!|ObjMap|\!)).\ \bigcup_\circ b \in_\circ \mathcal{R}_\circ\ (\mathfrak{G}(\!|ObjMap|\!)).\ Hom\ \mathfrak{B}\ a\ b) \in_\circ Vset\ \alpha$
      by
        (
          intro
            NTDom.HomCod.dg-Hom-vifunion-in-Vset
            NTDom.dghm-ObjMap-vrange
            NTDom.tm-dghm-ObjMap-in-Vset
            NTCod.dghm-ObjMap-vrange
            NTCod.tm-dghm-ObjMap-in-Vset
            vrange-in-VsetI

```
      )+
    moreover have
      ℛ∘ (𝔑(‖NTMap‖)) ⊆∘ (⋃∘a∈∘ℛ∘ (𝔉(‖ObjMap‖)). ⋃∘b∈∘ℛ∘ (𝔊(‖ObjMap‖)). Hom 𝔅 a b)
      by (rule tdghm-NTMap-vrange-vifunion)
    ultimately show ℛ∘ (𝔑(‖NTMap‖)) ∈∘ Vset α by (auto simp: dg-cs-simps)
  qed
    (
      insert NTCod.tm-dghm-HomDom-is-tiny-digraph,
      auto intro!: NTMap.vbrelation-axioms simp: dg-cs-simps
    )
qed

lemma small-all-tm-tdghms[simp]:
  small {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_DGHM.tm 𝔊 : 𝔄 ↦↦_DG.tmα 𝔅}
proof(rule down)
  show {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_DGHM.tm 𝔊 : 𝔄 ↦↦_DG.tmα 𝔅} ⊆
    elts (set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅})
  proof
    (
      simp only: elts-of-set small-all-tdghms if-True,
      rule subsetI,
      unfold mem-Collect-eq
    )
    fix 𝔑 assume ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_DGHM.tm 𝔊 : 𝔄 ↦↦_DG.tmα 𝔅
    then obtain 𝔉 𝔊 𝔄 𝔅 where 𝔑: 𝔑 : 𝔉 ↦_DGHM.tm 𝔊 : 𝔄 ↦↦_DG.tmα 𝔅
      by clarsimp
    interpret is-tm-tdghm α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule 𝔑)
    have 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅 by (auto intro: dg-cs-intros)
    then show ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_DGHM 𝔊 : 𝔄 ↦↦_DGα 𝔅 by auto
  qed
qed

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

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

Further elementary results.

```
lemma these-tm-tdghms-iff:
  𝔑 ∈∘ these-tm-tdghms α 𝔄 𝔅 𝔉 𝔊 ⟷
    𝔑 : 𝔉 ↦_DGHM.tm 𝔊 : 𝔄 ↦↦_DG.tmα 𝔅
```

by *auto*

Opposite transformation of digraph homomorphisms with tiny maps

lemma (in *is-tm-tdghm*) *is-tm-tdghm-op*:
op-tdghm $\mathfrak{N}$: *op-dghm* $\mathfrak{G}$ $\mapsto_{DGHM.tm}$ *op-dghm* $\mathfrak{F}$: *op-dg* $\mathfrak{A}$ $\mapsto\mapsto_{DG.tm\alpha}$ *op-dg* $\mathfrak{B}$
by (*intro is-tm-tdghmI*)
(*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros dg-op-intros*)+

lemma (in *is-tm-tdghm*) *is-tm-tdghm-op'*[*dg-op-intros*]:
assumes $\mathfrak{G}'$ = *op-dghm* $\mathfrak{G}$
and $\mathfrak{F}'$ = *op-dghm* $\mathfrak{F}$
and $\mathfrak{A}'$ = *op-dg* $\mathfrak{A}$
and $\mathfrak{B}'$ = *op-dg* $\mathfrak{B}$
shows *op-tdghm* $\mathfrak{N}$: $\mathfrak{G}'$ $\mapsto_{DGHM.tm}$ $\mathfrak{F}'$: $\mathfrak{A}'$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule is-tm-tdghm-op*)

lemmas *is-tm-tdghm-op*[*dg-op-intros*] = *is-tm-tdghm.is-tm-tdghm-op'*

Composition of a transformation of digraph homomorphisms with tiny maps and a digraph homomorphism with tiny maps

lemma *tdghm-dghm-comp-is-tm-tdghm*:
assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{DGHM.tm}$ $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{H}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{B}$
shows $\mathfrak{N}$ $\circ_{TDGHM-DGHM}$ $\mathfrak{H}$: $\mathfrak{F}$ $\circ_{DGHM}$ $\mathfrak{H}$ $\mapsto_{DGHM.tm}$ $\mathfrak{G}$ $\circ_{DGHM}$ $\mathfrak{H}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-tm-tdghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{H}$: *is-tm-dghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{H}$ **by** (*rule assms*(2))
show *?thesis*
by (*rule is-tm-tdghmI*)
(
cs-concl
cs-simp: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-small-cs-intros*
)+
qed

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

Composition of a digraph homomorphism with tiny maps and a transformation of digraph homomorphisms with tiny maps

lemma *dghm-tdghm-comp-is-tm-tdghm*:
assumes $\mathfrak{H}$: $\mathfrak{B}$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{DGHM.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{B}$
shows $\mathfrak{H}$ $\circ_{DGHM-TDGHM}$ $\mathfrak{N}$: $\mathfrak{H}$ $\circ_{DGHM}$ $\mathfrak{F}$ $\mapsto_{DGHM.tm}$ $\mathfrak{H}$ $\circ_{DGHM}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\mapsto_{DG.tm\alpha}$ $\mathfrak{C}$
proof–
interpret $\mathfrak{H}$: *is-tm-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-tm-tdghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))
show *?thesis*
by (*rule is-tm-tdghmI*)
(
cs-concl
cs-simp: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-small-cs-intros*

)+
qed

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

3.7.2 Transformation of homomorphisms of tiny digraphs

Definition and elementary properties

locale *is-tiny-tdghm* =
$\mathcal{Z}\ \alpha$ +
NTDom: *is-tiny-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ +
NTCod: *is-tiny-dghm* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{G}$ +
is-tdghm $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$
for $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$

syntax *-is-tiny-tdghm* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $\mapsto_{DGHM.tiny}$ - :/ - $\mapsto\mapsto_{DG.tiny1}$ -)$\rangle$ [51, 51, 51, 51, 51] 51)
syntax-consts *-is-tiny-tdghm* $\rightleftharpoons$ *is-tiny-tdghm*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B} \rightleftharpoons$
CONST is-tiny-tdghm $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$

abbreviation *all-tiny-tdghms* :: $V \Rightarrow V$
where *all-tiny-tdghms* $\alpha \equiv$
set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$

abbreviation *tiny-tdghms* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *tiny-tdghms* $\alpha\ \mathfrak{A}\ \mathfrak{B} \equiv$
set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$

abbreviation *these-tiny-tdghms* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *these-tiny-tdghms* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G} \equiv$
set $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$

Rules.

lemmas (**in** *is-tiny-tdghm*) [*dg-small-cs-intros*] = *is-tiny-tdghm-axioms*

mk-ide rf *is-tiny-tdghm-def*
|*intro is-tiny-tdghmI*[*intro*]|
|*dest is-tiny-tdghmD*[*dest*]|
|*elim is-tiny-tdghmE*[*elim*]|

lemmas [*dg-small-cs-intros*] = *is-tiny-tdghmD*(2,3,4)

Elementary properties.

sublocale *is-tiny-tdghm* $\subseteq$ *is-tm-tdghm*
by (*rule is-tm-tdghmI*)
(*auto simp*: *vfsequence-axioms dg-cs-intros dg-small-cs-intros*)

lemmas (**in** *is-tiny-tdghm*) *tiny-tdghm-is-tm-tdghm* = *is-tm-tdghm-axioms*

lemmas [*dg-small-cs-intros*] = *is-tiny-tdghm.tiny-tdghm-is-tm-tdghm*

Size.

lemma (**in** *is-tiny-tdghm*) *tiny-tdghm-in-Vset*: $\mathfrak{N} \in_{\circ} \mathit{Vset}\ \alpha$
proof–
note [*dg-cs-intros*] =
tm-tdghm-NTMap-in-Vset
NTDom.tiny-dghm-in-Vset
NTCod.tiny-dghm-in-Vset
NTDom.HomDom.tiny-dg-in-Vset
NTDom.HomCod.tiny-dg-in-Vset
show *?thesis*
by (*subst tdghm-def*)
(
cs-concl **cs-shallow**
cs-simp: *dg-cs-simps* **cs-intro**: *dg-cs-intros V-cs-intros*
)
qed

lemma *small-all-tiny-tdghms*[*simp*]:
small $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$
proof(*rule down*)
show $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\} \subseteq$
elts (*set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}\}$)
proof
(
simp only: *elts-of-set small-all-tdghms if-True*,
rule subsetI,
unfold mem-Collect-eq
)
fix $\mathfrak{N}$ **assume** $\exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}$
then obtain $\mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}$ **where** $\mathfrak{N}$: $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}$
by *clarsimp*
interpret *is-tiny-tdghm* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule* $\mathfrak{N}$)
have $\mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$ **by** (*auto intro*: *dg-cs-intros*)
then show $\exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$ **by** *auto*
qed
qed

lemma *small-tiny-tdghms*[*simp*]:
small $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$
by
(
rule
down[
of - ‹*set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$›
]
)
auto

lemma *small-these-tiny-tdghms*[*simp*]:
small $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$
by
(
rule
down[
of - ‹*set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{DGHM.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}\}$›
]
)
auto

lemma *tiny-tdghms-vsubset-Vset*[*simp*]:
set {𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦$_{DGHM.tiny}$ 𝔊 : 𝔄 ↦↦$_{DG.tiny\alpha}$ 𝔅} ⊆$_\circ$ *Vset* α
(**is** ‹*set ?tdghms* ⊆$_\circ$ -›)
proof(*cases* ‹*tiny-digraph* α 𝔄 ∧ *tiny-digraph* α 𝔅›)
case *True*
then have *tiny-digraph* α 𝔄 **and** *tiny-digraph* α 𝔅 **by** *auto*
show *?thesis*
proof(*rule vsubsetI*)
fix 𝔑 **assume** 𝔑 ∈$_\circ$ *set ?tdghms*
then obtain 𝔉 𝔊 **where** 𝔉: 𝔑 : 𝔉 ↦$_{DGHM.tiny}$ 𝔊 : 𝔄 ↦↦$_{DG.tiny\alpha}$ 𝔅
by *clarsimp*
interpret *is-tiny-tdghm* α 𝔄 𝔅 𝔉 𝔊 𝔑 **by** (*rule* 𝔉)
from *tiny-tdghm-in-Vset* **show** 𝔑 ∈$_\circ$ *Vset* α **by** *simp*
qed
next
case *False*
then have *set ?tdghms* = 0 **by** *fastforce*
then show *?thesis* **by** *simp*
qed

lemma (**in** *is-tdghm*) *tdghm-is-tiny-tdghm-if-ge-Limit*:
assumes $\mathcal{Z}$ β **and** α ∈$_\circ$ β
shows 𝔑 : 𝔉 ↦$_{DGHM.tiny}$ 𝔊 : 𝔄 ↦↦$_{DG.tiny\beta}$ 𝔅
proof(*intro is-tiny-tdghmI*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(1))
show 𝔑 : 𝔉 ↦$_{DGHM}$ 𝔊 : 𝔄 ↦↦$_{DG\beta}$ 𝔅
by (*intro tdghm-is-tdghm-if-ge-Limit*)
(*use assms*(2) **in** ‹*cs-concl cs-shallow cs-intro*: *dg-cs-intros*›)+
show 𝔉 : 𝔄 ↦↦$_{DG.tiny\beta}$ 𝔅 𝔊 : 𝔄 ↦↦$_{DG.tiny\beta}$ 𝔅
by
(
simp-all add:
NTDom.dghm-is-tiny-dghm-if-ge-Limit
NTCod.dghm-is-tiny-dghm-if-ge-Limit
β.$\mathcal{Z}$*-axioms*
assms(2)
)
qed (*rule assms*(1))

Further elementary results.

lemma *these-tiny-tdghms-iff*:
𝔑 ∈$_\circ$ *these-tiny-tdghms* α 𝔄 𝔅 𝔉 𝔊 ⟷
𝔑 : 𝔉 ↦$_{DGHM.tiny}$ 𝔊 : 𝔄 ↦↦$_{DG.tiny\alpha}$ 𝔅
by *auto*

Opposite transformation of homomorphisms of tiny digraphs

lemma (**in** *is-tiny-tdghm*) *is-tm-tdghm-op*: *op-tdghm* 𝔑 :
op-dghm 𝔊 ↦$_{DGHM.tiny}$ *op-dghm* 𝔉 : *op-dg* 𝔄 ↦↦$_{DG.tiny\alpha}$ *op-dg* 𝔅
by (*intro is-tiny-tdghmI*)
(*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros dg-op-intros*)+

lemma (**in** *is-tiny-tdghm*) *is-tiny-tdghm-op'*[*dg-op-intros*]:
assumes 𝔊′ = *op-dghm* 𝔊
and 𝔉′ = *op-dghm* 𝔉
and 𝔄′ = *op-dg* 𝔄
and 𝔅′ = *op-dg* 𝔅

shows *op-tdghm* $\mathfrak{N} : \mathfrak{G}' \mapsto_{DGHM.tiny} \mathfrak{F}' : \mathfrak{A}' \mapsto\mapsto_{DG.tiny\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-tm-tdghm-op*)

lemmas *is-tiny-tdghm-op*[*dg-op-intros*] = *is-tiny-tdghm.is-tiny-tdghm-op'*

3.8 Product digraph

3.8.1 Background

The concept of a product digraph, as presented in this work, is a generalization of the concept of a product category, as presented in Chapter II-3 in [39].

named-theorems *dg-prod-cs-simps*
named-theorems *dg-prod-cs-intros*

3.8.2 Product digraph: definition and elementary properties

definition *dg-prod* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  where *dg-prod* I $\mathfrak{A}$ =
    [
      $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!))$,
      $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!))$,
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Dom|\!))(\!|f(\!|i|\!)|\!)))$,
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Cod|\!))(\!|f(\!|i|\!)|\!)))$
    ]$_{\circ}$

syntax *-PDIGRAPH* :: *pttrn* $\Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  ($\langle(3\prod_{DG}$-$\in_{\circ}$-./ -)$\rangle$ [0, 0, 10] 10)
syntax-consts *-PDIGRAPH* $\rightleftharpoons$ *dg-prod*
translations $\prod_{DG} i \in_{\circ} I.\ \mathfrak{A} \rightleftharpoons$ *CONST dg-prod* I $(\lambda i.\ \mathfrak{A})$

Components.

lemma *dg-prod-components*:
  shows $(\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!) = (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!))$
    and $(\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!) = (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!))$
    and $(\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Dom|\!) =$
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Dom|\!))(\!|f(\!|i|\!)|\!)))$
    and $(\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Cod|\!) =$
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Cod|\!))(\!|f(\!|i|\!)|\!)))$
  unfolding *dg-prod-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

3.8.3 Local assumptions for a product digraph

locale *pdigraph-base* = $\mathcal{Z}$ α **for** α I **and** $\mathfrak{A}$ :: $V \Rightarrow V$ +
  assumes *pdg-digraphs*[*dg-prod-cs-intros*]: $i \in_{\circ} I \Longrightarrow$ *digraph* α $(\mathfrak{A}\ i)$
    and *pdg-index-in-Vset*[*dg-cs-intros*]: $I \in_{\circ}$ *Vset* α

Rules.

lemma (**in** *pdigraph-base*) *pdigraph-base-axioms'*[*dg-prod-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $I' = I$
  shows *pdigraph-base* α' I' $\mathfrak{A}$
  unfolding *assms* **by** (*rule pdigraph-base-axioms*)

mk-ide rf *pdigraph-base-def*[*unfolded pdigraph-base-axioms-def*]
  |*intro pdigraph-baseI*|
  |*dest pdigraph-baseD*[*dest*]|
  |*elim pdigraph-baseE*[*elim*]|

Elementary properties.

lemma (**in** *pdigraph-base*) *pdg-Obj-in-Vset*:
  assumes $\mathcal{Z}$ β **and** $\alpha \in_{\circ} \beta$
  shows $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!)) \in_{\circ}$ *Vset* β
proof(*rule Vset-trans*)

interpret β: $\mathcal{Z}$ β **by** (*rule assms*(1))
show $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!)) \in_{\circ} Vset\ (succ\ (succ\ \alpha))$
proof
(
rule vsubset-in-VsetI,
rule Limit-vproduct-vsubset-Vset-succI,
rule Limit-α,
intro dg-cs-intros
)
show $Vset\ (succ\ \alpha) \in_{\circ} Vset\ (succ\ (succ\ \alpha))$
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros*)
fix i **assume** $i \in_{\circ} I$
then interpret *digraph* α ‹$\mathfrak{A}$ i›
by (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros dg-prod-cs-intros*)
show $\mathfrak{A}\ i(\!|Obj|\!) \subseteq_{\circ} Vset\ \alpha$ **by** (*rule dg-Obj-vsubset-Vset*)
qed
from *assms*(2) **show** $Vset\ (succ\ (succ\ \alpha)) \in_{\circ} Vset\ \beta$
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros succ-in-Limit-iff*[*THEN iffD*2])
qed

lemma (**in** *pdigraph-base*) *pdg-Arr-in-Vset*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_{\circ} \beta$
shows $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)) \in_{\circ} Vset\ \beta$
proof(*rule Vset-trans*)
interpret β: $\mathcal{Z}$ β **by** (*rule assms*(1))
show $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)) \in_{\circ} Vset\ (succ\ (succ\ \alpha))$
proof
(
rule vsubset-in-VsetI,
rule Limit-vproduct-vsubset-Vset-succI,
rule Limit-α,
intro dg-cs-intros
)
fix i **assume** $i \in_{\circ} I$
then interpret *digraph* α ‹$\mathfrak{A}$ i›
by (*cs-concl* **cs-shallow cs-intro**: *dg-prod-cs-intros*)
show $\mathfrak{A}\ i(\!|Arr|\!) \subseteq_{\circ} Vset\ \alpha$ **by** (*rule dg-Arr-vsubset-Vset*)
qed (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros*)
from *assms*(2) **show** $Vset\ (succ\ (succ\ \alpha)) \in_{\circ} Vset\ \beta$
by (*cs-concl* **cs-shallow cs-intro**: *V-cs-intros succ-in-Limit-iff*[*THEN iffD*2])
qed

lemmas-with (**in** *pdigraph-base*) [*folded dg-prod-components*]:
pdg-dg-prod-Obj-in-Vset[*dg-cs-intros*] = *pdg-Obj-in-Vset*
and *pdg-dg-prod-Arr-in-Vset*[*dg-cs-intros*] = *pdg-Arr-in-Vset*

lemma (**in** *pdigraph-base*) *pdg-vsubset-index-pdigraph-base*:
assumes $J \subseteq_{\circ} I$
shows *pdigraph-base* α J $\mathfrak{A}$
using *assms*
by (*intro pdigraph-baseI*)
(*auto simp*: *vsubset-in-VsetI dg-cs-intros intro*: *dg-prod-cs-intros*)

Object

lemma *dg-prod-ObjI*:
assumes *vsv a* **and** $\mathcal{D}_{\circ}\ a = I$ **and** $\bigwedge i.\ i \in_{\circ} I \Longrightarrow a(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Obj|\!)$
shows $a \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$

using *assms* **unfolding** *dg-prod-components* **by** *auto*

lemma *dg-prod-ObjD*:
assumes $a \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
shows *vsv a* **and** $\mathcal{D}_{\circ}\ a = I$ **and** $\bigwedge i.\ i \in_{\circ} I \Longrightarrow a(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Obj|\!)$
using *assms* **unfolding** *dg-prod-components* **by** *auto*

lemma *dg-prod-ObjE*:
assumes $a \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
obtains *vsv a* **and** $\mathcal{D}_{\circ}\ a = I$ **and** $\bigwedge i.\ i \in_{\circ} I \Longrightarrow a(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Obj|\!)$
using *assms* **by** (*auto dest*: *dg-prod-ObjD*)

lemma *dg-prod-Obj-cong*:
assumes $g \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
and $f \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
and $\bigwedge i.\ i \in_{\circ} I \Longrightarrow g(\!|i|\!) = f(\!|i|\!)$
shows $g = f$
using *assms* **by** (*intro vsv-eqI*[*of g f*]) (*force simp*: *dg-prod-components*)+

Arrow

lemma *dg-prod-ArrI*:
assumes *vsv f* **and** $\mathcal{D}_{\circ}\ f = I$ **and** $\bigwedge i.\ i \in_{\circ} I \Longrightarrow f(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Arr|\!)$
shows $f \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
using *assms* **unfolding** *dg-prod-components* **by** *auto*

lemma *dg-prod-ArrD*:
assumes $f \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
shows *vsv f* **and** $\mathcal{D}_{\circ}\ f = I$ **and** $\bigwedge i.\ i \in_{\circ} I \Longrightarrow f(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Arr|\!)$
using *assms* **unfolding** *dg-prod-components* **by** *auto*

lemma *dg-prod-ArrE*:
assumes $f \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
obtains *vsv f* **and** $\mathcal{D}_{\circ}\ f = I$ **and** $\bigwedge i.\ i \in_{\circ} I \Longrightarrow f(\!|i|\!) \in_{\circ} \mathfrak{A}\ i(\!|Arr|\!)$
using *assms* **by** (*auto dest*: *dg-prod-ArrD*)

lemma *dg-prod-Arr-cong*:
assumes $g \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
and $f \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
and $\bigwedge i.\ i \in_{\circ} I \Longrightarrow g(\!|i|\!) = f(\!|i|\!)$
shows $g = f$
using *assms* **by** (*intro vsv-eqI*[*of g f*]) (*force simp*: *dg-prod-components*)+

Domain

mk-VLambda *dg-prod-components*(3)
|*vsv dg-prod-Dom-vsv*[*dg-cs-intros*]|
|*vdomain dg-prod-Dom-vdomain*[*folded dg-prod-components*, *dg-cs-simps*]|
|*app dg-prod-Dom-app*[*folded dg-prod-components*]|

lemma (**in** *pdigraph-base*) *dg-prod-Dom-app-in-Obj*[*dg-cs-intros*]:
assumes $f \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
shows $(\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Dom|\!)(\!|f|\!) \in_{\circ} (\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
unfolding *dg-prod-components*(1) *dg-prod-Dom-app*[*OF assms*]
proof(*intro vproductI ballI*)
fix *i* **assume** *prems*: $i \in_{\circ} I$
interpret *digraph* α ‹$\mathfrak{A}$ *i*›
by (*auto simp*: *prems intro*: *dg-prod-cs-intros*)

from *assms prems* **show** $(\lambda i \in_\circ I.\ \mathfrak{A}\ i(\!|Dom|\!)(\!|f(\!|i|\!)|\!))(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Obj|\!)$
unfolding *dg-prod-components*(2) **by** *force*
qed *simp-all*

lemma *dg-prod-Dom-app-component-app*[*dg-cs-simps*]:
assumes $f \in_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$ **and** $i \in_\circ I$
shows $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Dom|\!)(\!|f|\!)(\!|i|\!) = \mathfrak{A}\ i(\!|Dom|\!)(\!|f(\!|i|\!)|\!)$
using *assms*(2) **unfolding** *dg-prod-Dom-app*[*OF assms*(1)] **by** *simp*

Codomain

mk-VLambda *dg-prod-components*(4)
|*vsv dg-prod-Cod-vsv*[*dg-cs-intros*]|
|*vdomain dg-prod-Cod-vdomain*[*folded dg-prod-components*, *dg-cs-simps*]|
|*app dg-prod-Cod-app*[*folded dg-prod-components*]|

lemma (**in** *pdigraph-base*) *dg-prod-Cod-app-in-Obj*[*dg-cs-intros*]:
assumes $f \in_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
shows $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Cod|\!)(\!|f|\!) \in_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
unfolding *dg-prod-components*(1) *dg-prod-Cod-app*[*OF assms*]
proof(*rule vproductI*)
show $\forall\, i \in_\circ I.\ (\lambda i \in_\circ I.\ \mathfrak{A}\ i(\!|Cod|\!)(\!|f(\!|i|\!)|\!))(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Obj|\!)$
proof(*intro ballI*)
fix *i* **assume** *prems*: $i \in_\circ I$
then interpret *digraph* $\alpha\ \langle\mathfrak{A}\ i\rangle$
by (*auto intro*: *dg-prod-cs-intros*)
from *assms prems* **show** $(\lambda i \in_\circ I.\ \mathfrak{A}\ i(\!|Cod|\!)(\!|f(\!|i|\!)|\!))(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Obj|\!)$
unfolding *dg-prod-components*(2) **by** *force*
qed
qed *simp-all*

lemma *dg-prod-Cod-app-component-app*[*dg-cs-simps*]:
assumes $f \in_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$ **and** $i \in_\circ I$
shows $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Cod|\!)(\!|f|\!)(\!|i|\!) = \mathfrak{A}\ i(\!|Cod|\!)(\!|f(\!|i|\!)|\!)$
using *assms*(2) **unfolding** *dg-prod-Cod-app*[*OF assms*(1)] **by** *simp*

A product α-digraph is a tiny β-digraph

lemma (**in** *pdigraph-base*) *pdg-tiny-digraph-dg-prod*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows *tiny-digraph* β $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$
proof(*intro tiny-digraphI*)
show *vfsequence* $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$ **unfolding** *dg-prod-def* **by** *simp*
show *vcard* $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i) = 4_{\mathbb{N}}$
unfolding *dg-prod-def* **by** (*simp add*: *nat-omega-simps*)
show *vsv-dg-prod-Dom*: *vsv* $((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Dom|\!))$
unfolding *dg-prod-components* **by** *simp*
show *vdomain-dg-prod-Dom*: $\mathcal{D}_\circ\ ((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Dom|\!)) = (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
unfolding *dg-prod-components* **by** *simp*
show $\mathcal{R}_\circ\ ((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Dom|\!)) \subseteq_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
by (*rule vsubsetI*)
(
metis
dg-prod-Dom-app-in-Obj
dg-prod-Dom-vdomain
vsv.vrange-atE
vsv-dg-prod-Dom
)

```
  show vsv-dg-prod-Cod: vsv ((∏ DG i∈∘ I. 𝔄 i)(⦇Cod⦈))
    unfolding dg-prod-components by auto
  show vdomain-dg-prod-Cod: 𝒟∘ ((∏ DG i∈∘ I. 𝔄 i)(⦇Cod⦈)) = (∏ DG i∈∘ I. 𝔄 i)(⦇Arr⦈)
    unfolding dg-prod-components by auto
  show ℛ∘ ((∏ DG i∈∘ I. 𝔄 i)(⦇Cod⦈)) ⊆∘ (∏ DG i∈∘ I. 𝔄 i)(⦇Obj⦈)
    by (rule vsubsetI)
      (
        metis
          dg-prod-Cod-app-in-Obj
          vdomain-dg-prod-Cod
          vsv.vrange-atE
          vsv-dg-prod-Cod
      )
qed
  (
    auto simp:
      dg-cs-intros
      assms
      pdg-dg-prod-Arr-in-Vset[OF assms(1,2)]
      pdg-dg-prod-Obj-in-Vset[OF assms(1,2)]
  )


lemma (in pdigraph-base) pdg-tiny-digraph-dg-prod':
  tiny-digraph (α + ω) (∏ DG i∈∘ I. 𝔄 i)
  by (rule pdg-tiny-digraph-dg-prod)
    (simp-all add: 𝒵-α-αω 𝒵.intro 𝒵-Limit-αω 𝒵-ω-αω)
```

Arrow with a domain and a codomain

```
lemma (in pdigraph-base) dg-prod-is-arrI:
  assumes vsv f
    and 𝒟∘ f = I
    and vsv a
    and 𝒟∘ a = I
    and vsv b
    and 𝒟∘ b = I
    and ⋀i. i ∈∘ I ⟹ f(⦇i⦈) : a(⦇i⦈) ↦𝔄 i b(⦇i⦈)
  shows f : a ↦∏ DG i∈∘ I. 𝔄 i b
proof(intro is-arrI)
  interpret f: vsv f by (rule assms(1))
  interpret a: vsv a by (rule assms(3))
  interpret b: vsv b by (rule assms(5))
  from assms(7) have f-components: ⋀i. i ∈∘ I ⟹ f(⦇i⦈) ∈∘ 𝔄 i(⦇Arr⦈) by auto
  from assms(7) have a-components: ⋀i. i ∈∘ I ⟹ a(⦇i⦈) ∈∘ 𝔄 i(⦇Obj⦈)
    by (meson digraph.dg-is-arrD(2) pdg-digraphs)
  from assms(7) have b-components: ⋀i. i ∈∘ I ⟹ b(⦇i⦈) ∈∘ 𝔄 i(⦇Obj⦈)
    by (meson digraph.dg-is-arrD(3) pdg-digraphs)
  show f-in-Arr: f ∈∘ (∏ DG i∈∘ I. 𝔄 i)(⦇Arr⦈)
    unfolding dg-prod-components
    by (intro vproductI)
      (auto simp: f-components assms(2) f.vsv-vrange-vsubset-vifunion-app)
  show (∏ DG i∈∘ I. 𝔄 i)(⦇Dom⦈)(⦇f⦈) = a
  proof(rule vsv-eqI)
    from dg-prod-Dom-app-in-Obj[OF f-in-Arr] show vsv ((∏ DG i∈∘ I. 𝔄 i)(⦇Dom⦈)(⦇f⦈))
      unfolding dg-prod-components by clarsimp
    from dg-prod-Dom-app-in-Obj[OF f-in-Arr] assms(4) show [simp]:
      𝒟∘ ((∏ DG i∈∘ I. 𝔄 i)(⦇Dom⦈)(⦇f⦈)) = 𝒟∘ a
```

```
    unfolding dg-prod-components by clarsimp
  fix i assume i ∈ₒ 𝒟ₒ ((∏ₐ DG i∈ₒI. 𝔄 i)⦇Dom⦈⦇f⦈)
  then have i: i ∈ₒ I by (simp add: assms(4))
  from a-components assms(7) i show (∏ₐ DG i∈ₒI. 𝔄 i)⦇Dom⦈⦇f⦈⦇i⦈ = a⦇i⦈
    unfolding dg-prod-Dom-app-component-app[OF f-in-Arr i] by auto
 qed auto
 show (∏ₐ DG i∈ₒI. 𝔄 i)⦇Cod⦈⦇f⦈ = b
 proof(rule vsv-eqI)
  from dg-prod-Cod-app-in-Obj[OF f-in-Arr] show vsv ((∏ₐ DG i∈ₒI. 𝔄 i)⦇Cod⦈⦇f⦈)
    unfolding dg-prod-components by clarsimp
  from dg-prod-Cod-app-in-Obj[OF f-in-Arr] assms(6) show [simp]:
    𝒟ₒ ((∏ₐ DG i∈ₒI. 𝔄 i)⦇Cod⦈⦇f⦈) = 𝒟ₒ b
    unfolding dg-prod-components by clarsimp
  fix i assume i ∈ₒ 𝒟ₒ ((∏ₐ DG i∈ₒI. 𝔄 i)⦇Cod⦈⦇f⦈)
  then have i: i ∈ₒ I by (simp add: assms(6))
  from b-components assms(7) i show (∏ₐ DG i∈ₒI. 𝔄 i)⦇Cod⦈⦇f⦈⦇i⦈ = b⦇i⦈
    unfolding dg-prod-Cod-app-component-app[OF f-in-Arr i] by auto
 qed auto
qed

lemma (in pdigraph-base) dg-prod-is-arrD[dest]:
 assumes f : a ↦∏ₐ DG i∈ₒI. 𝔄 i b
 shows vsv f
  and 𝒟ₒ f = I
  and vsv a
  and 𝒟ₒ a = I
  and vsv b
  and 𝒟ₒ b = I
  and ⋀i. i ∈ₒ I ⟹ f⦇i⦈ : a⦇i⦈ ↦𝔄 i b⦇i⦈
proof–
 from is-arrD[OF assms] have f: f ∈ₒ (∏ₐ DG i∈ₒI. 𝔄 i)⦇Arr⦈
  and a: a ∈ₒ (∏ₐ DG i∈ₒI. 𝔄 i)⦇Obj⦈
  and b: b ∈ₒ (∏ₐ DG i∈ₒI. 𝔄 i)⦇Obj⦈
  by (auto intro: dg-cs-intros)
 then show 𝒟ₒ f = I 𝒟ₒ a = I 𝒟ₒ b = I vsv f vsv a vsv b
  unfolding dg-prod-components by auto
 fix i assume prems: i ∈ₒ I
 show f⦇i⦈ : a⦇i⦈ ↦𝔄 i b⦇i⦈
 proof(intro is-arrI)
  from assms(1) have f: f ∈ₒ (∏ₐ DG i∈ₒI. 𝔄 i)⦇Arr⦈
    and a: a ∈ₒ (∏ₐ DG i∈ₒI. 𝔄 i)⦇Obj⦈
    and b: b ∈ₒ (∏ₐ DG i∈ₒI. 𝔄 i)⦇Obj⦈
  by (auto intro: dg-cs-intros)
  from f prems show f⦇i⦈ ∈ₒ 𝔄 i⦇Arr⦈
    unfolding dg-prod-components by clarsimp
  from a b assms(1) prems dg-prod-components(2,3,4) show
    𝔄 i⦇Dom⦈⦇f⦇i⦈⦈ = a⦇i⦈ 𝔄 i⦇Cod⦈⦇f⦇i⦈⦈ = b⦇i⦈
    by fastforce+
 qed
qed

lemma (in pdigraph-base) dg-prod-is-arrE[elim]:
 assumes f : a ↦∏ₐ DG i∈ₒI. 𝔄 i b
 obtains vsv f
  and 𝒟ₒ f = I
  and vsv a
  and 𝒟ₒ a = I
  and vsv b
```

and $\mathcal{D}_\circ\ b = I$
and $\bigwedge i.\ i \in_\circ I \Longrightarrow f(\!|i|\!) : a(\!|i|\!) \mapsto_{\mathfrak{A}\ i} b(\!|i|\!)$
using *assms* **by** *auto*

3.8.4 Further local assumptions for product digraphs

Definition and elementary properties

locale *pdigraph* = *pdigraph-base* $\alpha\ I\ \mathfrak{A}$ **for** $\alpha\ I\ \mathfrak{A}$ +
assumes *pdg-Obj-vsubset-Vset*: $J \subseteq_\circ I \Longrightarrow (\prod_{DG} i \in_\circ J.\ \mathfrak{A}\ i)(\!|Obj|\!) \subseteq_\circ Vset\ \alpha$
and *pdg-Hom-vifunion-in-Vset*:
$[\![$
$J \subseteq_\circ I$;
$A \subseteq_\circ (\prod_{DG} i \in_\circ J.\ \mathfrak{A}\ i)(\!|Obj|\!)$;
$B \subseteq_\circ (\prod_{DG} i \in_\circ J.\ \mathfrak{A}\ i)(\!|Obj|\!)$;
$A \in_\circ Vset\ \alpha$;
$B \in_\circ Vset\ \alpha$
$]\!] \Longrightarrow (\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ Hom\ (\prod_{DG} i \in_\circ J.\ \mathfrak{A}\ i)\ a\ b) \in_\circ Vset\ \alpha$

Rules.

lemma (**in** *pdigraph*) *pdigraph-axioms'*[*dg-prod-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $I' = I$
shows *pdigraph* $\alpha'\ I'\ \mathfrak{A}$
unfolding *assms* **by** (*rule pdigraph-axioms*)

mk-ide rf *pdigraph-def*[*unfolded pdigraph-axioms-def*]
|*intro pdigraphI*|
|*dest pdigraphD*[*dest*]|
|*elim pdigraphE*[*elim*]|

lemmas [*dg-prod-cs-intros*] = *pdigraphD*(1)

Elementary properties.

lemma (**in** *pdigraph*) *pdg-Obj-vsubset-Vset'*: $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!) \subseteq_\circ Vset\ \alpha$
by (*rule pdg-Obj-vsubset-Vset*) *simp*

lemma (**in** *pdigraph*) *pdg-Hom-vifunion-in-Vset'*:
assumes $A \subseteq_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
and $B \subseteq_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
and $A \in_\circ Vset\ \alpha$
and $B \in_\circ Vset\ \alpha$
shows $(\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ Hom\ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)\ a\ b) \in_\circ Vset\ \alpha$
using *assms* **by** (*intro pdg-Hom-vifunion-in-Vset*) *simp-all*

lemma (**in** *pdigraph*) *pdg-vsubset-index-pdigraph*:
assumes $J \subseteq_\circ I$
shows *pdigraph* $\alpha\ J\ \mathfrak{A}$
proof(*intro pdigraphI*)
show *dg-prod* $J'\ \mathfrak{A}(\!|Obj|\!) \subseteq_\circ Vset\ \alpha$ **if** $\langle J' \subseteq_\circ J \rangle$ **for** J'
proof–
from *that assms* **have** $J' \subseteq_\circ I$ **by** *simp*
then show *dg-prod* $J'\ \mathfrak{A}(\!|Obj|\!) \subseteq_\circ Vset\ \alpha$ **by** (*rule pdg-Obj-vsubset-Vset*)
qed
fix $A\ B\ J'$ **assume** *prems*:
$J' \subseteq_\circ J$
$A \subseteq_\circ (\prod_{DG} i \in_\circ J'.\ \mathfrak{A}\ i)(\!|Obj|\!)$
$B \subseteq_\circ (\prod_{DG} i \in_\circ J'.\ \mathfrak{A}\ i)(\!|Obj|\!)$
$A \in_\circ Vset\ \alpha$

```
    B ∈∘ Vset α
  show (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (∏DG i∈∘J'. 𝔄 i) a b) ∈∘ Vset α
  proof–
    from prems(1) assms have J' ⊆∘ I by simp
    from pdg-Hom-vifunion-in-Vset[OF this prems(2–5)] show ?thesis.
  qed
qed (rule pdg-vsubset-index-pdigraph-base[OF assms])
```

A product α-digraph is an α-digraph

```
lemma (in pdigraph) pdg-digraph-dg-prod: digraph α (∏DG i∈∘I. 𝔄 i)
proof–
  interpret tiny-digraph ‹α + ω› ‹∏DG i∈∘I. 𝔄 i›
    by (intro pdg-tiny-digraph-dg-prod)
      (auto simp: 𝒵-α-αω 𝒵.intro 𝒵-Limit-αω 𝒵-ω-αω)
  show ?thesis
    by (rule digraph-if-digraph)
      (
        auto
          intro!: pdg-Hom-vifunion-in-Vset pdg-Obj-vsubset-Vset
          intro: dg-cs-intros
      )
qed
```

3.8.5 Local assumptions for a finite product digraph

Definition and elementary properties

```
locale finite-pdigraph = pdigraph-base α I 𝔄 for α I 𝔄 +
  assumes fin-pdg-index-vfinite: vfinite I
```

Rules.

```
lemma (in finite-pdigraph) finite-pdigraph-axioms'[dg-prod-cs-intros]:
  assumes α' = α and I' = I
  shows finite-pdigraph α' I' 𝔄
  unfolding assms by (rule finite-pdigraph-axioms)

mk-ide rf finite-pdigraph-def[unfolded finite-pdigraph-axioms-def]
  |intro finite-pdigraphI|
  |dest finite-pdigraphD[dest]|
  |elim finite-pdigraphE[elim]|

lemmas [dg-prod-cs-intros] = finite-pdigraphD(1)
```

Local assumptions for a finite product digraph and local assumptions for an arbitrary product digraph

```
sublocale finite-pdigraph ⊆ pdigraph α I 𝔄
proof(intro pdigraphI)
  show (∏DG i∈∘J. 𝔄 i)(|Obj|) ⊆∘ Vset α if J ⊆∘ I for J
    unfolding dg-prod-components
  proof–
    from that fin-pdg-index-vfinite have J: vfinite J
      by (cs-concl cs-shallow cs-intro: vfinite-vsubset)
    show (∏∘i∈∘J. 𝔄 i(|Obj|)) ⊆∘ Vset α
    proof(intro vsubsetI)
      fix A assume prems: A ∈∘ (∏∘i∈∘J. 𝔄 i(|Obj|))
      note A = vproductD[OF prems, rule-format]
      show A ∈∘ Vset α
```

```
    proof(rule vsv.vsv-Limit-vsv-in-VsetI)
      from that show 𝒟∘ A ∈∘ Vset α
        unfolding A(2) by (auto intro: pdg-index-in-Vset)
      show ℛ∘ A ⊆∘ Vset α
      proof(intro vsv.vsv-vrange-vsubset, unfold A(2))
        fix i assume prems': i ∈∘ J
        with that have i: i ∈∘ I by auto
        interpret digraph α ‹𝔄 i›
          by (cs-concl cs-shallow cs-intro: dg-prod-cs-intros i)
        have A(|i|) ∈∘ 𝔄 i(|Obj|) by (rule A(3)[OF prems'])
        then show A(|i|) ∈∘ Vset α by (cs-concl cs-intro: dg-cs-intros)
      qed (intro A(1))
    qed (auto simp: A(2) intro!: J A(1))
  qed
qed
show (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (∏DGi∈∘J. 𝔄 i) a b) ∈∘ Vset α
  if J: J ⊆∘ I
    and A: A ⊆∘ (∏DGi∈∘J. 𝔄 i)(|Obj|)
    and B: B ⊆∘ (∏DGi∈∘J. 𝔄 i)(|Obj|)
    and A-in-Vset: A ∈∘ Vset α
    and B-in-Vset: B ∈∘ Vset α
  for J A B
proof–
  interpret J: pdigraph-base α J 𝔄
    by (intro J pdg-vsubset-index-pdigraph-base)
  let ?UA = ‹⋃∘(⋃∘(⋃∘A))› and ?UB = ‹⋃∘(⋃∘(⋃∘B))›
  from that(4) have UA: ?UA ∈∘ Vset α by (intro VUnion-in-VsetI)
  from that(5) have UB: ?UB ∈∘ Vset α by (intro VUnion-in-VsetI)
  have (∏∘i∈∘J. (⋃∘a∈∘?UA. ⋃∘b∈∘?UB. Hom (𝔄 i) a b)) ∈∘ Vset α
  proof(intro Limit-vproduct-in-VsetI)
    from that(1) show J ∈∘ Vset α by (auto intro!: pdg-index-in-Vset)
    show (⋃∘a∈∘?UA. ⋃∘b∈∘?UB. Hom (𝔄 i) a b) ∈∘ Vset α if i: i ∈∘ J for i
    proof–
      from i J have i: i ∈∘ I by auto
      interpret digraph α ‹𝔄 i›
        using i by (cs-concl cs-intro: dg-prod-cs-intros)
      have [dg-cs-simps]: (⋃∘a∈∘?UA. ⋃∘b∈∘?UB. Hom (𝔄 i) a b) ⊆∘
        (⋃∘a∈∘?UA ∩∘ 𝔄 i(|Obj|). ⋃∘b∈∘?UB ∩∘ 𝔄 i(|Obj|). Hom (𝔄 i) a b)
      proof(intro vsubsetI)
        fix f assume f ∈∘ (⋃∘a∈∘?UA. ⋃∘b∈∘?UB. Hom (𝔄 i) a b)
        then obtain a b
          where a: a ∈∘ ?UA and b: b ∈∘ ?UB and f: f : a ↦𝔄 i b
          by (elim vifunionE, unfold in-Hom-iff)
        then show
          f ∈∘ (⋃∘a∈∘?UA ∩∘ 𝔄 i(|Obj|). ⋃∘b∈∘?UB ∩∘ 𝔄 i(|Obj|). Hom (𝔄 i) a b)
          by (intro vifunionI, unfold in-Hom-iff) (auto intro!: f b a)
      qed
      moreover from UA UB have
        (⋃∘a∈∘?UA ∩∘ 𝔄 i(|Obj|). ⋃∘b∈∘?UB ∩∘ 𝔄 i(|Obj|). Hom (𝔄 i) a b) ∈∘
          Vset α
        by (intro dg-Hom-vifunion-in-Vset) auto
      ultimately show ?thesis by auto
    qed
    from J show vfinite J
      by (rule vfinite-vsubset[OF fin-pdg-index-vfinite])
  qed auto
  moreover have
    (⋃∘a∈∘A. ⋃∘b∈∘B. Hom (∏DGi∈∘J. 𝔄 i) a b) ⊆∘
```

```
      (∏ₒ i∈ₒ J. (⋃ₒ a∈ₒ ?UA. ⋃ₒ b∈ₒ ?UB. Hom (𝔄 i) a b))
    proof(intro vsubsetI)
      fix f assume f ∈ₒ (⋃ₒ a∈ₒ A. ⋃ₒ b∈ₒ B. Hom (∏ DG i∈ₒ J. 𝔄 i) a b)
      then obtain a b
        where a: a ∈ₒ A and b: b ∈ₒ B and f: f ∈ₒ Hom (∏ DG i∈ₒ J. 𝔄 i) a b
        by auto
      from f have f: f : a ↦(∏ DG i∈ₒ J. 𝔄 i) b by simp
      show f ∈ₒ (∏ₒ i∈ₒ J. (⋃ₒ a∈ₒ ?UA. ⋃ₒ b∈ₒ ?UB. Hom (𝔄 i) a b))
      proof
        (
          intro vproductI,
          unfold Ball-def;
          (intro allI impI)?;
          (intro vifunionI)?;
          (unfold in-Hom-iff)?
        )
        from f show vsv f by (auto simp: dg-prod-components(2))
        from f show 𝒟ₒ f = J by (auto simp: dg-prod-components(2))
        fix i assume i: i ∈ₒ J
        show a⦇i⦈ ∈ₒ ?UA
          by
            (
              intro vprojection-in-VUnionI,
              rule that(2)[unfolded dg-prod-components(1)];
              intro a i
            )
        show b⦇i⦈ ∈ₒ ?UB
          by
            (
              intro vprojection-in-VUnionI,
              rule that(3)[unfolded dg-prod-components(1)];
              intro b i
            )
        show f⦇i⦈ : a⦇i⦈ ↦𝔄 i b⦇i⦈ by (rule J.dg-prod-is-arrD(7)[OF f i])
      qed
    qed
    ultimately show (⋃ₒ a∈ₒ A. ⋃ₒ b∈ₒ B. Hom (∏ DG i∈ₒ J. 𝔄 i) a b) ∈ₒ Vset α
      by blast
  qed
qed (intro pdigraph-base-axioms)
```

3.8.6 Binary union and complement

Application-specific methods

```
method vdiff-of-vunion uses rule assms subset =
  (
    rule
      rule
        [
          OF vintersection-complement assms,
          unfolded vunion-complement[OF subset]
        ]
  )

method vdiff-of-vunion′ uses rule assms subset =
  (
    rule
```

```
    rule
     [
      OF vintersection-complement complement-vsubset subset assms,
      unfolded vunion-complement[OF subset]
     ]
)
```

Results

```
lemma dg-prod-vunion-Obj-in-Obj:
  assumes vdisjnt J K
    and b ∈ₒ (∏ DG j∈ₒJ. 𝔄 j)(|Obj|)
    and c ∈ₒ (∏ DG k∈ₒK. 𝔄 k)(|Obj|)
  shows b ∪ₒ c ∈ₒ (∏ DG i∈ₒJ ∪ₒ K. 𝔄 i)(|Obj|)
proof–

  interpret b: vsv b using assms(2) unfolding dg-prod-components by clarsimp
  interpret c: vsv c using assms(3) unfolding dg-prod-components by clarsimp

  from assms(2,3) have dom-b: 𝒟ₒ b = J and dom-c: 𝒟ₒ c = K
    unfolding dg-prod-components by auto
  from assms(1) have disjnt: 𝒟ₒ b ∩ₒ 𝒟ₒ c = 0 unfolding dom-b dom-c by auto

  show ?thesis
    unfolding dg-prod-components
  proof(intro vproductI)
    show 𝒟ₒ (b ∪ₒ c) = J ∪ₒ K by (auto simp: vdomain-vunion dom-b dom-c)
    show ∀ i∈ₒJ ∪ₒ K. (b ∪ₒ c)(|i|) ∈ₒ 𝔄 i(|Obj|)
    proof(intro ballI)
      fix i assume prems: i ∈ₒ J ∪ₒ K
      then consider (ib) ‹i ∈ₒ 𝒟ₒ b› | (ic) ‹i ∈ₒ 𝒟ₒ c›
        unfolding dom-b dom-c by auto
      then show (b ∪ₒ c)(|i|) ∈ₒ 𝔄 i(|Obj|)
      proof cases
        case ib
        with prems disjnt have bc-i: (b ∪ₒ c)(|i|) = b(|i|)
          by (auto intro!: vsv-vunion-app-left)
        from assms(2) ib show ?thesis unfolding bc-i dg-prod-components by auto
      next
        case ic
        with prems disjnt have bc-i: (b ∪ₒ c)(|i|) = c(|i|)
          by (auto intro!: vsv-vunion-app-right)
        from assms(3) ic show ?thesis unfolding bc-i dg-prod-components by auto
      qed
    qed
  qed (auto simp: disjnt)

qed

lemma dg-prod-vdiff-vunion-Obj-in-Obj:
  assumes J ⊆ₒ I
    and b ∈ₒ (∏ DG k∈ₒI −ₒ J. 𝔄 k)(|Obj|)
    and c ∈ₒ (∏ DG j∈ₒJ. 𝔄 j)(|Obj|)
  shows b ∪ₒ c ∈ₒ (∏ DG i∈ₒI. 𝔄 i)(|Obj|)
  by
    (
      vdiff-of-vunion
        rule: dg-prod-vunion-Obj-in-Obj assms: assms(2,3) subset: assms(1)
```

)

lemma *dg-prod-vunion-Arr-in-Arr*:
assumes *vdisjnt J K*
and $b \in_\circ (\prod_{DG} j \in_\circ J.\ \mathfrak{A}\ j)(\!|Arr|\!)$
and $c \in_\circ (\prod_{DG} k \in_\circ K.\ \mathfrak{A}\ k)(\!|Arr|\!)$
shows $b \cup_\circ c \in_\circ (\prod_{DG} i \in_\circ J \cup_\circ K.\ \mathfrak{A}\ i)(\!|Arr|\!)$
unfolding *dg-prod-components*
proof(*intro vproductI*)

interpret *b*: *vsv b* **using** *assms*(2) **unfolding** *dg-prod-components* **by** *clarsimp*
interpret *c*: *vsv c* **using** *assms*(3) **unfolding** *dg-prod-components* **by** *clarsimp*

from *assms* **have** *dom-b*: $\mathcal{D}_\circ\ b = J$ **and** *dom-c*: $\mathcal{D}_\circ\ c = K$
unfolding *dg-prod-components* **by** *auto*
from *assms* **have** *disjnt*: $\mathcal{D}_\circ\ b \cap_\circ \mathcal{D}_\circ\ c = 0$ **unfolding** *dom-b dom-c* **by** *auto*

from *disjnt* **show** *vsv* $(b \cup_\circ c)$ **by** *auto*
show *dom-bc*: $\mathcal{D}_\circ\ (b \cup_\circ c) = J \cup_\circ K$
unfolding *vdomain-vunion dom-b dom-c* **by** *auto*

show $\forall\, i \in_\circ J \cup_\circ K.\ (b \cup_\circ c)(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Arr|\!)$
proof(*intro ballI*)
fix *i* **assume** *prems*: $i \in_\circ J \cup_\circ K$
then consider (*ib*) $\langle i \in_\circ \mathcal{D}_\circ\ b\rangle$ | (*ic*) $\langle i \in_\circ \mathcal{D}_\circ\ c\rangle$
unfolding *dom-b dom-c* **by** *auto*
then show $(b \cup_\circ c)(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Arr|\!)$
proof *cases*
case *ib*
with *prems disjnt* **have** *bc-i*: $(b \cup_\circ c)(\!|i|\!) = b(\!|i|\!)$
by (*auto intro*!: *vsv-vunion-app-left*)
from *assms*(2) *ib* **show** *?thesis* **unfolding** *bc-i dg-prod-components* **by** *auto*
next
case *ic*
with *prems disjnt* **have** *bc-i*: $(b \cup_\circ c)(\!|i|\!) = c(\!|i|\!)$
by (*auto intro*!: *vsv-vunion-app-right*)
from *assms*(3) *ic* **show** *?thesis* **unfolding** *bc-i dg-prod-components* **by** *auto*
qed
qed

qed

lemma *dg-prod-vdiff-vunion-Arr-in-Arr*:
assumes $J \subseteq_\circ I$
and $b \in_\circ (\prod_{DG} k \in_\circ I -_\circ J.\ \mathfrak{A}\ k)(\!|Arr|\!)$
and $c \in_\circ (\prod_{DG} j \in_\circ J.\ \mathfrak{A}\ j)(\!|Arr|\!)$
shows $b \cup_\circ c \in_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
by
(
vdiff-of-vunion
rule: *dg-prod-vunion-Arr-in-Arr assms*: *assms*(2,3) *subset*: *assms*(1)
)

lemma (**in** *pdigraph*) *pdg-dg-prod-vunion-is-arr*:
assumes *vdisjnt J K*
and $J \subseteq_\circ I$
and $K \subseteq_\circ I$
and $g : a \mapsto_{(\prod_{DG} j \in_\circ J.\ \mathfrak{A}\ j)} b$

```
    and f : c ↦(∏ DG k∈∘ K. 𝔄 k) d
  shows g ∪∘ f : a ∪∘ c ↦(∏ DG i∈∘ J ∪∘ K. 𝔄 i) b ∪∘ d
proof–

  interpret J𝔄: pdigraph α J 𝔄
    using assms(2) by (simp add: pdg-vsubset-index-pdigraph)
  interpret K𝔄: pdigraph α K 𝔄
    using assms(3) by (simp add: pdg-vsubset-index-pdigraph)
  interpret JK𝔄: pdigraph α ‹J ∪∘ K› 𝔄
    using assms(2,3) by (simp add: pdg-vsubset-index-pdigraph)

  show ?thesis
  proof(intro JK𝔄.dg-prod-is-arrI)

    note gD = J𝔄.dg-prod-is-arrD[OF assms(4)]
      and fD = K𝔄.dg-prod-is-arrD[OF assms(5)]

    from assms(1) gD fD show
      vsv (g ∪∘ f)
      𝒟∘ (g ∪∘ f) = J ∪∘ K
      vsv (a ∪∘ c)
      𝒟∘ (a ∪∘ c) = J ∪∘ K
      vsv (b ∪∘ d)
      𝒟∘ (b ∪∘ d) = J ∪∘ K
      by (auto simp: vdomain-vunion)

    fix i assume i ∈∘ J ∪∘ K
    then consider (iJ) ‹i ∈∘ J› | (iK) ‹i ∈∘ K› by auto
    then show (g ∪∘ f)(|i|) : (a ∪∘ c)(|i|) ↦𝔄 i (b ∪∘ d)(|i|)
    proof cases
      case iJ
      have gf-i: (g ∪∘ f)(|i|) = g(|i|) by (simp add: iJ assms(1) gD(1,2) fD(1,2))
      have ac-i: (a ∪∘ c)(|i|) = a(|i|) by (simp add: iJ assms(1) gD(3,4) fD(3,4))
      have bd-i: (b ∪∘ d)(|i|) = b(|i|) by (simp add: iJ assms(1) gD(5,6) fD(5,6))
      show ?thesis unfolding gf-i ac-i bd-i by (rule gD(7)[OF iJ])
    next
      case iK
      have gf-i: (g ∪∘ f)(|i|) = f(|i|) by (simp add: iK assms(1) gD(1,2) fD(1,2))
      have ac-i: (a ∪∘ c)(|i|) = c(|i|) by (simp add: iK assms(1) gD(3,4) fD(3,4))
      have bd-i: (b ∪∘ d)(|i|) = d(|i|) by (simp add: iK assms(1) gD(5,6) fD(5,6))
      show ?thesis unfolding gf-i ac-i bd-i by (rule fD(7)[OF iK])
    qed

  qed

qed

lemma (in pdigraph) pdg-dg-prod-vdiff-vunion-is-arr:
  assumes J ⊆∘ I
    and g : a ↦(∏ DG k∈∘ I −∘ J. 𝔄 k) b
    and f : c ↦(∏ DG j∈∘ J. 𝔄 j) d
  shows g ∪∘ f : a ∪∘ c ↦∏ DG i∈∘ I. 𝔄 i b ∪∘ d
  by
    (
      vdiff-of-vunion'
        rule: pdg-dg-prod-vunion-is-arr assms: assms(2,3) subset: assms(1)
    )
```

3.8.7 Projection

Definition and elementary properties

See Chapter II-3 in [39].

definition *dghm-proj* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow V$ ($\langle \pi_{DG} \rangle$)
  where $\pi_{DG}\ I\ \mathfrak{A}\ i =$
    [
      $(\lambda a \in_\circ ((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\lfloor Obj \rfloor)).\ a(\lfloor i \rfloor))$,
      $(\lambda f \in_\circ ((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\lfloor Arr \rfloor)).\ f(\lfloor i \rfloor))$,
      $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$,
      $\mathfrak{A}\ i$
    ]$_\circ$

Components.

lemma *dghm-proj-components*:
  shows $\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor ObjMap \rfloor) = (\lambda a \in_\circ ((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\lfloor Obj \rfloor)).\ a(\lfloor i \rfloor))$
    and $\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor ArrMap \rfloor) = (\lambda f \in_\circ ((\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\lfloor Arr \rfloor)).\ f(\lfloor i \rfloor))$
    and $\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor HomDom \rfloor) = (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$
    and $\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor HomCod \rfloor) = \mathfrak{A}\ i$
  unfolding *dghm-proj-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object map.

mk-VLambda *dghm-proj-components*(1)
  |*vsv dghm-proj-ObjMap-vsv*[*dg-cs-intros*]|
  |*vdomain dghm-proj-ObjMap-vdomain*[*dg-cs-simps*]|
  |*app dghm-proj-ObjMap-app*[*dg-cs-simps*]|

lemma (**in** *pdigraph*) *dghm-proj-ObjMap-vrange*:
  assumes $i \in_\circ I$
  shows $\mathcal{R}_\circ\ (\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor ObjMap \rfloor)) \subseteq_\circ \mathfrak{A}\ i(\lfloor Obj \rfloor)$
  using *assms*
  unfolding *dghm-proj-components*
  by (*intro vrange-VLambda-vsubset*) (*clarsimp simp*: *dg-prod-components*)

Arrow map.

mk-VLambda *dghm-proj-components*(2)
  |*vsv dghm-proj-ArrMap-vsv*[*dg-cs-intros*]|
  |*vdomain dghm-proj-ArrMap-vdomain*[*dg-cs-simps*]|
  |*app dghm-proj-ArrMap-app*[*dg-cs-simps*]|

lemma (**in** *pdigraph*) *dghm-proj-ArrMap-vrange*:
  assumes $i \in_\circ I$
  shows $\mathcal{R}_\circ\ (\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor ArrMap \rfloor)) \subseteq_\circ \mathfrak{A}\ i(\lfloor Arr \rfloor)$
  using *assms*
  unfolding *dghm-proj-components*
  by (*intro vrange-VLambda-vsubset*) (*clarsimp simp*: *dg-prod-components*)

A projection digraph homomorphism is a digraph homomorphism

lemma (**in** *pdigraph*) *pdg-dghm-proj-is-dghm*:
  assumes $i \in_\circ I$
  shows $\pi_{DG}\ I\ \mathfrak{A}\ i : (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i) \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$
proof(*intro is-dghmI*)
  show *vfsequence* ($\pi_{DG}\ I\ \mathfrak{A}\ i$) **unfolding** *dghm-proj-def* **by** *auto*
  show *vcard* ($\pi_{DG}\ I\ \mathfrak{A}\ i$) = $4_\mathbb{N}$
    unfolding *dghm-proj-def* **by** (*simp add*: *nat-omega-simps*)
  show $\pi_{DG}\ I\ \mathfrak{A}\ i(\lfloor HomDom \rfloor) = (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$

unfolding *dghm-proj-components* **by** *simp*
show $\pi_{DG}\ I\ \mathfrak{A}\ i(\!|HomCod|\!) = \mathfrak{A}\ i$
unfolding *dghm-proj-components* **by** *simp*
fix *f a b* **assume** $f : a \mapsto_{\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i}\ b$
with *assms pdg-digraph-dg-prod* **show**
$\pi_{DG}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|f|\!) : \pi_{DG}\ I\ \mathfrak{A}\ i(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\mathfrak{A}\ i}\ \pi_{DG}\ I\ \mathfrak{A}\ i(\!|ObjMap|\!)(\!|b|\!)$
by (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-prod-is-arrD*(7))
qed
(
auto simp:
dg-cs-simps dg-cs-intros dg-prod-cs-intros
assms pdg-digraph-dg-prod dghm-proj-ObjMap-vrange
)

lemma (**in** *pdigraph*) *pdg-dghm-proj-is-dghm′*:
assumes $i \in_\circ I$ **and** $\mathfrak{C} = (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$ **and** $\mathfrak{D} = \mathfrak{A}\ i$
shows $\pi_{DG}\ I\ \mathfrak{A}\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{D}$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule pdg-dghm-proj-is-dghm*)

lemmas [*dg-cs-intros*] = *pdigraph.pdg-dghm-proj-is-dghm′*

3.8.8 Digraph product universal property digraph homomorphism

Definition and elementary properties

The following digraph homomorphism is used in the proof of the universal property of the product digraph later in this work.

definition *dghm-up* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ =
[
$(\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ObjMap|\!)(\!|a|\!)))$,
$(\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ArrMap|\!)(\!|f|\!)))$,
$\mathfrak{C}$,
$(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$
]$_\circ$

Components.

lemma *dghm-up-components*:
shows *dghm-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 *dghm-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 *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|HomDom|\!) = \mathfrak{C}$
and *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|HomCod|\!) = (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$
unfolding *dghm-up-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object map

mk-VLambda *dghm-up-components*(1)
|*vsv dghm-up-ObjMap-vsv*[*dg-cs-intros*]|
|*vdomain dghm-up-ObjMap-vdomain*[*dg-cs-simps*]|
|*app dghm-up-ObjMap-app*|

lemma *dghm-up-ObjMap-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\textit{dghm-up I}\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|ObjMap|\!)) \subseteq_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Obj|\!)$
unfolding *dghm-up-components dg-prod-components*
proof(*intro vrange-VLambda-vsubset vproductI*)
fix *a* **assume** *prems*: $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
show $\forall\, i \in_\circ I.\ (\lambda i \in_\circ I.\ \varphi\ i(\!|ObjMap|\!)(\!|a|\!))(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Obj|\!)$

```
  proof(intro ballI)
    fix i assume prems': i ∈ₒ I
    interpret φ: is-dghm α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms[OF prems'])
    from prems prems' show (λi∈ₒI. φ i(|ObjMap|)(|a|))(|i|) ∈ₒ 𝔄 i(|Obj|)
      by (simp add: φ.dghm-ObjMap-app-in-HomCod-Obj)
  qed
qed auto

lemma dghm-up-ObjMap-app-vdomain[dg-cs-simps]:
  assumes a ∈ₒ 𝔈(|Obj|)
  shows 𝒟ₒ (dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|)) = I
  unfolding dghm-up-ObjMap-app[OF assms] by simp

lemma dghm-up-ObjMap-app-component[dg-cs-simps]:
  assumes a ∈ₒ 𝔈(|Obj|) and i ∈ₒ I
  shows dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|)(|i|) = φ i(|ObjMap|)(|a|)
  using assms unfolding dghm-up-components by simp

lemma dghm-up-ObjMap-app-vrange:
  assumes a ∈ₒ 𝔈(|Obj|) and ⋀i. i ∈ₒ I ⟹ φ i : 𝔈 ↦↦_{DG α} 𝔄 i
  shows ℛₒ (dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|)) ⊆ₒ (⋃ₒi∈ₒI. 𝔄 i(|Obj|))
proof(intro vsubsetI)
  fix b assume prems: b ∈ₒ ℛₒ (dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|))
  have vsv (dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|))
    unfolding dghm-up-ObjMap-app[OF assms(1)] by auto
  with prems obtain i where b-def: b = dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|)(|i|)
    and i: i ∈ₒ I
    by (auto elim: vsv.vrange-atE simp: dghm-up-ObjMap-app[OF assms(1)])
  interpret φi: is-dghm α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms(2)[OF i])
  from dghm-up-ObjMap-app-component[OF assms(1) i] b-def have b-def':
    b = φ i(|ObjMap|)(|a|)
    by simp
  from assms(1) have b ∈ₒ 𝔄 i(|Obj|)
    unfolding b-def' by (auto intro: dg-cs-intros)
  with i show b ∈ₒ (⋃ₒi∈ₒI. 𝔄 i(|Obj|)) by force
qed
```

Arrow map

```
mk-VLambda dghm-up-components(2)
  |vsv dghm-up-ArrMap-vsv[dg-cs-intros]|
  |vdomain dghm-up-ArrMap-vdomain[dg-cs-simps]|
  |app dghm-up-ArrMap-app|

lemma (in pdigraph) dghm-up-ArrMap-vrange:
  assumes ⋀i. i ∈ₒ I ⟹ φ i : 𝔈 ↦↦_{DG α} 𝔄 i
  shows ℛₒ (dghm-up I 𝔄 𝔈 φ(|ArrMap|)) ⊆ₒ (∏_{DG}i∈ₒI. 𝔄 i)(|Arr|)
  unfolding dghm-up-components dg-prod-components
proof(intro vrange-VLambda-vsubset vproductI)
  fix a assume prems: a ∈ₒ 𝔈(|Arr|)
  show ∀ i∈ₒI. (λi∈ₒI. φ i(|ArrMap|)(|a|))(|i|) ∈ₒ 𝔄 i(|Arr|)
  proof(intro ballI)
    fix i assume prems': i ∈ₒ I
    interpret φ: is-dghm α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms[OF prems'])
    from prems prems' show (λi∈ₒI. φ i(|ArrMap|)(|a|))(|i|) ∈ₒ 𝔄 i(|Arr|)
      by (auto intro: dg-cs-intros)
  qed
qed auto
```

lemma *dghm-up-ArrMap-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)) \subseteq_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{Arr}|\!)$
proof(*intro vsubsetI*)
fix A **assume** $A \in_\circ \mathcal{R}_\circ\ (\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!))$
then obtain a **where** *A-def*: $A = \mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!)$
and a: $a \in_\circ \mathfrak{C}(\!|\mathit{Arr}|\!)$
unfolding *dghm-up-ArrMap-vdomain dghm-up-components* **by** *auto*
have $(\lambda i \in_\circ I.\ \varphi\ i(\!|\mathit{ArrMap}|\!)(\!|a|\!)) \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|\mathit{Arr}|\!))$
proof(*intro vproductI*)
show $\forall\, i \in_\circ I.\ (\lambda i \in_\circ I.\ \varphi\ i(\!|\mathit{ArrMap}|\!)(\!|a|\!))(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|\mathit{Arr}|\!)$
proof(*intro ballI*)
fix i **assume** *prems*: $i \in_\circ I$
interpret φ: *is-dghm* $\alpha\ \mathfrak{C}\ \langle\mathfrak{A}\ i\rangle\ \langle\varphi\ i\rangle$ **by** (*rule assms*[*OF prems*])
from *prems a* **show** $(\lambda i \in_\circ I.\ \varphi\ i(\!|\mathit{ArrMap}|\!)(\!|a|\!))(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|\mathit{Arr}|\!)$
by (*auto intro*: *dg-cs-intros*)
qed
qed *simp-all*
with a **show** $A \in_\circ (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{Arr}|\!)$
unfolding *A-def dg-prod-components dghm-up-components* **by** *simp*
qed

lemma *dghm-up-ArrMap-app-vdomain*[*dg-cs-simps*]:
assumes $a \in_\circ \mathfrak{C}(\!|\mathit{Arr}|\!)$
shows $\mathcal{D}_\circ\ (\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!)) = I$
unfolding *dghm-up-ArrMap-app*[*OF assms*] **by** *simp*

lemma *dghm-up-ArrMap-app-component*[*dg-cs-simps*]:
assumes $a \in_\circ \mathfrak{C}(\!|\mathit{Arr}|\!)$ **and** $i \in_\circ I$
shows $\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!)(\!|i|\!) = \varphi\ i(\!|\mathit{ArrMap}|\!)(\!|a|\!)$
using *assms* **unfolding** *dghm-up-components* **by** *simp*

lemma *dghm-up-ArrMap-app-vrange*:
assumes $a \in_\circ \mathfrak{C}(\!|\mathit{Arr}|\!)$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!)) \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|\mathit{Arr}|\!))$
proof(*intro vsubsetI*)
fix b **assume** *prems*: $b \in_\circ \mathcal{R}_\circ\ (\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!))$
have *vsv* $(\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!))$
unfolding *dghm-up-ArrMap-app*[*OF assms*(1)] **by** *auto*
with *prems* **obtain** i **where** *b-def*: $b = \mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)(\!|a|\!)(\!|i|\!)$
and i: $i \in_\circ I$
by (*auto elim*: *vsv.vrange-atE simp*: *dghm-up-ArrMap-app*[*OF assms*(1)])
interpret φi: *is-dghm* $\alpha\ \mathfrak{C}\ \langle\mathfrak{A}\ i\rangle\ \langle\varphi\ i\rangle$ **by** (*rule assms*(2)[*OF i*])
from *dghm-up-ArrMap-app-component*[*OF assms*(1) i] *b-def* **have** *b-def'*:
$b = \varphi\ i(\!|\mathit{ArrMap}|\!)(\!|a|\!)$
by *simp*
from *assms*(1) **have** $b \in_\circ \mathfrak{A}\ i(\!|\mathit{Arr}|\!)$
unfolding *b-def'* **by** (*auto intro*: *dg-cs-intros*)
with i **show** $b \in_\circ (\bigcup_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|\mathit{Arr}|\!))$ **by** *force*
qed

Digraph product universal property digraph homomorphism is a digraph homomorphism

lemma (**in** *pdigraph*) *pdg-dghm-up-is-dghm*:
assumes *digraph* $\alpha\ \mathfrak{C}$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$
shows $\mathit{dghm\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi : \mathfrak{C} \mapsto\mapsto_{DG\alpha} (\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$

proof–
interpret $\mathfrak{C}$: *digraph* α $\mathfrak{C}$ **by** (*rule assms*(1))
show *?thesis*
proof(*intro is-dghmI*, *unfold dghm-up-components*(3,4))
show *vfsequence* (*dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ) **unfolding** *dghm-up-def* **by** *simp*
show *vcard* (*dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ) = $4_{\mathbb{N}}$
unfolding *dghm-up-def* **by** (*simp add*: *nat-omega-simps*)
from *assms*(2) **show** $\mathcal{R}_{\circ}$ (*dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ObjMap*|)) $\subseteq_{\circ}$ ($\prod_{DG} i \in_{\circ} I.$ $\mathfrak{A}$ *i*)(|*Obj*|)
by (*intro dghm-up-ObjMap-vrange*) *blast*
fix *f a b* **assume** *prems*: $f : a \mapsto_{\mathfrak{C}} b$
then have *f*: $f \in_{\circ} \mathfrak{C}$(|*Arr*|) **and** *a*: $a \in_{\circ} \mathfrak{C}$(|*Obj*|) **and** *b*: $b \in_{\circ} \mathfrak{C}$(|*Obj*|) **by** *auto*
show *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ArrMap*|)(|*f*|) :
dghm-up I $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ObjMap*|)(|*a*|) $\mapsto_{\prod_{DG} i \in_{\circ} I.\ \mathfrak{A}\ i}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ObjMap*|)(|*b*|)
proof(*rule dg-prod-is-arrI*)
fix *i* **assume** *prems'*: $i \in_{\circ} I$
interpret φ: *is-dghm* α $\mathfrak{C}$ ‹$\mathfrak{A}$ *i*› ‹φ *i*› **by** (*rule assms*(2)[*OF prems'*])
from φ.*is-dghm-axioms* $\mathfrak{C}$.*digraph-axioms prems pdigraph-axioms prems prems'*
show *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ArrMap*|)(|*f*|)(|*i*|) :
dghm-up I $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ObjMap*|)(|*a*|)(|*i*|) $\mapsto_{\mathfrak{A}\ i}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ObjMap*|)(|*b*|)(|*i*|)
by (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)
qed (*simp-all add*: *f a b dghm-up-ArrMap-app dghm-up-ObjMap-app*)
qed (*auto simp*: *dghm-up-components pdg-digraph-dg-prod dg-cs-intros*)
qed

Further properties

lemma (**in** *pdigraph*) *pdg-dghm-comp-dghm-proj-dghm-up*:
assumes *digraph* α $\mathfrak{C}$
and $\bigwedge i.\ i \in_{\circ} I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$
and $i \in_{\circ} I$
shows $\varphi\ i = \pi_{DG}\ I\ \mathfrak{A}\ i \circ_{DGHM}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ
proof(*rule dghm-eqI*[*of* α $\mathfrak{C}$ ‹$\mathfrak{A}$ *i*› - $\mathfrak{C}$ ‹$\mathfrak{A}$ *i*›])

interpret φ: *is-dghm* α $\mathfrak{C}$ ‹$\mathfrak{A}$ *i*› ‹φ *i*› **by** (*rule assms*(2)[*OF assms*(3)])

show $\varphi\ i : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{A}\ i$ **by** (*auto intro*: *dg-cs-intros*)

from *assms*(1,2) **have** *dghm-up*: *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ : $\mathfrak{C} \mapsto\mapsto_{DG\alpha}$ ($\prod_{DG} i \in_{\circ} I.$ $\mathfrak{A}$ *i*)
by (*simp add*: *pdg-dghm-up-is-dghm*)
note *dghm-proj* = *pdg-dghm-proj-is-dghm*[*OF assms*(3)]

from *assms*(3) *pdg-dghm-proj-is-dghm* **show**
$\pi_{DG}\ I\ \mathfrak{A}\ i \circ_{DGHM}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ : $\mathfrak{C} \mapsto\mapsto_{DG\alpha}$ $\mathfrak{A}$ *i*
by (*intro dghm-comp-is-dghm*[*of* α ‹($\prod_{DG} i \in_{\circ} I.$ $\mathfrak{A}$ *i*)›])
(*auto simp*: *assms dghm-up*)

show $\varphi\ i$(|*ObjMap*|) = ($\pi_{DG}\ I\ \mathfrak{A}\ i \circ_{DGHM}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ)(|*ObjMap*|)
proof(*rule vsv-eqI*)
show *vsv* (($\pi_{DG}\ I\ \mathfrak{A}\ i \circ_{DGHM}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ)(|*ObjMap*|))
unfolding *dghm-comp-components dghm-proj-components dghm-up-components*
by (*rule vsv-vcomp*) *simp-all*
from
dghm-up-ObjMap-vrange[
OF assms(2), *simplified*, *unfolded dg-prod-components*
]
have *rd*: $\mathcal{R}_{\circ}$ (*dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ(|*ObjMap*|)) $\subseteq_{\circ}$ $\mathcal{D}_{\circ}$ ($\pi_{DG}\ I\ \mathfrak{A}\ i$(|*ObjMap*|))
by (*simp add*: *dg-prod-components dg-cs-simps*)
show $\mathcal{D}_{\circ}$ ($\varphi\ i$(|*ObjMap*|)) = $\mathcal{D}_{\circ}$ (($\pi_{DG}\ I\ \mathfrak{A}\ i \circ_{DGHM}$ *dghm-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ)(|*ObjMap*|))

```
    unfolding dghm-comp-components vdomain-vcomp-vsubset[OF rd]
    by (simp add: dg-cs-simps)
  fix a assume a ∈ₒ 𝒟ₒ (φ i(|ObjMap|))
  then have a: a ∈ₒ 𝔈(|Obj|) by (simp add: dg-cs-simps)
  with dghm-up dghm-proj assms(3) show
    φ i(|ObjMap|)(|a|) = (π_DG I 𝔄 i ∘_DGHM dghm-up I 𝔄 𝔈 φ)(|ObjMap|)(|a|)
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
qed auto

show φ i(|ArrMap|) = (π_DG I 𝔄 i ∘_DGHM dghm-up I 𝔄 𝔈 φ)(|ArrMap|)
proof(rule vsv-eqI)
  show vsv ((π_DG I 𝔄 i ∘_DGHM dghm-up I 𝔄 𝔈 φ)(|ArrMap|))
    unfolding dghm-comp-components dghm-proj-components dghm-up-components
    by (rule vsv-vcomp) simp-all
  from
    dghm-up-ArrMap-vrange[
      OF assms(2), simplified, unfolded dg-prod-components
      ]
  have rd: ℛₒ (dghm-up I 𝔄 𝔈 φ(|ArrMap|)) ⊆ₒ 𝒟ₒ (π_DG I 𝔄 i(|ArrMap|))
    by (simp add: dg-prod-components dg-cs-simps)
  show 𝒟ₒ (φ i(|ArrMap|)) = 𝒟ₒ ((π_DG I 𝔄 i ∘_DGHM dghm-up I 𝔄 𝔈 φ)(|ArrMap|))
    unfolding dghm-comp-components vdomain-vcomp-vsubset[OF rd]
    by (simp add: dg-cs-simps)
  fix a assume a ∈ₒ 𝒟ₒ (φ i(|ArrMap|))
  then have a: a ∈ₒ 𝔈(|Arr|) by (simp add: dg-cs-simps)
  with dghm-up dghm-proj assms(3) show
    φ i(|ArrMap|)(|a|) = (π_DG I 𝔄 i ∘_DGHM dghm-up I 𝔄 𝔈 φ)(|ArrMap|)(|a|)
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-cs-intros)
qed auto

qed simp-all

lemma (in pdigraph) pdg-dghm-up-eq-dghm-proj:
  assumes 𝔉 : 𝔈 ↦↦_DGα (∏_DGi∈ₒI. 𝔄 i)
    and ⋀i. i ∈ₒ I ⟹ φ i = π_DG I 𝔄 i ∘_DGHM 𝔉
  shows dghm-up I 𝔄 𝔈 φ = 𝔉
proof(rule dghm-eqI)

  interpret 𝔉: is-dghm α 𝔈 ‹(∏_DGi∈ₒI. 𝔄 i)› 𝔉 by (rule assms(1))

  show dghm-up I 𝔄 𝔈 φ : 𝔈 ↦↦_DGα (∏_DGi∈ₒI. 𝔄 i)
  proof(rule pdg-dghm-up-is-dghm)
    fix i assume prems: i ∈ₒ I
    interpret π: is-dghm α ‹(∏_DGi∈ₒI. 𝔄 i)› ‹𝔄 i› ‹π_DG I 𝔄 i›
      using prems by (rule pdg-dghm-proj-is-dghm)
    show φ i : 𝔈 ↦↦_DGα 𝔄 i
      unfolding assms(2)[OF prems] by (auto intro: dg-cs-intros)
  qed (auto intro: dg-cs-intros)

  show dghm-up I 𝔄 𝔈 φ(|ObjMap|) = 𝔉(|ObjMap|)
  proof(rule vsv-eqI, unfold dghm-up-ObjMap-vdomain)
    fix a assume prems: a ∈ₒ 𝔈(|Obj|)
    show dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|) = 𝔉(|ObjMap|)(|a|)
    proof(rule vsv-eqI, unfold dghm-up-ObjMap-app-vdomain[OF prems])
      fix i assume prems': i ∈ₒ I
      with pdg-dghm-proj-is-dghm[OF prems'] 𝔉.is-dghm-axioms prems show
        dghm-up I 𝔄 𝔈 φ(|ObjMap|)(|a|)(|i|) = 𝔉(|ObjMap|)(|a|)(|i|)
        by
```

```
        (
          cs-concl cs-shallow
            cs-simp: dg-cs-simps assms(2) cs-intro: dg-cs-intros
        )
    qed
      (
        use 𝔉.dghm-ObjMap-app-in-HomCod-Obj prems in
          ‹auto simp: dg-prod-components dghm-up-ObjMap-app›
      )
  qed (auto simp: dghm-up-components dg-cs-simps)

  show dghm-up I 𝔄 ℭ φ(|ArrMap|) = 𝔉(|ArrMap|)
  proof(rule vsv-eqI, unfold dghm-up-ArrMap-vdomain)
    fix a assume prems: a ∈∘ ℭ(|Arr|)
    show dghm-up I 𝔄 ℭ φ(|ArrMap|)(|a|) = 𝔉(|ArrMap|)(|a|)
    proof(rule vsv-eqI, unfold dghm-up-ArrMap-app-vdomain[OF prems])
      fix i assume prems': i ∈∘ I
      with pdg-dghm-proj-is-dghm[OF prems'] 𝔉.is-dghm-axioms prems show
        dghm-up I 𝔄 ℭ φ(|ArrMap|)(|a|)(|i|) = 𝔉(|ArrMap|)(|a|)(|i|)
        by
          (
          cs-concl cs-shallow
            cs-simp: dg-cs-simps assms(2) cs-intro: dg-cs-intros
          )
    qed
      (
        use 𝔉.dghm-ArrMap-app-in-HomCod-Arr prems in
          ‹auto simp: dg-prod-components dghm-up-ArrMap-app›
      )+
  qed (auto simp: dghm-up-components dg-cs-simps)

qed (simp-all add: assms(1))
```

3.8.9 Singleton digraph

Object

```
lemma dg-singleton-ObjI:
  assumes A = set {⟨j, a⟩} and a ∈∘ ℭ(|Obj|)
  shows A ∈∘ (∏DG i∈∘set {j}. ℭ)(|Obj|)
  using assms unfolding dg-prod-components by auto

lemma dg-singleton-ObjE:
  assumes A ∈∘ (∏DG i∈∘set {j}. ℭ)(|Obj|)
  obtains a where A = set {⟨j, a⟩} and a ∈∘ ℭ(|Obj|)
proof–
  from vproductD[OF assms[unfolded dg-prod-components], rule-format]
  have vsv A and [simp]: 𝒟∘ A = set {j} and Aj: A(|j|) ∈∘ ℭ(|Obj|)
    by simp-all
  then interpret A: vsv A by simp
  from A.vsv-is-VLambda have A = set {⟨j, A(|j|)⟩}
    by (auto simp: VLambda-vsingleton)
  with Aj show ?thesis using that by simp
qed
```

Arrow

```
lemma dg-singleton-ArrI:
  assumes F = set {⟨j, a⟩} and a ∈∘ ℭ(|Arr|)
```

shows $F \in_{\circ} (\prod_{DG} j \in_{\circ} set\ \{j\}.\ \mathfrak{C})(\!|Arr|\!)$
using *assms* **unfolding** *dg-prod-components* **by** *auto*

lemma *dg-singleton-ArrE*:
assumes $F \in_{\circ} (\prod_{DG} j \in_{\circ} set\ \{j\}.\ \mathfrak{C})(\!|Arr|\!)$
obtains a **where** $F = set\ \{\langle j, a\rangle\}$ **and** $a \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
proof–
from *vproductD*[*OF assms*[*unfolded dg-prod-components*], *rule-format*]
have *vsv F* **and** [*simp*]: $\mathcal{D}_{\circ}\ F = set\ \{j\}$ **and** *Fj*: $F(\!|j|\!) \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
by *simp-all*
then interpret *F*: *vsv F* **by** *simp*
from *F.vsv-is-VLambda* **have** $F = set\ \{\langle j, F(\!|j|\!)\rangle\}$
by (*auto simp*: *VLambda-vsingleton*)
with *Fj* **show** *?thesis* **using** *that* **by** *simp*
qed

Singleton digraph is a digraph

lemma (**in** *digraph*) *dg-finite-pdigraph-dg-singleton*:
assumes $j \in_{\circ} Vset\ \alpha$
shows *finite-pdigraph* α $(set\ \{j\})$ $(\lambda i.\ \mathfrak{C})$
by (*intro finite-pdigraphI pdigraph-baseI*)
(*auto simp*: *digraph-axioms Limit-vsingleton-in-VsetI assms*)

lemma (**in** *digraph*) *dg-digraph-dg-singleton*:
assumes $j \in_{\circ} Vset\ \alpha$
shows *digraph* α $(\prod_{DG} j \in_{\circ} set\ \{j\}.\ \mathfrak{C})$
proof–
interpret *finite-pdigraph* α ‹*set* $\{j\}$› ‹$\lambda i.\ \mathfrak{C}$›
using *assms* **by** (*rule dg-finite-pdigraph-dg-singleton*)
show *?thesis* **by** (*rule pdg-digraph-dg-prod*)
qed

Arrow with a domain and a codomain

lemma (**in** *digraph*) *dg-singleton-is-arrI*:
assumes $j \in_{\circ} Vset\ \alpha$ **and** $f : a \mapsto_{\mathfrak{C}} b$
shows $set\ \{\langle j, f\rangle\} : set\ \{\langle j, a\rangle\} \mapsto_{(\prod_{DG} j \in_{\circ} set\ \{j\}.\ \mathfrak{C})} set\ \{\langle j, b\rangle\}$
proof–
interpret *finite-pdigraph* α ‹*set* $\{j\}$› ‹$\lambda i.\ \mathfrak{C}$›
by (*rule dg-finite-pdigraph-dg-singleton*[*OF assms*(1)])
from *assms*(2) **show** *?thesis* **by** (*intro dg-prod-is-arrI*) *auto*
qed

lemma (**in** *digraph*) *dg-singleton-is-arrD*:
assumes $set\ \{\langle j, f\rangle\} : set\ \{\langle j, a\rangle\} \mapsto_{(\prod_{DG} j \in_{\circ} set\ \{j\}.\ \mathfrak{C})} set\ \{\langle j, b\rangle\}$
and $j \in_{\circ} Vset\ \alpha$
shows $f : a \mapsto_{\mathfrak{C}} b$
proof–
interpret *finite-pdigraph* α ‹*set* $\{j\}$› ‹$\lambda i.\ \mathfrak{C}$›
by (*rule dg-finite-pdigraph-dg-singleton*[*OF assms*(2)])
from *dg-prod-is-arrD*(7)[*OF assms*(1)] **show** *?thesis* **by** *simp*
qed

lemma (**in** *digraph*) *dg-singleton-is-arrE*:
assumes $set\ \{\langle j, f\rangle\} : set\ \{\langle j, a\rangle\} \mapsto_{(\prod_{DG} j \in_{\circ} set\ \{j\}.\ \mathfrak{C})} set\ \{\langle j, b\rangle\}$
and $j \in_{\circ} Vset\ \alpha$
obtains $f : a \mapsto_{\mathfrak{C}} b$

using *assms dg-singleton-is-arrD* **by** *auto*

3.8.10 Singleton digraph homomorphism

definition *dghm-singleton* :: $V \Rightarrow V \Rightarrow V$
where *dghm-singleton j* $\mathfrak{C}$ =
[
($\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).$ *set* $\{\langle j, a\rangle\}$),
($\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).$ *set* $\{\langle j, f\rangle\}$),
$\mathfrak{C}$,
($\prod_{DG} j \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
]$_\circ$

Components.

lemma *dghm-singleton-components*:
shows *dghm-singleton j* $\mathfrak{C}(\!|ObjMap|\!)$ = ($\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!).$ *set* $\{\langle j, a\rangle\}$)
and *dghm-singleton j* $\mathfrak{C}(\!|ArrMap|\!)$ = ($\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).$ *set* $\{\langle j, f\rangle\}$)
and *dghm-singleton j* $\mathfrak{C}(\!|HomDom|\!)$ = $\mathfrak{C}$
and *dghm-singleton j* $\mathfrak{C}(\!|HomCod|\!)$ = ($\prod_{DG} j \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
unfolding *dghm-singleton-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Object map

mk-VLambda *dghm-singleton-components*(1)
|*vsv dghm-singleton-ObjMap-vsv*[*dg-cs-intros*]|
|*vdomain dghm-singleton-ObjMap-vdomain*[*dg-cs-simps*]|
|*app dghm-singleton-ObjMap-app*[*dg-prod-cs-simps*]|

lemma *dghm-singleton-ObjMap-vrange*[*dg-cs-simps*]:
$\mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ObjMap|\!)$) = ($\prod_{DG} j \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)$(\!|Obj|\!)$
proof(*intro vsubset-antisym vsubsetI*)
fix *A* **assume** *A* $\in_\circ \mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ObjMap|\!)$)
then obtain *a* **where** *A-def*: *A* = *set* $\{\langle j, a\rangle\}$ **and** *a*: *a* $\in_\circ \mathfrak{C}(\!|Obj|\!)$
unfolding *dghm-singleton-components* **by** *auto*
then show *A* $\in_\circ$ ($\prod_{DG} j \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)$(\!|Obj|\!)$
unfolding *dg-prod-components* **by** *auto*
next
fix *A* **assume** *A* $\in_\circ$ ($\prod_{DG} j \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)$(\!|Obj|\!)$
from *vproductD*[*OF this*[*unfolded dg-prod-components*], *rule-format*]
have *vsv A*
and [*simp*]: $\mathcal{D}_\circ$ *A* = *set* $\{j\}$
and *Ai*: $\bigwedge i.$ *i* $\in_\circ$ *set* $\{j\} \Longrightarrow A(\!|i|\!) \in_\circ \mathfrak{C}(\!|Obj|\!)$
by *auto*
then interpret *A*: *vsv A* **by** *simp*
from *Ai* **have** $A(\!|j|\!) \in_\circ \mathfrak{C}(\!|Obj|\!)$ **using** *Ai* **by** *auto*
moreover with *A.vsv-is-VLambda* **have** *A* = ($\lambda f \in_\circ \mathfrak{C}(\!|Obj|\!).$ *set* $\{\langle j, f\rangle\}$)$(\!|A(\!|j|\!)|\!)$
by (*simp add*: *VLambda-vsingleton*)
ultimately show *A* $\in_\circ \mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ObjMap|\!)$)
unfolding *dghm-singleton-components*
by
(
metis
dghm-singleton-ObjMap-vdomain
dghm-singleton-ObjMap-vsv
dghm-singleton-components(1)
vsv.vsv-vimageI2
)

qed

Arrow map

mk-VLambda *dghm-singleton-components*(2)
|*vsv dghm-singleton-ArrMap-vsv*[*dg-cs-intros*]|
|*vdomain dghm-singleton-ArrMap-vdomain*[*dg-cs-simps*]|
|*app dghm-singleton-ArrMap-app*[*dg-prod-cs-simps*]|

lemma *dghm-singleton-ArrMap-vrange*[*dg-cs-simps*]:
$\mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ArrMap|\!)$) = $(\prod_{DG} j \in_\circ set\ \{j\}.\ \mathfrak{C})(\!|Arr|\!)$
proof(*intro vsubset-antisym vsubsetI*)
fix *F* **assume** $F \in_\circ \mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ArrMap|\!)$)
then obtain *f* **where** $F = set\ \{\langle j, f\rangle\}$ **and** $f \in_\circ \mathfrak{C}(\!|Arr|\!)$
unfolding *dghm-singleton-components* **by** *auto*
then show $F \in_\circ (\prod_{DG} j \in_\circ set\ \{j\}.\ \mathfrak{C})(\!|Arr|\!)$
unfolding *dg-prod-components* **by** *auto*
next
fix *F* **assume** $F \in_\circ (\prod_{DG} j \in_\circ set\ \{j\}.\ \mathfrak{C})(\!|Arr|\!)$
from *vproductD*[*OF this*[*unfolded dg-prod-components*], *rule-format*]
have *vsv F*
and [*simp*]: $\mathcal{D}_\circ\ F = set\ \{j\}$
and *Fi*: $\bigwedge i.\ i \in_\circ set\ \{j\} \Longrightarrow F(\!|i|\!) \in_\circ \mathfrak{C}(\!|Arr|\!)$
by *auto*
then interpret *F*: *vsv F* **by** *simp*
from *Fi* **have** $F(\!|j|\!) \in_\circ \mathfrak{C}(\!|Arr|\!)$ **using** *Fi* **by** *auto*
moreover with *F.vsv-is-VLambda* **have** $F = (\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!).\ set\ \{\langle j, f\rangle\})(\!|F(\!|j|\!)|\!)$
by (*simp add*: *VLambda-vsingleton*)
ultimately show $F \in_\circ \mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ArrMap|\!)$)
unfolding *dghm-singleton-components*
by
(
metis
dghm-singleton-ArrMap-vdomain
dghm-singleton-ArrMap-vsv
dghm-singleton-components(2)
vsv.vsv-vimageI2
)
qed

Singleton digraph homomorphism is an isomorphism of digraphs

lemma (**in** *digraph*) *dg-dghm-singleton-is-dghm*:
assumes $j \in_\circ Vset\ \alpha$
shows *dghm-singleton j* $\mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{DG.iso\alpha} (\prod_{DG} j \in_\circ set\ \{j\}.\ \mathfrak{C})$
proof–
interpret *finite-pdigraph* α ‹*set* $\{j\}$› ‹$\lambda i.\ \mathfrak{C}$›
by (*rule dg-finite-pdigraph-dg-singleton*[*OF assms*])
show *?thesis*
proof(*intro is-iso-dghmI is-dghmI*)
show *vfsequence* (*dghm-singleton j* $\mathfrak{C}$) **unfolding** *dghm-singleton-def* **by** *simp*
show *vcard* (*dghm-singleton j* $\mathfrak{C}$) = $4_{\mathbb{N}}$
unfolding *dghm-singleton-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*dghm-singleton j* $\mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_\circ$ $(\prod_{DG} j \in_\circ set\ \{j\}.\ \mathfrak{C})(\!|Obj|\!)$
by (*simp add*: *dg-cs-simps*)
show *dghm-singleton j* $\mathfrak{C}(\!|ArrMap|\!)(\!|f|\!)$:
dghm-singleton j $\mathfrak{C}(\!|ObjMap|\!)(\!|a|\!) \mapsto_{\prod_{DG} j \in_\circ set\ \{j\}.\ \mathfrak{C}}$
dghm-singleton j $\mathfrak{C}(\!|ObjMap|\!)(\!|b|\!)$

```
    if f : a ↦_𝔈 b for f a b
    using that
  proof(intro dg-prod-is-arrI)
    fix k assume k ∈∘ set {j}
    then have k-def: k = j by simp
    from that show dghm-singleton j 𝔈(|ArrMap|)(|f|)(|k|) :
      dghm-singleton j 𝔈(|ObjMap|)(|a|)(|k|) ↦_𝔈 dghm-singleton j 𝔈(|ObjMap|)(|b|)(|k|)
      by
        (
          cs-concl
            cs-simp: k-def V-cs-simps dg-cs-simps dg-prod-cs-simps
            cs-intro: dg-cs-intros
        )
  qed
    (
      cs-concl
        cs-simp: V-cs-simps dg-prod-cs-simps
        cs-intro: V-cs-intros dg-cs-intros
    )+
  show ℛ∘ (dghm-singleton j 𝔈(|ObjMap|)) = (∏_DG j∈∘set {j}. 𝔈)(|Obj|)
    by (simp add: dg-cs-simps)
  show ℛ∘ (dghm-singleton j 𝔈(|ArrMap|)) = (∏_DG j∈∘set {j}. 𝔈)(|Arr|)
    by (simp add: dg-cs-simps)
qed
  (
    auto simp:
      dg-cs-intros
      dg-digraph-dg-singleton[OF assms]
      dghm-singleton-components
  )
qed
```

3.8.11 Product of two digraphs

Definition and elementary properties

See Chapter II-3 in [39].

```
definition dg-prod-2 :: V ⇒ V ⇒ V (infixr ‹×_DG› 80)
  where 𝔄 ×_DG 𝔅 ≡ dg-prod (2_ℕ) (if2 𝔄 𝔅)
```

Product of two digraphs is a digraph

```
context
  fixes α 𝔄 𝔅
  assumes 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅
begin

interpretation 𝒵 α by (rule digraphD[OF 𝔄(1)])
interpretation 𝔄: digraph α 𝔄 by (rule 𝔄)
interpretation 𝔅: digraph α 𝔅 by (rule 𝔅)

lemma finite-pdigraph-dg-prod-2: finite-pdigraph α (2_ℕ) (if2 𝔄 𝔅)
proof(intro finite-pdigraphI pdigraph-baseI)
  from Axiom-of-Infinity show z1-in-Vset: 2_ℕ ∈∘ Vset α by blast
  show digraph α (i = 0 ? 𝔄 : 𝔅) if i ∈∘ 2_ℕ for i
    by (auto intro: dg-cs-intros)
qed auto
```

interpretation *finite-pdigraph* α ‹$2_{\mathbb{N}}$› ‹*if*2 $\mathfrak{A}$ $\mathfrak{B}$›
by (*intro finite-pdigraph-dg-prod*-2 $\mathfrak{A}$ $\mathfrak{B}$)

lemma *digraph-dg-prod*-2[*dg-cs-intros*]: *digraph* α ($\mathfrak{A}$ $\times_{DG}$ $\mathfrak{B}$)
proof–
show *?thesis* **unfolding** *dg-prod*-2-*def* **by** (*rule pdg-digraph-dg-prod*)
qed

end

Object

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

lemma *dg-prod*-2-*ObjI*:
assumes a $\in_{\circ}$ $\mathfrak{A}(\!|Obj|\!)$ **and** b $\in_{\circ}$ $\mathfrak{B}(\!|Obj|\!)$
shows $[a, b]_{\circ}$ $\in_{\circ}$ ($\mathfrak{A}$ $\times_{DG}$ $\mathfrak{B}$)$(\!|Obj|\!)$
unfolding *dg-prod*-2-*def dg-prod-components*
proof(*intro vproductI ballI*)
show $\mathcal{D}_{\circ}$ $[a, b]_{\circ}$ = $2_{\mathbb{N}}$ **by** (*simp add*: *nat-omega-simps two*)
fix i **assume** i $\in_{\circ}$ $2_{\mathbb{N}}$
then consider ‹i = 0› | ‹i = $1_{\mathbb{N}}$› **unfolding** *two* **by** *auto*
then show $[a, b]_{\circ}(\!|i|\!)$ $\in_{\circ}$ (*if* i = 0 *then* $\mathfrak{A}$ *else* $\mathfrak{B}$)$(\!|Obj|\!)$
by *cases* (*simp-all add*: *nat-omega-simps assms*(1,2))
qed *auto*

lemma *dg-prod*-2-*ObjI'*[*dg-prod-cs-intros*]:
assumes ab = $[a, b]_{\circ}$ **and** a $\in_{\circ}$ $\mathfrak{A}(\!|Obj|\!)$ **and** b $\in_{\circ}$ $\mathfrak{B}(\!|Obj|\!)$
shows ab $\in_{\circ}$ ($\mathfrak{A}$ $\times_{DG}$ $\mathfrak{B}$)$(\!|Obj|\!)$
using *assms*(2,3) **unfolding** *assms*(1) **by** (*rule dg-prod*-2-*ObjI*)

lemma *dg-prod*-2-*ObjE*:
assumes ab $\in_{\circ}$ ($\mathfrak{A}$ $\times_{DG}$ $\mathfrak{B}$)$(\!|Obj|\!)$
obtains a b **where** ab = $[a, b]_{\circ}$ **and** a $\in_{\circ}$ $\mathfrak{A}(\!|Obj|\!)$ **and** b $\in_{\circ}$ $\mathfrak{B}(\!|Obj|\!)$
proof–
from *vproductD*[*OF assms*[*unfolded dg-prod*-2-*def dg-prod-components*]]
have *vsv-ab*: *vsv ab*
and *dom-ab*: $\mathcal{D}_{\circ}$ ab = $2_{\mathbb{N}}$
and *ab-app*: $\bigwedge i$. i $\in_{\circ}$ $2_{\mathbb{N}}$ $\Longrightarrow$ $ab(\!|i|\!)$ $\in_{\circ}$ (*if* i = 0 *then* $\mathfrak{A}$ *else* $\mathfrak{B}$)$(\!|Obj|\!)$
by *auto*
have *dom-ab*[*simp*]: $\mathcal{D}_{\circ}$ ab = $2_{\mathbb{N}}$
unfolding *dom-ab* **by** (*simp add*: *nat-omega-simps two*)
interpret *vsv ab* **by** (*rule vsv-ab*)
have ab = [*vpfst ab*, *vpsnd ab*]$_{\circ}$
by (*rule vsv-vfsequence-two*[*symmetric*]) *auto*
moreover from *ab-app*[*of* 0] **have** *vpfst ab* $\in_{\circ}$ $\mathfrak{A}(\!|Obj|\!)$ **by** *simp*
moreover from *ab-app*[*of* ‹$1_{\mathbb{N}}$›] **have** *vpsnd ab* $\in_{\circ}$ $\mathfrak{B}(\!|Obj|\!)$ **by** *simp*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

end

Arrow

context

fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

lemma *dg-prod-2-ArrI*:
assumes $g \in_\circ \mathfrak{A}(\!|Arr|\!)$ **and** $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
shows $[g, f]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
unfolding *dg-prod-2-def dg-prod-components*
proof(*intro vproductI ballI*)
show $\mathcal{D}_\circ\ [g, f]_\circ = 2_{\mathbb{N}}$ **by** (*simp add*: *nat-omega-simps two*)
fix i **assume** $i \in_\circ 2_{\mathbb{N}}$
then consider $\langle i = 0 \rangle \mid \langle i = 1_{\mathbb{N}} \rangle$ **unfolding** *two* **by** *auto*
then show $[g, f]_\circ(\!|i|\!) \in_\circ (\textit{if } i = 0 \textit{ then } \mathfrak{A} \textit{ else } \mathfrak{B})(\!|Arr|\!)$
by *cases* (*simp-all add*: *nat-omega-simps assms*(1,2))
qed *auto*

lemma *dg-prod-2-ArrI'*[*dg-prod-cs-intros*]:
assumes $gf = [g, f]_\circ$ **and** $g \in_\circ \mathfrak{A}(\!|Arr|\!)$ **and** $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
shows $[g, f]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
using *assms*(2,3) **unfolding** *assms*(1) **by** (*rule dg-prod-2-ArrI*)

lemma *dg-prod-2-ArrE*:
assumes $gf \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
obtains $g\ f$ **where** $gf = [g, f]_\circ$ **and** $g \in_\circ \mathfrak{A}(\!|Arr|\!)$ **and** $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
proof–
from *vproductD*[*OF assms*[*unfolded dg-prod-2-def dg-prod-components*]]
have *vsv-gf*: *vsv gf*
and *dom-gf*: $\mathcal{D}_\circ\ gf = 2_{\mathbb{N}}$
and *gf-app*: $\bigwedge i.\ i \in_\circ 2_{\mathbb{N}} \Longrightarrow gf(\!|i|\!) \in_\circ (\textit{if } i = 0 \textit{ then } \mathfrak{A} \textit{ else } \mathfrak{B})(\!|Arr|\!)$
by *auto*
have *dom-gf*[*simp*]: $\mathcal{D}_\circ\ gf = 2_{\mathbb{N}}$ **unfolding** *dom-gf* **by** (*simp add*: *two*)
interpret *vsv gf* **by** (*rule vsv-gf*)
have $gf = [\textit{vpfst } gf, \textit{vpsnd } gf]_\circ$
by (*rule vsv-vfsequence-two*[*symmetric*]) *auto*
moreover from *gf-app*[*of* 0] **have** *vpfst gf* $\in_\circ \mathfrak{A}(\!|Arr|\!)$ **by** *simp*
moreover from *gf-app*[*of* $\langle 1_{\mathbb{N}} \rangle$] **have** *vpsnd gf* $\in_\circ \mathfrak{B}(\!|Arr|\!)$ **by** *simp*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

end

Arrow with a domain and a codomain

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule digraphD*[*OF* $\mathfrak{A}(1)$])
interpretation $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *digraph* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation *finite-pdigraph* α $\langle 2_{\mathbb{N}} \rangle$ $\langle \textit{if2 } \mathfrak{A}\ \mathfrak{B} \rangle$
by (*intro finite-pdigraph-dg-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *dg-prod-2-is-arrI*:
assumes $g : a \mapsto_{\mathfrak{A}} c$ **and** $f : b \mapsto_{\mathfrak{B}} d$
shows $[g, f]_\circ : [a, b]_\circ \mapsto_{\mathfrak{A} \times_{DG} \mathfrak{B}} [c, d]_\circ$
unfolding *dg-prod-2-def*

proof(*rule dg-prod-is-arrI*)
show $[g,\ f]_\circ(\!|i|\!) : [a,\ b]_\circ(\!|i|\!) \mapsto_{\textit{if } i = 0 \textit{ then } \mathfrak{A} \textit{ else } \mathfrak{B}} [c,\ d]_\circ(\!|i|\!)$
if $i \in_\circ 2_{\mathbb{N}}$ **for** i
proof–
from *that* **consider** $\langle i = 0 \rangle \mid \langle i = 1_{\mathbb{N}} \rangle$ **unfolding** *two* **by** *auto*
then show $[g,\ f]_\circ(\!|i|\!) : [a,\ b]_\circ(\!|i|\!) \mapsto_{\textit{if } i = 0 \textit{ then } \mathfrak{A} \textit{ else } \mathfrak{B}} [c,\ d]_\circ(\!|i|\!)$
by *cases* (*simp-all add*: *nat-omega-simps assms*)
qed
qed (*auto simp*: *nat-omega-simps two*)

lemma *dg-prod-2-is-arrI'*[*dg-prod-cs-intros*]:
assumes $gf = [g,\ f]_\circ$
and $ab = [a,\ b]_\circ$
and $cd = [c,\ d]_\circ$
and $g : a \mapsto_{\mathfrak{A}} c$
and $f : b \mapsto_{\mathfrak{B}} d$
shows $gf : ab \mapsto_{\mathfrak{A} \times_{DG} \mathfrak{B}} cd$
using *assms*(4,5) **unfolding** *assms*(1,2,3) **by** (*rule dg-prod-2-is-arrI*)

lemma *dg-prod-2-is-arrE*:
assumes $gf : ab \mapsto_{\mathfrak{A} \times_{DG} \mathfrak{B}} cd$
obtains *g f a b c d*
where $gf = [g,\ f]_\circ$
and $ab = [a,\ b]_\circ$
and $cd = [c,\ d]_\circ$
and $g : a \mapsto_{\mathfrak{A}} c$
and $f : b \mapsto_{\mathfrak{B}} d$
proof–
from *dg-prod-is-arrD*[*OF assms*[*unfolded dg-prod-2-def*]]
have [*simp*]: *vsv gf* $\mathcal{D}_\circ$ *gf* $= 2_{\mathbb{N}}$ *vsv ab* $\mathcal{D}_\circ$ *ab* $= 2_{\mathbb{N}}$ *vsv cd* $\mathcal{D}_\circ$ *cd* $= 2_{\mathbb{N}}$
and *gf-app*:
$\bigwedge i.\ i \in_\circ 2_{\mathbb{N}} \Longrightarrow gf(\!|i|\!) : ab(\!|i|\!) \mapsto_{\textit{if } i = 0 \textit{ then } \mathfrak{A} \textit{ else } \mathfrak{B}} cd(\!|i|\!)$
by (*auto simp*: *two*)
have $gf = [vpfst\ gf,\ vpsnd\ gf]_\circ$ **by** (*simp add*: *vsv-vfsequence-two*)
moreover have $ab = [vpfst\ ab,\ vpsnd\ ab]_\circ$ **by** (*simp add*: *vsv-vfsequence-two*)
moreover have $cd = [vpfst\ cd,\ vpsnd\ cd]_\circ$ **by** (*simp add*: *vsv-vfsequence-two*)
moreover from *gf-app*[*of* 0] **have** *vpfst gf* : *vpfst ab* $\mapsto_{\mathfrak{A}}$ *vpfst cd* **by** *simp*
moreover from *gf-app*[*of* $\langle 1_{\mathbb{N}} \rangle$] **have** *vpsnd gf* : *vpsnd ab* $\mapsto_{\mathfrak{B}}$ *vpsnd cd*
by (*simp add*: *nat-omega-simps*)
ultimately show *?thesis* **using** *that* **by** *auto*
qed

end

Domain

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

lemma *dg-prod-2-Dom-vsv*: *vsv* $((\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!))$
unfolding *dg-prod-2-def dg-prod-components* **by** *simp*

lemma *dg-prod-2-Dom-vdomain*[*dg-cs-simps*]:
$\mathcal{D}_\circ\ ((\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!)) = (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
unfolding *dg-prod-2-def dg-prod-components* **by** *simp*

lemma *dg-prod-2-Dom-app*[*dg-prod-cs-simps*]:
assumes $[g, f]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
shows $(\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!)(\!|g, f|\!)_\bullet = [\mathfrak{A}(\!|Dom|\!)(\!|g|\!), \mathfrak{B}(\!|Dom|\!)(\!|f|\!)]_\circ$
proof–
from *assms* **obtain** *ab cd* **where** *gf*: $[g, f]_\circ : ab \mapsto_{\mathfrak{A} \times_{DG} \mathfrak{B}} cd$
by (*auto intro*: *is-arrI*)
then have *Dom-gf*: $(\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!)(\!|g, f|\!)_\bullet = ab$
by (*simp add*: *dg-cs-simps*)
from *gf* **obtain** *a b c d*
where *ab-def*: $ab = [a, b]_\circ$
and $cd = [c, d]_\circ$
and $g : a \mapsto_{\mathfrak{A}} c$
and $f : b \mapsto_{\mathfrak{B}} d$
by (*elim dg-prod-2-is-arrE*[*OF* $\mathfrak{A}$ $\mathfrak{B}$]) *simp*
then have *Dom-g*: $\mathfrak{A}(\!|Dom|\!)(\!|g|\!) = a$ **and** *Dom-f*: $\mathfrak{B}(\!|Dom|\!)(\!|f|\!) = b$
by (*simp-all add*: *dg-cs-simps*)
show *?thesis* **unfolding** *Dom-gf ab-def Dom-g Dom-f* **..**
qed

lemma *dg-prod-2-Dom-vrange*: $\mathcal{R}_\circ\ ((\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!)) \subseteq_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold dg-cs-simps*)
show *vsv* $((\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!))$ **by** (*rule dg-prod-2-Dom-vsv*)
fix *gf* **assume** *prems*: $gf \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
then obtain *g f* **where** *gf-def*: $gf = [g, f]_\circ$
and *g*: $g \in_\circ \mathfrak{A}(\!|Arr|\!)$
and *f*: $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
by (*elim dg-prod-2-ArrE*[*OF* $\mathfrak{A}$ $\mathfrak{B}$]) *simp*
from *g f* **obtain** *a b c d* **where** *g*: $g : a \mapsto_{\mathfrak{A}} c$ **and** *f*: $f : b \mapsto_{\mathfrak{B}} d$
by (*auto intro*!: *is-arrI*)
from $\mathfrak{A}$ $\mathfrak{B}$ *g f* **show** $(\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Dom|\!)(\!|gf|\!) \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Obj|\!)$
unfolding *gf-def dg-prod-2-Dom-app*[*OF prems*[*unfolded gf-def*]]
by (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-prod-cs-intros*)
qed

end

Codomain

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

lemma *dg-prod-2-Cod-vsv*: *vsv* $((\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Cod|\!))$
unfolding *dg-prod-2-def dg-prod-components* **by** *simp*

lemma *dg-prod-2-Cod-vdomain*[*dg-cs-simps*]:
$\mathcal{D}_\circ\ ((\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Cod|\!)) = (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
unfolding *dg-prod-2-def dg-prod-components* **by** *simp*

lemma *dg-prod-2-Cod-app*[*dg-prod-cs-simps*]:
assumes $[g, f]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
shows $(\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Cod|\!)(\!|g, f|\!)_\bullet = [\mathfrak{A}(\!|Cod|\!)(\!|g|\!), \mathfrak{B}(\!|Cod|\!)(\!|f|\!)]_\circ$
proof–
from *assms* **obtain** *ab cd* **where** *gf*: $[g, f]_\circ : ab \mapsto_{\mathfrak{A} \times_{DG} \mathfrak{B}} cd$
by (*auto intro*: *is-arrI*)
then have *Cod-gf*: $(\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Cod|\!)(\!|g, f|\!)_\bullet = cd$
by (*simp add*: *dg-cs-simps*)

```
  from gf obtain a b c d
    where ab = [a, b]∘
      and cd-def: cd = [c, d]∘
      and g : a ↦𝔄 c
      and f : b ↦𝔅 d
    by (elim dg-prod-2-is-arrE[OF 𝔄 𝔅]) simp
  then have Cod-g: 𝔄(|Cod|)(|g|) = c and Cod-f: 𝔅(|Cod|)(|f|) = d
    by (simp-all add: dg-cs-simps)
  show ?thesis unfolding Cod-gf cd-def Cod-g Cod-f ..
qed

lemma dg-prod-2-Cod-vrange: ℛ∘ ((𝔄 ×DG 𝔅)(|Cod|)) ⊆∘ (𝔄 ×DG 𝔅)(|Obj|)
proof(rule vsv.vsv-vrange-vsubset, unfold dg-cs-simps)
  show vsv ((𝔄 ×DG 𝔅)(|Cod|)) by (rule dg-prod-2-Cod-vsv)
  fix gf assume prems: gf ∈∘ (𝔄 ×DG 𝔅)(|Arr|)
  then obtain g f where gf-def: gf = [g, f]∘
    and g: g ∈∘ 𝔄(|Arr|)
    and f: f ∈∘ 𝔅(|Arr|)
    by (elim dg-prod-2-ArrE[OF 𝔄 𝔅]) simp
  from g f obtain a b c d where g: g : a ↦𝔄 c and f: f : b ↦𝔅 d
    by (auto intro!: is-arrI)
  from 𝔄 𝔅 g f show (𝔄 ×DG 𝔅)(|Cod|)(|gf|) ∈∘ (𝔄 ×DG 𝔅)(|Obj|)
    unfolding gf-def dg-prod-2-Cod-app[OF prems[unfolded gf-def]]
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-cs-simps cs-intro: dg-cs-intros dg-prod-cs-intros
      )
qed

end
```

Opposite product digraph

```
context
  fixes α 𝔄 𝔅
  assumes 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅
begin

interpretation 𝔄: digraph α 𝔄 by (rule 𝔄)
interpretation 𝔅: digraph α 𝔅 by (rule 𝔅)

lemma dg-prod-2-op-dg-dg-Obj[dg-op-simps]:
  (op-dg 𝔄 ×DG 𝔅)(|Obj|) = (𝔄 ×DG 𝔅)(|Obj|)
proof
  (
    intro vsubset-antisym vsubsetI;
    elim dg-prod-2-ObjE[OF 𝔄.digraph-op 𝔅] dg-prod-2-ObjE[OF 𝔄 𝔅],
    (unfold dg-op-simps)?
  )
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Obj|) b ∈∘ 𝔅(|Obj|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (𝔄 ×DG 𝔅)(|Obj|)
    unfolding prems(1) dg-op-simps
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros)
next
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Obj|) b ∈∘ 𝔅(|Obj|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (op-dg 𝔄 ×DG 𝔅)(|Obj|)
    unfolding prems(1) dg-op-simps
```

```
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-cs-simps cs-intro: dg-op-intros dg-prod-cs-intros
      )
qed

lemma dg-prod-2-dg-op-dg-Obj[dg-op-simps]:
  (𝔄 ×_DG op-dg 𝔅)(|Obj|) = (𝔄 ×_DG 𝔅)(|Obj|)
proof
  (
    intro vsubset-antisym vsubsetI;
    elim dg-prod-2-ObjE[OF 𝔄 𝔅.digraph-op] dg-prod-2-ObjE[OF 𝔄 𝔅],
    (unfold dg-op-simps)?
  )
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Obj|) b ∈∘ 𝔅(|Obj|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (𝔄 ×_DG 𝔅)(|Obj|)
    unfolding prems(1) dg-op-simps
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros)
next
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Obj|) b ∈∘ 𝔅(|Obj|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (𝔄 ×_DG op-dg 𝔅)(|Obj|)
    unfolding prems(1) dg-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros dg-op-intros
      )
qed

lemma dg-prod-2-op-dg-dg-Arr[dg-op-simps]:
  (op-dg 𝔄 ×_DG 𝔅)(|Arr|) = (𝔄 ×_DG 𝔅)(|Arr|)
proof
  (
    intro vsubset-antisym vsubsetI;
    elim dg-prod-2-ArrE[OF 𝔄.digraph-op 𝔅] dg-prod-2-ArrE[OF 𝔄 𝔅],
    (unfold dg-op-simps)?
  )
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Arr|) b ∈∘ 𝔅(|Arr|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (𝔄 ×_DG 𝔅)(|Arr|)
    unfolding prems(1) dg-op-simps
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros)
next
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Arr|) b ∈∘ 𝔅(|Arr|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (op-dg 𝔄 ×_DG 𝔅)(|Arr|)
    unfolding prems(1) dg-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros dg-op-intros
      )
qed

lemma dg-prod-2-dg-op-dg-Arr[dg-op-simps]:
  (𝔄 ×_DG op-dg 𝔅)(|Arr|) = (𝔄 ×_DG 𝔅)(|Arr|)
proof
  (
    intro vsubset-antisym vsubsetI;
```

```
    elim dg-prod-2-ArrE[OF 𝔄 𝔅.digraph-op] dg-prod-2-ArrE[OF 𝔄 𝔅],
    (unfold dg-op-simps)?
  )
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Arr|) b ∈∘ 𝔅(|Arr|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (𝔄 ×DG 𝔅)(|Arr|)
    unfolding prems(1) dg-op-simps
    by (cs-concl cs-shallow cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros)
next
  fix ab a b assume prems: ab = [a, b]∘ a ∈∘ 𝔄(|Arr|) b ∈∘ 𝔅(|Arr|)
  from 𝔄 𝔅 prems(2,3) show ab ∈∘ (𝔄 ×DG op-dg 𝔅)(|Arr|)
    unfolding prems(1) dg-op-simps
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-cs-simps cs-intro: dg-prod-cs-intros dg-op-intros
      )
qed

end

context
  fixes α 𝔄 𝔅
  assumes 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅
begin

lemma op-dg-dg-prod-2[dg-op-simps]: op-dg (𝔄 ×DG 𝔅) = op-dg 𝔄 ×DG op-dg 𝔅
proof(rule vsv-eqI)

  show vsv (op-dg (𝔄 ×DG 𝔅)) unfolding op-dg-def by auto
  show vsv (op-dg 𝔄 ×DG op-dg 𝔅) unfolding dg-prod-2-def dg-prod-def by auto
  have dom-lhs: 𝒟∘ (op-dg (𝔄 ×DG 𝔅)) = 4ℕ
    by (simp add: op-dg-def nat-omega-simps)
  show 𝒟∘ (op-dg (𝔄 ×DG 𝔅)) = 𝒟∘ (op-dg 𝔄 ×DG op-dg 𝔅)
    unfolding dom-lhs by (simp add: dg-prod-2-def dg-prod-def nat-omega-simps)

  have Cod-Dom: (𝔄 ×DG 𝔅)(|Cod|) = (op-dg 𝔄 ×DG op-dg 𝔅)(|Dom|)
  proof(rule vsv-eqI)
    from 𝔄 𝔅 show vsv ((𝔄 ×DG 𝔅)(|Cod|)) by (rule dg-prod-2-Cod-vsv)
    from 𝔄 𝔅 show vsv ((op-dg 𝔄 ×DG op-dg 𝔅)(|Dom|))
      by (cs-concl cs-shallow cs-intro: dg-prod-2-Dom-vsv dg-op-intros)+
    from 𝔄 𝔅 have dom-lhs: 𝒟∘ ((𝔄 ×DG 𝔅)(|Cod|)) = (𝔄 ×DG 𝔅)(|Arr|)
      by (cs-concl cs-shallow cs-simp: dg-cs-simps)
    from 𝔄 𝔅 show 𝒟∘ ((𝔄 ×DG 𝔅)(|Cod|)) = 𝒟∘ ((op-dg 𝔄 ×DG op-dg 𝔅)(|Dom|))
      unfolding dom-lhs
      by
        (
          cs-concl cs-shallow
            cs-simp: dg-cs-simps dg-op-simps cs-intro: dg-op-intros
        )
    show (𝔄 ×DG 𝔅)(|Cod|)(|gf|) = (op-dg 𝔄 ×DG op-dg 𝔅)(|Dom|)(|gf|)
      if gf ∈∘ 𝒟∘ ((𝔄 ×DG 𝔅)(|Cod|)) for gf
      using that unfolding dom-lhs
    proof–
      assume gf ∈∘ (𝔄 ×DG 𝔅)(|Arr|)
      then obtain g f
        where gf-def: gf = [g, f]∘
          and g: g ∈∘ 𝔄(|Arr|)
          and f: f ∈∘ 𝔅(|Arr|)
```

```
    by (rule dg-prod-2-ArrE[OF 𝔄 𝔅]) simp
  from 𝔄 𝔅 g f show (𝔄 ×_DG 𝔅)(|Cod|)(|gf|) = (op-dg 𝔄 ×_DG op-dg 𝔅)(|Dom|)(|gf|)
    unfolding gf-def
    by
      (
        cs-concl cs-shallow
          cs-simp: dg-prod-cs-simps dg-op-simps
          cs-intro: dg-prod-cs-intros dg-op-intros
      )
 qed
qed

have Dom-Cod: (𝔄 ×_DG 𝔅)(|Dom|) = (op-dg 𝔄 ×_DG op-dg 𝔅)(|Cod|)
proof(rule vsv-eqI)
 from 𝔄 𝔅 show vsv ((op-dg 𝔄 ×_DG op-dg 𝔅)(|Cod|))
   by (cs-concl cs-shallow cs-intro: dg-prod-2-Cod-vsv dg-op-intros)+
 from 𝔄 𝔅 have dom-lhs: 𝒟∘ ((𝔄 ×_DG 𝔅)(|Dom|)) = (𝔄 ×_DG 𝔅)(|Arr|)
   by (cs-concl cs-shallow cs-simp: dg-cs-simps)
 from 𝔄 𝔅 show 𝒟∘ ((𝔄 ×_DG 𝔅)(|Dom|)) = 𝒟∘ ((op-dg 𝔄 ×_DG op-dg 𝔅)(|Cod|))
   unfolding dom-lhs
   by
     (
       cs-concl cs-shallow
         cs-simp: dg-cs-simps dg-op-simps cs-intro: dg-op-intros
     )
 show (𝔄 ×_DG 𝔅)(|Dom|)(|gf|) = (op-dg 𝔄 ×_DG op-dg 𝔅)(|Cod|)(|gf|)
   if gf ∈∘ 𝒟∘ ((𝔄 ×_DG 𝔅)(|Dom|)) for gf
   using that unfolding dom-lhs
 proof–
   assume gf ∈∘ (𝔄 ×_DG 𝔅)(|Arr|)
   then obtain g f
     where gf-def: gf = [g, f]∘
       and g: g ∈∘ 𝔄(|Arr|)
       and f: f ∈∘ 𝔅(|Arr|)
     by (rule dg-prod-2-ArrE[OF 𝔄 𝔅]) simp
   from 𝔄 𝔅 g f show (𝔄 ×_DG 𝔅)(|Dom|)(|gf|) = (op-dg 𝔄 ×_DG op-dg 𝔅)(|Cod|)(|gf|)
     unfolding gf-def
     by
       (
         cs-concl cs-shallow
           cs-simp: dg-cs-simps dg-prod-cs-simps dg-op-simps
           cs-intro: dg-op-intros dg-prod-cs-intros
       )
 qed
qed (auto intro: 𝔄 𝔅 dg-prod-2-Dom-vsv)

show a ∈∘ 𝒟∘ (op-dg (𝔄 ×_DG 𝔅)) ⟹
 op-dg (𝔄 ×_DG 𝔅)(|a|) = (op-dg 𝔄 ×_DG op-dg 𝔅)(|a|)
 for a
proof
 (
   unfold dom-lhs,
   elim-in-numeral,
   fold dg-field-simps,
   unfold op-dg-components
 )
 from 𝔄 𝔅 show (𝔄 ×_DG 𝔅)(|Obj|) = (op-dg 𝔄 ×_DG op-dg 𝔅)(|Obj|)
   by (cs-concl cs-shallow cs-simp: dg-op-simps cs-intro: dg-op-intros)
```

from $\mathfrak{A}$ $\mathfrak{B}$ **show** $(\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!) = (op\text{-}dg\ \mathfrak{A} \times_{DG} op\text{-}dg\ \mathfrak{B})(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *dg-op-simps* **cs-intro**: *dg-op-intros*)
qed (*auto simp*: $\mathfrak{A}$ $\mathfrak{B}$ *Cod-Dom Dom-Cod*)

qed

end

3.8.12 Projections for the product of two digraphs

Definition and elementary properties

definition *dghm-proj-fst* :: $V \Rightarrow V \Rightarrow V$ ($\langle\pi_{DG.1}\rangle$)
where $\pi_{DG.1}$ $\mathfrak{A}$ $\mathfrak{B}$ = *dghm-proj* ($2_{\mathbb{N}}$) (*if*2 $\mathfrak{A}$ $\mathfrak{B}$) 0
definition *dghm-proj-snd* :: $V \Rightarrow V \Rightarrow V$ ($\langle\pi_{DG.2}\rangle$)
where $\pi_{DG.2}$ $\mathfrak{A}$ $\mathfrak{B}$ = *dghm-proj* ($2_{\mathbb{N}}$) (*if*2 $\mathfrak{A}$ $\mathfrak{B}$) ($1_{\mathbb{N}}$)

Object map for a projection of a product of two digraphs

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

lemma *dghm-proj-fst-ObjMap-app*[*dg-cs-simps*]:
assumes $[a, b]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Obj|\!)$
shows $\pi_{DG.1}$ $\mathfrak{A}$ $\mathfrak{B}(\!|ObjMap|\!)(\!|a, b|\!)_\bullet = a$
proof–
from *assms* **have** $[a, b]_\circ \in_\circ (\prod_\circ i \in_\circ 2_{\mathbb{N}}.\ (if\ i = 0\ then\ \mathfrak{A}\ else\ \mathfrak{B})(\!|Obj|\!))$
unfolding *dg-prod-2-def dg-prod-components* **by** *simp*
then show $\pi_{DG.1}$ $\mathfrak{A}$ $\mathfrak{B}(\!|ObjMap|\!)(\!|a, b|\!)_\bullet = a$
unfolding *dghm-proj-fst-def dghm-proj-components dg-prod-components* **by** *simp*
qed

lemma *dghm-proj-snd-ObjMap-app*[*dg-cs-simps*]:
assumes $[a, b]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Obj|\!)$
shows $\pi_{DG.2}$ $\mathfrak{A}$ $\mathfrak{B}(\!|ObjMap|\!)(\!|a, b|\!)_\bullet = b$
proof–
from *assms* **have** $[a, b]_\circ \in_\circ (\prod_\circ i \in_\circ 2_{\mathbb{N}}.\ (if\ i = 0\ then\ \mathfrak{A}\ else\ \mathfrak{B})(\!|Obj|\!))$
unfolding *dg-prod-2-def dg-prod-components* **by** *simp*
then show $\pi_{DG.2}$ $\mathfrak{A}$ $\mathfrak{B}(\!|ObjMap|\!)(\!|a, b|\!)_\bullet = b$
unfolding *dghm-proj-snd-def dghm-proj-components dg-prod-components*
by (*simp add*: *nat-omega-simps*)
qed

end

Arrow map for a projection of a product of two digraphs

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* α $\mathfrak{B}$
begin

lemma *dghm-proj-fst-ArrMap-app*[*dg-cs-simps*]:
assumes $[g, f]_\circ \in_\circ (\mathfrak{A} \times_{DG} \mathfrak{B})(\!|Arr|\!)$
shows $\pi_{DG.1}$ $\mathfrak{A}$ $\mathfrak{B}(\!|ArrMap|\!)(\!|g, f|\!)_\bullet = g$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ (\prod_\circ i \in_\circ 2_{\mathbb{N}}.\ (if\ i = 0\ then\ \mathfrak{A}\ else\ \mathfrak{B})(\!|Arr|\!))$

```
    unfolding dg-prod-2-def dg-prod-components by simp
  then show π_{DG.1} 𝔄 𝔅(|ArrMap|)(|g, f|)• = g
    unfolding dghm-proj-fst-def dghm-proj-components dg-prod-components by simp
qed

lemma dghm-proj-snd-ArrMap-app[dg-cs-simps]:
  assumes [g, f]∘ ∈∘ (𝔄 ×_{DG} 𝔅)(|Arr|)
  shows π_{DG.2} 𝔄 𝔅(|ArrMap|)(|g, f|)• = f
proof–
  from assms have [g, f]∘ ∈∘ (∏∘ i∈∘2_ℕ. (if i = 0 then 𝔄 else 𝔅)(|Arr|))
    unfolding dg-prod-2-def dg-prod-components by simp
  then show π_{DG.2} 𝔄 𝔅(|ArrMap|)(|g, f|)• = f
    unfolding dghm-proj-snd-def dghm-proj-components dg-prod-components
    by (simp add: nat-omega-simps)
qed

end
```

Domain and codomain of a projection of a product of two digraphs

```
lemma dghm-proj-fst-HomDom: π_{DG.1} 𝔄 𝔅(|HomDom|) = 𝔄 ×_{DG} 𝔅
  unfolding dghm-proj-fst-def dghm-proj-components dg-prod-2-def ..

lemma dghm-proj-fst-HomCod: π_{DG.1} 𝔄 𝔅(|HomCod|) = 𝔄
  unfolding dghm-proj-fst-def dghm-proj-components dg-prod-2-def by simp

lemma dghm-proj-snd-HomDom: π_{DG.2} 𝔄 𝔅(|HomDom|) = 𝔄 ×_{DG} 𝔅
  unfolding dghm-proj-snd-def dghm-proj-components dg-prod-2-def ..

lemma dghm-proj-snd-HomCod: π_{DG.2} 𝔄 𝔅(|HomCod|) = 𝔅
  unfolding dghm-proj-snd-def dghm-proj-components dg-prod-2-def by simp
```

Projection of a product of two digraphs is a digraph homomorphism

```
context
  fixes α 𝔄 𝔅
  assumes 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅
begin

interpretation finite-pdigraph α ‹2_ℕ› ‹if2 𝔄 𝔅›
  by (intro finite-pdigraph-dg-prod-2 𝔄 𝔅)

lemma dghm-proj-fst-is-dghm:
  assumes i ∈∘ I
  shows π_{DG.1} 𝔄 𝔅 : 𝔄 ×_{DG} 𝔅 ↦↦_{DGα} 𝔄
  by
    (
      rule pdg-dghm-proj-is-dghm[
        where i=0, simplified, folded dghm-proj-fst-def dg-prod-2-def
        ]
    )

lemma dghm-proj-fst-is-dghm'[dg-cs-intros]:
  assumes i ∈∘ I and ℭ = 𝔄 ×_{DG} 𝔅 and 𝔇 = 𝔄
  shows π_{DG.1} 𝔄 𝔅 : ℭ ↦↦_{DGα} 𝔇
  using assms(1) unfolding assms(2,3) by (rule dghm-proj-fst-is-dghm)

lemma dghm-proj-snd-is-dghm:
```

assumes $i \in_\circ I$
shows $\pi_{DG.2}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{A} \times_{DG} \mathfrak{B} \mapsto\mapsto_{DG\alpha} \mathfrak{B}$
by
(
rule pdg-dghm-proj-is-dghm[
where i=‹$1_{\mathbb{N}}$›, *simplified*, *folded dghm-proj-snd-def dg-prod-2-def*
]
)

lemma *dghm-proj-snd-is-dghm'*[*dg-cs-intros*]:
assumes $i \in_\circ I$ **and** $\mathfrak{C} = \mathfrak{A} \times_{DG} \mathfrak{B}$ **and** $\mathfrak{D} = \mathfrak{B}$
shows $\pi_{DG.2}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{C} \mapsto\mapsto_{DG\alpha} \mathfrak{D}$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule dghm-proj-snd-is-dghm*)

end

3.8.13 Product of three digraphs

definition *dg-prod-3* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (‹(- $\times_{DG3}$ - $\times_{DG3}$ -)› [81, 81, 81] 80)
where $\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C} = (\prod_{DG} i \in_\circ 3_{\mathbb{N}}.\ if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i)$

Product of three digraphs is a digraph

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$
assumes $\mathfrak{A}$: *digraph* $\alpha\ \mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* $\alpha\ \mathfrak{B}$ **and** $\mathfrak{C}$: *digraph* $\alpha\ \mathfrak{C}$
begin

interpretation $\mathcal{Z}\ \alpha$ **by** (*rule digraphD*[*OF* $\mathfrak{A}$(1)])
interpretation $\mathfrak{A}$: *digraph* $\alpha\ \mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *digraph* $\alpha\ \mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation $\mathfrak{B}$: *digraph* $\alpha\ \mathfrak{C}$ **by** (*rule* $\mathfrak{C}$)

lemma *finite-pdigraph-dg-prod-3*:
finite-pdigraph α ($3_{\mathbb{N}}$) (*if3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$)
proof(*intro finite-pdigraphI pdigraph-baseI*)
from *Axiom-of-Infinity* **show** *z1-in-Vset*: $3_{\mathbb{N}} \in_\circ$ *Vset* α **by** *blast*
show *digraph* α (*if3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i$) **if** $i \in_\circ 3_{\mathbb{N}}$ **for** i
by (*auto intro*: *dg-cs-intros*)
qed *auto*

interpretation *finite-pdigraph* α ‹$3_{\mathbb{N}}$› ‹*if3* $\mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$›
by (*intro finite-pdigraph-dg-prod-3* $\mathfrak{A}\ \mathfrak{B}$)

lemma *digraph-dg-prod-3*[*dg-cs-intros*]: *digraph* α ($\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C}$)
proof–
show *?thesis* **unfolding** *dg-prod-3-def* **by** (*rule pdg-digraph-dg-prod*)
qed

end

Object

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$
assumes $\mathfrak{A}$: *digraph* $\alpha\ \mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* $\alpha\ \mathfrak{B}$ **and** $\mathfrak{C}$: *digraph* $\alpha\ \mathfrak{C}$
begin

lemma *dg-prod-3-ObjI*:

assumes $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows $[a, b, c]_\circ \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Obj|\!)$
unfolding *dg-prod-3-def dg-prod-components*
proof(*intro vproductI ballI*)
show $\mathcal{D}_\circ [a, b, c]_\circ = 3_{\mathbb{N}}$ **by** (*simp add*: *nat-omega-simps*)
fix i **assume** $i \in_\circ 3_{\mathbb{N}}$
then consider $\langle i = 0 \rangle \mid \langle i = 1_{\mathbb{N}} \rangle \mid \langle i = 2_{\mathbb{N}} \rangle$ **unfolding** *three* **by** *auto*
then show $[a, b, c]_\circ(\!|i|\!) \in_\circ (if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i)(\!|Obj|\!)$
by *cases* (*simp-all add*: *nat-omega-simps assms*)
qed *auto*

lemma *dg-prod-3-ObjI'*[*dg-prod-cs-intros*]:
assumes $abc = [a, b, c]_\circ$ **and** $a \in_\circ \mathfrak{A}(\!|Obj|\!)$ **and** $b \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
shows $abc \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Obj|\!)$
using *assms*(2–4) **unfolding** *assms*(1) **by** (*rule dg-prod-3-ObjI*)

lemma *dg-prod-3-ObjE*:
assumes $abc \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Obj|\!)$
obtains $a\ b\ c$
where $abc = [a, b, c]_\circ$
and $a \in_\circ \mathfrak{A}(\!|Obj|\!)$
and $b \in_\circ \mathfrak{B}(\!|Obj|\!)$
and $c \in_\circ \mathfrak{C}(\!|Obj|\!)$
proof–
from *vproductD*[*OF assms*[*unfolded dg-prod-3-def dg-prod-components*]]
have *vsv-abc*: *vsv abc*
and *dom-abc*: $\mathcal{D}_\circ\ abc = 3_{\mathbb{N}}$
and *abc-app*: $\bigwedge i.\ i \in_\circ 3_{\mathbb{N}} \Longrightarrow abc(\!|i|\!) \in_\circ (if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i)(\!|Obj|\!)$
by *auto*
have *dom-abc*[*simp*]: $\mathcal{D}_\circ\ abc = 3_{\mathbb{N}}$
unfolding *dom-abc* **by** (*simp add*: *nat-omega-simps two*)
interpret *vsv abc* **by** (*rule vsv-abc*)
have $abc = [vpfst\ abc, vpsnd\ abc, vpthrd\ abc]_\circ$
by (*rule vsv-vfsequence-three*[*symmetric*]) *auto*
moreover from *abc-app*[*of* 0] **have** *vpfst abc* $\in_\circ \mathfrak{A}(\!|Obj|\!)$ **by** *simp*
moreover from *abc-app*[*of* $\langle 1_{\mathbb{N}} \rangle$] **have** *vpsnd abc* $\in_\circ \mathfrak{B}(\!|Obj|\!)$ **by** *simp*
moreover from *abc-app*[*of* $\langle 2_{\mathbb{N}} \rangle$] **have** *vpthrd abc* $\in_\circ \mathfrak{C}(\!|Obj|\!)$ **by** *simp*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

end

Arrow

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}$
assumes $\mathfrak{A}$: *digraph* $\alpha\ \mathfrak{A}$ **and** $\mathfrak{B}$: *digraph* $\alpha\ \mathfrak{B}$ **and** $\mathfrak{C}$: *digraph* $\alpha\ \mathfrak{C}$
begin

lemma *dg-prod-3-ArrI*:
assumes $h \in_\circ \mathfrak{A}(\!|Arr|\!)$ **and** $g \in_\circ \mathfrak{B}(\!|Arr|\!)$ **and** $f \in_\circ \mathfrak{C}(\!|Arr|\!)$
shows $[h, g, f]_\circ \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Arr|\!)$
unfolding *dg-prod-3-def dg-prod-components*
proof(*intro vproductI ballI*)
show $\mathcal{D}_\circ [h, g, f]_\circ = 3_{\mathbb{N}}$ **by** (*simp add*: *nat-omega-simps three*)
fix i **assume** $i \in_\circ 3_{\mathbb{N}}$
then consider $\langle i = 0 \rangle \mid \langle i = 1_{\mathbb{N}} \rangle \mid \langle i = 2_{\mathbb{N}} \rangle$ **unfolding** *three* **by** *auto*
then show $[h, g, f]_\circ(\!|i|\!) \in_\circ (if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i)(\!|Arr|\!)$

```
    by cases (simp-all add: nat-omega-simps assms)
qed auto

lemma dg-prod-3-ArrI'[dg-prod-cs-intros]:
  assumes hgf = [h, g, f]∘
    and h ∈∘ 𝔄(|Arr|)
    and g ∈∘ 𝔅(|Arr|)
    and f ∈∘ ℭ(|Arr|)
  shows [h, g, f]∘ ∈∘ (𝔄 ×_DG3 𝔅 ×_DG3 ℭ)(|Arr|)
  using assms(2–4) unfolding assms(1) by (rule dg-prod-3-ArrI)

lemma dg-prod-3-ArrE:
  assumes hgf ∈∘ (𝔄 ×_DG3 𝔅 ×_DG3 ℭ)(|Arr|)
  obtains h g f
    where hgf = [h, g, f]∘
      and h ∈∘ 𝔄(|Arr|)
      and g ∈∘ 𝔅(|Arr|)
      and f ∈∘ ℭ(|Arr|)
proof–
  from vproductD[OF assms[unfolded dg-prod-3-def dg-prod-components]]
  have vsv-hgf: vsv hgf
    and dom-hgf: 𝒟∘ hgf = 3_ℕ
    and hgf-app: ⋀i. i ∈∘ 3_ℕ ⟹ hgf(|i|) ∈∘ (if3 𝔄 𝔅 ℭ i)(|Arr|)
    by auto
  have dom-hgf[simp]: 𝒟∘ hgf = 3_ℕ unfolding dom-hgf by (simp add: three)
  interpret vsv hgf by (rule vsv-hgf)
  have hgf = [vpfst hgf, vpsnd hgf, vpthrd hgf]∘
    by (rule vsv-vfsequence-three[symmetric]) auto
  moreover from hgf-app[of 0] have vpfst hgf ∈∘ 𝔄(|Arr|) by simp
  moreover from hgf-app[of ‹1_ℕ›] have vpsnd hgf ∈∘ 𝔅(|Arr|) by simp
  moreover from hgf-app[of ‹2_ℕ›] have vpthrd hgf ∈∘ ℭ(|Arr|) by simp
  ultimately show ?thesis using that by auto
qed

end
```

Arrow with a domain and a codomain

```
context
  fixes α 𝔄 𝔅 ℭ
  assumes 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅 and ℭ: digraph α ℭ
begin

interpretation 𝒵 α by (rule digraphD[OF 𝔄(1)])
interpretation 𝔄: digraph α 𝔄 by (rule 𝔄)
interpretation 𝔅: digraph α 𝔅 by (rule 𝔅)
interpretation ℭ: digraph α ℭ by (rule ℭ)
interpretation finite-pdigraph α ‹3_ℕ› ‹if3 𝔄 𝔅 ℭ›
  by (intro finite-pdigraph-dg-prod-3 𝔄 𝔅 ℭ)

lemma dg-prod-3-is-arrI:
  assumes f : a ↦_𝔄 b and f' : a' ↦_𝔅 b' and f'' : a'' ↦_ℭ b''
  shows [f, f', f'']∘ : [a, a', a'']∘ ↦_𝔄 ×_DG3 𝔅 ×_DG3 ℭ [b, b', b'']∘
  unfolding dg-prod-3-def
proof(rule dg-prod-is-arrI)
  show [f, f', f'']∘(|i|) : [a, a', a'']∘(|i|) ↦_if3 𝔄 𝔅 ℭ i [b, b', b'']∘(|i|)
    if i ∈∘ 3_ℕ for i
  proof–
```

```
    from that consider ‹i = 0› | ‹i = 1ℕ› | ‹i = 2ℕ› unfolding three by auto
    then show
      [f, f′, f′′]∘(|i|) : [a, a′, a′′]∘(|i|) ↦if3 𝔄 𝔅 ℭ i [b, b′, b′′]∘(|i|)
      by cases (simp-all add: nat-omega-simps assms)
  qed
qed (auto simp: nat-omega-simps three)

lemma dg-prod-3-is-arrI'[dg-prod-cs-intros]:
  assumes F = [f, f′, f′′]∘
    and A = [a, a′, a′′]∘
    and B = [b, b′, b′′]∘
    and f : a ↦𝔄 b
    and f′ : a′ ↦𝔅 b′
    and f′′ : a′′ ↦ℭ b′′
  shows F : A ↦𝔄 ×DG3 𝔅 ×DG3 ℭ B
  using assms(4,5,6) unfolding assms(1,2,3) by (rule dg-prod-3-is-arrI)

lemma dg-prod-3-is-arrE:
  assumes F : A ↦𝔄 ×DG3 𝔅 ×DG3 ℭ B
  obtains f f′ f′′ a a′ a′′ b b′ b′′
    where F = [f, f′, f′′]∘
      and A = [a, a′, a′′]∘
      and B = [b, b′, b′′]∘
      and f : a ↦𝔄 b
      and f′ : a′ ↦𝔅 b′
      and f′′ : a′′ ↦ℭ b′′
proof–
  from dg-prod-is-arrD[OF assms[unfolded dg-prod-3-def]]
  have [simp]: vsv F 𝒟∘ F = 3ℕ vsv A 𝒟∘ A = 3ℕ vsv B 𝒟∘ B = 3ℕ
    and F-app: ⋀i. i ∈∘ 3ℕ ⟹ F(|i|) : A(|i|) ↦if3 𝔄 𝔅 ℭ i B(|i|)
    by (auto simp: three)
  have F = [vpfst F, vpsnd F, vpthrd F]∘
    by (simp add: vsv-vfsequence-three)
  moreover have A = [vpfst A, vpsnd A, vpthrd A]∘
    by (simp add: vsv-vfsequence-three)
  moreover have B = [vpfst B, vpsnd B, vpthrd B]∘
    by (simp add: vsv-vfsequence-three)
  moreover from F-app[of 0] have vpfst F : vpfst A ↦𝔄 vpfst B by simp
  moreover from F-app[of ‹1ℕ›] have vpsnd F : vpsnd A ↦𝔅 vpsnd B
    by (simp add: nat-omega-simps)
  moreover from F-app[of ‹2ℕ›] have vpthrd F : vpthrd A ↦ℭ vpthrd B
    by (simp add: nat-omega-simps)
  ultimately show ?thesis using that by auto
qed

end
```

Domain

```
context
  fixes α 𝔄 𝔅 ℭ
  assumes 𝔄: digraph α 𝔄 and 𝔅: digraph α 𝔅 and ℭ: digraph α ℭ
begin

interpretation 𝒵 α by (rule digraphD[OF 𝔄(1)])
interpretation 𝔄: digraph α 𝔄 by (rule 𝔄)
interpretation 𝔅: digraph α 𝔅 by (rule 𝔅)
interpretation ℭ: digraph α ℭ by (rule ℭ)
```

lemma *dg-prod-3-Dom-vsv*: *vsv* ((𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|))
unfolding *dg-prod-3-def dg-prod-components* **by** *simp*

lemma *dg-prod-3-Dom-vdomain*[*dg-cs-simps*]:
$\mathcal{D}_\circ$ ((𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|)) = (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Arr*|)
unfolding *dg-prod-3-def dg-prod-components* **by** *simp*

lemma *dg-prod-3-Dom-app*[*dg-prod-cs-simps*]:
assumes [*f*, *f'*, *f''*]$_\circ$ ∈$_\circ$ (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Arr*|)
shows (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|)(|*f*, *f'*, *f''*|)$_\bullet$ =
[𝔄(|*Dom*|)(|*f*|), 𝔅(|*Dom*|)(|*f'*|), ℭ(|*Dom*|)(|*f''*|)]$_\circ$
proof–
from *assms* **obtain** *A B* **where** *F*: [*f*, *f'*, *f''*]$_\circ$: *A* ↦$_{\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C}}$ *B*
by (*auto intro*: *is-arrI*)
then have *Dom-F*: (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|)(|*f*, *f'*, *f''*|)$_\bullet$ = *A*
by (*simp add*: *dg-cs-simps*)
from *F* **obtain** *a a' a'' b b' b''*
where *A-def*: *A* = [*a*, *a'*, *a''*]$_\circ$
and *B* = [*b*, *b'*, *b''*]$_\circ$
and *f* : *a* ↦$_{\mathfrak{A}}$ *b*
and *f'* : *a'* ↦$_{\mathfrak{B}}$ *b'*
and *f''* : *a''* ↦$_{\mathfrak{C}}$ *b''*
by (*elim dg-prod-3-is-arrE*[*OF* 𝔄 𝔅 ℭ]) *simp*
then have *Dom-f*: 𝔄(|*Dom*|)(|*f*|) = *a*
and *Dom-f'*: 𝔅(|*Dom*|)(|*f'*|) = *a'*
and *Dom-f''*: ℭ(|*Dom*|)(|*f''*|) = *a''*
by (*simp-all add*: *dg-cs-simps*)
show *?thesis* **unfolding** *Dom-F A-def Dom-f Dom-f' Dom-f''* **..**
qed

lemma *dg-prod-3-Dom-vrange*:
$\mathcal{R}_\circ$ ((𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|)) ⊆$_\circ$ (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Obj*|)
proof(*rule vsv.vsv-vrange-vsubset*, *unfold dg-cs-simps*)
show *vsv* ((𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|)) **by** (*rule dg-prod-3-Dom-vsv*)
fix *F* **assume** *prems*: *F* ∈$_\circ$ (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Arr*|)
then obtain *f f' f''* **where** *F-def*: *F* = [*f*, *f'*, *f''*]$_\circ$
and *f*: *f* ∈$_\circ$ 𝔄(|*Arr*|)
and *f'*: *f'* ∈$_\circ$ 𝔅(|*Arr*|)
and *f''*: *f''* ∈$_\circ$ ℭ(|*Arr*|)
by (*elim dg-prod-3-ArrE*[*OF* 𝔄 𝔅 ℭ]) *simp*
from *f f' f''* **obtain** *a a' a'' b b' b''*
where *f*: *f* : *a* ↦$_{\mathfrak{A}}$ *b*
and *f'*: *f'* : *a'* ↦$_{\mathfrak{B}}$ *b'*
and *f''*: *f''* : *a''* ↦$_{\mathfrak{C}}$ *b''*
by (*meson is-arrI*)
from 𝔄 𝔅 *f f' f''* **show** (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Dom*|)(|*F*|) ∈$_\circ$ (𝔄 ×$_{DG3}$ 𝔅 ×$_{DG3}$ ℭ)(|*Obj*|)
unfolding *F-def dg-prod-3-Dom-app*[*OF prems*[*unfolded F-def*]]
by (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-prod-cs-intros*)
qed

end

Codomain

context
fixes α 𝔄 𝔅 ℭ
assumes 𝔄: *digraph* α 𝔄 **and** 𝔅: *digraph* α 𝔅 **and** ℭ: *digraph* α ℭ

begin

interpretation $\mathcal{Z}$ α **by** (*rule digraphD*[*OF* $\mathfrak{A}(1)$])
interpretation $\mathfrak{A}$: *digraph* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *digraph* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation $\mathfrak{C}$: *digraph* α $\mathfrak{C}$ **by** (*rule* $\mathfrak{C}$)

lemma *dg-prod-3-Cod-vsv*: *vsv* $((\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!))$
unfolding *dg-prod-3-def dg-prod-components* **by** *simp*

lemma *dg-prod-3-Cod-vdomain*[*dg-cs-simps*]:
$\mathcal{D}_\circ\ ((\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!)) = (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Arr|\!)$
unfolding *dg-prod-3-def dg-prod-components* **by** *simp*

lemma *dg-prod-3-Cod-app*[*dg-prod-cs-simps*]:
assumes $[f, f', f'']_\circ \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Arr|\!)$
shows
$(\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!)(\!|f, f', f''|\!)_\bullet =$
$[\mathfrak{A}(\!|Cod|\!)(\!|f|\!), \mathfrak{B}(\!|Cod|\!)(\!|f'|\!), \mathfrak{C}(\!|Cod|\!)(\!|f''|\!)]_\circ$
proof–
from *assms* **obtain** A B **where** F: $[f, f', f'']_\circ : A \mapsto_{\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C}} B$
by (*auto intro*: *is-arrI*)
then have *Cod-F*: $(\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!)(\!|f, f', f''|\!)_\bullet = B$
by (*simp add*: *dg-cs-simps*)
from F **obtain** a a' a'' b b' b''
where $A = [a, a', a'']_\circ$
and *B-def*: $B = [b, b', b'']_\circ$
and $f : a \mapsto_{\mathfrak{A}} b$
and $f' : a' \mapsto_{\mathfrak{B}} b'$
and $f'' : a'' \mapsto_{\mathfrak{C}} b''$
by (*elim dg-prod-3-is-arrE*[*OF* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$]) *simp*
then have *Cod-f*: $\mathfrak{A}(\!|Cod|\!)(\!|f|\!) = b$
and *Cod-f'*: $\mathfrak{B}(\!|Cod|\!)(\!|f'|\!) = b'$
and *Cod-f''*: $\mathfrak{C}(\!|Cod|\!)(\!|f''|\!) = b''$
by (*simp-all add*: *dg-cs-simps*)
show *?thesis* **unfolding** *Cod-F B-def Cod-f Cod-f' Cod-f''* **..**
qed

lemma *dg-prod-3-Cod-vrange*:
$\mathcal{R}_\circ\ ((\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!)) \subseteq_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Obj|\!)$
proof(*rule vsv.vsv-vrange-vsubset*, *unfold dg-cs-simps*)
show *vsv* $((\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!))$ **by** (*rule dg-prod-3-Cod-vsv*)
fix F **assume** *prems*: $F \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Arr|\!)$
then obtain $f\ f'\ f''$ **where** *F-def*: $F = [f, f', f'']_\circ$
and f: $f \in_\circ \mathfrak{A}(\!|Arr|\!)$
and f': $f' \in_\circ \mathfrak{B}(\!|Arr|\!)$
and f'': $f'' \in_\circ \mathfrak{C}(\!|Arr|\!)$
by (*elim dg-prod-3-ArrE*[*OF* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$]) *simp*
from $f\ f'\ f''$ **obtain** a a' a'' b b' b''
where f: $f : a \mapsto_{\mathfrak{A}} b$
and f': $f' : a' \mapsto_{\mathfrak{B}} b'$
and f'': $f'' : a'' \mapsto_{\mathfrak{C}} b''$
by (*metis is-arrI*)
from $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ $f\ f'\ f''$ **show**
$(\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Cod|\!)(\!|F|\!) \in_\circ (\mathfrak{A} \times_{DG3} \mathfrak{B} \times_{DG3} \mathfrak{C})(\!|Obj|\!)$
unfolding *F-def dg-prod-3-Cod-app*[*OF prems*[*unfolded F-def*]]
by
(

```
      cs-concl cs-shallow
        cs-simp: dg-cs-simps cs-intro: dg-cs-intros dg-prod-cs-intros
    )
qed

end
```

3.9 Subdigraph

3.9.1 Background

In this body of work, a subdigraph is a natural generalization of the concept of a subcategory, as defined in Chapter I-3 in [39], to digraphs. It should be noted that a similar concept also exists in the conventional graph theory, but further details are considered to be outside of the scope of this work.

named-theorems *dg-sub-cs-intros*
named-theorems *dg-sub-bw-cs-intros*
named-theorems *dg-sub-fw-cs-intros*
named-theorems *dg-sub-bw-cs-simps*

3.9.2 Simple subdigraph

Definition and elementary properties

locale *subdigraph* = *sdg*: *digraph* α $\mathfrak{B}$ + *dg*: *digraph* α $\mathfrak{C}$ **for** α $\mathfrak{B}$ $\mathfrak{C}$ +
  assumes *subdg-Obj-vsubset*[*dg-sub-fw-cs-intros*]:
    $a \in_\circ \mathfrak{B}(\!|Obj|\!) \Longrightarrow a \in_\circ \mathfrak{C}(\!|Obj|\!)$
    and *subdg-is-arr-vsubset*[*dg-sub-fw-cs-intros*]:
      $f : a \mapsto_{\mathfrak{B}} b \Longrightarrow f : a \mapsto_{\mathfrak{C}} b$

abbreviation *is-subdigraph* ($\langle$(-/ $\subseteq_{DG^1}$ -)$\rangle$ [51, 51] 50)
  where $\mathfrak{B} \subseteq_{DG\alpha} \mathfrak{C} \equiv$ *subdigraph* α $\mathfrak{B}$ $\mathfrak{C}$

lemmas [*dg-sub-fw-cs-intros*] =
  subdigraph.subdg-Obj-vsubset
  subdigraph.subdg-is-arr-vsubset

Rules.

lemma (**in** *subdigraph*) *subdigraph-axioms'*[*dg-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
  shows $\mathfrak{B}' \subseteq_{DG_{\alpha'}} \mathfrak{C}$
  unfolding *assms* **by** (*rule subdigraph-axioms*)

lemma (**in** *subdigraph*) *subdigraph-axioms''*[*dg-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows $\mathfrak{B} \subseteq_{DG_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule subdigraph-axioms*)

mk-ide rf *subdigraph-def*[*unfolded subdigraph-axioms-def*]
  |*intro subdigraphI*|
  |*dest subdigraphD*[*dest*]|
  |*elim subdigraphE*[*elim*!]|

lemmas [*dg-sub-cs-intros*] = *subdigraphD*(1,2)

The opposite subdigraph.

lemma (**in** *subdigraph*) *subdg-subdigraph-op-dg-op-dg*: *op-dg* $\mathfrak{B} \subseteq_{DG\alpha}$ *op-dg* $\mathfrak{C}$
proof(*rule subdigraphI*)
  show $a \in_\circ$ *op-dg* $\mathfrak{B}(\!|Obj|\!) \Longrightarrow a \in_\circ$ *op-dg* $\mathfrak{C}(\!|Obj|\!)$ **for** a
    by (*auto simp*: *dg-op-simps subdg-Obj-vsubset*)
  show $f : a \mapsto_{op\text{-}dg} \mathfrak{B}\ b \Longrightarrow f : a \mapsto_{op\text{-}dg} \mathfrak{C}\ b$ **for** $f\ a\ b$
    by (*auto simp*: *dg-op-simps subdg-is-arr-vsubset*)
qed (*auto simp*: *dg-op-simps intro*: *dg-op-intros*)

lemmas *subdg-subdigraph-op-dg-op-dg*[*dg-op-intros*] =
subdigraph.subdg-subdigraph-op-dg-op-dg

Further rules.

lemma (**in** *subdigraph*) *subdg-objD*:
assumes $a \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows $a \in_\circ \mathfrak{C}(\!|Obj|\!)$
using *assms* **by** (*auto intro*: *subdg-Obj-vsubset*)

lemmas [*dg-sub-fw-cs-intros*] = *subdigraph.subdg-objD*

lemma (**in** *subdigraph*) *subdg-arrD*[*dg-sub-fw-cs-intros*]:
assumes $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
shows $f \in_\circ \mathfrak{C}(\!|Arr|\!)$
proof–
from *assms* **obtain** *a b* **where** $f : a \mapsto_{\mathfrak{B}} b$ **by** *auto*
then show $f \in_\circ \mathfrak{C}(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-intro**: *subdg-is-arr-vsubset dg-cs-intros*)
qed

lemmas [*dg-sub-fw-cs-intros*] = *subdigraph.subdg-arrD*

lemma (**in** *subdigraph*) *subdg-dom-simp*:
assumes $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
shows $\mathfrak{B}(\!|Dom|\!)(\!|f|\!) = \mathfrak{C}(\!|Dom|\!)(\!|f|\!)$
proof–
from *assms* **obtain** *a b* **where** $f : a \mapsto_{\mathfrak{B}} b$ **by** *auto*
then show $\mathfrak{B}(\!|Dom|\!)(\!|f|\!) = \mathfrak{C}(\!|Dom|\!)(\!|f|\!)$
by (*force dest*: *subdg-is-arr-vsubset simp*: *dg-cs-simps*)
qed

lemmas [*dg-sub-bw-cs-simps*] = *subdigraph.subdg-dom-simp*

lemma (**in** *subdigraph*) *subdg-cod-simp*:
assumes $f \in_\circ \mathfrak{B}(\!|Arr|\!)$
shows $\mathfrak{B}(\!|Cod|\!)(\!|f|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$
proof–
from *assms* **obtain** *a b* **where** $f : a \mapsto_{\mathfrak{B}} b$ **by** *auto*
then show $\mathfrak{B}(\!|Cod|\!)(\!|f|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$
by (*force dest*: *subdg-is-arr-vsubset simp*: *dg-cs-simps*)
qed

lemmas [*dg-sub-bw-cs-simps*] = *subdigraph.subdg-cod-simp*

lemma (**in** *subdigraph*) *subdg-is-arrD*:
assumes $f : a \mapsto_{\mathfrak{B}} b$
shows $f : a \mapsto_{\mathfrak{C}} b$
using *assms subdg-is-arr-vsubset* **by** *simp*

lemmas [*dg-sub-fw-cs-intros*] = *subdigraph.subdg-is-arrD*

The subdigraph relation is a partial order

lemma *subdg-refl*:
assumes *digraph* α $\mathfrak{A}$
shows $\mathfrak{A} \subseteq_{DG\alpha} \mathfrak{A}$
proof–
interpret *digraph* α $\mathfrak{A}$ **by** (*rule assms*)

```
  show ?thesis by unfold-locales simp
qed

lemma subdg-trans[trans]:
  assumes 𝔄 ⊆_DGα 𝔅 and 𝔅 ⊆_DGα ℭ
  shows 𝔄 ⊆_DGα ℭ
proof–
  interpret 𝔄𝔅: subdigraph α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅ℭ: subdigraph α 𝔅 ℭ by (rule assms(2))
  show ?thesis
    by  unfold-locales
      (
        insert 𝔄𝔅.subdigraph-axioms,
        auto simp:
          𝔅ℭ.subdg-Obj-vsubset
          𝔄𝔅.subdg-Obj-vsubset
          subdigraph.subdg-is-arr-vsubset
          𝔅ℭ.subdg-is-arr-vsubset
      )
qed

lemma subdg-antisym:
  assumes 𝔄 ⊆_DGα 𝔅 and 𝔅 ⊆_DGα 𝔄
  shows 𝔄 = 𝔅
proof–
  interpret 𝔄𝔅: subdigraph α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅𝔄: subdigraph α 𝔅 𝔄 by (rule assms(2))
  show ?thesis
  proof(rule dg-eqI)
    from assms show Arr: 𝔄(|Arr|) = 𝔅(|Arr|)
      by (intro vsubset-antisym vsubsetI)
        (auto simp: dg-sub-bw-cs-simps intro: dg-sub-fw-cs-intros)
    from assms show 𝔄(|Obj|) = 𝔅(|Obj|)
      by (intro vsubset-antisym vsubsetI)
        (auto simp: dg-sub-bw-cs-simps intro: dg-sub-fw-cs-intros)
    show 𝔄(|Dom|) = 𝔅(|Dom|)
      by (rule vsv-eqI) (auto simp: 𝔄𝔅.subdg-dom-simp Arr dg-cs-simps)
    show 𝔄(|Cod|) = 𝔅(|Cod|)
      by (rule vsv-eqI) (auto simp: 𝔄𝔅.subdg-cod-simp Arr dg-cs-simps)
  qed (cs-concl cs-shallow cs-intro: dg-cs-intros)+
qed
```

3.9.3 Inclusion digraph homomorphism

Definition and elementary properties

See Chapter I-3 in [39].

```
definition dghm-inc :: V ⇒ V ⇒ V
  where dghm-inc 𝔅 ℭ = [vid-on (𝔅(|Obj|)), vid-on (𝔅(|Arr|)), 𝔅, ℭ]∘
```

Components.

```
lemma dghm-inc-components:
  shows dghm-inc 𝔅 ℭ(|ObjMap|) = vid-on (𝔅(|Obj|))
    and dghm-inc 𝔅 ℭ(|ArrMap|) = vid-on (𝔅(|Arr|))
    and [dg-cs-simps]: dghm-inc 𝔅 ℭ(|HomDom|) = 𝔅
    and [dg-cs-simps]: dghm-inc 𝔅 ℭ(|HomCod|) = ℭ
  unfolding dghm-inc-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

Object map

mk-VLambda *dghm-inc-components*(1)[*folded VLambda-vid-on*]
|*vsv dghm-inc-ObjMap-vsv*[*dg-cs-intros*]|
|*vdomain dghm-inc-ObjMap-vdomain*[*dg-cs-simps*]|
|*app dghm-inc-ObjMap-app*[*dg-cs-simps*]|

Arrow map

mk-VLambda *dghm-inc-components*(2)[*folded VLambda-vid-on*]
|*vsv dghm-inc-ArrMap-vsv*[*dg-cs-intros*]|
|*vdomain dghm-inc-ArrMap-vdomain*[*dg-cs-simps*]|
|*app dghm-inc-ArrMap-app*[*dg-cs-simps*]|

Canonical inclusion digraph homomorphism associated with a subdigraph

sublocale *subdigraph* ⊆ *inc*: *is-ft-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ ‹*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$›
proof(*intro is-ft-dghmI is-dghmI*)
show *vfsequence* (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$) **unfolding** *dghm-inc-def* **by** *auto*
show *vcard* (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$) = $4_{\mathbb{N}}$
unfolding *dghm-inc-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_{\circ}$ (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)$) $\subseteq_{\circ}$ $\mathfrak{C}(\!|Obj|\!)$
unfolding *dghm-inc-components* **by** (*auto dest*: *subdg-objD*)
show *dghm-inc* $\mathfrak{B}$ $\mathfrak{C}(\!|ArrMap|\!)(\!|f|\!)$:
dghm-inc $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|a|\!)$ $\mapsto_{\mathfrak{C}}$ *dghm-inc* $\mathfrak{B}$ $\mathfrak{C}(\!|ObjMap|\!)(\!|b|\!)$
if $f : a \mapsto_{\mathfrak{B}} b$ **for** *a b f*
using *that*
by (*cs-concl* **cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros dg-sub-fw-cs-intros*)
show *v*11 (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}(\!|ArrMap|\!)$ $\restriction^{l}_{\circ}$ *Hom* $\mathfrak{B}$ *a b*)
if $a \in_{\circ} \mathfrak{B}(\!|Obj|\!)$ **and** $b \in_{\circ} \mathfrak{B}(\!|Obj|\!)$ **for** *a b*
using *that* **unfolding** *dghm-inc-components* **by** *simp*
qed (*cs-concl* **cs-shallow cs-simp**: *dg-cs-simps* **cs-intro**: *dg-cs-intros*)+

lemmas (**in** *subdigraph*) *subdg-dghm-inc-is-ft-dghm* = *inc.is-ft-dghm-axioms*

The inclusion digraph homomorphism for the opposite digraphs

lemma (**in** *subdigraph*) *subdg-dghm-inc-op-dg-is-dghm*[*dg-sub-cs-intros*]:
dghm-inc (*op-dg* $\mathfrak{B}$) (*op-dg* $\mathfrak{C}$) : *op-dg* $\mathfrak{B}$ $\mapsto\!\!\mapsto_{DG.faithful\alpha}$ *op-dg* $\mathfrak{C}$
by (*intro subdigraph.subdg-dghm-inc-is-ft-dghm subdg-subdigraph-op-dg-op-dg*)

lemmas [*dg-sub-cs-intros*] = *subdigraph.subdg-dghm-inc-op-dg-is-dghm*

lemma (**in** *subdigraph*) *subdg-op-dg-dghm-inc*[*dg-op-simps*]:
op-dghm (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$) = *dghm-inc* (*op-dg* $\mathfrak{B}$) (*op-dg* $\mathfrak{C}$)
by (*rule dghm-eqI*, *unfold dg-op-simps dghm-inc-components id-def*)
(
auto
simp: *subdg-dghm-inc-op-dg-is-dghm is-ft-dghmD*
intro: *dg-op-intros dg-cs-intros*
)

lemmas [*dg-op-simps*] = *subdigraph.subdg-op-dg-dghm-inc*

3.9.4 Full subdigraph

See Chapter I-3 in [39].

locale *fl-subdigraph* = *subdigraph* +

assumes *fl-subdg-is-fl-dghm-inc*: *dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$: $\mathfrak{B} \mapsto\mapsto_{DG.full\alpha}$ $\mathfrak{C}$

abbreviation *is-fl-subdigraph* (‹(-/ $\subseteq_{DG.full1}$ -)› [51, 51] 50)
where $\mathfrak{B} \subseteq_{DG.full\alpha}$ $\mathfrak{C}$ ≡ *fl-subdigraph* α $\mathfrak{B}$ $\mathfrak{C}$

sublocale *fl-subdigraph* ⊆ *inc*: *is-fl-dghm* α $\mathfrak{B}$ $\mathfrak{C}$ ‹*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$›
by (*rule fl-subdg-is-fl-dghm-inc*)

Rules.

lemma (**in** *fl-subdigraph*) *fl-subdigraph-axioms'*[*dg-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{B}' \subseteq_{DG.full_{\alpha'}}$ $\mathfrak{C}$
unfolding *assms* **by** (*rule fl-subdigraph-axioms*)

lemma (**in** *fl-subdigraph*) *fl-subdigraph-axioms''*[*dg-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{B} \subseteq_{DG.full_{\alpha'}}$ $\mathfrak{C}'$
unfolding *assms* **by** (*rule fl-subdigraph-axioms*)

mk-ide rf *fl-subdigraph-def*[*unfolded fl-subdigraph-axioms-def*]
|*intro fl-subdigraphI*|
|*dest fl-subdigraphD*[*dest*]|
|*elim fl-subdigraphE*[*elim*!]|

lemmas [*dg-sub-cs-intros*] = *fl-subdigraphD*(1)

Elementary properties.

lemma (**in** *fl-subdigraph*) *fl-subdg-Hom-eq*:
assumes $A \in_\circ \mathfrak{B}(\!|Obj|\!)$ **and** $B \in_\circ \mathfrak{B}(\!|Obj|\!)$
shows *Hom* $\mathfrak{B}$ *A B* = *Hom* $\mathfrak{C}$ *A B*
proof–
from *assms* **have** *Arr-AB*: $\mathfrak{B}(\!|Arr|\!) \cap_\circ$ *Hom* $\mathfrak{B}$ *A B* = *Hom* $\mathfrak{B}$ *A B*
by
(
intro vsubset-antisym vsubsetI,
unfold vintersection-iff in-Hom-iff;
(*elim conjE*)*?*;
(*intro conjI*)*?*
)
(*auto intro*: *dg-cs-intros*)
from *assms* **have** *A*: *vid-on* ($\mathfrak{B}(\!|Obj|\!)$)(|*A*|) = *A* **and** *B*: *vid-on* ($\mathfrak{B}(\!|Obj|\!)$)(|*B*|) = *B*
by *simp-all*
from *inc.fl-dghm-surj-on-Hom*[*OF assms*, *unfolded dghm-inc-components*] **show**
Hom $\mathfrak{B}$ *A B* = *Hom* $\mathfrak{C}$ *A B*
by (*auto simp*: *Arr-AB A B*)
qed

3.9.5 Wide subdigraph

Definition and elementary properties

See [3][3]).

locale *wide-subdigraph* = *subdigraph* +
assumes *wide-subdg-Obj*[*dg-sub-bw-cs-intros*]: $a \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow a \in_\circ \mathfrak{B}(\!|Obj|\!)$

abbreviation *is-wide-subdigraph* (‹(-/ $\subseteq_{DG.wide1}$ -)› [51, 51] 50)

[3] https://ncatlab.org/nlab/show/wide+subcategory

where $\mathfrak{B} \subseteq_{DG.wide\alpha} \mathfrak{C} \equiv$ *wide-subdigraph* α $\mathfrak{B}$ $\mathfrak{C}$

lemmas [*dg-sub-bw-cs-intros*] = *wide-subdigraph.wide-subdg-Obj*

Rules.

lemma (**in** *wide-subdigraph*) *wide-subdigraph-axioms'*[*dg-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{B}' \subseteq_{DG.wide_{\alpha'}} \mathfrak{C}$
unfolding *assms* **by** (*rule wide-subdigraph-axioms*)

lemma (**in** *wide-subdigraph*) *wide-subdigraph-axioms''*[*dg-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{B} \subseteq_{DG.wide_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule wide-subdigraph-axioms*)

mk-ide rf *wide-subdigraph-def*[*unfolded wide-subdigraph-axioms-def*]
|*intro wide-subdigraphI*|
|*dest wide-subdigraphD*[*dest*]|
|*elim wide-subdigraphE*[*elim*!]|

lemmas [*dg-sub-cs-intros*] = *wide-subdigraphD*(1)

Elementary properties.

lemma (**in** *wide-subdigraph*) *wide-subdg-obj-eq*[*dg-sub-bw-cs-simps*]:
$\mathfrak{B}(|Obj|) = \mathfrak{C}(|Obj|)$
using *subdg-Obj-vsubset wide-subdg-Obj* **by** *auto*

lemmas [*dg-sub-bw-cs-simps*] = *wide-subdigraph.wide-subdg-obj-eq*

The wide subdigraph relation is a partial order

lemma *wide-subdg-refl*:
assumes *digraph* α $\mathfrak{A}$
shows $\mathfrak{A} \subseteq_{DG.wide\alpha} \mathfrak{A}$
proof–
interpret *digraph* α $\mathfrak{A}$ **by** (*rule assms*)
show *?thesis* **by** *unfold-locales simp*
qed

lemma *wide-subdg-trans*[*trans*]:
assumes $\mathfrak{A} \subseteq_{DG.wide\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{DG.wide\alpha} \mathfrak{C}$
shows $\mathfrak{A} \subseteq_{DG.wide\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{AB}$: *wide-subdigraph* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(1))
interpret $\mathfrak{BC}$: *wide-subdigraph* α $\mathfrak{B}$ $\mathfrak{C}$ **by** (*rule assms*(2))
interpret $\mathfrak{AC}$: *subdigraph* α $\mathfrak{A}$ $\mathfrak{C}$
by (*rule subdg-trans*) (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)+
show *?thesis*
by (*cs-concl* **cs-intro**: *dg-sub-bw-cs-intros dg-cs-intros wide-subdigraphI*)
qed

lemma *wide-subdg-antisym*:
assumes $\mathfrak{A} \subseteq_{DG.wide\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{DG.wide\alpha} \mathfrak{A}$
shows $\mathfrak{A} = \mathfrak{B}$
proof–
interpret $\mathfrak{AB}$: *wide-subdigraph* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(1))
interpret $\mathfrak{BA}$: *wide-subdigraph* α $\mathfrak{B}$ $\mathfrak{A}$ **by** (*rule assms*(2))
show *?thesis*

by (*rule subdg-antisym*[*OF* $\mathfrak{AB}$.*subdigraph-axioms* $\mathfrak{BA}$.*subdigraph-axioms*])
qed

3.10 Simple digraphs

3.10.1 Background

The section presents a variety of simple digraphs, such as the empty digraph 0 and a digraph with one object and one arrow 1. All of the entities presented in this section are generalizations of certain simple categories, whose definitions can be found in [39].

3.10.2 Empty digraph 0

Definition and elementary properties

See Chapter I-2 in [39].

definition *dg*-0 :: *V*
  where *dg*-0 = [0, 0, 0, 0]$_{\circ}$

Components.

lemma *dg*-0-*components*:
  shows *dg*-0(|*Obj*|) = 0
    and *dg*-0(|*Arr*|) = 0
    and *dg*-0(|*Dom*|) = 0
    and *dg*-0(|*Cod*|) = 0
  unfolding *dg*-0-*def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

0 is a digraph

lemma (**in** $\mathcal{Z}$) *digraph-dg*-0[*dg-cs-intros*]: *digraph* α *dg*-0
proof(*intro digraphI*)
  show *vfsequence dg*-0 **unfolding** *dg*-0-*def* **by** (*simp add*: *nat-omega-simps*)
  show *vcard dg*-0 = $4_{\mathbb{N}}$ **unfolding** *dg*-0-*def* **by** (*simp add*: *nat-omega-simps*)
qed (*auto simp*: *dg*-0-*components*)

lemmas [*dg-cs-intros*] = $\mathcal{Z}$.*digraph-dg*-0

Opposite of the digraph 0

lemma *op-dg-dg*-0[*dg-op-simps*]: *op-dg* (*dg*-0) = *dg*-0
proof(*rule dg-eqI*, *unfold dg-op-simps*)
  define β **where** $\beta = \omega + \omega$
  interpret β: $\mathcal{Z}$ β **unfolding** β-*def* **by** (*rule* $\mathcal{Z}$-$\omega\omega$)
  show *digraph* β (*op-dg dg*-0)
    by (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros dg-op-intros*)
  show *digraph* β *dg*-0 **by** (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros*)
qed (*simp-all add*: *dg*-0-*components op-dg-components*)

Arrow with a domain and a codomain

lemma *dg*-0-*is-arr-iff*[*simp*]: $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}0} \mathfrak{B} \longleftrightarrow$ *False*
  by (*rule iffI*; (*elim is-arrE*)*?*) (*auto simp*: *dg*-0-*components*)

A digraph without objects is empty

lemma (**in** *digraph*) *dg-dg*-0-*if-Obj*-0:
  assumes $\mathfrak{C}$(|*Obj*|) = 0
  shows $\mathfrak{C}$ = *dg*-0
  by (*rule dg-eqI*[*of* α])
    (

auto simp:
dg-cs-intros
assms
digraph-dg-0
dg-0-*components*
dg-Arr-vempty-if-Obj-vempty
dg-Cod-vempty-if-Arr-vempty
dg-Dom-vempty-if-Arr-vempty
)

3.10.3 Empty digraph homomorphism

Definition and elementary properties

definition *dghm*-0 :: $V \Rightarrow V$
where *dghm*-0 $\mathfrak{A}$ = $[0, 0, dg\text{-}0, \mathfrak{A}]_\circ$

Components.

lemma *dghm*-0-*components*:
shows *dghm*-0 $\mathfrak{A}(\lfloor ObjMap \rfloor)$ = 0
and *dghm*-0 $\mathfrak{A}(\lfloor ArrMap \rfloor)$ = 0
and *dghm*-0 $\mathfrak{A}(\lfloor HomDom \rfloor)$ = *dg*-0
and *dghm*-0 $\mathfrak{A}(\lfloor HomCod \rfloor)$ = $\mathfrak{A}$
unfolding *dghm*-0-*def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Opposite empty digraph homomorphism.

lemma *op-dghm-dghm*-0: *op-dghm* (*dghm*-0 $\mathfrak{C}$) = *dghm*-0 (*op-dg* $\mathfrak{C}$)
unfolding
dghm-0-*def op-dg-def op-dghm-def dg*-0-*def dghm-field-simps dg-field-simps*
by (*simp add*: *nat-omega-simps*)

Object map

lemma *dghm*-0-*ObjMap-vsv*[*dg-cs-intros*]: *vsv* (*dghm*-0 $\mathfrak{C}(\lfloor ObjMap \rfloor)$)
unfolding *dghm*-0-*components* **by** *simp*

Arrow map

lemma *dghm*-0-*ArrMap-vsv*[*dg-cs-intros*]: *vsv* (*dghm*-0 $\mathfrak{C}(\lfloor ArrMap \rfloor)$)
unfolding *dghm*-0-*components* **by** *simp*

Empty digraph homomorphism is a faithful digraph homomorphism

lemma (**in** $\mathcal{Z}$) *dghm*-0-*is-ft-dghm*:
assumes *digraph* α $\mathfrak{A}$
shows *dghm*-0 $\mathfrak{A}$: *dg*-0 $\mapsto\!\!\!\mapsto_{DG.faithful\alpha}$ $\mathfrak{A}$
proof(*rule is-ft-dghmI*)
show *dghm*-0 $\mathfrak{A}$: *dg*-0 $\mapsto\!\!\!\mapsto_{DG\alpha}$ $\mathfrak{A}$
proof(*rule is-dghmI*)
show *vfsequence* (*dghm*-0 $\mathfrak{A}$) **unfolding** *dghm*-0-*def* **by** *simp*
show *vcard* (*dghm*-0 $\mathfrak{A}$) = $4_{\mathbb{N}}$
unfolding *dghm*-0-*def* **by** (*simp add*: *nat-omega-simps*)
qed (*auto simp*: *assms digraph-dg*-0 *dghm*-0-*components dg*-0-*components*)
qed (*auto simp*: *dg*-0-*components dghm*-0-*components*)

lemma (**in** $\mathcal{Z}$) *dghm*-0-*is-ft-dghm'*[*dghm-cs-intros*]:
assumes *digraph* α $\mathfrak{A}$
and $\mathfrak{B}'$ = $\mathfrak{A}$
and $\mathfrak{A}'$ = *dg*-0

shows *dghm*-0 $\mathfrak{A}$: $\mathfrak{A}' \mapsto\mapsto_{DG.faithful\alpha}$ $\mathfrak{B}'$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule dghm*-0-*is-ft-dghm*)

lemmas [*dghm-cs-intros*] = $\mathcal{Z}$.*dghm*-0-*is-ft-dghm*′

lemma (**in** $\mathcal{Z}$) *dghm*-0-*is-dghm*:
assumes *digraph* α $\mathfrak{A}$
shows *dghm*-0 $\mathfrak{A}$: *dg*-0 $\mapsto\mapsto_{DG\alpha}$ $\mathfrak{A}$
using *dghm*-0-*is-ft-dghm*[*OF assms*] **by** *auto*

lemma (**in** $\mathcal{Z}$) *dghm*-0-*is-dghm*′[*dg-cs-intros*]:
assumes *digraph* α $\mathfrak{A}$
and $\mathfrak{B}'$ = $\mathfrak{A}$
and $\mathfrak{A}'$ = *dg*-0
shows *dghm*-0 $\mathfrak{A}$: $\mathfrak{A}' \mapsto\mapsto_{DG\alpha}$ $\mathfrak{B}'$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule dghm*-0-*is-dghm*)

lemmas [*dg-cs-intros*] = $\mathcal{Z}$.*dghm*-0-*is-dghm*′

Further properties

lemma *is-dghm-is-dghm*-0-*if-dg*-0:
assumes $\mathfrak{F}$: *dg*-0 $\mapsto\mapsto_{DG\alpha}$ $\mathfrak{C}$
shows $\mathfrak{F}$ = *dghm*-0 $\mathfrak{C}$
proof(*rule dghm-eqI*)
interpret $\mathfrak{F}$: *is-dghm* α *dg*-0 $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(1))
show $\mathfrak{F}$: *dg*-0 $\mapsto\mapsto_{DG\alpha}$ $\mathfrak{C}$ **by** (*rule assms*(1))
then have *dom-lhs*: $\mathcal{D}_\circ$ ($\mathfrak{F}(\!|ObjMap|\!)$) = 0 $\mathcal{D}_\circ$ ($\mathfrak{F}(\!|ArrMap|\!)$) = 0
by (*cs-concl* **cs-simp**: *dg-cs-simps dg*-0-*components*)+
show *dghm*-0 $\mathfrak{C}$: *dg*-0 $\mapsto\mapsto_{DG\alpha}$ $\mathfrak{C}$ **by** (*cs-concl* **cs-intro**: *dg-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_\circ$ (*dghm*-0 $\mathfrak{C}(\!|ObjMap|\!)$) = 0 $\mathcal{D}_\circ$ (*dghm*-0 $\mathfrak{C}(\!|ArrMap|\!)$) = 0
by (*cs-concl* **cs-simp**: *dg-cs-simps dg*-0-*components*)+
show $\mathfrak{F}(\!|ObjMap|\!)$ = *dghm*-0 $\mathfrak{C}(\!|ObjMap|\!)$
by (*rule vsv-eqI*, *unfold dom-lhs dom-rhs*) (*auto intro*: *dg-cs-intros*)
show $\mathfrak{F}(\!|ArrMap|\!)$ = *dghm*-0 $\mathfrak{C}(\!|ArrMap|\!)$
by (*rule vsv-eqI*, *unfold dom-lhs dom-rhs*) (*auto intro*: *dg-cs-intros*)
qed *simp-all*

3.10.4 Empty transformation of digraph homomorphisms

Definition and elementary properties

definition *tdghm*-0 :: $V \Rightarrow V$
where *tdghm*-0 $\mathfrak{C}$ = [0, *dghm*-0 $\mathfrak{C}$, *dghm*-0 $\mathfrak{C}$, *dg*-0, $\mathfrak{C}$]$_\circ$

Components.

lemma *tdghm*-0-*components*:
shows *tdghm*-0 $\mathfrak{C}(\!|NTMap|\!)$ = 0
and [*dg-cs-simps*]: *tdghm*-0 $\mathfrak{C}(\!|NTDom|\!)$ = *dghm*-0 $\mathfrak{C}$
and [*dg-cs-simps*]: *tdghm*-0 $\mathfrak{C}(\!|NTCod|\!)$ = *dghm*-0 $\mathfrak{C}$
and [*dg-cs-simps*]: *tdghm*-0 $\mathfrak{C}(\!|NTDGDom|\!)$ = *dg*-0
and [*dg-cs-simps*]: *tdghm*-0 $\mathfrak{C}(\!|NTDGCod|\!)$ = $\mathfrak{C}$
unfolding *tdghm*-0-*def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Duality.

lemma *op-tdghm-tdghm*-0: *op-tdghm* (*tdghm*-0 $\mathfrak{C}$) = *tdghm*-0 (*op-dg* $\mathfrak{C}$)
by
(

simp-all add:
tdghm-0-*def op-tdghm-def op-dg-def dg*-0-*def op-dghm-dghm*-0
nt-field-simps dg-field-simps nat-omega-simps
)

Transformation map

lemma *tdghm*-0-*NTMap-vsv*[*dg-cs-intros*]: *vsv* (*tdghm*-0 $\mathfrak{C}$(|*NTMap*|))
unfolding *tdghm*-0-*components* **by** *simp*

lemma *tdghm*-0-*NTMap-vdomain*[*dg-cs-simps*]: $\mathcal{D}_\circ$ (*tdghm*-0 $\mathfrak{C}$(|*NTMap*|)) = 0
unfolding *tdghm*-0-*components* **by** *simp*

lemma *tdghm*-0-*NTMap-vrange*[*dg-cs-simps*]: $\mathcal{R}_\circ$ (*tdghm*-0 $\mathfrak{C}$(|*NTMap*|)) = 0
unfolding *tdghm*-0-*components* **by** *simp*

Empty transformation of digraph homomorphisms is a transformation of digraph homomorphisms

lemma (**in** *digraph*) *dg-tdghm*-0-*is-tdghmI*:
tdghm-0 $\mathfrak{C}$: *dghm*-0 $\mathfrak{C}$ $\mapsto_{DGHM}$ *dghm*-0 $\mathfrak{C}$: *dg*-0 $\mapsto\!\!\mapsto_{DG\alpha}$ $\mathfrak{C}$
proof(*intro is-tdghmI*)
show *vfsequence* (*tdghm*-0 $\mathfrak{C}$) **unfolding** *tdghm*-0-*def* **by** *simp*
show *vcard* (*tdghm*-0 $\mathfrak{C}$) = $5_{\mathbb{N}}$
unfolding *tdghm*-0-*def* **by** (*simp add*: *nat-omega-simps*)
show *tdghm*-0 $\mathfrak{C}$(|*NTMap*|)(|*a*|) : *dghm*-0 $\mathfrak{C}$(|*ObjMap*|)(|*a*|) $\mapsto_{\mathfrak{C}}$ *dghm*-0 $\mathfrak{C}$(|*ObjMap*|)(|*a*|)
if *a* $\in_\circ$ *dg*-0(|*Obj*|) **for** *a*
using *that* **unfolding** *dg*-0-*components* **by** *simp*
qed
(
cs-concl **cs-shallow**
cs-simp: *dg-cs-simps dg*-0-*components*(1) **cs-intro**: *dg-cs-intros*
)+

lemma (**in** *digraph*) *dg-tdghm*-0-*is-tdghmI'*[*dg-cs-intros*]:
assumes $\mathfrak{F}'$ = *dghm*-0 $\mathfrak{C}$
and $\mathfrak{G}'$ = *dghm*-0 $\mathfrak{C}$
and $\mathfrak{A}'$ = *dg*-0
and $\mathfrak{B}'$ = $\mathfrak{C}$
and $\mathfrak{F}'$ = $\mathfrak{F}$
and $\mathfrak{G}'$ = $\mathfrak{G}$
shows *tdghm*-0 $\mathfrak{C}$: $\mathfrak{F}'$ $\mapsto_{DGHM}$ $\mathfrak{G}'$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{DG\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule dg-tdghm*-0-*is-tdghmI*)

lemmas [*dg-cs-intros*] = *digraph.dg-tdghm*-0-*is-tdghmI'*

lemma *is-tdghm-is-tdghm*-0-*if-dg*-0:
assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{DGHM}$ $\mathfrak{G}$: *dg*-0 $\mapsto\!\!\mapsto_{DG\alpha}$ $\mathfrak{C}$
shows $\mathfrak{N}$ = *tdghm*-0 $\mathfrak{C}$ **and** $\mathfrak{F}$ = *dghm*-0 $\mathfrak{C}$ **and** $\mathfrak{G}$ = *dghm*-0 $\mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-tdghm* α *dg*-0 $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
show $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *dghm*-0 $\mathfrak{C}$ **and** $\mathfrak{G}$-*def*: $\mathfrak{G}$ = *dghm*-0 $\mathfrak{C}$
by (*auto intro*: *is-dghm-is-dghm*-0-*if-dg*-0 *dg-cs-intros*)
show $\mathfrak{N}$ = *tdghm*-0 $\mathfrak{C}$
proof(*rule tdghm-eqI*)
show $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{DGHM}$ $\mathfrak{G}$: *dg*-0 $\mapsto\!\!\mapsto_{DG\alpha}$ $\mathfrak{C}$ **by** (*rule assms*(1))
then have *dom-lhs*: $\mathcal{D}_\circ$ ($\mathfrak{N}$(|*NTMap*|)) = 0
by (*cs-concl* **cs-simp**: *dg-cs-simps dg*-0-*components*(1))

show *tdghm-0* $\mathfrak{C}$: *dghm-0* $\mathfrak{C} \mapsto_{DGHM}$ *dghm-0* $\mathfrak{C}$: *dg-0* $\mapsto\mapsto_{DG\alpha}$ $\mathfrak{C}$
by (*cs-concl* **cs-intro**: *dg-cs-intros*)
then have *dom-rhs*: $\mathcal{D}_\circ$ (*tdghm-0* $\mathfrak{C}(\!|NTMap|\!)$) = 0
by (*cs-concl* **cs-simp**: *dg-cs-simps dg-0-components*(1))
show $\mathfrak{N}(\!|NTMap|\!)$ = *tdghm-0* $\mathfrak{C}(\!|NTMap|\!)$
by (*rule vsv-eqI*, *unfold dom-lhs dom-rhs*) (*auto intro*: *dg-cs-intros*)
qed (*auto simp*: $\mathfrak{F}$-*def* $\mathfrak{G}$-*def*)
qed

3.10.5 10: digraph with one object and no arrows

Definition and elementary properties

definition *dg-10* :: *V* ⇒ *V*
where *dg-10* $\mathfrak{a}$ = [*set* {$\mathfrak{a}$}, 0, 0, 0]$_\circ$

Components.

lemma *dg-10-components*:
shows *dg-10* $\mathfrak{a}(\!|Obj|\!)$ = *set* {$\mathfrak{a}$}
and *dg-10* $\mathfrak{a}(\!|Arr|\!)$ = 0
and *dg-10* $\mathfrak{a}(\!|Dom|\!)$ = 0
and *dg-10* $\mathfrak{a}(\!|Cod|\!)$ = 0
unfolding *dg-10-def dg-field-simps* **by** (*auto simp*: *nat-omega-simps*)

10 is a digraph

lemma (**in** $\mathcal{Z}$) *digraph-dg-10*:
assumes $\mathfrak{a} \in_\circ$ *Vset* α
shows *digraph* α (*dg-10* $\mathfrak{a}$)
proof(*intro digraphI*)
show *vfsequence* (*dg-10* $\mathfrak{a}$) **unfolding** *dg-10-def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* (*dg-10* $\mathfrak{a}$) = $4_\mathbb{N}$ **unfolding** *dg-10-def* **by** (*simp add*: *nat-omega-simps*)
show ($\bigcup_\circ a' \in_\circ A.\ \bigcup_\circ b' \in_\circ B.$ *Hom* (*dg-10* $\mathfrak{a}$) *a'* *b'*) $\in_\circ$ *Vset* α **for** *A* *B*
proof–
have ($\bigcup_\circ a' \in_\circ A.\ \bigcup_\circ b' \in_\circ B.$ *Hom* (*dg-10* $\mathfrak{a}$) *a'* *b'*) $\subseteq_\circ$ *dg-10* $\mathfrak{a}(\!|Arr|\!)$ **by** *auto*
moreover have *dg-10* $\mathfrak{a}(\!|Arr|\!) \subseteq_\circ 0$ **unfolding** *dg-10-components* **by** *auto*
ultimately show *?thesis* **using** *vempty-is-zet vsubset-in-VsetI* **by** *presburger*
qed
qed (*auto simp*: *assms dg-10-components vsubset-vsingleton-leftI*)

Arrow with a domain and a codomain

lemma *dg-10-is-arr-iff*: $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}10\ \mathfrak{a}} \mathfrak{B} \longleftrightarrow$ *False*
unfolding *is-arr-def dg-10-components* **by** *simp*

3.10.6 1: digraph with one object and one arrow

Definition and elementary properties

definition *dg-1* :: *V* ⇒ *V* ⇒ *V*
where *dg-1* $\mathfrak{a}$ $\mathfrak{f}$ = [*set* {$\mathfrak{a}$}, *set* {$\mathfrak{f}$}, *set* {⟨$\mathfrak{f}$, $\mathfrak{a}$⟩}, *set* {⟨$\mathfrak{f}$, $\mathfrak{a}$⟩}]$_\circ$

Components.

lemma *dg-1-components*:
shows *dg-1* $\mathfrak{a}$ $\mathfrak{f}(\!|Obj|\!)$ = *set* {$\mathfrak{a}$}
and *dg-1* $\mathfrak{a}$ $\mathfrak{f}(\!|Arr|\!)$ = *set* {$\mathfrak{f}$}
and *dg-1* $\mathfrak{a}$ $\mathfrak{f}(\!|Dom|\!)$ = *set* {⟨$\mathfrak{f}$, $\mathfrak{a}$⟩}
and *dg-1* $\mathfrak{a}$ $\mathfrak{f}(\!|Cod|\!)$ = *set* {⟨$\mathfrak{f}$, $\mathfrak{a}$⟩}
unfolding *dg-1-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

1 is a digraph

lemma (**in** $\mathcal{Z}$) *digraph-dg-1*:
assumes $\mathfrak{a} \in_\circ$ *Vset* α **and** $\mathfrak{f} \in_\circ$ *Vset* α
shows *digraph* α (*dg-1* $\mathfrak{a}$ $\mathfrak{f}$)
proof(*intro digraphI*)
show *vfsequence* (*dg-1* $\mathfrak{a}$ $\mathfrak{f}$) **unfolding** *dg-1-def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* (*dg-1* $\mathfrak{a}$ $\mathfrak{f}$) = $4_{\mathbb{N}}$ **unfolding** *dg-1-def* **by** (*simp add*: *nat-omega-simps*)
show $(\bigcup_\circ a' \in_\circ A.\ \bigcup_\circ b' \in_\circ B.\ \textit{Hom}\ (\textit{dg-1}\ \mathfrak{a}\ \mathfrak{f})\ a'\ b') \in_\circ \textit{Vset}\ \alpha$ **for** *A B*
proof–
have $(\bigcup_\circ a' \in_\circ A.\ \bigcup_\circ b' \in_\circ B.\ \textit{Hom}\ (\textit{dg-1}\ \mathfrak{a}\ \mathfrak{f})\ a'\ b') \subseteq_\circ \textit{dg-1}\ \mathfrak{a}\ \mathfrak{f}(\!|\textit{Arr}|\!)$ **by** *auto*
moreover have *dg-1* $\mathfrak{a}$ $\mathfrak{f}(\!|\textit{Arr}|\!) \subseteq_\circ$ *set* $\{\mathfrak{f}\}$ **unfolding** *dg-1-components* **by** *auto*
moreover from *assms*(2) **have** *set* $\{\mathfrak{f}\} \in_\circ$ *Vset* α
by (*simp add*: *Limit-vsingleton-in-VsetI*)
ultimately show *?thesis*
unfolding *dg-1-components* **by** (*auto simp*: *vsubset-in-VsetI*)
qed
qed (*auto simp*: *assms dg-1-components vsubset-vsingleton-leftI*)

Arrow with a domain and a codomain

lemma *dg-1-is-arrI*:
assumes $a = \mathfrak{a}$ **and** $b = \mathfrak{a}$ **and** $f = \mathfrak{f}$
shows $f : a \mapsto_{\textit{dg-1}\ \mathfrak{a}\ \mathfrak{f}} b$
using *assms* **by** (*intro is-arrI*) (*auto simp*: *dg-1-components*)

lemma *dg-1-is-arrD*:
assumes $f : a \mapsto_{\textit{dg-1}\ \mathfrak{a}\ \mathfrak{f}} b$
shows $a = \mathfrak{a}$ **and** $b = \mathfrak{a}$ **and** $f = \mathfrak{f}$
using *assms* **by** (*all*‹*elim is-arrE*›) (*auto simp*: *dg-1-components*)

lemma *dg-1-is-arrE*:
assumes $f : a \mapsto_{\textit{dg-1}\ \mathfrak{a}\ \mathfrak{f}} b$
obtains $a = \mathfrak{a}$ **and** $b = \mathfrak{a}$ **and** $f = \mathfrak{f}$
using *assms* **by** (*elim is-arrE*) (*force simp*: *dg-1-components*)

lemma *dg-1-is-arr-iff*: $f : a \mapsto_{\textit{dg-1}\ \mathfrak{a}\ \mathfrak{f}} b \longleftrightarrow (a = \mathfrak{a} \wedge b = \mathfrak{a} \wedge f = \mathfrak{f})$
by (*rule iffI*; (*elim is-arrE*)*?*)
(*auto simp*: *dg-1-components intro*: *dg-1-is-arrI*)

3.11 *GRPH* as a digraph

3.11.1 Background

Conventionally, *GRPH* defined as a category of digraphs and digraph homomorphisms (e.g., see Chapter II-7 in [39]). However, there is little that can prevent one from exposing *GRPH* as a digraph and provide additional structure gradually later. Thus, in this section, α-*GRPH* is defined as a digraph of digraphs and digraph homomorphisms in V_α.

named-theorems *GRPH-cs-simps*
named-theorems *GRPH-cs-intros*

3.11.2 Definition and elementary properties

definition *dg-GRPH* :: $V \Rightarrow V$
  where *dg-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*$|\!))$
    ]$_\circ$

Components.

lemma *dg-GRPH-components*:
  shows *dg-GRPH* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *digraph* α $\mathfrak{C}\}$
    and *dg-GRPH* $\alpha(\!|$*Arr*$|\!)$ = *all-dghms* α
    and *dg-GRPH* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}{\in_\circ}$*all-dghms* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$
    and *dg-GRPH* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}{\in_\circ}$*all-dghms* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
  unfolding *dg-GRPH-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

3.11.3 Object

lemma *dg-GRPH-ObjI*:
  assumes *digraph* α $\mathfrak{A}$
  shows $\mathfrak{A}$ $\in_\circ$ *dg-GRPH* $\alpha(\!|$*Obj*$|\!)$
  using *assms* **unfolding** *dg-GRPH-components* **by** *auto*

lemma *dg-GRPH-ObjD*:
  assumes $\mathfrak{A}$ $\in_\circ$ *dg-GRPH* $\alpha(\!|$*Obj*$|\!)$
  shows *digraph* α $\mathfrak{A}$
  using *assms* **unfolding** *dg-GRPH-components* **by** *auto*

lemma *dg-GRPH-ObjE*:
  assumes $\mathfrak{A}$ $\in_\circ$ *dg-GRPH* $\alpha(\!|$*Obj*$|\!)$
  obtains *digraph* α $\mathfrak{A}$
  using *assms* **unfolding** *dg-GRPH-components* **by** *auto*

lemma *dg-GRPH-Obj-iff*[*GRPH-cs-simps*]:
  $\mathfrak{A}$ $\in_\circ$ *dg-GRPH* $\alpha(\!|$*Obj*$|\!)$ $\longleftrightarrow$ *digraph* α $\mathfrak{A}$
  unfolding *dg-GRPH-components* **by** *auto*

3.11.4 Domain

mk-VLambda *dg-GRPH-components*(3)
  |*vsv dg-GRPH-Dom-vsv*[*GRPH-cs-intros*]|
  |*vdomain dg-GRPH-Dom-vdomain*[*GRPH-cs-simps*]|
  |*app dg-GRPH-Dom-app*[*GRPH-cs-simps*]|

lemma *dg-GRPH-Dom-vrange*: $\mathcal{R}_\circ$ (*dg-GRPH* $\alpha(\!|Dom|\!)$) $\subseteq_\circ$ *dg-GRPH* $\alpha(\!|Obj|\!)$
unfolding *dg-GRPH-components* **by** (*rule vrange-VLambda-vsubset*) *auto*

3.11.5 Codomain

mk-VLambda *dg-GRPH-components*(4)
|*vsv dg-GRPH-Cod-vsv*[*GRPH-cs-intros*]|
|*vdomain dg-GRPH-Cod-vdomain*[*GRPH-cs-simps*]|
|*app dg-GRPH-Cod-app*[*GRPH-cs-simps*]|

lemma *dg-GRPH-Cod-vrange*: $\mathcal{R}_\circ$ (*dg-GRPH* $\alpha(\!|Cod|\!)$) $\subseteq_\circ$ *dg-GRPH* $\alpha(\!|Obj|\!)$
unfolding *dg-GRPH-components* **by** (*rule vrange-VLambda-vsubset*) *auto*

3.11.6 *GRPH* is a digraph

lemma (**in** $\mathcal{Z}$) *tiny-digraph-dg-GRPH*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *tiny-digraph* β (*dg-GRPH* α)
proof(*intro tiny-digraphI*)
show *vfsequence* (*dg-GRPH* α) **unfolding** *dg-GRPH-def* **by** *simp*
show *vcard* (*dg-GRPH* α) = $4_{\mathbb{N}}$
unfolding *dg-GRPH-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*dg-GRPH* $\alpha(\!|Dom|\!)$) $\subseteq_\circ$ *dg-GRPH* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-GRPH-Dom-vrange*)
show $\mathcal{R}_\circ$ (*dg-GRPH* $\alpha(\!|Cod|\!)$) $\subseteq_\circ$ *dg-GRPH* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-GRPH-Cod-vrange*)
show *dg-GRPH* $\alpha(\!|Obj|\!)$ $\in_\circ$ *Vset* β
unfolding *dg-GRPH-components* **by** (*rule digraphs-in-Vset*[*OF assms*])
show *dg-GRPH* $\alpha(\!|Arr|\!)$ $\in_\circ$ *Vset* β
unfolding *dg-GRPH-components* **by** (*rule all-dghms-in-Vset*[*OF assms*])
qed (*auto simp*: *assms dg-GRPH-components*)

3.11.7 Arrow with a domain and a codomain

lemma *dg-GRPH-is-arrI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{DG\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}GRPH\ \alpha} \mathfrak{B}$
proof(*intro is-arrI*; *unfold dg-GRPH-components*)
from *assms* **show** $\mathfrak{F} \in_\circ$ *all-dghms* α **by** *auto*
with *assms* **show**
$(\lambda\mathfrak{F}{\in_\circ}\textit{all-dghms}\ \alpha.\ \mathfrak{F}(\!|HomDom|\!))(\!|\mathfrak{F}|\!) = \mathfrak{A}$
$(\lambda\mathfrak{F}{\in_\circ}\textit{all-dghms}\ \alpha.\ \mathfrak{F}(\!|HomCod|\!))(\!|\mathfrak{F}|\!) = \mathfrak{B}$
by (*auto simp*: *GRPH-cs-simps*)
qed

lemma *dg-GRPH-is-arrD*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}GRPH\ \alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{DG\alpha} \mathfrak{B}$
using *assms* **by** (*elim is-arrE*) (*auto simp*: *dg-GRPH-components*)

lemma *dg-GRPH-is-arrE*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}GRPH\ \alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{DG\alpha} \mathfrak{B}$
using *assms* **by** (*simp add*: *dg-GRPH-is-arrD*)

lemma *dg-GRPH-is-arr-iff*[*GRPH-cs-simps*]:
$\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}GRPH\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{DG\alpha} \mathfrak{B}$
by (*auto intro*: *dg-GRPH-is-arrI dest*: *dg-GRPH-is-arrD*)

3.12 *Rel* as a digraph

3.12.1 Background

Rel is usually defined as a category of sets and binary relations (e.g., see Chapter I-7 in [39]). However, there is little that can prevent one from exposing *Rel* as a digraph and provide additional structure gradually later. Thus, in this section, α-*Rel* is defined as a digraph of sets and binary relations in V_α.

named-theorems *dg-Rel-shared-cs-simps*
named-theorems *dg-Rel-shared-cs-intros*

named-theorems *dg-Rel-cs-simps*
named-theorems *dg-Rel-cs-intros*

3.12.2 Canonical arrow for V

named-theorems *arr-field-simps*

definition *ArrVal* :: V **where** [*arr-field-simps*]: *ArrVal* = 0
definition *ArrDom* :: V **where** [*arr-field-simps*]: *ArrDom* = $1_{\mathbb{N}}$
definition *ArrCod* :: V **where** [*arr-field-simps*]: *ArrCod* = $2_{\mathbb{N}}$

lemma *ArrVal-eq-helper*:
assumes $f = g$
shows $f(\!|ArrVal|\!)(\!|a|\!) = g(\!|ArrVal|\!)(\!|a|\!)$
using *assms* **by** *simp*

3.12.3 Arrow for *Rel*

Definition and elementary properties

locale *arr-Rel* = $\mathcal{Z}$ α + *vfsequence* T + *ArrVal*: *vbrelation* ‹$T(\!|ArrVal|\!)$› **for** α T +
assumes *arr-Rel-length*[*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
vcard $T = 3_{\mathbb{N}}$
and *arr-Rel-ArrVal-vdomain*: $\mathcal{D}_\circ$ ($T(\!|ArrVal|\!)$) $\subseteq_\circ$ $T(\!|ArrDom|\!)$
and *arr-Rel-ArrVal-vrange*: $\mathcal{R}_\circ$ ($T(\!|ArrVal|\!)$) $\subseteq_\circ$ $T(\!|ArrCod|\!)$
and *arr-Rel-ArrDom-in-Vset*: $T(\!|ArrDom|\!)$ $\in_\circ$ *Vset* α
and *arr-Rel-ArrCod-in-Vset*: $T(\!|ArrCod|\!)$ $\in_\circ$ *Vset* α

lemmas [*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*] = *arr-Rel.arr-Rel-length*

Components.

lemma *arr-Rel-components*[*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
shows $[f, A, B]_\circ(\!|ArrVal|\!) = f$
and $[f, A, B]_\circ(\!|ArrDom|\!) = A$
and $[f, A, B]_\circ(\!|ArrCod|\!) = B$
unfolding *arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Rules.

lemma (**in** *arr-Rel*) *arr-Rel-axioms'*[*dg-cs-intros*, *dg-Rel-cs-intros*]:
assumes $\alpha' = \alpha$
shows *arr-Rel* α' T
unfolding *assms* **by** (*rule arr-Rel-axioms*)

mk-ide rf *arr-Rel-def*[*unfolded arr-Rel-axioms-def*]
|*intro arr-RelI*|
|*dest arr-RelD*[*dest*]|

|*elim arr-RelE*[*elim*!]|

lemma (**in** $\mathcal{Z}$) *arr-Rel-vfsequenceI*:
assumes *vbrelation r*
and $\mathcal{D}_\circ\ r \subseteq_\circ a$
and $\mathcal{R}_\circ\ r \subseteq_\circ b$
and $a \in_\circ Vset\ \alpha$
and $b \in_\circ Vset\ \alpha$
shows *arr-Rel* $\alpha\ [r, a, b]_\circ$
by (*intro arr-RelI*)
(*insert assms*, *auto simp*: *nat-omega-simps arr-Rel-components*)

Elementary properties.

lemma *arr-Rel-eqI*:
assumes *arr-Rel* $\alpha\ S$
and *arr-Rel* $\alpha\ T$
and $S(\!|ArrVal|\!) = T(\!|ArrVal|\!)$
and $S(\!|ArrDom|\!) = T(\!|ArrDom|\!)$
and $S(\!|ArrCod|\!) = T(\!|ArrCod|\!)$
shows $S = T$
proof–
interpret *S*: *arr-Rel* $\alpha\ S$ **by** (*rule assms*(1))
interpret *T*: *arr-Rel* $\alpha\ T$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule vsv-eqI*)
show $\mathcal{D}_\circ\ S = \mathcal{D}_\circ\ T$
by (*simp add*: *S.vfsequence-vdomain T.vfsequence-vdomain dg-Rel-cs-simps*)
have *dom-lhs*: $\mathcal{D}_\circ\ S = 3_{\mathbb{N}}$
by (*simp add*: *S.vfsequence-vdomain dg-Rel-cs-simps*)
show $a \in_\circ \mathcal{D}_\circ\ S \Longrightarrow S(\!|a|\!) = T(\!|a|\!)$ **for** *a*
by (*unfold dom-lhs*, *elim-in-numeral*, *insert assms*)
(*auto simp*: *arr-field-simps*)
qed *auto*
qed

lemma (**in** *arr-Rel*) *arr-Rel-def*: $T = [T(\!|ArrVal|\!),\ T(\!|ArrDom|\!),\ T(\!|ArrCod|\!)]_\circ$
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ\ T = 3_{\mathbb{N}}$ **by** (*simp add*: *vfsequence-vdomain dg-Rel-cs-simps*)
have *dom-rhs*: $\mathcal{D}_\circ\ [T(\!|ArrVal|\!),\ T(\!|ArrDom|\!),\ T(\!|ArrCod|\!)]_\circ = 3_{\mathbb{N}}$
by (*simp add*: *nat-omega-simps*)
then show $\mathcal{D}_\circ\ T = \mathcal{D}_\circ\ [T(\!|ArrVal|\!),\ T(\!|ArrDom|\!),\ T(\!|ArrCod|\!)]_\circ$
unfolding *dom-lhs dom-rhs* **by** *simp*
show $a \in_\circ \mathcal{D}_\circ\ T \Longrightarrow T(\!|a|\!) = [T(\!|ArrVal|\!),\ T(\!|ArrDom|\!),\ T(\!|ArrCod|\!)]_\circ(\!|a|\!)$ **for** *a*
unfolding *dom-lhs*
by *elim-in-numeral* (*simp-all add*: *arr-field-simps nat-omega-simps*)
qed (*auto simp*: *vsv-axioms*)

Size.

lemma (**in** *arr-Rel*) *arr-Rel-ArrVal-in-Vset*: $T(\!|ArrVal|\!) \in_\circ Vset\ \alpha$
proof–
from *arr-Rel-ArrVal-vdomain arr-Rel-ArrDom-in-Vset* **have**
$\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) \in_\circ Vset\ \alpha$
by *auto*
moreover from *arr-Rel-ArrVal-vrange arr-Rel-ArrCod-in-Vset* **have**
$\mathcal{R}_\circ\ (T(\!|ArrVal|\!)) \in_\circ Vset\ \alpha$
by *auto*
ultimately show $T(\!|ArrVal|\!) \in_\circ Vset\ \alpha$
by (*simp add*: *ArrVal.vbrelation-Limit-in-VsetI*)

qed

lemma (**in** *arr-Rel*) *arr-Rel-in-Vset*: $T \in_{\circ} Vset\ \alpha$
proof–
note [*dg-Rel-cs-intros*] =
arr-Rel-ArrVal-in-Vset arr-Rel-ArrDom-in-Vset arr-Rel-ArrCod-in-Vset
show *?thesis*
by (*subst arr-Rel-def*)
(*cs-concl* **cs-shallow cs-intro**: *dg-Rel-cs-intros V-cs-intros*)
qed

lemma *small-arr-Rel*[*simp*]: *small* $\{T.\ arr\text{-}Rel\ \alpha\ T\}$
by (*rule down*[*of* - ‹*Vset* α›]) (*auto intro*!: *arr-Rel.arr-Rel-in-Vset*)

Other elementary properties.

lemma *set-Collect-arr-Rel*[*simp*]:
$x \in_{\circ} set\ (Collect\ (arr\text{-}Rel\ \alpha)) \longleftrightarrow arr\text{-}Rel\ \alpha\ x$
by *auto*

lemma (**in** *arr-Rel*) *arr-Rel-ArrVal-vsubset-ArrDom-ArrCod*:
$T(\!|ArrVal|\!) \subseteq_{\circ} T(\!|ArrDom|\!) \times_{\circ} T(\!|ArrCod|\!)$
proof
fix *ab* **assume** $ab \in_{\circ} T(\!|ArrVal|\!)$
then obtain *a b* **where** $ab = \langle a, b \rangle$
and $a \in_{\circ} \mathcal{D}_{\circ}\ (T(\!|ArrVal|\!))$
and $b \in_{\circ} \mathcal{R}_{\circ}\ (T(\!|ArrVal|\!))$
by (*blast elim*: *ArrVal.vbrelation-vinE*)
with *arr-Rel-ArrVal-vdomain arr-Rel-ArrVal-vrange* **show**
$ab \in_{\circ} T(\!|ArrDom|\!) \times_{\circ} T(\!|ArrCod|\!)$
by *auto*
qed

Composition

See Chapter I-7 in [39].

definition *comp-Rel* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\circ_{Rel}$› 55)
where *comp-Rel S T* = $[S(\!|ArrVal|\!) \circ_{\circ} T(\!|ArrVal|\!),\ T(\!|ArrDom|\!),\ S(\!|ArrCod|\!)]_{\circ}$

Components.

lemma *comp-Rel-components*:
shows $(S \circ_{Rel} T)(\!|ArrVal|\!) = S(\!|ArrVal|\!) \circ_{\circ} T(\!|ArrVal|\!)$
and [*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
$(S \circ_{Rel} T)(\!|ArrDom|\!) = T(\!|ArrDom|\!)$
and [*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
$(S \circ_{Rel} T)(\!|ArrCod|\!) = S(\!|ArrCod|\!)$
unfolding *comp-Rel-def arr-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Elementary properties.

lemma *comp-Rel-vsv*[*dg-Rel-shared-cs-intros*, *dg-Rel-cs-intros*]:
vsv $(S \circ_{Rel} T)$
unfolding *comp-Rel-def* **by** *auto*

lemma *arr-Rel-comp-Rel*[*dg-Rel-cs-intros*]:
assumes *arr-Rel* α *S* **and** *arr-Rel* α *T*
shows *arr-Rel* α $(S \circ_{Rel} T)$
proof–
interpret *S*: *arr-Rel* α *S* **by** (*rule assms*(1))

```
  interpret T: arr-Rel α T by (rule assms(2))
  show ?thesis
  proof(intro arr-RelI)
    show vfsequence (S ∘Rel T) unfolding comp-Rel-def by simp
    show vcard (S ∘Rel T) = 3ℕ
      unfolding comp-Rel-def by (simp add: nat-omega-simps)
    from T.arr-Rel-ArrVal-vdomain show
      𝒟ₒ ((S ∘Rel T)(|ArrVal|)) ⊆ₒ (S ∘Rel T)(|ArrDom|)
      unfolding comp-Rel-components by auto
    show ℛₒ ((S ∘Rel T)(|ArrVal|)) ⊆ₒ (S ∘Rel T)(|ArrCod|)
      unfolding comp-Rel-components
    proof(intro vsubsetI)
      fix z assume z ∈ₒ ℛₒ (S(|ArrVal|) ∘ₒ T(|ArrVal|))
      then obtain x y where ⟨y, z⟩ ∈ₒ S(|ArrVal|) and ⟨x, y⟩ ∈ₒ T(|ArrVal|)
        by (meson vcomp-obtain-middle vrange-iff-vdomain)
      with S.arr-Rel-ArrVal-vrange show z ∈ₒ S(|ArrCod|) by auto
    qed
  qed
  (
    auto simp:
      comp-Rel-components T.arr-Rel-ArrDom-in-Vset S.arr-Rel-ArrCod-in-Vset
  )
qed

lemma arr-Rel-comp-Rel-assoc[dg-Rel-shared-cs-simps, dg-Rel-cs-simps]:
  (H ∘Rel G) ∘Rel F = H ∘Rel (G ∘Rel F)
  by (simp add: comp-Rel-def vcomp-assoc arr-field-simps nat-omega-simps)
```

Inclusion arrow

The definition of the inclusion arrow is based on the concept of the inclusion map, e.g., see [5][4]

```
definition incl-Rel A B = [vid-on A, A, B]ₒ
```

Components.

```
lemma incl-Rel-components:
  shows incl-Rel A B(|ArrVal|) = vid-on A
    and [dg-Rel-shared-cs-simps, dg-Rel-cs-simps]: incl-Rel A B(|ArrDom|) = A
    and [dg-Rel-shared-cs-simps, dg-Rel-cs-simps]: incl-Rel A B(|ArrCod|) = B
  unfolding incl-Rel-def arr-field-simps by (simp-all add: nat-omega-simps)
```

Arrow value.

```
lemma incl-Rel-ArrVal-vsv[dg-Rel-shared-cs-intros, dg-Rel-cs-intros]:
  vsv (incl-Rel A B(|ArrVal|))
  unfolding incl-Rel-components by simp

lemma incl-Rel-ArrVal-vdomain[dg-Rel-shared-cs-simps, dg-Rel-cs-simps]:
  𝒟ₒ (incl-Rel A B(|ArrVal|)) = A
  unfolding incl-Rel-components by simp

lemma incl-Rel-ArrVal-app[dg-Rel-shared-cs-simps, dg-Rel-cs-simps]:
  assumes a ∈ₒ A
  shows incl-Rel A B(|ArrVal|)(|a|) = a
  using assms unfolding incl-Rel-components by simp
```

Elementary properties.

[4] https://en.wikipedia.org/wiki/Inclusion_map

lemma *incl-Rel-vfsequence*[*dg-Rel-shared-cs-intros*, *dg-Rel-cs-intros*]:
vfsequence (*incl-Rel A B*)
unfolding *incl-Rel-def* **by** *simp*

lemma *incl-Rel-vcard*[*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
vcard (*incl-Rel A B*) = $3_{\mathbb{N}}$
unfolding *incl-Rel-def incl-Rel-def* **by** (*simp add*: *nat-omega-simps*)

lemma (**in** $\mathcal{Z}$) *arr-Rel-incl-RelI*:
assumes $A \in_{\circ}$ *Vset* α **and** $B \in_{\circ}$ *Vset* α **and** $A \subseteq_{\circ} B$
shows *arr-Rel* α (*incl-Rel A B*)
proof(*intro arr-RelI*)
show *vfsequence* (*incl-Rel A B*) **unfolding** *incl-Rel-def* **by** *simp*
show *vcard* (*incl-Rel A B*) = $3_{\mathbb{N}}$
unfolding *incl-Rel-def* **by** (*simp add*: *nat-omega-simps*)
qed (*auto simp*: *incl-Rel-components assms*)

Identity

See Chapter I-7 in [39].

definition *id-Rel* :: $V \Rightarrow V$
where *id-Rel A* = *incl-Rel A A*

Components.

lemma *id-Rel-components*:
shows *id-Rel A*(|*ArrVal*|) = *vid-on A*
and [*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]: *id-Rel A*(|*ArrDom*|) = *A*
and [*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]: *id-Rel A*(|*ArrCod*|) = *A*
unfolding *id-Rel-def incl-Rel-components* **by** *simp-all*

Elementary properties.

lemma *id-Rel-vfsequence*[*dg-Rel-shared-cs-intros*, *dg-Rel-cs-intros*]:
vfsequence (*id-Rel A*)
unfolding *id-Rel-def* **by** (*simp add*: *dg-Rel-cs-intros*)

lemma *id-Rel-vcard*[*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
vcard (*id-Rel A*) = $3_{\mathbb{N}}$
unfolding *id-Rel-def* **by** (*simp add*: *dg-Rel-cs-simps*)

lemma (**in** $\mathcal{Z}$) *arr-Rel-id-RelI*:
assumes $A \in_{\circ}$ *Vset* α
shows *arr-Rel* α (*id-Rel A*)
by (*intro arr-RelI*)
(*auto simp*: *id-Rel-components*(1) *assms dg-Rel-cs-intros dg-Rel-cs-simps*)

lemma *id-Rel-ArrVal-app*[*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
assumes $a \in_{\circ} A$
shows *id-Rel A*(|*ArrVal*|)(|*a*|) = *a*
using *assms* **unfolding** *id-Rel-components* **by** *simp*

lemma *arr-Rel-comp-Rel-id-Rel-left*[*dg-Rel-cs-simps*]:
assumes *arr-Rel* α *F* **and** *F*(|*ArrCod*|) = *A*
shows *id-Rel A* $\circ_{Rel}$ *F* = *F*
proof(*rule arr-Rel-eqI* [*of* α])
interpret *F*: *arr-Rel* α *F* **by** (*rule assms*(1))
from *assms*(2) **have** $A \in_{\circ}$ *Vset* α **by** (*auto intro*: *F.arr-Rel-ArrCod-in-Vset*)
with *assms*(1) **show** *arr-Rel* α (*id-Rel A* $\circ_{Rel}$ *F*)

```
    by (blast intro: F.arr-Rel-id-RelI intro!: arr-Rel-comp-Rel)
  from assms(2) F.arr-Rel-ArrVal-vrange show
    (id-Rel A ∘_Rel F)(|ArrVal|) = F(|ArrVal|)
    unfolding comp-Rel-components id-Rel-components by auto
qed
  (
    use assms(2) in
      ‹auto simp: assms(1) comp-Rel-components id-Rel-components›
  )

lemma arr-Rel-comp-Rel-id-Rel-right[dg-Rel-cs-simps]:
  assumes arr-Rel α F and F(|ArrDom|) = A
  shows F ∘_Rel id-Rel A = F
proof(rule arr-Rel-eqI[of α])
  interpret F: arr-Rel α F by (rule assms(1))
  from assms(2) have A ∈∘ Vset α by (auto intro: F.arr-Rel-ArrDom-in-Vset)
  with assms(1) show arr-Rel α (F ∘_Rel id-Rel A)
    by (blast intro: F.arr-Rel-id-RelI intro!: arr-Rel-comp-Rel)
  show arr-Rel α F by (simp add: assms(1))
  from assms(2) F.arr-Rel-ArrVal-vdomain show
    (F ∘_Rel id-Rel A)(|ArrVal|) = F(|ArrVal|)
    unfolding comp-Rel-components id-Rel-components by auto
qed (use assms(2) in ‹auto simp: comp-Rel-components id-Rel-components›)
```

Converse

As mentioned in Chapter I-7 in [39], the category *Rel* is usually equipped with an additional structure that is the operation of taking a converse of a relation. The operation is meant to be used almost exclusively as part of the dagger functor for *Rel*.

```
definition converse-Rel :: V ⇒ V (‹(-⁻¹_Rel)› [1000] 999)
  where converse-Rel T = [(T(|ArrVal|))⁻¹∘, T(|ArrCod|), T(|ArrDom|)]∘

lemma converse-Rel-components:
  shows T⁻¹_Rel(|ArrVal|) = (T(|ArrVal|))⁻¹∘
    and [dg-Rel-shared-cs-simps, dg-Rel-cs-simps]: T⁻¹_Rel(|ArrDom|) = T(|ArrCod|)
    and [dg-Rel-shared-cs-simps, dg-Rel-cs-simps]: T⁻¹_Rel(|ArrCod|) = T(|ArrDom|)
  unfolding converse-Rel-def arr-field-simps by (simp-all add: nat-omega-simps)
```

Elementary properties.

```
lemma (in arr-Rel) arr-Rel-converse-Rel: arr-Rel α (T⁻¹_Rel)
proof(rule arr-RelI, unfold converse-Rel-components)
  show vfsequence (T⁻¹_Rel) unfolding converse-Rel-def by simp
  show vcard (T⁻¹_Rel) = 3ℕ
    unfolding converse-Rel-def by (simp add: nat-omega-simps)
qed
  (
    auto simp:
      converse-Rel-components(1)
      arr-Rel-ArrDom-in-Vset
      arr-Rel-ArrCod-in-Vset
      arr-Rel-ArrVal-vdomain
      arr-Rel-ArrVal-vrange
  )

lemmas [dg-Rel-cs-intros] =
  arr-Rel.arr-Rel-converse-Rel
```

lemma (**in** *arr-Rel*)
arr-Rel-converse-Rel-converse-Rel[*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*]:
$(T^{-1}{}_{Rel})^{-1}{}_{Rel} = T$
proof(*rule arr-Rel-eqI*)
from *arr-Rel-axioms* **show** *arr-Rel* α $((T^{-1}{}_{Rel})^{-1}{}_{Rel})$
by (*cs-intro-step dg-Rel-cs-intros*)+
qed (*simp-all add*: *arr-Rel-axioms converse-Rel-components*)

lemmas [*dg-Rel-cs-simps*] =
arr-Rel.arr-Rel-converse-Rel-converse-Rel

lemma *arr-Rel-converse-Rel-eq-iff*[*dg-Rel-cs-simps*]:
assumes *arr-Rel* α F **and** *arr-Rel* α G
shows $F^{-1}{}_{Rel} = G^{-1}{}_{Rel} \longleftrightarrow F = G$
proof(*rule iffI*)
show $F^{-1}{}_{Rel} = G^{-1}{}_{Rel} \Longrightarrow F = G$
by (*metis arr-Rel.arr-Rel-converse-Rel-converse-Rel assms*)
qed *simp*

lemma *arr-Rel-converse-Rel-comp-Rel*[*dg-Rel-cs-simps*]:
assumes *arr-Rel* α G **and** *arr-Rel* α F
shows $(F \circ_{Rel} G)^{-1}{}_{Rel} = G^{-1}{}_{Rel} \circ_{Rel} F^{-1}{}_{Rel}$
proof(*rule arr-Rel-eqI*, *unfold converse-Rel-components comp-Rel-components*)
from *assms* **show** *arr-Rel* α $(G^{-1}{}_{Rel} \circ_{Rel} F^{-1}{}_{Rel})$
by (*cs-concl* **cs-shallow cs-intro**: *dg-Rel-cs-intros*)
from *assms* **show** *arr-Rel* α $((F \circ_{Rel} G)^{-1}{}_{Rel})$
by (*cs-intro-step dg-Rel-cs-intros*)+
qed (*simp-all add*: *vconverse-vcomp*)

lemma (**in** $\mathcal{Z}$) *arr-Rel-converse-Rel-id-Rel*:
assumes $c \in_\circ$ *Vset* α
shows *arr-Rel* α $((id\text{-}Rel\ c)^{-1}{}_{Rel})$
using *assms* $\mathcal{Z}$*-axioms*
by (*cs-concl* **cs-shallow cs-intro**: *dg-Rel-cs-intros arr-Rel-id-RelI*)+

lemma (**in** $\mathcal{Z}$) *arr-Rel-converse-Rel-id-Rel-eq-id-Rel*[
dg-Rel-shared-cs-simps, *dg-Rel-cs-simps*
]:
assumes $c \in_\circ$ *Vset* α
shows $(id\text{-}Rel\ c)^{-1}{}_{Rel} = id\text{-}Rel\ c$
by (*rule arr-Rel-eqI*[*of* α], *unfold converse-Rel-components id-Rel-components*)
(*simp-all add*: *assms arr-Rel-id-RelI arr-Rel-converse-Rel-id-Rel*)

lemmas [*dg-Rel-shared-cs-simps*, *dg-Rel-cs-simps*] =
$\mathcal{Z}$*.arr-Rel-converse-Rel-id-Rel-eq-id-Rel*

lemma *arr-Rel-comp-Rel-converse-Rel-left-if-v11*[*dg-Rel-cs-simps*]:
assumes *arr-Rel* α T
and $\mathcal{D}_\circ$ $(T(\!|ArrVal|\!)) = A$
and $T(\!|ArrDom|\!) = A$
and *v11* $(T(\!|ArrVal|\!))$
and $A \in_\circ$ *Vset* α
shows $T^{-1}{}_{Rel} \circ_{Rel} T = id\text{-}Rel\ A$
proof–
interpret T: *arr-Rel* α T **by** (*rule assms*(1))
interpret *v11*: *v11* ‹$T(\!|ArrVal|\!)$› **by** (*rule assms*(4))
show *?thesis*
by (*rule arr-Rel-eqI*[*of* α])

```
    (
      auto simp:
        converse-Rel-components
        comp-Rel-components
        id-Rel-components
        assms(1,3,5)
        arr-Rel.arr-Rel-converse-Rel
        arr-Rel-comp-Rel
        T.arr-Rel-id-RelI
        v11.v11-vcomp-vconverse[unfolded assms(2)]
    )
qed

lemma arr-Rel-comp-Rel-converse-Rel-right-if-v11[dg-Rel-cs-simps]:
  assumes arr-Rel α T
    and ℛ∘ (T(|ArrVal|)) = A
    and T(|ArrCod|) = A
    and v11 (T(|ArrVal|))
    and A ∈∘ Vset α
  shows T ∘Rel T⁻¹Rel = id-Rel A
proof–
  interpret T: arr-Rel α T by (rule assms(1))
  interpret v11: v11 ‹T(|ArrVal|)› by (rule assms(4))
  show ?thesis
    by (rule arr-Rel-eqI[of α])
      (
        auto simp:
          assms(1,3,5)
          comp-Rel-components
          converse-Rel-components
          id-Rel-components
          v11.v11-vcomp-vconverse'[unfolded assms(2)]
          T.arr-Rel-id-RelI
          arr-Rel.arr-Rel-converse-Rel
          arr-Rel-comp-Rel
      )
qed
```

3.12.4 *Rel* as a digraph

Definition and elementary properties

```
definition dg-Rel :: V ⇒ V
  where dg-Rel α =
    [
      Vset α,
      set {T. arr-Rel α T},
      (λT∈∘set {T. arr-Rel α T}. T(|ArrDom|)),
      (λT∈∘set {T. arr-Rel α T}. T(|ArrCod|))
    ]∘
```

Components.

```
lemma dg-Rel-components:
  shows dg-Rel α(|Obj|) = Vset α
    and dg-Rel α(|Arr|) = set {T. arr-Rel α T}
    and dg-Rel α(|Dom|) = (λT∈∘set {T. arr-Rel α T}. T(|ArrDom|))
    and dg-Rel α(|Cod|) = (λT∈∘set {T. arr-Rel α T}. T(|ArrCod|))
  unfolding dg-Rel-def dg-field-simps by (simp-all add: nat-omega-simps)
```

Object

lemma *dg-Rel-Obj-iff*: $x \in_{\circ} \textit{dg-Rel}\ \alpha(\!|\textit{Obj}|\!) \longleftrightarrow x \in_{\circ} \textit{Vset}\ \alpha$
unfolding *dg-Rel-components* **by** *auto*

Arrow

lemma *dg-Rel-Arr-iff*[*dg-Rel-cs-simps*]: $x \in_{\circ} \textit{dg-Rel}\ \alpha(\!|\textit{Arr}|\!) \longleftrightarrow \textit{arr-Rel}\ \alpha\ x$
unfolding *dg-Rel-components* **by** *auto*

Domain

mk-VLambda *dg-Rel-components*(3)
|*vsv dg-Rel-Dom-vsv*[*dg-Rel-cs-intros*]|
|*vdomain dg-Rel-Dom-vdomain*[*dg-Rel-cs-simps*]|
|*app dg-Rel-Dom-app*[*unfolded set-Collect-arr-Rel*, *dg-Rel-cs-simps*]|

lemma *dg-Rel-Dom-vrange*: $\mathcal{R}_{\circ}\ (\textit{dg-Rel}\ \alpha(\!|\textit{Dom}|\!)) \subseteq_{\circ} \textit{dg-Rel}\ \alpha(\!|\textit{Obj}|\!)$
unfolding *dg-Rel-components*
by (*rule vrange-VLambda-vsubset*, *unfold set-Collect-arr-Rel*) *auto*

Codomain

mk-VLambda *dg-Rel-components*(4)
|*vsv dg-Rel-Cod-vsv*[*dg-Rel-cs-intros*]|
|*vdomain dg-Rel-Cod-vdomain*[*dg-Rel-cs-simps*]|
|*app dg-Rel-Cod-app*[*unfolded set-Collect-arr-Rel*, *dg-Rel-cs-simps*]|

lemma *dg-Rel-Cod-vrange*: $\mathcal{R}_{\circ}\ (\textit{dg-Rel}\ \alpha(\!|\textit{Cod}|\!)) \subseteq_{\circ} \textit{dg-Rel}\ \alpha(\!|\textit{Obj}|\!)$
unfolding *dg-Rel-components*
by (*rule vrange-VLambda-vsubset*, *unfold set-Collect-arr-Rel*) *auto*

Arrow with a domain and a codomain

Rules.

lemma *dg-Rel-is-arrI*[*dg-Rel-cs-intros*]:
assumes *arr-Rel* $\alpha\ S$ **and** $S(\!|\textit{ArrDom}|\!) = A$ **and** $S(\!|\textit{ArrCod}|\!) = B$
shows $S : A \mapsto_{\textit{dg-Rel}\ \alpha} B$
using *assms* **by** (*intro is-arrI*, *unfold dg-Rel-components*) *simp-all*

lemma *dg-Rel-is-arrD*:
assumes $S : A \mapsto_{\textit{dg-Rel}\ \alpha} B$
shows *arr-Rel* $\alpha\ S$
and [*dg-cs-simps*]: $S(\!|\textit{ArrDom}|\!) = A$
and [*dg-cs-simps*]: $S(\!|\textit{ArrCod}|\!) = B$
using *is-arrD*[*OF assms*] **unfolding** *dg-Rel-components* **by** *simp-all*

lemma *dg-Rel-is-arrE*:
assumes $S : A \mapsto_{\textit{dg-Rel}\ \alpha} B$
obtains *arr-Rel* $\alpha\ S$ **and** $S(\!|\textit{ArrDom}|\!) = A$ **and** $S(\!|\textit{ArrCod}|\!) = B$
using *is-arrD*[*OF assms*] **unfolding** *dg-Rel-components* **by** *simp-all*

Elementary properties.

lemma (**in** $\mathcal{Z}$) *dg-Rel-incl-Rel-is-arr*:
assumes $A \in_{\circ} \textit{Vset}\ \alpha$ **and** $B \in_{\circ} \textit{Vset}\ \alpha$ **and** $A \subseteq_{\circ} B$
shows $\textit{incl-Rel}\ A\ B : A \mapsto_{\textit{dg-Rel}\ \alpha} B$
proof(*rule dg-Rel-is-arrI*)
show *arr-Rel* α (*incl-Rel* $A\ B$) **by** (*intro arr-Rel-incl-RelI assms*)

qed (*simp-all add*: *incl-Rel-components*)

lemma (**in** $\mathcal{Z}$) *dg-Rel-incl-Rel-is-arr'*[*dg-Rel-cs-intros*]:
assumes $A \in_\circ Vset\ \alpha$
and $B \in_\circ Vset\ \alpha$
and $A \subseteq_\circ B$
and $A' = A$
and $B' = B$
shows $incl\text{-}Rel\ A\ B : A' \mapsto_{dg\text{-}Rel\ \alpha} B'$
using *assms*(1–3) **unfolding** *assms*(4,5) **by** (*rule dg-Rel-incl-Rel-is-arr*)

lemmas [*dg-Rel-cs-intros*] = $\mathcal{Z}$.*dg-Rel-incl-Rel-is-arr'*

lemma *dg-Rel-is-arr-ArrValE*:
assumes $T : A \mapsto_{dg\text{-}Rel\ \alpha} B$ **and** $ab \in_\circ T(\!|ArrVal|\!)$
obtains *a b*
where $ab = \langle a, b \rangle$ **and** $a \in_\circ \mathcal{D}_\circ\ (T(\!|ArrVal|\!))$ **and** $b \in_\circ \mathcal{R}_\circ\ (T(\!|ArrVal|\!))$
proof–
note $T = dg\text{-}Rel\text{-}is\text{-}arrD[OF\ assms(1)]$
then interpret *T*: *arr-Rel* $\alpha\ T$
rewrites $T(\!|ArrDom|\!) = A$ **and** $T(\!|ArrCod|\!) = B$
by *simp-all*
from *assms*(2) **obtain** *a b*
where $ab = \langle a, b \rangle$ **and** $a \in_\circ \mathcal{D}_\circ\ (T(\!|ArrVal|\!))$ **and** $b \in_\circ \mathcal{R}_\circ\ (T(\!|ArrVal|\!))$
by (*blast elim*: *T.ArrVal.vbrelation-vinE*)
with *that* **show** *?thesis* **by** *simp*
qed

Rel is a digraph

lemma (**in** $\mathcal{Z}$) *dg-Rel-Hom-vifunion-in-Vset*:
assumes $X \in_\circ Vset\ \alpha$ **and** $Y \in_\circ Vset\ \alpha$
shows $(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B) \in_\circ Vset\ \alpha$
proof–
define *Q* **where**
$Q\ i = (if\ i = 0\ then\ VPow\ (\bigcup_\circ X \times_\circ \bigcup_\circ Y)\ else\ if\ i = 1_{\mathbb{N}}\ then\ X\ else\ Y)$
for *i*
have
$\{[r, A, B]_\circ \mid r\ A\ B.\ r \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y \wedge A \in_\circ X \wedge B \in_\circ Y\} \subseteq$
$elts\ (\prod_\circ i \in_\circ set\ \{0, 1_{\mathbb{N}}, 2_{\mathbb{N}}\}.\ Q\ i)$
proof(*intro subsetI*, *unfold mem-Collect-eq*, *elim exE conjE*)
fix *F r A B* **assume** *prems*:
$F = [r, A, B]_\circ$
$r \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y$
$A \in_\circ X$
$B \in_\circ Y$
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*: *three prems*(1) *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*
then show $F(\!|i|\!) \in_\circ Q\ i$ **by** *cases* (*auto simp*: *Q-def prems nat-omega-simps*)
qed (*auto simp*: *prems*(1))
qed
moreover then have *small*[*simp*]:
$small\ \{[r, A, B]_\circ \mid r\ A\ B.\ r \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y \wedge A \in_\circ X \wedge B \in_\circ Y\}$
by (*rule down*)

ultimately have
set $\{[r,\ A,\ B]_\circ \mid r\ A\ B.\ r \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y \wedge A \in_\circ X \wedge B \in_\circ Y\} \subseteq_\circ$
$(\prod_\circ i \in_\circ$ *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* α
proof(*rule Limit-vproduct-in-VsetI*)
show *set* $\{0,\ 1_{\mathbb{N}},\ 2_{\mathbb{N}}\} \in_\circ$ *Vset* α
by (*auto simp*: *three*[*symmetric*] *intro*!: *Axiom-of-Infinity*)
from *assms*(1,2) **have** *VPow* $(\bigcup_\circ X \times_\circ \bigcup_\circ Y) \in_\circ$ *Vset* α
by (*intro Limit-VPow-in-VsetI Limit-vtimes-in-VsetI*) *auto*
then show *Q i* $\in_\circ$ *Vset* α **if** $i \in_\circ$ *set* $\{0,\ 1_{\mathbb{N}},\ 2_{\mathbb{N}}\}$ **for** *i*
using *that assms*(1,2) **unfolding** *Q-def* **by** (*auto simp*: *nat-omega-simps*)
qed *auto*
moreover have
$(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B) \subseteq_\circ$
set $\{[r,\ A,\ B]_\circ \mid r\ A\ B.\ r \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y \wedge A \in_\circ X \wedge B \in_\circ Y\}$
proof(*rule vsubsetI*)
fix *F* **assume** *prems*: $F \in_\circ (\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B)$
then obtain *A* **where** *A*: $A \in_\circ X$ **and** *F-b*: $F \in_\circ (\bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B)$
unfolding *vifunion-iff* **by** *auto*
then obtain *B* **where** *B*: $B \in_\circ Y$ **and** *F-fba*: $F \in_\circ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B$
by *fastforce*
then have $F : A \mapsto_{dg\text{-}Rel\ \alpha} B$ **by** *simp*
note $F = dg\text{-}Rel\text{-}is\text{-}arrD[OF\ this]$
interpret *F*: *arr-Rel* α *F* **rewrites** $F(\!|ArrDom|\!) = A$ **and** $F(\!|ArrCod|\!) = B$
by (*intro F*)+
show *F* $\in_\circ$ *set* $\{[r,\ A,\ B]_\circ \mid r\ A\ B.\ r \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y \wedge A \in_\circ X \wedge B \in_\circ Y\}$
proof(*intro in-set-CollectI small exI conjI*)
from *F.arr-Rel-def* **show** $F = [F(\!|ArrVal|\!),\ A,\ B]_\circ$ **unfolding** *F*(2,3) **by** *simp*
from *A B* **have** $A \times_\circ B \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y$ **by** *auto*
moreover then have $F(\!|ArrVal|\!) \subseteq_\circ A \times_\circ B$
by (*auto simp*: *F.arr-Rel-ArrVal-vsubset-ArrDom-ArrCod*)
ultimately show $F(\!|ArrVal|\!) \subseteq_\circ \bigcup_\circ X \times_\circ \bigcup_\circ Y$ **by** *auto*
qed (*intro A B*)+
qed
ultimately show $(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B) \in_\circ$ *Vset* α **by** *blast*
qed

lemma (**in** $\mathcal{Z}$) *digraph-dg-Rel*: *digraph* α (*dg-Rel* α)
proof(*intro digraphI*)
show *vfsequence* (*dg-Rel* α) **unfolding** *dg-Rel-def* **by** *clarsimp*
show *vcard* (*dg-Rel* α) = $4_{\mathbb{N}}$
unfolding *dg-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*dg-Rel* $\alpha(\!|Dom|\!)$) $\subseteq_\circ$ *dg-Rel* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-Rel-Dom-vrange*)
show $\mathcal{R}_\circ$ (*dg-Rel* $\alpha(\!|Cod|\!)$) $\subseteq_\circ$ *dg-Rel* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-Rel-Cod-vrange*)
qed (*auto simp*: *dg-Rel-components dg-Rel-Hom-vifunion-in-Vset dg-Rel-Dom-vrange*)

3.12.5 Canonical dagger for *Rel*

Dagger categories are exposed explicitly later. In the context of this section, the "dagger" is viewed merely as an explicitly defined homomorphism. A definition of a dagger functor, upon which the definition presented in this section is based, can be found in nLab [3][5]. This reference also contains the majority of the results that are presented in this subsection.

[5] https://ncatlab.org/nlab/show/Rel)

Definition and elementary properties

definition *dghm-dag-Rel* :: $V \Rightarrow V$ ($\langle\dagger_{DG.Rel}\rangle$)
  where $\dagger_{DG.Rel}\ \alpha$ =
    [
      vid-on (*dg-Rel* $\alpha(\!|Obj|\!)$),
      VLambda (*dg-Rel* $\alpha(\!|Arr|\!)$) *converse-Rel*,
      op-dg (*dg-Rel* α),
      dg-Rel α
    ]$_\circ$

Components.

lemma *dghm-dag-Rel-components*:
  shows $\dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)$ = *vid-on* (*dg-Rel* $\alpha(\!|Obj|\!)$)
    and $\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)$ = *VLambda* (*dg-Rel* $\alpha(\!|Arr|\!)$) *converse-Rel*
    and $\dagger_{DG.Rel}\ \alpha(\!|HomDom|\!)$ = *op-dg* (*dg-Rel* α)
    and $\dagger_{DG.Rel}\ \alpha(\!|HomCod|\!)$ = *dg-Rel* α
  unfolding *dghm-dag-Rel-def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object map

mk-VLambda *dghm-dag-Rel-components*(1)[*folded VLambda-vid-on*]
  |*vsv dghm-dag-Rel-ObjMap-vsv*[*dg-Rel-cs-intros*]|
  |*vdomain*
    dghm-dag-Rel-ObjMap-vdomain[*unfolded dg-Rel-components*, *dg-Rel-cs-simps*]
  |
  |*app dghm-dag-Rel-ObjMap-app*[*unfolded dg-Rel-components*, *dg-Rel-cs-simps*]|

lemma *dghm-dag-Rel-ObjMap-vrange*[*dg-cs-simps*]: $\mathcal{R}_\circ$ ($\dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)$) = *Vset* α
  unfolding *dghm-dag-Rel-components dg-Rel-components* **by** *simp*

Arrow map

mk-VLambda *dghm-dag-Rel-components*(2)
  |*vsv dghm-dag-Rel-ArrMap-vsv*[*dg-Rel-cs-intros*]|
  |*vdomain dghm-dag-Rel-ArrMap-vdomain*[*dg-Rel-cs-simps*]|
  |*app dghm-dag-Rel-ArrMap-app*[*unfolded dg-Rel-cs-simps*, *dg-Rel-cs-simps*]|

lemma *dghm-dag-Rel-ArrMap-app-vdomain*[*dg-cs-simps*]:
  assumes $T : A \mapsto_{dg\text{-}Rel\ \alpha} B$
  shows $\mathcal{D}_\circ$ ($\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)(\!|ArrVal|\!)$) = $\mathcal{R}_\circ$ ($T(\!|ArrVal|\!)$)
proof–
  from *assms* **interpret** *T*: *arr-Rel* α *T* **by** (*simp add*: *dg-Rel-is-arrD*)
  from *dg-Rel-is-arrD*(1)[*OF assms*] **show** *?thesis*
    by (*cs-concl* **cs-simp**: *dg-Rel-cs-simps V-cs-simps converse-Rel-components*(1))
qed

lemma *dghm-dag-Rel-ArrMap-app-vrange*[*dg-cs-simps*]:
  assumes $T : A \mapsto_{dg\text{-}Rel\ \alpha} B$
  shows $\mathcal{R}_\circ$ ($\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)(\!|ArrVal|\!)$) = $\mathcal{D}_\circ$ ($T(\!|ArrVal|\!)$)
proof–
  from *assms* **interpret** *T*: *arr-Rel* α *T* **by** (*simp add*: *dg-Rel-is-arrD*)
  from *dg-Rel-is-arrD*(1)[*OF assms*] **show** *?thesis*
    by (*cs-concl* **cs-simp**: *dg-Rel-cs-simps V-cs-simps converse-Rel-components*(1))
qed

lemma *dghm-dag-Rel-ArrMap-app-iff*[*dg-cs-simps*]:
  assumes $T : A \mapsto_{dg\text{-}Rel\ \alpha} B$

```
  shows ⟨a, b⟩ ∈∘ †DG.Rel α(⟦ArrMap⟧)(⟦T⟧)(⟦ArrVal⟧) ⟷ ⟨b, a⟩ ∈∘ T(⟦ArrVal⟧)
proof–
  from assms interpret T: arr-Rel α T by (simp add: dg-Rel-is-arrD)
  note T = dg-Rel-is-arrD[OF assms]
  note [dg-Rel-cs-simps] = converse-Rel-components
  show ?thesis
  proof(intro iffI)
    assume prems: ⟨a, b⟩ ∈∘ †DG.Rel α(⟦ArrMap⟧)(⟦T⟧)(⟦ArrVal⟧)
    then have a: a ∈∘ 𝒟∘ (†DG.Rel α(⟦ArrMap⟧)(⟦T⟧)(⟦ArrVal⟧))
      and b: b ∈∘ ℛ∘ (†DG.Rel α(⟦ArrMap⟧)(⟦T⟧)(⟦ArrVal⟧))
      by auto
    with assms have a: a ∈∘ ℛ∘ (T(⟦ArrVal⟧)) and b: b ∈∘ 𝒟∘ (T(⟦ArrVal⟧))
      by (simp-all add: dg-cs-simps)
    from prems T(1) have ⟨a, b⟩ ∈∘ (T(⟦ArrVal⟧))⁻¹∘
      by (cs-prems cs-shallow cs-simp: dg-Rel-cs-simps)
    then show ⟨b, a⟩ ∈∘ T(⟦ArrVal⟧) by clarsimp
  next
    assume ⟨b, a⟩ ∈∘ T(⟦ArrVal⟧)
    then have ⟨a, b⟩ ∈∘ (T(⟦ArrVal⟧))⁻¹∘ by auto
    with T(1) show ⟨a, b⟩ ∈∘ †DG.Rel α(⟦ArrMap⟧)(⟦T⟧)(⟦ArrVal⟧)
      by (cs-concl cs-shallow cs-simp: dg-Rel-cs-simps)
  qed
qed
```

Further properties

```
lemma dghm-dag-Rel-ArrMap-vrange[dg-Rel-cs-simps]:
  ℛ∘ (†DG.Rel α(⟦ArrMap⟧)) = dg-Rel α(⟦Arr⟧)
proof(intro vsubset-antisym vsubsetI)
  interpret ArrMap: vsv ‹†DG.Rel α(⟦ArrMap⟧)›
    unfolding dghm-dag-Rel-components by simp
  fix T assume T ∈∘ ℛ∘ (†DG.Rel α(⟦ArrMap⟧))
  then obtain S where T-def: T = †DG.Rel α(⟦ArrMap⟧)(⟦S⟧)
    and S: S ∈∘ 𝒟∘ (†DG.Rel α(⟦ArrMap⟧))
    by (blast dest: ArrMap.vrange-atD)
  from S show T ∈∘ dg-Rel α(⟦Arr⟧)
    by
      (
        simp add:
          T-def
          dghm-dag-Rel-components
          dg-Rel-components
          arr-Rel.arr-Rel-converse-Rel
      )
next
  interpret ArrMap: vsv ‹†DG.Rel α(⟦ArrMap⟧)›
    unfolding dghm-dag-Rel-components by simp
  fix T assume T ∈∘ dg-Rel α(⟦Arr⟧)
  then have arr-Rel α T by (simp add: dg-Rel-components)
  then have (T⁻¹Rel)⁻¹Rel = T and arr-Rel α (T⁻¹Rel)
    by
      (
        auto simp:
          arr-Rel.arr-Rel-converse-Rel-converse-Rel arr-Rel.arr-Rel-converse-Rel
      )
  then have †DG.Rel α(⟦ArrMap⟧)(⟦T⁻¹Rel⟧) = T T⁻¹Rel ∈∘ 𝒟∘ (†DG.Rel α(⟦ArrMap⟧))
    by (simp-all add: dg-Rel-components(2) dghm-dag-Rel-components(2))
  then show T ∈∘ ℛ∘ (†DG.Rel α(⟦ArrMap⟧)) by blast
```

qed

lemma *dghm-dag-Rel-ArrMap-app-is-arr*:
assumes $T : b \mapsto_{dg\text{-}Rel\ \alpha} a$
shows
$\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!) : \dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)(\!|a|\!) \mapsto_{dg\text{-}Rel\ \alpha} \dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)(\!|b|\!)$
proof(*intro is-arrI*)
from *assms* **have** *a*: $a \in_\circ Vset\ \alpha$ **and** *b*: $b \in_\circ Vset\ \alpha$
unfolding *dg-Rel-components* **by** (*fastforce simp*: *dg-Rel-components*)+
from *assms* **have** *T*: *arr-Rel* α *T* **by** (*auto simp*: *dg-Rel-is-arrD*(1))
then show *dag-T*: $\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!) \in_\circ$ *dg-Rel* $\alpha(\!|Arr|\!)$
by (*cs-concl* **cs-shallow cs-simp**: *dg-Rel-cs-simps* **cs-intro**: *dg-Rel-cs-intros*)
from *a assms T* **show** *dg-Rel* $\alpha(\!|Dom|\!)(\!|\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)|\!) = \dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)(\!|a|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *dg-cs-simps dg-Rel-cs-simps* **cs-intro**: *dg-Rel-cs-intros*
)
from *b assms T* **show** *dg-Rel* $\alpha(\!|Cod|\!)(\!|\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)|\!) = \dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)(\!|b|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *dg-cs-simps dg-Rel-cs-simps* **cs-intro**: *dg-Rel-cs-intros*
)
qed

Canonical dagger for *Rel* is a digraph isomorphism

lemma (**in** $\mathcal{Z}$) *dghm-dag-Rel-is-iso-dghm*:
$\dagger_{DG.Rel}\ \alpha : op\text{-}dg\ (dg\text{-}Rel\ \alpha) \mapsto\mapsto_{DG.iso\alpha} dg\text{-}Rel\ \alpha$
proof(*rule is-iso-dghmI*)
interpret *digraph* α ‹*dg-Rel* α› **by** (*simp add*: *digraph-dg-Rel*)
show $\dagger_{DG.Rel}\ \alpha : op\text{-}dg\ (dg\text{-}Rel\ \alpha) \mapsto\mapsto_{DG\alpha} dg\text{-}Rel\ \alpha$
proof(*rule is-dghmI*, *unfold dg-op-simps dghm-dag-Rel-components*(3,4))
show *vfsequence* $(\dagger_{DG.Rel}\ \alpha)$
unfolding *dghm-dag-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* $(\dagger_{DG.Rel}\ \alpha) = 4_{\mathbb{N}}$
unfolding *dghm-dag-Rel-def* **by** (*simp add*: *nat-omega-simps*)
fix *T a b* **assume** $T : b \mapsto_{dg\text{-}Rel\ \alpha} a$
then show
$\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!) : \dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)(\!|a|\!) \mapsto_{dg\text{-}Rel\ \alpha} \dagger_{DG.Rel}\ \alpha(\!|ObjMap|\!)(\!|b|\!)$
by (*rule dghm-dag-Rel-ArrMap-app-is-arr*)
qed (*auto simp*: *dghm-dag-Rel-components intro*: *dg-cs-intros dg-op-intros*)
show *v*11 $(\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!))$
proof
(
*intro vsv.vsv-valeq-v*11*I*,
unfold dghm-dag-Rel-ArrMap-vdomain dg-Rel-Arr-iff
)
fix *S T* **assume** *prems*:
arr-Rel α *S*
arr-Rel α *T*
$\dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|S|\!) = \dagger_{DG.Rel}\ \alpha(\!|ArrMap|\!)(\!|T|\!)$
from *prems* **show** *S* = *T*
by
(
auto simp:
dg-Rel-components

```
        dg-Rel-cs-simps
        dghm-dag-Rel-ArrMap-app[OF prems(1)]
        dghm-dag-Rel-ArrMap-app[OF prems(2)]
    )
  qed (auto intro: dg-Rel-cs-intros)
  show ℛ∘ (†DG.Rel α(‖ArrMap‖)) = dg-Rel α(‖Arr‖) by (simp add: dg-Rel-cs-simps)
qed (simp-all add: dghm-dag-Rel-components)
```

Further properties of the canonical dagger

```
lemma (in 𝒵) dghm-cn-comp-dghm-dag-Rel-dghm-dag-Rel:
  †DG.Rel α DGHM∘ †DG.Rel α = dghm-id (dg-Rel α)
proof–
  interpret digraph α ‹dg-Rel α› by (simp add: digraph-dg-Rel)
  from dghm-dag-Rel-is-iso-dghm have dag:
    †DG.Rel α : dg-Rel α DG↦↦α dg-Rel α
    by (simp add: is-iso-dghm-def)
  show ?thesis
  proof(rule dghm-eqI)
    show (†DG.Rel α DGHM∘ †DG.Rel α)(‖ArrMap‖) = dghm-id (dg-Rel α)(‖ArrMap‖)
    proof(rule vsv-eqI)
      show vsv ((†DG.Rel α DGHM∘ †DG.Rel α)(‖ArrMap‖))
        by (auto simp: dghm-cn-comp-components dghm-dag-Rel-components)
      fix a assume a ∈∘ 𝒟∘ ((†DG.Rel α DGHM∘ †DG.Rel α)(‖ArrMap‖))
      then have a: arr-Rel α a
        unfolding dg-Rel-cs-simps dghm-cn-comp-ArrMap-vdomain[OF dag dag] by simp
      from a dghm-dag-Rel-is-iso-dghm show
        (†DG.Rel α DGHM∘ †DG.Rel α)(‖ArrMap‖)(‖a‖) = dghm-id (dg-Rel α)(‖ArrMap‖)(‖a‖)
        by
          (
          cs-concl
            cs-simp: dg-Rel-cs-simps dg-cs-simps dg-cn-cs-simps
            cs-intro: dg-Rel-cs-intros dghm-cs-intros
          )
    qed (simp-all add: dghm-cn-comp-components dghm-id-components dg-Rel-cs-simps)
    show dghm-id (dg-Rel α) : dg-Rel α ↦↦DGα dg-Rel α
      by (simp-all add: digraph.dg-dghm-id-is-dghm digraph-axioms)
  qed
    (
    auto simp:
      dghm-cn-comp-is-dghm[OF digraph-axioms dag dag]
      dghm-cn-comp-components
      dghm-dag-Rel-components
      dghm-id-components
    )
qed
```

3.13 *Par* as a digraph

3.13.1 Background

Par is usually defined as a category of sets and partial functions (see nLab [3][6]). However, there is little that can prevent one from exposing *Par* as a digraph and provide additional structure gradually in subsequent installments of this work. Thus, in this section, α-*Par* is defined as a digraph of sets and partial functions in V_α

named-theorems *dg-Par-cs-simps*
named-theorems *dg-Par-cs-intros*

lemmas [*dg-Par-cs-simps*] = *dg-Rel-shared-cs-simps*
lemmas [*dg-Par-cs-intros*] = *dg-Rel-shared-cs-intros*

3.13.2 Arrow for *Par*

Definition and elementary properties

locale *arr-Par* = $\mathcal{Z}$ α + *vfsequence* T + *ArrVal*: *vsv* ‹$T(\lfloor ArrVal \rfloor)$› **for** α T +
  assumes *arr-Par-length*[*dg-Rel-shared-cs-simps*, *dg-Par-cs-simps*]:
    vcard $T = 3_{\mathbb{N}}$
    and *arr-Par-ArrVal-vdomain*: $\mathcal{D}_\circ\ (T(\lfloor ArrVal \rfloor)) \subseteq_\circ T(\lfloor ArrDom \rfloor)$
    and *arr-Par-ArrVal-vrange*: $\mathcal{R}_\circ\ (T(\lfloor ArrVal \rfloor)) \subseteq_\circ T(\lfloor ArrCod \rfloor)$
    and *arr-Par-ArrDom-in-Vset*: $T(\lfloor ArrDom \rfloor) \in_\circ Vset\ \alpha$
    and *arr-Par-ArrCod-in-Vset*: $T(\lfloor ArrCod \rfloor) \in_\circ Vset\ \alpha$

Elementary properties.

sublocale *arr-Par* ⊆ *arr-Rel*
  by *unfold-locales*
    (
      simp-all add:
        dg-Par-cs-simps
        arr-Par-ArrVal-vdomain
        arr-Par-ArrVal-vrange
        arr-Par-ArrDom-in-Vset
        arr-Par-ArrCod-in-Vset
)

lemmas (**in** *arr-Par*) [*dg-Par-cs-simps*] = *dg-Rel-shared-cs-simps*

Rules.

lemma (**in** *arr-Par*) *arr-Par-axioms'*[*dg-cs-intros*, *dg-Par-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *arr-Par* α' T
  unfolding *assms* **by** (*rule arr-Par-axioms*)

mk-ide rf *arr-Par-def*[*unfolded arr-Par-axioms-def*]
  |*intro arr-ParI*|
  |*dest arr-ParD*[*dest*]|
  |*elim arr-ParE*[*elim*!]|

lemma (**in** $\mathcal{Z}$) *arr-Par-vfsequenceI*:
  assumes *vsv* *r*
    and $\mathcal{D}_\circ\ r \subseteq_\circ a$
    and $\mathcal{R}_\circ\ r \subseteq_\circ b$
    and $a \in_\circ Vset\ \alpha$

[6] https://ncatlab.org/nlab/show/partial+function

```
    and b ∈∘ Vset α
  shows arr-Par α [r, a, b]∘
  by (intro arr-ParI)
    (insert assms, auto simp: arr-Rel-components nat-omega-simps)

lemma arr-Par-arr-RelI:
  assumes arr-Rel α T and vsv (T(|ArrVal|))
  shows arr-Par α T
proof–
  interpret arr-Rel α T by (rule assms(1))
  show ?thesis
    by (intro arr-ParI)
      (
        auto simp:
          dg-Rel-cs-simps
          assms(2)
          vfsequence-axioms
          arr-Rel-ArrVal-vdomain
          arr-Rel-ArrVal-vrange
          arr-Rel-ArrDom-in-Vset
          arr-Rel-ArrCod-in-Vset
      )
qed

lemma arr-Par-arr-RelD:
  assumes arr-Par α T
  shows arr-Rel α T and vsv (T(|ArrVal|))
proof–
  interpret arr-Par α T by (rule assms)
  show arr-Rel α T and vsv (T(|ArrVal|))
    by (rule arr-Rel-axioms) auto
qed

lemma arr-Par-arr-RelE:
  assumes arr-Par α T
  obtains arr-Rel α T and vsv (T(|ArrVal|))
  using assms by (auto simp: arr-Par-arr-RelD)
```

Further properties.

```
lemma arr-Par-eqI:
  assumes arr-Par α S
    and arr-Par α T
    and S(|ArrVal|) = T(|ArrVal|)
    and S(|ArrDom|) = T(|ArrDom|)
    and S(|ArrCod|) = T(|ArrCod|)
  shows S = T
proof(rule vsv-eqI)
  interpret S: arr-Par α S by (rule assms(1))
  interpret T: arr-Par α T by (rule assms(2))
  show vsv S by (rule S.vsv-axioms)
  show vsv T by (rule T.vsv-axioms)
  show 𝒟∘ S = 𝒟∘ T
    by (simp add: S.vfsequence-vdomain T.vfsequence-vdomain dg-Par-cs-simps)
  have dom: 𝒟∘ S = 3ℕ by (simp add: S.vfsequence-vdomain dg-Par-cs-simps)
  show a ∈∘ 𝒟∘ S ⟹ S(|a|) = T(|a|) for a
    by (unfold dom, elim-in-numeral, insert assms)
      (auto simp: arr-field-simps)
qed
```

lemma *small-arr-Par*[*simp*]: *small* {*T*. *arr-Par* α *T*}
proof(*rule smaller-than-small*)
  show {*T*. *arr-Par* α *T*} $\subseteq$ {*T*. *arr-Rel* α *T*}
    by (*simp add*: *Collect-mono arr-Par-arr-RelD*(1))
qed *simp*

lemma *set-Collect-arr-Par*[*simp*]:
  $T \in_\circ$ *set* (*Collect* (*arr-Par* α)) $\longleftrightarrow$ *arr-Par* α *T*
  by *auto*

Composition

abbreviation (*input*) *comp-Par* :: *V* $\Rightarrow$ *V* $\Rightarrow$ *V* (**infixl** $\langle\circ_{Par}\rangle$ 55)
  where *comp-Par* $\equiv$ *comp-Rel*

lemma *arr-Par-comp-Par*[*dg-Par-cs-intros*]:
  assumes *arr-Par* α *S* **and** *arr-Par* α *T*
  shows *arr-Par* α (*S* $\circ_{Par}$ *T*)
proof(*intro arr-Par-arr-RelI*)
  interpret *S*: *arr-Par* α *S* **by** (*rule assms*(1))
  interpret *T*: *arr-Par* α *T* **by** (*rule assms*(2))
  show *arr-Rel* α (*S* $\circ_{Par}$ *T*)
    by (*auto simp*: *S.arr-Rel-axioms T.arr-Rel-axioms arr-Rel-comp-Rel*)
  show *vsv* ((*S* $\circ_{Par}$ *T*)(|*ArrVal*|))
    unfolding *comp-Rel-components*
    by (*simp add*: *S.ArrVal.vsv-axioms T.ArrVal.vsv-axioms vsv-vcomp*)
qed

lemma *arr-Par-comp-Par-ArrVal-app*:
  assumes *arr-Par* α *S*
    and *arr-Par* α *T*
    and *x* $\in_\circ$ $\mathcal{D}_\circ$ (*T*(|*ArrVal*|))
    and *T*(|*ArrVal*|)(|*x*|) $\in_\circ$ $\mathcal{D}_\circ$ (*S*(|*ArrVal*|))
  shows (*S* $\circ_{Par}$ *T*)(|*ArrVal*|)(|*x*|) = *S*(|*ArrVal*|)(|*T*(|*ArrVal*|)(|*x*|)|)
  using *assms* **unfolding** *comp-Rel-components* **by** (*intro vcomp-atI*) *auto*

Inclusion

abbreviation (*input*) *incl-Par* :: *V* $\Rightarrow$ *V* $\Rightarrow$ *V*
  where *incl-Par* $\equiv$ *incl-Rel*

lemma (**in** $\mathcal{Z}$) *arr-Par-incl-ParI*:
  assumes *A* $\in_\circ$ *Vset* α **and** *B* $\in_\circ$ *Vset* α **and** *A* $\subseteq_\circ$ *B*
  shows *arr-Par* α (*incl-Par A B*)
proof(*intro arr-Par-arr-RelI*)
  from *assms* **show** *arr-Rel* α (*incl-Par A B*)
    by (*force intro*: *arr-Rel-incl-RelI*)
qed (*simp add*: *incl-Rel-components*)

Identity

abbreviation (*input*) *id-Par* :: *V* $\Rightarrow$ *V*
  where *id-Par* $\equiv$ *id-Rel*

lemma (**in** $\mathcal{Z}$) *arr-Par-id-ParI*:
  assumes *A* $\in_\circ$ *Vset* α
  shows *arr-Par* α (*id-Par A*)
  using *assms*

by (*intro arr-Par-arr-RelI*)
(*auto intro*: *arr-Rel-id-RelI simp*: *id-Rel-components*)

lemma *arr-Par-comp-Par-id-Par-left*[*dg-Par-cs-simps*]:
assumes *arr-Par* α *f* **and** $f(\!|ArrCod|\!) = A$
shows *id-Par A* $\circ_{Par}$ *f* = *f*
proof–
interpret *f*: *arr-Par* α *f* **by** (*rule assms*(1))
have *arr-Rel* α *f* **by** (*simp add*: *f.arr-Rel-axioms*)
from *arr-Rel-comp-Rel-id-Rel-left*[*OF this assms*(2)] **show** *?thesis* **.**
qed

lemma *arr-Par-comp-Par-id-Par-right*[*dg-Par-cs-simps*]:
assumes *arr-Par* α *f* **and** $f(\!|ArrDom|\!) = A$
shows *f* $\circ_{Par}$ *id-Par A* = *f*
proof–
interpret *f*: *arr-Par* α *f* **by** (*rule assms*(1))
have *arr-Rel* α *f* **by** (*simp add*: *f.arr-Rel-axioms*)
from *arr-Rel-comp-Rel-id-Rel-right*[*OF this assms*(2)] **show** *?thesis***.**
qed

3.13.3 *Par* as a digraph

Definition and elementary properties

definition *dg-Par* :: $V \Rightarrow V$
where *dg-Par* α =
[
Vset α,
set $\{T.$ *arr-Par* α *T*$\}$,
$(\lambda T \in_\circ set\ \{T.$ *arr-Par* α *T*$\}.$ $T(\!|ArrDom|\!))$,
$(\lambda T \in_\circ set\ \{T.$ *arr-Par* α *T*$\}.$ $T(\!|ArrCod|\!))$
]$_\circ$

Components.

lemma *dg-Par-components*:
shows *dg-Par* $\alpha(\!|Obj|\!)$ = *Vset* α
and *dg-Par* $\alpha(\!|Arr|\!) = set\ \{T.$ *arr-Par* α *T*$\}$
and *dg-Par* $\alpha(\!|Dom|\!) = (\lambda T \in_\circ set\ \{T.$ *arr-Par* α *T*$\}.$ $T(\!|ArrDom|\!))$
and *dg-Par* $\alpha(\!|Cod|\!) = (\lambda T \in_\circ set\ \{T.$ *arr-Par* α *T*$\}.$ $T(\!|ArrCod|\!))$
unfolding *dg-Par-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object

lemma *dg-Par-Obj-iff*: $x \in_\circ$ *dg-Par* $\alpha(\!|Obj|\!) \longleftrightarrow x \in_\circ$ *Vset* α
unfolding *dg-Par-components* **by** *auto*

Arrow

lemma *dg-Par-Arr-iff*[*dg-Par-cs-simps*]: $x \in_\circ$ *dg-Par* $\alpha(\!|Arr|\!) \longleftrightarrow$ *arr-Par* α *x*
unfolding *dg-Par-components* **by** *auto*

Domain

mk-VLambda *dg-Par-components*(3)
|*vsv dg-Par-Dom-vsv*[*dg-Par-cs-intros*]|
|*vdomain dg-Par-Dom-vdomain*[*dg-Par-cs-simps*]|
|*app dg-Par-Dom-app*[*unfolded set-Collect-arr-Par*, *dg-Par-cs-simps*]|

lemma *dg-Par-Dom-vrange*: $\mathcal{R}_\circ\ (dg\text{-}Par\ \alpha(\!|Dom|\!))\subseteq_\circ dg\text{-}Par\ \alpha(\!|Obj|\!)$
unfolding *dg-Par-components*
by (*rule vrange-VLambda-vsubset*, *unfold set-Collect-arr-Par*) *auto*

Codomain

mk-VLambda *dg-Par-components*(4)
|*vsv dg-Par-Cod-vsv*[*dg-Par-cs-intros*]|
|*vdomain dg-Par-Cod-vdomain*[*dg-Par-cs-simps*]|
|*app dg-Par-Cod-app*[*unfolded set-Collect-arr-Par*, *dg-Par-cs-simps*]|

lemma *dg-Par-Cod-vrange*: $\mathcal{R}_\circ\ (dg\text{-}Par\ \alpha(\!|Cod|\!))\subseteq_\circ dg\text{-}Par\ \alpha(\!|Obj|\!)$
unfolding *dg-Par-components*
by (*rule vrange-VLambda-vsubset*, *unfold set-Collect-arr-Par*) *auto*

Arrow with a domain and a codomain

Rules.

lemma *dg-Par-is-arrI*:
assumes *arr-Par* α S **and** $S(\!|ArrDom|\!)=A$ **and** $S(\!|ArrCod|\!)=B$
shows $S:A\mapsto_{dg\text{-}Par\ \alpha}B$
using *assms* **by** (*intro is-arrI*, *unfold dg-Par-components*) *simp-all*

lemmas [*dg-Par-cs-intros*] = *dg-Par-is-arrI*

lemma *dg-Par-is-arrD*:
assumes $S:A\mapsto_{dg\text{-}Par\ \alpha}B$
shows *arr-Par* α S
and [*dg-cs-simps*]: $S(\!|ArrDom|\!)=A$
and [*dg-cs-simps*]: $S(\!|ArrCod|\!)=B$
using *is-arrD*[*OF assms*] **unfolding** *dg-Par-components* **by** *simp-all*

lemma *dg-Par-is-arrE*:
assumes $S:A\mapsto_{dg\text{-}Par\ \alpha}B$
obtains *arr-Par* α S **and** $S(\!|ArrDom|\!)=A$ **and** $S(\!|ArrCod|\!)=B$
using *is-arrD*[*OF assms*] **unfolding** *dg-Par-components* **by** *simp-all*

Elementary properties.

lemma *dg-Par-is-arr-dg-Rel-is-arr*:
assumes $r:a\mapsto_{dg\text{-}Par\ \alpha}b$
shows $r:a\mapsto_{dg\text{-}Rel\ \alpha}b$
using *assms arr-Par-arr-RelD*(1)
by (*intro dg-Rel-is-arrI*; *elim dg-Par-is-arrE*) *auto*

lemma *dg-Par-Hom-vsubset-dg-Rel-Hom*:
assumes $a\in_\circ dg\text{-}Par\ \alpha(\!|Obj|\!)$ $b\in_\circ dg\text{-}Par\ \alpha(\!|Obj|\!)$
shows *Hom* (*dg-Par* α) a $b\subseteq_\circ$ *Hom* (*dg-Rel* α) a b
by (*rule vsubsetI*) (*simp add*: *dg-Par-is-arr-dg-Rel-is-arr*)

lemma (**in** $\mathcal{Z}$) *dg-Par-incl-Par-is-arr*:
assumes $A\in_\circ dg\text{-}Par\ \alpha(\!|Obj|\!)$ **and** $B\in_\circ dg\text{-}Par\ \alpha(\!|Obj|\!)$ **and** $A\subseteq_\circ B$
shows *incl-Par* A $B:A\mapsto_{dg\text{-}Par\ \alpha}B$
by (*rule dg-Par-is-arrI*)
(
auto
simp: *incl-Rel-components*
intro!: *arr-Par-incl-ParI assms*[*unfolded dg-Par-components*(1)]

)

lemma (in $\mathcal{Z}$**)** *dg-Par-incl-Par-is-arr'*[*dg-Par-cs-intros*]:
assumes $A \in_\circ$ *dg-Par* $\alpha(\!|Obj|\!)$
and $B \in_\circ$ *dg-Par* $\alpha(\!|Obj|\!)$
and $A \subseteq_\circ B$
and $A' = A$
and $B' = B$
shows *incl-Par* A $B : A' \mapsto_{dg\text{-}Par\ \alpha} B'$
using *assms*(1–3) **unfolding** *assms*(4,5) **by** (*rule dg-Par-incl-Par-is-arr*)

lemmas [*dg-Par-cs-intros*] = $\mathcal{Z}$.*dg-Par-incl-Par-is-arr'*

Par is a digraph

lemma (in $\mathcal{Z}$**)** *dg-Par-Hom-vifunion-in-Vset*:
assumes $X \in_\circ$ *Vset* α **and** $Y \in_\circ$ *Vset* α
shows $(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Par\ \alpha)\ A\ B) \in_\circ Vset\ \alpha$
proof–
have
$(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Par\ \alpha)\ A\ B) \subseteq_\circ$
$(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B)$
proof(*intro vsubsetI*)
fix F **assume** $F \in_\circ (\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Par\ \alpha)\ A\ B)$
then obtain B **where** B: $B \in_\circ Y$ **and** $F \in_\circ (\bigcup_\circ A \in_\circ X.\ Hom\ (dg\text{-}Par\ \alpha)\ A\ B)$
by *fast*
then obtain A **where** A: $A \in_\circ X$ **and** *F-AB*: $F \in_\circ Hom\ (dg\text{-}Par\ \alpha)\ A\ B$ **by** *fast*
from A B *assms* **have** $A \in_\circ$ *dg-Par* $\alpha(\!|Obj|\!)$ $B \in_\circ$ *dg-Par* $\alpha(\!|Obj|\!)$
unfolding *dg-Par-components* **by** *auto*
from *F-AB* A B *dg-Par-Hom-vsubset-dg-Rel-Hom*[*OF this*] **show**
$F \in_\circ (\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Rel\ \alpha)\ A\ B)$
by (*intro vifunionI*) (*auto elim*!: *vsubsetE simp*: *in-Hom-iff*)
qed
with *dg-Rel-Hom-vifunion-in-Vset*[*OF assms*] **show** *?thesis* **by** *blast*
qed

lemma (in $\mathcal{Z}$**)** *digraph-dg-Par*: *digraph* α (*dg-Par* α)
proof(*intro digraphI*)
show *vfsequence* (*dg-Par* α) **unfolding** *dg-Par-def* **by** *simp*
show *vcard* (*dg-Par* α) = $4_{\mathbb{N}}$
unfolding *dg-Par-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*dg-Par* $\alpha(\!|Dom|\!)$) $\subseteq_\circ$ *dg-Par* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-Par-Dom-vrange*)
show $\mathcal{R}_\circ$ (*dg-Par* $\alpha(\!|Cod|\!)$) $\subseteq_\circ$ *dg-Par* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-Par-Cod-vrange*)
qed (*auto simp*: *dg-Par-components dg-Par-Hom-vifunion-in-Vset*)

Par is a wide subdigraph of *Rel*

lemma (in $\mathcal{Z}$**)** *wide-subdigraph-dg-Par-dg-Rel*: *dg-Par* $\alpha \subseteq_{DG.wide\alpha}$ *dg-Rel* α
proof(*intro wide-subdigraphI*)
show *dg-Par* $\alpha \subseteq_{DG\alpha}$ *dg-Rel* α
by (*intro subdigraphI*, *unfold dg-Par-components*)
(
auto simp:
dg-Rel-components
digraph-dg-Par
digraph-dg-Rel
dg-Par-is-arr-dg-Rel-is-arr
)

qed (*simp-all add*: *dg-Rel-components dg-Par-components*)

3.14 *Set* as a digraph

3.14.1 Background

Set is usually defined as a category of sets and total functions (see Chapter I-2 in [39]). However, there is little that can prevent one from exposing *Set* as a digraph and provide additional structure gradually later. Thus, in this section, α-*Set* is defined as a digraph of sets and binary relations in the set V_α.

named-theorems *dg-Set-cs-simps*
named-theorems *dg-Set-cs-intros*

lemmas [*dg-Set-cs-simps*] = *dg-Rel-shared-cs-simps*
lemmas [*dg-Set-cs-intros*] = *dg-Rel-shared-cs-intros*

3.14.2 Arrow for *Set*

Definition and elementary properties

locale *arr-Set* = $\mathcal{Z}$ α + *vfsequence* T + *ArrVal*: *vsv* ‹$T(\!|ArrVal|\!)$› **for** α T +
  assumes *arr-Set-length*[*dg-Rel-shared-cs-simps*, *dg-Set-cs-simps*]:
    vcard $T = 3_{\mathbb{N}}$
   and *arr-Set-ArrVal-vdomain*[*dg-Rel-shared-cs-simps*, *dg-Set-cs-simps*]:
    $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = T(\!|ArrDom|\!)$
   and *arr-Set-ArrVal-vrange*: $\mathcal{R}_\circ\ (T(\!|ArrVal|\!)) \subseteq_\circ T(\!|ArrCod|\!)$
   and *arr-Set-ArrDom-in-Vset*: $T(\!|ArrDom|\!) \in_\circ Vset\ \alpha$
   and *arr-Set-ArrCod-in-Vset*: $T(\!|ArrCod|\!) \in_\circ Vset\ \alpha$

lemmas [*dg-Set-cs-simps*] = *arr-Set.arr-Set-ArrVal-vdomain*

Elementary properties.

sublocale *arr-Set* ⊆ *arr-Par*
  by *unfold-locales*
   (
    simp-all add:
      dg-Set-cs-simps
      arr-Set-ArrVal-vrange arr-Set-ArrDom-in-Vset arr-Set-ArrCod-in-Vset
)

Rules.

lemma (**in** *arr-Set*) *arr-Set-axioms'*[*dg-cs-intros*, *dg-Set-cs-intros*]:
  assumes $\alpha' = \alpha$
  shows *arr-Set* α' T
  unfolding *assms* **by** (*rule arr-Set-axioms*)

mk-ide rf *arr-Set-def*[*unfolded arr-Set-axioms-def*]
  |*intro arr-SetI*|
  |*dest arr-SetD*[*dest*]|
  |*elim arr-SetE*[*elim*!]|

lemma (**in** $\mathcal{Z}$) *arr-Set-vfsequenceI*:
  assumes *vsv* r
    and $\mathcal{D}_\circ\ r = a$
    and $\mathcal{R}_\circ\ r \subseteq_\circ b$
    and $a \in_\circ Vset\ \alpha$
    and $b \in_\circ Vset\ \alpha$
  shows *arr-Set* α $[r, a, b]_\circ$
  by (*intro arr-SetI*)

(*insert assms*, *auto simp*: *arr-Rel-components nat-omega-simps*)

lemma *arr-Set-arr-ParI*:
assumes *arr-Par* α T **and** $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = T(\!|ArrDom|\!)$
shows *arr-Set* α T
proof–
interpret *arr-Par* α T **by** (*rule assms*(1))
show *?thesis*
by (*intro arr-SetI*)
(
auto simp:
dg-Par-cs-simps
assms(2)
vfsequence-axioms
arr-Rel-ArrVal-vrange
arr-Rel-ArrDom-in-Vset
arr-Rel-ArrCod-in-Vset
)
qed

lemma *arr-Set-arr-ParD*:
assumes *arr-Set* α T
shows *arr-Par* α T **and** $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = T(\!|ArrDom|\!)$
proof–
interpret *arr-Set* α T **by** (*rule assms*)
show *arr-Par* α T **and** $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = T(\!|ArrDom|\!)$
by (*rule arr-Par-axioms*) (*auto simp*: *dg-Set-cs-simps*)
qed

lemma *arr-Set-arr-ParE*:
assumes *arr-Set* α T
obtains *arr-Par* α T **and** $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = T(\!|ArrDom|\!)$
using *assms* **by** (*auto simp*: *arr-Set-arr-ParD*)

Further properties.

lemma *arr-Set-eqI*:
assumes *arr-Set* α S
and *arr-Set* α T
and $S(\!|ArrVal|\!) = T(\!|ArrVal|\!)$
and $S(\!|ArrDom|\!) = T(\!|ArrDom|\!)$
and $S(\!|ArrCod|\!) = T(\!|ArrCod|\!)$
shows $S = T$
proof–
interpret S: *arr-Set* α S **by** (*rule assms*(1))
interpret T: *arr-Set* α T **by** (*rule assms*(2))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom*: $\mathcal{D}_\circ\ S = 3_\mathbb{N}$ **by** (*simp add*: *S.vfsequence-vdomain dg-Set-cs-simps*)
show $a \in_\circ \mathcal{D}_\circ\ S \Longrightarrow S(\!|a|\!) = T(\!|a|\!)$ **for** a
by (*unfold dom*, *elim-in-numeral*, *insert assms*)
(*auto simp*: *arr-field-simps*)
qed (*auto simp*: *S.vfsequence-vdomain T.vfsequence-vdomain dg-Set-cs-simps*)
qed

lemma *small-arr-Set*[*simp*]: *small* $\{T.\ arr\text{-}Set\ \alpha\ T\}$
proof(*rule smaller-than-small*)
show $\{T.\ arr\text{-}Set\ \alpha\ T\} \subseteq \{T.\ arr\text{-}Par\ \alpha\ T\}$
by (*simp add*: *Collect-mono arr-Set-arr-ParD*(1))

qed *simp*

lemma *set-Collect-arr-Set*[*simp*]:
  $T \in_\circ set\ (Collect\ (arr\text{-}Set\ \alpha)) \longleftrightarrow arr\text{-}Set\ \alpha\ T$
  by *auto*

Composition

See [39]).

abbreviation (*input*) *comp-Set* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$\circ_{Set}$› 55)
  where *comp-Set* ≡ *comp-Rel*

lemma *arr-Set-comp-Set*[*dg-Set-cs-intros*]:
  assumes *arr-Set* α *S* **and** *arr-Set* α *T* **and** $\mathcal{R}_\circ\ (T(\!|ArrVal|\!)) \subseteq_\circ \mathcal{D}_\circ\ (S(\!|ArrVal|\!))$
  shows *arr-Set* α $(S \circ_{Set} T)$
proof(*intro arr-Set-arr-ParI*)
  interpret *S*: *arr-Set* α *S* **by** (*rule assms*(1))
  interpret *T*: *arr-Set* α *T* **by** (*rule assms*(2))
  show *arr-Par* α $(S \circ_{Set} T)$
    by (*auto simp*: *S.arr-Par-axioms T.arr-Par-axioms arr-Par-comp-Par*)
  show $\mathcal{D}_\circ\ ((S \circ_{Rel} T)(\!|ArrVal|\!)) = (S \circ_{Rel} T)(\!|ArrDom|\!)$
    unfolding *comp-Rel-components vdomain-vcomp-vsubset*[*OF assms*(3)]
    by (*simp add*: *dg-Set-cs-simps*)
qed

lemma *arr-Set-comp-Set-ArrVal-app*:
  assumes *arr-Set* α *S*
    and *arr-Set* α *T*
    and $x \in_\circ \mathcal{D}_\circ\ (T(\!|ArrVal|\!))$
    and $T(\!|ArrVal|\!)(\!|x|\!) \in_\circ \mathcal{D}_\circ\ (S(\!|ArrVal|\!))$
  shows $(S \circ_{Set} T)(\!|ArrVal|\!)(\!|x|\!) = S(\!|ArrVal|\!)(\!|T(\!|ArrVal|\!)(\!|x|\!)|\!)$
proof–
  interpret *S*: *arr-Set* α *S* + *T*: *arr-Set* α *T* **by** (*simp-all add*: *assms*(1,2))
  from *assms* **show** *?thesis*
    unfolding *comp-Rel-components* **by** (*intro vcomp-atI*) *auto*
qed

Inclusion

abbreviation (*input*) *incl-Set* :: $V \Rightarrow V \Rightarrow V$
  where *incl-Set* ≡ *incl-Rel*

lemma (**in** $\mathcal{Z}$) *arr-Set-incl-SetI*:
  assumes $A \in_\circ Vset\ \alpha$ **and** $B \in_\circ Vset\ \alpha$ **and** $A \subseteq_\circ B$
  shows *arr-Set* α (*incl-Set A B*)
proof(*intro arr-Set-arr-ParI*)
  from *assms* **show** *arr-Par* α (*incl-Set A B*)
    by (*force intro*: *arr-Par-incl-ParI*)
qed (*simp add*: *incl-Rel-components*)

Identity

abbreviation (*input*) *id-Set* :: $V \Rightarrow V$
  where *id-Set* ≡ *id-Rel*

lemma (**in** $\mathcal{Z}$) *arr-Set-id-SetI*:
  assumes $A \in_\circ Vset\ \alpha$
  shows *arr-Set* α (*id-Set A*)

proof(*intro arr-Set-arr-ParI*)
from *assms* **show** *arr-Par* α (*id-Par A*)
by (*force intro*: *arr-Par-id-ParI*)
qed (*simp add*: *id-Rel-components*)

lemma *arr-Set-comp-Set-id-Set-left*[*dg-Set-cs-simps*]:
assumes *arr-Set* α *F* **and** $F(\!|ArrCod|\!) = A$
shows *id-Set A* $\circ_{Set}$ *F* = *F*
proof–
interpret *F*: *arr-Set* α *F* **by** (*rule assms*(1))
have *arr-Rel* α *F* **by** (*simp add*: *F.arr-Rel-axioms*)
from *arr-Rel-comp-Rel-id-Rel-left*[*OF this assms*(2)] **show** *?thesis*.
qed

lemma *arr-Set-comp-Set-id-Set-right*[*dg-Set-cs-simps*]:
assumes *arr-Set* α *F* **and** $F(\!|ArrDom|\!) = A$
shows *F* $\circ_{Set}$ *id-Set A* = *F*
proof–
interpret *F*: *arr-Set* α *F* **by** (*rule assms*(1))
have *arr-Rel* α *F* **by** (*simp add*: *F.arr-Rel-axioms*)
from *arr-Rel-comp-Rel-id-Rel-right*[*OF this assms*(2)] **show** *?thesis*.
qed

3.14.3 *Set* as a digraph

Definition and elementary properties

definition *dg-Set* :: $V \Rightarrow V$
where *dg-Set* α =
[
Vset α,
set $\{T.$ *arr-Set* α $T\}$,
$(\lambda T \in_{\circ} set\ \{T.$ *arr-Set* α $T\}.\ T(\!|ArrDom|\!))$,
$(\lambda T \in_{\circ} set\ \{T.$ *arr-Set* α $T\}.\ T(\!|ArrCod|\!))$
]$_{\circ}$

Components.

lemma *dg-Set-components*:
shows *dg-Set* $\alpha(\!|Obj|\!)$ = *Vset* α
and *dg-Set* $\alpha(\!|Arr|\!)$ = *set* $\{T.$ *arr-Set* α $T\}$
and *dg-Set* $\alpha(\!|Dom|\!) = (\lambda T \in_{\circ} set\ \{T.$ *arr-Set* α $T\}.\ T(\!|ArrDom|\!))$
and *dg-Set* $\alpha(\!|Cod|\!) = (\lambda T \in_{\circ} set\ \{T.$ *arr-Set* α $T\}.\ T(\!|ArrCod|\!))$
unfolding *dg-Set-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Object

lemma *dg-Set-Obj-iff*: $x \in_{\circ}$ *dg-Set* $\alpha(\!|Obj|\!) \longleftrightarrow x \in_{\circ}$ *Vset* α
unfolding *dg-Set-components* **by** *auto*

Arrow

lemma *dg-Set-Arr-iff*[*dg-Set-cs-simps*]: $x \in_{\circ}$ *dg-Set* $\alpha(\!|Arr|\!) \longleftrightarrow$ *arr-Set* α *x*
unfolding *dg-Set-components* **by** *auto*

Domain

mk-VLambda *dg-Set-components*(3)
|*vsv dg-Set-Dom-vsv*[*dg-Set-cs-intros*]|
|*vdomain dg-Set-Dom-vdomain*[*dg-Set-cs-simps*]|

|*app dg-Set-Dom-app*[*unfolded set-Collect-arr-Set*, *dg-Set-cs-simps*]|

lemma *dg-Set-Dom-vrange*: $\mathcal{R}_\circ\ (dg\text{-}Set\ \alpha(\!|Dom|\!))\subseteq_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$
unfolding *dg-Set-components*
by (*rule vrange-VLambda-vsubset*, *unfold set-Collect-arr-Set*) *auto*

Codomain

mk-VLambda *dg-Set-components*(4)
|*vsv dg-Set-Cod-vsv*[*dg-Set-cs-intros*]|
|*vdomain dg-Set-Cod-vdomain*[*dg-Set-cs-simps*]|
|*app dg-Set-Cod-app*[*unfolded set-Collect-arr-Set*, *dg-Set-cs-simps*]|

lemma *dg-Set-Cod-vrange*: $\mathcal{R}_\circ\ (dg\text{-}Set\ \alpha(\!|Cod|\!))\subseteq_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$
unfolding *dg-Set-components*
by (*rule vrange-VLambda-vsubset*, *unfold set-Collect-arr-Set*) *auto*

Arrow with a domain and a codomain

Rules.

lemma *dg-Set-is-arrI*[*dg-Set-cs-intros*]:
assumes *arr-Set* $\alpha\ S$ **and** $S(\!|ArrDom|\!) = A$ **and** $S(\!|ArrCod|\!) = B$
shows $S : A \mapsto_{dg\text{-}Set\ \alpha} B$
using *assms* **by** (*intro is-arrI*, *unfold dg-Set-components*) *simp-all*

lemma *dg-Set-is-arrD*:
assumes $S : A \mapsto_{dg\text{-}Set\ \alpha} B$
shows *arr-Set* $\alpha\ S$
and [*dg-cs-simps*]: $S(\!|ArrDom|\!) = A$
and [*dg-cs-simps*]: $S(\!|ArrCod|\!) = B$
using *is-arrD*[*OF assms*] **unfolding** *dg-Set-components* **by** *simp-all*

lemma *dg-Set-is-arrE*:
assumes $S : A \mapsto_{dg\text{-}Set\ \alpha} B$
obtains *arr-Set* $\alpha\ S$ **and** $S(\!|ArrDom|\!) = A$ **and** $S(\!|ArrCod|\!) = B$
using *is-arrD*[*OF assms*] **unfolding** *dg-Set-components* **by** *simp-all*

lemma *dg-Set-ArrVal-vdomain*[*dg-Set-cs-simps*, *dg-cs-simps*]:
assumes $T : A \mapsto_{dg\text{-}Set\ \alpha} B$
shows $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = A$
proof–
interpret *T*: *arr-Set* $\alpha\ T$ **using** *assms* **by** (*auto simp*: *dg-Set-is-arrD*)
from *assms* **show** *?thesis* **by** (*auto simp*: *dg-Set-is-arrD dg-Set-cs-simps*)
qed

Elementary properties.

lemma *dg-Set-ArrVal-app-vrange*[*dg-Set-cs-intros*]:
assumes $F : A \mapsto_{dg\text{-}Set\ \alpha} B$ **and** $a \in_\circ A$
shows $F(\!|ArrVal|\!)(\!|a|\!) \in_\circ B$
proof–
interpret *F*: *arr-Set* $\alpha\ F$
rewrites $F(\!|ArrDom|\!) = A$ **and** $F(\!|ArrCod|\!) = B$
by (*intro dg-Set-is-arrD*[*OF assms*(1)])+
from *assms F.arr-Par-ArrVal-vrange* **show** *?thesis*
by (*auto simp*: *F.ArrVal.vsv-vimageI2 vsubset-iff dg-Set-cs-simps*)
qed

lemma *dg-Set-is-arr-dg-Par-is-arr*:
assumes $T : A \mapsto_{dg\text{-}Set\ \alpha} B$
shows $T : A \mapsto_{dg\text{-}Par\ \alpha} B$
using *assms arr-Set-arr-ParD*(1)
by (*intro dg-Par-is-arrI*; *elim dg-Set-is-arrE*) *auto*

lemma *dg-Set-Hom-vsubset-dg-Par-Hom*:
assumes $a \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$ $b \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$
shows $Hom\ (dg\text{-}Set\ \alpha)\ a\ b \subseteq_\circ Hom\ (dg\text{-}Par\ \alpha)\ a\ b$
by (*rule vsubsetI*) (*simp add*: *dg-Set-is-arr-dg-Par-is-arr*)

lemma (**in** $\mathcal{Z}$) *dg-Set-incl-Set-is-arr*:
assumes $A \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$ **and** $B \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$ **and** $A \subseteq_\circ B$
shows $incl\text{-}Set\ A\ B : A \mapsto_{dg\text{-}Set\ \alpha} B$
proof(*rule dg-Set-is-arrI*)
from *assms*(1,2) **show** $arr\text{-}Set\ \alpha\ (incl\text{-}Set\ A\ B)$
unfolding *dg-Set-components*(1) **by** (*intro arr-Set-incl-SetI assms*)
qed (*simp-all add*: *dg-Set-components incl-Rel-components*)

lemma (**in** $\mathcal{Z}$) *dg-Set-incl-Set-is-arr'*[*dg-cs-intros*, *dg-Set-cs-intros*]:
assumes $A \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$
and $B \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$
and $A \subseteq_\circ B$
and $A' = A$
and $B' = B$
and $\mathfrak{C}' = dg\text{-}Set\ \alpha$
shows $incl\text{-}Set\ A\ B : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(1–3) **unfolding** *assms*(4–6) **by** (*rule dg-Set-incl-Set-is-arr*)

lemmas [*dg-Set-cs-intros*] = $\mathcal{Z}$.*dg-Set-incl-Set-is-arr'*

Set is a digraph

lemma (**in** $\mathcal{Z}$) *dg-Set-Hom-vifunion-in-Vset*:
assumes $X \in_\circ Vset\ \alpha$ **and** $Y \in_\circ Vset\ \alpha$
shows $(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Set\ \alpha)\ A\ B) \in_\circ Vset\ \alpha$
proof–
have
$(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Set\ \alpha)\ A\ B) \subseteq_\circ$
$(\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Par\ \alpha)\ A\ B)$
proof
fix F **assume** $F \in_\circ (\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Set\ \alpha)\ A\ B)$
then obtain B **where** B: $B \in_\circ Y$ **and** *F-b*:
$F \in_\circ (\bigcup_\circ A \in_\circ X.\ Hom\ (dg\text{-}Set\ \alpha)\ A\ B)$
by *fast*
then obtain A **where** A: $A \in_\circ X$ **and** *F-AB*: $F \in_\circ Hom\ (dg\text{-}Set\ \alpha)\ A\ B$
by *fast*
from A B *assms* **have** $A \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$ $B \in_\circ dg\text{-}Set\ \alpha(\!|Obj|\!)$
unfolding *dg-Set-components* **by** *auto*
from *F-AB A B dg-Set-Hom-vsubset-dg-Par-Hom*[*OF this*] **show**
$F \in_\circ (\bigcup_\circ A \in_\circ X.\ \bigcup_\circ B \in_\circ Y.\ Hom\ (dg\text{-}Par\ \alpha)\ A\ B)$
by (*intro vifunionI*) (*auto elim*!: *vsubsetE simp*: *in-Hom-iff*)
qed
with *dg-Par-Hom-vifunion-in-Vset*[*OF assms*] **show** *?thesis* **by** *blast*
qed

lemma (**in** $\mathcal{Z}$) *digraph-dg-Set*: $digraph\ \alpha\ (dg\text{-}Set\ \alpha)$
proof(*intro digraphI*)

show *vfsequence* (*dg-Set* α) **unfolding** *dg-Set-def* **by** *simp*
show *vcard* (*dg-Set* α) = $4_{\mathbb{N}}$
unfolding *dg-Set-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ$ (*dg-Set* $\alpha(\!|Dom|\!)$) $\subseteq_\circ$ *dg-Set* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-Set-Dom-vrange*)
show $\mathcal{R}_\circ$ (*dg-Set* $\alpha(\!|Cod|\!)$) $\subseteq_\circ$ *dg-Set* $\alpha(\!|Obj|\!)$ **by** (*simp add*: *dg-Set-Cod-vrange*)
qed (*auto simp*: *dg-Set-components dg-Set-Hom-vifunion-in-Vset*)

Set is a wide subdigraph of *Par*

lemma (**in** $\mathcal{Z}$) *wide-subdigraph-dg-Set-dg-Par*: *dg-Set* $\alpha \subseteq_{DG.wide\alpha}$ *dg-Par* α
proof(*intro wide-subdigraphI*)
interpret *Set*: *digraph* α ‹*dg-Set* α› **by** (*rule digraph-dg-Set*)
interpret *Par*: *digraph* α ‹*dg-Par* α› **by** (*rule digraph-dg-Par*)
show *dg-Set* $\alpha \subseteq_{DG\alpha}$ *dg-Par* α
proof(*intro subdigraphI*, *unfold dg-Set-components*)
show $F : A \mapsto_{dg\text{-}Par\ \alpha} B$ **if** $F : A \mapsto_{dg\text{-}Set\ \alpha} B$ **for** $F\ A\ B$
using *that* **by** (*rule dg-Set-is-arr-dg-Par-is-arr*)
qed (*auto simp*: *dg-Par-components digraph-dg-Set digraph-dg-Par*)
qed (*simp-all add*: *dg-Par-components dg-Set-components*)

Chapter 4

Semicategories

4.1 Introduction

4.1.1 Background

Many concepts that are normally associated with category theory can be generalized to semicategories. It is the goal of this chapter to expose these generalized concepts and provide a foundation for the development of the notion of a category in [41].

4.1.2 Preliminaries

named-theorems *smc-op-simps*
named-theorems *smc-op-intros*

named-theorems *smc-cs-simps*
named-theorems *smc-cs-intros*

named-theorems *smc-arrow-cs-intros*

4.1.3 CS setup for foundations

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

4.2 Semicategory

4.2.1 Background

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

Slicing

Slicing is a term that is introduced in this work for the description of the process of the conversion of more specialized mathematical objects to their generalizations.

The terminology was adapted from the informal imperative object oriented programming, where the term slicing often refers to the process of copying an object of a subclass type to an object of a superclass type [5][1]. However, it is important to note that the term has other meanings in programming and computer science.

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

Components.

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

Regular definitions.

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

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

Composition and composable arrows

The definition of a set of *composable-arrs* is equivalent to the definition of *composable pairs* presented on page 10 in [39] (see theorem *dg-composable-arrs′* below). Nonetheless, the definition is meant to be used sparingly. Normally, the arrows are meant to be specified explicitly using the predicate *is-arr*.

definition *Comp* :: V
  where [*dg-field-simps*]: *Comp* = $4_{\mathbf{N}}$

abbreviation *Comp-app* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$ (**infixl** $\langle \circ_{A1} \rangle$ 55)
  where *Comp-app* $\mathfrak{C}$ a b ≡ $\mathfrak{C}(\!|Comp|\!)(\!|a, b|\!)_\bullet$

definition *composable-arrs* :: $V \Rightarrow V$
  where *composable-arrs* $\mathfrak{C}$ = *set*
    $\{[g, f]_\circ \mid g\ f.\ \exists a\ b\ c.\ g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b\}$

lemma *small-composable-arrs*[*simp*]:
  small $\{[g, f]_\circ \mid g\ f.\ \exists a\ b\ c.\ g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b\}$
proof(*intro down*[*of* - $\langle \mathfrak{C}(\!|Arr|\!) \mathbin{\widehat{\ }}_\times 2_{\mathbf{N}} \rangle$] *subsetI*)
  fix x **assume** $x \in \{[g, f]_\circ \mid g\ f.\ \exists a\ b\ c.\ g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b\}$
  then obtain $g\ f\ a\ b\ c$
    where *x-def*: $x = [g, f]_\circ$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$

[1] https://en.wikipedia.org/wiki/Object_slicing

by *clarsimp*
with *vfsequence-vcpower-two-vpair* **show** $x \in_{\circ} \mathfrak{C}(\!|Arr|\!) \mathbin{\widehat{\times}} 2_{\mathbb{N}}$
unfolding *x-def* **by** *auto*
qed

Rules.

lemma *composable-arrsI*[*smc-cs-intros*]:
assumes $gf = [g, f]_{\circ}$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
shows $gf \in_{\circ}$ *composable-arrs* $\mathfrak{C}$
using *assms*(2,3) *small-composable-arrs*
unfolding *assms*(1) *composable-arrs-def*
by *auto*

lemma *composable-arrsE*[*elim*!]:
assumes $gf \in_{\circ}$ *composable-arrs* $\mathfrak{C}$
obtains $g\ f\ a\ b\ c$ **where** $gf = [g, f]_{\circ}$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
using *assms small-composable-arrs* **unfolding** *composable-arrs-def* **by** *clarsimp*

lemma *small-composable-arrs'*[*simp*]:
$small\ \{[g, f]_{\circ} \mid g\ f.\ g \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge \mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)\}$
proof(*intro down*[*of* - ‹$\mathfrak{C}(\!|Arr|\!) \mathbin{\widehat{\times}} 2_{\mathbb{N}}$›] *subsetI*)
fix *gf* **assume**
$gf \in \{[g, f]_{\circ} \mid g\ f.\ g \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge \mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)\}$
then obtain *g f*
where *gf-def*: $gf = [g, f]_{\circ}$
and $g \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
and $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
and $\mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$
by *clarsimp*
with *vfsequence-vcpower-two-vpair* **show** $gf \in_{\circ} \mathfrak{C}(\!|Arr|\!) \mathbin{\widehat{\times}} 2_{\mathbb{N}}$
unfolding *gf-def* **by** *auto*
qed

lemma *dg-composable-arrs'*:
$set\ \{[g, f]_{\circ} \mid g\ f.\ g \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge \mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)\} =$
composable-arrs $\mathfrak{C}$
proof–
have $\{[g, f]_{\circ} \mid g\ f.\ g \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge f \in_{\circ} \mathfrak{C}(\!|Arr|\!) \wedge \mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)\} =$
$\{[g, f]_{\circ} \mid g\ f.\ \exists\, a\ b\ c.\ g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b\}$
proof(*intro subset-antisym subsetI*, *unfold mem-Collect-eq*; *elim exE conjE*)
fix *gf g f*
assume *gf-def*: $gf = [g, f]_{\circ}$
and $g \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
and $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
and *gf*: $\mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$
then obtain $a\ b\ b'\ c$ **where** *g*: $g : b' \mapsto_{\mathfrak{C}} c$ **and** *f*: $f : a \mapsto_{\mathfrak{C}} b$
by (*auto intro*!: *is-arrI*)
moreover have $b' = b$
unfolding *is-arrD*(2,3)[*OF g*, *symmetric*] *is-arrD*(2,3)[*OF f*, *symmetric*]
by (*rule gf*)
ultimately have $\exists\, a\ b\ c.\ g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b$ **by** *auto*
then show $\exists\, g\ f.\ gf = [g, f]_{\circ} \wedge (\exists\, a\ b\ c.\ g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b)$
unfolding *gf-def* **by** *auto*
next
fix *gf g f a b c*
assume *gf-def*: $gf = [g, f]_{\circ}$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
then have $g \in_{\circ} \mathfrak{C}(\!|Arr|\!)\ f \in_{\circ} \mathfrak{C}(\!|Arr|\!)\ \mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$ **by** *auto*
then show

$\exists g\ f.\ gf = [g, f]_\circ \wedge g \in_\circ \mathfrak{C}(\!|Arr|\!) \wedge f \in_\circ \mathfrak{C}(\!|Arr|\!) \wedge \mathfrak{C}(\!|Dom|\!)(\!|g|\!) = \mathfrak{C}(\!|Cod|\!)(\!|f|\!)$
unfolding *gf-def* **by** *auto*
qed
then show *?thesis* **unfolding** *composable-arrs-def* **by** *auto*
qed

4.2.2 Definition and elementary properties

The definition of a semicategory that is used in this work is similar to the definition that was used in [42]. It is also a natural generalization of the definition of a category that is presented in Chapter I-2 in [39]. The generalization is performed by omitting the identity and the axioms associated with it. The amendments to the definitions that are associated with size have already been explained in the previous chapter.

locale *semicategory* = $\mathcal{Z}\ \alpha$ + *vfsequence* $\mathfrak{C}$ + *Comp*: *vsv* $\langle\mathfrak{C}(\!|Comp|\!)\rangle$ **for** $\alpha\ \mathfrak{C}$ +
assumes *smc-length*[*smc-cs-simps*]: *vcard* $\mathfrak{C} = 5_{\mathbb{N}}$
and *smc-digraph*[*slicing-intros*]: *digraph* α (*smc-dg* $\mathfrak{C}$)
and *smc-Comp-vdomain*: $gf \in_\circ \mathcal{D}_\circ(\mathfrak{C}(\!|Comp|\!)) \longleftrightarrow$
$(\exists g\ f\ b\ c\ a.\ gf = [g, f]_\circ \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b)$
and *smc-Comp-is-arr*:
$[\![\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
and *smc-Comp-assoc*[*smc-cs-simps*]:
$[\![\ h : c \mapsto_{\mathfrak{C}} d;\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
$(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$

lemmas [*smc-cs-simps*] =
semicategory.smc-length
semicategory.smc-Comp-assoc

lemma (**in** *semicategory*) *smc-Comp-is-arr'*[*smc-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 smc-Comp-is-arr*)

lemmas [*smc-cs-intros*] =
semicategory.smc-Comp-is-arr'
semicategory.smc-Comp-is-arr

lemmas [*slicing-intros*] = *semicategory.smc-digraph*

Rules.

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

mk-ide rf *semicategory-def*[*unfolded semicategory-axioms-def*]
|*intro semicategoryI*|
|*dest semicategoryD*[*dest*]|
|*elim semicategoryE*[*elim*]|

lemma *semicategoryI'*:
assumes $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{C}$
and *vsv* $(\mathfrak{C}(\!|Comp|\!))$
and *vcard* $\mathfrak{C} = 5_{\mathbb{N}}$

and *vsv* $(\mathfrak{C}(\!|Dom|\!))$
and *vsv* $(\mathfrak{C}(\!|Cod|\!))$
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 $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{C}} c; f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
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 $\mathfrak{C}(\!|Obj|\!) \subseteq_\circ Vset\ \alpha$
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 *semicategory* $\alpha\ \mathfrak{C}$
by (*intro semicategoryI digraphI*, *unfold slicing-simps*)
(*simp-all add*: *assms nat-omega-simps smc-dg-def*)

lemma *semicategoryD′*:
assumes *semicategory* $\alpha\ \mathfrak{C}$
shows $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{C}$
and *vsv* $(\mathfrak{C}(\!|Comp|\!))$
and *vcard* $\mathfrak{C} = 5_{\mathbb{N}}$
and *vsv* $(\mathfrak{C}(\!|Dom|\!))$
and *vsv* $(\mathfrak{C}(\!|Cod|\!))$
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 $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{C}} c; f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
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 $\mathfrak{C}(\!|Obj|\!) \subseteq_\circ Vset\ \alpha$
and $\bigwedge A\ B.\ [\![\ A \subseteq_\circ \mathfrak{C}(\!|Obj|\!); B \subseteq_\circ \mathfrak{C}(\!|Obj|\!); A \in_\circ Vset\ \alpha; B \in_\circ Vset\ \alpha\ ]\!] \Longrightarrow$
$(\bigcup_\circ a \in_\circ A.\ \bigcup_\circ b \in_\circ B.\ Hom\ \mathfrak{C}\ a\ b) \in_\circ Vset\ \alpha$
by
(
simp-all add:
semicategoryD(2–8)[*OF assms*]
digraphD[*OF semicategoryD*(5)[*OF assms*], *unfolded slicing-simps*]
)

lemma *semicategoryE′*:
assumes *semicategory* $\alpha\ \mathfrak{C}$
obtains $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{C}$
and *vsv* $(\mathfrak{C}(\!|Comp|\!))$
and *vcard* $\mathfrak{C} = 5_{\mathbb{N}}$
and *vsv* $(\mathfrak{C}(\!|Dom|\!))$
and *vsv* $(\mathfrak{C}(\!|Cod|\!))$
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 $\bigwedge gf.\ gf \in_\circ \mathcal{D}_\circ\ (\mathfrak{C}(\!|Comp|\!)) \longleftrightarrow$

```
    (∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦ℭ c ∧ f : a ↦ℭ b)
  and ⋀b c g a f. ⟦ g : b ↦ℭ c; f : a ↦ℭ b ⟧ ⟹ g ∘A ℭ f : a ↦ℭ c
  and ⋀c d h b g a f. ⟦ h : c ↦ℭ d; g : b ↦ℭ c; f : a ↦ℭ b ⟧ ⟹
      (h ∘A ℭ g) ∘A ℭ f = h ∘A ℭ (g ∘A ℭ f)
  and ℭ(|Obj|) ⊆∘ Vset α
  and ⋀A B. ⟦ A ⊆∘ ℭ(|Obj|); B ⊆∘ ℭ(|Obj|); A ∈∘ Vset α; B ∈∘ Vset α ⟧ ⟹
      (⋃∘a∈∘A. ⋃∘b∈∘B. Hom ℭ a b) ∈∘ Vset α
using assms by (simp add: semicategoryD′)
```

While using the sublocale infrastructure in conjunction with the rewrite morphisms is plausible for achieving automation of slicing, this approach has certain limitations. For example, the rewrite morphisms cannot be added to a given interpretation that was achieved using the command **sublocale**[2]. Thus, instead of using a partial solution based on the command **sublocale**, the rewriting is performed manually for selected theorems. However, it is hoped that better automation will be provided in the future.

```
context semicategory
begin

interpretation dg: digraph α ‹smc-dg ℭ› by (rule smc-digraph)

sublocale Dom: vsv ‹ℭ(|Dom|)› by (rule dg.Dom.vsv-axioms[unfolded slicing-simps])
sublocale Cod: vsv ‹ℭ(|Cod|)› by (rule dg.Cod.vsv-axioms[unfolded slicing-simps])

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

end

lemmas [smc-cs-intros] =
  semicategory.smc-is-arrD(1–3)
```

[2] https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2019-September/msg00074.html

semicategory.smc-Hom-in-Vset

Elementary properties.

lemma *smc-eqI*:
assumes *semicategory* α $\mathfrak{A}$
and *semicategory* 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 $\mathfrak{A}(\lfloor Comp \rfloor) = \mathfrak{B}(\lfloor Comp \rfloor)$
shows $\mathfrak{A} = \mathfrak{B}$
proof–
interpret $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule assms*(1))
interpret $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule vsv-eqI*)
have *dom*: $\mathcal{D}_\circ\ \mathfrak{A} = 5_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps V-cs-simps*)
show $\mathcal{D}_\circ\ \mathfrak{A} = \mathcal{D}_\circ\ \mathfrak{B}$
by (*cs-concl* **cs-shallow cs-simp**: *dom smc-cs-simps V-cs-simps*)
show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{A} \Longrightarrow \mathfrak{A}(\lfloor a \rfloor) = \mathfrak{B}(\lfloor a \rfloor)$ **for** *a*
by (*unfold dom*, *elim-in-numeral*, *insert assms*) (*auto simp*: *dg-field-simps*)
qed *auto*
qed

lemma *smc-dg-eqI*:
assumes *semicategory* α $\mathfrak{A}$
and *semicategory* α $\mathfrak{B}$
and $\mathfrak{A}(\lfloor Comp \rfloor) = \mathfrak{B}(\lfloor Comp \rfloor)$
and *smc-dg* $\mathfrak{A}$ = *smc-dg* $\mathfrak{B}$
shows $\mathfrak{A} = \mathfrak{B}$
proof(*rule smc-eqI*)
from *assms*(4) **have**
smc-dg $\mathfrak{A}(\lfloor Obj \rfloor)$ = *smc-dg* $\mathfrak{B}(\lfloor Obj \rfloor)$
smc-dg $\mathfrak{A}(\lfloor Arr \rfloor)$ = *smc-dg* $\mathfrak{B}(\lfloor Arr \rfloor)$
smc-dg $\mathfrak{A}(\lfloor Dom \rfloor)$ = *smc-dg* $\mathfrak{B}(\lfloor Dom \rfloor)$
smc-dg $\mathfrak{A}(\lfloor Cod \rfloor)$ = *smc-dg* $\mathfrak{B}(\lfloor Cod \rfloor)$
by *auto*
then show
$\mathfrak{A}(\lfloor Obj \rfloor) = \mathfrak{B}(\lfloor Obj \rfloor)$ $\mathfrak{A}(\lfloor Arr \rfloor) = \mathfrak{B}(\lfloor Arr \rfloor)$ $\mathfrak{A}(\lfloor Dom \rfloor) = \mathfrak{B}(\lfloor Dom \rfloor)$ $\mathfrak{A}(\lfloor Cod \rfloor) = \mathfrak{B}(\lfloor Cod \rfloor)$
unfolding *slicing-simps* **by** *simp-all*
qed (*auto intro*: *assms*)

lemma (**in** *semicategory*) *smc-def*: $\mathfrak{C} = [\mathfrak{C}(\lfloor Obj \rfloor), \mathfrak{C}(\lfloor Arr \rfloor), \mathfrak{C}(\lfloor Dom \rfloor), \mathfrak{C}(\lfloor Cod \rfloor), \mathfrak{C}(\lfloor Comp \rfloor)]_\circ$
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ\ \mathfrak{C} = 5_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps V-cs-simps*)
have *dom-rhs*: $\mathcal{D}_\circ\ [\mathfrak{C}(\lfloor Obj \rfloor), \mathfrak{C}(\lfloor Arr \rfloor), \mathfrak{C}(\lfloor Dom \rfloor), \mathfrak{C}(\lfloor Cod \rfloor), \mathfrak{C}(\lfloor Comp \rfloor)]_\circ = 5_{\mathbb{N}}$
by (*simp add*: *nat-omega-simps*)
then show $\mathcal{D}_\circ\ \mathfrak{C} = \mathcal{D}_\circ\ [\mathfrak{C}(\lfloor Obj \rfloor), \mathfrak{C}(\lfloor Arr \rfloor), \mathfrak{C}(\lfloor Dom \rfloor), \mathfrak{C}(\lfloor Cod \rfloor), \mathfrak{C}(\lfloor Comp \rfloor)]_\circ$
unfolding *dom-lhs dom-rhs* **by** *simp*
show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{C} \Longrightarrow \mathfrak{C}(\lfloor a \rfloor) = [\mathfrak{C}(\lfloor Obj \rfloor), \mathfrak{C}(\lfloor Arr \rfloor), \mathfrak{C}(\lfloor Dom \rfloor), \mathfrak{C}(\lfloor Cod \rfloor), \mathfrak{C}(\lfloor Comp \rfloor)]_\circ(\lfloor a \rfloor)$
for *a*
unfolding *dom-lhs*
by *elim-in-numeral* (*simp-all add*: *dg-field-simps nat-omega-simps*)
qed *auto*

lemma (in *semicategory*) *smc-Comp-vdomainI*[*smc-cs-intros*]:
assumes $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **and** $gf = [g, f]_{\circ}$
shows $gf \in_{\circ} \mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!))$
using *assms* **by** (*intro smc-Comp-vdomain*[*THEN iffD2*]) *auto*

lemmas [*smc-cs-intros*] = *semicategory.smc-Comp-vdomainI*

lemma (in *semicategory*) *smc-Comp-vdomainE*[*elim*!]:
assumes $gf \in_{\circ} \mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!))$
obtains *g f a b c* **where** $gf = [g, f]_{\circ}$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
proof–
from *smc-Comp-vdomain*[*THEN iffD1*, *OF assms*(1)] **obtain** *g f b c a*
where $gf = [g, f]_{\circ}$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
by *clarsimp*
with *that* **show** *?thesis* **by** *simp*
qed

lemma (in *semicategory*) *smc-Comp-vdomain-is-composable-arrs*:
$\mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!)) =$ *composable-arrs* $\mathfrak{C}$
by (*intro vsubset-antisym vsubsetI*) (*auto intro*!: *smc-cs-intros*)+

lemma (in *semicategory*) *smc-Comp-vrange*: $\mathcal{R}_{\circ}(\mathfrak{C}(\!|Comp|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!)$
proof(*rule Comp.vsv-vrange-vsubset*)
fix *gf* **assume** $gf \in_{\circ} \mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!))$
from *smc-Comp-vdomain*[*THEN iffD1*, *OF this*] **obtain** *g f b c a*
where *gf-def*: $gf = [g, f]_{\circ}$
and *g*: $g : b \mapsto_{\mathfrak{C}} c$
and *f*: $f : a \mapsto_{\mathfrak{C}} b$
by *clarsimp*
from *semicategory-axioms g f* **show** $\mathfrak{C}(\!|Comp|\!)(\!|gf|\!) \in_{\circ} \mathfrak{C}(\!|Arr|\!)$
by
(
cs-concl **cs-shallow**
cs-simp: *gf-def smc-cs-simps* **cs-intro**: *smc-cs-intros*
)
qed

sublocale *semicategory* $\subseteq$ *Comp*: *pbinop* $\langle\mathfrak{C}(\!|Arr|\!)\rangle$ $\langle\mathfrak{C}(\!|Comp|\!)\rangle$
proof *unfold-locales*
show $\mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!) \mathbin{\widehat{\times}} 2_{\mathbb{N}}$
proof(*intro vsubsetI*; *unfold smc-Comp-vdomain*)
fix *gf* **assume** $\exists\, g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b$
then obtain *a b c g f*
where *x-def*: $gf = [g, f]_{\circ}$ **and** $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$
by *auto*
then have $g \in_{\circ} \mathfrak{C}(\!|Arr|\!)$ $f \in_{\circ} \mathfrak{C}(\!|Arr|\!)$ **by** *auto*
then show $gf \in_{\circ} \mathfrak{C}(\!|Arr|\!) \mathbin{\widehat{\times}} 2_{\mathbb{N}}$
unfolding *x-def* **by** (*auto simp*: *nat-omega-simps*)
qed
show $\mathcal{R}_{\circ}(\mathfrak{C}(\!|Comp|\!)) \subseteq_{\circ} \mathfrak{C}(\!|Arr|\!)$ **by** (*rule smc-Comp-vrange*)
qed *auto*

Size.

lemma (in *semicategory*) *smc-Comp-vsubset-Vset*: $\mathfrak{C}(\!|Comp|\!) \subseteq_{\circ} Vset\ \alpha$
proof(*intro vsubsetI*)
fix *gfh* **assume** $gfh \in_{\circ} \mathfrak{C}(\!|Comp|\!)$
then obtain *gf h*
where *gfh-def*: $gfh = \langle gf, h \rangle$

```
    and gf: gf ∈∘ 𝒟∘ (𝔈(|Comp|))
    and h: h ∈∘ ℛ∘ (𝔈(|Comp|))
  by (blast elim: Comp.vbrelation-vinE)
from gf obtain g f a b c
  where gf-def: gf = [g, f]∘ and g: g : b ↦.𝔈 c and f: f : a ↦.𝔈 b
  by clarsimp
from h smc-Comp-vrange have h ∈∘ 𝔈(|Arr|) by auto
with g f show gfh ∈∘ Vset α
  unfolding gfh-def gf-def
  by (cs-concl cs-shallow cs-intro: smc-cs-intros V-cs-intros)
qed

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

lemma (in semicategory) smc-Comp-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔈(|Comp|) ∈∘ Vset β
  using smc-Comp-vsubset-Vset by (meson Vset-in-mono assms(2) vsubset-in-VsetI)

lemma (in semicategory) smc-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔈 ∈∘ Vset β
proof–
  interpret β: 𝒵 β by (rule assms(1))
  note [smc-cs-intros] =
    smc-Obj-in-Vset
    smc-Arr-in-Vset
    smc-Dom-in-Vset
    smc-Cod-in-Vset
    smc-Comp-in-Vset
  from assms(2) show ?thesis
    by (subst smc-def) (cs-concl cs-shallow cs-intro: smc-cs-intros V-cs-intros)
qed

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

lemma *small-semicategory*[*simp*]: *small* {𝔈. *semicategory* α 𝔈}
proof(*cases* ‹𝒵 α›)
case *True*
from *semicategory.smc-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. *semicategory* α 𝔈} = {} **by** *auto*
then show *?thesis* **by** *simp*
qed

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

lemma *semicategory-if-semicategory*:
assumes *semicategory* 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 *semicategory* α 𝔈
proof–
interpret *semicategory* β 𝔈 **by** (*rule assms*(1))
interpret α: 𝒵 α **by** (*rule assms*(2))
show *?thesis*
proof(*intro semicategoryI*)
show *vfsequence* 𝔈 **by** (*simp add*: *vfsequence-axioms*)
show *digraph* α (*smc-dg* 𝔈)
by (*rule digraph-if-digraph*, *unfold slicing-simps*)
(*auto intro*!: *assms*(1,3,4) *slicing-intros*)
qed (*auto intro*: *smc-cs-intros simp*: *smc-cs-simps*)
qed

Further properties.

lemma (**in** *semicategory*) *smc-Comp-vempty-if-Arr-vempty*:
assumes 𝔈(|*Arr*|) = 0
shows 𝔈(|*Comp*|) = 0
using *assms smc-Comp-vrange* **by** (*auto intro*: *Comp.vsv-vrange-vempty*)

4.2.3 Opposite semicategory

Definition and elementary properties

See Chapter II-2 in [39].

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

Components.

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

lemma *op-smc-component-intros*[*smc-op-intros*]:
  shows $a \in_\circ \mathfrak{C}(\!|Obj|\!) \Longrightarrow a \in_\circ$ *op-smc* $\mathfrak{C}(\!|Obj|\!)$
    and $f \in_\circ \mathfrak{C}(\!|Arr|\!) \Longrightarrow f \in_\circ$ *op-smc* $\mathfrak{C}(\!|Arr|\!)$
  unfolding *smc-op-simps* **by** *simp-all*

Slicing.

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

Regular definitions.

lemma *op-smc-Comp-vdomain*[*smc-op-simps*]:
  $\mathcal{D}_\circ$ (*op-smc* $\mathfrak{C}(\!|Comp|\!)$) = $(\mathcal{D}_\circ\ (\mathfrak{C}(\!|Comp|\!)))^{-1}_\bullet$
  unfolding *op-smc-components* **by** *simp*

lemma *op-smc-is-arr*[*smc-op-simps*]: $f : b \mapsto_{\mathit{op\text{-}smc}\ \mathfrak{C}} a \longleftrightarrow f : a \mapsto_{\mathfrak{C}} b$
  unfolding *smc-op-simps is-arr-def* **by** *auto*

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

lemma (**in** *semicategory*) *op-smc-Comp-vrange*[*smc-op-simps*]:
  $\mathcal{R}_\circ$ (*op-smc* $\mathfrak{C}(\!|Comp|\!)$) = $\mathcal{R}_\circ\ (\mathfrak{C}(\!|Comp|\!))$
  using *Comp.vrange-fflip* **unfolding** *op-smc-components* **by** *simp*

lemmas [*smc-op-simps*] = *semicategory.op-smc-Comp-vrange*

lemma (**in** *semicategory*) *op-smc-Comp*[*smc-op-simps*]:
  assumes $f : b \mapsto_{\mathfrak{C}} c$ **and** $g : a \mapsto_{\mathfrak{C}} b$
  shows $g \circ_{A\ \mathit{op\text{-}smc}\ \mathfrak{C}} f = f \circ_{A\ \mathfrak{C}} g$
  using *assms*
  unfolding *op-smc-components*
  by (*auto intro*!: *fflip-app smc-cs-intros*)

lemmas [*smc-op-simps*] = *semicategory.op-smc-Comp*

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

Further properties

lemma (**in** *semicategory*) *semicategory-op*[*smc-op-intros*]:

```
  semicategory α (op-smc 𝔈)
proof(intro semicategoryI)
  from semicategory-axioms smc-digraph show digraph α (smc-dg (op-smc 𝔈))
    by
      (
        cs-concl cs-shallow
          cs-simp: slicing-commute[symmetric] cs-intro: dg-op-intros
      )
  show vfsequence (op-smc 𝔈) unfolding op-smc-def by simp
  show vcard (op-smc 𝔈) = 5ℕ
    unfolding op-smc-def by (simp add: nat-omega-simps)
  show (gf ∈∘ 𝒟∘ (op-smc 𝔈(|Comp|))) ⟷
    (∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦op-smc 𝔈 c ∧ f : a ↦op-smc 𝔈 b)
    for gf
  proof(rule iffI; unfold smc-op-simps)
    assume prems: gf ∈∘ (𝒟∘ (𝔈(|Comp|)))⁻¹∘
    then obtain g′ f′ where gf-def: gf = [g′, f′]∘ by clarsimp
    with prems have [f′, g′]∘ ∈∘ 𝒟∘ (𝔈(|Comp|)) by (auto intro: smc-cs-intros)
    with smc-Comp-vdomain show
      ∃ g f b c a. gf = [g, f]∘ ∧ g : c ↦𝔈 b ∧ f : b ↦𝔈 a
      unfolding gf-def by auto
  next
    assume ∃ g f b c a. gf = [g, f]∘ ∧ g : c ↦𝔈 b ∧ f : b ↦𝔈 a
    then obtain g f b c a
      where gf-def: gf = [g, f]∘ and g: g : c ↦𝔈 b and f: f : b ↦𝔈 a
      by clarsimp
    then have g ∈∘ 𝔈(|Arr|) and f ∈∘ 𝔈(|Arr|) by force+
    from g f have [f, g]∘ ∈∘ 𝒟∘ (𝔈(|Comp|))
      unfolding gf-def by (intro smc-Comp-vdomainI) auto
    then show gf ∈∘ (𝒟∘ (𝔈(|Comp|)))⁻¹∘
      unfolding gf-def by (auto intro: smc-cs-intros)
  qed
  from semicategory-axioms show
    ⟦ g : b ↦op-smc 𝔈 c; f : a ↦op-smc 𝔈 b ⟧ ⟹
      g ∘A op-smc 𝔈 f : a ↦op-smc 𝔈 c
    for g b c f a
    unfolding smc-op-simps
    by (cs-concl cs-shallow cs-simp: smc-op-simps cs-intro: smc-cs-intros)
  fix h c d g b f a
  assume h : c ↦op-smc 𝔈 d g : b ↦op-smc 𝔈 c f : a ↦op-smc 𝔈 b
  with semicategory-axioms show
    (h ∘A op-smc 𝔈 g) ∘A op-smc 𝔈 f = h ∘A op-smc 𝔈 (g ∘A op-smc 𝔈 f)
    unfolding smc-op-simps
    by (cs-concl cs-simp: smc-op-simps smc-cs-simps cs-intro: smc-cs-intros)
qed (auto simp: fflip-vsv op-smc-components(5))

lemmas semicategory-op[smc-op-intros] = semicategory.semicategory-op

lemma (in semicategory) smc-op-smc-op-smc[smc-op-simps]: op-smc (op-smc 𝔈) = 𝔈
  by (rule smc-eqI, unfold smc-op-simps op-smc-components)
    (
      auto simp:
        Comp.pbinop-fflip-fflip
        semicategory-axioms
        semicategory.semicategory-op semicategory-op
        intro: smc-cs-intros
    )
```

lemmas *smc-op-smc-op-smc*[*smc-op-simps*] = *semicategory.smc-op-smc-op-smc*

lemma *eq-op-smc-iff*[*smc-op-simps*]:
assumes *semicategory* α $\mathfrak{A}$ **and** *semicategory* α $\mathfrak{B}$
shows *op-smc* $\mathfrak{A}$ = *op-smc* $\mathfrak{B}$ $\longleftrightarrow$ $\mathfrak{A}$ = $\mathfrak{B}$
proof
interpret $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule assms*(1))
interpret $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule assms*(2))
assume *prems*: *op-smc* $\mathfrak{A}$ = *op-smc* $\mathfrak{B}$ **show** $\mathfrak{A}$ = $\mathfrak{B}$
proof(*rule smc-eqI*)
show
$\mathfrak{A}(\!|Obj|\!) = \mathfrak{B}(\!|Obj|\!)$
$\mathfrak{A}(\!|Arr|\!) = \mathfrak{B}(\!|Arr|\!)$
$\mathfrak{A}(\!|Dom|\!) = \mathfrak{B}(\!|Dom|\!)$
$\mathfrak{A}(\!|Cod|\!) = \mathfrak{B}(\!|Cod|\!)$
$\mathfrak{A}(\!|Comp|\!) = \mathfrak{B}(\!|Comp|\!)$
by (*metis prems* $\mathfrak{A}$.*smc-op-smc-op-smc* $\mathfrak{B}$.*smc-op-smc-op-smc*)+
qed (*auto intro*: *assms*)
qed *auto*

4.2.4 Arrow with a domain and a codomain

lemma (**in** *semicategory*) *smc-assoc-helper*:
assumes $f : a \mapsto_{\mathfrak{C}} b$
and $g : b \mapsto_{\mathfrak{C}} c$
and $h : c \mapsto_{\mathfrak{C}} d$
and $h \circ_{A\mathfrak{C}} g = q$
shows $h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f) = q \circ_{A\mathfrak{C}} f$
using *semicategory-axioms assms*(1–4)
by (*cs-concl* **cs-simp**: *assms*(4) *semicategory.smc-Comp-assoc*[*symmetric*])

lemma (**in** *semicategory*) *smc-pattern-rectangle-right*:
assumes $aa' : a \mapsto_{\mathfrak{C}} a'$
and $a'a'' : a' \mapsto_{\mathfrak{C}} a''$
and $a''b'' : a'' \mapsto_{\mathfrak{C}} b''$
and $ab : a \mapsto_{\mathfrak{C}} b$
and $bb' : b \mapsto_{\mathfrak{C}} b'$
and $b'b'' : b' \mapsto_{\mathfrak{C}} b''$
and $a'b' : a' \mapsto_{\mathfrak{C}} b'$
and $a'b' \circ_{A\mathfrak{C}} aa' = bb' \circ_{A\mathfrak{C}} ab$
and $b'b'' \circ_{A\mathfrak{C}} a'b' = a''b'' \circ_{A\mathfrak{C}} a'a''$
shows $a''b'' \circ_{A\mathfrak{C}} (a'a'' \circ_{A\mathfrak{C}} aa') = (b'b'' \circ_{A\mathfrak{C}} bb') \circ_{A\mathfrak{C}} ab$
proof–
from *semicategory-axioms assms*(3,2,1) **have**
$a''b'' \circ_{A\mathfrak{C}} (a'a'' \circ_{A\mathfrak{C}} aa') = (a''b'' \circ_{A\mathfrak{C}} a'a'') \circ_{A\mathfrak{C}} aa'$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
also have ... = $(b'b'' \circ_{A\mathfrak{C}} a'b') \circ_{A\mathfrak{C}} aa'$ **unfolding** *assms*(9) **..**
also from *semicategory-axioms assms*(1,6,7) **have**
... = $b'b'' \circ_{A\mathfrak{C}} (a'b' \circ_{A\mathfrak{C}} aa')$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
also have ... = $b'b'' \circ_{A\mathfrak{C}} (bb' \circ_{A\mathfrak{C}} ab)$ **unfolding** *assms*(8) **..**
also from *semicategory-axioms assms*(6,5,4) **have**
... = $(b'b'' \circ_{A\mathfrak{C}} bb') \circ_{A\mathfrak{C}} ab$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
finally show *?thesis* **by** *simp*
qed

lemmas (**in** *semicategory*) *smc-pattern-rectangle-left* =

smc-pattern-rectangle-right[*symmetric*]

4.2.5 Monic arrow and epic arrow

See Chapter I-5 in [39].

definition *is-monic-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *is-monic-arr* $\mathfrak{C}$ b c $m \longleftrightarrow$
    $m : b \mapsto_{\mathfrak{C}} c \wedge$
    (
      $\forall f\ g\ a.$
        $f : a \mapsto_{\mathfrak{C}} b \longrightarrow g : a \mapsto_{\mathfrak{C}} b \longrightarrow m \circ_{A\mathfrak{C}} f = m \circ_{A\mathfrak{C}} g \longrightarrow f = g$
)

syntax *-is-monic-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  ($\langle$- : - $\mapsto_{mon}$1 -$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-monic-arr* $\rightleftharpoons$ *is-monic-arr*
translations $m : b \mapsto_{mon\mathfrak{C}} c \rightleftharpoons$ *CONST is-monic-arr* $\mathfrak{C}$ b c m

definition *is-epic-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *is-epic-arr* $\mathfrak{C}$ a b $e \equiv e : b \mapsto_{mon\, op\text{-}smc\, \mathfrak{C}} a$

syntax *-is-epic-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  ($\langle$- : - $\mapsto_{epi}$1 -$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-epic-arr* $\rightleftharpoons$ *is-epic-arr*
translations $e : a \mapsto_{epi\mathfrak{C}} b \rightleftharpoons$ *CONST is-epic-arr* $\mathfrak{C}$ a b e

Rules.

mk-ide rf *is-monic-arr-def*
  |*intro is-monic-arrI*|
  |*dest is-monic-arrD*[*dest*]|
  |*elim is-monic-arrE*[*elim*!]|

lemmas [*smc-arrow-cs-intros*] = *is-monic-arrD*(1)

lemma (**in** *semicategory*) *is-epic-arrI*:
  assumes $e : a \mapsto_{\mathfrak{C}} b$
    and $\bigwedge f\ g\ c.\ [\![\ f : b \mapsto_{\mathfrak{C}} c;\ g : b \mapsto_{\mathfrak{C}} c;\ f \circ_{A\mathfrak{C}} e = g \circ_{A\mathfrak{C}} e\ ]\!] \Longrightarrow$
      $f = g$
  shows $e : a \mapsto_{epi\mathfrak{C}} b$
  unfolding *is-epic-arr-def*
proof(*intro is-monic-arrI*, *unfold smc-op-simps*)
  fix $f\ g\ a$
  assume *prems*:
    $f : b \mapsto_{\mathfrak{C}} a\ g : b \mapsto_{\mathfrak{C}} a\ e \circ_{A\, op\text{-}smc\, \mathfrak{C}} f = e \circ_{A\, op\text{-}smc\, \mathfrak{C}} g$
  show $f = g$
  proof–
    from *prems*(3,1,2) *assms*(1) *semicategory-axioms* **have** $g \circ_{A\mathfrak{C}} e = f \circ_{A\mathfrak{C}} e$
      by
        (
          cs-prems **cs-shallow**
            cs-simp: *smc-cs-simps smc-op-simps*
            cs-intro: *smc-cs-intros smc-op-intros*
)
      simp
    from *assms*(2)[*OF prems*(2,1) *this*] **show** *?thesis* **..**
  qed
qed (*rule assms*(1))

lemma *is-epic-arr-is-arr*[*smc-arrow-cs-intros*, *dest*]:
assumes $e : a \mapsto_{epi\mathfrak{C}} b$
shows $e : a \mapsto_{\mathfrak{C}} b$
using *assms* **unfolding** *is-epic-arr-def is-monic-arr-def smc-op-simps* **by** *simp*

lemma (**in** *semicategory*) *is-epic-arrD*[*dest*]:
assumes $e : a \mapsto_{epi\mathfrak{C}} b$
shows $e : a \mapsto_{\mathfrak{C}} b$
and $\bigwedge f\ g\ c.\ [\![f : b \mapsto_{\mathfrak{C}} c;\ g : b \mapsto_{\mathfrak{C}} c;\ f \circ_{A\mathfrak{C}} e = g \circ_{A\mathfrak{C}} e]\!] \Longrightarrow f = g$
proof–
note *is-monic-arrD* = *assms*(1)[*unfolded is-epic-arr-def is-monic-arr-def smc-op-simps*]
from *is-monic-arrD*[*THEN conjunct*1] **show** *e*: $e : a \mapsto_{\mathfrak{C}} b$ **by** *simp*
fix $f\ g\ c$
assume *prems*: $f : b \mapsto_{\mathfrak{C}} c\ g : b \mapsto_{\mathfrak{C}} c\ f \circ_{A\mathfrak{C}} e = g \circ_{A\mathfrak{C}} e$
with *semicategory-axioms e* **have** $e \circ_{A\ op\text{-}smc\ \mathfrak{C}} f = e \circ_{A\ op\text{-}smc\ \mathfrak{C}} g$
by (*cs-concl* **cs-shallow cs-simp**: *smc-op-simps* **cs-intro**: *smc-cs-intros*)
then show $f = g$
by (*rule is-monic-arrD*[*THEN conjunct*2, *rule-format*, *OF prems*(1,2)])
qed

lemma (**in** *semicategory*) *is-epic-arrE*[*elim*!]:
assumes $e : a \mapsto_{epi\mathfrak{C}} b$
obtains $e : a \mapsto_{\mathfrak{C}} b$
and $\bigwedge f\ g\ c.\ [\![f : b \mapsto_{\mathfrak{C}} c;\ g : b \mapsto_{\mathfrak{C}} c;\ f \circ_{A\mathfrak{C}} e = g \circ_{A\mathfrak{C}} e]\!] \Longrightarrow f = g$
using *assms* **by** *auto*

Elementary properties.

lemma (**in** *semicategory*) *op-smc-is-epic-arr*[*smc-op-simps*]:
$f : b \mapsto_{epi\ op\text{-}smc\ \mathfrak{C}} a \longleftrightarrow f : a \mapsto_{mon\mathfrak{C}} b$
unfolding *is-monic-arr-def is-epic-arr-def smc-op-simps* **..**

lemma (**in** *semicategory*) *op-smc-is-monic-arr*[*smc-op-simps*]:
$f : b \mapsto_{mon\ op\text{-}smc\ \mathfrak{C}} a \longleftrightarrow f : a \mapsto_{epi\mathfrak{C}} b$
unfolding *is-monic-arr-def is-epic-arr-def smc-op-simps* **..**

lemma (**in** *semicategory*) *smc-Comp-is-monic-arr*[*smc-arrow-cs-intros*]:
assumes $g : b \mapsto_{mon\mathfrak{C}} c$ **and** $f : a \mapsto_{mon\mathfrak{C}} b$
shows $g \circ_{A\mathfrak{C}} f : a \mapsto_{mon\mathfrak{C}} c$
proof(*intro is-monic-arrI*)
from *assms* **show** $g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$ **by** (*auto intro*: *smc-cs-intros*)
fix $f'\ g'\ a'$
assume f': $f' : a' \mapsto_{\mathfrak{C}} a$
and g': $g' : a' \mapsto_{\mathfrak{C}} a$
and $g \circ_{A\mathfrak{C}} f \circ_{A\mathfrak{C}} f' = g \circ_{A\mathfrak{C}} f \circ_{A\mathfrak{C}} g'$
with *assms* **have** $g \circ_{A\mathfrak{C}} (f \circ_{A\mathfrak{C}} f') = g \circ_{A\mathfrak{C}} (f \circ_{A\mathfrak{C}} g')$
by (*force simp*: *smc-Comp-assoc*)
moreover from *assms* **have** $f \circ_{A\mathfrak{C}} f' : a' \mapsto_{\mathfrak{C}} b\ f \circ_{A\mathfrak{C}} g' : a' \mapsto_{\mathfrak{C}} b$
by (*auto intro*: $f'\ g'$ *smc-cs-intros*)
ultimately have $f \circ_{A\mathfrak{C}} f' = f \circ_{A\mathfrak{C}} g'$ **using** *assms*(1) **by** *clarsimp*
with *assms* $f'\ g'$ **show** $f' = g'$ **by** *clarsimp*
qed

lemmas [*smc-arrow-cs-intros*] = *semicategory.smc-Comp-is-monic-arr*

lemma (in *semicategory*) *smc-Comp-is-epic-arr*[*smc-arrow-cs-intros*]:
assumes $g : b \mapsto_{epi\mathfrak{C}} c$ **and** $f : a \mapsto_{epi\mathfrak{C}} b$
shows $g \circ_{A\mathfrak{C}} f : a \mapsto_{epi\mathfrak{C}} c$
proof–
from *assms op-smc-is-arr* **have** $g : b \mapsto_{\mathfrak{C}} c$ $f : a \mapsto_{\mathfrak{C}} b$
unfolding *is-epic-arr-def* **by** *auto*
with *semicategory-axioms* **have** $f \circ_{A\, op\text{-}smc\ \mathfrak{C}} g = g \circ_{A\mathfrak{C}} f$
by (*cs-concl* **cs-shallow cs-simp**: *smc-op-simps*)
with
semicategory.smc-Comp-is-monic-arr[
OF semicategory-op,
OF assms(2,1)[*unfolded is-epic-arr-def*],
folded is-epic-arr-def
]
show *?thesis*
by *auto*
qed

lemmas [*smc-arrow-cs-intros*] = *semicategory.smc-Comp-is-epic-arr*

lemma (in *semicategory*) *smc-Comp-is-monic-arr-is-monic-arr*:
assumes $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **and** $g \circ_{A\mathfrak{C}} f : a \mapsto_{mon\mathfrak{C}} c$
shows $f : a \mapsto_{mon\mathfrak{C}} b$
proof(*intro is-monic-arrI*)
fix f' g' a'
assume f': $f' : a' \mapsto_{\mathfrak{C}} a$
and g': $g' : a' \mapsto_{\mathfrak{C}} a$
and $f'gg'g$: $f \circ_{A\mathfrak{C}} f' = f \circ_{A\mathfrak{C}} g'$
from *assms*(1,2) f' g' **have** $(g \circ_{A\mathfrak{C}} f) \circ_{A\mathfrak{C}} f' = (g \circ_{A\mathfrak{C}} f) \circ_{A\mathfrak{C}} g'$
by (*auto simp*: *smc-Comp-assoc* $f'gg'g$)
with *assms*(3) f' g' **show** $f' = g'$ **by** *clarsimp*
qed (*simp add*: *assms*(2))

lemma (in *semicategory*) *smc-Comp-is-epic-arr-is-epic-arr*:
assumes $g : a \mapsto_{\mathfrak{C}} b$ **and** $f : b \mapsto_{\mathfrak{C}} c$ **and** $f \circ_{A\mathfrak{C}} g : a \mapsto_{epi\mathfrak{C}} c$
shows $f : b \mapsto_{epi\mathfrak{C}} c$
proof–
from *assms* **have** $g : b \mapsto_{op\text{-}smc\ \mathfrak{C}} a$ $f : c \mapsto_{op\text{-}smc\ \mathfrak{C}} b$
unfolding *smc-op-simps* **by** *simp-all*
moreover from *semicategory-axioms assms* **have** $g \circ_{A\, op\text{-}smc\ \mathfrak{C}} f : a \mapsto_{epi\mathfrak{C}} c$
by (*cs-concl* **cs-shallow cs-simp**: *smc-op-simps*)
ultimately show *?thesis*
using
semicategory.smc-Comp-is-monic-arr-is-monic-arr[
OF semicategory-op, *folded is-epic-arr-def*
]
by *auto*
qed

4.2.6 Idempotent arrow

See Chapter I-5 in [39].

definition *is-idem-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
where *is-idem-arr* $\mathfrak{C}$ b $f \longleftrightarrow f : b \mapsto_{\mathfrak{C}} b \wedge f \circ_{A\mathfrak{C}} f = f$

syntax *-is-idem-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow$ *bool* ($\langle$- : $\mapsto_{ide1}$ -$\rangle$ [51, 51] 51)
syntax-consts *-is-idem-arr* $\rightleftharpoons$ *is-idem-arr*

translations $f : \mapsto_{ide_{\mathfrak{C}}} b \rightleftharpoons CONST\ is\text{-}idem\text{-}arr\ \mathfrak{C}\ b\ f$

Rules.

mk-ide rf *is-idem-arr-def*
|*intro is-idem-arrI*|
|*dest is-idem-arrD*[*dest*]|
|*elim is-idem-arrE*[*elim*!]|

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

Elementary properties.

lemma (**in** *semicategory*) *op-smc-is-idem-arr*[*smc-op-simps*]:
$f : \mapsto_{ide_{op\text{-}smc\ \mathfrak{C}}} b \longleftrightarrow f : \mapsto_{ide_{\mathfrak{C}}} b$
using *op-smc-Comp* **unfolding** *is-idem-arr-def smc-op-simps* **by** *auto*

4.2.7 Terminal object and initial object

See Chapter I-5 in [39].

definition *obj-terminal* :: $V \Rightarrow V \Rightarrow bool$
where $obj\text{-}terminal\ \mathfrak{C}\ t \longleftrightarrow$
$t \in_{\circ} \mathfrak{C}(\!|Obj|\!) \wedge (\forall a.\ a \in_{\circ} \mathfrak{C}(\!|Obj|\!) \longrightarrow (\exists!f.\ f : a \mapsto_{\mathfrak{C}} t))$

definition *obj-initial* :: $V \Rightarrow V \Rightarrow bool$
where $obj\text{-}initial\ \mathfrak{C} \equiv obj\text{-}terminal\ (op\text{-}smc\ \mathfrak{C})$

Rules.

mk-ide rf *obj-terminal-def*
|*intro obj-terminalI*|
|*dest obj-terminalD*[*dest*]|
|*elim obj-terminalE*[*elim*]|

lemma *obj-initialI*:
assumes $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $\bigwedge b.\ b \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \exists!f.\ f : a \mapsto_{\mathfrak{C}} b$
shows *obj-initial* $\mathfrak{C}\ a$
unfolding *obj-initial-def*
by (*simp add*: *obj-terminalI*[*of* - ‹*op-smc* $\mathfrak{C}$›, *unfolded smc-op-simps*, *OF assms*])

lemma *obj-initialD*[*dest*]:
assumes *obj-initial* $\mathfrak{C}\ a$
shows $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $\bigwedge b.\ b \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \exists!f.\ f : a \mapsto_{\mathfrak{C}} b$
by
(
simp-all add:
obj-terminalD[*OF assms*[*unfolded obj-initial-def*], *unfolded smc-op-simps*]
)

lemma *obj-initialE*[*elim*]:
assumes *obj-initial* $\mathfrak{C}\ a$
obtains $a \in_{\circ} \mathfrak{C}(\!|Obj|\!)$ **and** $\bigwedge b.\ b \in_{\circ} \mathfrak{C}(\!|Obj|\!) \Longrightarrow \exists!f.\ f : a \mapsto_{\mathfrak{C}} b$
using *assms* **by** (*auto simp*: *obj-initialD*)

Elementary properties.

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

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

4.2.8 Null object

See Chapter I-5 in [39].

definition *obj-null* :: $V \Rightarrow V \Rightarrow$ *bool*
where *obj-null* $\mathfrak{C}$ $a \longleftrightarrow$ *obj-initial* $\mathfrak{C}$ $a \wedge$ *obj-terminal* $\mathfrak{C}$ a

Rules.

mk-ide rf *obj-null-def*
|*intro obj-nullI*|
|*dest obj-nullD*[*dest*]|
|*elim obj-nullE*[*elim*]|

Elementary properties.

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

4.2.9 Zero arrow

definition *is-zero-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
where *is-zero-arr* $\mathfrak{C}$ a b $h \longleftrightarrow$
$(\exists z\ g\ f.$ *obj-null* $\mathfrak{C}$ $z \wedge h = g \circ_{A\mathfrak{C}} f \wedge f : a \mapsto_{\mathfrak{C}} z \wedge g : z \mapsto_{\mathfrak{C}} b)$

syntax *-is-zero-arr* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
$(\langle$- : - $\mapsto_{01}$ -$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-zero-arr* $\rightleftharpoons$ *is-zero-arr*
translations $h : a \mapsto_{0\mathfrak{C}} b \rightleftharpoons$ *CONST is-zero-arr* $\mathfrak{C}$ a b h

Rules.

lemma *is-zero-arrI*:
assumes *obj-null* $\mathfrak{C}$ z
and $h = g \circ_{A\mathfrak{C}} f$
and $f : a \mapsto_{\mathfrak{C}} z$
and $g : z \mapsto_{\mathfrak{C}} b$
shows $h : a \mapsto_{0\mathfrak{C}} b$
using *assms* **unfolding** *is-zero-arr-def* **by** *auto*

lemma *is-zero-arrD*[*dest*]:
assumes $h : a \mapsto_{0\mathfrak{C}} b$
shows $\exists z\ g\ f.$ *obj-null* $\mathfrak{C}$ $z \wedge h = g \circ_{A\mathfrak{C}} f \wedge f : a \mapsto_{\mathfrak{C}} z \wedge g : z \mapsto_{\mathfrak{C}} b$
using *assms* **unfolding** *is-zero-arr-def* **by** *simp*

lemma *is-zero-arrE*[*elim*]:
assumes $h : a \mapsto_{0\mathfrak{C}} b$
obtains $z\ g\ f$
where *obj-null* $\mathfrak{C}$ z
and $h = g \circ_{A\mathfrak{C}} f$
and $f : a \mapsto_{\mathfrak{C}} z$
and $g : z \mapsto_{\mathfrak{C}} b$
using *assms* **by** *auto*

Elementary properties.

lemma (in *semicategory*) *op-smc-is-zero-arr*[*smc-op-simps*]:

$f : b \mapsto_{0\,op\text{-}smc\ \mathfrak{C}} a \longleftrightarrow f : a \mapsto_{0\mathfrak{C}} b$
using *op-smc-Comp* **unfolding** *is-zero-arr-def smc-op-simps* **by** *metis*

lemma (**in** *semicategory*) *smc-is-zero-arr-Comp-right*:
assumes $h : b \mapsto_{0\mathfrak{C}} c$ **and** $h' : a \mapsto_{\mathfrak{C}} b$
shows $h \circ_{A\mathfrak{C}} h' : a \mapsto_{0\mathfrak{C}} c$
proof–
from *assms*(1) **obtain** $z\ g\ f$
where *obj-null* $\mathfrak{C}\ z$
and $h = g \circ_{A\mathfrak{C}} f$
and $f : b \mapsto_{\mathfrak{C}} z$
and $g : z \mapsto_{\mathfrak{C}} c$
by *auto*
with *assms* **show** *?thesis*
by (*auto simp*: *smc-cs-simps intro*: *is-zero-arrI smc-cs-intros*)
qed

lemmas [*smc-arrow-cs-intros*] = *semicategory.smc-is-zero-arr-Comp-right*

lemma (**in** *semicategory*) *smc-is-zero-arr-Comp-left*:
assumes $h' : b \mapsto_{\mathfrak{C}} c$ **and** $h : a \mapsto_{0\mathfrak{C}} b$
shows $h' \circ_{A\mathfrak{C}} h : a \mapsto_{0\mathfrak{C}} c$
proof–
from *assms*(2) **obtain** $z\ g\ f$
where *obj-null* $\mathfrak{C}\ z$
and $h = g \circ_{A\mathfrak{C}} f$
and $f : a \mapsto_{\mathfrak{C}} z$
and $g : z \mapsto_{\mathfrak{C}} b$
by *auto*
with *assms*(1) **show** *?thesis*
by (*intro is-zero-arrI*[*of* - - - ‹$h' \circ_{A\mathfrak{C}} g$›])
(*auto simp*: *smc-Comp-assoc intro*: *is-zero-arrI smc-cs-intros*)
qed

lemmas [*smc-arrow-cs-intros*] = *semicategory.smc-is-zero-arr-Comp-left*

4.3 Smallness for semicategories

4.3.1 Background

An explanation of the methodology chosen for the exposition of all matters related to the size of the semicategories and associated entities is given in the previous chapter.

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

4.3.2 Tiny semicategory

Definition and elementary properties

locale *tiny-semicategory* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ + *Comp*: *vsv* ‹$\mathfrak{C}(\!|Comp|\!)$› **for** α $\mathfrak{C}$ +
  assumes *tiny-smc-length*[*smc-cs-simps*]: *vcard* $\mathfrak{C} = 5_{\mathbb{N}}$
    and *tiny-smc-tiny-digraph*[*slicing-intros*]: *tiny-digraph* α (*smc-dg* $\mathfrak{C}$)
    and *tiny-smc-Comp-vdomain*: $gf \in_{\circ} \mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!)) \longleftrightarrow$
      $(\exists g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b)$
    and *tiny-smc-Comp-is-arr*[*smc-cs-intros*]:
      $[\![\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
    and *tiny-smc-assoc*[*smc-cs-simps*]:
      $[\![\ h : c \mapsto_{\mathfrak{C}} d;\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
        $(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$

lemmas [*smc-cs-simps*] =
  tiny-semicategory.tiny-smc-length
  tiny-semicategory.tiny-smc-assoc

lemmas [*slicing-intros*] =
  tiny-semicategory.tiny-smc-Comp-is-arr

Rules.

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

mk-ide rf *tiny-semicategory-def*[*unfolded tiny-semicategory-axioms-def*]
  |*intro tiny-semicategoryI*|
  |*dest tiny-semicategoryD*[*dest*]|
  |*elim tiny-semicategoryE*[*elim*]|

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

lemma *tiny-semicategoryI''*:
  assumes $\mathcal{Z}$ α
    and *vfsequence* $\mathfrak{C}$

```
    and vsv (𝔈(|Comp|))
    and vcard 𝔈 = 5_ℕ
    and vsv (𝔈(|Dom|))
    and vsv (𝔈(|Cod|))
    and 𝒟∘ (𝔈(|Dom|)) = 𝔈(|Arr|)
    and ℛ∘ (𝔈(|Dom|)) ⊆∘ 𝔈(|Obj|)
    and 𝒟∘ (𝔈(|Cod|)) = 𝔈(|Arr|)
    and ℛ∘ (𝔈(|Cod|)) ⊆∘ 𝔈(|Obj|)
    and ⋀gf. gf ∈∘ 𝒟∘ (𝔈(|Comp|)) ⟷
      (∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦_𝔈 c ∧ f : a ↦_𝔈 b)
    and ⋀b c g a f. [[ g : b ↦_𝔈 c; f : a ↦_𝔈 b ]] ⟹ g ∘_A𝔈 f : a ↦_𝔈 c
    and ⋀c d h b g a f. [[ h : c ↦_𝔈 d; g : b ↦_𝔈 c; f : a ↦_𝔈 b ]] ⟹
        (h ∘_A𝔈 g) ∘_A𝔈 f = h ∘_A𝔈 (g ∘_A𝔈 f)
    and 𝔈(|Obj|) ∈∘ Vset α
    and 𝔈(|Arr|) ∈∘ Vset α
  shows tiny-semicategory α 𝔈
  by (intro tiny-semicategoryI tiny-digraphI, unfold slicing-simps)
    (simp-all add: smc-dg-def nat-omega-simps assms)
```

Slicing.

```
context tiny-semicategory
begin

interpretation dg: tiny-digraph α ‹smc-dg 𝔈› by (rule tiny-smc-tiny-digraph)

lemmas-with [unfolded slicing-simps]:
  tiny-smc-Obj-in-Vset[smc-small-cs-intros] = dg.tiny-dg-Obj-in-Vset
  and tiny-smc-Arr-in-Vset[smc-small-cs-intros] = dg.tiny-dg-Arr-in-Vset
  and tiny-smc-Dom-in-Vset[smc-small-cs-intros] = dg.tiny-dg-Dom-in-Vset
  and tiny-smc-Cod-in-Vset[smc-small-cs-intros] = dg.tiny-dg-Cod-in-Vset

end
```

Elementary properties.

```
sublocale tiny-semicategory ⊆ semicategory
  by (rule semicategoryI)
    (
      auto
        simp:
          vfsequence-axioms
          tiny-digraph.tiny-dg-digraph
          tiny-smc-tiny-digraph
          tiny-smc-Comp-vdomain
        intro: smc-cs-intros smc-cs-simps
    )

lemmas (in tiny-semicategory) tiny-smc-semicategory = semicategory-axioms

lemmas [smc-small-cs-intros] = tiny-semicategory.tiny-smc-semicategory
```

Size.

```
lemma (in tiny-semicategory) tiny-smc-Comp-in-Vset: 𝔈(|Comp|) ∈∘ Vset α
proof–
  have 𝔈(|Arr|) ∈∘ Vset α by (simp add: tiny-smc-Arr-in-Vset)
  with Axiom-of-Infinity have 𝔈(|Arr|) ^× 2_ℕ ∈∘ Vset α
    by (intro Limit-vcpower-in-VsetI) auto
  with Comp.pnop-vdomain have D: 𝒟∘ (𝔈(|Comp|)) ∈∘ Vset α by auto
```

```
  moreover from tiny-smc-Arr-in-Vset smc-Comp-vrange have
    ℛ∘ (𝔈(‖Comp‖)) ∈∘ Vset α
    by auto
  ultimately show ?thesis by (simp add: Comp.vbrelation-Limit-in-VsetI)
qed

lemma (in tiny-semicategory) tiny-smc-in-Vset: 𝔈 ∈∘ Vset α
proof–
  note [smc-cs-intros] =
    tiny-smc-Obj-in-Vset
    tiny-smc-Arr-in-Vset
    tiny-smc-Dom-in-Vset
    tiny-smc-Cod-in-Vset
    tiny-smc-Comp-in-Vset
  show ?thesis
    by (subst smc-def) (cs-concl cs-shallow cs-intro: smc-cs-intros V-cs-intros)
qed

lemma small-tiny-semicategories[simp]: small {𝔈. tiny-semicategory α 𝔈}
proof(rule down)
  show {𝔈. tiny-semicategory α 𝔈} ⊆ elts (set {𝔈. semicategory α 𝔈})
    by (auto intro: smc-small-cs-intros)
qed

lemma tiny-semicategories-vsubset-Vset:
  set {𝔈. tiny-semicategory α 𝔈} ⊆∘ Vset α
  by (rule vsubsetI) (simp add: tiny-semicategory.tiny-smc-in-Vset)

lemma (in semicategory) smc-tiny-semicategory-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows tiny-semicategory β 𝔈
proof(intro tiny-semicategoryI)
  show tiny-digraph β (smc-dg 𝔈)
    by (rule digraph.dg-tiny-digraph-if-ge-Limit, rule smc-digraph; intro assms)
qed
  (
    auto simp:
      assms(1)
      smc-cs-simps
      smc-cs-intros
      smc-digraph digraph.dg-tiny-digraph-if-ge-Limit
      smc-Comp-vdomain vfsequence-axioms
  )
```

Opposite tiny semicategory

```
lemma (in tiny-semicategory) tiny-semicategory-op:
  tiny-semicategory α (op-smc 𝔈)
  by (intro tiny-semicategoryI', unfold smc-op-simps)
    (auto simp: smc-op-intros smc-small-cs-intros)

lemmas tiny-semicategory-op[smc-op-intros] =
  tiny-semicategory.tiny-semicategory-op
```

4.3.3 Finite semicategory

Definition and elementary properties

A finite semicategory is a generalization of the concept of a finite category, as presented in nLab [3] [3].

locale *finite-semicategory* = $\mathcal{Z}$ α + *vfsequence* $\mathfrak{C}$ + *Comp*: *vsv* ‹$\mathfrak{C}(\!|Comp|\!)$› **for** α $\mathfrak{C}$ +
  assumes *fin-smc-length*[*smc-cs-simps*]: *vcard* $\mathfrak{C} = 5_{\mathbb{N}}$
    and *fin-smc-finite-digraph*[*slicing-intros*]: *finite-digraph* α (*smc-dg* $\mathfrak{C}$)
    and *fin-smc-Comp-vdomain*: $gf \in_{\circ} \mathcal{D}_{\circ}(\mathfrak{C}(\!|Comp|\!)) \longleftrightarrow$
      $(\exists g\ f\ b\ c\ a.\ gf = [g, f]_{\circ} \wedge g : b \mapsto_{\mathfrak{C}} c \wedge f : a \mapsto_{\mathfrak{C}} b)$
    and *fin-smc-Comp-is-arr*[*smc-cs-intros*]:
      $[\![\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{C}} f : a \mapsto_{\mathfrak{C}} c$
    and *fin-smc-assoc*[*smc-cs-simps*]:
      $[\![\ h : c \mapsto_{\mathfrak{C}} d;\ g : b \mapsto_{\mathfrak{C}} c;\ f : a \mapsto_{\mathfrak{C}} b\ ]\!] \Longrightarrow$
        $(h \circ_{A\mathfrak{C}} g) \circ_{A\mathfrak{C}} f = h \circ_{A\mathfrak{C}} (g \circ_{A\mathfrak{C}} f)$

lemmas [*smc-cs-simps*] =
  finite-semicategory.fin-smc-length
  finite-semicategory.fin-smc-assoc

lemmas [*slicing-intros*] =
  finite-semicategory.fin-smc-Comp-is-arr

Rules.

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

mk-ide rf *finite-semicategory-def*[*unfolded finite-semicategory-axioms-def*]
  |*intro finite-semicategoryI*|
  |*dest finite-semicategoryD*[*dest*]|
  |*elim finite-semicategoryE*[*elim*]|

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

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

Slicing.

context *finite-semicategory*

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

begin

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

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

end

Elementary properties.

sublocale *finite-semicategory* ⊆ *tiny-semicategory*
by (*rule tiny-semicategoryI*)
(
auto simp:
vfsequence-axioms
fin-smc-Comp-vdomain
fin-smc-finite-digraph
finite-digraph.fin-dg-tiny-digraph
intro: *smc-cs-intros smc-cs-simps*
)

lemmas (**in** *finite-semicategory*) *fin-smc-tiny-semicategory* =
tiny-semicategory-axioms

lemmas [*smc-small-cs-intros*] = *finite-semicategory.fin-smc-tiny-semicategory*

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

Size.

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

lemma *finite-semicategories-vsubset-Vset*:
set $\{\mathfrak{C}.\ finite\text{-}semicategory\ \alpha\ \mathfrak{C}\} \subseteq_{\circ} Vset\ \alpha$
by (*rule vsubsetI*) (*simp add*: *finite-semicategory.fin-smc-in-Vset*)

Opposite finite semicategory

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

lemmas *finite-semicategory-op*[*smc-op-intros*] =
finite-semicategory.finite-semicategory-op

4.4 Semifunctor

4.4.1 Background

named-theorems *smcf-cs-simps*
named-theorems *smcf-cs-intros*

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

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

Slicing

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

Components.

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

4.4.2 Definition and elementary properties

See Chapter I-3 in [39] and the description of the concept of a digraph homomorphism in the previous chapter.

locale *is-semifunctor* =
  $\mathcal{Z}$ α +
  vfsequence $\mathfrak{F}$ +
  HomDom: *semicategory* α $\mathfrak{A}$ +
  HomCod: *semicategory* α $\mathfrak{B}$
  for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *smcf-length*[*smc-cs-simps*]: *vcard* $\mathfrak{F}$ = $4_{\mathbb{N}}$
    and *smcf-is-dghm*[*slicing-intros*]:
      smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{B}$
    and *smcf-HomDom*[*smc-cs-simps*]: $\mathfrak{F}(\!|HomDom|\!)$ = $\mathfrak{A}$
    and *smcf-HomCod*[*smc-cs-simps*]: $\mathfrak{F}(\!|HomCod|\!)$ = $\mathfrak{B}$
    and *smcf-ArrMap-Comp*[*smc-cs-simps*]: $[\![\ g : b \mapsto_{\mathfrak{A}} c;\ f : a \mapsto_{\mathfrak{A}} b\ ]\!] \Longrightarrow$
      $\mathfrak{F}(\!|ArrMap|\!)(\!|g \circ_{A\mathfrak{A}} f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$

syntax *-is-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  $(\langle(\text{-} :/ \text{-} \mapsto\!\!\mapsto_{SMC^1} \text{-})\rangle\ [51, 51, 51]\ 51)$
syntax-consts *-is-semifunctor* $\rightleftharpoons$ *is-semifunctor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation (*input*) *is-cn-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow bool$
  where *is-cn-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\equiv \mathfrak{F} :$ *op-smc* $\mathfrak{A} \mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{B}$

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

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

abbreviation *smcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *smcfs* $\alpha\ \mathfrak{A}\ \mathfrak{B} \equiv$ *set* $\{\mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}\}$

lemmas [*smc-cs-simps*] =
is-semifunctor.smcf-HomDom
is-semifunctor.smcf-HomCod
is-semifunctor.smcf-ArrMap-Comp

lemma *smcf-is-dghm'*[*slicing-intros*]:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{A}' =$ *smc-dg* $\mathfrak{A}$
and $\mathfrak{B}' =$ *smc-dg* $\mathfrak{B}$
shows *smcf-dghm* $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{DG\alpha} \mathfrak{B}'$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule is-semifunctor.smcf-is-dghm*)

lemma *cn-dghm-comp-is-dghm*:
assumes $\mathfrak{F}$: *op-smc* $\mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows *smcf-dghm* $\mathfrak{F}$: *op-dg* (*smc-dg* $\mathfrak{A}$) $\mapsto\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{B}$
using *assms*
unfolding *slicing-simps slicing-commute*
by (*cs-concl* **cs-shallow cs-intro**: *slicing-intros*)

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

Rules.

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

mk-ide rf *is-semifunctor-def*[*unfolded is-semifunctor-axioms-def*]
|*intro is-semifunctorI*|
|*dest is-semifunctorD*[*dest*]|
|*elim is-semifunctorE*[*elim*]|

lemmas [*smc-cs-intros*] =
is-semifunctorD(3,4)

lemma *is-semifunctorI'*:
assumes $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{F}$
and *semicategory* $\alpha\ \mathfrak{A}$
and *semicategory* $\alpha\ \mathfrak{B}$
and *vcard* $\mathfrak{F} = 4_{\mathbb{N}}$
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 $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{A}} c;\ f : a \mapsto_{\mathfrak{A}} b\ ]\!] \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)(\!|g \circ_{A\mathfrak{A}} f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
by (*intro is-semifunctorI is-dghmI*, *unfold smcf-dghm-components slicing-simps*)
(*simp-all add*: *assms smcf-dghm-def nat-omega-simps semicategory.smc-digraph*)

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

lemma *is-semifunctorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
obtains $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{F}$
and *semicategory* $\alpha\ \mathfrak{A}$
and *semicategory* $\alpha\ \mathfrak{B}$
and *vcard* $\mathfrak{F} = 4_{\mathbb{N}}$
and $\mathfrak{F}(\!|HomDom|\!) = \mathfrak{A}$
and $\mathfrak{F}(\!|HomCod|\!) = \mathfrak{B}$
and *vsv* $(\mathfrak{F}(\!|ObjMap|\!))$
and *vsv* $(\mathfrak{F}(\!|ArrMap|\!))$
and $\mathcal{D}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!))$ = $\mathfrak{A}(\!|Obj|\!)$
and $\mathcal{R}_{\circ}\ (\mathfrak{F}(\!|ObjMap|\!)) \subseteq_{\circ} \mathfrak{B}(\!|Obj|\!)$
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 $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{A}} c;\ f : a \mapsto_{\mathfrak{A}} b\ ]\!] \Longrightarrow$
$\mathfrak{F}(\!|ArrMap|\!)(\!|g \circ_{A\mathfrak{A}} f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
using *assms* **by** (*simp add*: *is-semifunctorD'*)

Slicing.

context *is-semifunctor*
begin

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

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

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

end

lemmas [*smc-cs-simps*] =
is-semifunctor.smcf-ObjMap-vdomain
is-semifunctor.smcf-ArrMap-vdomain

lemmas [*smc-cs-intros*] =
is-semifunctor.smcf-ObjMap-app-in-HomCod-Obj
is-semifunctor.smcf-ArrMap-app-in-HomCod-Arr
is-semifunctor.smcf-ArrMap-is-arr'

Elementary properties.

lemma *cn-smcf-ArrMap-Comp*[*smc-cs-simps*]:
assumes *semicategory* α $\mathfrak{A}$
and $\mathfrak{F}$: *op-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{B}$
and g : c $\mapsto_{\mathfrak{A}}$ b
and f : b $\mapsto_{\mathfrak{A}}$ a
shows $\mathfrak{F}(\!|ArrMap|\!)(\!|f \circ_{A\mathfrak{A}} g|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)$
proof–
from *assms*(3,4) **have** *gf*:
$\mathfrak{F}(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) = \mathfrak{F}(\!|ArrMap|\!)(\!|g \circ_{A\,op\text{-}smc\ \mathfrak{A}} f|\!)$
by
(
force
intro: *is-semifunctor.smcf-ArrMap-Comp*[*OF assms*(2), *symmetric*]
simp: *slicing-simps smc-op-simps*
)

```
  from assms show ?thesis
    unfolding gf by (cs-concl cs-shallow cs-simp: smc-op-simps)
qed

lemma smcf-eqI:
  assumes 𝔊 : 𝔄 ↦↦_{SMC α} 𝔅
    and 𝔉 : ℭ ↦↦_{SMC α} 𝔇
    and 𝔊(|ObjMap|) = 𝔉(|ObjMap|)
    and 𝔊(|ArrMap|) = 𝔉(|ArrMap|)
    and 𝔄 = ℭ
    and 𝔅 = 𝔇
  shows 𝔊 = 𝔉
proof–
  interpret L: is-semifunctor α 𝔄 𝔅 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α ℭ 𝔇 𝔉 by (rule assms(2))
  show ?thesis
  proof(rule vsv-eqI)
    have dom: 𝒟∘ 𝔊 = 4_ℕ
      by (cs-concl cs-shallow cs-simp: smc-cs-simps V-cs-simps)
    show 𝒟∘ 𝔊 = 𝒟∘ 𝔉
      by (cs-concl cs-shallow cs-simp: smc-cs-simps V-cs-simps)
    from assms(5,6) have sup: 𝔊(|HomDom|) = 𝔉(|HomDom|) 𝔊(|HomCod|) = 𝔉(|HomCod|)
      by (simp-all add: smc-cs-simps)
    show a ∈∘ 𝒟∘ 𝔊 ⟹ 𝔊(|a|) = 𝔉(|a|) for a
      by (unfold dom, elim-in-numeral, insert assms(3,4) sup)
        (auto simp: dghm-field-simps)
  qed auto
qed

lemma smcf-dghm-eqI:
  assumes 𝔊 : 𝔄 ↦↦_{SMC α} 𝔅
    and 𝔉 : ℭ ↦↦_{SMC α} 𝔇
    and 𝔄 = ℭ
    and 𝔅 = 𝔇
    and smcf-dghm 𝔊 = smcf-dghm 𝔉
  shows 𝔊 = 𝔉
proof(rule smcf-eqI)
  from assms(5) have
    smcf-dghm 𝔊(|ObjMap|) = smcf-dghm 𝔉(|ObjMap|)
    smcf-dghm 𝔊(|ArrMap|) = smcf-dghm 𝔉(|ArrMap|)
    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)

lemma (in is-semifunctor) smcf-def:
  𝔉 = [𝔉(|ObjMap|), 𝔉(|ArrMap|), 𝔉(|HomDom|), 𝔉(|HomCod|)]∘
proof(rule vsv-eqI)
  have dom-lhs: 𝒟∘ 𝔉 = 4_ℕ
    by (cs-concl cs-shallow cs-simp: smc-cs-simps V-cs-simps)
  have dom-rhs: 𝒟∘ [𝔉(|Obj|), 𝔉(|Arr|), 𝔉(|Dom|), 𝔉(|Cod|)]∘ = 4_ℕ
    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)

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

lemma (in is-semifunctor) smcf-is-semifunctor-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔉 : 𝔄 ↦↦_{SMC β} 𝔅
  by (rule is-semifunctorI)
    (
      simp-all add:
        assms
        vfsequence-axioms
        smcf-is-dghm-if-ge-Limit
        HomDom.smc-semicategory-if-ge-Limit
        HomCod.smc-semicategory-if-ge-Limit
        smc-cs-simps
    )

lemma small-all-smcfs[simp]: small {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-semifunctor.smcf-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 ↦↦_{SMC α} 𝔅} = {} by auto
  then show ?thesis by simp
qed

lemma (in is-semifunctor) smcf-in-Vset-7: 𝔉 ∈∘ Vset (α + 7_ℕ)
proof–
  note [folded VPow-iff, folded Vset-succ[OF Ord-α], smc-cs-intros] =
    smcf-ObjMap-vsubset-Vset
    smcf-ArrMap-vsubset-Vset
  from HomDom.smc-semicategory-in-Vset-4 have [smc-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.smc-semicategory-in-Vset-4 have [smc-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 smcf-def, succ-of-numeral)
      (
        cs-concl
          cs-simp: plus-V-succ-right V-cs-simps smc-cs-simps
          cs-intro: smc-cs-intros V-cs-intros
      )
qed

lemma (in 𝒵) all-smcfs-in-Vset:
  assumes 𝒵 β and α ∈∘ β
  shows all-smcfs α ∈∘ Vset β
proof(rule vsubset-in-VsetI)
  interpret β: 𝒵 β by (rule assms(1))
  show all-smcfs α ⊆∘ Vset (α + 7ℕ)
  proof(intro vsubsetI)
    fix 𝔉 assume 𝔉 ∈∘ all-smcfs α
    then obtain 𝔄 𝔅 where 𝔉: 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅 by clarsimp
    then interpret is-semifunctor α 𝔄 𝔅 𝔉 .
    show 𝔉 ∈∘ Vset (α + 7ℕ) by (rule smcf-in-Vset-7)
  qed
  from assms(2) show Vset (α + 7ℕ) ∈∘ Vset β
    by (cs-concl cs-shallow cs-intro: V-cs-intros Ord-cs-intros)
qed

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

4.4.3 Opposite semifunctor

Definition and elementary properties

See Chapter II-2 in [39].

```
definition op-smcf :: V ⇒ V
  where op-smcf 𝔉 =
    [𝔉(|ObjMap|), 𝔉(|ArrMap|), op-smc (𝔉(|HomDom|)), op-smc (𝔉(|HomCod|))]∘
```

Components.

```
lemma op-smcf-components[smc-op-simps]:
  shows op-smcf 𝔉(|ObjMap|) = 𝔉(|ObjMap|)
    and op-smcf 𝔉(|ArrMap|) = 𝔉(|ArrMap|)
    and op-smcf 𝔉(|HomDom|) = op-smc (𝔉(|HomDom|))
    and op-smcf 𝔉(|HomCod|) = op-smc (𝔉(|HomCod|))
  unfolding op-smcf-def dghm-field-simps by (auto simp: nat-omega-simps)
```

Slicing.

```
lemma op-dghm-smcf-dghm[slicing-commute]:
  op-dghm (smcf-dghm 𝔉) = smcf-dghm (op-smcf 𝔉)
proof(rule vsv-eqI)
  have dom-lhs: 𝒟∘ (op-dghm (smcf-dghm 𝔉)) = 4ℕ
    unfolding op-dghm-def by (auto simp: nat-omega-simps)
  have dom-rhs: 𝒟∘ (smcf-dghm (op-smcf 𝔉)) = 4ℕ
    unfolding smcf-dghm-def by (auto simp: nat-omega-simps)
  show 𝒟∘ (op-dghm (smcf-dghm 𝔉)) = 𝒟∘ (smcf-dghm (op-smcf 𝔉))
    unfolding dom-lhs dom-rhs by simp
  show a ∈∘ 𝒟∘ (op-dghm (smcf-dghm 𝔉)) ⟹
```

```
    op-dghm (smcf-dghm 𝔉)(|a|) = smcf-dghm (op-smcf 𝔉)(|a|)
    for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold smcf-dghm-def op-smcf-def op-dghm-def dghm-field-simps
      )
      (auto simp: nat-omega-simps slicing-simps slicing-commute)
qed (auto simp: smcf-dghm-def op-dghm-def)
```

Further properties

```
lemma (in is-semifunctor) is-semifunctor-op:
  op-smcf 𝔉 : op-smc 𝔄 ↦↦SMC α op-smc 𝔅
proof(intro is-semifunctorI)
  show vfsequence (op-smcf 𝔉) unfolding op-smcf-def by simp
  show vcard (op-smcf 𝔉) = 4ℕ
    unfolding op-smcf-def by (auto simp: nat-omega-simps)
  fix g b c f a assume g : b ↦op-smc 𝔄 c f : a ↦op-smc 𝔄 b
  then have g : c ↦𝔄 b and f : b ↦𝔄 a
    unfolding smc-op-simps by simp-all
  with is-semifunctor-axioms show
    op-smcf 𝔉(|ArrMap|)(|g ∘A op-smc 𝔄 f|) =
      op-smcf 𝔉(|ArrMap|)(|g|) ∘A op-smc 𝔅 op-smcf 𝔉(|ArrMap|)(|f|)
    by
      (
        cs-concl
          cs-simp: smc-op-simps smc-cs-simps
          cs-intro: smc-op-intros smc-cs-intros
      )
qed
  (
    auto simp:
      smc-cs-simps
      smc-op-simps
      slicing-simps
      slicing-commute[symmetric]
      is-dghm.is-dghm-op
      smcf-is-dghm
      HomCod.semicategory-op
      HomDom.semicategory-op
  )

lemma (in is-semifunctor) is-semifunctor-op′:
  assumes 𝔄′ = op-smc 𝔄 and 𝔅′ = op-smc 𝔅 and α′ = α
  shows op-smcf 𝔉 : 𝔄′ ↦↦SMC α′ 𝔅′
  unfolding assms by (rule is-semifunctor-op)

lemmas is-semifunctor-op′[smc-op-intros] = is-semifunctor.is-semifunctor-op′

lemma (in is-semifunctor) smcf-op-smcf-op-smcf[smc-op-simps]:
  op-smcf (op-smcf 𝔉) = 𝔉
proof(rule smcf-eqI, unfold smc-op-simps)
  show op-smcf (op-smcf 𝔉) : 𝔄 ↦↦SMC α 𝔅
    by
      (
        metis
```

```
    HomCod.smc-op-smc-op-smc
    HomDom.smc-op-smc-op-smc
    is-semifunctor.is-semifunctor-op
    is-semifunctor-op
  )
qed (simp-all add: is-semifunctor-axioms)

lemmas smcf-op-smcf-op-smcf[smc-op-simps] = is-semifunctor.smcf-op-smcf-op-smcf

lemma eq-op-smcf-iff[smc-op-simps]:
  assumes 𝔊 : 𝔄 ↦↦_{SMCα} 𝔅 and 𝔉 : ℭ ↦↦_{SMCα} 𝔇
  shows op-smcf 𝔊 = op-smcf 𝔉 ⟷ 𝔊 = 𝔉
proof
  interpret L: is-semifunctor α 𝔄 𝔅 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α ℭ 𝔇 𝔉 by (rule assms(2))
  assume prems: op-smcf 𝔊 = op-smcf 𝔉
  show 𝔊 = 𝔉
  proof(rule smcf-eqI[OF assms])
    from prems R.smcf-op-smcf-op-smcf L.smcf-op-smcf-op-smcf show
      𝔊(|ObjMap|) = 𝔉(|ObjMap|) and 𝔊(|ArrMap|) = 𝔉(|ArrMap|)
      by metis+
    from prems R.smcf-op-smcf-op-smcf L.smcf-op-smcf-op-smcf have
      𝔊(|HomDom|) = 𝔉(|HomDom|) 𝔊(|HomCod|) = 𝔉(|HomCod|)
      by auto
    then show 𝔄 = ℭ 𝔅 = 𝔇 by (simp-all add: smc-cs-simps)
  qed
qed auto
```

4.4.4 Composition of covariant semifunctors

Definition and elementary properties

```
abbreviation (input) smcf-comp :: V ⇒ V ⇒ V (infixl ‹∘_{SMCF}› 55)
  where smcf-comp ≡ dghm-comp
```

Slicing.

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

Object map

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

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

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

lemma *smcf-comp-ObjMap-app*[*smc-cs-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and [*simp*]: $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
shows $(\mathfrak{G} \circ_{SMCF} \mathfrak{F})(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) = \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)$
proof–
interpret *L*: *is-semifunctor* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{G}$ **by** (*rule assms*(1))
interpret *R*: *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*(2))
show *?thesis*
by
(
rule dghm-comp-ObjMap-app
[
OF L.smcf-is-dghm R.smcf-is-dghm,
unfolded slicing-simps slicing-commute,
OF assms(3)
]
)
qed

Arrow map

lemma *smcf-comp-ArrMap-vsv*[*smc-cs-intros*]:

```
  assumes 𝔊 : 𝔅 ↦↦_{SMCα} ℭ and 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
  shows vsv ((𝔊 ∘_{SMCF} 𝔉)(|ArrMap|))
proof–
  interpret L: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule dghm-comp-ArrMap-vsv
          [
            OF L.smcf-is-dghm R.smcf-is-dghm,
            unfolded slicing-simps slicing-commute
          ]
      )
qed

lemma smcf-comp-ArrMap-vdomain[smc-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦_{SMCα} ℭ and 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
  shows 𝒟∘ ((𝔊 ∘_{SMCF} 𝔉)(|ArrMap|)) = 𝔄(|Arr|)
proof–
  interpret L: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule dghm-comp-ArrMap-vdomain
          [
            OF L.smcf-is-dghm R.smcf-is-dghm,
            unfolded slicing-simps slicing-commute
          ]
      )
qed

lemma smcf-comp-ArrMap-vrange:
  assumes 𝔊 : 𝔅 ↦↦_{SMCα} ℭ and 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
  shows ℛ∘ ((𝔊 ∘_{SMCF} 𝔉)(|ArrMap|)) ⊆∘ ℭ(|Arr|)
proof–
  interpret L: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
        rule dghm-comp-ArrMap-vrange
          [
            OF L.smcf-is-dghm R.smcf-is-dghm,
            unfolded slicing-simps slicing-commute
          ]
      )
qed

lemma smcf-comp-ArrMap-app[smc-cs-simps]:
  assumes 𝔊 : 𝔅 ↦↦_{SMCα} ℭ
    and 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
    and [simp]: f ∈∘ 𝔄(|Arr|)
  shows (𝔊 ∘_{SMCF} 𝔉)(|ArrMap|)(|f|) = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|f|)|)
proof–
  interpret L: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
```

```
  show ?thesis
    by
      (
        rule dghm-comp-ArrMap-app
          [
            OF L.smcf-is-dghm R.smcf-is-dghm,
            unfolded slicing-simps slicing-commute,
            OF assms(3)
          ]
      )
qed
```

Further properties

```
lemma smcf-comp-is-semifunctor[smc-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦_{SMCα} ℭ and 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
  shows 𝔊 ∘_{SMCF} 𝔉 : 𝔄 ↦↦_{SMCα} ℭ
proof–
  interpret L: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
  proof(rule is-semifunctorI, unfold dghm-comp-components(3,4))
    show vfsequence (𝔊 ∘_{SMCF} 𝔉) by (simp add: dghm-comp-def)
    show vcard (𝔊 ∘_{SMCF} 𝔉) = 4_ℕ
      unfolding dghm-comp-def by (simp add: nat-omega-simps)
    fix g b c f a assume g : b ↦_𝔄 c f : a ↦_𝔄 b
    with assms show (𝔊 ∘_{SMCF} 𝔉)(|ArrMap|)(|g ∘_{A𝔄} f|) =
      (𝔊 ∘_{SMCF} 𝔉)(|ArrMap|)(|g|) ∘_{Aℭ} (𝔊 ∘_{SMCF} 𝔉)(|ArrMap|)(|f|)
      by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  qed
    (
      auto
        simp: slicing-commute[symmetric] smc-cs-simps smc-cs-intros
        intro: dg-cs-intros slicing-intros
    )
qed

lemma smcf-comp-assoc[smc-cs-simps]:
  assumes ℌ : ℭ ↦↦_{SMCα} 𝔇
    and 𝔊 : 𝔅 ↦↦_{SMCα} ℭ
    and 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
  shows (ℌ ∘_{SMCF} 𝔊) ∘_{SMCF} 𝔉 = ℌ ∘_{SMCF} (𝔊 ∘_{SMCF} 𝔉)
proof(rule smcf-eqI[of α 𝔄 𝔇 - 𝔄 𝔇])
  interpret ℌ: is-semifunctor α ℭ 𝔇 ℌ by (rule assms(1))
  interpret 𝔊: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(2))
  interpret 𝔉: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(3))
  from 𝔉.is-semifunctor-axioms 𝔊.is-semifunctor-axioms ℌ.is-semifunctor-axioms
  show ℌ ∘_{SMCF} (𝔊 ∘_{SMCF} 𝔉) : 𝔄 ↦↦_{SMCα} 𝔇
    and ℌ ∘_{SMCF} 𝔊 ∘_{SMCF} 𝔉 : 𝔄 ↦↦_{SMCα} 𝔇
    by (auto intro: smc-cs-intros)
qed (simp-all add: dghm-comp-components vcomp-assoc)

lemma op-smcf-smcf-comp[smc-op-simps]:
  op-smcf (𝔊 ∘_{SMCF} 𝔉) = op-smcf 𝔊 ∘_{SMCF} op-smcf 𝔉
  unfolding dghm-comp-def op-smcf-def dghm-field-simps
  by (simp add: nat-omega-simps)
```

4.4.5 Composition of contravariant semifunctors

Definition and elementary properties

See section 1.2 in [15].

definition *smcf-cn-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** ‹$_{SMCF}\circ$› 55)
  where $\mathfrak{G}$ $_{SMCF}\circ$ $\mathfrak{F}$ =
    [
      $\mathfrak{G}(\lfloor ObjMap \rfloor) \circ_{\circ} \mathfrak{F}(\lfloor ObjMap \rfloor)$,
      $\mathfrak{G}(\lfloor ArrMap \rfloor) \circ_{\circ} \mathfrak{F}(\lfloor ArrMap \rfloor)$,
      op-smc ($\mathfrak{F}(\lfloor HomDom \rfloor)$),
      $\mathfrak{G}(\lfloor HomCod \rfloor)$
    ]$_{\circ}$

Components.

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

Slicing.

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

Object map: two contravariant semifunctors

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

lemma *smcf-cn-comp-ObjMap-vdomain*[*smc-cn-cs-simps*]:
  assumes $\mathfrak{G} : \mathfrak{B}$ $_{SMC}\mapsto\mapsto_{\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A}$ $_{SMC}\mapsto\mapsto_{\alpha}$ $\mathfrak{B}$
  shows $\mathcal{D}_{\circ}$ (($\mathfrak{G}$ $_{SMCF}\circ$ $\mathfrak{F}$)($\lfloor ObjMap \rfloor$)) = $\mathfrak{A}(\lfloor Obj \rfloor)$
proof–
  interpret *L*: *is-semifunctor* α ‹*op-smc* $\mathfrak{B}$› $\mathfrak{C}$ $\mathfrak{G}$ **by** (*rule assms*(1))
  interpret *R*: *is-semifunctor* α ‹*op-smc* $\mathfrak{A}$› $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
  show *?thesis*
    by
      (

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

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

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

Arrow map: two contravariant semifunctors

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

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

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

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

lemma smcf-cn-comp-ArrMap-app[smc-cn-cs-simps]:
  assumes 𝔊 : 𝔅 SMC↦↦α ℭ and 𝔉 : 𝔄 SMC↦↦α 𝔅 and a ∈∘ 𝔄(|Arr|)
  shows (𝔊 SMCF∘ 𝔉)(|ArrMap|)(|a|) = 𝔊(|ArrMap|)(|𝔉(|ArrMap|)(|a|)|)
proof–
  interpret L: is-semifunctor α ‹op-smc 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α ‹op-smc 𝔄› 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
```

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

Object map: contravariant and covariant semifunctors

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

lemma smcf-cn-cov-comp-ObjMap-vdomain[smc-cn-cs-simps]:
  assumes 𝔊 : 𝔅 SMC↦↦α ℭ and 𝔉 : 𝔄 ↦↦SMCα 𝔅
  shows 𝒟∘ ((𝔊 SMCF∘ 𝔉)(|ObjMap|)) = 𝔄(|Obj|)
proof–
  interpret L: is-semifunctor α ‹op-smc 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
    by
      (
      rule dghm-cn-cov-comp-ObjMap-vdomain
        [
          OF
            L.smcf-is-dghm[unfolded slicing-commute[symmetric]]
            R.smcf-is-dghm,
          unfolded slicing-commute slicing-simps
        ]
      )
qed

lemma smcf-cn-cov-comp-ObjMap-vrange:
  assumes 𝔊 : 𝔅 SMC↦↦α ℭ and 𝔉 : 𝔄 ↦↦SMCα 𝔅
  shows ℛ∘ ((𝔊 SMCF∘ 𝔉)(|ObjMap|)) ⊆∘ ℭ(|Obj|)
proof–
  interpret L: is-semifunctor α ‹op-smc 𝔅› ℭ 𝔊 by (rule assms(1))
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
```

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

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

Arrow map: contravariant and covariant semifunctors

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

lemma smcf-cn-cov-comp-ArrMap-vdomain[smc-cn-cs-simps]:
  assumes 𝔊 : 𝔅 SMC↦↦α ℭ and 𝔉 : 𝔄 ↦↦SMCα 𝔅
  shows 𝒟∘ ((𝔊 SMCF∘ 𝔉)(|ArrMap|)) = 𝔄(|Arr|)
proof–
```

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

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

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

Opposite of the contravariant composition of semifunctors

```
lemma op-smcf-smcf-cn-comp[smc-op-simps]:
  op-smcf (𝔊 SMCF∘ 𝔉) = op-smcf 𝔊 SMCF∘ op-smcf 𝔉
```

```
  unfolding op-smcf-def smcf-cn-comp-def dghm-field-simps
  by (auto simp: nat-omega-simps)
```

Further properties

```
lemma smcf-cn-comp-is-semifunctor[smc-cn-cs-intros]:
  — See section 1.2 in [15].
  assumes semicategory α 𝔄 and 𝔊 : 𝔅 SMC↦↦α ℭ and 𝔉 : 𝔄 SMC↦↦α 𝔅
  shows 𝔊 SMCF∘ 𝔉 : 𝔄 ↦↦SMCα ℭ
proof–
  interpret L: is-semifunctor α ‹op-smc 𝔅› ℭ 𝔊
    rewrites f : b ↦op-smc ℭ' a = f : a ↦ℭ' b for ℭ' f b a
    by (rule assms(2)) (simp-all add: smc-op-simps)
  interpret R: is-semifunctor α ‹op-smc 𝔄› 𝔅 𝔉
    rewrites f : b ↦op-smc ℭ' a = f : a ↦ℭ' b for ℭ' f b a
    by (rule assms(3)) (simp-all add: smc-op-simps)
  interpret 𝔄: semicategory α 𝔄 by (rule assms(1))
  show ?thesis
  proof(rule is-semifunctorI, unfold smcf-cn-comp-components(3,4) smc-op-simps)
    from assms show smcf-dghm (𝔊 SMCF∘ 𝔉) : smc-dg 𝔄 ↦↦DGα smc-dg ℭ
      by
        (
          cs-concl cs-shallow
            cs-simp: slicing-commute[symmetric]
            cs-intro: dg-cn-cs-intros slicing-intros
        )
    fix g b c f a assume g : b ↦𝔄 c f : a ↦𝔄 b
    with assms show (𝔊 SMCF∘ 𝔉)(‖ArrMap‖)(‖g ∘A𝔄 f‖) =
      (𝔊 SMCF∘ 𝔉)(‖ArrMap‖)(‖g‖) ∘Aℭ (𝔊 SMCF∘ 𝔉)(‖ArrMap‖)(‖f‖)
      by
        (
          cs-concl cs-shallow
            cs-simp: smc-cs-simps smc-cn-cs-simps smc-op-simps
            cs-intro: smc-cs-intros
        )
  qed
    (
      auto simp:
        smcf-cn-comp-def
        nat-omega-simps
        smc-cs-simps
        smc-op-simps
        smc-cs-intros
    )
qed

lemma smcf-cn-cov-comp-is-semifunctor[smc-cs-intros]:
  — See section 1.2 in [15].
  assumes 𝔊 : 𝔅 SMC↦↦α ℭ and 𝔉 : 𝔄 ↦↦SMCα 𝔅
  shows 𝔊 SMCF∘ 𝔉 : 𝔄 SMC↦↦α ℭ
proof–
  interpret L: is-semifunctor α ‹op-smc 𝔅› ℭ 𝔊
    rewrites f : b ↦op-smc ℭ' a = f : a ↦ℭ' b for ℭ' f b a
    by (rule assms(1)) (simp-all add: smc-op-simps)
  interpret R: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  show ?thesis
  proof(rule is-semifunctorI, unfold smcf-cn-comp-components(3,4) smc-op-simps)
    from assms show
```

```
    smcf-dghm (𝔊 SMCF∘ 𝔉) : smc-dg (op-smc 𝔄) ↦↦DGα smc-dg ℭ
    by
      (
        cs-concl cs-shallow
          cs-simp: slicing-commute[symmetric]
          cs-intro: dg-cn-cs-intros slicing-intros
      )
  show vfsequence (𝔊 SMCF∘ 𝔉) unfolding smcf-cn-comp-def by simp
  show vcard (𝔊 SMCF∘ 𝔉) = 4ℕ
    unfolding smcf-cn-comp-def by (auto simp: nat-omega-simps)
  show op-smc (𝔉(|HomDom|)) = op-smc 𝔄 by (simp add: smc-cs-simps)
  show 𝔊(|HomCod|) = ℭ by (simp add: smc-cs-simps)
  fix g b c f a assume g : c ↦𝔄 b f : b ↦𝔄 a
  with assms show
    (𝔊 SMCF∘ 𝔉)(|ArrMap|)(|f ∘A𝔄 g|) =
      (𝔊 SMCF∘ 𝔉)(|ArrMap|)(|g|) ∘Aℭ (𝔊 SMCF∘ 𝔉)(|ArrMap|)(|f|)
    by
      (
        cs-concl cs-shallow
          cs-simp: smc-cs-simps smc-cn-cs-simps smc-op-simps
          cs-intro: smc-cs-intros
      )
qed (auto intro: smc-cs-intros smc-op-intros)
qed

lemma smcf-cov-cn-comp-is-semifunctor[smc-cn-cs-intros]:
  — See section 1.2 in [15].
  assumes 𝔊 : 𝔅 ↦↦SMCα ℭ and 𝔉 : 𝔄 SMC↦↦α 𝔅
  shows 𝔊 ∘SMCF 𝔉 : 𝔄 SMC↦↦α ℭ
  using assms by (rule smcf-comp-is-semifunctor)
```

4.4.6 Identity semifunctor

Definition and elementary properties

See Chapter I-3 in [39].

```
abbreviation (input) smcf-id :: V ⇒ V where smcf-id ≡ dghm-id
```

Slicing.

```
lemma smcf-dghm-smcf-id[slicing-commute]:
  dghm-id (smc-dg ℭ) = smcf-dghm (smcf-id ℭ)
  unfolding dghm-id-def smc-dg-def smcf-dghm-def dghm-field-simps dg-field-simps
  by (simp add: nat-omega-simps)

context semicategory
begin

interpretation dg: digraph α ‹smc-dg ℭ› by (rule smc-digraph)

lemmas-with [unfolded slicing-simps]:
  smc-dghm-id-is-dghm = dg.dg-dghm-id-is-dghm

end
```

Object map

```
lemmas [smc-cs-simps] = dghm-id-ObjMap-app
```

Arrow map

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

Opposite identity semifunctor

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

An identity semifunctor is a semifunctor

lemma (**in** *semicategory*) *smc-smcf-id-is-semifunctor*: *smcf-id* $\mathfrak{C}$: $\mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
proof(*rule is-semifunctorI*, *unfold dghm-id-components*)
from *smc-dghm-id-is-dghm* **show**
smcf-dghm (*smcf-id* $\mathfrak{C}$) : *smc-dg* $\mathfrak{C} \mapsto\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{C}$
by (*auto simp*: *slicing-simps slicing-commute*)
fix *g b c f a* **assume** $g : b \mapsto_{\mathfrak{C}} c$ $f : a \mapsto_{\mathfrak{C}} b$
then show *vid-on* ($\mathfrak{C}$⦇*Arr*⦈)(⦇$g \circ_{A\mathfrak{C}} f$⦈) =
vid-on ($\mathfrak{C}$⦇*Arr*⦈)(⦇*g*⦈) $\circ_{A\mathfrak{C}}$ *vid-on* ($\mathfrak{C}$⦇*Arr*⦈)(⦇*f*⦈)
by (*metis smc-is-arrD*(1) *smc-Comp-is-arr vid-on-eq-atI*)
qed (*auto simp*: *semicategory-axioms dghm-id-def nat-omega-simps*)

lemma (**in** *semicategory*) *smc-smcf-id-is-semifunctor′*:
assumes $\mathfrak{A}$ = $\mathfrak{C}$ **and** $\mathfrak{B}$ = $\mathfrak{C}$
shows *smcf-id* $\mathfrak{C}$: $\mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule smc-smcf-id-is-semifunctor*)

lemmas [*smc-cs-intros*] = *semicategory.smc-smcf-id-is-semifunctor′*

Further properties

lemma (**in** *is-semifunctor*) *smcf-smcf-comp-smcf-id-left*[*smc-cs-simps*]:
— See Chapter I-3 in [39]).
smcf-id $\mathfrak{B}$ $\circ_{SMCF}$ $\mathfrak{F}$ = $\mathfrak{F}$
by (*rule smcf-eqI*, *unfold dghm-id-components dghm-comp-components*)
(*auto simp*: *smcf-ObjMap-vrange smcf-ArrMap-vrange intro*: *smc-cs-intros*)

lemmas [*smc-cs-simps*] = *is-semifunctor.smcf-smcf-comp-smcf-id-left*

lemma (**in** *is-semifunctor*) *smcf-smcf-comp-smcf-id-right*[*smc-cs-simps*]:
— See Chapter I-3 in [39]).
$\mathfrak{F}$ $\circ_{SMCF}$ *smcf-id* $\mathfrak{A}$ = $\mathfrak{F}$
by (*rule smcf-eqI*, *unfold dghm-id-components dghm-comp-components*)
(
auto
simp: *smcf-ObjMap-vrange smcf-ArrMap-vrange smc-cs-simps*
intro: *smc-cs-intros*
)

lemmas [*smc-cs-simps*] = *is-semifunctor.smcf-smcf-comp-smcf-id-right*

4.4.7 Constant semifunctor

Definition and elementary properties

See Chapter III-3 in [39].

abbreviation (*input*) *smcf-const* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *smcf-const* ≡ *dghm-const*

Slicing.

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

Object map

lemmas [*smc-cs-simps*] =
dghm-const-ObjMap-app

Arrow map

lemmas [*smc-cs-simps*] =
dghm-const-ArrMap-app

Opposite constant semifunctor

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

A constant semifunctor is a semifunctor

lemma *smcf-const-is-semifunctor*:
assumes *semicategory* α $\mathfrak{C}$
and *semicategory* α $\mathfrak{D}$
and *f* : *a* $\mapsto_{\mathfrak{D}}$ *a*
and [*simp*]: *f* $\circ_{A\mathfrak{D}}$ *f* = *f*
shows *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a f* : $\mathfrak{C}$ $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{D}$
proof–
interpret $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$ **by** (*rule assms*(1))
interpret $\mathfrak{D}$: *semicategory* α $\mathfrak{D}$ **by** (*rule assms*(2))
show *?thesis*
proof(*intro is-semifunctorI*, *tactic*‹*distinct-subgoals-tac*›)
from *assms* **show**
smcf-dghm (*dghm-const* $\mathfrak{C}$ $\mathfrak{D}$ *a f*) : *smc-dg* $\mathfrak{C}$ $\mapsto\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{D}$
by
(
cs-concl **cs-shallow**
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *dg-cs-intros slicing-intros*
)
show *vfsequence* (*smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a f*) **unfolding** *dghm-const-def* **by** *simp*
show *vcard* (*smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a f*) = $4_{\mathbb{N}}$
unfolding *dghm-const-def* **by** (*simp add*: *nat-omega-simps*)
fix *g′ b c f′ a′* **assume** *g′* : *b* $\mapsto_{\mathfrak{C}}$ *c f′* : *a′* $\mapsto_{\mathfrak{C}}$ *b*
with *assms*(1–3) **show** *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a f*(|*ArrMap*|)(|*g′* $\circ_{A\mathfrak{C}}$ *f′*|) =
smcf-const $\mathfrak{C}$ $\mathfrak{D}$ *a f*(|*ArrMap*|)(|*g′*|) $\circ_{A\mathfrak{D}}$ *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ *a f*(|*ArrMap*|)(|*f′*|)
by (*cs-concl* **cs-simp**: *assms*(4) *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
qed (*auto simp*: *assms*(1,2) *dghm-const-components*)
qed

lemma *smcf-const-is-semifunctor′*[*smc-cs-intros*]:
assumes *semicategory* α $\mathfrak{C}$
and *semicategory* α $\mathfrak{D}$

and $f : a \mapsto_{\mathfrak{D}} a$
and $f \circ_{A\mathfrak{D}} f = f$
and $\mathfrak{A} = \mathfrak{C}$
and $\mathfrak{B} = \mathfrak{D}$
shows *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
using *assms*(1–4) **unfolding** *assms*(5,6) **by** (*rule smcf-const-is-semifunctor*)

Further properties

lemma (in *is-semifunctor*) *smcf-smcf-comp-smcf-const*[*smc-cs-simps*]:
assumes *semicategory* α $\mathfrak{C}$ **and** $f : a \mapsto_{\mathfrak{C}} a$ **and** $f \circ_{A\mathfrak{C}} f = f$
shows *smcf-const* $\mathfrak{B}$ $\mathfrak{C}$ a $f \circ_{SMCF}$ $\mathfrak{F}$ = *smcf-const* $\mathfrak{A}$ $\mathfrak{C}$ a f
proof(*rule smcf-dghm-eqI*)
interpret $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$ **by** (*rule assms*(1))
from *assms*(2) **show** *smcf-const* $\mathfrak{B}$ $\mathfrak{C}$ a $f \circ_{SMCF}$ $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
by
(
cs-concl **cs-shallow**
cs-simp: *smc-cs-simps assms*(3) **cs-intro**: *smc-cs-intros*
)
from *assms*(2) **show** *smcf-const* $\mathfrak{A}$ $\mathfrak{C}$ a $f : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
by
(
cs-concl **cs-shallow**
cs-simp: *smc-cs-simps assms*(3) **cs-intro**: *smc-cs-intros*
)
from *is-dghm.dghm-dghm-comp-dghm-const*[
OF smcf-is-dghm $\mathfrak{C}$.*smc-digraph*, *unfolded slicing-simps*, *OF assms*(2)
]
show *smcf-dghm* (*smcf-const* $\mathfrak{B}$ $\mathfrak{C}$ a $f \circ_{SMCF}$ $\mathfrak{F}$) = *smcf-dghm* (*smcf-const* $\mathfrak{A}$ $\mathfrak{C}$ a f)
by (*cs-prems* **cs-shallow cs-simp**: *slicing-simps slicing-commute*)
qed *simp-all*

lemmas [*smc-cs-simps*] = *is-semifunctor.smcf-smcf-comp-smcf-const*

4.4.8 Faithful semifunctor

Definition and elementary properties

See Chapter I-3 in [39].

locale *is-ft-semifunctor* = *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *ft-smcf-is-ft-dghm*:
smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A} \mapsto\mapsto_{DG.faithful\alpha}$ *smc-dg* $\mathfrak{B}$

syntax *-is-ft-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $\mapsto\mapsto_{SMC.faithful1}$ -)$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-ft-semifunctor* $\rightleftharpoons$ *is-ft-semifunctor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.faithful\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-ft-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

lemma (in *is-ft-semifunctor*) *ft-smcf-is-ft-dghm'*[*slicing-intros*]:
assumes $\mathfrak{A}'$ = *smc-dg* $\mathfrak{A}$ **and** $\mathfrak{B}'$ = *smc-dg* $\mathfrak{B}$
shows *smcf-dghm* $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{DG.faithful\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule ft-smcf-is-ft-dghm*)

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

Rules.

lemma (in *is-ft-semifunctor*) *is-ft-semifunctor-axioms'*[*smcf-cs-intros*]:

assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{SMC.faithful_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-ft-semifunctor-axioms*)

mk-ide rf *is-ft-semifunctor-def*[*unfolded is-ft-semifunctor-axioms-def*]
|*intro is-ft-semifunctorI*|
|*dest is-ft-semifunctorD*[*dest*]|
|*elim is-ft-semifunctorE*[*elim*]|

lemmas [*smcf-cs-intros*] = *is-ft-semifunctorD*(1)

lemma *is-ft-semifunctorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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 $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.faithful\alpha} \mathfrak{B}$
using *assms*
by (*intro is-ft-semifunctorI*)
(
simp-all add:
assms(1)
is-ft-dghmI[*OF is-semifunctorD*(6)[*OF assms*(1)], *unfolded slicing-simps*]
)

lemma *is-ft-semifunctorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.faithful\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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-semifunctorD[*OF assms*(1)]
is-ft-dghmD(2)[
OF is-ft-semifunctorD(2)[*OF assms*(1)], *unfolded slicing-simps*
]
)

lemma *is-ft-semifunctorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.faithful\alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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-semifunctorD'*)

lemma *is-ft-semifunctorI''*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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 $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.faithful\alpha} \mathfrak{B}$
by
(
intro is-ft-semifunctorI assms,
rule is-ft-dghmI'',
unfold slicing-simps,
rule is-semifunctor.smcf-is-dghm[*OF assms*(1)],
rule assms(2)
)

Elementary properties.

context *is-ft-semifunctor*

begin

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

lemmas-with [*unfolded slicing-simps*]:
*ft-smcf-v*11*-on-Hom* = *dghm.ft-dghm-v*11*-on-Hom*
and *ft-smcf-ArrMap-eqD* = *dghm.ft-dghm-ArrMap-eqD*

end

Opposite faithful semifunctor

lemma (**in** *is-ft-semifunctor*) *is-ft-semifunctor-op*:
op-smcf $\mathfrak{F}$: *op-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.faithful\alpha}$ *op-smc* $\mathfrak{B}$
by
(
rule is-ft-semifunctorI,
unfold smc-op-simps slicing-simps slicing-commute[*symmetric*]
)
(
simp-all add:
is-semifunctor-op is-ft-dghm.ft-dghm-op-dghm-is-ft-dghm
ft-smcf-is-ft-dghm
)

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

lemmas *is-ft-semifunctor-op*[*smc-op-intros*] =
is-ft-semifunctor.is-ft-semifunctor-op′

The composition of faithful semifunctors is a faithful semifunctor

lemma *smcf-comp-is-ft-semifunctor*[*smcf-cs-intros*]:
— See Chapter I-3 in [39].
assumes $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{SMC.faithful\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.faithful\alpha}$ $\mathfrak{B}$
shows $\mathfrak{G}$ $\circ_{SMCF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.faithful\alpha}$ $\mathfrak{C}$
proof(*intro is-ft-semifunctorI*)
interpret $\mathfrak{G}$: *is-ft-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **by** (*simp add*: *assms*(1))
interpret $\mathfrak{F}$: *is-ft-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*simp add*: *assms*(2))
from $\mathfrak{F}$.*is-semifunctor-axioms* $\mathfrak{G}$.*is-semifunctor-axioms* **show** $\mathfrak{GF}$:
$\mathfrak{G}$ $\circ_{SMCF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{C}$
by (*auto intro*: *smc-cs-intros*)
then interpret *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{C}$ ‹$\mathfrak{G}$ $\circ_{SMCF}$ $\mathfrak{F}$› .
show *smcf-dghm* ($\mathfrak{G}$ $\circ_{SMCF}$ $\mathfrak{F}$) : *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG.faithful\alpha}$ *smc-dg* $\mathfrak{C}$
unfolding *slicing-simps slicing-commute*[*symmetric*]
by (*auto intro*: *dghm-cs-intros slicing-intros*)
qed

4.4.9 Full semifunctor

Definition and elementary properties

See Chapter I-3 in [39].

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

assumes *fl-smcf-is-fl-dghm*:
smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG.full\alpha}$ *smc-dg* $\mathfrak{B}$

syntax *-is-fl-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $\mapsto\!\!\mapsto_{SMC.full1}$ -)$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-fl-semifunctor* $\rightleftharpoons$ *is-fl-semifunctor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC.full\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-fl-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

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

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

Rules.

mk-ide rf *is-fl-semifunctor-def*[*unfolded is-fl-semifunctor-axioms-def*]
|*intro is-fl-semifunctorI*|
|*dest is-fl-semifunctorD*[*dest*]|
|*elim is-fl-semifunctorE*[*elim*]|

lemmas [*smcf-cs-intros*] = *is-fl-semifunctorD*(1)

lemma *is-fl-semifunctorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC\alpha} \mathfrak{B}$
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 $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC.full\alpha} \mathfrak{B}$
using *assms*
by (*intro is-fl-semifunctorI*)
(
simp-all add:
assms(1)
is-fl-dghmI[*OF is-semifunctorD*(6)[*OF assms*(1)], *unfolded slicing-simps*]
)

lemma *is-fl-semifunctorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC.full\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC\alpha} \mathfrak{B}$
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-semifunctorD[*OF assms*(1)]
is-fl-dghmD(2)[
OF is-fl-semifunctorD(2)[*OF assms*(1)], *unfolded slicing-simps*
]
)

lemma *is-fl-semifunctorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC.full\alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\!\!\mapsto_{SMC\alpha} \mathfrak{B}$
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-semifunctorD'*)

Elementary properties.

context *is-fl-semifunctor*
begin

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

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

end

Opposite full semifunctor

lemma (**in** *is-fl-semifunctor*) *is-fl-semifunctor-op*:
op-smcf $\mathfrak{F}$: *op-smc* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.full\alpha}$ *op-smc* $\mathfrak{B}$
by
(
rule is-fl-semifunctorI,
unfold smc-op-simps slicing-simps slicing-commute[*symmetric*]
)
(
simp-all add:
is-semifunctor-op
is-fl-dghm.fl-dghm-op-dghm-is-fl-dghm
fl-smcf-is-fl-dghm
)

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

lemmas *is-fl-semifunctor-op*[*smc-op-intros*] =
is-fl-semifunctor.is-fl-semifunctor-op

The composition of full semifunctors is a full semifunctor

lemma *smcf-comp-is-fl-semifunctor*[*smcf-cs-intros*]:
— See Chapter I-3 in [39].
assumes $\mathfrak{G}$: $\mathfrak{B}$ $\mapsto\!\!\mapsto_{SMC.full\alpha}$ $\mathfrak{C}$ **and** $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.full\alpha}$ $\mathfrak{B}$
shows $\mathfrak{G}$ $\circ_{SMCF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.full\alpha}$ $\mathfrak{C}$
proof(*intro is-fl-semifunctorI*)
interpret $\mathfrak{F}$: *is-fl-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **using** *assms*(2) **by** *simp*
interpret $\mathfrak{G}$: *is-fl-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **using** *assms*(1) **by** *simp*
from $\mathfrak{F}$.*is-semifunctor-axioms* $\mathfrak{G}$.*is-semifunctor-axioms* **show**
$\mathfrak{G}$ $\circ_{SMCF}$ $\mathfrak{F}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{C}$
by (*auto intro*: *smc-cs-intros*)
show *smcf-dghm* ($\mathfrak{G}$ $\circ_{DGHM}$ $\mathfrak{F}$) : *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG.full\alpha}$ *smc-dg* $\mathfrak{C}$
unfolding *slicing-commute*[*symmetric*]
by (*auto intro*: *dghm-cs-intros slicing-intros*)
qed

4.4.10 Fully faithful semifunctor

Definition and elementary properties

See Chapter I-3 in [39]).

locale *is-ff-semifunctor* =

is-ft-semifunctor α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ + *is-fl-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

syntax *-is-ff-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $\mapsto\mapsto_{SMC.ff^1}$ -)$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-ff-semifunctor* $\rightleftharpoons$ *is-ff-semifunctor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.ff\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-ff-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

Rules.

mk-ide rf *is-ff-semifunctor-def*
|*intro is-ff-semifunctorI*|
|*dest is-ff-semifunctorD*[*dest*]|
|*elim is-ff-semifunctorE*[*elim*]|

lemmas [*smcf-cs-intros*] = *is-ff-semifunctorD*

Elementary properties.

lemma (**in** *is-ff-semifunctor*) *ff-smcf-is-ff-dghm*:
smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A} \mapsto\mapsto_{DG.ff\alpha}$ *smc-dg* $\mathfrak{B}$
by (*rule is-ff-dghmI*) (*auto intro*: *slicing-intros*)

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

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

Opposite fully faithful semifunctor

lemma (**in** *is-ff-semifunctor*) *is-ff-semifunctor-op*:
op-smcf $\mathfrak{F}$: *op-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.ff\alpha}$ *op-smc* $\mathfrak{B}$
by (*rule is-ff-semifunctorI*)
(*auto simp*: *is-fl-semifunctor-op is-ft-semifunctor-op*)

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

lemmas *is-ff-semifunctor-op*[*smc-op-intros*] =
is-ff-semifunctor.is-ff-semifunctor-op'

The composition of fully faithful semifunctors is a fully faithful semifunctor

lemma *smcf-comp-is-ff-semifunctor*[*smcf-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC.ff\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.ff\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{SMCF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.ff\alpha} \mathfrak{C}$
using *assms*
by (*intro is-ff-semifunctorI*, *elim is-ff-semifunctorE*)
(*auto intro*: *smcf-cs-intros*)

4.4.11 Isomorphism of semicategories

Definition and elementary properties

See Chapter I-3 in [39].

locale *is-iso-semifunctor* = *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
assumes *iso-smcf-is-iso-dghm*:

smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A}$ $\mapsto\mapsto_{DG.iso\alpha}$ *smc-dg* $\mathfrak{B}$

syntax *-is-iso-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
($\langle$(- :/ - $\mapsto\mapsto_{SMC.iso1}$ -)$\rangle$ [51, 51, 51] 51)
syntax-consts *-is-iso-semifunctor* $\rightleftharpoons$ *is-iso-semifunctor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-iso-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

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

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

Rules.

lemma (in *is-iso-semifunctor*) *is-iso-semifunctor-axioms'*[*smcf-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{A}' = \mathfrak{A}$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A}' \mapsto\mapsto_{SMC.iso_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule is-iso-semifunctor-axioms*)

mk-ide rf *is-iso-semifunctor-def*[*unfolded is-iso-semifunctor-axioms-def*]
|*intro is-iso-semifunctorI*|
|*dest is-iso-semifunctorD*[*dest*]|
|*elim is-iso-semifunctorE*[*elim*]|

lemma *is-iso-semifunctorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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 $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
using *assms*
by (*intro is-iso-semifunctorI*)
(
simp-all add:
assms(1)
is-iso-dghmI[*OF is-semifunctorD*(6)[*OF assms*(1)], *unfolded slicing-simps*]
)

lemma *is-iso-semifunctorD'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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-semifunctorD[*OF assms*(1)]
is-iso-dghmD(2–5)[
OF is-iso-semifunctorD(2)[*OF assms*(1)], *unfolded slicing-simps*
]
)

lemma *is-iso-semifunctorE'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$

obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
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-semifunctorD'*)

Elementary properties.

context *is-iso-semifunctor*
begin

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

lemmas-with [*unfolded slicing-simps*]:
iso-smcf-ObjMap-vrange[*smcf-cs-simps*] = *dghm.iso-dghm-ObjMap-vrange*
and *iso-smcf-ArrMap-vrange*[*smcf-cs-simps*] = *dghm.iso-dghm-ArrMap-vrange*

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

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

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

end

sublocale *is-iso-semifunctor* ⊆ *is-ff-semifunctor*
proof–
interpret *dghm*: *is-iso-dghm* α ‹*smc-dg* $\mathfrak{A}$› ‹*smc-dg* $\mathfrak{B}$› ‹*smcf-dghm* $\mathfrak{F}$›
by (*rule iso-smcf-is-iso-dghm*)
show $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.ff\alpha} \mathfrak{B}$ **by** *unfold-locales*
qed

lemmas (**in** *is-iso-semifunctor*) *iso-smcf-is-ff-semifunctor* =
is-ff-semifunctor-axioms

lemmas [*smcf-cs-intros*] = *is-iso-semifunctor.iso-smcf-is-ff-semifunctor*

Opposite isomorphism of semicategories

lemma (**in** *is-iso-semifunctor*) *is-iso-semifunctor-op*:
op-smcf $\mathfrak{F}$: *op-smc* $\mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha}$ *op-smc* $\mathfrak{B}$
by
(
rule is-iso-semifunctorI,
unfold smc-op-simps slicing-simps slicing-commute[*symmetric*]

```
  )
  (
    simp-all add:
      is-semifunctor-op is-iso-dghm.is-iso-dghm-op iso-smcf-is-iso-dghm
  )
```

lemmas *is-iso-semifunctor-op*[*smc-op-intros*] =
is-iso-semifunctor.is-iso-semifunctor-op

The composition of isomorphisms of semicategories is an isomorphism of semicategories

lemma *smcf-comp-is-iso-semifunctor*[*smcf-cs-intros*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{SMCF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{C}$
proof(*intro is-iso-semifunctorI*)
interpret $\mathfrak{F}$: *is-iso-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **using** *assms* **by** *auto*
interpret $\mathfrak{G}$: *is-iso-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{G}$ **using** *assms* **by** *auto*
from $\mathfrak{F}$.*is-semifunctor-axioms* $\mathfrak{G}$.*is-semifunctor-axioms* **show**
$\mathfrak{G} \circ_{SMCF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
by (*auto intro*: *smcf-cs-intros*)
show *smcf-dghm* ($\mathfrak{G} \circ_{DGHM} \mathfrak{F}$) : *smc-dg* $\mathfrak{A} \mapsto\mapsto_{DG.iso\alpha}$ *smc-dg* $\mathfrak{C}$
by
(
auto
intro: *dghm-cs-intros slicing-intros*
simp: *slicing-commute*[*symmetric*]
)
qed

4.4.12 Inverse semifunctor

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

lemmas [*smc-cs-simps*] = *inv-dghm-components*(3,4)

Slicing.

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

context *is-iso-semifunctor*
begin

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

lemmas-with [*unfolded slicing-simps slicing-commute*]:
*inv-smcf-ObjMap-v*11 = *dghm.inv-dghm-ObjMap-v*11
and *inv-smcf-ObjMap-vdomain* = *dghm.inv-dghm-ObjMap-vdomain*
and *inv-smcf-ObjMap-app* = *dghm.inv-dghm-ObjMap-app*
and *inv-smcf-ObjMap-vrange* = *dghm.inv-dghm-ObjMap-vrange*
and *inv-smcf-ArrMap-v*11 = *dghm.inv-dghm-ArrMap-v*11
and *inv-smcf-ArrMap-vdomain* = *dghm.inv-dghm-ArrMap-vdomain*
and *inv-smcf-ArrMap-app* = *dghm.inv-dghm-ArrMap-app*
and *inv-smcf-ArrMap-vrange* = *dghm.inv-dghm-ArrMap-vrange*

and *iso-smcf-ObjMap-inv-smcf-ObjMap-app*[*smcf-cs-simps*] =
dghm.iso-dghm-ObjMap-inv-dghm-ObjMap-app
and *iso-smcf-ArrMap-inv-smcf-ArrMap-app*[*smcf-cs-simps*] =
dghm.iso-dghm-ArrMap-inv-dghm-ArrMap-app
and *iso-smcf-HomDom-is-arr-conv* = *dghm.iso-dghm-HomDom-is-arr-conv*
and *iso-smcf-HomCod-is-arr-conv* = *dghm.iso-dghm-HomCod-is-arr-conv*
and *iso-inv-smcf-ObjMap-smcf-ObjMap-app*[*smcf-cs-simps*] =
dghm.iso-inv-dghm-ObjMap-dghm-ObjMap-app
and *iso-inv-smcf-ArrMap-smcf-ArrMap-app*[*smcf-cs-simps*] =
dghm.iso-inv-dghm-ArrMap-dghm-ArrMap-app

end

lemmas [*smcf-cs-intros*] =
*is-iso-semifunctor.inv-smcf-ObjMap-v*11
*is-iso-semifunctor.inv-smcf-ArrMap-v*11

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

4.4.13 An isomorphism of semicategories is an isomorphism in the category *SemiCAT*

lemma *is-iso-arr-is-iso-semifunctor*:
— See Chapter I-3 in [39].
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{A}$
and $\mathfrak{G} \circ_{SMCF} \mathfrak{F}$ = *smcf-id* $\mathfrak{A}$
and $\mathfrak{F} \circ_{SMCF} \mathfrak{G}$ = *smcf-id* $\mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{F}$: *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(1))
interpret $\mathfrak{G}$: *is-semifunctor* α $\mathfrak{B}$ $\mathfrak{A}$ $\mathfrak{G}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule is-iso-semifunctorI*)
have *dg-*$\mathfrak{GFA}$: *smcf-dghm* $\mathfrak{G} \circ_{DGHM}$ *smcf-dghm* $\mathfrak{F}$ = *dghm-id* (*smc-dg* $\mathfrak{A}$)
by (*simp add*: *assms*(3) *smcf-dghm-smcf-id smcf-dghm-smcf-comp*)
have *dg-*$\mathfrak{FGB}$: *smcf-dghm* $\mathfrak{F} \circ_{DGHM}$ *smcf-dghm* $\mathfrak{G}$ = *dghm-id* (*smc-dg* $\mathfrak{B}$)
by (*simp add*: *assms*(4) *smcf-dghm-smcf-id smcf-dghm-smcf-comp*)
from $\mathfrak{F}$.*smcf-is-dghm* $\mathfrak{G}$.*smcf-is-dghm dg-*$\mathfrak{GFA}$ *dg-*$\mathfrak{FGB}$ **show**
smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A} \mapsto\mapsto_{DG.iso\alpha}$ *smc-dg* $\mathfrak{B}$
by (*rule is-iso-arr-is-iso-dghm*)
qed (*simp add*: $\mathfrak{F}$.*is-semifunctor-axioms*)
qed

lemma *is-iso-semifunctor-is-iso-arr*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
shows [*smcf-cs-intros*]: *inv-smcf* $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{A}$
and [*smcf-cs-simps*]: *inv-smcf* $\mathfrak{F} \circ_{SMCF} \mathfrak{F}$ = *smcf-id* $\mathfrak{A}$

```
  and [smcf-cs-simps]: 𝔉 ∘_SMCF inv-smcf 𝔉 = smcf-id 𝔅
proof–

  let ?𝔊 = ‹inv-smcf 𝔉›

  interpret is-iso-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(1))

  note is-iso-dghm = is-iso-dghm-is-iso-arr[OF iso-smcf-is-iso-dghm]

  show 𝔊: ?𝔊 : 𝔅 ↦↦_SMC.isoα 𝔄
  proof
    (
      intro is-iso-semifunctorI is-semifunctorI;
      (unfold slicing-commute[symmetric])?
    )
    show vfsequence (inv-smcf 𝔉) unfolding inv-dghm-def by simp
    show vcard (inv-smcf 𝔉) = 4_ℕ
      unfolding inv-dghm-def by (simp add: nat-omega-simps)
    show inv-iso-dghm-𝔉:
      inv-dghm (smcf-dghm 𝔉) : smc-dg 𝔅 ↦↦_DG.isoα smc-dg 𝔄
      by (rule is-iso-dghm(1))
    show inv-dghm-𝔉: inv-dghm (smcf-dghm 𝔉) : smc-dg 𝔅 ↦↦_DGα smc-dg 𝔄
      by (rule is-iso-dghmD(1)[OF inv-iso-dghm-𝔉])
    fix b c g a f assume prems: g : b ↦_𝔅 c f : a ↦_𝔅 b
    note is-arr-inv = is-dghm.dghm-ArrMap-is-arr[
        OF inv-dghm-𝔉, unfolded slicing-simps slicing-commute
        ]
    from prems is-arr-inv[OF prems(1)] is-arr-inv[OF prems(2)] show
      inv-smcf 𝔉(|ArrMap|)(|g ∘_A𝔅 f|) =
        inv-smcf 𝔉(|ArrMap|)(|g|) ∘_A𝔄 inv-smcf 𝔉(|ArrMap|)(|f|)
       unfolding inv-dghm-components
       by (intro v11.v11-vconverse-app)
         (
           cs-concl cs-shallow
            cs-intro: smc-cs-intros V-cs-intros
            cs-simp: V-cs-simps smc-cs-simps
         )+
  qed (auto simp: smc-cs-simps intro: smc-cs-intros)

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

  show 𝔉 ∘_SMCF inv-smcf 𝔉 = smcf-id 𝔅
  proof(rule smcf-eqI, unfold dghm-comp-components inv-dghm-components)
    from 𝔊 is-semifunctor-axioms show 𝔉 ∘_SMCF inv-smcf 𝔉 : 𝔅 ↦↦_SMCα 𝔅
      by (blast intro: smc-cs-intros)
  qed
    (
```

simp-all add:
HomCod.smc-smcf-id-is-semifunctor
ObjMap.v11-vcomp-vconverse′
ArrMap.v11-vcomp-vconverse′
dghm-id-components
)

qed

An identity semifunctor is an isomorphism of semicategories

lemma (**in** *semicategory*) *smc-smcf-id-is-iso-semifunctor*:
smcf-id $\mathfrak{C} : \mathfrak{C} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{C}$
by (*rule is-iso-semifunctorI*, *unfold slicing-simps slicing-commute*[*symmetric*])
(
simp-all add:
smc-smcf-id-is-semifunctor digraph.dg-dghm-id-is-iso-dghm smc-digraph
)

lemma (**in** *semicategory*) *smc-smcf-id-is-iso-semifunctor′*[*smcf-cs-intros*]:
assumes $\mathfrak{A}' = \mathfrak{C}$ **and** $\mathfrak{B}' = \mathfrak{C}$
shows *smcf-id* $\mathfrak{C} : \mathfrak{A}' \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}'$
unfolding *assms* **by** (*rule smc-smcf-id-is-iso-semifunctor*)

lemmas [*smcf-cs-intros*] = *semicategory.smc-smcf-id-is-iso-semifunctor′*

4.4.14 Isomorphic semicategories

Definition and elementary properties

See Chapter I-3 in [39]).

locale *iso-semicategory* = *L*: *semicategory* α $\mathfrak{A}$ + *R*: *semicategory* α $\mathfrak{B}$
for α $\mathfrak{A}$ $\mathfrak{B}$ +
assumes *iso-smc-is-iso-semifunctor*: $\exists \mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$

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

Rules.

lemma *iso-semicategoryI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
shows $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$
using *assms*
unfolding *iso-semicategory-def iso-semicategory-axioms-def*
by *blast*

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

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

Elementary properties.

lemma (**in** *iso-semicategory*) *iso-smc-iso-digraph*: *smc-dg* $\mathfrak{A} \approx_{DG\alpha}$ *smc-dg* $\mathfrak{B}$
using *iso-smc-is-iso-semifunctor*
by (*auto intro*: *slicing-intros iso-digraphI*)

A semicategory isomorphism is an equivalence relation

lemma *iso-semicategory-refl*:
assumes *semicategory* α $\mathfrak{A}$
shows $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{A}$
proof(*rule iso-semicategoryI*[*of* - - - ‹*smcf-id* $\mathfrak{A}$›])
interpret *semicategory* α $\mathfrak{A}$ **by** (*rule assms*)
show *smcf-id* $\mathfrak{A} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{A}$
by (*simp add*: *smc-smcf-id-is-iso-semifunctor*)
qed

lemma *iso-semicategory-sym*[*sym*]:
assumes $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$
shows $\mathfrak{B} \approx_{SMC\alpha} \mathfrak{A}$
proof–
interpret *iso-semicategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*)
from *iso-smc-is-iso-semifunctor* **obtain** $\mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{B}$
by *clarsimp*
then have *inv-smcf* $\mathfrak{F} : \mathfrak{B} \mapsto\mapsto_{SMC.iso\alpha} \mathfrak{A}$
by (*simp add*: *is-iso-semifunctor-is-iso-arr*(1))
then show *?thesis* **by** (*auto intro*: *iso-semicategoryI*)
qed

lemma *iso-semicategory-trans*[*trans*]:
assumes $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \approx_{SMC\alpha} \mathfrak{C}$
shows $\mathfrak{A} \approx_{SMC\alpha} \mathfrak{C}$
proof–
interpret *L*: *iso-semicategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(1))
interpret *R*: *iso-semicategory* α $\mathfrak{B}$ $\mathfrak{C}$ **by** (*rule assms*(2))
from *L.iso-smc-is-iso-semifunctor R.iso-smc-is-iso-semifunctor* **show** *?thesis*
by (*auto intro*: *iso-semicategoryI smcf-cs-intros*)
qed

4.5 Smallness for semifunctors

4.5.1 Semifunctor with tiny maps

Definition and elementary properties

locale *is-tm-semifunctor* = *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *tm-smcf-is-tm-dghm*[*slicing-intros*]:
    smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG.tm\alpha}$ *smc-dg* $\mathfrak{B}$

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

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

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

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

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

lemma (**in** *is-tm-semifunctor*) *tm-smcf-is-tm-dghm'*:
  assumes $\alpha' = \alpha$
    and $\mathfrak{A}' =$ *smc-dg* $\mathfrak{A}$
    and $\mathfrak{B}' =$ *smc-dg* $\mathfrak{B}$
  shows *smcf-dghm* $\mathfrak{F}$: $\mathfrak{A}' \mapsto\!\!\mapsto_{DG.tm_{\alpha'}} \mathfrak{B}'$
  unfolding *assms* **by** (*rule tm-smcf-is-tm-dghm*)

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

Rules.

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

mk-ide rf *is-tm-semifunctor-def*[*unfolded is-tm-semifunctor-axioms-def*]
  |*intro is-tm-semifunctorI*|
  |*dest is-tm-semifunctorD*[*dest*]|
  |*elim is-tm-semifunctorE*[*elim*]|

lemmas [*smc-small-cs-intros*] = *is-tm-semifunctorD*(1)

Slicing.

context *is-tm-semifunctor*
begin

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

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

tm-smcf-ObjMap-in-Vset[*smc-small-cs-intros*] = *dghm.tm-dghm-ObjMap-in-Vset*
and *tm-smcf-ArrMap-in-Vset*[*smc-small-cs-intros*] = *dghm.tm-dghm-ArrMap-in-Vset*

end

Elementary properties.

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

Further rules.

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

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

lemmas [*smc-small-cs-intros*] = *is-tm-semifunctorD'*(1)

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

Size.

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

```
    elts (set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC.tmα} 𝔅})
  proof
    (
    simp only: elts-of-set small-all-smcfs if-True,
    rule subsetI,
    unfold mem-Collect-eq
    )
    fix 𝔉 assume ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC.tmα} 𝔅
    then obtain 𝔄 𝔅 where 𝔉 : 𝔄 ↦↦_{SMC.tmα} 𝔅 by clarsimp
    then interpret is-tm-semifunctor α 𝔄 𝔅 𝔉 by simp
    show ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅 by (auto intro: is-semifunctor-axioms)
  qed
qed
```

Opposite semifunctor with tiny maps

```
lemma (in is-tm-semifunctor) is-tm-semifunctor-op:
  op-smcf 𝔉 : op-smc 𝔄 ↦↦_{SMC.tmα} op-smc 𝔅
  by (intro is-tm-semifunctorI', unfold smc-op-simps)
    (cs-concl cs-intro: smc-cs-intros smc-op-intros smc-small-cs-intros)

lemma (in is-tm-semifunctor) is-tm-semifunctor-op'[smc-op-intros]:
  assumes 𝔄' = op-smc 𝔄 and 𝔅' = op-smc 𝔅 and α' = α
  shows op-smcf 𝔉 : 𝔄' ↦↦_{SMC.tm_α'} 𝔅'
  unfolding assms by (rule is-tm-semifunctor-op)

lemmas is-tm-semifunctor-op[smc-op-intros] = is-tm-semifunctor.is-tm-semifunctor-op'
```

Composition of semifunctors with tiny maps

```
lemma smcf-comp-is-tm-semifunctor[smc-small-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦_{SMC.tmα} ℭ and 𝔉 : 𝔄 ↦↦_{SMC.tmα} 𝔅
  shows 𝔊 ∘_{SMCF} 𝔉 : 𝔄 ↦↦_{SMC.tmα} ℭ
proof(rule is-tm-semifunctorI)
  interpret 𝔉: is-tm-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  interpret 𝔊: is-tm-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  show smcf-dghm (𝔊 ∘_{SMCF} 𝔉) : smc-dg 𝔄 ↦↦_{DG.tmα} smc-dg ℭ
    unfolding slicing-commute[symmetric]
    using 𝔉.tm-smcf-is-tm-dghm 𝔊.tm-smcf-is-tm-dghm
    by (auto simp: dg-small-cs-intros)
  show 𝔊 ∘_{SMCF} 𝔉 : 𝔄 ↦↦_{SMCα} ℭ by (auto intro: smc-cs-intros)
qed
```

Finite semicategories and semifunctors with tiny maps

```
lemma (in is-semifunctor) smcf-is-tm-semifunctor-if-HomDom-finite-semicategory:
  assumes finite-semicategory α 𝔄
  shows 𝔉 : 𝔄 ↦↦_{SMC.tmα} 𝔅
proof(intro is-tm-semifunctorI)
  interpret 𝔄: finite-semicategory α 𝔄 by (rule assms(1))
  show smcf-dghm 𝔉 : smc-dg 𝔄 ↦↦_{DG.tmα} smc-dg 𝔅
    by
      (
      rule is-dghm.dghm-is-tm-dghm-if-HomDom-finite-digraph[
        OF smcf-is-dghm 𝔄.fin-smc-finite-digraph
        ]
      )
qed (auto intro: smc-cs-intros)
```

Constant semifunctor with tiny maps

lemma *smcf-const-is-tm-semifunctor*:
  assumes *tiny-semicategory* α $\mathfrak{C}$
    and *semicategory* α $\mathfrak{D}$
    and $f : a \mapsto_{\mathfrak{D}} a$
    and $f \circ_{A\mathfrak{D}} f = f$
  shows *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{C} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{D}$
proof(*intro is-tm-semifunctorI*)
  interpret $\mathfrak{C}$: *tiny-semicategory* α $\mathfrak{C}$ **by** (*rule assms*(1))
  interpret $\mathfrak{D}$: *semicategory* α $\mathfrak{D}$ **by** (*rule assms*(2))
  show *smcf-dghm* (*smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a f) : *smc-dg* $\mathfrak{C}$ $\mapsto\mapsto_{DG.tm\alpha}$ *smc-dg* $\mathfrak{D}$
    unfolding *slicing-commute*[*symmetric*]
    by (*rule dghm-const-is-tm-dghm*)
      (*auto simp*: *slicing-simps* $\mathfrak{C}$.*tiny-smc-tiny-digraph assms*(3) $\mathfrak{D}$.*smc-digraph*)
  from *assms* **show** *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
    by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
qed

lemma *smcf-const-is-tm-semifunctor′*:
  assumes *tiny-semicategory* α $\mathfrak{C}$
    and *semicategory* α $\mathfrak{D}$
    and $f : a \mapsto_{\mathfrak{D}} a$
    and $f \circ_{A\mathfrak{D}} f = f$
    and $\mathfrak{C}' = \mathfrak{C}$
    and $\mathfrak{D}' = \mathfrak{D}$
  shows *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{C}' \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{D}'$
  using *assms*(1–4) **unfolding** *assms*(5,6) **by** (*rule smcf-const-is-tm-semifunctor*)

lemmas [*smc-small-cs-intros*] = *smcf-const-is-tm-semifunctor′*

4.5.2 Tiny semifunctor

Definition and elementary properties

locale *is-tiny-semifunctor* = *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **for** α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  assumes *tiny-smcf-is-tiny-dghm*[*slicing-intros*]:
    smcf-dghm $\mathfrak{F}$: *smc-dg* $\mathfrak{A}$ $\mapsto\mapsto_{DG.tiny\alpha}$ *smc-dg* $\mathfrak{B}$

syntax *-is-tiny-semifunctor* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (‹(- :/ - $\mapsto\mapsto_{SMC.tiny_1}$ -)› [51, 51, 51] 51)
syntax-consts *-is-tiny-semifunctor* $\rightleftharpoons$ *is-tiny-semifunctor*
translations $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B} \rightleftharpoons$ *CONST is-tiny-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation (*input*) *is-cn-tiny-smcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  where *is-cn-tiny-smcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F} \equiv \mathfrak{F}$: *op-smc* $\mathfrak{A}$ $\mapsto\mapsto_{SMC.tiny\alpha}$ $\mathfrak{B}$

syntax *-is-cn-tiny-smcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  (‹(- :/ - $_{SMC.tiny}\mapsto\mapsto_1$ -)› [51, 51, 51] 51)
syntax-consts *-is-cn-tiny-smcf* $\rightleftharpoons$ *is-cn-tiny-smcf*
translations $\mathfrak{F} : \mathfrak{A}$ $_{SMC.tiny}\mapsto\mapsto_{\alpha}$ $\mathfrak{B} \rightharpoonup$ *CONST is-cn-tiny-smcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$

abbreviation *all-tiny-smcfs* :: $V \Rightarrow V$
  where *all-tiny-smcfs* $\alpha \equiv$ *set* $\{\mathfrak{F}.\ \exists \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}\}$

abbreviation *tiny-smcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *tiny-smcfs* α $\mathfrak{A}$ $\mathfrak{B} \equiv$ *set* $\{\mathfrak{F}.\ \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}\}$

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

Rules.

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

mk-ide rf *is-tiny-semifunctor-def*[*unfolded is-tiny-semifunctor-axioms-def*]
|*intro is-tiny-semifunctorI*|
|*dest is-tiny-semifunctorD*[*dest*]|
|*elim is-tiny-semifunctorE*[*elim*]|

lemmas [*smc-small-cs-intros*] = *is-tiny-semifunctorD*(1)

Elementary properties.

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

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

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

Further rules.

lemma *is-tiny-semifunctorI'*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and *tiny-semicategory* α $\mathfrak{A}$
and *tiny-semicategory* α $\mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}$
using *assms*
by
(
auto simp:
smc-cs-simps
smc-cs-intros
is-semifunctor.smcf-is-dghm

```
    is-tiny-dghm.intro
    is-tiny-semifunctorI
    tiny-semicategory.tiny-smc-tiny-digraph
  )

lemma is-tiny-semifunctorD':
  assumes 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅
  shows 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅
    and tiny-semicategory α 𝔄
    and tiny-semicategory α 𝔅
proof–
  interpret is-tiny-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(1))
  show 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅
    and tiny-semicategory α 𝔄
    and tiny-semicategory α 𝔅
    by (auto intro: smc-small-cs-intros)
qed

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

lemma is-tiny-semifunctorE':
  assumes 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅
  obtains 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅
    and tiny-semicategory α 𝔄
    and tiny-semicategory α 𝔅
  using is-tiny-semifunctorD'[OF assms] by auto

lemma is-tiny-semifunctor-iff:
  𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅 ⟷
    (𝔉 : 𝔄 ↦↦_{SMC α} 𝔅 ∧ tiny-semicategory α 𝔄 ∧ tiny-semicategory α 𝔅)
  by (auto intro: is-tiny-semifunctorI' dest: is-tiny-semifunctorD'(2,3))
```

Size.

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

lemma small-all-tiny-smcfs[simp]: small {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅}
proof(rule down)
  show
    {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅} ⊆
      elts (set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅})
  proof
    (
      simp only: elts-of-set small-all-smcfs if-True,
      rule subsetI,
      unfold mem-Collect-eq
```

```
    )
    fix 𝔉 assume ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅
    then obtain 𝔄 𝔅 where 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅 by clarsimp
    then interpret is-tiny-semifunctor α 𝔄 𝔅 𝔉 by simp
    show ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC α} 𝔅 using is-semifunctor-axioms by auto
  qed
qed

lemma tiny-smcfs-vsubset-Vset[simp]:
  set {𝔉. ∃𝔄 𝔅. 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅} ⊆∘ Vset α
proof(rule vsubsetI)
  fix 𝔉 assume 𝔉 ∈∘ all-tiny-smcfs α
  then obtain 𝔄 𝔅 where 𝔉: 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅 by clarsimp
  then show 𝔉 ∈∘ Vset α by (auto simp: is-tiny-semifunctor.tiny-smcf-in-Vset)
qed

lemma (in is-semifunctor) smcf-is-tiny-semifunctor-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔉 : 𝔄 ↦↦_{SMC.tiny β} 𝔅
proof(intro is-tiny-semifunctorI)
  show smcf-dghm 𝔉 : smc-dg 𝔄 ↦↦_{DG.tiny β} smc-dg 𝔅
    by
      (
        rule is-dghm.dghm-is-tiny-dghm-if-ge-Limit,
        rule smcf-is-dghm;
        intro assms
      )
qed (simp add: smcf-is-semifunctor-if-ge-Limit assms)
```

Opposite tiny semifunctor

```
lemma (in is-tiny-semifunctor) is-tiny-semifunctor-op:
  op-smcf 𝔉 : op-smc 𝔄 ↦↦_{SMC.tiny α} op-smc 𝔅
  by (intro is-tiny-semifunctorI')
    (cs-concl cs-shallow cs-intro: smc-small-cs-intros smc-op-intros)+

lemma (in is-tiny-semifunctor) is-tiny-semifunctor-op'[smc-op-intros]:
  assumes 𝔄' = op-smc 𝔄 and 𝔅' = op-smc 𝔅 and α' = α
  shows op-smcf 𝔉 : 𝔄' ↦↦_{SMC.tiny_{α'}} 𝔅'
  unfolding assms by (rule is-tiny-semifunctor-op)

lemmas is-tiny-semifunctor-op[smc-op-intros] =
  is-tiny-semifunctor.is-tiny-semifunctor-op'
```

Composition of tiny semifunctors

```
lemma smcf-comp-is-tiny-semifunctor[smc-small-cs-intros]:
  assumes 𝔊 : 𝔅 ↦↦_{SMC.tiny α} ℭ and 𝔉 : 𝔄 ↦↦_{SMC.tiny α} 𝔅
  shows 𝔊 ∘_{SMCF} 𝔉 : 𝔄 ↦↦_{SMC.tiny α} ℭ
proof–
  interpret 𝔉: is-tiny-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(2))
  interpret 𝔊: is-tiny-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(1))
  show ?thesis
    by (rule is-tiny-semifunctorI')
      (cs-concl cs-intro: smc-cs-intros smc-small-cs-intros)
qed
```

Tiny constant semifunctor

lemma *smcf-const-is-tiny-semifunctor*:
  assumes *tiny-semicategory* α $\mathfrak{C}$
    and *tiny-semicategory* α $\mathfrak{D}$
    and $f : a \mapsto_{\mathfrak{D}} a$
    and $f \circ_{A\mathfrak{D}} f = f$
  shows *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{C} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{D}$
proof(*intro is-tiny-semifunctorI'*)
  from *assms* **show** *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
    by (*cs-concl* **cs-simp**: *smc-cs-simps* **cs-intro**: *smc-small-cs-intros*)
qed (*auto simp*: *assms*(1,2))

lemma *smcf-const-is-tiny-semifunctor'*[*smc-small-cs-intros*]:
  assumes *tiny-semicategory* α $\mathfrak{C}$
    and *tiny-semicategory* α $\mathfrak{D}$
    and $f : a \mapsto_{\mathfrak{D}} a$
    and $f \circ_{A\mathfrak{D}} f = f$
    and $\mathfrak{C}' = \mathfrak{C}$
    and $\mathfrak{D}' = \mathfrak{D}$
  shows *smcf-const* $\mathfrak{C}$ $\mathfrak{D}$ a $f : \mathfrak{C}' \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{D}'$
  using *assms*(1–4) **unfolding** *assms*(5,6) **by** (*rule smcf-const-is-tiny-semifunctor*)

4.6 Natural transformation of a semifunctor

4.6.1 Background

named-theorems *ntsmcf-cs-simps*
named-theorems *ntsmcf-cs-intros*

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

Slicing

definition *ntsmcf-tdghm* :: $V \Rightarrow V$
  where *ntsmcf-tdghm* $\mathfrak{N}$ =
    [
      $\mathfrak{N}(\lfloor NTMap \rfloor)$,
      smcf-dghm $(\mathfrak{N}(\lfloor NTDom \rfloor))$,
      smcf-dghm $(\mathfrak{N}(\lfloor NTCod \rfloor))$,
      smc-dg $(\mathfrak{N}(\lfloor NTDGDom \rfloor))$,
      smc-dg $(\mathfrak{N}(\lfloor NTDGCod \rfloor))$
    ]$_\circ$

Components.

lemma *ntsmcf-tdghm-components*:
  shows [*slicing-simps*]: *ntsmcf-tdghm* $\mathfrak{N}(\lfloor NTMap \rfloor)$ = $\mathfrak{N}(\lfloor NTMap \rfloor)$
    and [*slicing-commute*]: *ntsmcf-tdghm* $\mathfrak{N}(\lfloor NTDom \rfloor)$ = *smcf-dghm* $(\mathfrak{N}(\lfloor NTDom \rfloor))$
    and [*slicing-commute*]: *ntsmcf-tdghm* $\mathfrak{N}(\lfloor NTCod \rfloor)$ = *smcf-dghm* $(\mathfrak{N}(\lfloor NTCod \rfloor))$
    and [*slicing-commute*]: *ntsmcf-tdghm* $\mathfrak{N}(\lfloor NTDGDom \rfloor)$ = *smc-dg* $(\mathfrak{N}(\lfloor NTDGDom \rfloor))$
    and [*slicing-commute*]: *ntsmcf-tdghm* $\mathfrak{N}(\lfloor NTDGCod \rfloor)$ = *smc-dg* $(\mathfrak{N}(\lfloor NTDGCod \rfloor))$
  unfolding *ntsmcf-tdghm-def nt-field-simps* **by** (*auto simp*: *nat-omega-simps*)

4.6.2 Definition and elementary properties

A natural transformation of semifunctors, as presented in this work, is a generalization of the concept of a natural transformation, as presented in Chapter I-4 in [39], to semicategories and semifunctors.

locale *is-ntsmcf* =
  $\mathcal{Z}$ α +
  vfsequence $\mathfrak{N}$ +
  NTDom: *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ +
  NTCod: *is-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$
  for α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ +
  assumes *ntsmcf-length*[*smc-cs-simps*]: *vcard* $\mathfrak{N}$ = $5_{\mathbb{N}}$
    and *ntsmcf-is-tdghm*[*slicing-intros*]: *ntsmcf-tdghm* $\mathfrak{N}$:
      smcf-dghm $\mathfrak{F}$ $\mapsto_{DGHM}$ *smcf-dghm* $\mathfrak{G}$: *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{B}$
    and *ntsmcf-NTDom*[*smc-cs-simps*]: $\mathfrak{N}(\lfloor NTDom \rfloor)$ = $\mathfrak{F}$
    and *ntsmcf-NTCod*[*smc-cs-simps*]: $\mathfrak{N}(\lfloor NTCod \rfloor)$ = $\mathfrak{G}$
    and *ntsmcf-NTDGDom*[*smc-cs-simps*]: $\mathfrak{N}(\lfloor NTDGDom \rfloor)$ = $\mathfrak{A}$
    and *ntsmcf-NTDGCod*[*smc-cs-simps*]: $\mathfrak{N}(\lfloor NTDGCod \rfloor)$ = $\mathfrak{B}$
    and *ntsmcf-Comp-commute*[*smc-cs-intros*]: $f : a \mapsto_{\mathfrak{A}} b \Longrightarrow$
      $\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{B}} \mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)$ = $\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{B}} \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$

syntax *-is-ntsmcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
  ($\langle$(- :/ - $\mapsto_{SMCF}$ - :/ - $\mapsto\!\!\mapsto_{SMC^1}$ -)$\rangle$ [51, 51, 51, 51, 51] 51)
syntax-consts *-is-ntsmcf* $\rightleftharpoons$ *is-ntsmcf*
translations $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{SMCF}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$
  CONST is-ntsmcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

abbreviation *all-ntsmcfs* :: $V \Rightarrow V$
  where *all-ntsmcfs* $\alpha \equiv$ *set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}\}$

abbreviation *ntsmcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *ntsmcfs* $\alpha\ \mathfrak{A}\ \mathfrak{B} \equiv$ *set* $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}\}$

abbreviation *these-ntsmcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
  where *these-ntsmcfs* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G} \equiv$ *set* $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}\}$

lemmas [*smc-cs-simps*] =
  is-ntsmcf.ntsmcf-length
  is-ntsmcf.ntsmcf-NTDom
  is-ntsmcf.ntsmcf-NTCod
  is-ntsmcf.ntsmcf-NTDGDom
  is-ntsmcf.ntsmcf-NTDGCod
  is-ntsmcf.ntsmcf-Comp-commute

lemmas [*smc-cs-intros*] = *is-ntsmcf.ntsmcf-Comp-commute*

lemma (**in** *is-ntsmcf*) *ntsmcf-is-tdghm'*:
  assumes $\mathfrak{F}' =$ *smcf-dghm* $\mathfrak{F}$
    and $\mathfrak{G}' =$ *smcf-dghm* $\mathfrak{G}$
    and $\mathfrak{A}' =$ *smc-dg* $\mathfrak{A}$
    and $\mathfrak{B}' =$ *smc-dg* $\mathfrak{B}$
  shows *ntsmcf-tdghm* $\mathfrak{N} : \mathfrak{F}' \mapsto_{DGHM} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{DG\alpha} \mathfrak{B}'$
  unfolding *assms*(1–4) **by** (*rule ntsmcf-is-tdghm*)

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

Rules.

lemma (**in** *is-ntsmcf*) *is-ntsmcf-axioms'*[*smc-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 $\mathfrak{N} : \mathfrak{F}' \mapsto_{SMCF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{SMC_{\alpha'}} \mathfrak{B}'$
  unfolding *assms* **by** (*rule is-ntsmcf-axioms*)

mk-ide rf *is-ntsmcf-def*[*unfolded is-ntsmcf-axioms-def*]
  |*intro is-ntsmcfI*|
  |*dest is-ntsmcfD*[*dest*]|
  |*elim is-ntsmcfE*[*elim*]|

lemmas [*smc-cs-intros*] =
  is-ntsmcfD(3,4)

lemma *is-ntsmcfI'*:
  assumes $\mathcal{Z}\ \alpha$
    and *vfsequence* $\mathfrak{N}$
    and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
    and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
    and *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
    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 $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
by (*intro is-ntsmcfI is-tdghmI*, *unfold ntsmcf-tdghm-components slicing-simps*)
(
simp-all add:
assms nat-omega-simps
ntsmcf-tdghm-def
is-semifunctorD(6)[*OF assms*(3)]
is-semifunctorD(6)[*OF assms*(4)]
)

lemma *is-ntsmcfD'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{N}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
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-ntsmcfD(2–11)[*OF assms*]
is-tdghmD[*OF is-ntsmcfD*(6)[*OF assms*], *unfolded slicing-simps*]
)

lemma *is-ntsmcfE'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
obtains $\mathcal{Z}\ \alpha$
and *vfsequence* $\mathfrak{N}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and *vcard* $\mathfrak{N} = 5_{\mathbb{N}}$
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-ntsmcfD'*)

Slicing.

context *is-ntsmcf*
begin

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

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

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

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

end

lemmas [*smc-cs-intros*] = *is-ntsmcf.ntsmcf-NTMap-is-arr'*

lemma (**in** *is-ntsmcf*) *ntsmcf-Comp-commute'*:
assumes $f : a \mapsto_{\mathfrak{A}} b$ **and** $g : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
shows
$\mathfrak{N}(\!|NTMap|\!)(\!|b|\!) \circ_{A\mathfrak{B}} (\mathfrak{F}(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{B}} g) =$
$(\mathfrak{G}(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{B}} \mathfrak{N}(\!|NTMap|\!)(\!|a|\!)) \circ_{A\mathfrak{B}} g$
using *assms*
by
(
cs-concl **cs-shallow**
cs-simp: *ntsmcf-Comp-commute semicategory.smc-Comp-assoc*[*symmetric*]
cs-intro: *smc-cs-intros*
)

lemma (**in** *is-ntsmcf*) *ntsmcf-Comp-commute''*:
assumes $f : a \mapsto_{\mathfrak{A}} b$ **and** $g : c \mapsto_{\mathfrak{B}} \mathfrak{F}(\!|ObjMap|\!)(\!|a|\!)$
shows
$\mathfrak{G}(\!|ArrMap|\!)(\!|f|\!) \circ_{A\mathfrak{B}} (\mathfrak{N}(\!|NTMap|\!)(\!|a|\!) \circ_{A\mathfrak{B}} g) =$
$(\mathfrak{N}(\!|NTMap|\!)(\!|b|\!) \circ_{A\mathfrak{B}} \mathfrak{F}(\!|ArrMap|\!)(\!|f|\!)) \circ_{A\mathfrak{B}} g$
using *assms*
by
(
cs-concl
cs-simp: *ntsmcf-Comp-commute semicategory.smc-Comp-assoc*[*symmetric*]
cs-intro: *smc-cs-intros*
)

Elementary properties.

lemma *ntsmcf-eqI*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{N}' : \mathfrak{F}' \mapsto_{SMCF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{SMC\alpha} \mathfrak{B}'$
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-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret *R*: *is-ntsmcf* α $\mathfrak{A}'$ $\mathfrak{B}'$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{N}'$ **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**: *smc-cs-simps V-cs-simps*)
show $\mathcal{D}_\circ\ \mathfrak{N} = \mathcal{D}_\circ\ \mathfrak{N}'$
by (*cs-concl* **cs-shallow cs-simp**: *smc-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*: *smc-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 *ntsmcf-tdghm-eqI*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto\mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{N}' : \mathfrak{F}' \mapsto\mapsto_{SMCF} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{SMC\alpha} \mathfrak{B}'$
and $\mathfrak{F} = \mathfrak{F}'$
and $\mathfrak{G} = \mathfrak{G}'$
and $\mathfrak{A} = \mathfrak{A}'$
and $\mathfrak{B} = \mathfrak{B}'$
and *ntsmcf-tdghm* $\mathfrak{N}$ = *ntsmcf-tdghm* $\mathfrak{N}'$
shows $\mathfrak{N} = \mathfrak{N}'$
proof(*rule ntsmcf-eqI*[*of* α])
from *assms*(7) **have** *ntsmcf-tdghm* $\mathfrak{N}(\!|NTMap|\!)$ = *ntsmcf-tdghm* $\mathfrak{N}'(\!|NTMap|\!)$ **by** *simp*
then show $\mathfrak{N}(\!|NTMap|\!) = \mathfrak{N}'(\!|NTMap|\!)$ **unfolding** *slicing-simps* **by** *simp-all*
from *assms*(3–6) **show** $\mathfrak{F} = \mathfrak{F}'$ $\mathfrak{G} = \mathfrak{G}'$ $\mathfrak{A} = \mathfrak{A}'$ $\mathfrak{B} = \mathfrak{B}'$ **by** *simp-all*
qed (*simp-all add*: *assms*(1,2))

lemma (**in** *is-ntsmcf*) *ntsmcf-def*:
$\mathfrak{N} = [\mathfrak{N}(\!|NTMap|\!), \mathfrak{N}(\!|NTDom|\!), \mathfrak{N}(\!|NTCod|\!), \mathfrak{N}(\!|NTDGDom|\!), \mathfrak{N}(\!|NTDGCod|\!)]_\circ$
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ\ \mathfrak{N} = 5_{\mathbb{N}}$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps V-cs-simps*)
have *dom-rhs*:
$\mathcal{D}_\circ\ [\mathfrak{N}(\!|NTMap|\!), \mathfrak{N}(\!|NTDGDom|\!), \mathfrak{N}(\!|NTDGCod|\!), \mathfrak{N}(\!|NTDom|\!), \mathfrak{N}(\!|NTCod|\!)]_\circ = 5_{\mathbb{N}}$
by (*simp add*: *nat-omega-simps*)
then show $\mathcal{D}_\circ\ \mathfrak{N} = \mathcal{D}_\circ\ [\mathfrak{N}(\!|NTMap|\!), \mathfrak{N}(\!|NTDom|\!), \mathfrak{N}(\!|NTCod|\!), \mathfrak{N}(\!|NTDGDom|\!), \mathfrak{N}(\!|NTDGCod|\!)]_\circ$
unfolding *dom-lhs dom-rhs* **by** (*simp add*: *nat-omega-simps*)
show $a \in_\circ \mathcal{D}_\circ\ \mathfrak{N} \Longrightarrow$
$\mathfrak{N}(\!|a|\!) = [\mathfrak{N}(\!|NTMap|\!), \mathfrak{N}(\!|NTDom|\!), \mathfrak{N}(\!|NTCod|\!), \mathfrak{N}(\!|NTDGDom|\!), \mathfrak{N}(\!|NTDGCod|\!)]_\circ(\!|a|\!)$
for *a*
by (*unfold dom-lhs*, *elim-in-numeral*, *unfold nt-field-simps*)
(*simp-all add*: *nat-omega-simps*)
qed (*auto simp*: *vsv-axioms*)

Size.

lemma (**in** *is-ntsmcf*) *ntsmcf-in-Vset*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows $\mathfrak{N} \in_\circ$ *Vset* β
proof–
interpret β: $\mathcal{Z}\ \beta$ **by** (*rule assms*(1))

```
  note [smc-cs-intros] =
    ntsmcf-NTMap-in-Vset
    NTDom.smcf-in-Vset
    NTCod.smcf-in-Vset
    NTDom.HomDom.smc-in-Vset
    NTDom.HomCod.smc-in-Vset
  from assms(2) show ?thesis
    by (subst ntsmcf-def)
      (
        cs-concl cs-shallow
          cs-simp: smc-cs-simps cs-intro: smc-cs-intros V-cs-intros
      )
qed

lemma (in is-ntsmcf) ntsmcf-is-ntsmcf-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMC β 𝔅
proof(intro is-ntsmcfI )
  show ntsmcf-tdghm 𝔑 :
    smcf-dghm 𝔉 ↦_DGHM smcf-dghm 𝔊 : smc-dg 𝔄 ↦↦_DG β smc-dg 𝔅
    by (rule is-tdghm.tdghm-is-tdghm-if-ge-Limit[OF ntsmcf-is-tdghm assms])
  show 𝔑(⁅NTMap⁆)(⁅b⁆) ∘_A𝔅 𝔉(⁅ArrMap⁆)(⁅f⁆) = 𝔊(⁅ArrMap⁆)(⁅f⁆) ∘_A𝔅 𝔑(⁅NTMap⁆)(⁅a⁆)
    if f : a ↦_𝔄 b for f a b
    by
      (
        use that in
          ‹cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros›
      )+
qed
  (
    cs-concl cs-shallow
      cs-simp: smc-cs-simps
      cs-intro:
        smc-cs-intros
        V-cs-intros
        assms
        NTDom.smcf-is-semifunctor-if-ge-Limit
        NTCod.smcf-is-semifunctor-if-ge-Limit
   )+

lemma small-all-ntsmcfs[simp]:
  small {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅}
proof(cases ‹𝒵 α›)
  case True
  from is-ntsmcf.ntsmcf-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 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅} = {} by auto
  then show ?thesis by simp
qed

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

lemma small-these-ntcfs[simp]: small {𝔑. 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅}
```

by (*rule down*[*of* - ‹*set* {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦$_{SMCF}$ 𝔊 : 𝔄 ↦↦$_{SMC\alpha}$ 𝔅}›])
auto

Further elementary results.

lemma *these-ntsmcfs-iff*:
𝔑 ∈$_\circ$ *these-ntsmcfs* α 𝔄 𝔅 𝔉 𝔊 ⟷ 𝔑 : 𝔉 ↦$_{SMCF}$ 𝔊 : 𝔄 ↦↦$_{SMC\alpha}$ 𝔅
by *auto*

4.6.3 Opposite natural transformation of semifunctors

Definition and elementary properties

See section 1.5 in [15].

definition *op-ntsmcf* :: *V* ⇒ *V*
where *op-ntsmcf* 𝔑 =
[
𝔑(|*NTMap*|),
op-smcf (𝔑(|*NTCod*|)),
op-smcf (𝔑(|*NTDom*|)),
op-smc (𝔑(|*NTDGDom*|)),
op-smc (𝔑(|*NTDGCod*|))
]$_\circ$

Components.

lemma *op-ntsmcf-components*[*smc-op-simps*]:
shows *op-ntsmcf* 𝔑(|*NTMap*|) = 𝔑(|*NTMap*|)
and *op-ntsmcf* 𝔑(|*NTDom*|) = *op-smcf* (𝔑(|*NTCod*|))
and *op-ntsmcf* 𝔑(|*NTCod*|) = *op-smcf* (𝔑(|*NTDom*|))
and *op-ntsmcf* 𝔑(|*NTDGDom*|) = *op-smc* (𝔑(|*NTDGDom*|))
and *op-ntsmcf* 𝔑(|*NTDGCod*|) = *op-smc* (𝔑(|*NTDGCod*|))
unfolding *op-ntsmcf-def nt-field-simps* **by** (*auto simp*: *nat-omega-simps*)

Slicing.

lemma *op-tdghm-ntsmcf-tdghm*[*slicing-commute*]:
op-tdghm (*ntsmcf-tdghm* 𝔑) = *ntsmcf-tdghm* (*op-ntsmcf* 𝔑)
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ$ (*op-tdghm* (*ntsmcf-tdghm* 𝔑)) = $5_{\mathbb{N}}$
unfolding *op-tdghm-def* **by** (*auto simp*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*ntsmcf-tdghm* (*op-ntsmcf* 𝔑)) = $5_{\mathbb{N}}$
unfolding *ntsmcf-tdghm-def* **by** (*auto simp*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*op-tdghm* (*ntsmcf-tdghm* 𝔑)) = $\mathcal{D}_\circ$ (*ntsmcf-tdghm* (*op-ntsmcf* 𝔑))
unfolding *dom-lhs dom-rhs* **by** *simp*
show *a* ∈$_\circ$ $\mathcal{D}_\circ$ (*op-tdghm* (*ntsmcf-tdghm* 𝔑)) ⟹
op-tdghm (*ntsmcf-tdghm* 𝔑)(|*a*|) = *ntsmcf-tdghm* (*op-ntsmcf* 𝔑)(|*a*|)
for *a*
by
(
unfold dom-lhs,
elim-in-numeral,
unfold ntsmcf-tdghm-def op-ntsmcf-def op-tdghm-def nt-field-simps
)
(*auto simp*: *nat-omega-simps slicing-commute*[*symmetric*])
qed (*auto simp*: *ntsmcf-tdghm-def op-tdghm-def*)

Further properties

lemma (**in** *is-ntsmcf*) *is-ntsmcf-op*:

```
  op-ntsmcf 𝔑 : op-smcf 𝔊 ↦_SMCF op-smcf 𝔉 : op-smc 𝔄 ↦↦_SMCα op-smc 𝔅
proof(rule is-ntsmcfI, unfold smc-op-simps)
  show vfsequence (op-ntsmcf 𝔑) by (simp add: op-ntsmcf-def)
  show vcard (op-ntsmcf 𝔑) = 5_ℕ by (simp add: op-ntsmcf-def nat-omega-simps)
  fix f a b assume f : b ↦_𝔄 a
  with is-ntsmcf-axioms show
    𝔑(|NTMap|)(|b|) ∘_A op-smc 𝔅 𝔊(|ArrMap|)(|f|) =
      𝔉(|ArrMap|)(|f|) ∘_A op-smc 𝔅 𝔑(|NTMap|)(|a|)
    by (cs-concl cs-simp: smc-cs-simps smc-op-simps cs-intro: smc-cs-intros)
qed
  (
    insert is-ntsmcf-axioms,
    (
      cs-concl cs-shallow
        cs-simp: smc-cs-simps slicing-commute[symmetric]
        cs-intro: smc-cs-intros smc-op-intros dg-op-intros slicing-intros
    )+
  )

lemma (in is-ntsmcf) is-ntsmcf-op'[smc-op-intros]:
  assumes 𝔊' = op-smcf 𝔊
    and 𝔉' = op-smcf 𝔉
    and 𝔄' = op-smc 𝔄
    and 𝔅' = op-smc 𝔅
  shows op-ntsmcf 𝔑 : 𝔊' ↦_SMCF 𝔉' : 𝔄' ↦↦_SMCα 𝔅'
  unfolding assms by (rule is-ntsmcf-op)

lemmas [smc-op-intros] = is-ntsmcf.is-ntsmcf-op'

lemma (in is-ntsmcf) ntsmcf-op-ntsmcf-op-ntsmcf[smc-op-simps]:
  op-ntsmcf (op-ntsmcf 𝔑) = 𝔑
proof(rule ntsmcf-eqI[of α 𝔄 𝔅 𝔉 𝔊 - 𝔄 𝔅 𝔉 𝔊], unfold smc-op-simps)
  interpret op:
    is-ntsmcf α ‹op-smc 𝔄› ‹op-smc 𝔅› ‹op-smcf 𝔊› ‹op-smcf 𝔉› ‹op-ntsmcf 𝔑›
    by (rule is-ntsmcf-op)
  from op.is-ntsmcf-op show
    op-ntsmcf (op-ntsmcf 𝔑) : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    by (simp add: smc-op-simps)
qed (auto simp: smc-cs-intros)

lemmas ntsmcf-op-ntsmcf-op-ntsmcf[smc-op-simps] =
  is-ntsmcf.ntsmcf-op-ntsmcf-op-ntsmcf

lemma eq-op-ntsmcf-iff:
  assumes 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑' : 𝔉' ↦_SMCF 𝔊' : 𝔄' ↦↦_SMCα 𝔅'
  shows op-ntsmcf 𝔑 = op-ntsmcf 𝔑' ⟷ 𝔑 = 𝔑'
proof
  interpret L: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
  interpret R: is-ntsmcf α 𝔄' 𝔅' 𝔉' 𝔊' 𝔑' by (rule assms(2))
  assume prems: op-ntsmcf 𝔑 = op-ntsmcf 𝔑'
  show 𝔑 = 𝔑'
  proof(rule ntsmcf-eqI[OF assms])
    from prems L.ntsmcf-op-ntsmcf-op-ntsmcf R.ntsmcf-op-ntsmcf-op-ntsmcf show
      𝔑(|NTMap|) = 𝔑'(|NTMap|)
      by metis+
    from prems L.ntsmcf-op-ntsmcf-op-ntsmcf R.ntsmcf-op-ntsmcf-op-ntsmcf
    have 𝔑(|NTDom|) = 𝔑'(|NTDom|)
```

```
    and 𝔑(|NTCod|) = 𝔑'(|NTCod|)
    and 𝔑(|NTDGDom|) = 𝔑'(|NTDGDom|)
    and 𝔑(|NTDGCod|) = 𝔑'(|NTDGCod|)
    by metis+
  then show 𝔉 = 𝔉' 𝔊 = 𝔊' 𝔄 = 𝔄' 𝔅 = 𝔅' by (auto simp: smc-cs-simps)
 qed
qed auto
```

4.6.4 Vertical composition of natural transformations

Definition and elementary properties

See Chapter II-4 in [39].

```
definition ntsmcf-vcomp :: V ⇒ V ⇒ V (infixl ‹∘NTSMCF› 55)
 where ntsmcf-vcomp 𝔐 𝔑 =
  [
   (λa∈ₒ𝔑(|NTDGDom|)(|Obj|). (𝔐(|NTMap|)(|a|)) ∘A𝔑(|NTDGCod|) (𝔑(|NTMap|)(|a|))),
   𝔑(|NTDom|),
   𝔐(|NTCod|),
   𝔑(|NTDGDom|),
   𝔐(|NTDGCod|)
  ]ₒ
```

Components.

```
lemma ntsmcf-vcomp-components:
 shows
  (𝔐 ∘NTSMCF 𝔑)(|NTMap|) =
   (λa∈ₒ𝔑(|NTDGDom|)(|Obj|). (𝔐(|NTMap|)(|a|)) ∘A𝔑(|NTDGCod|) (𝔑(|NTMap|)(|a|)))
  and [dg-shared-cs-simps, smc-cs-simps]: (𝔐 ∘NTSMCF 𝔑)(|NTDom|) = 𝔑(|NTDom|)
  and [dg-shared-cs-simps, smc-cs-simps]: (𝔐 ∘NTSMCF 𝔑)(|NTCod|) = 𝔐(|NTCod|)
  and [dg-shared-cs-simps, smc-cs-simps]:
   (𝔐 ∘NTSMCF 𝔑)(|NTDGDom|) = 𝔑(|NTDGDom|)
  and [dg-shared-cs-simps, smc-cs-simps]:
   (𝔐 ∘NTSMCF 𝔑)(|NTDGCod|) = 𝔐(|NTDGCod|)
 unfolding nt-field-simps ntsmcf-vcomp-def by (simp-all add: nat-omega-simps)
```

Natural transformation map

```
lemma ntsmcf-vcomp-NTMap-vsv[dg-shared-cs-intros, smc-cs-intros]:
 vsv ((𝔐 ∘NTSMCF 𝔑)(|NTMap|))
 unfolding ntsmcf-vcomp-components by simp

lemma ntsmcf-vcomp-NTMap-vdomain[smc-cs-simps]:
 assumes 𝔑 : 𝔉 ↦SMCF 𝔊 : 𝔄 ↦↦SMCα 𝔅
 shows 𝒟ₒ ((𝔐 ∘NTSMCF 𝔑)(|NTMap|)) = 𝔄(|Obj|)
proof–
 interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 using assms by auto
 show ?thesis unfolding ntsmcf-vcomp-components by (simp add: smc-cs-simps)
qed

lemma ntsmcf-vcomp-NTMap-app[smc-cs-simps]:
 assumes 𝔐 : 𝔊 ↦SMCF ℌ : 𝔄 ↦↦SMCα 𝔅
   and 𝔑 : 𝔉 ↦SMCF 𝔊 : 𝔄 ↦↦SMCα 𝔅
   and a ∈ₒ 𝔄(|Obj|)
 shows (𝔐 ∘NTSMCF 𝔑)(|NTMap|)(|a|) = 𝔐(|NTMap|)(|a|) ∘A𝔅 𝔑(|NTMap|)(|a|)
proof–
 interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔊 ℌ 𝔐 using assms by auto
```

```
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 using assms by auto
  from assms show ?thesis
    unfolding ntsmcf-vcomp-components by (simp add: smc-cs-simps)
qed

lemma ntsmcf-vcomp-NTMap-vrange:
  assumes 𝔐 : 𝔊 ↦_SMCF ℌ : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
  shows ℛ∘ ((𝔐 ∙_NTSMCF 𝔑)(|NTMap|)) ⊆∘ 𝔅(|Arr|)
  unfolding ntsmcf-vcomp-components
proof(rule vrange-VLambda-vsubset)
  fix x assume prems: x ∈∘ 𝔑(|NTDGDom|)(|Obj|)
  from prems assms show 𝔐(|NTMap|)(|x|) ∘_A𝔑(|NTDGCod|) 𝔑(|NTMap|)(|x|) ∈∘ 𝔅(|Arr|)
    by (cs-prems cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
      (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
qed
```

Further properties

```
lemma ntsmcf-vcomp-composable-commute[smc-cs-simps]:
  — See Chapter II-4 in [39]).
  assumes 𝔐 : 𝔊 ↦_SMCF ℌ : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and f : a ↦_𝔄 b
  shows
    (𝔐(|NTMap|)(|b|) ∘_A𝔅 𝔑(|NTMap|)(|b|)) ∘_A𝔅 𝔉(|ArrMap|)(|f|) =
      ℌ(|ArrMap|)(|f|) ∘_A𝔅 (𝔐(|NTMap|)(|a|) ∘_A𝔅 𝔑(|NTMap|)(|a|))
    (is ‹(?MC ∘_A𝔅 ?NC) ∘_A𝔅 ?R = ?T ∘_A𝔅 (?MD ∘_A𝔅 ?ND)›)
proof–
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔊 ℌ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  from assms show ?thesis
    by (intro 𝔐.NTDom.HomCod.smc-pattern-rectangle-left)
      (cs-concl cs-intro: smc-cs-intros cs-simp: 𝔑.ntsmcf-Comp-commute)
qed

lemma ntsmcf-vcomp-is-ntsmcf[smc-cs-intros]:
  — See Chapter II-4 in [39].
  assumes 𝔐 : 𝔊 ↦_SMCF ℌ : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
  shows 𝔐 ∙_NTSMCF 𝔑 : 𝔉 ↦_SMCF ℌ : 𝔄 ↦↦_SMCα 𝔅
proof–
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔊 ℌ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  show ?thesis
  proof(intro is-ntsmcfI')
    show vfsequence (𝔐 ∙_NTSMCF 𝔑) by (simp add: ntsmcf-vcomp-def)
    show vcard (𝔐 ∙_NTSMCF 𝔑) = 5_ℕ
      by (auto simp: nat-omega-simps ntsmcf-vcomp-def)
    show vsv ((𝔐 ∙_NTSMCF 𝔑)(|NTMap|))
      unfolding ntsmcf-vcomp-components by simp
    from assms show (𝔐 ∙_NTSMCF 𝔑)(|NTMap|)(|a|) : 𝔉(|ObjMap|)(|a|) ↦_𝔅 ℌ(|ObjMap|)(|a|)
      if a ∈∘ 𝔄(|Obj|) for a
      by
        (
          use that in
            ‹cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros›
        )
```

```
    fix f a b assume f : a ↦_𝔄 b
    with assms show
      (𝔐 ∘_NTSMCF 𝔑)(⦇NTMap⦈)(⦇b⦈) ∘_A𝔅 𝔉(⦇ArrMap⦈)(⦇f⦈) =
        𝔊(⦇ArrMap⦈)(⦇f⦈) ∘_A𝔅 (𝔐 ∘_NTSMCF 𝔑)(⦇NTMap⦈)(⦇a⦈)
      by
        (
          cs-concl
            cs-simp: smc-cs-simps is-ntsmcf.ntsmcf-Comp-commute′
            cs-intro: smc-cs-intros
        )
  qed (use assms in ‹auto simp: smc-cs-simps ntsmcf-vcomp-NTMap-vrange›)
qed

lemma ntsmcf-vcomp-assoc[smc-cs-simps]:
  — See Chapter II-4 in [39].
  assumes 𝔏 : 𝔊 ↦_SMCF 𝔎 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔐 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑 : 𝔈 ↦_SMCF 𝔉 : 𝔄 ↦↦_SMCα 𝔅
  shows (𝔏 ∘_NTSMCF 𝔐) ∘_NTSMCF 𝔑 = 𝔏 ∘_NTSMCF (𝔐 ∘_NTSMCF 𝔑)
proof–
  interpret 𝔏: is-ntsmcf α 𝔄 𝔅 𝔊 𝔎 𝔏 by (rule assms(1))
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔈 𝔉 𝔑 by (rule assms(3))
  show ?thesis
  proof(rule ntsmcf-eqI[of α])
    show ((𝔏 ∘_NTSMCF 𝔐) ∘_NTSMCF 𝔑)(⦇NTMap⦈) = (𝔏 ∘_NTSMCF (𝔐 ∘_NTSMCF 𝔑))(⦇NTMap⦈)
    proof(rule vsv-eqI)
      fix a assume a ∈_∘ 𝒟_∘ ((𝔏 ∘_NTSMCF 𝔐 ∘_NTSMCF 𝔑)(⦇NTMap⦈))
      then have a ∈_∘ 𝔄(⦇Obj⦈)
        unfolding ntsmcf-vcomp-components by (simp add: smc-cs-simps)
      with assms show
        ((𝔏 ∘_NTSMCF 𝔐) ∘_NTSMCF 𝔑)(⦇NTMap⦈)(⦇a⦈) =
          (𝔏 ∘_NTSMCF (𝔐 ∘_NTSMCF 𝔑))(⦇NTMap⦈)(⦇a⦈)
        by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
    qed (simp-all add: ntsmcf-vcomp-components)
  qed (auto intro: smc-cs-intros)
qed
```

Opposite of the vertical composition of natural transformations of semifunctors

```
lemma op-ntsmcf-ntsmcf-vcomp[smc-op-simps]:
  assumes 𝔐 : 𝔊 ↦_SMCF 𝔎 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
  shows op-ntsmcf (𝔐 ∘_NTSMCF 𝔑) = op-ntsmcf 𝔑 ∘_NTSMCF op-ntsmcf 𝔐
proof–
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔊 𝔎 𝔐 using assms(1) by auto
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 using assms(2) by auto
  show ?thesis
  proof(rule ntsmcf-eqI[of α]; (intro symmetric)?)
    show op-ntsmcf (𝔐 ∘_NTSMCF 𝔑)(⦇NTMap⦈) =
      (op-ntsmcf 𝔑 ∘_NTSMCF op-ntsmcf 𝔐)(⦇NTMap⦈)
    proof(rule vsv-eqI)
      fix a assume a ∈_∘ 𝒟_∘ (op-ntsmcf (𝔐 ∘_NTSMCF 𝔑)(⦇NTMap⦈))
      then have a: a ∈_∘ 𝔄(⦇Obj⦈)
        unfolding smc-op-simps ntsmcf-vcomp-NTMap-vdomain[OF assms(2)] by simp
      with
        𝔐.NTDom.HomCod.op-smc-Comp
        𝔐.ntsmcf-NTMap-is-arr[OF a]
```

```
    𝔑.ntsmcf-NTMap-is-arr[OF a]
  show op-ntsmcf (𝔐 ∘_NTSMCF 𝔑)(|NTMap|)(|a|) =
    (op-ntsmcf 𝔑 ∘_NTSMCF op-ntsmcf 𝔐)(|NTMap|)(|a|)
    unfolding smc-op-simps ntsmcf-vcomp-components
    by (simp add: smc-cs-simps)
 qed (simp-all add: smc-op-simps smc-cs-simps ntsmcf-vcomp-components(1))
qed (auto intro: smc-cs-intros smc-op-intros)
qed
```

4.6.5 Horizontal composition of natural transformations

Definition and elementary properties

See Chapter II-5 in [39].

```
definition ntsmcf-hcomp :: V ⇒ V ⇒ V (infixl ‹∘_NTSMCF› 55)
  where ntsmcf-hcomp 𝔐 𝔑 =
    [
      (
        λa∈∘𝔑(|NTDGDom|)(|Obj|).
          (
            𝔐(|NTCod|)(|ArrMap|)(𝔑(|NTMap|)(|a|)) ∘_A𝔐(|NTDGCod|)
            𝔐(|NTMap|)(|𝔑(|NTDom|)(|ObjMap|)(|a|))
          )
      ),
      (𝔐(|NTDom|) ∘_SMCF 𝔑(|NTDom|)),
      (𝔐(|NTCod|) ∘_SMCF 𝔑(|NTCod|)),
      (𝔑(|NTDGDom|)),
      (𝔐(|NTDGCod|))
    ]∘
```

Components.

```
lemma ntsmcf-hcomp-components:
  shows
    (𝔐 ∘_NTSMCF 𝔑)(|NTMap|) =
      (
        λa∈∘𝔑(|NTDGDom|)(|Obj|).
          (
            𝔐(|NTCod|)(|ArrMap|)(𝔑(|NTMap|)(|a|)) ∘_A𝔐(|NTDGCod|)
            𝔐(|NTMap|)(|𝔑(|NTDom|)(|ObjMap|)(|a|))
          )
      )
    and [dg-shared-cs-simps, smc-cs-simps]:
      (𝔐 ∘_NTSMCF 𝔑)(|NTDom|) = 𝔐(|NTDom|) ∘_SMCF 𝔑(|NTDom|)
    and [dg-shared-cs-simps, smc-cs-simps]:
      (𝔐 ∘_NTSMCF 𝔑)(|NTCod|) = 𝔐(|NTCod|) ∘_SMCF 𝔑(|NTCod|)
    and [dg-shared-cs-simps, smc-cs-simps]:
      (𝔐 ∘_NTSMCF 𝔑)(|NTDGDom|) = 𝔑(|NTDGDom|)
    and [dg-shared-cs-simps, smc-cs-simps]:
      (𝔐 ∘_NTSMCF 𝔑)(|NTDGCod|) = 𝔐(|NTDGCod|)
  unfolding nt-field-simps ntsmcf-hcomp-def by (auto simp: nat-omega-simps)
```

Natural transformation map

```
lemma ntsmcf-hcomp-NTMap-vsv[smc-cs-intros]: vsv ((𝔐 ∘_NTSMCF 𝔑)(|NTMap|))
  unfolding ntsmcf-hcomp-components by auto

lemma ntsmcf-hcomp-NTMap-vdomain[smc-cs-simps]:
```

assumes $\mathfrak{N} : \mathfrak{F} \mapsto\mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathcal{D}_\circ\ ((\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
proof–
interpret $\mathfrak{N}$: *is-ntsmcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(1))
show *?thesis* **unfolding** *ntsmcf-hcomp-components* **by** (*simp add*: *smc-cs-simps*)
qed

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

lemma *ntsmcf-hcomp-NTMap-vrange*:
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto\mapsto_{SMCF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto\mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ\ ((\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)) \subseteq_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
proof
interpret $\mathfrak{M}$: *is-ntsmcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}'\ \mathfrak{G}'\ \mathfrak{M}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-ntsmcf* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(2))
fix f **assume** $f \in_\circ \mathcal{R}_\circ\ ((\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor))$
with *ntsmcf-hcomp-NTMap-vdomain* **obtain** a
where a: $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$ **and** *f-def*: $f = (\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
unfolding *ntsmcf-hcomp-components* **by** (*force simp*: *smc-cs-simps*)
have $\mathfrak{F}a$: $\mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \in_\circ \mathfrak{B}(\lfloor Obj \rfloor)$
by (*simp add*: $\mathfrak{N}$.*NTDom.smcf-ObjMap-app-in-HomCod-Obj a*)
from $\mathfrak{N}$.*ntsmcf-NTMap-is-arr*[*OF a*] **have** $\mathfrak{G}'(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) \rfloor) :$
$\mathfrak{G}'(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor) \mapsto_{\mathfrak{C}} \mathfrak{G}'(\lfloor ObjMap \rfloor)(\lfloor \mathfrak{G}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor)$
by (*force intro*: *smc-cs-intros*)
then have $\mathfrak{G}'(\lfloor ArrMap \rfloor)(\lfloor \mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor) \rfloor) \circ_{A\mathfrak{C}} \mathfrak{M}(\lfloor NTMap \rfloor)(\lfloor \mathfrak{F}(\lfloor ObjMap \rfloor)(\lfloor a \rfloor) \rfloor) \in_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
by
(
meson
$\mathfrak{M}$.*ntsmcf-NTMap-is-arr*[*OF* $\mathfrak{F}a$]
$\mathfrak{M}$.*NTDom.HomCod.smc-is-arrE*
$\mathfrak{M}$.*NTDom.HomCod.smc-Comp-is-arr*
)
with a **show** $f \in_\circ \mathfrak{C}(\lfloor Arr \rfloor)$
unfolding *f-def ntsmcf-hcomp-components* **by** (*simp add*: *smc-cs-simps*)
qed

Further properties

lemma *ntsmcf-hcomp-composable-commute*:
— See Chapter II-5 in [39].
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto\mapsto_{SMCF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto\mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $f : a \mapsto_{\mathfrak{A}} b$
shows
$(\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{F}' \circ_{SMCF} \mathfrak{F})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) =$
$(\mathfrak{G}' \circ_{SMCF} \mathfrak{G})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$

```
proof–
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 ℭ 𝔉′ 𝔊′ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  from assms(3) have [simp]: b ∈∘ 𝔄⦇Obj⦈ and a: a ∈∘ 𝔄⦇Obj⦈ by auto
  from 𝔐.is-ntsmcf-axioms 𝔑.is-ntsmcf-axioms have 𝔐𝔑b:
    (𝔐 ∘NTSMCF 𝔑)⦇NTMap⦈⦇b⦈ =
      (𝔊′⦇ArrMap⦈⦇𝔑⦇NTMap⦈⦇b⦈⦈) ∘A ℭ (𝔐⦇NTMap⦈⦇𝔉⦇ObjMap⦈⦇b⦈⦈)
    by (auto simp: smc-cs-simps)
  let ?𝔊′𝔉f = ‹𝔊′⦇ArrMap⦈⦇𝔉⦇ArrMap⦈⦇f⦈⦈›
  from a 𝔐.is-ntsmcf-axioms 𝔑.is-ntsmcf-axioms have 𝔐𝔑a:
    (𝔐 ∘NTSMCF 𝔑)⦇NTMap⦈⦇a⦈ =
      𝔊′⦇ArrMap⦈⦇𝔑⦇NTMap⦈⦇a⦈⦈ ∘A ℭ 𝔐⦇NTMap⦈⦇𝔉⦇ObjMap⦈⦇a⦈⦈
    by (cs-concl cs-shallow cs-simp: smc-cs-simps)+
  note 𝔐.NTCod.smcf-ArrMap-Comp[smc-cs-simps del]
  from assms show ?thesis
    unfolding 𝔐𝔑b 𝔐𝔑a
    by (intro 𝔐.NTDom.HomCod.smc-pattern-rectangle-left)
      (
        cs-concl
          cs-simp: smc-cs-simps is-semifunctor.smcf-ArrMap-Comp[symmetric]
          cs-intro: smc-cs-intros
      )+
qed

lemma ntsmcf-hcomp-is-ntsmcf:
  — See Chapter II-5 in [39].
  assumes 𝔐 : 𝔉′ ↦SMCF 𝔊′ : 𝔅 ↦↦SMCα ℭ
    and 𝔑 : 𝔉 ↦SMCF 𝔊 : 𝔄 ↦↦SMCα 𝔅
  shows 𝔐 ∘NTSMCF 𝔑 : 𝔉′ ∘SMCF 𝔉 ↦SMCF 𝔊′ ∘SMCF 𝔊 : 𝔄 ↦↦SMCα ℭ
proof–
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 ℭ 𝔉′ 𝔊′ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  show ?thesis
  proof(intro is-ntsmcfI′, unfold ntsmcf-hcomp-components(3,4))
    show vfsequence (𝔐 ∘NTSMCF 𝔑) unfolding ntsmcf-hcomp-def by auto
    show vcard (𝔐 ∘NTSMCF 𝔑) = 5ℕ
      unfolding ntsmcf-hcomp-def by (simp add: nat-omega-simps)
    from assms show (𝔐 ∘NTSMCF 𝔑)⦇NTMap⦈⦇a⦈ :
      (𝔉′ ∘SMCF 𝔉)⦇ObjMap⦈⦇a⦈ ↦ℭ (𝔊′ ∘SMCF 𝔊)⦇ObjMap⦈⦇a⦈
      if a ∈∘ 𝔄⦇Obj⦈ for a
      by
        (
          use that in
            ‹cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros›
        )
    fix f a b assume f : a ↦𝔄 b
    with ntsmcf-hcomp-composable-commute[OF assms]
    show (𝔐 ∘NTSMCF 𝔑)⦇NTMap⦈⦇b⦈ ∘A ℭ (𝔉′ ∘SMCF 𝔉)⦇ArrMap⦈⦇f⦈ =
      (𝔊′ ∘SMCF 𝔊)⦇ArrMap⦈⦇f⦈ ∘A ℭ (𝔐 ∘NTSMCF 𝔑)⦇NTMap⦈⦇a⦈
      by auto
  qed (auto simp: ntsmcf-hcomp-components(1) smc-cs-simps intro: smc-cs-intros)
qed

lemma ntsmcf-hcomp-is-ntsmcf′[smc-cs-intros]:
  assumes 𝔐 : 𝔉′ ↦SMCF 𝔊′ : 𝔅 ↦↦SMCα ℭ
    and 𝔑 : 𝔉 ↦SMCF 𝔊 : 𝔄 ↦↦SMCα 𝔅
    and 𝔖 = 𝔉′ ∘SMCF 𝔉
    and 𝔖′ = 𝔊′ ∘SMCF 𝔊
```

shows $\mathfrak{M} \circ_{NTSMCF} \mathfrak{N} : \mathfrak{S} \mapsto_{SMCF} \mathfrak{S}' : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
using *assms*(1,2) **unfolding** *assms*(3,4) **by** (*rule ntsmcf-hcomp-is-ntsmcf*)

lemma *ntsmcf-hcomp-assoc*[*smc-cs-simps*]:
— See Chapter II-5 in [39].
assumes $\mathfrak{L} : \mathfrak{F}'' \mapsto_{SMCF} \mathfrak{G}'' : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
and $\mathfrak{M} : \mathfrak{F}' \mapsto_{SMCF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $(\mathfrak{L} \circ_{NTSMCF} \mathfrak{M}) \circ_{NTSMCF} \mathfrak{N} = \mathfrak{L} \circ_{NTSMCF} (\mathfrak{M} \circ_{NTSMCF} \mathfrak{N})$
proof–
interpret $\mathfrak{L}$: *is-ntsmcf* α $\mathfrak{C}$ $\mathfrak{D}$ $\mathfrak{F}''$ $\mathfrak{G}''$ $\mathfrak{L}$ **by** (*rule assms*(1))
interpret $\mathfrak{M}$: *is-ntsmcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}'$ $\mathfrak{G}'$ $\mathfrak{M}$ **by** (*rule assms*(2))
interpret $\mathfrak{N}$: *is-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(3))
interpret $\mathfrak{LM}$: *is-ntsmcf* α $\mathfrak{B}$ $\mathfrak{D}$ $\langle\mathfrak{F}'' \circ_{SMCF} \mathfrak{F}'\rangle$ $\langle\mathfrak{G}'' \circ_{SMCF} \mathfrak{G}'\rangle$ $\langle\mathfrak{L} \circ_{NTSMCF} \mathfrak{M}\rangle$
by (*auto intro*: *smc-cs-intros*)
interpret $\mathfrak{MN}$: *is-ntsmcf* α $\mathfrak{A}$ $\mathfrak{C}$ $\langle\mathfrak{F}' \circ_{SMCF} \mathfrak{F}\rangle$ $\langle\mathfrak{G}' \circ_{SMCF} \mathfrak{G}\rangle$ $\langle\mathfrak{M} \circ_{NTSMCF} \mathfrak{N}\rangle$
by (*auto intro*: *smc-cs-intros*)
note *smcf-axioms* =
$\mathfrak{L}$.*NTDom.is-semifunctor-axioms*
$\mathfrak{L}$.*NTCod.is-semifunctor-axioms*
$\mathfrak{M}$.*NTDom.is-semifunctor-axioms*
$\mathfrak{M}$.*NTCod.is-semifunctor-axioms*
$\mathfrak{N}$.*NTDom.is-semifunctor-axioms*
$\mathfrak{N}$.*NTCod.is-semifunctor-axioms*
show *?thesis*
proof(*rule ntsmcf-eqI*)
from *assms* **show**
$\mathfrak{L} \circ_{NTSMCF} \mathfrak{M} \circ_{NTSMCF} \mathfrak{N} :$
$(\mathfrak{F}'' \circ_{SMCF} \mathfrak{F}') \circ_{SMCF} \mathfrak{F} \mapsto_{SMCF} (\mathfrak{G}'' \circ_{SMCF} \mathfrak{G}') \circ_{SMCF} \mathfrak{G} :$
$\mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
by (*auto intro*: *smc-cs-intros*)
from $\mathfrak{LM}$.*is-ntsmcf-axioms* $\mathfrak{N}$.*is-ntsmcf-axioms* **have** *dom-lhs*:
$\mathcal{D}_\circ((\mathfrak{L} \circ_{NTSMCF} \mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
by (*simp add*: *smc-cs-simps*)
from $\mathfrak{MN}$.*is-ntsmcf-axioms* $\mathfrak{L}$.*is-ntsmcf-axioms* **have** *dom-rhs*:
$\mathcal{D}_\circ((\mathfrak{L} \circ_{NTSMCF} (\mathfrak{M} \circ_{NTSMCF} \mathfrak{N}))(\lfloor NTMap \rfloor)) = \mathfrak{A}(\lfloor Obj \rfloor)$
by (*simp add*: *smc-cs-simps*)
show $(\mathfrak{L} \circ_{NTSMCF} \mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor) = (\mathfrak{L} \circ_{NTSMCF} (\mathfrak{M} \circ_{NTSMCF} \mathfrak{N}))(\lfloor NTMap \rfloor)$
proof(*rule vsv-eqI*, *unfold dom-lhs dom-rhs*)
fix *a* **assume** $a \in_\circ \mathfrak{A}(\lfloor Obj \rfloor)$
with *assms* **show**
$(\mathfrak{L} \circ_{NTSMCF} \mathfrak{M} \circ_{NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor a \rfloor) =$
$(\mathfrak{L} \circ_{NTSMCF} (\mathfrak{M} \circ_{NTSMCF} \mathfrak{N}))(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
by (*cs-concl* **cs-shallow cs-simp**: *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
qed (*simp-all add*: *ntsmcf-hcomp-components*)
qed
(
insert smcf-axioms,
auto simp: *smcf-comp-assoc intro*!: *smc-cs-intros*
)
qed

Opposite of the horizontal composition of the natural transformation of semifunctors

lemma *op-ntsmcf-ntsmcf-hcomp*[*smc-op-simps*]:
assumes $\mathfrak{M} : \mathfrak{F}' \mapsto_{SMCF} \mathfrak{G}' : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$

```
  shows op-ntsmcf (𝔐 ∘_NTSMCF 𝔑) = op-ntsmcf 𝔐 ∘_NTSMCF op-ntsmcf 𝔑
proof–
  interpret 𝔐: is-ntsmcf α 𝔅 ℭ 𝔉′ 𝔊′ 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  have op-𝔐: op-ntsmcf 𝔐 :
    op-smcf 𝔊′ ↦_SMCF op-smcf 𝔉′ : op-smc 𝔅 ↦↦_SMCα op-smc ℭ
    and op-𝔑: op-ntsmcf 𝔑 :
    op-smcf 𝔊 ↦_SMCF op-smcf 𝔉 : op-smc 𝔄 ↦↦_SMCα op-smc 𝔅
    by
      (
        cs-concl cs-shallow
          cs-simp: smc-op-simps cs-intro: smc-cs-intros smc-op-intros
      )
  show ?thesis
  proof(rule sym, rule ntsmcf-eqI, unfold smc-op-simps slicing-simps)
    show
      op-ntsmcf 𝔐 ∘_NTSMCF op-ntsmcf 𝔑 :
        op-smcf 𝔊′ ∘_SMCF op-smcf 𝔊 ↦_SMCF op-smcf 𝔉′ ∘_SMCF op-smcf 𝔉 :
        op-smc 𝔄 ↦↦_SMCα op-smc ℭ
      by
        (
          cs-concl cs-shallow
            cs-simp: smc-op-simps cs-intro: smc-cs-intros smc-op-intros
        )
    show op-ntsmcf (𝔐 ∘_NTSMCF 𝔑) :
      op-smcf 𝔊′ ∘_SMCF op-smcf 𝔊 ↦_SMCF op-smcf 𝔉′ ∘_SMCF op-smcf 𝔉 :
      op-smc 𝔄 ↦↦_SMCα op-smc ℭ
      by
        (
          cs-concl cs-shallow
            cs-simp: smc-op-simps cs-intro: smc-cs-intros smc-op-intros
        )
    show (op-ntsmcf 𝔐 ∘_NTSMCF op-ntsmcf 𝔑)(|NTMap|) = (𝔐 ∘_NTSMCF 𝔑)(|NTMap|)
    proof
      (
        rule vsv-eqI,
        unfold
          ntsmcf-hcomp-NTMap-vdomain[OF assms(2)]
          ntsmcf-hcomp-NTMap-vdomain[OF op-𝔑]
          smc-op-simps
      )
      fix a assume a ∈∘ 𝔄(|Obj|)
      with assms show
        (op-ntsmcf 𝔐 ∘_NTSMCF op-ntsmcf 𝔑)(|NTMap|)(|a|) = (𝔐 ∘_NTSMCF 𝔑)(|NTMap|)(|a|)
        by
          (
            cs-concl cs-shallow
              cs-simp: smc-cs-simps smc-op-simps
              cs-intro: smc-cs-intros smc-op-intros
          )
    qed (auto simp: ntsmcf-hcomp-components)
  qed simp-all
qed
```

4.6.6 Interchange law

lemma *ntsmcf-comp-interchange-law*:
— See Chapter II-5 in [39].

```
  assumes 𝔐 : 𝔊 ↦_SMCF 𝔎 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔐' : 𝔊' ↦_SMCF 𝔎' : 𝔅 ↦↦_SMCα ℭ
    and 𝔑' : 𝔉' ↦_SMCF 𝔊' : 𝔅 ↦↦_SMCα ℭ
  shows
    ((𝔐' ·_NTSMCF 𝔑') ∘_NTSMCF (𝔐 ·_NTSMCF 𝔑)) =
      (𝔐' ∘_NTSMCF 𝔐) ·_NTSMCF (𝔑' ∘_NTSMCF 𝔑)
proof–
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔊 𝔎 𝔐 by (rule assms(1))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(2))
  interpret 𝔐': is-ntsmcf α 𝔅 ℭ 𝔊' 𝔎' 𝔐' by (rule assms(3))
  interpret 𝔑': is-ntsmcf α 𝔅 ℭ 𝔉' 𝔊' 𝔑' by (rule assms(4))
  interpret 𝔑'𝔑:
    is-ntsmcf α 𝔄 ℭ ‹𝔉' ∘_SMCF 𝔉› ‹𝔊' ∘_SMCF 𝔊› ‹𝔑' ∘_NTSMCF 𝔑›
    by (auto intro: smc-cs-intros)
  interpret 𝔐𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔎 ‹𝔐 ·_NTSMCF 𝔑›
    by (auto intro: smc-cs-intros)
  show ?thesis
  proof(rule ntsmcf-eqI[of α])
    show
      (𝔐' ·_NTSMCF 𝔑' ∘_NTSMCF (𝔐 ·_NTSMCF 𝔑))(|NTMap|) =
        (𝔐' ∘_NTSMCF 𝔐 ·_NTSMCF (𝔑' ∘_NTSMCF 𝔑))(|NTMap|)
    proof
      (
        rule vsv-eqI,
        unfold
          ntsmcf-vcomp-NTMap-vdomain[OF 𝔑'𝔑.is-ntsmcf-axioms]
          ntsmcf-hcomp-NTMap-vdomain[OF 𝔐𝔑.is-ntsmcf-axioms]
      )
      fix a assume a ∈_∘ 𝔄(|Obj|)
      with assms show
        (𝔐' ·_NTSMCF 𝔑' ∘_NTSMCF (𝔐 ·_NTSMCF 𝔑))(|NTMap|)(|a|) =
          ((𝔐' ∘_NTSMCF 𝔐) ·_NTSMCF (𝔑' ∘_NTSMCF 𝔑))(|NTMap|)(|a|)
        by
          (
            cs-concl
              cs-simp: smc-cs-simps is-ntsmcf.ntsmcf-Comp-commute'
              cs-intro: smc-cs-intros
          )
    qed (auto intro: smc-cs-intros)
  qed (auto intro: smc-cs-intros)
qed
```

4.6.7 Composition of a natural transformation of semifunctors and a semifunctor

Definition and elementary properties

```
abbreviation (input) ntsmcf-smcf-comp :: V ⇒ V ⇒ V (infixl ‹∘_NTSMCF-SMCF› 55)
  where ntsmcf-smcf-comp ≡ tdghm-dghm-comp
```

Slicing.

```
lemma ntsmcf-tdghm-ntsmcf-smcf-comp[slicing-commute]:
  ntsmcf-tdghm 𝔑 ∘_TDGHM-DGHM smcf-dghm 𝔎 = ntsmcf-tdghm (𝔑 ∘_NTSMCF-SMCF 𝔎)
  unfolding
    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*)

Natural transformation map

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

lemmas [*smc-cs-simps*] =
is-semifunctor.ntsmcf-smcf-comp-NTMap-vdomain
is-semifunctor.ntsmcf-smcf-comp-NTMap-app

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

Opposite of the composition of a natural transformation of semifunctors and a semifunctor

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

Composition of a natural transformation of semifunctors and a semifunctors is a natural transformation of semifunctors

lemma *ntsmcf-smcf-comp-is-ntsmcf*[*intro*]:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathfrak{N} \circ_{NTSMCF\text{-}SMCF} \mathfrak{H} : \mathfrak{F} \circ_{SMCF} \mathfrak{H} \mapsto_{SMCF} \mathfrak{G} \circ_{SMCF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-ntsmcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{H}$: *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{H}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule is-ntsmcfI*)
show *vfsequence* $(\mathfrak{N} \circ_{NTSMCF\text{-}SMCF} \mathfrak{H})$
unfolding *tdghm-dghm-comp-def* **by** (*simp add*: *nat-omega-simps*)

```
    from assms show 𝔉 ∘SMCF 𝔎 : 𝔄 ↦↦SMCα ℭ
      by (cs-concl cs-intro: smc-cs-intros)
    from assms show 𝔊 ∘SMCF 𝔎 : 𝔄 ↦↦SMCα ℭ
      by (cs-concl cs-intro: smc-cs-intros)
    show vcard (𝔑 ∘NTSMCF-SMCF 𝔎) = 5ℕ
      unfolding tdghm-dghm-comp-def by (simp add: nat-omega-simps)
    from assms show
      ntsmcf-tdghm (𝔑 ∘NTSMCF-SMCF 𝔎) :
        smcf-dghm (𝔉 ∘SMCF 𝔎) ↦DGHM smcf-dghm (𝔊 ∘SMCF 𝔎) :
        smc-dg 𝔄 ↦↦DGα smc-dg ℭ
      by
        (
          cs-concl
            cs-simp: slicing-commute[symmetric]
            cs-intro: slicing-intros dg-cs-intros
        )
    show
      (𝔑 ∘NTSMCF-SMCF 𝔎)(|NTMap|)(|b|) ∘A ℭ (𝔉 ∘SMCF 𝔎)(|ArrMap|)(|f|) =
        (𝔊 ∘SMCF 𝔎)(|ArrMap|)(|f|) ∘A ℭ (𝔑 ∘NTSMCF-SMCF 𝔎)(|NTMap|)(|a|)
      if f : a ↦𝔄 b for a b f
      using that by (cs-concl cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  qed (auto simp: smc-cs-simps)
qed

lemma ntsmcf-smcf-comp-is-semifunctor'[smc-cs-intros]:
  assumes 𝔑 : 𝔉 ↦SMCF 𝔊 : 𝔅 ↦↦SMCα ℭ
    and 𝔎 : 𝔄 ↦↦SMCα 𝔅
    and 𝔉' = 𝔉 ∘SMCF 𝔎
    and 𝔊' = 𝔊 ∘SMCF 𝔎
  shows 𝔑 ∘NTSMCF-SMCF 𝔎 : 𝔉' ↦SMCF 𝔊' : 𝔄 ↦↦SMCα ℭ
  using assms(1,2) unfolding assms(3,4) ..
```

Further properties

```
lemma ntsmcf-smcf-comp-ntsmcf-smcf-comp-assoc:
  assumes 𝔑 : 𝔎 ↦SMCF 𝔎' : ℭ ↦↦SMCα 𝔇
    and 𝔊 : 𝔅 ↦↦SMCα ℭ
    and 𝔉 : 𝔄 ↦↦SMCα 𝔅
  shows (𝔑 ∘NTSMCF-SMCF 𝔊) ∘NTSMCF-SMCF 𝔉 = 𝔑 ∘NTSMCF-SMCF (𝔊 ∘SMCF 𝔉)
proof–
  interpret 𝔑: is-ntsmcf α ℭ 𝔇 𝔎 𝔎' 𝔑 by (rule assms(1))
  interpret 𝔊: is-semifunctor α 𝔅 ℭ 𝔊 by (rule assms(2))
  interpret 𝔉: is-semifunctor α 𝔄 𝔅 𝔉 by (rule assms(3))
  show ?thesis
  proof(rule ntsmcf-tdghm-eqI)
    from assms show
      (𝔑 ∘NTSMCF-SMCF 𝔊) ∘NTSMCF-SMCF 𝔉 :
        𝔎 ∘SMCF 𝔊 ∘SMCF 𝔉 ↦SMCF 𝔎' ∘SMCF 𝔊 ∘SMCF 𝔉 :
        𝔄 ↦↦SMCα 𝔇
      by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
    show 𝔑 ∘NTSMCF-SMCF (𝔊 ∘SMCF 𝔉) :
      𝔎 ∘SMCF 𝔊 ∘SMCF 𝔉 ↦SMCF 𝔎' ∘SMCF 𝔊 ∘SMCF 𝔉 :
      𝔄 ↦↦SMCα 𝔇
      by (cs-concl cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
    from assms show
      ntsmcf-tdghm ((𝔑 ∘NTSMCF-SMCF 𝔊) ∘NTSMCF-SMCF 𝔉) =
        ntsmcf-tdghm (𝔑 ∘NTSMCF-SMCF (𝔊 ∘SMCF 𝔉))
      by
```

```
    (
      cs-concl
        cs-simp: slicing-commute[symmetric]
        cs-intro: slicing-intros tdghm-dghm-comp-tdghm-dghm-comp-assoc
    )
  qed simp-all
qed

lemma (in is-ntsmcf) ntsmcf-ntsmcf-smcf-comp-smcf-id[smc-cs-simps]:
  𝔑 ∘_NTSMCF-SMCF smcf-id 𝔄 = 𝔑
proof(rule ntsmcf-tdghm-eqI)
  show 𝔑 ∘_NTSMCF-SMCF smcf-id 𝔄 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    by (cs-concl cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  show 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  show ntsmcf-tdghm (𝔑 ∘_NTSMCF-SMCF smcf-id 𝔄) = ntsmcf-tdghm 𝔑
    by
      (
        cs-concl cs-shallow
          cs-simp: slicing-simps slicing-commute[symmetric]
          cs-intro: smc-cs-intros slicing-intros dg-cs-simps
      )
qed simp-all

lemmas [smc-cs-simps] = is-ntsmcf.ntsmcf-ntsmcf-smcf-comp-smcf-id

lemma ntsmcf-vcomp-ntsmcf-smcf-comp[smc-cs-simps]:
  assumes 𝔎 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔐 : 𝔊 ↦_SMCF ℌ : 𝔅 ↦↦_SMCα ℭ
    and 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔅 ↦↦_SMCα ℭ
  shows
    (𝔐 ∘_NTSMCF-SMCF 𝔎) ·_NTSMCF (𝔑 ∘_NTSMCF-SMCF 𝔎) =
      (𝔐 ·_NTSMCF 𝔑) ∘_NTSMCF-SMCF 𝔎
proof(rule ntsmcf-eqI)
  from assms show (𝔐 ·_NTSMCF 𝔑) ∘_NTSMCF-SMCF 𝔎 :
    𝔉 ∘_SMCF 𝔎 ↦_SMCF ℌ ∘_SMCF 𝔎 : 𝔄 ↦↦_SMCα ℭ
    by (cs-concl cs-shallow cs-intro: smc-cs-intros)
  from assms show 𝔐 ∘_NTSMCF-SMCF 𝔎 ·_NTSMCF (𝔑 ∘_NTSMCF-SMCF 𝔎) :
    𝔉 ∘_SMCF 𝔎 ↦_SMCF ℌ ∘_SMCF 𝔎 : 𝔄 ↦↦_SMCα ℭ
    by (cs-concl cs-shallow cs-intro: smc-cs-intros)
  from assms have dom-lhs:
    𝒟∘ ((𝔐 ∘_NTSMCF-SMCF 𝔎 ·_NTSMCF (𝔑 ∘_NTSMCF-SMCF 𝔎))(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
    by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  from assms have dom-rhs: 𝒟∘ ((𝔐 ·_NTSMCF 𝔑 ∘_NTSMCF-SMCF 𝔎)(⌊NTMap⌋)) = 𝔄(⌊Obj⌋)
    by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  show
    (𝔐 ∘_NTSMCF-SMCF 𝔎 ·_NTSMCF (𝔑 ∘_NTSMCF-SMCF 𝔎))(⌊NTMap⌋) =
      (𝔐 ·_NTSMCF 𝔑 ∘_NTSMCF-SMCF 𝔎)(⌊NTMap⌋)
  proof(rule vsv-eqI, unfold dom-lhs dom-rhs)
    fix a assume a ∈∘ 𝔄(⌊Obj⌋)
    with assms show
      (𝔐 ∘_NTSMCF-SMCF 𝔎 ·_NTSMCF (𝔑 ∘_NTSMCF-SMCF 𝔎))(⌊NTMap⌋)(⌊a⌋) =
        (𝔐 ·_NTSMCF 𝔑 ∘_NTSMCF-SMCF 𝔎)(⌊NTMap⌋)(⌊a⌋)
      by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
  qed (cs-concl cs-shallow cs-intro: smc-cs-intros)+
qed simp-all
```

4.6.8 Composition of a semifunctor and a natural transformation of semifunctors

Definition and elementary properties

abbreviation (*input*) *smcf-ntsmcf-comp* :: $V \Rightarrow V \Rightarrow V$ (**infixl** $\langle\circ_{SMCF-NTSMCF}\rangle$ 55)
where *smcf-ntsmcf-comp* ≡ *dghm-tdghm-comp*

Slicing.

lemma *ntsmcf-tdghm-smcf-ntsmcf-comp*[*slicing-commute*]:
smcf-dghm $\mathfrak{H}$ $\circ_{DGHM-TDGHM}$ *ntsmcf-tdghm* $\mathfrak{N}$ = *ntsmcf-tdghm* ($\mathfrak{H}$ $\circ_{SMCF-NTSMCF}$ $\mathfrak{N}$)
unfolding
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*)

Natural transformation map

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

lemmas [*smc-cs-simps*] =
is-ntsmcf.smcf-ntsmcf-comp-NTMap-vdomain
is-ntsmcf.smcf-ntsmcf-comp-NTMap-app

lemma *smcf-ntsmcf-comp-NTMap-vrange*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathcal{R}_\circ$ (($\mathfrak{H}$ $\circ_{SMCF-NTSMCF}$ $\mathfrak{N}$)(⦇*NTMap*⦈)) $\subseteq_\circ$ $\mathfrak{C}$(⦇*Arr*⦈)
proof–
interpret $\mathfrak{H}$: *is-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-ntsmcf* 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*: *smc-cs-simps intro*: *smc-cs-intros*)
qed

Opposite of the composition of a semifunctor and a natural transformation of semifunctors

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

Composition of a semifunctor and a natural transformation of semifunctors is a natural transformation of semifunctors

lemma *smcf-ntsmcf-comp-is-ntsmcf*[*intro*]:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N} : \mathfrak{H} \circ_{SMCF} \mathfrak{F} \mapsto_{SMCF} \mathfrak{H} \circ_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{H}$: *is-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule is-ntsmcfI*)
show *vfsequence* $(\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N})$ **unfolding** *dghm-tdghm-comp-def* **by** *simp*
from *assms* **show** $\mathfrak{H} \circ_{SMCF} \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *smc-cs-intros*)
from *assms* **show** $\mathfrak{H} \circ_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
by (*cs-concl* **cs-intro**: *smc-cs-intros*)
show *vcard* $(\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N}) = 5_{\mathbb{N}}$
unfolding *dghm-tdghm-comp-def* **by** (*simp add*: *nat-omega-simps*)
from *assms* **show** *ntsmcf-tdghm* $(\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N})$:
smcf-dghm $(\mathfrak{H} \circ_{SMCF} \mathfrak{F}) \mapsto_{DGHM}$ *smcf-dghm* $(\mathfrak{H} \circ_{SMCF} \mathfrak{G})$:
smc-dg $\mathfrak{A} \mapsto\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{C}$
by
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *dg-cs-intros slicing-intros*
)
have [*smc-cs-simps*]:
$\mathfrak{H}(\lfloor ArrMap \rfloor)(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor b \rfloor)) \circ_{A\mathfrak{C}} \mathfrak{H}(\lfloor ArrMap \rfloor)(\mathfrak{F}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)) =$
$\mathfrak{H}(\lfloor ArrMap \rfloor)(\mathfrak{G}(\lfloor ArrMap \rfloor)(\lfloor f \rfloor)) \circ_{A\mathfrak{C}} \mathfrak{H}(\lfloor ArrMap \rfloor)(\mathfrak{N}(\lfloor NTMap \rfloor)(\lfloor a \rfloor))$
if $f : a \mapsto_{\mathfrak{A}} b$ **for** a b f
using *assms that*
by
(
cs-concl
cs-simp:
is-ntsmcf.ntsmcf-Comp-commute
is-semifunctor.smcf-ArrMap-Comp[*symmetric*]
cs-intro: *smc-cs-intros*
)
from *assms* **show**
$(\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor b \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{H} \circ_{SMCF} \mathfrak{F})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) =$
$(\mathfrak{H} \circ_{SMCF} \mathfrak{G})(\lfloor ArrMap \rfloor)(\lfloor f \rfloor) \circ_{A\mathfrak{C}} (\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N})(\lfloor NTMap \rfloor)(\lfloor a \rfloor)$
if $f : a \mapsto_{\mathfrak{A}} b$ **for** a b f
using *assms that*
by (*cs-concl* **cs-simp**: *smc-cs-simps* **cs-intro**: *smc-cs-intros*)
qed (*auto simp*: *smc-cs-simps*)
qed

lemma *smcf-ntsmcf-comp-is-semifunctor'*[*smc-cs-intros*]:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{F}' = \mathfrak{H} \circ_{SMCF} \mathfrak{F}$
and $\mathfrak{G}' = \mathfrak{H} \circ_{SMCF} \mathfrak{G}$
shows $\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N} : \mathfrak{F}' \mapsto_{SMCF} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
using *assms*(1,2) **unfolding** *assms*(3,4) **..**

Further properties

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

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

lemmas [*smc-cs-simps*] = *is-ntsmcf.ntsmcf-smcf-ntsmcf-comp-smcf-id*

lemma *smcf-ntsmcf-comp-ntsmcf-smcf-comp-assoc*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{C}$
and $\mathfrak{H} : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
and $\mathfrak{K} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $(\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N}) \circ_{NTSMCF-SMCF} \mathfrak{K} = \mathfrak{H} \circ_{SMCF-NTSMCF} (\mathfrak{N} \circ_{NTSMCF-SMCF} \mathfrak{K})$
proof–
interpret $\mathfrak{N}$: *is-ntsmcf* $\alpha\ \mathfrak{B}\ \mathfrak{C}\ \mathfrak{F}\ \mathfrak{G}\ \mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{H}$: *is-semifunctor* $\alpha\ \mathfrak{C}\ \mathfrak{D}\ \mathfrak{H}$ **by** (*rule assms*(2))
interpret $\mathfrak{K}$: *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{K}$ **by** (*rule assms*(3))
show *?thesis*
by (*rule ntsmcf-tdghm-eqI*)
(

```
      use assms in
        ‹
          cs-concl
            cs-simp: smc-cs-simps slicing-commute[symmetric]
            cs-intro:
              smc-cs-intros
              slicing-intros
              dghm-tdghm-comp-tdghm-dghm-comp-assoc
        ›
    )+
qed

lemma smcf-ntsmcf-comp-ntsmcf-vcomp:
  assumes 𝔎 : 𝔅 ↦↦_{SMC α} ℭ
    and 𝔐 : 𝔊 ↦_{SMCF} ℌ : 𝔄 ↦↦_{SMC α} 𝔅
    and 𝔑 : 𝔉 ↦_{SMCF} 𝔊 : 𝔄 ↦↦_{SMC α} 𝔅
  shows
    𝔎 ∘_{SMCF-NTSMCF} (𝔐 ·_{NTSMCF} 𝔑) =
      (𝔎 ∘_{SMCF-NTSMCF} 𝔐) ·_{NTSMCF} (𝔎 ∘_{SMCF-NTSMCF} 𝔑)
proof–
  interpret 𝔎: is-semifunctor α 𝔅 ℭ 𝔎 by (rule assms(1))
  interpret 𝔐: is-ntsmcf α 𝔄 𝔅 𝔊 ℌ 𝔐 by (rule assms(2))
  interpret 𝔑: is-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(3))
  show
    𝔎 ∘_{SMCF-NTSMCF} (𝔐 ·_{NTSMCF} 𝔑) = 𝔎 ∘_{SMCF-NTSMCF} 𝔐 ·_{NTSMCF} (𝔎 ∘_{SMCF-NTSMCF}
𝔑)
  proof(rule ntsmcf-eqI)
    have dom-lhs: 𝒟∘ ((𝔎 ∘_{SMCF-NTSMCF} (𝔐 ·_{NTSMCF} 𝔑))(|NTMap|)) = 𝔄(|Obj|)
      unfolding dghm-tdghm-comp-components smc-cs-simps by simp
    have dom-rhs:
      𝒟∘ ((𝔎 ∘_{SMCF-NTSMCF} 𝔐 ·_{NTSMCF} (𝔎 ∘_{SMCF-NTSMCF} 𝔑))(|NTMap|)) = 𝔄(|Obj|)
      unfolding ntsmcf-vcomp-components smc-cs-simps by simp
    show
      (𝔎 ∘_{SMCF-NTSMCF} (𝔐 ·_{NTSMCF} 𝔑))(|NTMap|) =
        (𝔎 ∘_{SMCF-NTSMCF} 𝔐 ·_{NTSMCF} (𝔎 ∘_{SMCF-NTSMCF} 𝔑))(|NTMap|)
    proof(rule vsv-eqI, unfold dom-lhs dom-rhs smc-cs-simps)
      fix a assume a ∈∘ 𝔄(|Obj|)
      then show
        (𝔎 ∘_{SMCF-NTSMCF} (𝔐 ·_{NTSMCF} 𝔑))(|NTMap|)(|a|) =
          (𝔎 ∘_{SMCF-NTSMCF} 𝔐 ·_{NTSMCF} (𝔎 ∘_{SMCF-NTSMCF} 𝔑))(|NTMap|)(|a|)
        by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
    qed (cs-concl cs-shallow cs-intro: smc-cs-intros)+
  qed (cs-concl cs-shallow cs-intro: smc-cs-intros)+
qed
```

4.7 Smallness for natural transformations of semifunctors

4.7.1 Natural transformation of semifunctors with tiny maps

Definition and elementary properties

locale *is-tm-ntsmcf* = *is-ntsmcf* 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-ntsmcf-is-tm-tdghm*[*slicing-intros*]: *ntsmcf-tdghm* $\mathfrak{N}$:
smcf-dghm $\mathfrak{F}$ $\mapsto_{DGHM.tm}$ *smcf-dghm* $\mathfrak{G}$: *smc-dg* $\mathfrak{A}$ $\mapsto\!\!\mapsto_{DG.tm\alpha}$ *smc-dg* $\mathfrak{B}$

syntax *-is-tm-ntsmcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\mapsto_{SMCF.tm}$ - :/ - $\mapsto\!\!\mapsto_{SMC.tm1}$ -)› [51, 51, 51, 51, 51] 51)
syntax-consts *-is-tm-ntsmcf* $\rightleftharpoons$ *is-tm-ntsmcf*
translations $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{SMCF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.tm\alpha}$ $\mathfrak{B}$ $\rightleftharpoons$
CONST is-tm-ntsmcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

abbreviation *all-tm-ntsmcfs* :: $V \Rightarrow V$
where *all-tm-ntsmcfs* $\alpha \equiv$
set {$\mathfrak{N}$. $\exists \mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{A}$ $\mathfrak{B}$. $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{SMCF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.tm\alpha}$ $\mathfrak{B}$}

abbreviation *tm-ntsmcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *tm-ntsmcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\equiv$
set {$\mathfrak{N}$. $\exists \mathfrak{F}$ $\mathfrak{G}$. $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{SMCF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.tm\alpha}$ $\mathfrak{B}$}

abbreviation *these-tm-ntsmcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *these-tm-ntsmcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\equiv$
set {$\mathfrak{N}$. $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{SMCF.tm}$ $\mathfrak{G}$: $\mathfrak{A}$ $\mapsto\!\!\mapsto_{SMC.tm\alpha}$ $\mathfrak{B}$}

lemma (**in** *is-tm-ntsmcf*) *tm-ntsmcf-is-tm-tdghm'*:
assumes α' = α
and $\mathfrak{F}'$ = *smcf-dghm* $\mathfrak{F}$
and $\mathfrak{G}'$ = *smcf-dghm* $\mathfrak{G}$
and $\mathfrak{A}'$ = *smc-dg* $\mathfrak{A}$
and $\mathfrak{B}'$ = *smc-dg* $\mathfrak{B}$
shows *ntsmcf-tdghm* $\mathfrak{N}$:
$\mathfrak{F}'$ $\mapsto_{DGHM.tm}$ $\mathfrak{G}'$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{DG.tm_{\alpha'}}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule tm-ntsmcf-is-tm-tdghm*)

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

Rules.

lemma (**in** *is-tm-ntsmcf*) *is-tm-ntsmcf-axioms'*[*smc-small-cs-intros*]:
assumes α' = α **and** $\mathfrak{A}'$ = $\mathfrak{A}$ **and** $\mathfrak{B}'$ = $\mathfrak{B}$ **and** $\mathfrak{F}'$ = $\mathfrak{F}$ **and** $\mathfrak{G}'$ = $\mathfrak{G}$
shows $\mathfrak{N}$: $\mathfrak{F}'$ $\mapsto_{SMCF.tm}$ $\mathfrak{G}'$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{SMC.tm\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule is-tm-ntsmcf-axioms*)

mk-ide rf *is-tm-ntsmcf-def*[*unfolded is-tm-ntsmcf-axioms-def*]
|*intro is-tm-ntsmcfI*|
|*dest is-tm-ntsmcfD*[*dest*]|
|*elim is-tm-ntsmcfE*[*elim*]|

lemmas [*smc-small-cs-intros*] = *is-tm-ntsmcfD*(1)

Slicing.

context *is-tm-ntsmcf*
begin

interpretation *tdghm*: *is-tm-tdghm*
α ‹*smc-dg* $\mathfrak{A}$› ‹*smc-dg* $\mathfrak{B}$› ‹*smcf-dghm* $\mathfrak{F}$› ‹*smcf-dghm* $\mathfrak{G}$› ‹*ntsmcf-tdghm* $\mathfrak{N}$›

by (*rule tm-ntsmcf-is-tm-tdghm*)

lemmas-with [*unfolded slicing-simps*]:
tm-ntsmcf-NTMap-in-Vset = *tdghm.tm-tdghm-NTMap-in-Vset*

end

Elementary properties.

sublocale *is-tm-ntsmcf* ⊆ *NTDom*: *is-tm-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$
using *tm-ntsmcf-is-tm-tdghm*
by (*intro is-tm-semifunctorI*) (*auto simp*: *smc-cs-intros*)

sublocale *is-tm-ntsmcf* ⊆ *NTCod*: *is-tm-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$
using *tm-ntsmcf-is-tm-tdghm*
by (*intro is-tm-semifunctorI*) (*auto simp*: *smc-cs-intros*)

Further rules.

lemma *is-tm-ntsmcfI'*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
and $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
and $\mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
shows $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
proof–
interpret *is-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{F}$: *is-tm-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ **by** (*rule assms*(2))
interpret $\mathfrak{G}$: *is-tm-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ **by** (*rule assms*(3))
show *?thesis*
proof(*intro is-tm-ntsmcfI*)
show *ntsmcf-tdghm* $\mathfrak{N}$:
smcf-dghm $\mathfrak{F} \mapsto_{DGHM.tm}$ *smcf-dghm* $\mathfrak{G}$: *smc-dg* $\mathfrak{A} \mapsto\mapsto_{DG.tm\alpha}$ *smc-dg* $\mathfrak{B}$
by (*intro is-tm-tdghmI*) (*auto simp*: *slicing-intros*)
qed (*auto simp*: *assms*(2,3) *vfsequence-axioms smc-cs-intros*)
qed

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

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

Size.

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

```
    simp only: elts-of-set small-all-ntsmcfs if-True,
    rule subsetI,
    unfold mem-Collect-eq
  )
  fix 𝔑 assume ∃ℱ 𝒢 𝒜 ℬ. 𝔑 : ℱ ↦_SMCF.tm 𝒢 : 𝒜 ↦↦_SMC.tmα ℬ
  then obtain ℱ 𝒢 𝒜 ℬ where 𝔑 : ℱ ↦_SMCF.tm 𝒢 : 𝒜 ↦↦_SMC.tmα ℬ
    by clarsimp
  then interpret is-tm-ntsmcf α 𝒜 ℬ ℱ 𝒢 𝔑 .
  have 𝔑 : ℱ ↦_SMCF 𝒢 : 𝒜 ↦↦_SMCα ℬ by (auto simp: smc-cs-intros)
  then show ∃ℱ 𝒢 𝒜 ℬ. 𝔑 : ℱ ↦_SMCF 𝒢 : 𝒜 ↦↦_SMCα ℬ by auto
 qed
qed

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

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

Further elementary results.

```
lemma these-tm-ntsmcfs-iff:
 𝔑 ∈∘ these-tm-ntsmcfs α 𝒜 ℬ ℱ 𝒢 ⟷
  𝔑 : ℱ ↦_SMCF.tm 𝒢 : 𝒜 ↦↦_SMC.tmα ℬ
 by auto
```

Opposite natural transformation of semifunctors with tiny maps

```
lemma (in is-tm-ntsmcf) is-tm-ntsmcf-op: op-ntsmcf 𝔑 :
 op-smcf 𝒢 ↦_SMCF.tm op-smcf ℱ : op-smc 𝒜 ↦↦_SMC.tmα op-smc ℬ
 by (intro is-tm-ntsmcfI′)
  (cs-concl cs-shallow cs-intro: smc-cs-intros smc-op-intros)+

lemma (in is-tm-ntsmcf) is-tm-ntsmcf-op′[smc-op-intros]:
 assumes 𝒢′ = op-smcf 𝒢
  and ℱ′ = op-smcf ℱ
  and 𝒜′ = op-smc 𝒜
  and ℬ′ = op-smc ℬ
 shows op-ntsmcf 𝔑 : 𝒢′ ↦_SMCF.tm ℱ′ : 𝒜′ ↦↦_SMC.tmα ℬ′
 unfolding assms by (rule is-tm-ntsmcf-op)

lemmas is-tm-ntsmcf-op[smc-op-intros] = is-tm-ntsmcf.is-tm-ntsmcf-op′
```

Vertical composition of natural transformations of semifunctors with tiny maps

lemma *ntsmcf-vcomp-is-tm-ntsmcf*[*smc-small-cs-intros*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{SMCF.tm} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
shows $\mathfrak{M} \bullet_{NTSMCF} \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{M}$: *is-tm-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-tm-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))
show *?thesis*
by (*rule is-tm-ntsmcfI'*) (*auto intro*: *smc-cs-intros smc-small-cs-intros*)
qed

Composition of a natural transformation of semifunctors and a semifunctor

lemma *ntsmcf-smcf-comp-is-tm-ntsmcf*:
assumes $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{G} : \mathfrak{B} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
shows $\mathfrak{N} \circ_{NTSMCF-SMCF} \mathfrak{H} : \mathfrak{F} \circ_{SMCF} \mathfrak{H} \mapsto_{SMCF.tm} \mathfrak{G} \circ_{SMCF} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-tm-ntsmcf* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
interpret $\mathfrak{H}$: *is-tm-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{H}$ **by** (*rule assms*(2))
from *assms* **show** *?thesis*
by (*intro is-tm-ntsmcfI*)
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *smc-cs-intros dg-small-cs-intros slicing-intros*
)+
qed

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

Composition of a semifunctor and a natural transformation of semifunctors

lemma *smcf-ntsmcf-comp-is-tm-ntsmcf*:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{C}$ **and** $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
shows $\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N} : \mathfrak{H} \circ_{SMCF} \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{H} \circ_{SMCF} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{H}$: *is-tm-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ $\mathfrak{H}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-tm-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))
from *assms* **show** *?thesis*
by (*intro is-tm-ntsmcfI*)
(
cs-concl
cs-simp: *slicing-commute*[*symmetric*]
cs-intro: *smc-cs-intros dg-small-cs-intros slicing-intros*
)+
qed

lemma *smcf-ntsmcf-comp-is-tm-ntsmcf'*[*smc-small-cs-intros*]:
assumes $\mathfrak{H} : \mathfrak{B} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{C}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tm} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{B}$
and $\mathfrak{F}' = \mathfrak{H} \circ_{SMCF} \mathfrak{F}$

and $\mathfrak{G}' = \mathfrak{H} \circ_{SMCF} \mathfrak{G}$
shows $\mathfrak{H} \circ_{SMCF-NTSMCF} \mathfrak{N} : \mathfrak{F}' \mapsto_{SMCF.tm} \mathfrak{G}' : \mathfrak{A} \mapsto\mapsto_{SMC.tm\alpha} \mathfrak{C}$
using *assms*(1,2) **unfolding** *assms*(3,4) **by** (*rule smcf-ntsmcf-comp-is-tm-ntsmcf*)

4.7.2 Tiny natural transformation of semifunctors

Definition and elementary properties

locale *is-tiny-ntsmcf* = *is-ntsmcf* 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-ntsmcf-is-tdghm*[*slicing-intros*]: *ntsmcf-tdghm* $\mathfrak{N}$:
smcf-dghm $\mathfrak{F} \mapsto_{DGHM.tiny}$ *smcf-dghm* $\mathfrak{G}$: *smc-dg* $\mathfrak{A} \mapsto\mapsto_{DG.tiny\alpha}$ *smc-dg* $\mathfrak{B}$

syntax *-is-tiny-ntsmcf* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow$ *bool*
(‹(- :/ - $\mapsto_{SMCF.tiny}$ - :/ - $\mapsto\mapsto_{SMC.tiny}$₁ -)› [51, 51, 51, 51, 51] 51)
syntax-consts *-is-tiny-ntsmcf* $\rightleftharpoons$ *is-tiny-ntsmcf*
translations $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B} \rightleftharpoons$
CONST is-tiny-ntsmcf α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$

abbreviation *all-tiny-ntsmcfs* :: $V \Rightarrow V$
where *all-tiny-ntsmcfs* $\alpha \equiv$
set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}\ \mathfrak{A}\ \mathfrak{B}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}\}$

abbreviation *tiny-ntsmcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
where *tiny-ntsmcfs* α $\mathfrak{A}$ $\mathfrak{B} \equiv$
set $\{\mathfrak{N}.\ \exists \mathfrak{F}\ \mathfrak{G}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}\}$

abbreviation *these-tiny-ntsmcfs* :: $V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V \Rightarrow V$
where *these-tiny-ntsmcfs* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G} \equiv$
set $\{\mathfrak{N}.\ \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}\}$

lemma (**in** *is-tiny-ntsmcf*) *tiny-ntsmcf-is-tdghm'*:
assumes $\alpha' = \alpha$
and $\mathfrak{F}' =$ *smcf-dghm* $\mathfrak{F}$
and $\mathfrak{G}' =$ *smcf-dghm* $\mathfrak{G}$
and $\mathfrak{A}' =$ *smc-dg* $\mathfrak{A}$
and $\mathfrak{B}' =$ *smc-dg* $\mathfrak{B}$
shows *ntsmcf-tdghm* $\mathfrak{N} : \mathfrak{F}' \mapsto_{DGHM.tiny} \mathfrak{G}' : \mathfrak{A}' \mapsto\mapsto_{DG.tiny_{\alpha'}} \mathfrak{B}'$
unfolding *assms* **by** (*rule tiny-ntsmcf-is-tdghm*)

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

Rules.

lemma (**in** *is-tiny-ntsmcf*) *is-tiny-ntsmcf-axioms'*[*smc-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 $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}$
unfolding *assms* **by** (*rule is-tiny-ntsmcf-axioms*)

mk-ide rf *is-tiny-ntsmcf-def*[*unfolded is-tiny-ntsmcf-axioms-def*]
|*intro is-tiny-ntsmcfI*|
|*dest is-tiny-ntsmcfD*[*dest*]|
|*elim is-tiny-ntsmcfE*[*elim*]|

Elementary properties.

sublocale *is-tiny-ntsmcf* ⊆ *NTDom*: *is-tiny-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$
using *tiny-ntsmcf-is-tdghm*
by (*intro is-tiny-semifunctorI*) (*auto simp*: *smc-cs-intros*)

sublocale *is-tiny-ntsmcf* ⊆ *NTCod*: *is-tiny-semifunctor* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$

```
  using tiny-ntsmcf-is-tdghm
  by (intro is-tiny-semifunctorI) (auto simp: smc-cs-intros)

sublocale is-tiny-ntsmcf ⊆ is-tm-ntsmcf
  by (rule is-tm-ntsmcfI')
    (auto simp: tiny-ntsmcf-is-tdghm smc-small-cs-intros smc-cs-intros)
```

Further rules.

```
lemma is-tiny-ntsmcfI':
  assumes 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔉 : 𝔄 ↦↦_SMC.tinyα 𝔅
    and 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
  shows 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
  using assms by (auto intro: is-tiny-ntsmcfI)

lemma is-tiny-ntsmcfD':
  assumes 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
  shows 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔉 : 𝔄 ↦↦_SMC.tinyα 𝔅
    and 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
proof–
  interpret is-tiny-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 by (rule assms(1))
  show 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔉 : 𝔄 ↦↦_SMC.tinyα 𝔅
    and 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
    by
      (
        auto
          simp: is-ntsmcf-axioms
          intro:
            NTDom.is-tiny-semifunctor-axioms
            NTCod.is-tiny-semifunctor-axioms
      )
qed

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

lemma is-tiny-ntsmcfE':
  assumes 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
  obtains 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
    and 𝔉 : 𝔄 ↦↦_SMC.tinyα 𝔅
    and 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
  using assms by (auto dest: is-tiny-ntsmcfD'(2,3))

lemma is-tiny-ntsmcf-iff:
  𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅 ⟷
    (
      𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅 ∧
      𝔉 : 𝔄 ↦↦_SMC.tinyα 𝔅 ∧
      𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
    )
  using is-tiny-ntsmcfI' is-tiny-ntsmcfD' by (intro iffI) auto
```

Size.

```
lemma (in is-tiny-ntsmcf) tiny-ntsmcf-in-Vset: 𝔑 ∈ₒ Vset α
proof–
  note [smc-cs-intros] =
    tm-ntsmcf-NTMap-in-Vset
```

```
    NTDom.tiny-smcf-in-Vset
    NTCod.tiny-smcf-in-Vset
    NTDom.HomDom.tiny-smc-in-Vset
    NTDom.HomCod.tiny-smc-in-Vset
  show ?thesis
    by (subst ntsmcf-def)
      (
        cs-concl cs-shallow
          cs-simp: smc-cs-simps cs-intro: smc-cs-intros V-cs-intros
      )
qed

lemma small-all-tiny-ntsmcfs[simp]:
  small {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅}
proof(rule down)
  show {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅} ⊆
    elts (set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅})
  proof
    (
      simp only: elts-of-set small-all-ntsmcfs if-True,
      rule subsetI,
      unfold mem-Collect-eq
    )
    fix 𝔑 assume ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
    then obtain 𝔉 𝔊 𝔄 𝔅 where 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅
      by clarsimp
    then interpret is-tiny-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 .
    have 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅
      by (auto intro: smc-cs-intros)
    then show ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF 𝔊 : 𝔄 ↦↦_SMCα 𝔅 by auto
  qed
qed

lemma small-tiny-ntsmcfs[simp]:
  small {𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅}
  by
    (
      rule
        down[
          of
            -
            ‹set {𝔑. ∃𝔉 𝔊 𝔄 𝔅. 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅}›
          ]
    )
    auto

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

lemma tiny-ntsmcfs-vsubset-Vset[simp]:
  set {𝔑. ∃𝔉 𝔊. 𝔑 : 𝔉 ↦_SMCF.tiny 𝔊 : 𝔄 ↦↦_SMC.tinyα 𝔅} ⊆∘ Vset α
```

```
  (is ‹set ?ntsmcfs ⊆∘ -›)
proof(cases ‹tiny-semicategory α 𝔄 ∧ tiny-semicategory α 𝔅›)
  case True
  then have tiny-semicategory α 𝔄 and tiny-semicategory α 𝔅 by auto
  show ?thesis
  proof(rule vsubsetI)
    fix 𝔑 assume 𝔑 ∈∘ set ?ntsmcfs
    then obtain 𝔉 𝔊 where 𝔑 : 𝔉 ↦SMCF.tiny 𝔊 : 𝔄 ↦↦SMC.tinyα 𝔅 by auto
    then interpret is-tiny-ntsmcf α 𝔄 𝔅 𝔉 𝔊 𝔑 .
    from tiny-ntsmcf-in-Vset show 𝔑 ∈∘ Vset α by simp
  qed
next
  case False
  then have set ?ntsmcfs = 0
    unfolding is-tiny-ntsmcf-iff is-tiny-semifunctor-iff by auto
  then show ?thesis by simp
qed

lemma (in is-ntsmcf) ntsmcf-is-tiny-ntsmcf-if-ge-Limit:
  assumes 𝒵 β and α ∈∘ β
  shows 𝔑 : 𝔉 ↦SMCF.tiny 𝔊 : 𝔄 ↦↦SMC.tinyβ 𝔅
proof(intro is-tiny-ntsmcfI)
  interpret β: 𝒵 β by (rule assms(1))
  show 𝔑 : 𝔉 ↦SMCF 𝔊 : 𝔄 ↦↦SMCβ 𝔅
    by (intro ntsmcf-is-ntsmcf-if-ge-Limit)
      (use assms(2) in ‹cs-concl cs-shallow cs-intro: dg-cs-intros›)+
  show ntsmcf-tdghm 𝔑 :
    smcf-dghm 𝔉 ↦DGHM.tiny smcf-dghm 𝔊 : smc-dg 𝔄 ↦↦DG.tinyβ smc-dg 𝔅
    by
      (
        rule is-tdghm.tdghm-is-tiny-tdghm-if-ge-Limit,
        rule ntsmcf-is-tdghm;
        intro assms
      )
qed
```

Further elementary results.

```
lemma these-tiny-ntsmcfs-iff:
  𝔑 ∈∘ these-tiny-ntsmcfs α 𝔄 𝔅 𝔉 𝔊 ⟷
    𝔑 : 𝔉 ↦SMCF.tiny 𝔊 : 𝔄 ↦↦SMC.tinyα 𝔅
  by auto
```

Opposite natural transformation of tiny semifunctors

```
lemma (in is-tiny-ntsmcf) is-tm-ntsmcf-op: op-ntsmcf 𝔑 :
  op-smcf 𝔊 ↦SMCF.tiny op-smcf 𝔉 : op-smc 𝔄 ↦↦SMC.tinyα op-smc 𝔅
  by (intro is-tiny-ntsmcfI′)
    (cs-concl cs-shallow cs-intro: smc-cs-intros smc-op-intros)+

lemma (in is-tiny-ntsmcf) is-tiny-ntsmcf-op′[smc-op-intros]:
  assumes 𝔊′ = op-smcf 𝔊
    and 𝔉′ = op-smcf 𝔉
    and 𝔄′ = op-smc 𝔄
    and 𝔅′ = op-smc 𝔅
  shows op-ntsmcf 𝔑 : 𝔊′ ↦SMCF.tiny 𝔉′ : 𝔄′ ↦↦SMC.tinyα 𝔅′
  unfolding assms by (rule is-tm-ntsmcf-op)

lemmas is-tiny-ntsmcf-op[smc-op-intros] = is-tiny-ntsmcf.is-tiny-ntsmcf-op′
```

Vertical composition of tiny natural transformations of semifunctors

lemma *ntsmcf-vcomp-is-tiny-ntsmcf*[*smc-small-cs-intros*]:
assumes $\mathfrak{M} : \mathfrak{G} \mapsto_{SMCF.tiny} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}$
and $\mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{G} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}$
shows $\mathfrak{M} \bullet_{NTSMCF} \mathfrak{N} : \mathfrak{F} \mapsto_{SMCF.tiny} \mathfrak{H} : \mathfrak{A} \mapsto\mapsto_{SMC.tiny\alpha} \mathfrak{B}$
proof–
interpret $\mathfrak{M}$: *is-tiny-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{G}$ $\mathfrak{H}$ $\mathfrak{M}$ **by** (*rule assms*(1))
interpret $\mathfrak{N}$: *is-tiny-ntsmcf* α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(2))
show *?thesis* **by** (*rule is-tiny-ntsmcfI'*) (*auto intro*: *smc-small-cs-intros*)
qed

4.8 Product semicategory

4.8.1 Background

The concept of a product semicategory, as presented in this work, is a generalization of the concept of a product category, as presented in Chapter II-3 in [39].

named-theorems *smc-prod-cs-simps*
named-theorems *smc-prod-cs-intros*

4.8.2 Product semicategory: definition and elementary properties

definition *smc-prod* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  where *smc-prod* I $\mathfrak{A}$ =
    [
      $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!))$,
      $(\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!))$,
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Dom|\!))(\!|f(\!|i|\!)|\!)))$,
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Cod|\!))(\!|f(\!|i|\!)|\!)))$,
      $(\lambda gf \in_{\circ} composable\text{-}arrs\ (dg\text{-}prod\ I\ \mathfrak{A}).\ (\lambda i \in_{\circ} I.\ gf(\!|0|\!)(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ gf(\!|1_{\mathbb{N}}|\!)(\!|i|\!)))$
    ]$_{\circ}$

syntax *-PSEMICATEGORY* :: *pttrn* $\Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
  ($\langle$($3\prod_{SMC}$-$\in_{\circ}$-./ -)$\rangle$ [0, 0, 10] 10)
syntax-consts *-PSEMICATEGORY* $\rightleftharpoons$ *smc-prod*
translations $\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A} \rightleftharpoons CONST\ smc\text{-}prod\ I\ (\lambda i.\ \mathfrak{A})$

Components.

lemma *smc-prod-components*:
  shows $(\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Obj|\!) = (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Obj|\!))$
    and $(\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Arr|\!) = (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!))$
    and $(\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Dom|\!) =$
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Dom|\!))(\!|f(\!|i|\!)|\!)))$
    and $(\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Cod|\!) =$
      $(\lambda f \in_{\circ} (\prod_{\circ} i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Arr|\!)).\ (\lambda i \in_{\circ} I.\ \mathfrak{A}\ i(\!|Cod|\!))(\!|f(\!|i|\!)|\!)))$
    and $(\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)(\!|Comp|\!) =$
      $(\lambda gf \in_{\circ} composable\text{-}arrs\ (dg\text{-}prod\ I\ \mathfrak{A}).\ (\lambda i \in_{\circ} I.\ gf(\!|0|\!)(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ gf(\!|1_{\mathbb{N}}|\!)(\!|i|\!)))$
  unfolding *smc-prod-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-smc-prod*[*slicing-commute*]:
  dg-prod I $(\lambda i.\ smc\text{-}dg\ (\mathfrak{A}\ i)) = smc\text{-}dg\ (smc\text{-}prod\ I\ \mathfrak{A})$
  unfolding *dg-prod-def smc-dg-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** $\mathfrak{A}$=$\langle\lambda i.\ smc\text{-}dg\ (\mathfrak{A}\ i)\rangle$, *unfolded slicing-simps slicing-commute*]:
  smc-prod-ObjI = *dg-prod-ObjI*
  and *smc-prod-ObjD* = *dg-prod-ObjD*
  and *smc-prod-ObjE* = *dg-prod-ObjE*
  and *smc-prod-Obj-cong* = *dg-prod-Obj-cong*
  and *smc-prod-ArrI* = *dg-prod-ArrI*
  and *smc-prod-ArrD* = *dg-prod-ArrD*

and *smc-prod-ArrE* = *dg-prod-ArrE*
and *smc-prod-Arr-cong* = *dg-prod-Arr-cong*
and *smc-prod-Dom-vsv*[*smc-cs-intros*] = *dg-prod-Dom-vsv*
and *smc-prod-Dom-vdomain*[*smc-cs-simps*] = *dg-prod-Dom-vdomain*
and *smc-prod-Dom-app* = *dg-prod-Dom-app*
and *smc-prod-Dom-app-component-app*[*smc-cs-simps*] =
dg-prod-Dom-app-component-app
and *smc-prod-Cod-vsv*[*smc-cs-intros*] = *dg-prod-Cod-vsv*
and *smc-prod-Cod-app* = *dg-prod-Cod-app*
and *smc-prod-Cod-vdomain*[*smc-cs-simps*] = *dg-prod-Cod-vdomain*
and *smc-prod-Cod-app-component-app*[*smc-cs-simps*] =
dg-prod-Cod-app-component-app
and *smc-prod-vunion-Obj-in-Obj* = *dg-prod-vunion-Obj-in-Obj*
and *smc-prod-vdiff-vunion-Obj-in-Obj* = *dg-prod-vdiff-vunion-Obj-in-Obj*
and *smc-prod-vunion-Arr-in-Arr* = *dg-prod-vunion-Arr-in-Arr*
and *smc-prod-vdiff-vunion-Arr-in-Arr* = *dg-prod-vdiff-vunion-Arr-in-Arr*

end

lemma *smc-prod-dg-prod-is-arr*:
$g : b \mapsto_{\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i}\ c \longleftrightarrow g : b \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ c$
unfolding *is-arr-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 $(\prod_{DG} i \in_\circ I.\ \mathfrak{A}\ i)$ = *composable-arrs* $(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)$
unfolding *composable-arrs-def smc-prod-dg-prod-is-arr* **by** *simp*

4.8.3 Local assumptions for a product semicategory

locale *psemicategory-base* = $\mathcal{Z}\ \alpha$ **for** $\alpha\ I\ \mathfrak{A}$ +
assumes *psmc-semicategories*[*smc-prod-cs-intros*]:
$i \in_\circ I \Longrightarrow$ *semicategory* $\alpha\ (\mathfrak{A}\ i)$
and *psmc-index-in-Vset*[*smc-cs-intros*]: $I \in_\circ$ *Vset* α

Rules.

lemma (**in** *psemicategory-base*) *psemicategory-base-axioms'*[*smc-prod-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $I' = I$
shows *psemicategory-base* $\alpha'\ I'\ \mathfrak{A}$
unfolding *assms* **by** (*rule psemicategory-base-axioms*)

mk-ide rf *psemicategory-base-def*[*unfolded psemicategory-base-axioms-def*]
|*intro psemicategory-baseI*|
|*dest psemicategory-baseD*[*dest*]|
|*elim psemicategory-baseE*[*elim*]|

lemma *psemicategory-base-pdigraph-baseI*:
assumes *pdigraph-base* $\alpha\ I\ (\lambda i.\ \textit{smc-dg}\ (\mathfrak{A}\ i))$
and $\bigwedge i.\ i \in_\circ I \Longrightarrow$ *semicategory* $\alpha\ (\mathfrak{A}\ i)$
shows *psemicategory-base* $\alpha\ I\ \mathfrak{A}$
proof–
interpret *pdigraph-base* $\alpha\ I\ \langle\lambda i.\ \textit{smc-dg}\ (\mathfrak{A}\ i)\rangle$
rewrites *smc-dg* $\mathfrak{C}'(\lfloor Obj \rfloor) = \mathfrak{C}'(\lfloor Obj \rfloor)$ **and** *smc-dg* $\mathfrak{C}'(\lfloor Arr \rfloor) = \mathfrak{C}'(\lfloor Arr \rfloor)$ **for** $\mathfrak{C}'$
by (*rule assms*(1)) (*simp-all add*: *slicing-simps*)
show *?thesis*
by (*intro psemicategory-baseI*)
(*auto simp*: *assms*(2) *pdg-index-in-Vset pdg-Obj-in-Vset pdg-Arr-in-Vset*)
qed

Product semicategory is a product digraph.

context *psemicategory-base*
begin

lemma *psmc-pdigraph-base*: *pdigraph-base* α I $(\lambda i.\ smc\text{-}dg\ (\mathfrak{A}\ i))$
proof(*intro pdigraph-baseI*)
show *digraph* α $(smc\text{-}dg\ (\mathfrak{A}\ i))$ **if** $i \in_\circ I$ **for** i
by
(
use that **in**
‹*cs-concl cs-shallow cs-intro*: *slicing-intros smc-prod-cs-intros*›
)
show $I \in_\circ Vset\ \alpha$ **by** (*cs-concl* **cs-shallow cs-intro**: *smc-cs-intros*)
qed *auto*

interpretation *pdg*: *pdigraph-base* α I ‹$\lambda i.\ smc\text{-}dg\ (\mathfrak{A}\ i)$›
by (*rule psmc-pdigraph-base*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
psmc-Obj-in-Vset = *pdg.pdg-Obj-in-Vset*
and *psmc-Arr-in-Vset* = *pdg.pdg-Arr-in-Vset*
and *psmc-smc-prod-Obj-in-Vset* = *pdg.pdg-dg-prod-Obj-in-Vset*
and *psmc-smc-prod-Arr-in-Vset* = *pdg.pdg-dg-prod-Arr-in-Vset*
and *smc-prod-Dom-app-in-Obj*[*smc-cs-intros*] = *pdg.dg-prod-Dom-app-in-Obj*
and *smc-prod-Cod-app-in-Obj*[*smc-cs-intros*] = *pdg.dg-prod-Cod-app-in-Obj*
and *smc-prod-is-arrI* = *pdg.dg-prod-is-arrI*
and *smc-prod-is-arrD*[*dest*] = *pdg.dg-prod-is-arrD*
and *smc-prod-is-arrE*[*elim*] = *pdg.dg-prod-is-arrE*

end

lemmas [*smc-cs-intros*] = *psemicategory-base.smc-prod-is-arrD*(7)

Elementary properties.

lemma (**in** *psemicategory-base*) *psmc-vsubset-index-psemicategory-base*:
assumes $J \subseteq_\circ I$
shows *psemicategory-base* α J $\mathfrak{A}$
proof(*intro psemicategory-baseI*)
show *semicategory* α $(\mathfrak{A}\ i)$ **if** $i \in_\circ J$ **for** i
using *that assms* **by** (*auto intro*: *smc-prod-cs-intros*)
from *assms* **show** $J \in_\circ Vset\ \alpha$ **by** (*simp add*: *vsubset-in-VsetI smc-cs-intros*)
qed *auto*

Composition

lemma *smc-prod-Comp*:
$(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Comp|\!) =$
(
$\lambda gf \in_\circ composable\text{-}arrs\ (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i).$
$(\lambda i \in_\circ I.\ gf(\!|0|\!)(\!|i|\!) \circ_{A\mathfrak{A}\ i} gf(\!|1_{\mathbb{N}}|\!)(\!|i|\!))$
)
unfolding *smc-prod-components smc-prod-composable-arrs-dg-prod* **by** *simp*

lemma *smc-prod-Comp-vdomain*[*smc-cs-simps*]:
$\mathcal{D}_\circ\ ((\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Comp|\!)) = composable\text{-}arrs\ (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)$
unfolding *smc-prod-Comp* **by** *simp*

lemma *smc-prod-Comp-app*:

assumes $g : b \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ c$ **and** $f : a \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ b$
shows $g \circ_{A(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)}\ f = (\lambda i \in_\circ I.\ g(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ f(\!|i|\!))$
proof–
from *assms* **have** $[g, f]_\circ \in_\circ$ *composable-arrs* $(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)$
by (*auto intro*: *smc-cs-intros*)
then show *?thesis* **unfolding** *smc-prod-Comp* **by** (*auto simp*: *nat-omega-simps*)
qed

lemma *smc-prod-Comp-app-component*[*smc-cs-simps*]:
assumes $g : b \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ c$
and $f : a \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ b$
and $i \in_\circ I$
shows $(g \circ_{A(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)}\ f)(\!|i|\!) = g(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ f(\!|i|\!)$
using *assms*(3) **unfolding** *smc-prod-Comp-app*[*OF assms*(1,2)] **by** *simp*

lemma (**in** *psemicategory-base*) *smc-prod-Comp-vrange*:
$\mathcal{R}_\circ\ ((\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Comp|\!)) \subseteq_\circ (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
proof(*intro vsubsetI*)
fix h **assume** *prems*: $h \in_\circ \mathcal{R}_\circ\ ((\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Comp|\!))$
then obtain *gf*
where *h-def*: $h = (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Comp|\!)(\!|gf|\!)$
and $gf \in_\circ$ *composable-arrs* $(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)$
by (*auto simp*: *smc-prod-Comp intro*: *smc-cs-intros*)
then obtain $g\ f\ a\ b\ c$
where *gf-def*: $gf = [g, f]_\circ$
and *g*: $g : b \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ c$
and *f*: $f : a \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ b$
by *clarsimp*
from $g\ f$ **have** *gf-comp*: $g \circ_{A(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)}\ f = (\lambda i \in_\circ I.\ g(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ f(\!|i|\!))$
by (*rule smc-prod-Comp-app*)
show $h \in_\circ (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|Arr|\!)$
unfolding *smc-prod-components*
unfolding *h-def gf-def gf-comp*
proof(*rule VLambda-in-vproduct*)
fix i **assume** *prems*: $i \in_\circ I$
interpret *semicategory* α ‹$\mathfrak{A}\ i$›
using *prems* **by** (*simp add*: *smc-prod-cs-intros*)
from *prems smc-prod-is-arrD*(7)[*OF g*] *smc-prod-is-arrD*(7)[*OF f*] **have**
$g(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ f(\!|i|\!) : a(\!|i|\!) \mapsto_{\mathfrak{A}\ i}\ c(\!|i|\!)$
by (*auto intro*: *smc-cs-intros*)
then show $g(\!|i|\!) \circ_{A\mathfrak{A}\ i}\ f(\!|i|\!) \in_\circ \mathfrak{A}\ i(\!|Arr|\!)$ **by** (*simp add*: *smc-cs-intros*)
qed
qed

lemma *smc-prod-Comp-app-vdomain*[*smc-cs-simps*]:
assumes $g : b \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ c$ **and** $f : a \mapsto_{\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i}\ b$
shows $\mathcal{D}_\circ\ (g \circ_{A(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)}\ f) = I$
unfolding *smc-prod-Comp-app*[*OF assms*] **by** *simp*

A product α-semicategory is a tiny β-semicategory

lemma (**in** *psemicategory-base*) *psmc-tiny-semicategory-smc-prod*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows *tiny-semicategory* β $(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)$
proof(*intro tiny-semicategoryI*, (*unfold slicing-simps*)*?*)

show *tiny-digraph* β (*smc-dg* (*smc-prod* $I\ \mathfrak{A}$))
unfolding *slicing-commute*[*symmetric*]

```
  by
    (
      intro pdigraph-base.pdg-tiny-digraph-dg-prod;
      (rule assms psmc-pdigraph-base)?
    )

show vfsequence (∏ SMC i∈∘ I. 𝔄 i) unfolding smc-prod-def by auto
show vcard (∏ SMC i∈∘ I. 𝔄 i) = 5ℕ
  unfolding smc-prod-def by (simp add: nat-omega-simps)
show vsv ((∏ SMC i∈∘ I. 𝔄 i)(|Comp|)) unfolding smc-prod-Comp by simp

show
  (gf ∈∘ 𝒟∘ ((∏ SMC i∈∘ I. 𝔄 i)(|Comp|))) ⟷
    (
      ∃ g f b c a.
        gf = [g, f]∘ ∧ g : b ↦smc-prod I 𝔄 c ∧ f : a ↦smc-prod I 𝔄 b
    )
  for gf
  by (auto intro: smc-cs-intros simp: smc-cs-simps)

show Comp-is-arr[intro]: g ∘A(∏ SMC i∈∘ I. 𝔄 i) f : a ↦∏ SMC i∈∘ I. 𝔄 i c
  if g : b ↦∏ SMC i∈∘ I. 𝔄 i c and f : a ↦∏ SMC i∈∘ I. 𝔄 i b
  for g b c f a
proof(intro smc-prod-is-arrI)
  from that show vsv (g ∘A smc-prod I 𝔄 f)
    by (auto simp: smc-prod-Comp-app)
  from that show 𝒟∘ (g ∘A smc-prod I 𝔄 f) = I
    by (auto simp: smc-prod-Comp-app)
  from that(2) have f: f ∈∘ (∏ SMC i∈∘ I. 𝔄 i)(|Arr|)
    by (elim is-arrE) (auto simp: smc-prod-components)
  from that(1) have g: g ∈∘ (∏ SMC i∈∘ I. 𝔄 i)(|Arr|)
    by (elim is-arrE) (auto simp: smc-prod-components)
  from f have a: a ∈∘ (∏ SMC i∈∘ I. 𝔄 i)(|Obj|)
    by (rule smc-prod-Dom-app-in-Obj[of f, unfolded is-arrD(2)[OF that(2)]])
  then show vsv a by (auto simp: smc-prod-components)
  from a show 𝒟∘ a = I by (auto simp: smc-prod-components)
  from g have c: c ∈∘ (∏ SMC i∈∘ I. 𝔄 i)(|Obj|)
    by (rule smc-prod-Cod-app-in-Obj[of g, unfolded is-arrD(3)[OF that(1)]])
  then show vsv c by (auto simp: smc-prod-components)
  from c show 𝒟∘ c = I by (auto simp: smc-prod-components)
  fix i assume prems: i ∈∘ I
  interpret semicategory α ‹𝔄 i›
    using prems by (auto intro: smc-prod-cs-intros)
  from
    prems
    smc-prod-is-arrD(7)[OF that(1) prems]
    smc-prod-is-arrD(7)[OF that(2) prems]
  show (g ∘A smc-prod I 𝔄 f)(|i|) : a(|i|) ↦𝔄 i c(|i|)
    unfolding smc-prod-Comp-app[OF that] by (auto intro: smc-cs-intros)
qed

show
  h ∘A smc-prod I 𝔄 g ∘A smc-prod I 𝔄 f =
    h ∘A smc-prod I 𝔄 (g ∘A smc-prod I 𝔄 f)
  if h : c ↦smc-prod I 𝔄 d
    and g : b ↦smc-prod I 𝔄 c
    and f : a ↦smc-prod I 𝔄 b
```

```
    for h c d g b f a
  proof(rule smc-prod-Arr-cong)
    show (h ∘_A smc-prod I 𝔄 g) ∘_A smc-prod I 𝔄 f ∈_∘ (∏_SMC i∈_∘ I. 𝔄 i)(|Arr|)
      by (meson that Comp-is-arr is-arrD)
    show h ∘_A smc-prod I 𝔄 (g ∘_A smc-prod I 𝔄 f) ∈_∘ smc-prod I 𝔄(|Arr|)
      by (meson that Comp-is-arr is-arrD)
    fix i assume prems: i ∈_∘ I
    then interpret semicategory α ‹𝔄 i› by (simp add: smc-prod-cs-intros)
    from prems that have h(|i|) : c(|i|) ↦_𝔄 i d(|i|)
      and g(|i|) : b(|i|) ↦_𝔄 i c(|i|)
      and f(|i|) : a(|i|) ↦_𝔄 i b(|i|)
      and h ∘_A smc-prod I 𝔄 g : b ↦_smc-prod I 𝔄 d
      and g ∘_A smc-prod I 𝔄 f : a ↦_smc-prod I 𝔄 c
      by (auto simp: smc-prod-is-arrD)
    with prems that show
      (h ∘_A smc-prod I 𝔄 g ∘_A smc-prod I 𝔄 f)(|i|) =
        (h ∘_A smc-prod I 𝔄 (g ∘_A smc-prod I 𝔄 f))(|i|)
      by (simp add: smc-prod-Comp-app-component smc-Comp-assoc)
  qed

qed (intro assms)
```

4.8.4 Further local assumptions for product semicategories

Definition and elementary properties

```
locale psemicategory = psemicategory-base α I 𝔄 for α I 𝔄 +
  assumes psmc-Obj-vsubset-Vset:
    J ⊆_∘ I ⟹ (∏_SMC i∈_∘ J. 𝔄 i)(|Obj|) ⊆_∘ Vset α
    and psmc-Hom-vifunion-in-Vset:
      ⟦
        J ⊆_∘ I;
        A ⊆_∘ (∏_SMC i∈_∘ J. 𝔄 i)(|Obj|);
        B ⊆_∘ (∏_SMC i∈_∘ J. 𝔄 i)(|Obj|);
        A ∈_∘ Vset α;
        B ∈_∘ Vset α
      ⟧ ⟹ (⋃_∘ a∈_∘ A. ⋃_∘ b∈_∘ B. Hom (∏_SMC i∈_∘ J. 𝔄 i) a b) ∈_∘ Vset α
```

Rules.

```
lemma (in psemicategory) psemicategory-axioms'[smc-prod-cs-intros]:
  assumes α' = α and I' = I
  shows psemicategory α' I' 𝔄
  unfolding assms by (rule psemicategory-axioms)

mk-ide rf psemicategory-def[unfolded psemicategory-axioms-def]
  |intro psemicategoryI|
  |dest psemicategoryD[dest]|
  |elim psemicategoryE[elim]|

lemmas [smc-prod-cs-intros] = psemicategoryD(1)

lemma psemicategory-pdigraphI:
  assumes pdigraph α I (λi. smc-dg (𝔄 i))
    and ⋀i. i ∈_∘ I ⟹ semicategory α (𝔄 i)
  shows psemicategory α I 𝔄
proof–
  interpret pdigraph α I ‹λi. smc-dg (𝔄 i)› by (rule assms(1))
  note [unfolded slicing-simps slicing-commute, smc-cs-intros] =
```

pdg-Obj-vsubset-Vset
pdg-Hom-vifunion-in-Vset
show *?thesis*
by (*intro psemicategoryI psemicategory-base-pdigraph-baseI*)
(*auto simp*: *assms*(2) *dg-prod-cs-intros intro*!: *smc-cs-intros*)
qed

Product semicategory is a product digraph.

context *psemicategory*
begin

lemma *psmc-pdigraph*: *pdigraph* α *I* (λi. *smc-dg* ($\mathfrak{A}$ *i*))
proof(*intro pdigraphI*, *unfold slicing-simps slicing-commute*)
show *pdigraph-base* α *I* (λi. *smc-dg* ($\mathfrak{A}$ *i*)) **by** (*rule psmc-pdigraph-base*)
qed (*auto intro*!: *psmc-Obj-vsubset-Vset psmc-Hom-vifunion-in-Vset*)

interpretation *pdg*: *pdigraph* α *I* ‹λi. *smc-dg* ($\mathfrak{A}$ *i*)› **by** (*rule psmc-pdigraph*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
psmc-Obj-vsubset-Vset' = *pdg.pdg-Obj-vsubset-Vset'*
and *psmc-Hom-vifunion-in-Vset'* = *pdg.pdg-Hom-vifunion-in-Vset'*
and *psmc-smc-prod-vunion-is-arr* = *pdg.pdg-dg-prod-vunion-is-arr*
and *psmc-smc-prod-vdiff-vunion-is-arr* = *pdg.pdg-dg-prod-vdiff-vunion-is-arr*

end

Elementary properties.

lemma (**in** *psemicategory*) *psmc-vsubset-index-psemicategory*:
assumes $J \subseteq_\circ I$
shows *psemicategory* α *J* $\mathfrak{A}$
proof(*intro psemicategoryI psemicategory-pdigraphI*)
show *smc-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 *smc-prod J'* $\mathfrak{A}$(|*Obj*|) $\subseteq_\circ$ *Vset* α **by** (*rule psmc-Obj-vsubset-Vset*)
qed
fix *A B J'* **assume** *prems*:
$J' \subseteq_\circ J$
$A \subseteq_\circ (\prod_{SMC} i \in_\circ J'.\ \mathfrak{A}\ i)$(|*Obj*|)
$B \subseteq_\circ (\prod_{SMC} i \in_\circ J'.\ \mathfrak{A}\ i)$(|*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* $(\prod_{SMC} i \in_\circ J'.\ \mathfrak{A}\ i)\ a\ b) \in_\circ$ *Vset* α
proof–
from *prems*(1) *assms* **have** $J' \subseteq_\circ I$ **by** *simp*
from *psmc-Hom-vifunion-in-Vset*[*OF this prems*(2–5)] **show** *?thesis*.
qed
qed (*rule psmc-vsubset-index-psemicategory-base*[*OF assms*])

A product α-semicategory is an α-semicategory

lemma (**in** *psemicategory*) *psmc-semicategory-smc-prod*:
semicategory α $(\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)$
proof–
interpret *tiny-semicategory* ‹$\alpha + \omega$› ‹$\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i$›
by (*intro psmc-tiny-semicategory-smc-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 semicategory-if-semicategory)
     (
       auto
         intro!: psmc-Hom-vifunion-in-Vset psmc-Obj-vsubset-Vset
         intro: smc-cs-intros
     )
qed
```

4.8.5 Local assumptions for a finite product semicategory

Definition and elementary properties

```
locale finite-psemicategory = psemicategory-base α I 𝔄 for α I 𝔄 +
  assumes fin-psmc-index-vfinite: vfinite I
```

Rules.

```
lemma (in finite-psemicategory) finite-psemicategory-axioms[smc-prod-cs-intros]:
  assumes α' = α and I' = I
  shows finite-psemicategory α' I' 𝔄
  unfolding assms by (rule finite-psemicategory-axioms)

mk-ide rf finite-psemicategory-def[unfolded finite-psemicategory-axioms-def]
  |intro finite-psemicategoryI|
  |dest finite-psemicategoryD[dest]|
  |elim finite-psemicategoryE[elim]|

lemmas [smc-prod-cs-intros] = finite-psemicategoryD(1)

lemma finite-psemicategory-finite-pdigraphI:
  assumes finite-pdigraph α I (λi. smc-dg (𝔄 i))
    and ⋀i. i ∈₀ I ⟹ semicategory α (𝔄 i)
  shows finite-psemicategory α I 𝔄
proof–
  interpret finite-pdigraph α I ‹λi. smc-dg (𝔄 i)› by (rule assms(1))
  show ?thesis
    by
     (
       intro
         assms
         finite-psemicategoryI
         psemicategory-base-pdigraph-baseI
         finite-pdigraphD(1)[OF assms(1)]
         fin-pdg-index-vfinite
     )
qed
```

Local assumptions for a finite product semicategory and local assumptions for an arbitrary product semicategory

```
sublocale finite-psemicategory ⊆ psemicategory α I 𝔄
proof–
  interpret finite-pdigraph α I ‹λi. smc-dg (𝔄 i)›
  proof(intro finite-pdigraphI pdigraph-baseI)
    fix i assume i: i ∈₀ I
    interpret 𝔄i: semicategory α ‹𝔄 i› by (simp add: i psmc-semicategories)
    show digraph α (smc-dg (𝔄 i)) by (simp add: 𝔄i.smc-digraph)
  qed (auto intro!: smc-cs-intros fin-psmc-index-vfinite)
  show psemicategory α I 𝔄
```

by (*intro psemicategory-pdigraphI*)
(*simp-all add*: *psmc-semicategories pdigraph-axioms*)
qed

4.8.6 Binary union and complement

lemma (**in** *psemicategory*) *psmc-smc-prod-vunion-Comp*:
assumes *vdisjnt J K*
and $J \subseteq_\circ I$
and $K \subseteq_\circ I$
and $g : b \mapsto_{(\prod_{SMC} j \in_\circ J.\ \mathfrak{A}\ j)}\ c$
and $g' : b' \mapsto_{(\prod_{SMC} k \in_\circ K.\ \mathfrak{A}\ k)}\ c'$
and $f : a \mapsto_{(\prod_{SMC} j \in_\circ J.\ \mathfrak{A}\ j)}\ b$
and $f' : a' \mapsto_{(\prod_{SMC} k \in_\circ K.\ \mathfrak{A}\ k)}\ b'$
shows $(g \circ_{A(\prod_{SMC} j \in_\circ J.\ \mathfrak{A}\ j)} f) \cup_\circ (g' \circ_{A(\prod_{SMC} j \in_\circ K.\ \mathfrak{A}\ j)} f') =$
$g \cup_\circ g' \circ_{A(\prod_{SMC} j \in_\circ J \cup_\circ K.\ \mathfrak{A}\ j)} f \cup_\circ f'$
proof–

interpret $J\mathfrak{A}$: *psemicategory* α J $\mathfrak{A}$
using *assms*(2) **by** (*simp add*: *psmc-vsubset-index-psemicategory*)
interpret $K\mathfrak{A}$: *psemicategory* α K $\mathfrak{A}$
using *assms*(3) **by** (*simp add*: *psmc-vsubset-index-psemicategory*)
interpret $JK\mathfrak{A}$: *psemicategory* α ‹$J \cup_\circ K$› $\mathfrak{A}$
using *assms*(2,3) **by** (*simp add*: *psmc-vsubset-index-psemicategory*)

interpret $J\mathfrak{A}'$: *semicategory* α ‹*smc-prod J* $\mathfrak{A}$›
by (*rule J*$\mathfrak{A}$.*psmc-semicategory-smc-prod*)
interpret $K\mathfrak{A}'$: *semicategory* α ‹*smc-prod K* $\mathfrak{A}$›
by (*rule K*$\mathfrak{A}$.*psmc-semicategory-smc-prod*)
interpret $JK\mathfrak{A}'$: *semicategory* α ‹*smc-prod* ($J \cup_\circ K$) $\mathfrak{A}$›
by (*rule JK*$\mathfrak{A}$.*psmc-semicategory-smc-prod*)

note gg' = *psmc-smc-prod-vunion-is-arr*[*OF assms*(1–3,4,5)]
and ff' = *psmc-smc-prod-vunion-is-arr*[*OF assms*(1–3,6,7)]

note gD = $J\mathfrak{A}$.*smc-prod-is-arrD*[*OF assms*(4)]
and $g'D$ = $K\mathfrak{A}$.*smc-prod-is-arrD*[*OF assms*(5)]
and fD = $J\mathfrak{A}$.*smc-prod-is-arrD*[*OF assms*(6)]
and $f'D$ = $K\mathfrak{A}$.*smc-prod-is-arrD*[*OF assms*(7)]

from *assms*(4,6) **have** *gf*:
$g \circ_{A\,smc\text{-}prod\ J\ \mathfrak{A}} f : a \mapsto_{(\prod_{SMC} j \in_\circ J.\ \mathfrak{A}\ j)}\ c$
by (*auto intro*: *smc-cs-intros*)
from *assms*(5,7) **have** $g'f'$:
$g' \circ_{A\,smc\text{-}prod\ K\ \mathfrak{A}} f' : a' \mapsto_{(\prod_{SMC} k \in_\circ K.\ \mathfrak{A}\ k)}\ c'$
by (*auto intro*: *smc-cs-intros*)
from *gf* **have** $g \circ_{A\,smc\text{-}prod\ J\ \mathfrak{A}} f \in_\circ$ *smc-prod J* $\mathfrak{A}(\!|Arr|\!)$ **by** *auto*
from $g'f'$ **have** $g' \circ_{A\,smc\text{-}prod\ K\ \mathfrak{A}} f' \in_\circ$ *smc-prod K* $\mathfrak{A}(\!|Arr|\!)$ **by** *auto*
from gg' ff' **have** gg'-ff':
$g \cup_\circ g' \circ_{A\,smc\text{-}prod\ (J \cup_\circ K)\ \mathfrak{A}} f \cup_\circ f' :$
$a \cup_\circ a' \mapsto_{smc\text{-}prod\ (J \cup_\circ K)\ \mathfrak{A}}\ c \cup_\circ c'$
by (*simp add*: *smc-cs-intros*)

show *?thesis*
proof(*rule smc-prod-Arr-cong*[*of* - ‹$J \cup_\circ K$› $\mathfrak{A}$])
from *gf* $g'f'$ *assms*(1) **show**

```
    (g ∘_A smc-prod J 𝔄 f) ∪∘ (g′ ∘_A smc-prod K 𝔄 f′) ∈∘
      smc-prod (J ∪∘ K) 𝔄(|Arr|)
    by (auto intro: smc-prod-vunion-Arr-in-Arr)
  from gg′-ff′ show
    g ∪∘ g′ ∘_A smc-prod (J ∪∘ K) 𝔄 f ∪∘ f′ ∈∘ smc-prod (J ∪∘ K) 𝔄(|Arr|)
    by auto
  fix i assume prems: i ∈∘ J ∪∘ K
  then consider (iJ) ‹i ∈∘ J› | (iK) ‹i ∈∘ K› by auto
  then show
    ((g ∘_A smc-prod J 𝔄 f) ∪∘ (g′ ∘_A smc-prod K 𝔄 f′))(|i|) =
      (g ∪∘ g′ ∘_A smc-prod (J ∪∘ K) 𝔄 f ∪∘ f′)(|i|)
  proof cases
    case iJ
    have [simp]:
      ((g ∘_A smc-prod J 𝔄 f) ∪∘ (g′ ∘_A smc-prod K 𝔄 f′))(|i|) =
        g(|i|) ∘_A𝔄 i f(|i|)
    proof
      (
        fold smc-prod-Comp-app-component[OF assms(4,6) iJ],
        rule vsv-vunion-app-left
      )
      from gf show vsv (g ∘_A smc-prod J 𝔄 f) by auto
      from g′f′ show vsv (g′ ∘_A smc-prod K 𝔄 f′) by auto
    qed
      (
        use assms(4–7) in
          ‹simp-all add: iJ assms(1) smc-prod-Comp-app-vdomain›
      )
    have gg′-i: (g ∪∘ g′)(|i|) = g(|i|)
      by (simp add: iJ assms(1) gD(1,2) g′D(1,2))
    have ff′-i: (f ∪∘ f′)(|i|) = f(|i|)
      by (simp add: iJ assms(1) fD(1,2) f′D(1,2))
    have [simp]:
      (g ∪∘ g′ ∘_A smc-prod (J ∪∘ K) 𝔄 f ∪∘ f′)(|i|) = g(|i|) ∘_A𝔄 i f(|i|)
      by (fold gg′-i ff′-i)
        (rule smc-prod-Comp-app-component[OF gg′ ff′ prems])
    show ?thesis by simp
  next
    case iK
    have [simp]:
      ((g ∘_A smc-prod J 𝔄 f) ∪∘ (g′ ∘_A smc-prod K 𝔄 f′))(|i|) =
        g′(|i|) ∘_A𝔄 i f′(|i|)
    proof
      (
        fold smc-prod-Comp-app-component[OF assms(5,7) iK],
        rule vsv-vunion-app-right
      )
      from gf show vsv (g ∘_A smc-prod J 𝔄 f) by auto
      from g′f′ show vsv (g′ ∘_A smc-prod K 𝔄 f′) by auto
    qed
      (
        use assms(4–7) in
          ‹simp-all add: iK smc-prod-Comp-app-vdomain assms(1)›
      )
    have gg′-i: (g ∪∘ g′)(|i|) = g′(|i|)
      by (simp add: iK assms(1) gD(1,2) g′D(1,2))
    have ff′-i: (f ∪∘ f′)(|i|) = f′(|i|)
```

```
      by (simp add: iK assms(1) fD(1,2) f'D(1,2))
    have [simp]:
      (g ∪∘ g' ∘_A smc-prod (J ∪∘ K) 𝔄 f ∪∘ f')(⦇i⦈) = g'(⦇i⦈) ∘_A 𝔄 i f'(⦇i⦈)
      by (fold gg'-i ff'-i)
        (rule smc-prod-Comp-app-component[OF gg' ff' prems])
    show ?thesis by simp
  qed
qed

qed

lemma (in psemicategory) psmc-smc-prod-vdiff-vunion-Comp:
  assumes J ⊆∘ I
    and g : b ↦(∏_SMC j∈∘I -∘ J. 𝔄 j) c
    and g' : b' ↦(∏_SMC k∈∘J. 𝔄 k) c'
    and f : a ↦(∏_SMC j∈∘I -∘ J. 𝔄 j) b
    and f' : a' ↦(∏_SMC k∈∘J. 𝔄 k) b'
  shows (g ∘_A(∏_SMC j∈∘I -∘ J. 𝔄 j) f) ∪∘ (g' ∘_A(∏_SMC j∈∘J. 𝔄 j) f') =
    g ∪∘ g' ∘_A(∏_SMC j∈∘I. 𝔄 j) f ∪∘ f'
  by
    (
      vdiff-of-vunion'
        rule: psmc-smc-prod-vunion-Comp assms: assms(2–5) subset: assms(1)
    )
```

4.8.7 Projection

Definition and elementary properties

See Chapter II-3 in [39].

```
definition smcf-proj :: V ⇒ (V ⇒ V) ⇒ V ⇒ V (‹π_SMC›)
  where π_SMC I 𝔄 i =
    [
      (λa∈∘(∏∘i∈∘I. 𝔄 i(⦇Obj⦈)). a(⦇i⦈)),
      (λf∈∘(∏∘i∈∘I. 𝔄 i(⦇Arr⦈)). f(⦇i⦈)),
      (∏_SMC i∈∘I. 𝔄 i),
      𝔄 i
    ]∘
```

Components.

```
lemma smcf-proj-components:
  shows (π_SMC I 𝔄 i)(⦇ObjMap⦈) = (λa∈∘(∏∘i∈∘I. 𝔄 i(⦇Obj⦈)). a(⦇i⦈))
    and (π_SMC I 𝔄 i)(⦇ArrMap⦈) = (λf∈∘(∏∘i∈∘I. 𝔄 i(⦇Arr⦈)). f(⦇i⦈))
    and (π_SMC I 𝔄 i)(⦇HomDom⦈) = (∏_SMC i∈∘I. 𝔄 i)
    and (π_SMC I 𝔄 i)(⦇HomCod⦈) = 𝔄 i
  unfolding smcf-proj-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

Slicing

```
lemma smcf-dghm-smcf-proj[slicing-commute]:
  π_DG I (λi. smc-dg (𝔄 i)) i = smcf-dghm (π_SMC I 𝔄 i)
  unfolding
    smc-dg-def
    smcf-dghm-def
    smcf-proj-def
    dghm-proj-def
    smc-prod-def
```

dg-prod-def
dg-field-simps
dghm-field-simps
by (*simp add*: *nat-omega-simps*)

context *psemicategory*
begin

interpretation *pdg*: *pdigraph* α I ‹$\lambda i.$ *smc-dg* ($\mathfrak{A}$ i)› **by** (*rule psmc-pdigraph*)

lemmas-with [*unfolded slicing-simps slicing-commute*]:
smcf-proj-is-dghm = *pdg.pdg-dghm-proj-is-dghm*

end

Projection semifunctor is a semifunctor

lemma (**in** *psemicategory*) *psmc-smcf-proj-is-semifunctor*:
assumes $i \in_\circ I$
shows $\pi_{SMC}\ I\ \mathfrak{A}\ i : (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i) \mapsto\mapsto_{SMC\alpha} \mathfrak{A}\ i$
proof(*intro is-semifunctorI*)
show *vfsequence* ($\pi_{SMC}\ I\ \mathfrak{A}\ i$)
unfolding *smcf-proj-def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* ($\pi_{SMC}\ I\ \mathfrak{A}\ i$) = $4_{\mathbb{N}}$
unfolding *smcf-proj-def* **by** (*simp add*: *nat-omega-simps*)
interpret $\mathfrak{A}$: *semicategory* α ‹*smc-prod* I $\mathfrak{A}$›
by (*rule psmc-semicategory-smc-prod*)
interpret $\mathfrak{A}i$: *semicategory* α ‹$\mathfrak{A}$ i›
using *assms* **by** (*simp add*: *smc-prod-cs-intros*)
show $\pi_{SMC}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|g \circ_{A\,smc\text{-}prod\ I\ \mathfrak{A}} f|\!) =$
$\pi_{SMC}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{A}\ i} \pi_{SMC}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|f|\!)$
if $g : b \mapsto_{smc\text{-}prod\ I\ \mathfrak{A}} c$ **and** $f : a \mapsto_{smc\text{-}prod\ I\ \mathfrak{A}} b$ **for** $g\ b\ c\ f\ a$
proof–
from *that* **have** $g \circ_{A\,smc\text{-}prod\ I\ \mathfrak{A}} f : a \mapsto_{smc\text{-}prod\ I\ \mathfrak{A}} c$
by (*auto simp*: *smc-cs-intros*)
then have $g \circ_{A\,smc\text{-}prod\ I\ \mathfrak{A}} f \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|Arr|\!))$
unfolding *smc-prod-components*[*symmetric*] **by** *auto*
then have π-*gf*: $\pi_{SMC}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|g \circ_{A\,smc\text{-}prod\ I\ \mathfrak{A}} f|\!) = g(\!|i|\!) \circ_{A\mathfrak{A}\ i} f(\!|i|\!)$
unfolding *smcf-proj-components*
by (*simp add*: *smc-prod-Comp-app-component*[*OF that assms*])
from *that*(1) **have** *g*: $g \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|Arr|\!))$
unfolding *smc-prod-components*[*symmetric*] **by** *auto*
from *that*(2) **have** *f*: $f \in_\circ (\prod_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|Arr|\!))$
unfolding *smc-prod-components*[*symmetric*] **by** *auto*
from *g f* **have** πg-πf:
$\pi_{SMC}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|g|\!) \circ_{A\mathfrak{A}\ i} \pi_{SMC}\ I\ \mathfrak{A}\ i(\!|ArrMap|\!)(\!|f|\!) = g(\!|i|\!) \circ_{A\mathfrak{A}\ i} f(\!|i|\!)$
unfolding *smcf-proj-components* **by** *simp*
from π-*gf* πg-πf **show** *?thesis* **by** *simp*
qed
qed
(
auto simp:
smc-prod-cs-intros
assms
smcf-proj-components
psmc-semicategory-smc-prod
smcf-proj-is-dghm
)

lemma (in *psemicategory*) *psmc-smcf-proj-is-semifunctor'*:
assumes $i \in_{\circ} I$ **and** $\mathfrak{C} = (\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)$ **and** $\mathfrak{D} = \mathfrak{A}\ i$
shows $\pi_{SMC}\ I\ \mathfrak{A}\ i : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule psmc-smcf-proj-is-semifunctor*)

lemmas [*smc-cs-intros*] = *psemicategory.psmc-smcf-proj-is-semifunctor'*

4.8.8 Semicategory product universal property semifunctor

Definition and elementary properties

The following semifunctor is used in the proof of the universal property of the product semicategory later in this work.

definition *smcf-up* :: $V \Rightarrow (V \Rightarrow V) \Rightarrow V \Rightarrow (V \Rightarrow V) \Rightarrow V$
where *smcf-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ =
[
$(\lambda a \in_{\circ} \mathfrak{C}(\!|Obj|\!).\ (\lambda i \in_{\circ} I.\ \varphi\ i(\!|ObjMap|\!)(\!|a|\!)))$,
$(\lambda f \in_{\circ} \mathfrak{C}(\!|Arr|\!).\ (\lambda i \in_{\circ} I.\ \varphi\ i(\!|ArrMap|\!)(\!|f|\!)))$,
$\mathfrak{C}$,
$(\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)$
]$_{\circ}$

Components.

lemma *smcf-up-components*:
shows *smcf-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 *smcf-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 *smcf-up I* $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|HomDom|\!) = \mathfrak{C}$
and *smcf-up I* $\mathfrak{A}$ $\mathfrak{C}$ $\varphi(\!|HomCod|\!) = (\prod_{SMC} i \in_{\circ} I.\ \mathfrak{A}\ i)$
unfolding *smcf-up-def dghm-field-simps* **by** (*simp all add*: *nat-omega-simps*)

Slicing.

lemma *smcf-dghm-smcf-up*[*slicing-commute*]:
dghm-up I $(\lambda i.\ smc\text{-}dg\ (\mathfrak{A}\ i))$ $(smc\text{-}dg\ \mathfrak{C})$ $(\lambda i.\ smcf\text{-}dghm\ (\varphi\ i))$ =
smcf-dghm (*smcf-up I* $\mathfrak{A}$ $\mathfrak{C}$ φ)
unfolding
smc-dg-def
smcf-dghm-def
smcf-up-def
dghm-up-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 $\mathfrak{A}$=‹$\lambda i.\ smc\text{-}dg\ (\mathfrak{A}\ i)$› **and** φ=‹$\lambda i.\ smcf\text{-}dghm\ (\varphi\ i)$› **and** $\mathfrak{C}$=‹$smc\text{-}dg\ \mathfrak{C}$›,
unfolded slicing-simps slicing-commute
]:
smcf-up-ObjMap-vdomain[*smc-cs-simps*] = *dghm-up-ObjMap-vdomain*

and *smcf-up-ObjMap-app = dghm-up-ObjMap-app*
and *smcf-up-ObjMap-app-vdomain*[*smc-cs-simps*] = *dghm-up-ObjMap-app-vdomain*
and *smcf-up-ObjMap-app-component*[*smc-cs-simps*] = *dghm-up-ObjMap-app-component*
and *smcf-up-ArrMap-vdomain*[*smc-cs-simps*] = *dghm-up-ArrMap-vdomain*
and *smcf-up-ArrMap-app = dghm-up-ArrMap-app*
and *smcf-up-ArrMap-app-vdomain*[*smc-cs-simps*] = *dghm-up-ArrMap-app-vdomain*
and *smcf-up-ArrMap-app-component*[*smc-cs-simps*] = *dghm-up-ArrMap-app-component*

lemma *smcf-up-ObjMap-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{smcf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ObjMap}|\!)) \subseteq_\circ (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{Obj}|\!)$
proof
(
rule dghm-up-ObjMap-vrange
[
where $\mathfrak{A}$=‹$\lambda i.$ *smc-dg* ($\mathfrak{A}\ i$)›
and φ=‹$\lambda i.$ *smcf-dghm* ($\varphi\ i$)›
and $\mathfrak{C}$=‹*smc-dg* $\mathfrak{C}$›,
unfolded slicing-simps slicing-commute
]
)
fix i **assume** $i \in_\circ I$
then interpret *is-semifunctor* α $\mathfrak{C}$ ‹$\mathfrak{A}\ i$› ‹$\varphi\ i$› **by** (*rule assms*)
show *smcf-dghm* ($\varphi\ i$) : *smc-dg* $\mathfrak{C} \mapsto\mapsto_{DG\alpha}$ *smc-dg* ($\mathfrak{A}\ i$)
by (*rule smcf-is-dghm*)
qed

lemma *smcf-up-ObjMap-app-vrange*:
assumes $a \in_\circ \mathfrak{C}(\!|\mathit{Obj}|\!)$ **and** $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{smcf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ObjMap}|\!)(\!|a|\!)) \subseteq_\circ (\bigcup_\circ i \in_\circ I.\ \mathfrak{A}\ i(\!|\mathit{Obj}|\!))$
proof
(
rule dghm-up-ObjMap-app-vrange
[
where $\mathfrak{A}$=‹$\lambda i.$ *smc-dg* ($\mathfrak{A}\ i$)›
and φ=‹$\lambda i.$ *smcf-dghm* ($\varphi\ i$)›
and $\mathfrak{C}$=‹*smc-dg* $\mathfrak{C}$›,
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-semifunctor* α $\mathfrak{C}$ ‹$\mathfrak{A}\ i$› ‹$\varphi\ i$› **by** (*rule assms*(2))
show *smcf-dghm* ($\varphi\ i$) : *smc-dg* $\mathfrak{C} \mapsto\mapsto_{DG\alpha}$ *smc-dg* ($\mathfrak{A}\ i$)
by (*rule smcf-is-dghm*)
qed

lemma *smcf-up-ArrMap-vrange*:
assumes $\bigwedge i.\ i \in_\circ I \Longrightarrow \varphi\ i : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{A}\ i$
shows $\mathcal{R}_\circ\ (\mathit{smcf\text{-}up}\ I\ \mathfrak{A}\ \mathfrak{C}\ \varphi(\!|\mathit{ArrMap}|\!)) \subseteq_\circ (\prod_{SMC} i \in_\circ I.\ \mathfrak{A}\ i)(\!|\mathit{Arr}|\!)$
proof
(
rule dghm-up-ArrMap-vrange
[
where $\mathfrak{A}$=‹$\lambda i.$ *smc-dg* ($\mathfrak{A}\ i$)›
and φ=‹$\lambda i.$ *smcf-dghm* ($\varphi\ i$)›
and $\mathfrak{C}$=‹*smc-dg* $\mathfrak{C}$›,
unfolded slicing-simps slicing-commute

```
    ]
  )
  fix i assume i ∈ₒ I
  then interpret is-semifunctor α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms)
  show smcf-dghm (φ i) : smc-dg 𝔈 ↦↦_DGα smc-dg (𝔄 i)
    by (rule smcf-is-dghm)
qed

lemma smcf-up-ArrMap-app-vrange:
  assumes a ∈ₒ 𝔈(|Arr|) and ⋀i. i ∈ₒ I ⟹ φ i : 𝔈 ↦↦_SMCα 𝔄 i
  shows ℛₒ (smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|a|)) ⊆ₒ (⋃ₒ i∈ₒI. 𝔄 i(|Arr|))
proof
  (
    rule dghm-up-ArrMap-app-vrange[
      where 𝔄=‹λi. smc-dg (𝔄 i)›
        and φ=‹λi. smcf-dghm (φ i)›
        and 𝔈=‹smc-dg 𝔈›,
      unfolded slicing-simps slicing-commute
      ]
  )
  show a ∈ₒ 𝔈(|Arr|) by (rule assms)
  fix i assume i ∈ₒ I
  then interpret is-semifunctor α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms(2))
  show smcf-dghm (φ i) : smc-dg 𝔈 ↦↦_DGα smc-dg (𝔄 i)
    by (rule smcf-is-dghm)
qed

end

context psemicategory
begin

interpretation pdg: pdigraph α I ‹λi. smc-dg (𝔄 i)› by (rule psmc-pdigraph)

lemmas-with [unfolded slicing-simps slicing-commute]:
  psmc-dghm-comp-dghm-proj-dghm-up = pdg.pdg-dghm-comp-dghm-proj-dghm-up
  and psmc-dghm-up-eq-dghm-proj = pdg.pdg-dghm-up-eq-dghm-proj

end
```

Semicategory product universal property semifunctor is a semifunctor

```
lemma (in psemicategory) psmc-smcf-up-is-semifunctor:
  assumes semicategory α 𝔈 and ⋀i. i ∈ₒ I ⟹ φ i : 𝔈 ↦↦_SMCα 𝔄 i
  shows smcf-up I 𝔄 𝔈 φ : 𝔈 ↦↦_SMCα (∏_SMC i∈ₒI. 𝔄 i)
proof(intro is-semifunctorI)
  interpret 𝔈: semicategory α 𝔈 by (simp add: assms(1))
  interpret 𝔄: semicategory α ‹smc-prod I 𝔄›
    by (rule psmc-semicategory-smc-prod)
  show vfsequence (smcf-up I 𝔄 𝔈 φ)
    unfolding smcf-up-def by simp
  show vcard (smcf-up I 𝔄 𝔈 φ) = 4ℕ
    unfolding smcf-up-def by (simp add: nat-omega-simps)
  show dghm-smcf-up:
    smcf-dghm (smcf-up I 𝔄 𝔈 φ) : smc-dg 𝔈 ↦↦_DGα smc-dg (smc-prod I 𝔄)
    by
      (
        simp add:
```

```
      assms
      slicing-commute[symmetric]
      psmc-pdigraph
      is-semifunctor.smcf-is-dghm
      pdigraph.pdg-dghm-up-is-dghm
      semicategory.smc-digraph
    )
  interpret smcf-up:
    is-dghm α ‹smc-dg 𝔈› ‹smc-dg (smc-prod I 𝔄)› ‹smcf-dghm (smcf-up I 𝔄 𝔈 φ)›
    by (rule dghm-smcf-up)
  show smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|g ∘_A𝔈 f|) =
    smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|g|) ∘_A smc-prod I 𝔄 smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|f|)
    if g : b ↦_𝔈 c and f : a ↦_𝔈 b for g b c f a
  proof(rule smc-prod-Arr-cong[of - I 𝔄])
    note smcf-up-f =
      smcf-up.dghm-ArrMap-is-arr[unfolded slicing-simps, OF that(2)]
      and smcf-up-g =
      smcf-up.dghm-ArrMap-is-arr[unfolded slicing-simps, OF that(1)]
    from that have gf: g ∘_A𝔈 f : a ↦_𝔈 c
      by (simp add: smc-cs-intros)
    from smcf-up.dghm-ArrMap-is-arr[unfolded slicing-simps, OF this] show
      smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|g ∘_A𝔈 f|) ∈_∘ smc-prod I 𝔄(|Arr|)
      by (simp add: smc-cs-intros)
    from smcf-up-g smcf-up-f show
      smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|g|) ∘_A smc-prod I 𝔄 smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|f|) ∈_∘
        smc-prod I 𝔄(|Arr|)
      by (meson 𝔄.smc-is-arrE 𝔄.smc-Comp-is-arr)
    fix i assume prems: i ∈_∘ I
    from gf have gf': g ∘_A𝔈 f ∈_∘ 𝔈(|Arr|) by (simp add: smc-cs-intros)
    from that have g: g ∈_∘ 𝔈(|Arr|) and f: f ∈_∘ 𝔈(|Arr|) by auto
    interpret φ: is-semifunctor α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms(2)[OF prems])
    from that show smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|g ∘_A𝔈 f|)(|i|) =
      (
        smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|g|) ∘_A smc-prod I 𝔄 smcf-up I 𝔄 𝔈 φ(|ArrMap|)(|f|)
      )(|i|)
      unfolding
        smcf-up-ArrMap-app-component[OF gf' prems]
        smc-prod-Comp-app-component[OF smcf-up-g smcf-up-f prems]
        smcf-up-ArrMap-app-component[OF g prems]
        smcf-up-ArrMap-app-component[OF f prems]
      by (rule φ.smcf-ArrMap-Comp)
  qed
qed (auto simp: assms(1) psmc-semicategory-smc-prod smcf-up-components)
```

Further properties

```
lemma (in psemicategory) psmc-Comp-smcf-proj-smcf-up:
  assumes semicategory α 𝔈
    and ⋀i. i ∈_∘ I ⟹ φ i : 𝔈 ↦↦_SMCα 𝔄 i
    and i ∈_∘ I
  shows φ i = π_SMC I 𝔄 i ∘_SMCF smcf-up I 𝔄 𝔈 φ
proof(rule smcf-dghm-eqI)
  interpret φ: is-semifunctor α 𝔈 ‹𝔄 i› ‹φ i› by (rule assms(2)[OF assms(3)])
  interpret π: is-semifunctor α ‹smc-prod I 𝔄› ‹𝔄 i› ‹π_SMC I 𝔄 i›
    by (simp add: assms(3) psmc-smcf-proj-is-semifunctor)
  interpret up: is-semifunctor α 𝔈 ‹smc-prod I 𝔄› ‹smcf-up I 𝔄 𝔈 φ›
    by
      (
```

```
      simp add:
        assms(2) φ.HomDom.semicategory-axioms psmc-smcf-up-is-semifunctor
    )
  show φ i : 𝔈 ↦↦_SMCα 𝔄 i by (simp add: smc-cs-intros)
  show π_SMC I 𝔄 i ∘_SMCF smcf-up I 𝔄 𝔈 φ : 𝔈 ↦↦_SMCα 𝔄 i
    by (auto intro: smc-cs-intros)
  from assms show
   smcf-dghm (φ i) = smcf-dghm (π_SMC I 𝔄 i ∘_SMCF smcf-up I 𝔄 𝔈 φ)
   unfolding slicing-simps[symmetric] slicing-commute[symmetric]
   by
    (
     intro
      psmc-dghm-comp-dghm-proj-dghm-up
       [
        where φ=‹λi. smcf-dghm (φ i)›,
        unfolded slicing-simps[symmetric] slicing-commute[symmetric]
       ]
    )
    (auto simp: is-semifunctor.smcf-is-dghm)
qed simp-all

lemma (in psemicategory) psmc-smcf-up-eq-smcf-proj:
  assumes 𝔉 : 𝔈 ↦↦_SMCα (∏_SMC i∈_∘ I. 𝔄 i)
    and ⋀i. i ∈_∘ I ⟹ φ i = π_SMC I 𝔄 i ∘_SMCF 𝔉
  shows smcf-up I 𝔄 𝔈 φ = 𝔉
proof(rule smcf-dghm-eqI)
  interpret 𝔉: is-semifunctor α 𝔈 ‹(∏_SMC i∈_∘ I. 𝔄 i)› 𝔉 by (rule assms(1))
  show smcf-up I 𝔄 𝔈 φ : 𝔈 ↦↦_SMCα (∏_SMC i∈_∘ I. 𝔄 i)
  proof(rule psmc-smcf-up-is-semifunctor)
    fix i assume prems: i ∈_∘ I
    interpret π: is-semifunctor α ‹(∏_SMC i∈_∘ I. 𝔄 i)› ‹𝔄 i› ‹π_SMC I 𝔄 i›
      using prems by (rule psmc-smcf-proj-is-semifunctor)
    show φ i : 𝔈 ↦↦_SMCα 𝔄 i
      unfolding assms(2)[OF prems] by (auto intro: smc-cs-intros)
  qed (auto intro: smc-cs-intros)
  show 𝔉 : 𝔈 ↦↦_SMCα (∏_SMC i∈_∘ I. 𝔄 i) by (rule assms(1))
  from assms show smcf-dghm (smcf-up I 𝔄 𝔈 φ) = smcf-dghm 𝔉
    unfolding slicing-simps[symmetric] slicing-commute[symmetric]
    by (intro psmc-dghm-up-eq-dghm-proj)
      (auto simp: slicing-simps slicing-commute)
qed simp-all
```

4.8.9 Singleton semicategory

Slicing

```
context
  fixes 𝔈 :: V
begin

lemmas-with [where 𝔈=‹smc-dg 𝔈›, unfolded slicing-simps slicing-commute]:
  smc-singleton-ObjI = dg-singleton-ObjI
  and smc-singleton-ObjE = dg-singleton-ObjE
  and smc-singleton-ArrI = dg-singleton-ArrI
  and smc-singleton-ArrE = dg-singleton-ArrE

end
```

context *semicategory*
begin

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

lemmas-with [*unfolded slicing-simps slicing-commute*]:
smc-finite-pdigraph-smc-singleton = *dg.dg-finite-pdigraph-dg-singleton*
and *smc-singleton-is-arrI* = *dg.dg-singleton-is-arrI*
and *smc-singleton-is-arrD* = *dg.dg-singleton-is-arrD*
and *smc-singleton-is-arrE* = *dg.dg-singleton-is-arrE*

end

Singleton semicategory is a semicategory

lemma (**in** *semicategory*) *smc-finite-psemicategory-smc-singleton*:
assumes $j \in_\circ$ *Vset* α
shows *finite-psemicategory* α (*set* $\{j\}$) ($\lambda i.$ $\mathfrak{C}$)
by
(
auto intro:
assms
semicategory-axioms
finite-psemicategory-finite-pdigraphI
smc-finite-pdigraph-smc-singleton
)

lemma (**in** *semicategory*) *smc-semicategory-smc-singleton*:
assumes $j \in_\circ$ *Vset* α
shows *semicategory* α ($\prod_{SMC} i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
proof–
interpret *finite-psemicategory* α ‹*set* $\{j\}$› ‹$\lambda i.$ $\mathfrak{C}$›
using *assms* **by** (*rule smc-finite-psemicategory-smc-singleton*)
show *?thesis* **by** (*rule psmc-semicategory-smc-prod*)
qed

4.8.10 Singleton semifunctor

Definition and elementary properties

definition *smcf-singleton* :: $V \Rightarrow V \Rightarrow V$
where *smcf-singleton* j $\mathfrak{C}$ =
[
($\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!)$. *set* $\{\langle j, a\rangle\}$),
($\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!)$. *set* $\{\langle j, f\rangle\}$),
$\mathfrak{C}$,
($\prod_{SMC} i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
]$_\circ$

Components.

lemma *smcf-singleton-components*:
shows *smcf-singleton* j $\mathfrak{C}(\!|ObjMap|\!)$ = ($\lambda a \in_\circ \mathfrak{C}(\!|Obj|\!)$. *set* $\{\langle j, a\rangle\}$)
and *smcf-singleton* j $\mathfrak{C}(\!|ArrMap|\!)$ = ($\lambda f \in_\circ \mathfrak{C}(\!|Arr|\!)$. *set* $\{\langle j, f\rangle\}$)
and *smcf-singleton* j $\mathfrak{C}(\!|HomDom|\!)$ = $\mathfrak{C}$
and *smcf-singleton* j $\mathfrak{C}(\!|HomCod|\!)$ = ($\prod_{SMC} i \in_\circ$ *set* $\{j\}$. $\mathfrak{C}$)
unfolding *smcf-singleton-def dghm-field-simps*
by (*simp-all add*: *nat-omega-simps*)

Slicing.

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

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

lemmas-with [**where** $\mathfrak{C}$=‹*smc-dg* $\mathfrak{C}$›, *unfolded slicing-simps slicing-commute*]:
smcf-singleton-ObjMap-vsv[*smc-cs-intros*] = *dghm-singleton-ObjMap-vsv*
and *smcf-singleton-ObjMap-vdomain*[*smc-cs-simps*] =
dghm-singleton-ObjMap-vdomain
and *smcf-singleton-ObjMap-vrange* = *dghm-singleton-ObjMap-vrange*
and *smcf-singleton-ObjMap-app*[*smc-prod-cs-simps*] = *dghm-singleton-ObjMap-app*
and *smcf-singleton-ArrMap-vsv*[*smc-cs-intros*] = *dghm-singleton-ArrMap-vsv*
and *smcf-singleton-ArrMap-vdomain*[*smc-cs-simps*] =
dghm-singleton-ArrMap-vdomain
and *smcf-singleton-ArrMap-vrange* = *dghm-singleton-ArrMap-vrange*
and *smcf-singleton-ArrMap-app*[*smc-prod-cs-simps*] = *dghm-singleton-ArrMap-app*

end

context *semicategory*
begin

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

lemmas-with [*unfolded slicing-simps slicing-commute*]:
smc-smcf-singleton-is-dghm = *dg.dg-dghm-singleton-is-dghm*

end

Singleton semifunctor is an isomorphism of semicategories

lemma (**in** *semicategory*) *smc-smcf-singleton-is-iso-semifunctor*:
assumes $j \in_\circ$ *Vset* α
shows *smcf-singleton j* $\mathfrak{C}$: $\mathfrak{C} \mapsto\!\!\!\mapsto_{SMC.iso\alpha} (\prod_{SMC} i \in_\circ set\ \{j\}.\ \mathfrak{C})$
proof(*intro is-iso-semifunctorI is-semifunctorI*)
show *dghm-singleton*:
smcf-dghm (*smcf-singleton j* $\mathfrak{C}$) :
smc-dg $\mathfrak{C} \mapsto\!\!\!\mapsto_{DG.iso\alpha}$ *smc-dg* $(\prod_{SMC} i \in_\circ set\ \{j\}.\ \mathfrak{C})$
by (*rule smc-smcf-singleton-is-dghm*[*OF assms*, *unfolded slicing-simps*])
show *vfsequence* (*smcf-singleton j* $\mathfrak{C}$) **unfolding** *smcf-singleton-def* **by** *simp*
show *vcard* (*smcf-singleton j* $\mathfrak{C}$) = $4_{\mathbb{N}}$
unfolding *smcf-singleton-def* **by** (*simp add*: *nat-omega-simps*)
from *dghm-singleton* **show**
smcf-dghm (*smcf-singleton j* $\mathfrak{C}$) :
smc-dg $\mathfrak{C} \mapsto\!\!\!\mapsto_{DG\alpha}$ *smc-dg* $(\prod_{SMC} i \in_\circ set\ \{j\}.\ \mathfrak{C})$
by (*simp add*: *is-iso-dghm.axioms*(1))
show *smcf-singleton j* $\mathfrak{C}$(|*ArrMap*|)(|$g \circ_{A\mathfrak{C}} f$|) =
smcf-singleton j $\mathfrak{C}$(|*ArrMap*|)(|*g*|) $\circ_{A \prod_{SMC} i \in_\circ set\ \{j\}.\ \mathfrak{C}}$
smcf-singleton j $\mathfrak{C}$(|*ArrMap*|)(|*f*|)
if $g : b \mapsto_{\mathfrak{C}} c$ **and** $f : a \mapsto_{\mathfrak{C}} b$ **for** *g b c f a*

```
  proof–
    let ?jg = ‹smcf-singleton j 𝔈(⁅ArrMap⁆)(⁅g⁆)›
      and ?jf = ‹smcf-singleton j 𝔈(⁅ArrMap⁆)(⁅f⁆)›
    from that have [simp]: ?jg = set {⟨j, g⟩} ?jf = set {⟨j, f⟩}
      by (simp-all add: smcf-singleton-ArrMap-app smc-cs-intros)
   from that have g ∘_A𝔈 f : a ↦_𝔈 c by (auto intro: smc-cs-intros)
    then have smcf-singleton j 𝔈(⁅ArrMap⁆)(⁅g ∘_A𝔈 f⁆) = set {⟨j, g ∘_A𝔈 f⟩}
      by (simp-all add: smcf-singleton-ArrMap-app smc-cs-intros)
    moreover from
      smc-singleton-is-arrI[OF assms that(1)]
      smc-singleton-is-arrI[OF assms that(2)]
    have ?jg ∘_A∏_SMC i∈_∘set {j}. 𝔈 ?jf = set {⟨j, g ∘_A𝔈 f⟩}
      by (simp add: smc-prod-Comp-app VLambda-vsingleton)
    ultimately show ?thesis by auto
  qed
qed
  (
    auto intro:
      smc-cs-intros
      assms
      smc-semicategory-smc-singleton
      smcf-singleton-components
  )

lemmas [smc-cs-intros] = semicategory.smc-smcf-singleton-is-iso-semifunctor
```

4.8.11 Product of two semicategories

Definition and elementary properties.

See Chapter II-3 in [39].

```
definition smc-prod-2 :: V ⇒ V ⇒ V (infixr ‹×_SMC› 80)
  where 𝔄 ×_SMC 𝔅 ≡ smc-prod (2_ℕ) (λi. (i = 0 ? 𝔄 : 𝔅))
```

Slicing.

```
lemma smc-dg-smc-prod-2[slicing-commute]:
  smc-dg 𝔄 ×_DG smc-dg 𝔅 = smc-dg (𝔄 ×_SMC 𝔅)
  unfolding smc-prod-2-def dg-prod-2-def slicing-commute[symmetric] if-distrib
  by simp

context
  fixes α 𝔄 𝔅
  assumes 𝔄: semicategory α 𝔄 and 𝔅: semicategory α 𝔅
begin

interpretation 𝔄: semicategory α 𝔄 by (rule 𝔄)
interpretation 𝔅: semicategory α 𝔅 by (rule 𝔅)

lemmas-with
  [
    where 𝔄=‹smc-dg 𝔄› and 𝔅=‹smc-dg 𝔅›,
    unfolded slicing-simps slicing-commute,
    OF 𝔄.smc-digraph 𝔅.smc-digraph
  ]:
  smc-prod-2-ObjI = dg-prod-2-ObjI
  and smc-prod-2-ObjI'[smc-prod-cs-intros] = dg-prod-2-ObjI'
  and smc-prod-2-ObjE = dg-prod-2-ObjE
```

and *smc-prod-2-ArrI* = *dg-prod-2-ArrI*
and *smc-prod-2-ArrI'*[*smc-prod-cs-intros*] = *dg-prod-2-ArrI'*
and *smc-prod-2-ArrE* = *dg-prod-2-ArrE*
and *smc-prod-2-is-arrI* = *dg-prod-2-is-arrI*
and *smc-prod-2-is-arrI'*[*smc-prod-cs-intros*] = *dg-prod-2-is-arrI'*
and *smc-prod-2-is-arrE* = *dg-prod-2-is-arrE*
and *smc-prod-2-Dom-vsv* = *dg-prod-2-Dom-vsv*
and *smc-prod-2-Dom-vdomain*[*smc-cs-simps*] = *dg-prod-2-Dom-vdomain*
and *smc-prod-2-Dom-app*[*smc-prod-cs-simps*] = *dg-prod-2-Dom-app*
and *smc-prod-2-Dom-vrange* = *dg-prod-2-Dom-vrange*
and *smc-prod-2-Cod-vsv* = *dg-prod-2-Cod-vsv*
and *smc-prod-2-Cod-vdomain*[*smc-cs-simps*] = *dg-prod-2-Cod-vdomain*
and *smc-prod-2-Cod-app*[*smc-prod-cs-simps*] = *dg-prod-2-Cod-app*
and *smc-prod-2-Cod-vrange* = *dg-prod-2-Cod-vrange*
and *smc-prod-2-op-smc-smc-Obj*[*smc-op-simps*] = *dg-prod-2-op-dg-dg-Obj*
and *smc-prod-2-smc-op-smc-Obj*[*smc-op-simps*] = *dg-prod-2-dg-op-dg-Obj*
and *smc-prod-2-op-smc-smc-Arr*[*smc-op-simps*] = *dg-prod-2-op-dg-dg-Arr*
and *smc-prod-2-smc-op-smc-Arr*[*smc-op-simps*] = *dg-prod-2-dg-op-dg-Arr*

end

Product of two semicategories is a semicategory

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule semicategoryD*[*OF* $\mathfrak{A}$])
interpretation $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

lemma *finite-psemicategory-smc-prod-2*:
finite-psemicategory α ($2_{\mathbb{N}}$) (*if2* $\mathfrak{A}$ $\mathfrak{B}$)
proof(*intro finite-psemicategoryI psemicategory-baseI*)
from *Axiom-of-Infinity* **show** *z1-in-Vset*: $2_{\mathbb{N}} \in_{\circ} Vset\ \alpha$ **by** *blast*
show *semicategory* α ($i = 0$ *?* $\mathfrak{A}$: $\mathfrak{B}$) **if** $i \in_{\circ} 2_{\mathbb{N}}$ **for** i
by (*auto simp*: *smc-cs-intros*)
qed *auto*

interpretation *finite-psemicategory* α ‹$2_{\mathbb{N}}$› ‹*if2* $\mathfrak{A}$ $\mathfrak{B}$›
by (*intro finite-psemicategory-smc-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *semicategory-smc-prod-2*[*smc-cs-intros*]: *semicategory* α ($\mathfrak{A} \times_{SMC} \mathfrak{B}$)
unfolding *smc-prod-2-def* **by** (*rule psmc-semicategory-smc-prod*)

end

Composition

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule semicategoryD*[*OF* $\mathfrak{A}$])

interpretation *finite-psemicategory* α ‹$2_{\mathbb{N}}$› ‹*if2* $\mathfrak{A}$ $\mathfrak{B}$›

by (*intro finite-psemicategory-smc-prod-2* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *smc-prod-2-Comp-app*[*smc-prod-cs-simps*]:
assumes $[g,\ g']_\circ : [b,\ b']_\circ \mapsto_{\mathfrak{A} \times_{SMC} \mathfrak{B}} [c,\ c']_\circ$
and $[f,\ f']_\circ : [a,\ a']_\circ \mapsto_{\mathfrak{A} \times_{SMC} \mathfrak{B}} [b,\ b']_\circ$
shows $[g,\ g']_\circ \circ_{A\mathfrak{A} \times_{SMC} \mathfrak{B}} [f,\ f']_\circ = [g \circ_{A\mathfrak{A}} f,\ g' \circ_{A\mathfrak{B}} f']_\circ$
proof–
have $[g,\ g']_\circ \circ_{A\mathfrak{A} \times_{SMC} \mathfrak{B}} [f,\ f']_\circ =$
$(\lambda i \in_\circ 2_{\mathbb{N}}.\ [g,\ g']_\circ(\!|i|\!) \circ_{A\, i = 0\ ?\ \mathfrak{A} :\ \mathfrak{B}} [f,\ f']_\circ(\!|i|\!))$
by
(
rule smc-prod-Comp-app[
OF assms[*unfolded smc-prod-2-def*], *folded smc-prod-2-def*
]
)
also have
$(\lambda i \in_\circ 2_{\mathbb{N}}.\ [g,\ g']_\circ(\!|i|\!) \circ_{A\, i = 0\ ?\ \mathfrak{A} :\ \mathfrak{B}} [f,\ f']_\circ(\!|i|\!)) =$
$[g \circ_{A\mathfrak{A}} f,\ g' \circ_{A\mathfrak{B}} f']_\circ$
proof(*rule vsv-eqI*, *unfold vdomain-VLambda*)
fix i **assume** $i \in_\circ 2_{\mathbb{N}}$
then consider $\langle i = 0 \rangle \mid \langle i = 1_{\mathbb{N}} \rangle$ **unfolding** *two* **by** *auto*
then show
$(\lambda i \in_\circ 2_{\mathbb{N}}.\ [g,\ g']_\circ(\!|i|\!) \circ_{A\, i = 0\ ?\ \mathfrak{A} :\ \mathfrak{B}} [f,\ f']_\circ(\!|i|\!))(\!|i|\!) =$
$[g \circ_{A\mathfrak{A}} f,\ g' \circ_{A\mathfrak{B}} f']_\circ(\!|i|\!)$
by *cases* (*simp-all add*: *two nat-omega-simps*)
qed (*auto simp*: *two nat-omega-simps*)
finally show *?thesis* **by** *simp*
qed

end

Opposite product semicategory

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
begin

interpretation $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

lemma *op-smc-smc-prod-2*[*smc-op-simps*]:
op-smc $(\mathfrak{A} \times_{SMC} \mathfrak{B})$ = *op-smc* $\mathfrak{A} \times_{SMC}$ *op-smc* $\mathfrak{B}$
proof(*rule smc-dg-eqI*[*of* α])

from $\mathfrak{A}$ $\mathfrak{B}$ **show** *smc-lhs*: *semicategory* α (*op-smc* $(\mathfrak{A} \times_{SMC} \mathfrak{B})$)
by
(
cs-concl **cs-shallow**
cs-simp: *smc-cs-simps smc-op-simps*
cs-intro: *smc-cs-intros smc-op-intros*
)
interpret *smc-lhs*: *semicategory* α ‹*op-smc* $(\mathfrak{A} \times_{SMC} \mathfrak{B})$› **by** (*rule smc-lhs*)

from $\mathfrak{A}$ $\mathfrak{B}$ **show** *smc-rhs*: *semicategory* α (*op-smc* $\mathfrak{A} \times_{SMC}$ *op-smc* $\mathfrak{B}$)
by
(
cs-concl **cs-shallow**

```
      cs-simp: smc-cs-simps smc-op-simps
      cs-intro: smc-cs-intros smc-op-intros
  )
interpret smc-rhs: semicategory α ‹op-smc 𝔄 ×_SMC op-smc 𝔅› by (rule smc-rhs)

show op-smc (𝔄 ×_SMC 𝔅)⦇Comp⦈ = (op-smc 𝔄 ×_SMC op-smc 𝔅)⦇Comp⦈
proof(rule vsv-eqI)
  show vsv (op-smc (𝔄 ×_SMC 𝔅)⦇Comp⦈)
    unfolding op-smc-components by (rule fflip-vsv)
  show vsv ((op-smc 𝔄 ×_SMC op-smc 𝔅)⦇Comp⦈)
    unfolding smc-prod-2-def smc-prod-components by simp
  show 𝒟∘ (op-smc (𝔄 ×_SMC 𝔅)⦇Comp⦈) = 𝒟∘ ((op-smc 𝔄 ×_SMC op-smc 𝔅)⦇Comp⦈)
  proof(intro vsubset-antisym vsubsetI)
    fix gg′ff′ assume gf: gg′ff′ ∈∘ 𝒟∘ (op-smc (𝔄 ×_SMC 𝔅)⦇Comp⦈)
    then obtain gg′ ff′ aa′ bb′ cc′
      where gg′ff′-def: gg′ff′ = [gg′, ff′]∘
        and gg′: bb′ ↦_op-smc (𝔄 ×_SMC 𝔅) cc′
        and ff′: aa′ ↦_op-smc (𝔄 ×_SMC 𝔅) bb′
      by clarsimp
    then have gg′: gg′ : cc′ ↦_𝔄 ×_SMC 𝔅 bb′
      and ff′: ff′ : bb′ ↦_𝔄 ×_SMC 𝔅 aa′
      unfolding smc-op-simps by simp-all
    from gg′ obtain g g′ b b′ c c′
      where gg′-def: gg′ = [g, g′]∘
        and cc′ = [c, c′]∘
        and bb′ = [b, b′]∘
        and g: g : c ↦_𝔄 b
        and g′: g′ : c′ ↦_𝔅 b′
      by (elim smc-prod-2-is-arrE[OF 𝔄 𝔅])
    with ff′ obtain f f′ a a′
      where ff′-def: ff′ = [f, f′]∘
        and bb′ = [b, b′]∘
        and aa′ = [a, a′]∘
        and f: f : b ↦_𝔄 a
        and f′: f′ : b′ ↦_𝔅 a′
      by (auto elim: smc-prod-2-is-arrE[OF 𝔄 𝔅])
    from 𝔄 𝔅 g g′ f f′ show gg′ff′ ∈∘ 𝒟∘ ((op-smc 𝔄 ×_SMC op-smc 𝔅)⦇Comp⦈)
      by
        (
          intro smc-rhs.smc-Comp-vdomainI[OF - - gg′ff′-def],
          unfold gg′-def ff′-def
        )
        (
          cs-concl cs-shallow
            cs-simp: smc-cs-simps smc-op-simps
            cs-intro: smc-op-intros smc-prod-cs-intros
        )
  next
    fix gg′ff′ assume gf: gg′ff′ ∈∘ 𝒟∘ ((op-smc 𝔄 ×_SMC op-smc 𝔅)⦇Comp⦈)
    then obtain gg′ ff′ aa′ bb′ cc′
      where gg′ff′-def: gg′ff′ = [gg′, ff′]∘
        and gg′: gg′ : bb′ ↦_op-smc 𝔄 ×_SMC op-smc 𝔅 cc′
        and ff′: ff′ : aa′ ↦_op-smc 𝔄 ×_SMC op-smc 𝔅 bb′
      by clarsimp
    from gg′ obtain g g′ b b′ c c′
      where gg′-def: gg′ = [g, g′]∘
        and bb′ = [b, b′]∘
        and cc′ = [c, c′]∘
```

```
      and g: g : b ↦_op-smc 𝔄 c
      and g': g' : b' ↦_op-smc 𝔅 c'
    by (elim smc-prod-2-is-arrE[OF 𝔄.semicategory-op 𝔅.semicategory-op])
  with ff' obtain f f' a a'
    where ff'-def: ff' = [f, f']∘
      and aa' = [a, a']∘
      and bb' = [b, b']∘
      and f: f : a ↦_op-smc 𝔄 b
      and f': f' : a' ↦_op-smc 𝔅 b'
    by
      (
        auto elim:
          smc-prod-2-is-arrE[OF 𝔄.semicategory-op 𝔅.semicategory-op]
      )
  from 𝔄 𝔅 g g' f f' show gg'ff' ∈∘ 𝒟∘ (op-smc (𝔄 ×_SMC 𝔅)(‖Comp‖))
    by
      (
        intro smc-lhs.smc-Comp-vdomainI[OF - - gg'ff'-def],
        unfold gg'-def ff'-def smc-op-simps
      )
      (
        cs-concl cs-shallow
          cs-simp: smc-cs-simps smc-op-simps
          cs-intro: smc-op-intros smc-prod-cs-intros
      )
qed
fix gg'ff' assume gg'ff' ∈∘ 𝒟∘ (op-smc (𝔄 ×_SMC 𝔅)(‖Comp‖))
then obtain gg' ff' aa' bb' cc'
  where gg'ff'-def: gg'ff' = [gg', ff']∘
    and gg' : bb' ↦_op-smc (𝔄 ×_SMC 𝔅) cc'
    and ff' : aa' ↦_op-smc (𝔄 ×_SMC 𝔅) bb'
  by clarsimp
then have gg': gg' : cc' ↦_𝔄 ×_SMC 𝔅 bb'
  and ff': ff' : bb' ↦_𝔄 ×_SMC 𝔅 aa'
  unfolding smc-op-simps by simp-all
from gg' obtain g g' b b' c c'
  where gg'-def[smc-cs-simps]: gg' = [g, g']∘
    and cc' = [c, c']∘
    and bb' = [b, b']∘
    and g: g : c ↦_𝔄 b
    and g': g' : c' ↦_𝔅 b'
  by (elim smc-prod-2-is-arrE[OF 𝔄 𝔅])
with ff' obtain f f' a a'
  where ff'-def[smc-cs-simps]: ff' = [f, f']∘
    and bb' = [b, b']∘
    and aa' = [a, a']∘
    and f: f : b ↦_𝔄 a
    and f': f' : b' ↦_𝔅 a'
  by (auto elim: smc-prod-2-is-arrE[OF 𝔄 𝔅])
from 𝔄 𝔅 g g' f f' show op-smc (𝔄 ×_SMC 𝔅)(‖Comp‖)(‖gg'ff'‖) =
  (op-smc 𝔄 ×_SMC op-smc 𝔅)(‖Comp‖)(‖gg'ff'‖)
  unfolding gg'ff'-def
  by
    (
      cs-concl cs-shallow
        cs-simp: smc-cs-simps smc-op-simps smc-prod-cs-simps
        cs-intro: smc-cs-intros smc-op-intros smc-prod-cs-intros
```

)
qed

from $\mathfrak{A}$ $\mathfrak{B}$ **show**
smc-dg (*op-smc* ($\mathfrak{A}$ $\times_{SMC}$ $\mathfrak{B}$)) = *smc-dg* (*op-smc* $\mathfrak{A}$ $\times_{SMC}$ *op-smc* $\mathfrak{B}$)
unfolding *slicing-commute*[*symmetric*]
by (*cs-concl* **cs-shallow cs-simp**: *dg-op-simps* **cs-intro**: *slicing-intros*)

qed

end

4.8.12 Projections for the product of two semicategories

Definition and elementary properties

See Chapter II-3 in [39].

definition *smcf-proj-fst* :: $V \Rightarrow V \Rightarrow V$ ($\langle\pi_{SMC.1}\rangle$)
where $\pi_{SMC.1}$ $\mathfrak{A}$ $\mathfrak{B}$ = *smcf-proj* ($2_{\mathbb{N}}$) ($\lambda i.$ (i = 0 *?* $\mathfrak{A}$: $\mathfrak{B}$)) 0
definition *smcf-proj-snd* :: $V \Rightarrow V \Rightarrow V$ ($\langle\pi_{SMC.2}\rangle$)
where $\pi_{SMC.2}$ $\mathfrak{A}$ $\mathfrak{B}$ = *smcf-proj* ($2_{\mathbb{N}}$) ($\lambda i.$ (i = 0 *?* $\mathfrak{A}$: $\mathfrak{B}$)) ($1_{\mathbb{N}}$)

Slicing

lemma *smcf-dghm-smcf-proj-fst*[*slicing-commute*]:
$\pi_{DG.1}$ (*smc-dg* $\mathfrak{A}$) (*smc-dg* $\mathfrak{B}$) = *smcf-dghm* ($\pi_{SMC.1}$ $\mathfrak{A}$ $\mathfrak{B}$)
unfolding
smcf-proj-fst-def dghm-proj-fst-def slicing-commute[*symmetric*] *if-distrib*
..

lemma *smcf-dghm-smcf-proj-snd*[*slicing-commute*]:
$\pi_{DG.2}$ (*smc-dg* $\mathfrak{A}$) (*smc-dg* $\mathfrak{B}$) = *smcf-dghm* ($\pi_{SMC.2}$ $\mathfrak{A}$ $\mathfrak{B}$)
unfolding
smcf-proj-snd-def dghm-proj-snd-def slicing-commute[*symmetric*] *if-distrib*
..

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule semicategoryD*[*OF* $\mathfrak{A}$])
interpretation $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)

lemmas-with
[
where $\mathfrak{A}$=$\langle$*smc-dg* $\mathfrak{A}\rangle$ **and** $\mathfrak{B}$=$\langle$*smc-dg* $\mathfrak{B}\rangle$,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*smc-digraph* $\mathfrak{B}$.*smc-digraph*
]:
smcf-proj-fst-ObjMap-app[*smc-cs-simps*] = *dghm-proj-fst-ObjMap-app*
and *smcf-proj-snd-ObjMap-app*[*smc-cs-simps*] = *dghm-proj-snd-ObjMap-app*
and *smcf-proj-fst-ArrMap-app*[*smc-cs-simps*] = *dghm-proj-fst-ArrMap-app*
and *smcf-proj-snd-ArrMap-app*[*smc-cs-simps*] = *dghm-proj-snd-ArrMap-app*

end

Domain and codomain of a projection of a product of two semicategories

lemma *smcf-proj-fst-HomDom*: $\pi_{SMC.1}\ \mathfrak{A}\ \mathfrak{B}(\|HomDom\|) = \mathfrak{A} \times_{SMC} \mathfrak{B}$
unfolding *smcf-proj-fst-def smcf-proj-components smc-prod-2-def* **..**

lemma *smcf-proj-fst-HomCod*: $\pi_{SMC.1}\ \mathfrak{A}\ \mathfrak{B}(\|HomCod\|) = \mathfrak{A}$
unfolding *smcf-proj-fst-def smcf-proj-components smc-prod-2-def* **by** *simp*

lemma *smcf-proj-snd-HomDom*: $\pi_{SMC.2}\ \mathfrak{A}\ \mathfrak{B}(\|HomDom\|) = \mathfrak{A} \times_{SMC} \mathfrak{B}$
unfolding *smcf-proj-snd-def smcf-proj-components smc-prod-2-def* **..**

lemma *smcf-proj-snd-HomCod*: $\pi_{SMC.2}\ \mathfrak{A}\ \mathfrak{B}(\|HomCod\|) = \mathfrak{B}$
unfolding *smcf-proj-snd-def smcf-proj-components smc-prod-2-def* **by** *simp*

Projection of a product of two semicategories is a semifunctor

context
fixes $\alpha\ \mathfrak{A}\ \mathfrak{B}$
assumes $\mathfrak{A}$: *semicategory* $\alpha\ \mathfrak{A}$ **and** $\mathfrak{B}$: *semicategory* $\alpha\ \mathfrak{B}$
begin

interpretation $\mathcal{Z}\ \alpha$ **by** (*rule semicategoryD*[*OF* $\mathfrak{A}$])
interpretation *finite-psemicategory* $\alpha\ \langle 2_{\mathbb{N}}\rangle\ \langle if2\ \mathfrak{A}\ \mathfrak{B}\rangle$
by (*intro finite-psemicategory-smc-prod-2* $\mathfrak{A}\ \mathfrak{B}$)

lemma *smcf-proj-fst-is-semifunctor*:
assumes $i \in_{\circ} I$
shows $\pi_{SMC.1}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{A} \times_{SMC} \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{A}$
by
(
rule
psmc-smcf-proj-is-semifunctor[
where *i*=0, *simplified*, *folded smcf-proj-fst-def smc-prod-2-def*
]
)

lemma *smcf-proj-fst-is-semifunctor'*[*smc-cs-intros*]:
assumes $i \in_{\circ} I$ **and** $\mathfrak{C} = \mathfrak{A} \times_{SMC} \mathfrak{B}$ **and** $\mathfrak{D} = \mathfrak{A}$
shows $\pi_{SMC.1}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule smcf-proj-fst-is-semifunctor*)

lemma *smcf-proj-snd-is-semifunctor*:
assumes $i \in_{\circ} I$
shows $\pi_{SMC.2}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{A} \times_{SMC} \mathfrak{B} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
by
(
rule
psmc-smcf-proj-is-semifunctor[
where *i*=$\langle 1_{\mathbb{N}}\rangle$, *simplified*, *folded smcf-proj-snd-def smc-prod-2-def*
]
)

lemma *smcf-proj-snd-is-semifunctor'*[*smc-cs-intros*]:
assumes $i \in_{\circ} I$ **and** $\mathfrak{C} = \mathfrak{A} \times_{SMC} \mathfrak{B}$ **and** $\mathfrak{D} = \mathfrak{B}$
shows $\pi_{SMC.2}\ \mathfrak{A}\ \mathfrak{B} : \mathfrak{C} \mapsto\mapsto_{SMC\alpha} \mathfrak{D}$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule smcf-proj-snd-is-semifunctor*)

end

4.8.13 Product of three semicategories

Definition and elementary properties.

definition *smc-prod-3* :: $V \Rightarrow V \Rightarrow V \Rightarrow V$
($\langle$(- $\times_{SMC3}$ - $\times_{SMC3}$ -)$\rangle$ [81, 81, 81] 80)
where $\mathfrak{A} \times_{SMC3} \mathfrak{B} \times_{SMC3} \mathfrak{C} = (\prod_{SMC} i \in_{\circ} 3_{\mathbb{N}}.\ if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i)$

Slicing.

lemma *smc-dg-smc-prod-3*[*slicing-commute*]:
smc-dg $\mathfrak{A}$ $\times_{DG3}$ *smc-dg* $\mathfrak{B}$ $\times_{DG3}$ *smc-dg* $\mathfrak{C}$ = *smc-dg* ($\mathfrak{A}$ $\times_{SMC3}$ $\mathfrak{B}$ $\times_{SMC3}$ $\mathfrak{C}$)
unfolding *smc-prod-3-def dg-prod-3-def slicing-commute*[*symmetric*] *if-distrib*
by (*simp add*: *if-distrib*[*symmetric*])

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$
and $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
and $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$
begin

interpretation $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$ **by** (*rule* $\mathfrak{C}$)

lemmas-with
[
where $\mathfrak{A}$=$\langle$*smc-dg* $\mathfrak{A}$$\rangle$ **and** $\mathfrak{B}$=$\langle$*smc-dg* $\mathfrak{B}$$\rangle$ **and** $\mathfrak{C}$=$\langle$*smc-dg* $\mathfrak{C}$$\rangle$,
unfolded slicing-simps slicing-commute,
OF $\mathfrak{A}$.*smc-digraph* $\mathfrak{B}$.*smc-digraph* $\mathfrak{C}$.*smc-digraph*
]:
smc-prod-3-ObjI = *dg-prod-3-ObjI*
and *smc-prod-3-ObjI'*[*smc-prod-cs-intros*] = *dg-prod-3-ObjI'*
and *smc-prod-3-ObjE* = *dg-prod-3-ObjE*
and *smc-prod-3-ArrI* = *dg-prod-3-ArrI*
and *smc-prod-3-ArrI'*[*smc-prod-cs-intros*] = *dg-prod-3-ArrI'*
and *smc-prod-3-ArrE* = *dg-prod-3-ArrE*
and *smc-prod-3-is-arrI* = *dg-prod-3-is-arrI*
and *smc-prod-3-is-arrI'*[*smc-prod-cs-intros*] = *dg-prod-3-is-arrI'*
and *smc-prod-3-is-arrE* = *dg-prod-3-is-arrE*
and *smc-prod-3-Dom-vsv* = *dg-prod-3-Dom-vsv*
and *smc-prod-3-Dom-vdomain*[*smc-cs-simps*] = *dg-prod-3-Dom-vdomain*
and *smc-prod-3-Dom-app*[*smc-prod-cs-simps*] = *dg-prod-3-Dom-app*
and *smc-prod-3-Dom-vrange* = *dg-prod-3-Dom-vrange*
and *smc-prod-3-Cod-vsv* = *dg-prod-3-Cod-vsv*
and *smc-prod-3-Cod-vdomain*[*smc-cs-simps*] = *dg-prod-3-Cod-vdomain*
and *smc-prod-3-Cod-app*[*smc-prod-cs-simps*] = *dg-prod-3-Cod-app*
and *smc-prod-3-Cod-vrange* = *dg-prod-3-Cod-vrange*

end

Product of three semicategories is a semicategory

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$
and $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
and $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule semicategoryD*[*OF* $\mathfrak{A}$])
interpretation $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$ **by** (*rule* $\mathfrak{A}$)
interpretation $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule* $\mathfrak{B}$)
interpretation $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$ **by** (*rule* $\mathfrak{C}$)

lemma *finite-psemicategory-smc-prod-3*:
finite-psemicategory α ($3_{\mathbb{N}}$) (*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$)
proof(*intro finite-psemicategoryI psemicategory-baseI*)
from *Axiom-of-Infinity* **show** *z1-in-Vset*: $3_{\mathbb{N}} \in_{\circ} Vset\ \alpha$ **by** *blast*
show *semicategory* α (*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$ *i*) **if** $i \in_{\circ} 3_{\mathbb{N}}$ **for** *i*
by (*auto simp*: *smc-cs-intros*)
qed *auto*

interpretation *finite-psemicategory* α ‹$3_{\mathbb{N}}$› ‹*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$›
by (*intro finite-psemicategory-smc-prod-3* $\mathfrak{A}$ $\mathfrak{B}$)

lemma *semicategory-smc-prod-3*[*smc-cs-intros*]:
semicategory α ($\mathfrak{A} \times_{SMC3} \mathfrak{B} \times_{SMC3} \mathfrak{C}$)
unfolding *smc-prod-3-def* **by** (*rule psmc-semicategory-smc-prod*)

end

Composition

context
fixes α $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
assumes $\mathfrak{A}$: *semicategory* α $\mathfrak{A}$
and $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$
and $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$
begin

interpretation $\mathcal{Z}$ α **by** (*rule semicategoryD*[*OF* $\mathfrak{A}$])

interpretation *finite-psemicategory* α ‹$3_{\mathbb{N}}$› ‹*if3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$›
by (*intro finite-psemicategory-smc-prod-3* $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$)

lemma *smc-prod-3-Comp-app*[*smc-prod-cs-simps*]:
assumes $[g, g', g'']_{\circ} : [b, b', b'']_{\circ} \mapsto_{\mathfrak{A} \times_{SMC3} \mathfrak{B} \times_{SMC3} \mathfrak{C}} [c, c', c'']_{\circ}$
and $[f, f', f'']_{\circ} : [a, a', a'']_{\circ} \mapsto_{\mathfrak{A} \times_{SMC3} \mathfrak{B} \times_{SMC3} \mathfrak{C}} [b, b', b'']_{\circ}$
shows
$[g, g', g'']_{\circ} \circ_{A\mathfrak{A} \times_{SMC3} \mathfrak{B} \times_{SMC3} \mathfrak{C}} [f, f', f'']_{\circ} =$
$[g \circ_{A\mathfrak{A}} f, g' \circ_{A\mathfrak{B}} f', g'' \circ_{A\mathfrak{C}} f'']_{\circ}$
proof–
have
$[g, g', g'']_{\circ} \circ_{A\mathfrak{A} \times_{SMC3} \mathfrak{B} \times_{SMC3} \mathfrak{C}} [f, f', f'']_{\circ} =$
$(\lambda i \in_{\circ} 3_{\mathbb{N}}.\ [g, g', g'']_{\circ}(\!|i|\!) \circ_{A\,if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i} [f, f', f'']_{\circ}(\!|i|\!))$
by
(
rule smc-prod-Comp-app[
OF assms[*unfolded smc-prod-3-def*], *folded smc-prod-3-def*
]
)
also have
$(\lambda i \in_{\circ} 3_{\mathbb{N}}.\ [g, g', g'']_{\circ}(\!|i|\!) \circ_{A\,if3\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{C}\ i} [f, f', f'']_{\circ}(\!|i|\!)) =$
$[g \circ_{A\mathfrak{A}} f, g' \circ_{A\mathfrak{B}} f', g'' \circ_{A\mathfrak{C}} f'']_{\circ}$
proof(*rule vsv-eqI*, *unfold vdomain-VLambda*)
fix *i* **assume** $i \in_{\circ} 3_{\mathbb{N}}$

```
    then consider ‹i = 0› | ‹i = 1_ℕ› | ‹i = 2_ℕ› unfolding three by auto
    then show
      (λi∈₀3_ℕ. [g, g', g'']₀(|i|) ∘_A if3 𝔄 𝔅 ℭ i [f, f', f'']₀(|i|))(|i|) =
        [g ∘_A𝔄 f, g' ∘_A𝔅 f', g'' ∘_Aℭ f'']₀(|i|)
      by cases (simp-all add: three nat-omega-simps)
  qed (auto simp: three nat-omega-simps)
  finally show ?thesis by simp
qed

end
```

4.9 Subsemicategory

4.9.1 Background

named-theorems *smc-sub-cs-intros*
named-theorems *smc-sub-bw-cs-intros*
named-theorems *smc-sub-fw-cs-intros*
named-theorems *smc-sub-bw-cs-simps*

4.9.2 Simple subsemicategory

Definition and elementary properties

See Chapter I-3 in [39].

locale *subsemicategory* =
  sdg: *semicategory* α $\mathfrak{B}$ + *dg*: *semicategory* α $\mathfrak{C}$ **for** α $\mathfrak{B}$ $\mathfrak{C}$ +
  assumes *subsmc-subdigraph*[*slicing-intros*]: *smc-dg* $\mathfrak{B}$ $\subseteq_{DG\alpha}$ *smc-dg* $\mathfrak{C}$
    and *subsmc-Comp*[*smc-sub-fw-cs-intros*]:
      $[\![\ g : b \mapsto_{\mathfrak{B}} c;\ f : a \mapsto_{\mathfrak{B}} b\ ]\!] \Longrightarrow g \circ_{A\mathfrak{B}} f = g \circ_{A\mathfrak{C}} f$

abbreviation *is-subsemicategory* ($\langle$(-/ $\subseteq_{SMC^1}$ -)$\rangle$ [51, 51] 50)
  where $\mathfrak{B} \subseteq_{SMC\alpha} \mathfrak{C} \equiv$ *subsemicategory* α $\mathfrak{B}$ $\mathfrak{C}$

lemmas [*smc-sub-fw-cs-intros*] = *subsemicategory.subsmc-Comp*

Rules.

lemma (**in** *subsemicategory*) *subsemicategory-axioms'*[*smc-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
  shows $\mathfrak{B}' \subseteq_{SMC_{\alpha'}} \mathfrak{C}$
  unfolding *assms* **by** (*rule subsemicategory-axioms*)

lemma (**in** *subsemicategory*) *subsemicategory-axioms''*[*smc-cs-intros*]:
  assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
  shows $\mathfrak{B} \subseteq_{SMC_{\alpha'}} \mathfrak{C}'$
  unfolding *assms* **by** (*rule subsemicategory-axioms*)

mk-ide rf *subsemicategory-def*[*unfolded subsemicategory-axioms-def*]
  |*intro subsemicategoryI*|
  |*dest subsemicategoryD*[*dest*]|
  |*elim subsemicategoryE*[*elim*!]|

lemmas [*smc-sub-cs-intros*] = *subsemicategoryD*(1,2)

lemma *subsemicategoryI'*:
  assumes *semicategory* α $\mathfrak{B}$
    and *semicategory* 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 $\bigwedge b\ c\ g\ a\ f.\ [\![\ g : b \mapsto_{\mathfrak{B}} c;\ f : a \mapsto_{\mathfrak{B}} b\ ]\!] \Longrightarrow$
      $g \circ_{A\mathfrak{B}} f = g \circ_{A\mathfrak{C}} f$
  shows $\mathfrak{B} \subseteq_{SMC\alpha} \mathfrak{C}$
proof–
  interpret $\mathfrak{B}$: *semicategory* α $\mathfrak{B}$ **by** (*rule assms*(1))
  interpret $\mathfrak{C}$: *semicategory* α $\mathfrak{C}$ **by** (*rule assms*(2))
  show *?thesis*
    by
      (
        intro subsemicategoryI subdigraphI,

```
    unfold slicing-simps;
    (intro 𝔅.smc-digraph ℭ.smc-digraph assms)?
  )
qed
```

Subsemicategory is a subdigraph.

```
context subsemicategory
begin

interpretation subdg: subdigraph α ‹smc-dg 𝔅› ‹smc-dg ℭ›
  by (rule subsmc-subdigraph)

lemmas-with [unfolded slicing-simps]:
  subsmc-Obj-vsubset = subdg.subdg-Obj-vsubset
  and subsmc-is-arr-vsubset = subdg.subdg-is-arr-vsubset
  and subsmc-subdigraph-op-dg-op-dg = subdg.subdg-subdigraph-op-dg-op-dg
  and subsmc-objD = subdg.subdg-objD
  and subsmc-arrD = subdg.subdg-arrD
  and subsmc-dom-simp = subdg.subdg-dom-simp
  and subsmc-cod-simp = subdg.subdg-cod-simp
  and subsmc-is-arrD = subdg.subdg-is-arrD
  and subsmc-dghm-inc-op-dg-is-dghm = subdg.subdg-dghm-inc-op-dg-is-dghm
  and subsmc-op-dg-dghm-inc = subdg.subdg-op-dg-dghm-inc
  and subsmc-inc-is-ft-dghm-axioms = subdg.inc.is-ft-dghm-axioms

end

lemmas subsmc-subdigraph-op-dg-op-dg[intro] =
  subsemicategory.subsmc-subdigraph-op-dg-op-dg

lemmas [smc-sub-fw-cs-intros] =
  subsemicategory.subsmc-Obj-vsubset
  subsemicategory.subsmc-is-arr-vsubset
  subsemicategory.subsmc-objD
  subsemicategory.subsmc-arrD
  subsemicategory.subsmc-is-arrD

lemmas [smc-sub-bw-cs-simps] =
  subsemicategory.subsmc-dom-simp
  subsemicategory.subsmc-cod-simp
```

The opposite subsemicategory.

```
lemma (in subsemicategory) subsmc-subsemicategory-op-smc:
  op-smc 𝔅 ⊆_{SMC α} op-smc ℭ
proof(rule subsemicategoryI)
  fix g b c f a assume prems: g : b ↦_{op-smc 𝔅} c f : a ↦_{op-smc 𝔅} b
  then have g : c ↦_𝔅 b and f : b ↦_𝔅 a
    by (simp-all add: smc-op-simps)
  with subsemicategory-axioms have g: g : c ↦_ℭ b and f: f : b ↦_ℭ a
    by (cs-concl cs-shallow cs-intro: smc-sub-fw-cs-intros)+
  from dg.op-smc-Comp[OF this(2,1)] have g ∘_{A op-smc ℭ} f = f ∘_{Aℭ} g.
  with prems show g ∘_{A op-smc 𝔅} f = g ∘_{A op-smc ℭ} f
    by (simp add: smc-op-simps subsmc-Comp)
qed
  (
    auto
      simp:
        smc-op-simps slicing-commute[symmetric] subsmc-subdigraph-op-dg-op-dg
```

```
    intro: smc-op-intros
  )

lemmas subsmc-subsemicategory-op-smc[intro, smc-op-intros] =
  subsemicategory.subsmc-subsemicategory-op-smc
```

Further rules.

```
lemma (in subsemicategory) subsmc-Comp-simp:
  assumes g : b ↦_𝔅 c and f : a ↦_𝔅 b
  shows g ∘_A𝔅 f = g ∘_A𝔈 f
  using assms subsmc-Comp by auto

lemmas [smc-sub-bw-cs-simps] = subsemicategory.subsmc-Comp-simp

lemma (in subsemicategory) subsmc-is-idem-arrD:
  assumes f : ↦_ide𝔅 b
  shows f : ↦_ide𝔈 b
  using assms subsemicategory-axioms
  by (intro is-idem-arrI; elim is-idem-arrE)
    (
      cs-concl cs-shallow
        cs-simp: smc-sub-bw-cs-simps[symmetric] cs-intro: smc-sub-fw-cs-intros
    )

lemmas [smc-sub-fw-cs-intros] = subsemicategory.subsmc-is-idem-arrD
```

Subsemicategory relation is a partial order

```
lemma subsmc-refl:
  assumes semicategory α 𝔄
  shows 𝔄 ⊆_SMCα 𝔄
proof–
  interpret semicategory α 𝔄 by (rule assms)
  show ?thesis
    by (auto intro: smc-cs-intros slicing-intros subdg-refl subsemicategoryI)
qed

lemma subsmc-trans[trans]:
  assumes 𝔄 ⊆_SMCα 𝔅 and 𝔅 ⊆_SMCα 𝔈
  shows 𝔄 ⊆_SMCα 𝔈
proof–
  interpret 𝔄𝔅: subsemicategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅𝔈: subsemicategory α 𝔅 𝔈 by (rule assms(2))
  show ?thesis
  proof(rule subsemicategoryI)
    from 𝔄𝔅.subsmc-subdigraph 𝔅𝔈.subsmc-subdigraph
    show smc-dg 𝔄 ⊆_DGα smc-dg 𝔈 by (meson subdg-trans)
    show g ∘_A𝔄 f = g ∘_A𝔈 f
      if g : b ↦_𝔄 c and f : a ↦_𝔄 b for g b c f a
      by
        (
          metis
            that
            𝔄𝔅.subsmc-is-arr-vsubset
            𝔄𝔅.subsmc-Comp-simp
            𝔅𝔈.subsmc-Comp-simp
        )
  qed (auto intro: smc-cs-intros)
```

qed

lemma *subsmc-antisym*:
assumes $\mathfrak{A} \subseteq_{SMC\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{SMC\alpha} \mathfrak{A}$
shows $\mathfrak{A} = \mathfrak{B}$
proof–
interpret $\mathfrak{AB}$: *subsemicategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(1))
interpret $\mathfrak{BA}$: *subsemicategory* α $\mathfrak{B}$ $\mathfrak{A}$ **by** (*rule assms*(2))
show *?thesis*
proof(*rule smc-eqI*)
from *subdg-antisym*[*OF* $\mathfrak{AB}$.*subsmc-subdigraph* $\mathfrak{BA}$.*subsmc-subdigraph*] **have**
smc-dg $\mathfrak{A}(\!|Obj|\!)$ = *smc-dg* $\mathfrak{B}(\!|Obj|\!)$ *smc-dg* $\mathfrak{A}(\!|Arr|\!)$ = *smc-dg* $\mathfrak{B}(\!|Arr|\!)$
by *simp-all*
then show $\mathfrak{A}(\!|Obj|\!) = \mathfrak{B}(\!|Obj|\!)$ **and** *Arr*: $\mathfrak{A}(\!|Arr|\!) = \mathfrak{B}(\!|Arr|\!)$
unfolding *slicing-simps* **by** *simp-all*
show $\mathfrak{A}(\!|Dom|\!) = \mathfrak{B}(\!|Dom|\!)$
by (*rule vsv-eqI*) (*auto simp*: *smc-cs-simps* $\mathfrak{AB}$.*subsmc-dom-simp Arr*)
show $\mathfrak{A}(\!|Cod|\!) = \mathfrak{B}(\!|Cod|\!)$
by (*rule vsv-eqI*) (*auto simp*: *smc-cs-simps* $\mathfrak{BA}$.*subsmc-cod-simp Arr*)
show $\mathfrak{A}(\!|Comp|\!) = \mathfrak{B}(\!|Comp|\!)$
proof(*rule vsv-eqI*)
show $\mathcal{D}_\circ\ (\mathfrak{A}(\!|Comp|\!)) = \mathcal{D}_\circ\ (\mathfrak{B}(\!|Comp|\!))$
proof(*intro vsubset-antisym vsubsetI*)
fix *gf* **assume** $gf \in_\circ \mathcal{D}_\circ\ (\mathfrak{A}(\!|Comp|\!))$
then obtain *g f b c a*
where *gf-def*: $gf = [g, f]_\circ$
and *g*: $g : b \mapsto_{\mathfrak{A}} c$
and *f*: $f : a \mapsto_{\mathfrak{A}} b$
by (*auto simp*: $\mathfrak{AB}$.*sdg.smc-Comp-vdomain*)
from *g f* **show** $gf \in_\circ \mathcal{D}_\circ\ (\mathfrak{B}(\!|Comp|\!))$
unfolding *gf-def* **by** (*meson* $\mathfrak{AB}$.*dg.smc-Comp-vdomainI* $\mathfrak{AB}$.*subsmc-is-arrD*)
next
fix *gf* **assume** $gf \in_\circ \mathcal{D}_\circ\ (\mathfrak{B}(\!|Comp|\!))$
then obtain *g f b c a*
where *gf-def*: $gf = [g, f]_\circ$
and *g*: $g : b \mapsto_{\mathfrak{B}} c$
and *f*: $f : a \mapsto_{\mathfrak{B}} b$
by (*auto simp*: $\mathfrak{AB}$.*dg.smc-Comp-vdomain*)
from *g f* **show** $gf \in_\circ \mathcal{D}_\circ\ (\mathfrak{A}(\!|Comp|\!))$
unfolding *gf-def* **by** (*meson* $\mathfrak{AB}$.*sdg.smc-Comp-vdomainI* $\mathfrak{BA}$.*subsmc-is-arrD*)
qed
show $a \in_\circ \mathcal{D}_\circ\ (\mathfrak{A}(\!|Comp|\!)) \Longrightarrow \mathfrak{A}(\!|Comp|\!)(\!|a|\!) = \mathfrak{B}(\!|Comp|\!)(\!|a|\!)$ **for** *a*
by (*metis* $\mathfrak{AB}$.*sdg.smc-Comp-vdomain* $\mathfrak{AB}$.*subsmc-Comp-simp*)
qed *auto*
qed (*auto intro*: *smc-cs-intros*)
qed

4.9.3 Inclusion semifunctor

Definition and elementary properties

See Chapter I-3 in [39].

abbreviation (*input*) *smcf-inc* :: $V \Rightarrow V \Rightarrow V$
where *smcf-inc* ≡ *dghm-inc*

Slicing.

lemma *dghm-smcf-inc*[*slicing-commute*]:
dghm-inc (*smc-dg* $\mathfrak{B}$) (*smc-dg* $\mathfrak{C}$) = *smcf-dghm* (*smcf-inc* $\mathfrak{B}$ $\mathfrak{C}$)

unfolding
smcf-dghm-def dghm-inc-def smc-dg-def dg-field-simps dghm-field-simps
by (*simp-all add*: *nat-omega-simps*)

Elementary properties.

lemmas [*smc-cs-simps*] =
dghm-inc-ObjMap-app
dghm-inc-ArrMap-app

Canonical inclusion semifunctor associated with a subsemicategory

sublocale *subsemicategory* ⊆ *inc*: *is-ft-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ ‹*smcf-inc* $\mathfrak{B}$ $\mathfrak{C}$›
proof(*rule is-ft-semifunctorI*)
show *smcf-inc* $\mathfrak{B}$ $\mathfrak{C}$: $\mathfrak{B}$ $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{C}$
proof(*rule is-semifunctorI*)
show *vfsequence* (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$) **unfolding** *dghm-inc-def* **by** *auto*
show *vcard* (*dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$) = $4_{\mathbb{N}}$
unfolding *dghm-inc-def* **by** (*simp add*: *nat-omega-simps*)
fix *g b c f a* **assume** *prems*: *g* : *b* $\mapsto_{\mathfrak{B}}$ *c f* : *a* $\mapsto_{\mathfrak{B}}$ *b*
then have *g* $\circ_{A\mathfrak{B}}$ *f* : *a* $\mapsto_{\mathfrak{B}}$ *c* **by** (*simp add*: *smc-cs-intros*)
with *subsemicategory-axioms prems* **have** [*simp*]:
vid-on ($\mathfrak{B}$(|*Arr*|))(|*g* $\circ_{A\mathfrak{B}}$ *f*|) = *g* $\circ_{A\mathfrak{C}}$ *f*
by (*auto simp*: *smc-sub-bw-cs-simps*)
from *prems* **show** *dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|*g* $\circ_{A\mathfrak{B}}$ *f*|) =
dghm-inc $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|*g*|) $\circ_{A\mathfrak{C}}$ *dghm-inc* $\mathfrak{B}$ $\mathfrak{C}$(|*ArrMap*|)(|*f*|)
by
(
cs-concl
cs-simp: *smc-cs-simps* **cs-intro**: *smc-cs-intros smc-sub-fw-cs-intros*
)
qed
(
insert subsmc-inc-is-ft-dghm-axioms,
auto simp: *slicing-commute*[*symmetric*] *dghm-inc-components smc-cs-intros*
)
qed (*auto simp*: *slicing-commute*[*symmetric*] *subsmc-inc-is-ft-dghm-axioms*)

lemmas (**in** *subsemicategory*) *subsmc-smcf-inc-is-ft-semifunctor* =
inc.is-ft-semifunctor-axioms

Inclusion semifunctor for the opposite semicategories

lemma (**in** *subsemicategory*)
subsemicategory-smcf-inc-op-smc-is-semifunctor[*smc-sub-cs-intros*]:
smcf-inc (*op-smc* $\mathfrak{B}$) (*op-smc* $\mathfrak{C}$) : *op-smc* $\mathfrak{B}$ $\mapsto\mapsto_{SMC.faithful\alpha}$ *op-smc* $\mathfrak{C}$
by
(
intro
subsemicategory.subsmc-smcf-inc-is-ft-semifunctor
subsmc-subsemicategory-op-smc
)

lemmas [*smc-sub-cs-intros*] =
subsemicategory.subsemicategory-smcf-inc-op-smc-is-semifunctor

lemma (**in** *subsemicategory*) *subdg-op-smc-smcf-inc*[*smc-op-simps*]:
op-smcf (*smcf-inc* $\mathfrak{B}$ $\mathfrak{C}$) = *smcf-inc* (*op-smc* $\mathfrak{B}$) (*op-smc* $\mathfrak{C}$)
by

(
rule smcf-eqI[*of* α ‹*op-smc* $\mathfrak{B}$› ‹*op-smc* $\mathfrak{C}$›],
unfold smc-op-simps dghm-inc-components
)
(
auto simp:
is-ft-semifunctorD
subsemicategory-smcf-inc-op-smc-is-semifunctor
inc.is-semifunctor-op
)

lemmas [*smc-op-simps*] = *subsemicategory.subdg-op-smc-smcf-inc*

4.9.4 Full subsemicategory

See Chapter I-3 in [39].

locale *fl-subsemicategory* = *subsemicategory* +
assumes *fl-subsemicategory-fl-subdigraph*: *smc-dg* $\mathfrak{B}$ $\subseteq_{DG.full\alpha}$ *smc-dg* $\mathfrak{C}$

abbreviation *is-fl-subsemicategory* (‹(-/ $\subseteq_{SMC.full1}$ -)› [51, 51] 50)
where $\mathfrak{B}$ $\subseteq_{SMC.full\alpha}$ $\mathfrak{C}$ ≡ *fl-subsemicategory* α $\mathfrak{B}$ $\mathfrak{C}$

Rules.

lemma (**in** *fl-subsemicategory*) *fl-subsemicategory-axioms'*[*smc-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{B}' \subseteq_{SMC.full_{\alpha'}} \mathfrak{C}$
unfolding *assms* **by** (*rule fl-subsemicategory-axioms*)

lemma (**in** *fl-subsemicategory*) *fl-subsemicategory-axioms''*[*smc-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{B} \subseteq_{SMC.full_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule fl-subsemicategory-axioms*)

mk-ide rf *fl-subsemicategory-def*[*unfolded fl-subsemicategory-axioms-def*]
|*intro fl-subsemicategoryI*|
|*dest fl-subsemicategoryD*[*dest*]|
|*elim fl-subsemicategoryE*[*elim*!]|

lemmas [*smc-sub-cs-intros*] = *fl-subsemicategoryD*(1)

Full subsemicategory.

sublocale *fl-subsemicategory* ⊆ *inc*: *is-fl-semifunctor* α $\mathfrak{B}$ $\mathfrak{C}$ ‹*smcf-inc* $\mathfrak{B}$ $\mathfrak{C}$›
using *fl-subsemicategory-fl-subdigraph inc.is-semifunctor-axioms*
by (*intro is-fl-semifunctorI*) (*auto simp*: *slicing-commute*[*symmetric*])

4.9.5 Wide subsemicategory

Definition and elementary properties

See [3][4]).

locale *wide-subsemicategory* = *subsemicategory* +
assumes *wide-subsmc-wide-subdigraph*: *smc-dg* $\mathfrak{B}$ $\subseteq_{DG.wide\alpha}$ *smc-dg* $\mathfrak{C}$

abbreviation *is-wide-subsemicategory* (‹(-/ $\subseteq_{SMC.wide1}$ -)› [51, 51] 50)
where $\mathfrak{B}$ $\subseteq_{SMC.wide\alpha}$ $\mathfrak{C}$ ≡ *wide-subsemicategory* α $\mathfrak{B}$ $\mathfrak{C}$

[4] https://ncatlab.org/nlab/show/wide+subcategory

Rules.

lemma (in *wide-subsemicategory*) *wide-subsemicategory-axioms'*[*smc-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{B}' = \mathfrak{B}$
shows $\mathfrak{B}' \subseteq_{SMC.wide_{\alpha'}} \mathfrak{C}$
unfolding *assms* **by** (*rule wide-subsemicategory-axioms*)

lemma (in *wide-subsemicategory*) *wide-subsemicategory-axioms''*[*smc-cs-intros*]:
assumes $\alpha' = \alpha$ **and** $\mathfrak{C}' = \mathfrak{C}$
shows $\mathfrak{B} \subseteq_{SMC.wide_{\alpha'}} \mathfrak{C}'$
unfolding *assms* **by** (*rule wide-subsemicategory-axioms*)

mk-ide rf *wide-subsemicategory-def*[*unfolded wide-subsemicategory-axioms-def*]
|*intro wide-subsemicategoryI*|
|*dest wide-subsemicategoryD*[*dest*]|
|*elim wide-subsemicategoryE*[*elim*!]|

lemmas [*smc-sub-cs-intros*] = *wide-subsemicategoryD*(1)

Wide subsemicategory is wide subdigraph.

context *wide-subsemicategory*
begin

interpretation *wide-subdg*: *wide-subdigraph* α ‹*smc-dg* $\mathfrak{B}$› ‹*smc-dg* $\mathfrak{C}$›
by (*rule wide-subsmc-wide-subdigraph*)

lemmas-with [*unfolded slicing-simps*]:
wide-subsmc-Obj[*dg-sub-bw-cs-intros*] = *wide-subdg.wide-subdg-Obj*
and *wide-subsmc-obj-eq*[*dg-sub-bw-cs-simps*] = *wide-subdg.wide-subdg-obj-eq*

end

lemmas [*dg-sub-bw-cs-intros*] = *wide-subsemicategory.wide-subsmc-Obj*
lemmas [*dg-sub-bw-cs-simps*] = *wide-subsemicategory.wide-subsmc-obj-eq*

The wide subsemicategory relation is a partial order

lemma *wide-subsmc-refl*:
assumes *semicategory* α $\mathfrak{A}$
shows $\mathfrak{A} \subseteq_{SMC.wide\alpha} \mathfrak{A}$
proof–
interpret *semicategory* α $\mathfrak{A}$ **by** (*rule assms*)
show *?thesis*
by
(
auto intro:
assms
slicing-intros
wide-subdg-refl
wide-subsemicategoryI
subsmc-refl
)
qed

lemma *wide-subsmc-trans*[*trans*]:
assumes $\mathfrak{A} \subseteq_{SMC.wide\alpha} \mathfrak{B}$ **and** $\mathfrak{B} \subseteq_{SMC.wide\alpha} \mathfrak{C}$
shows $\mathfrak{A} \subseteq_{SMC.wide\alpha} \mathfrak{C}$
proof–
interpret $\mathfrak{AB}$: *wide-subsemicategory* α $\mathfrak{A}$ $\mathfrak{B}$ **by** (*rule assms*(1))

```
  interpret 𝔅𝔈: wide-subsemicategory α 𝔅 𝔈 by (rule assms(2))
  show ?thesis
    by
      (
        intro
          wide-subsemicategoryI
          subsmc-trans[
            OF 𝔄𝔅.subsemicategory-axioms 𝔅𝔈.subsemicategory-axioms
            ],
        rule wide-subdg-trans,
        rule 𝔄𝔅.wide-subsmc-wide-subdigraph,
        rule 𝔅𝔈.wide-subsmc-wide-subdigraph
      )
qed

lemma wide-subsmc-antisym:
  assumes 𝔄 ⊆_{SMC.wide α} 𝔅 and 𝔅 ⊆_{SMC.wide α} 𝔄
  shows 𝔄 = 𝔅
proof–
  interpret 𝔄𝔅: wide-subsemicategory α 𝔄 𝔅 by (rule assms(1))
  interpret 𝔅𝔄: wide-subsemicategory α 𝔅 𝔄 by (rule assms(2))
  show ?thesis
    by
      (
        rule subsmc-antisym[
          OF 𝔄𝔅.subsemicategory-axioms 𝔅𝔄.subsemicategory-axioms
          ]
      )
qed
```

4.10 Simple semicategories

4.10.1 Background

The section presents a variety of simple semicategories, such as the empty semicategory 0 and a semicategory with one object and one arrow 1. All of the entities presented in this section are generalizations of certain simple categories, whose definitions can be found in [39].

4.10.2 Empty semicategory 0

Definition and elementary properties

See Chapter I-2 in [39].

definition *smc*-0 :: *V*
  where *smc*-0 = [0, 0, 0, 0, 0]$_\circ$

Components.

lemma *smc*-0-*components*:
  shows *smc*-0(|*Obj*|) = 0
    and *smc*-0(|*Arr*|) = 0
    and *smc*-0(|*Dom*|) = 0
    and *smc*-0(|*Cod*|) = 0
    and *smc*-0(|*Comp*|) = 0
  unfolding *smc*-0-*def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-smc*-0: *smc-dg smc*-0 = *dg*-0
  unfolding *smc-dg-def smc*-0-*def dg*-0-*def dg-field-simps*
  by (*simp add*: *nat-omega-simps*)

lemmas-with (**in** $\mathcal{Z}$) [*folded smc-dg-smc*-0, *unfolded slicing-simps*]:
  smc-0-*is-arr-iff* = *dg*-0-*is-arr-iff*

0 is a semicategory

lemma (**in** $\mathcal{Z}$) *semicategory-smc*-0[*smc-cs-intros*]: *semicategory* α *smc*-0
proof(*intro semicategoryI*)
  show *vfsequence smc*-0 **unfolding** *smc*-0-*def* **by** (*simp add*: *nat-omega-simps*)
  show *vcard smc*-0 = $5_{\mathbb{N}}$ **unfolding** *smc*-0-*def* **by** (*simp add*: *nat-omega-simps*)
  show *digraph* α (*smc-dg smc*-0)
    by (*simp add*: *smc-dg-smc*-0 $\mathcal{Z}$.*digraph-dg*-0 $\mathcal{Z}$-*axioms*)
qed (*auto simp*: *smc*-0-*components smc*-0-*is-arr-iff*)

lemmas [*smc-cs-intros*] = $\mathcal{Z}$.*semicategory-smc*-0

Opposite of the semicategory 0

lemma *op-smc-smc*-0[*smc-op-simps*]: *op-smc* (*smc*-0) = *smc*-0
proof(*rule smc-dg-eqI*)
  define β **where** $\beta = \omega + \omega$
  interpret β: $\mathcal{Z}$ β **unfolding** β-*def* **by** (*rule* $\mathcal{Z}$-$\omega\omega$)
  show *semicategory* β (*op-smc smc*-0)
    by (*cs-concl* **cs-shallow cs-intro**: *smc-cs-intros smc-op-intros*)
  show *semicategory* β *smc*-0 **by** (*cs-concl* **cs-shallow cs-intro**: *smc-cs-intros*)
qed
  (
    simp-all add:

smc-0-*components op-smc-components smc-dg-smc*-0
slicing-commute[*symmetric*] *dg-op-simps*
)

A semicategory without objects is empty

lemma (**in** *semicategory*) *smc-smc*-0-*if-Obj*-0:
assumes $\mathfrak{C}(\!|Obj|\!) = 0$
shows $\mathfrak{C} = smc\text{-}0$
by (*rule smc-eqI*[*of* α])
(
auto simp:
smc-cs-intros
assms
semicategory-smc-0
smc-0-*components*
smc-Arr-vempty-if-Obj-vempty
smc-Cod-vempty-if-Arr-vempty
smc-Dom-vempty-if-Arr-vempty
smc-Comp-vempty-if-Arr-vempty
)

4.10.3 Empty semifunctor

An empty semifunctor is defined as a semifunctor between an empty semicategory and an arbitrary semicategory.

Definition and elementary properties

definition *smcf*-0 :: $V \Rightarrow V$
where *smcf*-0 $\mathfrak{A} = [0, 0, smc\text{-}0, \mathfrak{A}]_\circ$

Components.

lemma *smcf*-0-*components*:
shows *smcf*-0 $\mathfrak{A}(\!|ObjMap|\!) = 0$
and *smcf*-0 $\mathfrak{A}(\!|ArrMap|\!) = 0$
and *smcf*-0 $\mathfrak{A}(\!|HomDom|\!) = smc\text{-}0$
and *smcf*-0 $\mathfrak{A}(\!|HomCod|\!) = \mathfrak{A}$
unfolding *smcf*-0-*def dghm-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smcf-dghm-smcf*-0: *smcf-dghm* (*smcf*-0 $\mathfrak{A}$) = *dghm*-0 (*smc-dg* $\mathfrak{A}$)
unfolding
smcf-dghm-def smcf-0-*def dg*-0-*def smc*-0-*def dghm*-0-*def smc-dg-def*
dg-field-simps dghm-field-simps
by (*simp add*: *nat-omega-simps*)

Opposite empty semicategory homomorphism.

lemma *op-smcf-smcf*-0: *op-smcf* (*smcf*-0 $\mathfrak{C}$) = *smcf*-0 (*op-smc* $\mathfrak{C}$)
unfolding
smcf-0-*def op-smc-def op-smcf-def smc*-0-*def dghm-field-simps dg-field-simps*
by (*simp add*: *nat-omega-simps*)

Object map

lemma *smcf*-0-*ObjMap-vsv*[*smc-cs-intros*]: *vsv* (*smcf*-0 $\mathfrak{C}(\!|ObjMap|\!)$)
unfolding *smcf*-0-*components* **by** *simp*

Arrow map

lemma *smcf*-0-*ArrMap-vsv*[*smc-cs-intros*]: *vsv* (*smcf*-0 $\mathfrak{C}(\lfloor ArrMap \rfloor)$)
unfolding *smcf*-0-*components* **by** *simp*

Empty semifunctor is a faithful semifunctor

lemma (**in** $\mathcal{Z}$) *smcf*-0-*is-ft-semifunctor*:
assumes *semicategory* α $\mathfrak{A}$
shows *smcf*-0 $\mathfrak{A}$: *smc*-0 $\mapsto\mapsto_{SMC.faithful\alpha}$ $\mathfrak{A}$
proof(*rule is-ft-semifunctorI*)
show *smcf*-0 $\mathfrak{A}$: *smc*-0 $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{A}$
proof(*rule is-semifunctorI*, *unfold smc-dg-smc*-0 *smcf-dghm-smcf*-0)
show *vfsequence* (*smcf*-0 $\mathfrak{A}$) **unfolding** *smcf*-0-*def* **by** *simp*
show *vcard* (*smcf*-0 $\mathfrak{A}$) = $4_{\mathbb{N}}$
unfolding *smcf*-0-*def* **by** (*simp add*: *nat-omega-simps*)
show *dghm*-0 (*smc-dg* $\mathfrak{A}$) : *dg*-0 $\mapsto\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{A}$
by
(
simp add:
assms
dghm-0-*is-ft-dghm*
is-ft-dghm.axioms(1)
semicategory.smc-digraph
)
qed (*auto simp*: *assms semicategory-smc*-0 *smcf*-0-*components smc*-0-*is-arr-iff*)
show *smcf-dghm* (*smcf*-0 $\mathfrak{A}$) : *smc-dg smc*-0 $\mapsto\mapsto_{DG.faithful\alpha}$ *smc-dg* $\mathfrak{A}$
by
(
auto simp:
assms
$\mathcal{Z}$.*dghm*-0-*is-ft-dghm*
$\mathcal{Z}$-*axioms*
smc-dg-smc-0
semicategory.smc-digraph
smcf-dghm-smcf-0
)
qed

lemma (**in** $\mathcal{Z}$) *smcf*-0-*is-ft-semifunctor'*[*smcf-cs-intros*]:
assumes *semicategory* α $\mathfrak{A}$
and $\mathfrak{B}' = \mathfrak{A}$
and $\mathfrak{A}' =$ *smc*-0
shows *smcf*-0 $\mathfrak{A}$: $\mathfrak{A}' \mapsto\mapsto_{SMC.faithful\alpha}$ $\mathfrak{B}'$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule smcf*-0-*is-ft-semifunctor*)

lemmas [*smcf-cs-intros*] = $\mathcal{Z}$.*smcf*-0-*is-ft-semifunctor'*

lemma (**in** $\mathcal{Z}$) *smcf*-0-*is-semifunctor*:
assumes *semicategory* α $\mathfrak{A}$
shows *smcf*-0 $\mathfrak{A}$: *smc*-0 $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{A}$
using *smcf*-0-*is-ft-semifunctor*[*OF assms*] **by** *auto*

lemma (**in** $\mathcal{Z}$) *smcf*-0-*is-semifunctor'*[*smc-cs-intros*]:
assumes *semicategory* α $\mathfrak{A}$
and $\mathfrak{B}' = \mathfrak{A}$
and $\mathfrak{A}' =$ *smc*-0
shows *smcf*-0 $\mathfrak{A}$: $\mathfrak{A}' \mapsto\mapsto_{SMC\alpha}$ $\mathfrak{B}'$
using *assms*(1) **unfolding** *assms*(2,3) **by** (*rule smcf*-0-*is-semifunctor*)

lemmas [*smc-cs-intros*] = $\mathcal{Z}$.*smcf*-0-*is-semifunctor'*

Further properties

lemma *is-semifunctor-is-smcf*-0-*if-smc*-0:
assumes $\mathfrak{F}$: *smc*-0 $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{C}$
shows $\mathfrak{F}$ = *smcf*-0 $\mathfrak{C}$
proof(*rule smcf-dghm-eqI*)
interpret $\mathfrak{F}$: *is-semifunctor* α *smc*-0 $\mathfrak{C}$ $\mathfrak{F}$ **by** (*rule assms*(1))
show $\mathfrak{F}$: *smc*-0 $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{C}$ **by** (*rule assms*(1))
then have *dom-lhs*: $\mathcal{D}_\circ$ ($\mathfrak{F}$(|*ObjMap*|)) = 0 $\mathcal{D}_\circ$ ($\mathfrak{F}$(|*ArrMap*|)) = 0
by (*cs-concl* **cs-simp**: *smc-cs-simps smc*-0-*components*)+
show *smcf*-0 $\mathfrak{C}$: *smc*-0 $\mapsto\mapsto_{SMC\alpha}$ $\mathfrak{C}$ **by** (*cs-concl* **cs-intro**: *smc-cs-intros*)
show *smcf-dghm* $\mathfrak{F}$ = *smcf-dghm* (*smcf*-0 $\mathfrak{C}$)
unfolding *smcf-dghm-smcf*-0
by
(
rule is-dghm-is-dghm-0-*if-dg*-0,
rule $\mathfrak{F}$.*smcf-is-dghm*[*unfolded slicing-simps smc-dg-smc*-0]
)
qed *simp-all*

4.10.4 Empty natural transformation of semifunctors

Definition and elementary properties

definition *ntsmcf*-0 :: *V* ⇒ *V*
where *ntsmcf*-0 $\mathfrak{C}$ = [0, *smcf*-0 $\mathfrak{C}$, *smcf*-0 $\mathfrak{C}$, *smc*-0, $\mathfrak{C}$]$_\circ$

Components.

lemma *ntsmcf*-0-*components*:
shows *ntsmcf*-0 $\mathfrak{C}$(|*NTMap*|) = 0
and [*smc-cs-simps*]: *ntsmcf*-0 $\mathfrak{C}$(|*NTDom*|) = *smcf*-0 $\mathfrak{C}$
and [*smc-cs-simps*]: *ntsmcf*-0 $\mathfrak{C}$(|*NTCod*|) = *smcf*-0 $\mathfrak{C}$
and [*smc-cs-simps*]: *ntsmcf*-0 $\mathfrak{C}$(|*NTDGDom*|) = *smc*-0
and [*smc-cs-simps*]: *ntsmcf*-0 $\mathfrak{C}$(|*NTDGCod*|) = $\mathfrak{C}$
unfolding *ntsmcf*-0-*def nt-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *ntsmcf-tdghm-ntsmcf*-0: *ntsmcf-tdghm* (*ntsmcf*-0 $\mathfrak{A}$) = *tdghm*-0 (*smc-dg* $\mathfrak{A}$)
unfolding
ntsmcf-tdghm-def ntsmcf-0-*def tdghm*-0-*def smcf-dghm-def*
smcf-0-*def smc-dg-def smc*-0-*def dghm*-0-*def dg*-0-*def*
dg-field-simps dghm-field-simps nt-field-simps
by (*simp add*: *nat-omega-simps*)

Duality.

lemma *op-ntsmcf-ntsmcf*-0: *op-ntsmcf* (*ntsmcf*-0 $\mathfrak{C}$) = *ntsmcf*-0 (*op-smc* $\mathfrak{C}$)
by
(
simp-all add:
op-ntsmcf-def ntsmcf-0-*def op-smc-def op-smcf-smcf*-0 *smc*-0-*def*
nt-field-simps dg-field-simps nat-omega-simps
)

Natural transformation map

lemma *ntsmcf*-0-*NTMap-vsv*[*smc-cs-intros*]: *vsv* (*ntsmcf*-0 $\mathfrak{C}$(|*NTMap*|))

unfolding *ntsmcf*-0-*components* **by** *simp*

lemma *ntsmcf*-0-*NTMap-vdomain*[*smc-cs-simps*]: $\mathcal{D}_{\circ}$ (*ntsmcf*-0 $\mathfrak{C}$(|*NTMap*|)) = 0
unfolding *ntsmcf*-0-*components* **by** *simp*

lemma *ntsmcf*-0-*NTMap-vrange*[*smc-cs-simps*]: $\mathcal{R}_{\circ}$ (*ntsmcf*-0 $\mathfrak{C}$(|*NTMap*|)) = 0
unfolding *ntsmcf*-0-*components* **by** *simp*

Empty natural transformation of semifunctors is a natural transformation of semifunctors

lemma (**in** *semicategory*) *smc-ntsmcf*-0-*is-ntsmcfI*:
ntsmcf-0 $\mathfrak{C}$: *smcf*-0 $\mathfrak{C}$ $\mapsto_{SMCF}$ *smcf*-0 $\mathfrak{C}$: *smc*-0 $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{C}$
proof(*intro is-ntsmcfI*)
show *vfsequence* (*ntsmcf*-0 $\mathfrak{C}$) **unfolding** *ntsmcf*-0-*def* **by** *simp*
show *vcard* (*ntsmcf*-0 $\mathfrak{C}$) = $\mathfrak{5}_{\mathbb{N}}$
unfolding *ntsmcf*-0-*def* **by** (*simp add*: *nat-omega-simps*)
show *ntsmcf-tdghm* (*ntsmcf*-0 $\mathfrak{C}$) :
smcf-dghm (*smcf*-0 $\mathfrak{C}$) $\mapsto_{DGHM}$ *smcf-dghm* (*smcf*-0 $\mathfrak{C}$) :
smc-dg smc-0 $\mapsto\!\!\mapsto_{DG\alpha}$ *smc-dg* $\mathfrak{C}$
unfolding *ntsmcf-tdghm-ntsmcf*-0 *smcf-dghm-smcf*-0 *smc-dg-smc*-0
by (*cs-concl* **cs-shallow cs-intro**: *dg-cs-intros slicing-intros*)
show
ntsmcf-0 $\mathfrak{C}$(|*NTMap*|)(|*b*|) $\circ_{A\mathfrak{C}}$ *smcf*-0 $\mathfrak{C}$(|*ArrMap*|)(|*f*|) =
smcf-0 $\mathfrak{C}$(|*ArrMap*|)(|*f*|) $\circ_{A\mathfrak{C}}$ *ntsmcf*-0 $\mathfrak{C}$(|*NTMap*|)(|*a*|)
if *f* : *a* $\mapsto_{smc\text{-}0}$ *b* **for** *a b f*
using *that* **by** (*elim is-arrE*) (*auto simp*: *smc*-0-*components*)
qed
(
cs-concl **cs-shallow**
cs-simp: *smc-cs-simps smc*-0-*components*(1) **cs-intro**: *smc-cs-intros*
)+

lemma (**in** *semicategory*) *smc-ntsmcf*-0-*is-ntsmcfI'*[*smc-cs-intros*]:
assumes $\mathfrak{F}'$ = *smcf*-0 $\mathfrak{C}$
and $\mathfrak{G}'$ = *smcf*-0 $\mathfrak{C}$
and $\mathfrak{A}'$ = *smc*-0
and $\mathfrak{B}'$ = $\mathfrak{C}$
and $\mathfrak{F}'$ = $\mathfrak{F}$
and $\mathfrak{G}'$ = $\mathfrak{G}$
shows *ntsmcf*-0 $\mathfrak{C}$: $\mathfrak{F}'$ $\mapsto_{SMCF}$ $\mathfrak{G}'$: $\mathfrak{A}'$ $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{B}'$
unfolding *assms* **by** (*rule smc-ntsmcf*-0-*is-ntsmcfI*)

lemmas [*smc-cs-intros*] = *semicategory.smc-ntsmcf*-0-*is-ntsmcfI'*

lemma *is-ntsmcf-is-ntsmcf*-0-*if-smc*-0:
assumes $\mathfrak{N}$: $\mathfrak{F}$ $\mapsto_{SMCF}$ $\mathfrak{G}$: *smc*-0 $\mapsto\!\!\mapsto_{SMC\alpha}$ $\mathfrak{C}$
shows $\mathfrak{N}$ = *ntsmcf*-0 $\mathfrak{C}$ **and** $\mathfrak{F}$ = *smcf*-0 $\mathfrak{C}$ **and** $\mathfrak{G}$ = *smcf*-0 $\mathfrak{C}$
proof–
interpret $\mathfrak{N}$: *is-ntsmcf* α *smc*-0 $\mathfrak{C}$ $\mathfrak{F}$ $\mathfrak{G}$ $\mathfrak{N}$ **by** (*rule assms*(1))
note *is-tdghm-is-tdghm*-0-*if-dg*-0 = *is-tdghm-is-tdghm*-0-*if-dg*-0
[
OF $\mathfrak{N}$.*ntsmcf-is-tdghm*[*unfolded smc-dg-smc*-0],
folded smcf-dghm-smcf-0 *ntsmcf-tdghm-ntsmcf*-0
]
show $\mathfrak{F}$-*def*: $\mathfrak{F}$ = *smcf*-0 $\mathfrak{C}$ **and** $\mathfrak{G}$-*def*: $\mathfrak{G}$ = *smcf*-0 $\mathfrak{C}$
by (*all*‹*intro is-semifunctor-is-smcf*-0-*if-smc*-0›)
(*cs-concl* **cs-shallow cs-intro**: *smc-cs-intros*)+

```
  show 𝔑 = ntsmcf-0 𝔈
  proof(rule ntsmcf-tdghm-eqI)
    show 𝔑 : 𝔉 ↦_SMCF 𝔊 : smc-0 ↦↦_SMCα 𝔈 by (rule assms(1))
    show ntsmcf-0 𝔈 : 𝔉 ↦_SMCF 𝔊 : smc-0 ↦↦_SMCα 𝔈
      by (cs-concl cs-simp: 𝔉-def 𝔊-def cs-intro: smc-cs-intros)
  qed (simp-all add: 𝔉-def 𝔊-def is-tdghm-is-tdghm-0-if-dg-0)
qed
```

Further properties

```
lemma ntsmcf-vcomp-ntsmcf-ntsmcf-0[smc-cs-simps]:
  assumes 𝔑 : 𝔉 ↦_SMCF 𝔊 : smc-0 ↦↦_SMCα 𝔈
  shows 𝔑 ∘_NTSMCF ntsmcf-0 𝔈 = ntsmcf-0 𝔈
proof–
  interpret 𝔑: is-ntsmcf α smc-0 𝔈 𝔉 𝔊 𝔑 by (rule assms(1))
  show ?thesis
    unfolding is-ntsmcf-is-ntsmcf-0-if-smc-0[OF assms]
  proof(rule ntsmcf-eqI)
    show ntsmcf-0 𝔈 ∘_NTSMCF ntsmcf-0 𝔈 :
      smcf-0 𝔈 ↦_SMCF smcf-0 𝔈 : smc-0 ↦↦_SMCα 𝔈
      by (cs-concl cs-intro: smc-cs-intros)
    then have dom-lhs: 𝒟∘ ((ntsmcf-0 𝔈 ∘_NTSMCF ntsmcf-0 𝔈)(⌊NTMap⌋)) = 0
      by
        (
          cs-concl
            cs-simp: smc-cs-simps smc-0-components cs-intro: smc-cs-intros
        )
    show ntsmcf-0 𝔈 : smcf-0 𝔈 ↦_SMCF smcf-0 𝔈 : smc-0 ↦↦_SMCα 𝔈
      by (cs-concl cs-intro: smc-cs-intros)
    then have dom-rhs: 𝒟∘ (ntsmcf-0 𝔈(⌊NTMap⌋)) = 0
      by
        (
          cs-concl
            cs-simp: smc-cs-simps smc-0-components cs-intro: smc-cs-intros
        )
    show (ntsmcf-0 𝔈 ∘_NTSMCF ntsmcf-0 𝔈)(⌊NTMap⌋) = ntsmcf-0 𝔈(⌊NTMap⌋)
      by (rule vsv-eqI, unfold dom-lhs dom-rhs) (auto intro: smc-cs-intros)
  qed simp-all
qed

lemma ntsmcf-vcomp-ntsmcf-0-ntsmcf[smc-cs-simps]:
  assumes 𝔑 : 𝔉 ↦_SMCF 𝔊 : smc-0 ↦↦_SMCα 𝔈
  shows ntsmcf-0 𝔈 ∘_NTSMCF 𝔑 = ntsmcf-0 𝔈
proof–
  interpret 𝔑: is-ntsmcf α smc-0 𝔈 𝔉 𝔊 𝔑 by (rule assms(1))
  show ?thesis
    unfolding is-ntsmcf-is-ntsmcf-0-if-smc-0[OF assms]
    by (cs-concl cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
qed
```

4.10.5 10: semicategory with one object and no arrows

Definition and elementary properties

```
definition smc-10 :: V ⇒ V
  where smc-10 𝔞 = [set {𝔞}, 0, 0, 0, 0]∘
```

Components.

```
lemma smc-10-components:
```

shows *smc*-10 𝔞(|*Obj*|) = *set* {𝔞}
and *smc*-10 𝔞(|*Arr*|) = 0
and *smc*-10 𝔞(|*Dom*|) = 0
and *smc*-10 𝔞(|*Cod*|) = 0
and *smc*-10 𝔞(|*Comp*|) = 0
unfolding *smc*-10-*def dg-field-simps* **by** (*auto simp*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-smc*-10: *smc-dg* (*smc*-10 𝔞) = (*dg*-10 𝔞)
unfolding *smc-dg-def smc*-10-*def dg*-10-*def dg-field-simps*
by (*simp add*: *nat-omega-simps*)

lemmas-with (**in** 𝒵) [*folded smc-dg-smc*-10, *unfolded slicing-simps*]:
smc-10-*is-arr-iff* = *dg*-10-*is-arr-iff*

10 is a semicategory

lemma (**in** 𝒵) *semicategory-smc*-10:
assumes 𝔞 ∈∘ *Vset* α
shows *semicategory* α (*smc*-10 𝔞)
proof(*intro semicategoryI*)
show *vfsequence* (*smc*-10 𝔞)
unfolding *smc*-10-*def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* (*smc*-10 𝔞) = $5_{\mathbf{N}}$
unfolding *smc*-10-*def* **by** (*simp add*: *nat-omega-simps*)
show *digraph* α (*smc-dg* (*smc*-10 𝔞))
unfolding *smc-dg-smc*-10 **by** (*rule digraph-dg*-10[*OF assms*])
qed (*auto simp*: *smc*-10-*components smc*-10-*is-arr-iff vsubset-vsingleton-leftI*)

Arrow with a domain and a codomain

lemma *smc*-10-*is-arr-iff*: 𝔉 : 𝔄 ↦$_{smc\text{-}10\ 𝔞}$ 𝔅 ⟷ *False*
unfolding *is-arr-def smc*-10-*components* **by** *simp*

4.10.6 1: semicategory with one object and one arrow

Definition and elementary properties

definition *smc*-1 :: *V* ⇒ *V* ⇒ *V*
where *smc*-1 𝔞 𝔣 =
[*set* {𝔞}, *set* {𝔣}, *set* {⟨𝔣, 𝔞⟩}, *set* {⟨𝔣, 𝔞⟩}, *set* {⟨⟨[𝔣, 𝔣]∘, 𝔣⟩⟩}]∘

Components.

lemma *smc*-1-*components*:
shows *smc*-1 𝔞 𝔣(|*Obj*|) = *set* {𝔞}
and *smc*-1 𝔞 𝔣(|*Arr*|) = *set* {𝔣}
and *smc*-1 𝔞 𝔣(|*Dom*|) = *set* {⟨𝔣, 𝔞⟩}
and *smc*-1 𝔞 𝔣(|*Cod*|) = *set* {⟨𝔣, 𝔞⟩}
and *smc*-1 𝔞 𝔣(|*Comp*|) = *set* {⟨⟨[𝔣, 𝔣]∘, 𝔣⟩⟩}
unfolding *smc*-1-*def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *dg-smc*-1: *smc-dg* (*smc*-1 𝔞 𝔣) = *dg*-1 𝔞 𝔣
unfolding *smc-dg-def smc*-1-*def dg*-1-*def dg-field-simps*
by (*simp add*: *nat-omega-simps*)

lemmas-with [*folded dg-smc*-1, *unfolded slicing-simps*]:
smc-1-*is-arrI* = *dg*-1-*is-arrI*

and *smc*-1-*is-arrD* = *dg*-1-*is-arrD*
and *smc*-1-*is-arrE* = *dg*-1-*is-arrE*
and *smc*-1-*is-arr-iff* = *dg*-1-*is-arr-iff*

Composition

lemma *smc*-1-*Comp-app*[*simp*]: $\mathfrak{f} \circ_{A\,smc\text{-}1}\ \mathfrak{a}\ \mathfrak{f}\ \mathfrak{f} = \mathfrak{f}$
unfolding *smc*-1-*components* **by** *simp*

1 is a semicategory

lemma (**in** $\mathcal{Z}$) *semicategory-smc*-1:
assumes $\mathfrak{a} \in_{\circ}$ *Vset* α **and** $\mathfrak{f} \in_{\circ}$ *Vset* α
shows *semicategory* α (*smc*-1 $\mathfrak{a}$ $\mathfrak{f}$)
proof(*intro semicategoryI*, *unfold dg-smc*-1)
show *vfsequence* (*smc*-1 $\mathfrak{a}$ $\mathfrak{f}$)
unfolding *smc*-1-*def* **by** (*simp add*: *nat-omega-simps*)
show *vcard* (*smc*-1 $\mathfrak{a}$ $\mathfrak{f}$) = $5_{\mathbb{N}}$
unfolding *smc*-1-*def* **by** (*simp add*: *nat-omega-simps*)
qed
(
auto simp:
assms
digraph-dg-1
smc-1-*is-arr-iff*
smc-1-*components*
vsubset-vsingleton-leftI
)

4.11 *Rel* as a semicategory

4.11.1 Background

The methodology chosen for the exposition of *Rel* as a semicategory is analogous to the one used in the previous chapter for the exposition of *Rel* as a digraph. The general references for this section are Chapter I-7 in [39] and nLab [3][5].

named-theorems *smc-Rel-cs-simps*
named-theorems *smc-Rel-cs-intros*

lemmas (**in** *arr-Rel*) [*smc-Rel-cs-simps*] =
dg-Rel-shared-cs-simps

lemmas (**in** *arr-Rel*) [*smc-cs-intros*, *smc-Rel-cs-intros*] =
arr-Rel-axioms′

lemmas [*smc-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-v*11
*arr-Rel-comp-Rel-converse-Rel-right-if-v*11

lemmas [*smc-Rel-cs-intros*] =
dg-Rel-shared-cs-intros
arr-Rel-comp-Rel
arr-Rel.arr-Rel-converse-Rel

4.11.2 *Rel* as a semicategory

Definition and elementary properties

definition *smc-Rel* :: $V \Rightarrow V$
where *smc-Rel* α =
[
Vset α,
set $\{T.\ \textit{arr-Rel}\ \alpha\ T\}$,
$(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$,
$(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$,
$(\lambda ST \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Rel}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$
]$_{\circ}$

Components.

lemma *smc-Rel-components*:
shows *smc-Rel* $\alpha(\!|\textit{Obj}|\!)$ = *Vset* α
and *smc-Rel* $\alpha(\!|\textit{Arr}|\!)$ = *set* $\{T.\ \textit{arr-Rel}\ \alpha\ T\}$
and *smc-Rel* $\alpha(\!|\textit{Dom}|\!)$ = $(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$
and *smc-Rel* $\alpha(\!|\textit{Cod}|\!)$ = $(\lambda T \in_{\circ} \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$
and *smc-Rel* $\alpha(\!|\textit{Comp}|\!)$ = $(\lambda ST \in_{\circ} \textit{composable-arrs}\ (\textit{dg-Rel}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$
unfolding *smc-Rel-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

[5] https://ncatlab.org/nlab/show/Rel

lemma *smc-dg-smc-Rel*: *smc-dg* (*smc-Rel* α) = *dg-Rel* α
proof(*rule vsv-eqI*)
show *vsv* (*smc-dg* (*smc-Rel* α)) **unfolding** *smc-dg-def* **by** *auto*
show *vsv* (*dg-Rel* α) **unfolding** *dg-Rel-def* **by** *auto*
have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-Rel* α)) = $4_{\mathbb{N}}$
unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-Rel* α) = $4_{\mathbb{N}}$
unfolding *dg-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*smc-dg* (*smc-Rel* α)) = $\mathcal{D}_\circ$ (*dg-Rel* α)
unfolding *dom-lhs dom-rhs* **by** *simp*
show *a* $\in_\circ$ $\mathcal{D}_\circ$ (*smc-dg* (*smc-Rel* α)) $\Longrightarrow$ *smc-dg* (*smc-Rel* α)($\lfloor a \rfloor$) = *dg-Rel* α($\lfloor a \rfloor$)
for *a*
by
(
unfold dom-lhs,
elim-in-numeral,
unfold smc-dg-def dg-field-simps smc-Rel-def dg-Rel-def
)
(*auto simp*: *nat-omega-simps*)
qed

lemmas-with [*folded smc-dg-smc-Rel*, *unfolded slicing-simps*]:
smc-Rel-Obj-iff = *dg-Rel-Obj-iff*
and *smc-Rel-Arr-iff*[*smc-Rel-cs-simps*] = *dg-Rel-Arr-iff*
and *smc-Rel-Dom-vsv*[*smc-Rel-cs-intros*] = *dg-Rel-Dom-vsv*
and *smc-Rel-Dom-vdomain*[*smc-Rel-cs-simps*] = *dg-Rel-Dom-vdomain*
and *smc-Rel-Dom-app*[*smc-Rel-cs-simps*] = *dg-Rel-Dom-app*
and *smc-Rel-Dom-vrange* = *dg-Rel-Dom-vrange*
and *smc-Rel-Cod-vsv*[*smc-Rel-cs-intros*] = *dg-Rel-Cod-vsv*
and *smc-Rel-Cod-vdomain*[*smc-Rel-cs-simps*] = *dg-Rel-Cod-vdomain*
and *smc-Rel-Cod-app*[*smc-Rel-cs-simps*] = *dg-Rel-Cod-app*
and *smc-Rel-Cod-vrange* = *dg-Rel-Cod-vrange*
and *smc-Rel-is-arrI*[*smc-Rel-cs-intros*] = *dg-Rel-is-arrI*
and *smc-Rel-is-arrD* = *dg-Rel-is-arrD*
and *smc-Rel-is-arrE* = *dg-Rel-is-arrE*
and *smc-Rel-is-arr-ArrValE* = *dg-Rel-is-arr-ArrValE*

lemmas [*smc-cs-simps*] = *smc-Rel-is-arrD*(2,3)

lemmas-with (**in** $\mathcal{Z}$) [*folded smc-dg-smc-Rel*, *unfolded slicing-simps*]:
smc-Rel-Hom-vifunion-in-Vset = *dg-Rel-Hom-vifunion-in-Vset*
and *smc-Rel-incl-Rel-is-arr* = *dg-Rel-incl-Rel-is-arr*
and *smc-Rel-incl-Rel-is-arr'*[*smc-Rel-cs-intros*] = *dg-Rel-incl-Rel-is-arr'*

lemmas [*smc-Rel-cs-intros*] = $\mathcal{Z}$.*smc-Rel-incl-Rel-is-arr'*

Composable arrows

lemma *smc-Rel-composable-arrs-dg-Rel*:
composable-arrs (*dg-Rel* α) = *composable-arrs* (*smc-Rel* α)
unfolding *composable-arrs-def smc-dg-smc-Rel*[*symmetric*] *slicing-simps* **by** *simp*

lemma *smc-Rel-Comp*:
smc-Rel α($\lfloor$*Comp*$\rfloor$) = ($\lambda ST \in_\circ$ *composable-arrs* (*smc-Rel* α). $ST(\lfloor 0 \rfloor) \circ_{Rel} ST(\lfloor 1_{\mathbb{N}} \rfloor)$)
unfolding *smc-Rel-components smc-Rel-composable-arrs-dg-Rel* **..**

Composition

lemma *smc-Rel-Comp-app*[*smc-Rel-cs-simps*]:
assumes $S : b \mapsto_{\textit{smc-Rel}\ \alpha}\ c$ **and** $T : a \mapsto_{\textit{smc-Rel}\ \alpha}\ b$
shows $S \circ_{A\ \textit{smc-Rel}\ \alpha}\ T = S \circ_{Rel}\ T$
proof–
from *assms* **have** $[S,\ T]_\circ \in_\circ \textit{composable-arrs}\ (\textit{smc-Rel}\ \alpha)$
by (*auto intro*: *smc-cs-intros*)
then show $S \circ_{A\ \textit{smc-Rel}\ \alpha}\ T = S \circ_{Rel}\ T$
unfolding *smc-Rel-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-Rel-Comp-vdomain*: $\mathcal{D}_\circ\ (\textit{smc-Rel}\ \alpha(\!|\textit{Comp}|\!)) = \textit{composable-arrs}\ (\textit{smc-Rel}\ \alpha)$
unfolding *smc-Rel-Comp* **by** *simp*

lemma (**in** $\mathcal{Z}$) *smc-Rel-Comp-vrange*: $\mathcal{R}_\circ\ (\textit{smc-Rel}\ \alpha(\!|\textit{Comp}|\!)) \subseteq_\circ \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}$
proof(*rule vsubsetI*)
interpret *digraph* α ‹*smc-dg* (*smc-Rel* α)›
unfolding *smc-dg-smc-Rel* **by** (*simp add*: *digraph-dg-Rel*)
fix R **assume** $R \in_\circ \mathcal{R}_\circ\ (\textit{smc-Rel}\ \alpha(\!|\textit{Comp}|\!))$
then obtain ST
where *R-def*: $R = \textit{smc-Rel}\ \alpha(\!|\textit{Comp}|\!)(\!|ST|\!)$
and $ST \in_\circ \mathcal{D}_\circ\ (\textit{smc-Rel}\ \alpha(\!|\textit{Comp}|\!))$
unfolding *smc-Rel-components* **by** (*auto intro*: *smc-cs-intros*)
then obtain $S\ T\ a\ b\ c$
where $ST = [S,\ T]_\circ$
and *S*: $S : b \mapsto_{\textit{smc-Rel}\ \alpha}\ c$
and *T*: $T : a \mapsto_{\textit{smc-Rel}\ \alpha}\ b$
by (*auto simp*: *smc-Rel-Comp-vdomain*)
with *R-def* **have** *R-def'*: $R = S \circ_{A\ \textit{smc-Rel}\ \alpha}\ T$ **by** *simp*
note *S-D* = *dg-is-arrD*(1)[*unfolded slicing-simps*, *OF S*]
note *T-D* = *dg-is-arrD*(1)[*unfolded slicing-simps*, *OF T*]
from *S-D T-D* **have** *arr-Rel* α *S arr-Rel* α *T*
by (*simp-all add*: *smc-Rel-components*)
from *this* **show** $R \in_\circ \textit{set}\ \{T.\ \textit{arr-Rel}\ \alpha\ T\}$
unfolding *R-def' smc-Rel-Comp-app*[*OF S T*] **by** (*auto simp*: *arr-Rel-comp-Rel*)
qed

Rel is a semicategory

lemma (**in** $\mathcal{Z}$) *semicategory-smc-Rel*: *semicategory* α (*smc-Rel* α)
proof(*rule semicategoryI*, *unfold smc-dg-smc-Rel*)
show *vfsequence* (*smc-Rel* α) **unfolding** *smc-Rel-def* **by** *simp*
show *vcard* (*smc-Rel* α) = $5_{\mathbb{N}}$
unfolding *smc-Rel-def* **by** (*simp add*: *nat-omega-simps*)
show $gf \in_\circ \mathcal{D}_\circ\ (\textit{smc-Rel}\ \alpha(\!|\textit{Comp}|\!)) \longleftrightarrow$
$(\exists g\ f\ b\ c\ a.\ gf = [g,\ f]_\circ \wedge g : b \mapsto_{\textit{smc-Rel}\ \alpha}\ c \wedge f : a \mapsto_{\textit{smc-Rel}\ \alpha}\ b)$
for *gf*
unfolding *smc-Rel-Comp-vdomain* **by** (*auto intro*: *composable-arrsI*)
show $g \circ_{A\ \textit{smc-Rel}\ \alpha}\ f : a \mapsto_{\textit{smc-Rel}\ \alpha}\ c$
if $g : b \mapsto_{\textit{smc-Rel}\ \alpha}\ c$ **and** $f : a \mapsto_{\textit{smc-Rel}\ \alpha}\ b$ **for** $g\ b\ c\ f\ a$
proof–
from *that* **have** *arr-Rel* α *g* **and** *arr-Rel* α *f*
by (*auto simp*: *smc-Rel-is-arrD*(1))
with *that* **show** *?thesis*
by
(
cs-concl **cs-shallow**
cs-simp: *smc-cs-simps smc-Rel-cs-simps* **cs-intro**: *smc-Rel-cs-intros*

```
      )
  qed
  show h ∘_A smc-Rel α g ∘_A smc-Rel α f = h ∘_A smc-Rel α (g ∘_A smc-Rel α f)
    if h : c ↦_smc-Rel α d
      and g : b ↦_smc-Rel α c
      and f : a ↦_smc-Rel α b
    for h c d g b f a
  proof–
    from that have arr-Rel α h and arr-Rel α g and arr-Rel α f
      by (auto simp: smc-Rel-is-arrD(1))
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: smc-cs-simps smc-Rel-cs-simps
            cs-intro: smc-Rel-cs-intros
        )
  qed
qed (auto simp: digraph-dg-Rel smc-Rel-components)
```

4.11.3 Canonical dagger for *Rel*

Definition and elementary properties

```
definition smcf-dag-Rel :: V ⇒ V (‹†_SMC.Rel›)
  where †_SMC.Rel α =
    [
      vid-on (smc-Rel α(|Obj|)),
      VLambda (smc-Rel α(|Arr|)) converse-Rel,
      op-smc (smc-Rel α),
      smc-Rel α
    ]∘
```

Components.

```
lemma smcf-dag-Rel-components:
  shows †_SMC.Rel α(|ObjMap|) = vid-on (smc-Rel α(|Obj|))
    and †_SMC.Rel α(|ArrMap|) = VLambda (smc-Rel α(|Arr|)) converse-Rel
    and †_SMC.Rel α(|HomDom|) = op-smc (smc-Rel α)
    and †_SMC.Rel α(|HomCod|) = smc-Rel α
  unfolding smcf-dag-Rel-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

Slicing.

```
lemma smcf-dghm-smcf-dag-Rel: smcf-dghm (†_SMC.Rel α) = †_DG.Rel α
proof(rule vsv-eqI)
  show vsv (smcf-dghm (†_SMC.Rel α)) unfolding smcf-dghm-def by auto
  show vsv (†_DG.Rel α) unfolding dghm-dag-Rel-def by auto
  have dom-lhs: 𝒟∘ (smcf-dghm (†_SMC.Rel α)) = 4_ℕ
    unfolding smcf-dghm-def by (simp add: nat-omega-simps)
  have dom-rhs: 𝒟∘ (†_DG.Rel α) = 4_ℕ
    unfolding dghm-dag-Rel-def by (simp add: nat-omega-simps)
  show 𝒟∘ (smcf-dghm (†_SMC.Rel α)) = 𝒟∘ (†_DG.Rel α)
    unfolding dom-lhs dom-rhs by simp
  show a ∈∘ 𝒟∘ (smcf-dghm (†_SMC.Rel α)) ⟹
    smcf-dghm (†_SMC.Rel α)(|a|) = †_DG.Rel α(|a|)
    for a
    by
      (
        unfold dom-lhs,
```

```
    elim-in-numeral,
    unfold dghm-field-simps[symmetric],
    unfold
      smc-dg-smc-Rel
      slicing-commute[symmetric]
      smcf-dghm-components
      dghm-dag-Rel-components
      smcf-dag-Rel-components
      dg-Rel-components
      smc-Rel-components
  )
  simp-all
qed

lemmas-with [
  folded smc-dg-smc-Rel smcf-dghm-smcf-dag-Rel, unfolded slicing-simps
  ]:
 smcf-dag-Rel-ObjMap-vsv[smc-Rel-cs-intros] = dghm-dag-Rel-ObjMap-vsv
 and smcf-dag-Rel-ObjMap-vdomain[smc-Rel-cs-simps] =
  dghm-dag-Rel-ObjMap-vdomain
 and smcf-dag-Rel-ObjMap-app[smc-Rel-cs-simps] = dghm-dag-Rel-ObjMap-app
 and smcf-dag-Rel-ObjMap-vrange[smc-Rel-cs-simps] = dghm-dag-Rel-ObjMap-vrange
 and smcf-dag-Rel-ArrMap-vsv[smc-Rel-cs-intros] = dghm-dag-Rel-ArrMap-vsv
 and smcf-dag-Rel-ArrMap-vdomain[smc-Rel-cs-simps] =
  dghm-dag-Rel-ArrMap-vdomain
 and smcf-dag-Rel-ArrMap-app[smc-Rel-cs-simps] = dghm-dag-Rel-ArrMap-app
 and smcf-dag-Rel-ArrMap-app-vdomain[smc-cs-simps] =
  dghm-dag-Rel-ArrMap-app-vdomain
 and smcf-dag-Rel-ArrMap-app-vrange[smc-cs-simps] =
  dghm-dag-Rel-ArrMap-app-vrange
 and smcf-dag-Rel-ArrMap-vrange[smc-Rel-cs-simps] = dghm-dag-Rel-ArrMap-vrange
 and smcf-dag-Rel-ArrMap-app-iff[smc-cs-simps] = dghm-dag-Rel-ArrMap-app-iff
 and smcf-dag-Rel-app-is-arr = dghm-dag-Rel-ArrMap-app-is-arr
```

Canonical dagger is a contravariant isomorphism of *Rel*

```
lemma (in Z) smcf-dag-Rel-is-iso-semifunctor:
 †_{SMC.Rel} α : op-smc (smc-Rel α) ↦↦_{SMC.iso α} smc-Rel α
proof(rule is-iso-semifunctorI)
 interpret dag: is-iso-dghm α ‹op-dg (dg-Rel α)› ‹dg-Rel α› ‹†_{DG.Rel} α›
  by (rule dghm-dag-Rel-is-iso-dghm)
 interpret Rel: semicategory α ‹smc-Rel α›
  by (rule semicategory-smc-Rel)
 show †_{SMC.Rel} α : op-smc (smc-Rel α) ↦↦_{SMC α} smc-Rel α
 proof
  (
    rule is-semifunctorI,
    unfold
      smc-dg-smc-Rel
      smcf-dghm-smcf-dag-Rel
      smc-op-simps
      slicing-commute[symmetric]
      smcf-dag-Rel-components(3,4)
  )
  show vfsequence (†_{SMC.Rel} α)
    unfolding smcf-dag-Rel-def by (simp add: nat-omega-simps)
  show vcard (†_{SMC.Rel} α) = 4_N
    unfolding smcf-dag-Rel-def by (simp add: nat-omega-simps)
```

```
    show †SMC.Rel α(|ArrMap|)(|f ∘A smc-Rel α g|) =
      †SMC.Rel α(|ArrMap|)(|g|) ∘A smc-Rel α †SMC.Rel α(|ArrMap|)(|f|)
      if g : c ↦smc-Rel α b and f : b ↦smc-Rel α a
      for g b c f a
    proof–
      from that have arr-Rel α g and arr-Rel α f
        by (auto simp: smc-Rel-is-arrD(1))
      with that show ?thesis
        by
          (
            cs-concl cs-shallow
              cs-simp: smc-cs-simps smc-Rel-cs-simps
              cs-intro: smc-Rel-cs-intros
          )
    qed
  qed (auto simp: dg-cs-intros smc-op-intros semicategory-smc-Rel)

  show smcf-dghm (†SMC.Rel α) :
    smc-dg (op-smc (smc-Rel α)) ↦↦DG.isoα smc-dg (smc-Rel α)
    by
      (
        simp add:
          smc-dg-smc-Rel
          smcf-dghm-smcf-dag-Rel
          smc-op-simps
          slicing-simps
          slicing-commute[symmetric]
          dghm-dag-Rel-is-iso-dghm
      )

qed
```

Further properties of the canonical dagger

```
lemma (in 𝒵) smcf-cn-comp-smcf-dag-Rel-smcf-dag-Rel:
  †SMC.Rel α SMCF∘ †SMC.Rel α = smcf-id (smc-Rel α)
proof(rule smcf-dghm-eqI)
  interpret semicategory α ‹smc-Rel α› by (simp add: semicategory-smc-Rel)
  from smcf-dag-Rel-is-iso-semifunctor have dag:
    †SMC.Rel α : op-smc (smc-Rel α) ↦↦SMCα smc-Rel α
    by (simp add: is-iso-semifunctor.axioms(1))
  from smcf-cn-comp-is-semifunctor[OF semicategory-axioms dag dag] show
    †SMC.Rel α SMCF∘ †SMC.Rel α : smc-Rel α ↦↦SMCα smc-Rel α .
  show smcf-id (smc-Rel α) : smc-Rel α ↦↦SMCα smc-Rel α
    by (auto simp: smc-smcf-id-is-semifunctor)
  show smcf-dghm (†SMC.Rel α SMCF∘ †SMC.Rel α) = smcf-dghm (smcf-id (smc-Rel α))
    unfolding
      slicing-simps slicing-commute[symmetric]
      smc-dg-smc-Rel
      smcf-dghm-smcf-dag-Rel
    by (simp add: dghm-cn-comp-dghm-dag-Rel-dghm-dag-Rel)
qed simp-all

lemma smcf-dag-Rel-ArrMap-smc-Rel-Comp:
  assumes S : b ↦smc-Rel α c and T : a ↦smc-Rel α b
  shows †SMC.Rel α(|ArrMap|)(|S ∘A smc-Rel α T|) =
    †SMC.Rel α(|ArrMap|)(|T|) ∘A smc-Rel α †SMC.Rel α(|ArrMap|)(|S|)
proof–
```

```
  from assms have arr-Rel α S and arr-Rel α T
    by (auto simp: smc-Rel-is-arrD(1))
  with assms show ?thesis
    by
      (
        cs-concl cs-shallow
          cs-simp: smc-cs-simps smc-Rel-cs-simps cs-intro: smc-Rel-cs-intros
      )
qed
```

4.11.4 Monic arrow and epic arrow

The conditions for an arrow of *Rel* to be either monic or epic are outlined in nLab [3][6].

```
context
begin

private lemma smc-Rel-is-monic-arr-vsubset:
  assumes T : A ↦_{smc-Rel α} B
    and R : A' ↦_{smc-Rel α} A
    and S : A' ↦_{smc-Rel α} A
    and T ∘_{A smc-Rel α} R = T ∘_{A smc-Rel α} S
    and ⋀y z X.
      ⟦ y ⊆∘ A; z ⊆∘ A; T⟨ArrVal⟩ ‘∘ y = X; T⟨ArrVal⟩ ‘∘ z = X ⟧ ⟹ y = z
  shows R⟨ArrVal⟩ ⊆∘ S⟨ArrVal⟩
proof–
  interpret R: arr-Rel α R
    rewrites R⟨ArrDom⟩ = A' and R⟨ArrCod⟩ = A
    by (intro smc-Rel-is-arrD[OF assms(2)])+
  interpret S: arr-Rel α S
    rewrites S⟨ArrDom⟩ = A' and S⟨ArrCod⟩ = A
    by (intro smc-Rel-is-arrD[OF assms(3)])+
  interpret Rel: semicategory α ‹smc-Rel α› by (rule R.semicategory-smc-Rel)
  from assms(4) have (T ∘_{A smc-Rel α} R)⟨ArrVal⟩ = (T ∘_{A smc-Rel α} S)⟨ArrVal⟩
    by simp
  then have eq: T⟨ArrVal⟩ ∘∘ R⟨ArrVal⟩ = T⟨ArrVal⟩ ∘∘ S⟨ArrVal⟩
    unfolding
      smc-Rel-Comp-app[OF assms(1,2)]
      smc-Rel-Comp-app[OF assms(1,3)]
      comp-Rel-components
    by simp
  show R⟨ArrVal⟩ ⊆∘ S⟨ArrVal⟩
  proof(rule vsubsetI)
    fix ab assume ab[intro]: ab ∈∘ R⟨ArrVal⟩
    with R.ArrVal.vbrelation obtain a b where ab-def: ab = ⟨a, b⟩ by auto
    with ab R.arr-Rel-ArrVal-vrange have a ∈∘ 𝒟∘ (R⟨ArrVal⟩) and b ∈∘ A
      by auto
    define B' and C' where B' = R⟨ArrVal⟩ ‘∘ set {a} and C' = T⟨ArrVal⟩ ‘∘ B'
    have ne-C': C' ≠ 0
    proof(rule ccontr, unfold not-not)
      assume prems: C' = 0
      from ab have b ∈∘ B' unfolding ab-def B'-def by simp
      with C'-def[unfolded prems] have b0: T⟨ArrVal⟩ ‘∘ set {b} = 0 by auto
      from assms(5)[OF - - b0, of 0] ‹b ∈∘ A› show False by auto
    qed
    have cac''[intro, simp]:
      c ∈∘ C' ⟹ ⟨a, c⟩ ∈∘ T⟨ArrVal⟩ ∘∘ S⟨ArrVal⟩ for c
```

[6] https://ncatlab.org/nlab/show/Rel

```
      unfolding eq[symmetric] C'-def B'-def
      by (metis vcomp-vimage vimage-vsingleton-iff)
    define A'' where A'' = (T(|ArrVal|) ∘ₒ S(|ArrVal|)) -`ₒ C'
    define B'' where B'' = S(|ArrVal|) `ₒ set {a}
    define C'' where C'' = T(|ArrVal|) `ₒ B''
    have a'': a ∈ₒ A''
    proof–
      from ne-C' obtain c' where [intro]: c' ∈ₒ C'
        by (auto intro!: vsubset-antisym)
      then have ⟨a, c'⟩ ∈ₒ T(|ArrVal|) ∘ₒ S(|ArrVal|) by simp
      then show ?thesis unfolding A''-def by auto
    qed
    have C' ⊆ₒ C''
      unfolding C''-def B''-def A''-def C'-def B'-def
      by (rule vsubsetI) (metis eq vcomp-vimage)
    have C' = C''
    proof(rule ccontr)
      assume C' ≠ C''
      with ‹C' ⊆ₒ C''› obtain c' where c': c' ∈ₒ C'' -ₒ C'
        by (auto intro!: vsubset-antisym)
      then obtain b'' where b'' ∈ₒ B'' and ⟨b'', c'⟩ ∈ₒ T(|ArrVal|)
        unfolding C''-def by auto
      then have ⟨a, c'⟩ ∈ₒ T(|ArrVal|) ∘ₒ R(|ArrVal|) unfolding eq B''-def by auto
      with c' show False unfolding B'-def C'-def by auto
    qed
    then have T(|ArrVal|) `ₒ B'' = T(|ArrVal|) `ₒ B' by (simp add: C''-def C'-def)
    moreover have B' ⊆ₒ A and B'' ⊆ₒ A
      using R.arr-Rel-ArrVal-vrange S.arr-Rel-ArrVal-vrange
      unfolding B'-def B''-def
      by auto
    ultimately have B'' = B' by (simp add: assms(5))
    with ab have b ∈ₒ B'' unfolding B'-def ab-def by simp
    then show ab ∈ₒ S(|ArrVal|) unfolding ab-def B''-def by simp
  qed
qed

lemma smc-Rel-is-monic-arrI:
  assumes T : A ↦_smc-Rel α B
    and ⋀y z X. [[ y ⊆ₒ A; z ⊆ₒ A; T(|ArrVal|) `ₒ y = X; T(|ArrVal|) `ₒ z = X ]] ⟹
      y = z
  shows T : A ↦_mon smc-Rel α B
proof(rule is-monic-arrI)

  interpret T: arr-Rel α T
    rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
    by (intro smc-Rel-is-arrD[OF assms(1)])+

  interpret Rel: semicategory α ‹smc-Rel α› by (simp add: T.semicategory-smc-Rel)

  fix R S A' assume prems:
    R : A' ↦_smc-Rel α A
    S : A' ↦_smc-Rel α A
    T ∘_A smc-Rel α R = T ∘_A smc-Rel α S

  interpret R: arr-Rel α R
    rewrites [simp]: R(|ArrDom|) = A' and [simp]: R(|ArrCod|) = A
    by (intro smc-Rel-is-arrD[OF prems(1)])+
  interpret S: arr-Rel α S
```

```
    rewrites [simp]: S(|ArrDom|) = A' and [simp]: S(|ArrCod|) = A
    by (intro smc-Rel-is-arrD[OF prems(2)])+

  from assms prems have
    R(|ArrVal|) ⊆∘ S(|ArrVal|) S(|ArrVal|) ⊆∘ R(|ArrVal|)
    by (auto simp: smc-Rel-is-monic-arr-vsubset)
  then show R = S
    using R.arr-Rel-axioms S.arr-Rel-axioms
    by (intro arr-Rel-eqI[of α R S]) auto

qed (rule assms(1))

end

lemma smc-Rel-is-monic-arrD[dest]:
  assumes T : A ↦_mon smc-Rel α B
    and y ⊆∘ A
    and z ⊆∘ A
    and T(|ArrVal|) `∘ y = X
    and T(|ArrVal|) `∘ z = X
  shows y = z
proof–

  from assms have T: T : A ↦_smc-Rel α B by (simp add: is-monic-arr-def)

  interpret T: arr-Rel α T
    rewrites T(|ArrDom|) = A and [simp]: T(|ArrCod|) = B
    by (intro smc-Rel-is-arrD[OF T])+

  interpret Rel: semicategory α ‹smc-Rel α›
    by (simp add: T.semicategory-smc-Rel)

  define R where R = [set {0} ×∘ y, set {0}, A]∘
  define S where S = [set {0} ×∘ z, set {0}, A]∘

  have R: R : set {0} ↦_smc-Rel α A
  proof(intro smc-Rel-is-arrI)
    show arr-Rel α R
      unfolding R-def
    proof(intro T.arr-Rel-vfsequenceI)
      from assms(2) show R_∘ (set {0} ×∘ y) ⊆∘ A by auto
    qed (auto simp: T.arr-Rel-ArrDom-in-Vset)
  qed (simp-all add: R-def arr-Rel-components)

  from assms(3) have S: S : set {0} ↦_smc-Rel α A
  proof(intro smc-Rel-is-arrI)
    show arr-Rel α S
      unfolding S-def
    proof(intro T.arr-Rel-vfsequenceI)
      from assms(3) show R_∘ (set {0} ×∘ z) ⊆∘ A by auto
    qed (auto simp: T.arr-Rel-ArrDom-in-Vset)
  qed (simp-all add: S-def arr-Rel-components)

  from assms(4) have T ∘_A smc-Rel α R = [set {0} ×∘ X, set {0}, B]∘
    unfolding smc-Rel-Comp-app[OF T R]
    unfolding comp-Rel-components R-def comp-Rel-def arr-Rel-components
    by (simp add: vcomp-vimage-vtimes-right)
  moreover from assms have T ∘_A smc-Rel α S = [set {0} ×∘ X, set {0}, B]∘
```

```
    unfolding smc-Rel-Comp-app[OF T S]
    unfolding comp-Rel-components S-def comp-Rel-def arr-Rel-components
    by (simp add: vcomp-vimage-vtimes-right)
  ultimately have T ∘_{A smc-Rel α} R = T ∘_{A smc-Rel α} S by simp
  from R S assms(1) this have R = S by (elim is-monic-arrE)
  then show y = z unfolding R-def S-def by auto

qed

lemma smc-Rel-is-monic-arr:
  T : A ↦_{mon smc-Rel α} B ⟷
    T : A ↦_{smc-Rel α} B ∧
    (
      ∀ y z X.
        y ⊆_∘ A ⟶
        z ⊆_∘ A ⟶
        (T⦇ArrVal⦈) `_∘ y = X ⟶
        (T⦇ArrVal⦈) `_∘ z = X ⟶
        y = z
    )
  by (rule iffI allI impI)
    (auto simp: smc-Rel-is-monic-arrD smc-Rel-is-monic-arrI)

lemma smc-Rel-is-monic-arr-is-epic-arr:
  assumes T : A ↦_{smc-Rel α} B
    and (†_{SMC.Rel α})(⦇ArrMap⦈)(⦇T⦈) : B ↦_{mon smc-Rel α} A
  shows T : A ↦_{epi smc-Rel α} B
proof–

  interpret T: arr-Rel α T
    rewrites T(⦇ArrDom⦈) = A and [simp]: T(⦇ArrCod⦈) = B
    by (intro smc-Rel-is-arrD[OF assms(1)])+

  interpret is-iso-semifunctor α ‹op-smc (smc-Rel α)› ‹smc-Rel α› ‹†_{SMC.Rel} α›
    rewrites op-smc 𝔊′(⦇Obj⦈) = 𝔊′(⦇Obj⦈)
      and op-smc 𝔊′(⦇Arr⦈) = 𝔊′(⦇Arr⦈)
      and f : b ↦_{op-smc 𝔊′} a ⟷ f : a ↦_{𝔊′} b
      for 𝔊′ f a b
    unfolding smc-op-simps by (auto simp: T.smcf-dag-Rel-is-iso-semifunctor)

  show ?thesis
  proof(intro HomCod.is-epic-arrI)

    show T: T : A ↦_{smc-Rel α} B by (rule assms(1))

    fix f g a assume prems:
      f : B ↦_{smc-Rel α} a
      g : B ↦_{smc-Rel α} a
      f ∘_{A smc-Rel α} T = g ∘_{A smc-Rel α} T

    from prems(1) have †_{SMC.Rel} α(⦇ArrMap⦈)(⦇f⦈) :
      †_{SMC.Rel} α(⦇ObjMap⦈)(⦇a⦈) ↦_{smc-Rel α} †_{SMC.Rel} α(⦇ObjMap⦈)(⦇B⦈)
      by (auto intro: smc-cs-intros)
    with prems(1) HomCod.smc-is-arrD(3) T have dag-f:
      †_{SMC.Rel} α(⦇ArrMap⦈)(⦇f⦈) : a ↦_{smc-Rel α} B
      unfolding smcf-dag-Rel-components(1) by auto
    from prems(2) have †_{SMC.Rel} α(⦇ArrMap⦈)(⦇g⦈) :
      †_{SMC.Rel} α(⦇ObjMap⦈)(⦇a⦈) ↦_{smc-Rel α} †_{SMC.Rel} α(⦇ObjMap⦈)(⦇B⦈)
```

```
      by (auto intro: smc-cs-intros)
    with prems(2) have dag-g: †SMC.Rel α(|ArrMap|)(|g|) : a ↦smc-Rel α B
      unfolding smcf-dag-Rel-components(1)
      by (metis HomCod.smc-is-arrD(3) T vid-on-eq-atI)
    from prems T have
      †SMC.Rel α(|ArrMap|)(|T|) ∘Asmc-Rel α †SMC.Rel α(|ArrMap|)(|f|) =
        †SMC.Rel α(|ArrMap|)(|T|) ∘Asmc-Rel α †SMC.Rel α(|ArrMap|)(|g|)
      by (simp add: smcf-dag-Rel-ArrMap-smc-Rel-Comp[symmetric])
    from is-monic-arrD(2)[OF assms(2) dag-f dag-g this] show f = g
      by (meson prems HomDom.smc-is-arrD(1) ArrMap.v11-eq-iff)

  qed
qed

lemma smc-Rel-is-epic-arr-is-monic-arr:
  assumes T : A ↦epismc-Rel α B
  shows †SMC.Rel α(|ArrMap|)(|T|) : B ↦monsmc-Rel α A
proof(rule is-monic-arrI)

  from assms(1) have T: T : A ↦smc-Rel α B
    by (simp add: is-epic-arr-def is-monic-arr-def smc-op-simps)

  interpret T: arr-Rel α T
    rewrites T(|ArrDom|) = A and [simp]: T(|ArrCod|) = B
    by (intro smc-Rel-is-arrD[OF T])+

  interpret is-iso-semifunctor α ‹op-smc (smc-Rel α)› ‹smc-Rel α› ‹†SMC.Rel α›
    rewrites f : b ↦op-smc 𝔈′ a ⟷ f : a ↦𝔈′ b for 𝔈′ f a b
    unfolding smc-op-simps by (auto simp: T.smcf-dag-Rel-is-iso-semifunctor)

  have dag: †SMC.Rel α : op-smc (smc-Rel α) ↦↦SMCα smc-Rel α
    by (auto intro: smc-cs-intros)

  from HomCod.is-epic-arrD(1)[OF assms] have T: T : A ↦smc-Rel α B.

  from T have †SMC.Rel α(|ArrMap|)(|T|) :
    †SMC.Rel α(|ObjMap|)(|B|) ↦smc-Rel α †SMC.Rel α(|ObjMap|)(|A|)
    by (auto intro: smc-cs-intros)
  with T show dag-T: †SMC.Rel α(|ArrMap|)(|T|) : B ↦smc-Rel α A
    unfolding smcf-dag-Rel-components(1)
    by (metis HomCod.smc-is-arrD(2) HomCod.smc-is-arrD(3) vid-on-eq-atI)

  fix f g a assume prems:
    f : a ↦smc-Rel α B
    g : a ↦smc-Rel α B
    †SMC.Rel α(|ArrMap|)(|T|) ∘Asmc-Rel α f = †SMC.Rel α(|ArrMap|)(|T|) ∘Asmc-Rel α g
  then have a: a ∈∘ smc-Rel α(|Obj|) by auto
  from prems(1) have †SMC.Rel α(|ArrMap|)(|f|) :
    †SMC.Rel α(|ObjMap|)(|B|) ↦smc-Rel α †SMC.Rel α(|ObjMap|)(|a|)
    by (auto intro: smc-cs-intros)
  with prems(1) have dag-f: †SMC.Rel α(|ArrMap|)(|f|) : B ↦smc-Rel α a
    by (cs-concl cs-intro: smc-cs-intros cs-simp: smc-Rel-cs-simps)
  from prems(2) have †SMC.Rel α(|ArrMap|)(|g|) :
    †SMC.Rel α(|ObjMap|)(|B|) ↦smc-Rel α †SMC.Rel α(|ObjMap|)(|a|)
    by (cs-concl cs-shallow cs-intro: smc-cs-intros cs-simp:)
  with prems(2) have dag-g: †SMC.Rel α(|ArrMap|)(|g|) : B ↦smc-Rel α a
    by (cs-concl cs-intro: smc-cs-intros cs-simp: smc-Rel-cs-simps)
  from T dag have
```

```
    †SMC.Rel α(|ArrMap|)(|†SMC.Rel α(|ArrMap|)(|T|)|) =
      (†SMC.Rel α SMCF∘ †SMC.Rel α)(|ArrMap|)(|T|)
    by
      (
        cs-concl
          cs-intro: smc-cs-intros
          cs-simp: smc-Rel-cs-simps smc-cn-cs-simps smc-cs-simps
      )
  also from T have ... = T
    unfolding dghm-id-components T.smcf-cn-comp-smcf-dag-Rel-smcf-dag-Rel
    by auto
  finally have dag-dag-T: †SMC.Rel α(|ArrMap|)(|†SMC.Rel α(|ArrMap|)(|T|)|) = T by simp
  have
    †SMC.Rel α(|ArrMap|)(|f|) ∘A smc-Rel α T = †SMC.Rel α(|ArrMap|)(|g|) ∘A smc-Rel α T
    by (metis dag-T dag-dag-T prems smcf-dag-Rel-ArrMap-smc-Rel-Comp)
  from HomCod.is-epic-arrD(2)[OF assms dag-f dag-g this] prems ArrMap.v11-eq-iff
  show f = g
    by blast

qed

lemma smc-Rel-is-epic-arrI:
  assumes T : A ↦smc-Rel α B
    and ⋀y z X. [[ y ⊆∘ B; z ⊆∘ B; T(|ArrVal|) -`∘ y = X; T(|ArrVal|) -`∘ z = X ]] ⟹
      y = z
  shows T : A ↦epi smc-Rel α B
proof–

  interpret T: arr-Rel α T
    rewrites T(|ArrDom|) = A and [simp]: T(|ArrCod|) = B
    by (intro smc-Rel-is-arrD[OF assms(1)])+

  interpret is-iso-semifunctor α ‹op-smc (smc-Rel α)› ‹smc-Rel α› ‹†SMC.Rel α›
    rewrites f : b ↦op-smc 𝔊' a ⟷ f : a ↦𝔊' b for 𝔊' f a b
    unfolding smc-op-simps by (auto simp: T.smcf-dag-Rel-is-iso-semifunctor)

  have †SMC.Rel α(|ArrMap|)(|T|) : B ↦mon smc-Rel α A
  proof(rule smc-Rel-is-monic-arrI)
    from assms(1) have †SMC.Rel α(|ArrMap|)(|T|) :
      †SMC.Rel α(|ObjMap|)(|B|) ↦smc-Rel α †SMC.Rel α(|ObjMap|)(|A|)
      by (cs-concl cs-shallow cs-intro: smc-cs-intros)
    with assms(1) show †SMC.Rel α(|ArrMap|)(|T|) : B ↦smc-Rel α A
      by (cs-concl cs-intro: smc-cs-intros cs-simp: smc-Rel-cs-simps)
    fix y z X
    assume
      y ⊆∘ B
      z ⊆∘ B
      †SMC.Rel α(|ArrMap|)(|T|)(|ArrVal|) `∘ y = X
      †SMC.Rel α(|ArrMap|)(|T|)(|ArrVal|) `∘ z = X
    then show y = z
      unfolding
        converse-Rel-components
        smcf-dag-Rel-ArrMap-app[OF T.arr-Rel-axioms]
        app-invimage-def[symmetric]
      by (rule assms(2))
  qed
  from smc-Rel-is-monic-arr-is-epic-arr[OF assms(1) this] show ?thesis by simp
qed
```

lemma *smc-Rel-is-epic-arrD*[*dest*]:
assumes $T : A \mapsto_{epi} \textit{smc-Rel}\ \alpha\ B$
and $y \subseteq_\circ B$
and $z \subseteq_\circ B$
and $T(\!|\textit{ArrVal}|\!) -`_\circ y = X$
and $T(\!|\textit{ArrVal}|\!) -`_\circ z = X$
shows $y = z$
proof–

from *assms*(1) **have** *T*: $T : A \mapsto_{\textit{smc-Rel}\ \alpha}\ B$
by (*simp add*: *is-epic-arr-def is-monic-arr-def smc-op-simps*)

interpret *T*: *arr-Rel* α *T*
rewrites $T(\!|\textit{ArrDom}|\!) = A$ **and** [*simp*]: $T(\!|\textit{ArrCod}|\!) = B$
by (*intro smc-Rel-is-arrD*[*OF T*])+

interpret *is-iso-semifunctor* α ‹*op-smc* (*smc-Rel* α)› ‹*smc-Rel* α› ‹$\dagger_{SMC.Rel}\ \alpha$›
rewrites $f : b \mapsto_{\textit{op-smc}\ \mathfrak{C}'} a \longleftrightarrow f : a \mapsto_{\mathfrak{C}'} b$
for $\mathfrak{C}'\ f\ a\ b$
unfolding *smc-op-simps* **by** (*auto simp*: *T.smcf-dag-Rel-is-iso-semifunctor*)
have *dag-T*: $\dagger_{SMC.Rel}\ \alpha(\!|\textit{ArrMap}|\!)(\!|T|\!) : B \mapsto_{mon\textit{smc-Rel}\ \alpha} A$
by (*rule smc-Rel-is-epic-arr-is-monic-arr*[*OF assms*(1)])
from *HomCod.is-epic-arrD*(1)[*OF assms*(1)] **have** *T*: $T : A \mapsto_{\textit{smc-Rel}\ \alpha}\ B$**.**
then have *T*: *arr-Rel* α *T* **by** (*auto simp*: *smc-Rel-is-arrD*(1))
from
assms(4,5)
smc-Rel-is-monic-arrD
[
OF dag-T assms(2,3),
unfolded
smc-dg-smc-Rel
smcf-dghm-smcf-dag-Rel
converse-Rel-components
smcf-dag-Rel-ArrMap-app[*OF T*]
]
show *?thesis*
by (*auto simp*: *app-invimage-def*)
qed

lemma *smc-Rel-is-epic-arr*:
$T : A \mapsto_{epi\textit{smc-Rel}\ \alpha}\ B \longleftrightarrow$
$T : A \mapsto_{\textit{smc-Rel}\ \alpha}\ B \wedge$
(
$\forall\, y\ z\ X.$
$y \subseteq_\circ B \longrightarrow$
$z \subseteq_\circ B \longrightarrow$
$T(\!|\textit{ArrVal}|\!) -`_\circ y = X \longrightarrow$
$T(\!|\textit{ArrVal}|\!) -`_\circ z = X \longrightarrow$
$y = z$
)
proof(*intro iffI allI impI conjI*)
show $T : A \mapsto_{epi\textit{smc-Rel}\ \alpha}\ B \Longrightarrow T : A \mapsto_{\textit{smc-Rel}\ \alpha}\ B$
by (*simp add*: *is-epic-arr-def is-monic-arr-def op-smc-is-arr*)
qed (*auto simp*: *smc-Rel-is-epic-arrI*)

4.11.5 Terminal object, initial object and null object

An object in the semicategory *Rel* is terminal/initial/null if and only if it is the empty set (see nLab [3])[7].

lemma (in $\mathcal{Z}$**)** *smc-Rel-obj-terminal*: *obj-terminal* (*smc-Rel* α) $A \longleftrightarrow A = 0$
proof–

interpret *semicategory* α ‹*smc-Rel* α› **by** (*rule semicategory-smc-Rel*)

have $(\forall A \in_\circ Vset\ \alpha.\ \exists! T.\ T : A \mapsto_{smc\text{-}Rel\ \alpha} B) \longleftrightarrow B = 0$ **for** B
proof(*intro iffI allI ballI*)

assume *prems*[*rule-format*]: $\forall A \in_\circ Vset\ \alpha.\ \exists! T.\ T : A \mapsto_{smc\text{-}Rel\ \alpha} B$

then obtain T **where** $T : 0 \mapsto_{smc\text{-}Rel\ \alpha} B$ **by** (*meson vempty-is-zet*)
then have [*simp*]: $B \in_\circ Vset\ \alpha$ **by** (*fastforce simp*: *smc-Rel-components*(1))

show $B = 0$
proof(*rule ccontr*)
assume $B \neq 0$
with *trad-foundation* **obtain** b **where** $b \in_\circ B$ **by** *auto*
let *?b0B* = ‹$[set\ \{\langle 0, b\rangle\}, set\ \{0\}, B]_\circ$›
let *?z0B* = ‹$[0, set\ \{0\}, B]_\circ$›
have *?b0B* : $set\ \{0\} \mapsto_{smc\text{-}Rel\ \alpha} B$
proof(*intro smc-Rel-is-arrI*)
show *b0B*: *arr-Rel* α *?b0B*
by (*intro arr-Rel-vfsequenceI*)
(*force simp*: ‹$b \in_\circ B$› *vsubset-vsingleton-leftI*)+
qed (*simp-all add*: *arr-Rel-components*)
moreover have *?z0B* : $set\ \{0\} \mapsto_{smc\text{-}Rel\ \alpha} B$
proof(*intro smc-Rel-is-arrI*)
show *b0B*: *arr-Rel* α *?z0B*
by (*intro arr-Rel-vfsequenceI*)
(*force simp*: ‹$b \in_\circ B$› *vsubset-vsingleton-leftI*)+
qed (*simp-all add*: *arr-Rel-components*)
moreover have $[set\ \{\langle 0, b\rangle\}, set\ \{0\}, B]_\circ \neq [0, set\ \{0\}, B]_\circ$ **by** *simp*
ultimately show *False*
by (*metis prems smc-is-arrE smc-Rel-components*(1))
qed

next

fix A **assume** *prems*[*simp*]: $B = 0\ A \in_\circ Vset\ \alpha$
let *?zAz* = ‹$[0, A, 0]_\circ$›
have *zAz*: *arr-Rel* α *?zAz*
by
(
simp add:
$\mathcal{Z}$*.arr-Rel-vfsequenceI*
$\mathcal{Z}$*-axioms*
smc-Rel-components(2)
vbrelation-vempty
)

show $\exists! T.\ T : A \mapsto_{smc\text{-}Rel\ \alpha} B$
proof(*rule ex1I*[*of* - ‹*?zAz*›])

[7] https://ncatlab.org/nlab/show/database+of+categories

```
    show ?zAz : A ↦_smc-Rel α B
      by (intro smc-Rel-is-arrI)
        (
          simp-all add:
            zAz
            smc-Rel-Dom-app[OF zAz]
            smc-Rel-Cod-app[OF zAz]
            arr-Rel-components
        )
    fix T assume T : A ↦_smc-Rel α B
    then have T: T : A ↦_smc-Rel α 0 by simp
    then interpret T: arr-Rel α T by (fastforce simp: smc-Rel-components(2))
    show T = [0, A, 0]∘
    proof
      (
        subst T.arr-Rel-def,
        rule arr-Rel-eqI[of α],
        unfold arr-Rel-components
      )
      show arr-Rel α [T(|ArrVal|), T(|ArrDom|), T(|ArrCod|)]∘
        by (fold T.arr-Rel-def) (simp add: T.arr-Rel-axioms)
      from zAz show arr-Rel α ?zAz
        by (simp add: arr-Rel-vfsequenceI vbrelationI)
      from T have T ∈∘ smc-Rel α(|Arr|) by (auto intro: smc-cs-intros)
      with is-arrD(2,3)[OF T] show T(|ArrDom|) = A T(|ArrCod|) = 0
        using T smc-Rel-is-arrD(2,3) by auto
      with T.arr-Rel-ArrVal-vrange T.ArrVal.vbrelation-vintersection-vrange
      show T(|ArrVal|) = 0
        by auto
    qed

  qed

  then show ?thesis
    apply(intro iffI obj-terminalI)
    subgoal by (metis smc-is-arrD(2) obj-terminalE)
    subgoal by blast
    subgoal by (metis smc-Rel-components(1))
    done

qed


lemma (in 𝒵) smc-Rel-obj-initial: obj-initial (smc-Rel α) A ⟷ A = 0
proof–

  interpret semicategory α ‹smc-Rel α› by (rule semicategory-smc-Rel)

  have (∀ B∈∘Vset α. ∃!T. T : A ↦_smc-Rel α B) ⟷ A = 0 for A
  proof(intro iffI allI ballI)

    assume prems[rule-format]: ∀ B∈∘Vset α. ∃!T. T : A ↦_smc-Rel α B

    then obtain T where TA0: T : A ↦_smc-Rel α 0 by (meson vempty-is-zet)
    then have [simp]: A ∈∘ Vset α by (fastforce simp: smc-Rel-components(1))
```

```
  show A = 0
  proof(rule ccontr)
    assume A ≠ 0
    with trad-foundation obtain a where a ∈ₒ A by auto
    have [set {⟨a, 0⟩}, A, set {0}]ₒ : A ↦_smc-Rel α set {0}
    proof(intro smc-Rel-is-arrI)
      show arr-Rel α [set {⟨a, 0⟩}, A, set {0}]ₒ
        by (intro arr-Rel-vfsequenceI)
          (auto simp: ‹a ∈ₒ A› vsubset-vsingleton-leftI)
    qed (simp-all add: arr-Rel-components)
    moreover have [0, A, set {0}]ₒ : A ↦_smc-Rel α set {0}
    proof(intro smc-Rel-is-arrI)
      show arr-Rel α [0, A, set {0}]ₒ
        by (intro arr-Rel-vfsequenceI)
          (auto simp: ‹a ∈ₒ A› vsubset-vsingleton-leftI)
    qed (simp-all add: arr-Rel-components)
    moreover have [set {⟨a, 0⟩}, A, set {0}]ₒ ≠ [0, A, set {0}]ₒ by simp
    ultimately show False
      by (metis prems smc-is-arrE smc-Rel-components(1))
  qed
next

  fix B assume [simp]: A = 0 B ∈ₒ Vset α
  show ∃!T. T : A ↦_smc-Rel α B
  proof(rule ex1I[of - ‹[0, 0, B]ₒ›])
    show [0, 0, B]ₒ : A ↦_smc-Rel α B
      by (rule is-arrI)
        (
          simp-all add:
            smc-Rel-cs-simps
            smc-Rel-components(2)
            vbrelation-vempty
            arr-Rel-vfsequenceI
        )
    fix T assume T : A ↦_smc-Rel α B
    then have T: T : 0 ↦_smc-Rel α B by simp
    interpret T: arr-Rel α T
      using T by (fastforce simp: smc-Rel-components(2))
    show T = [0, 0, B]ₒ
    proof
      (
        subst T.arr-Rel-def,
        rule arr-Rel-eqI[of α],
        unfold arr-Rel-components
      )
      show arr-Rel α [T(|ArrVal|), T(|ArrDom|), T(|ArrCod|)]ₒ
        by (fold T.arr-Rel-def) (simp add: T.arr-Rel-axioms)
      show arr-Rel α [0, 0, B]ₒ
        by (simp add: arr-Rel-vfsequenceI vbrelationI)
      from T have T ∈ₒ smc-Rel α(|Arr|) by (auto intro: smc-cs-intros)
      with T is-arrD(2,3)[OF T] show T(|ArrDom|) = 0 T(|ArrCod|) = B
        by (auto simp: smc-Rel-is-arrD(2,3))
      with
        T.arr-Rel-ArrVal-vrange
        T.arr-Rel-ArrVal-vdomain
        T.ArrVal.vbrelation-vintersection-vdomain
      show T(|ArrVal|) = 0
        by auto
```

```
    qed
  qed
qed

then show ?thesis
  apply(intro iffI obj-initialI, elim obj-initialE)
  subgoal by (metis smc-Rel-components(1))
  subgoal by (simp add: smc-Rel-components(1))
  subgoal by (metis smc-Rel-components(1))
  done

qed

lemma (in 𝒵) smc-Rel-obj-terminal-obj-initial:
  obj-initial (smc-Rel α) A ⟷ obj-terminal (smc-Rel α) A
  unfolding smc-Rel-obj-initial smc-Rel-obj-terminal by simp

lemma (in 𝒵) smc-Rel-obj-null: obj-null (smc-Rel α) A ⟷ A = 0
  unfolding obj-null-def smc-Rel-obj-terminal smc-Rel-obj-initial by simp
```

4.11.6 Zero arrow

A zero arrow for *Rel* is any admissible V-arrow, such that its value is the empty set. A reference for this result is not given, but the result is not expected to be original.

```
lemma (in 𝒵) smc-Rel-is-zero-arr:
  assumes A ∈₀ smc-Rel α(|Obj|) and B ∈₀ smc-Rel α(|Obj|)
  shows T : A ↦₀₍smc-Rel α₎ B ⟷ T = [0, A, B]₀
proof(rule HOL.ext iffI)

  interpret Rel: semicategory α ‹smc-Rel α› by (rule semicategory-smc-Rel)

  fix T A B assume T : A ↦₀₍smc-Rel α₎ B
  then obtain R S
    where T-def: T = R ∘₍A smc-Rel α₎ S
      and S: S : A ↦₍smc-Rel α₎ 0
      and R: R : 0 ↦₍smc-Rel α₎ B
    by (elim is-zero-arrE) (simp add: obj-null-def smc-Rel-obj-terminal)

  interpret S: arr-Rel α S
    rewrites [simp]: S(|ArrDom|) = A and [simp]: S(|ArrCod|) = 0
    by (intro smc-Rel-is-arrD[OF S])+
  interpret R: arr-Rel α R
    rewrites [simp]: R(|ArrDom|) = 0 and [simp]: R(|ArrCod|) = B
    by (intro smc-Rel-is-arrD[OF R])+

  have S-def: S = [0, A, 0]₀
    by
      (
        rule arr-Rel-eqI[of α],
        unfold arr-Rel-components,
        insert S.arr-Rel-ArrVal-vrange S.ArrVal.vbrelation-vintersection-vrange
      )
      (
        auto simp:
          S.arr-Rel-axioms
          S.arr-Rel-ArrDom-in-Vset
          arr-Rel-vfsequenceI
```

```
        vbrelationI
    )
  show T = [0, A, B]∘
    unfolding T-def smc-Rel-Comp-app[OF R S]
    by (rule arr-Rel-eqI[of α], unfold comp-Rel-components)
     (
       auto simp:
        S-def
        Z-axioms
        R.arr-Rel-axioms
        S.arr-Rel-axioms
        arr-Rel-comp-Rel
        arr-Rel-components
        R.arr-Rel-ArrCod-in-Vset
        S.arr-Rel-ArrDom-in-Vset
        Z.arr-Rel-vfsequenceI
        vbrelation-vempty
     )

next

  assume prems: T = [0, A, B]∘
  let ?S = ‹[0, A, 0]∘› and ?R = ‹[0, 0, B]∘›
  have S: arr-Rel α ?S and R: arr-Rel α ?R
    by (all‹intro arr-Rel-vfsequenceI›)
      (auto simp: assms[unfolded smc-Rel-components])
  have SA0: ?S : A ↦_smc-Rel α 0
    by (intro smc-Rel-is-arrI) (simp-all add: S arr-Rel-components)
  moreover have R0B: ?R : 0 ↦_smc-Rel α B
    by (intro smc-Rel-is-arrI) (simp-all add: R arr-Rel-components)
  moreover have T = ?R ∘_A smc-Rel α ?S
    unfolding smc-Rel-Comp-app[OF R0B SA0]
  proof(rule arr-Rel-eqI, unfold comp-Rel-components arr-Rel-components prems)
    show arr-Rel α [0, A, B]∘
      unfolding prems
      by (intro arr-Rel-vfsequenceI)
        (auto simp: assms[unfolded smc-Rel-components(1)])
  qed (use R S in ‹auto simp: smc-Rel-cs-intros›)
  ultimately show T : A ↦_0 smc-Rel α B
    by (simp add: is-zero-arrI smc-Rel-obj-null)

qed
```

4.12 *Par* as a semicategory

4.12.1 Background

The methodology chosen for the exposition of *Par* as a semicategory is analogous to the one used in the previous chapter for the exposition of *Par* as a digraph.

named-theorems *smc-Par-cs-simps*
named-theorems *smc-Par-cs-intros*

lemmas (**in** *arr-Par*) [*smc-Par-cs-simps*] =
dg-Rel-shared-cs-simps

lemmas (**in** *arr-Par*) [*smc-cs-intros*, *smc-Par-cs-intros*] =
arr-Par-axioms′

lemmas [*smc-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 [*smc-Par-cs-intros*] =
dg-Rel-shared-cs-intros
arr-Par-comp-Par

4.12.2 *Par* as a semicategory

Definition and elementary properties

definition *smc-Par* :: $V \Rightarrow V$
where *smc-Par* α =
[
Vset α,
set $\{T.$ *arr-Par* α $T\}$,
$(\lambda T \in_\circ set\ \{T.$ *arr-Par* α $T\}.$ $T(\!|ArrDom|\!))$,
$(\lambda T \in_\circ set\ \{T.$ *arr-Par* α $T\}.$ $T(\!|ArrCod|\!))$,
$(\lambda ST \in_\circ composable\text{-}arrs\ (dg\text{-}Par\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$
]$_\circ$

Components.

lemma *smc-Par-components*:
shows *smc-Par* $\alpha(\!|Obj|\!)$ = *Vset* α
and *smc-Par* $\alpha(\!|Arr|\!)$ = *set* $\{T.$ *arr-Par* α $T\}$
and *smc-Par* $\alpha(\!|Dom|\!) = (\lambda T \in_\circ set\ \{T.$ *arr-Par* α $T\}.$ $T(\!|ArrDom|\!))$
and *smc-Par* $\alpha(\!|Cod|\!) = (\lambda T \in_\circ set\ \{T.$ *arr-Par* α $T\}.$ $T(\!|ArrCod|\!))$
and *smc-Par* $\alpha(\!|Comp|\!) = (\lambda ST \in_\circ composable\text{-}arrs\ (dg\text{-}Par\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$
unfolding *smc-Par-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-smc-Par*: *smc-dg* (*smc-Par* α) = *dg-Par* α
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-Par* α)) = $4_{\mathbb{N}}$
unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-Par* α) = $4_{\mathbb{N}}$
unfolding *dg-Par-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*smc-dg* (*smc-Par* α)) = $\mathcal{D}_\circ$ (*dg-Par* α)
unfolding *dom-lhs dom-rhs* **by** *simp*
show $a \in_\circ \mathcal{D}_\circ$ (*smc-dg* (*smc-Par* α)) $\Longrightarrow$ *smc-dg* (*smc-Par* α)$(\!|a|\!)$ = *dg-Par* $\alpha(\!|a|\!)$

```
    for a
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold smc-dg-def dg-field-simps smc-Par-def dg-Par-def
      )
      (auto simp: nat-omega-simps)
qed (auto simp: dg-Par-def smc-dg-def)

lemmas-with [folded smc-dg-smc-Par, unfolded slicing-simps]:
  smc-Par-Obj-iff = dg-Par-Obj-iff
  and smc-Par-Arr-iff[smc-Par-cs-simps] = dg-Par-Arr-iff
  and smc-Par-Dom-vsv[smc-Par-cs-intros] = dg-Par-Dom-vsv
  and smc-Par-Dom-vdomain[smc-Par-cs-simps] = dg-Par-Dom-vdomain
  and smc-Par-Dom-vrange = dg-Par-Dom-vrange
  and smc-Par-Dom-app[smc-Par-cs-simps] = dg-Par-Dom-app
  and smc-Par-Cod-vsv[smc-Par-cs-intros] = dg-Par-Cod-vsv
  and smc-Par-Cod-vdomain[smc-Par-cs-simps] = dg-Par-Cod-vdomain
  and smc-Par-Cod-vrange = dg-Par-Cod-vrange
  and smc-Par-Cod-app[smc-Par-cs-simps] = dg-Par-Cod-app
  and smc-Par-is-arrI = dg-Par-is-arrI
  and smc-Par-is-arrD = dg-Par-is-arrD
  and smc-Par-is-arrE = dg-Par-is-arrE

lemmas [smc-cs-simps] = smc-Par-is-arrD(2,3)

lemmas [smc-Par-cs-intros] = smc-Par-is-arrI

lemmas-with (in 𝒵) [folded smc-dg-smc-Par, unfolded slicing-simps]:
  smc-Par-Hom-vifunion-in-Vset = dg-Par-Hom-vifunion-in-Vset
  and smc-Par-incl-Par-is-arr = dg-Par-incl-Par-is-arr
  and smc-Par-incl-Par-is-arr'[smc-Par-cs-intros] = dg-Par-incl-Par-is-arr'

lemmas [smc-Par-cs-intros] = 𝒵.smc-Par-incl-Par-is-arr'
```

Composable arrows

```
lemma smc-Par-composable-arrs-dg-Par:
  composable-arrs (dg-Par α) = composable-arrs (smc-Par α)
  unfolding composable-arrs-def smc-dg-smc-Par[symmetric] slicing-simps by simp

lemma smc-Par-Comp:
  smc-Par α(|Comp|) = (λST∈∘composable-arrs (smc-Par α). ST(|0|) ∘Rel ST(|1N|))
  unfolding smc-Par-components smc-Par-composable-arrs-dg-Par ..
```

Composition

```
lemma smc-Par-Comp-app[smc-Par-cs-simps]:
  assumes S : B ↦smc-Par α C and T : A ↦smc-Par α B
  shows S ∘Asmc-Par α T = S ∘Rel T
proof–
  from assms have [S, T]∘ ∈∘ composable-arrs (smc-Par α)
    by (auto simp: smc-cs-intros)
  then show S ∘Asmc-Par α T = S ∘Rel T
    unfolding smc-Par-Comp by (simp add: nat-omega-simps)
qed
```

lemma *smc-Par-Comp-vdomain*: $\mathcal{D}_\circ\ (\mathit{smc\text{-}Par}\ \alpha(\!|\mathit{Comp}|\!))\ =\ \mathit{composable\text{-}arrs}\ (\mathit{smc\text{-}Par}\ \alpha)$
unfolding *smc-Par-Comp* **by** *simp*

lemma (**in** $\mathcal{Z}$) *smc-Par-Comp-vrange*: $\mathcal{R}_\circ\ (\mathit{smc\text{-}Par}\ \alpha(\!|\mathit{Comp}|\!))\ \subseteq_\circ\ \mathit{set}\ \{T.\ \mathit{arr\text{-}Par}\ \alpha\ T\}$
proof(*rule vsubsetI*)
interpret *digraph* α ‹*smc-dg* (*smc-Par* α)›
unfolding *smc-dg-smc-Par* **by** (*simp add*: *digraph-dg-Par*)
fix R **assume** $R \in_\circ \mathcal{R}_\circ\ (\mathit{smc\text{-}Par}\ \alpha(\!|\mathit{Comp}|\!))$
then obtain ST
where *R-def*: $R = \mathit{smc\text{-}Par}\ \alpha(\!|\mathit{Comp}|\!)(\!|ST|\!)$
and $ST \in_\circ \mathcal{D}_\circ\ (\mathit{smc\text{-}Par}\ \alpha(\!|\mathit{Comp}|\!))$
unfolding *smc-Par-components* **by** (*blast dest*: *rel-VLambda.vrange-atD*)
then obtain $S\ T\ A\ B\ C$
where $ST = [S,\ T]_\circ$
and *S*: $S : B \mapsto_{\mathit{smc\text{-}Par}\ \alpha} C$
and *T*: $T : A \mapsto_{\mathit{smc\text{-}Par}\ \alpha} B$
by (*auto simp*: *smc-Par-Comp-vdomain*)
with *R-def* **have** *R-def'*: $R = S \circ_{A\,\mathit{smc\text{-}Par}\ \alpha} T$ **by** *simp*
note *S-D* = *dg-is-arrD*(1)[*unfolded slicing-simps*, *OF S*]
and *T-D* = *dg-is-arrD*(1)[*unfolded slicing-simps*, *OF T*]
from *S-D T-D* **have** *arr-Par* α *S arr-Par* α *T*
by (*simp-all add*: *smc-Par-components*)
from *this* **show** $R \in_\circ \mathit{set}\ \{T.\ \mathit{arr\text{-}Par}\ \alpha\ T\}$
unfolding *R-def' smc-Par-Comp-app*[*OF S T*] **by** (*auto simp*: *arr-Par-comp-Par*)
qed

Par is a semicategory

lemma (**in** $\mathcal{Z}$) *semicategory-smc-Par*: *semicategory* α (*smc-Par* α)
proof(*intro semicategoryI*, *unfold smc-dg-smc-Par*)
show *vfsequence* (*smc-Par* α) **unfolding** *smc-Par-def* **by** *simp*
show *vcard* (*smc-Par* α) = $5_{\mathbb{N}}$
unfolding *smc-Par-def* **by** (*simp add*: *nat-omega-simps*)
show $(GF \in_\circ \mathcal{D}_\circ\ (\mathit{smc\text{-}Par}\ \alpha(\!|\mathit{Comp}|\!))) \longleftrightarrow$
$(\exists\, G\ F\ B\ C\ A.\ GF = [G,\ F]_\circ \wedge G : B \mapsto_{\mathit{smc\text{-}Par}\ \alpha} C \wedge F : A \mapsto_{\mathit{smc\text{-}Par}\ \alpha} B)$
for *GF*
unfolding *smc-Par-Comp-vdomain* **by** (*auto intro*: *composable-arrsI*)
show [*intro*]: $G \circ_{A\,\mathit{smc\text{-}Par}\ \alpha} F : A \mapsto_{\mathit{smc\text{-}Par}\ \alpha} C$
if $G : B \mapsto_{\mathit{smc\text{-}Par}\ \alpha} C$ **and** $F : A \mapsto_{\mathit{smc\text{-}Par}\ \alpha} B$ **for** *G B C F A*
proof–
from *that* **have** *arr-Par* α *G arr-Par* α *F* **by** (*auto elim*: *smc-Par-is-arrE*)
with *that* **show** *?thesis*
by
(
cs-concl **cs-shallow**
cs-simp: *smc-cs-simps smc-Par-cs-simps*
cs-intro: *smc-Par-cs-intros*
)
qed
show $H \circ_{A\,\mathit{smc\text{-}Par}\ \alpha} G \circ_{A\,\mathit{smc\text{-}Par}\ \alpha} F = H \circ_{A\,\mathit{smc\text{-}Par}\ \alpha} (G \circ_{A\,\mathit{smc\text{-}Par}\ \alpha} F)$
if $H : C \mapsto_{\mathit{smc\text{-}Par}\ \alpha} D$
and $G : B \mapsto_{\mathit{smc\text{-}Par}\ \alpha} C$
and $F : A \mapsto_{\mathit{smc\text{-}Par}\ \alpha} B$
for *H C D G B F A*
proof–
from *that* **have** *arr-Par* α *H arr-Par* α *G arr-Par* α *F*
by (*auto simp*: *smc-Par-is-arrD*)
with *that* **show** *?thesis*

```
    by
      (
        cs-concl cs-shallow
          cs-simp: smc-cs-simps smc-Par-cs-simps
          cs-intro: smc-Par-cs-intros
      )
  qed
qed (auto simp: digraph-dg-Par smc-Par-components)
```

Par is a wide subsemicategory of *Rel*

```
lemma (in 𝒵) wide-subsemicategory-smc-Par-smc-Rel:
  smc-Par α ⊆_{SMC.wide α} smc-Rel α
proof–
  interpret Rel: semicategory α ‹smc-Rel α› by (rule semicategory-smc-Rel)
  interpret Par: semicategory α ‹smc-Par α› by (rule semicategory-smc-Par)
  show ?thesis
  proof
    (
      intro wide-subsemicategoryI subsemicategoryI,
      unfold smc-dg-smc-Par smc-dg-smc-Rel
    )
    from wide-subdigraph-dg-Par-dg-Rel show wsd:
      dg-Par α ⊆_{DG α} dg-Rel α dg-Par α ⊆_{DG.wide α} dg-Rel α
      by auto
    interpret wide-subdigraph α ‹dg-Par α› ‹dg-Rel α› by (rule wsd(2))
    show G ∘_{A smc-Par α} F = G ∘_{A smc-Rel α} F
      if G : B ↦_{smc-Par α} C and F : A ↦_{smc-Par α} B for G B C F A
    proof–
      from that have G : B ↦_{dg-Par α} C and F : A ↦_{dg-Par α} B
        by
          (
            cs-concl cs-shallow
              cs-simp: smc-dg-smc-Par[symmetric] cs-intro: slicing-intros
          )+
      then have G : B ↦_{dg-Rel α} C and F : A ↦_{dg-Rel α} B
        by (cs-concl cs-shallow cs-intro: dg-sub-fw-cs-intros)+
      then have G : B ↦_{smc-Rel α} C and F : A ↦_{smc-Rel α} B
        unfolding smc-dg-smc-Rel[symmetric] slicing-simps by simp-all
      from that this show G ∘_{A smc-Par α} F = G ∘_{A smc-Rel α} F
        by (cs-concl cs-shallow cs-simp: smc-Par-cs-simps smc-Rel-cs-simps)
    qed
  qed (auto simp: smc-cs-intros)
qed
```

4.12.3 Monic arrow and epic arrow

```
lemma smc-Par-is-monic-arrI[intro]:
  assumes T : A ↦_{smc-Par α} B and v11 (T(|ArrVal|)) and 𝒟∘ (T(|ArrVal|)) = A
  shows T : A ↦_{mon smc-Par α} B
proof(intro is-monic-arrI)

  interpret T: arr-Par α T by (intro smc-Par-is-arrD(1)[OF assms(1)])
  interpret Par-Rel: wide-subsemicategory α ‹smc-Par α› ‹smc-Rel α›
    by (rule T.wide-subsemicategory-smc-Par-smc-Rel)
  interpret v11: v11 ‹T(|ArrVal|)› by (rule assms(2))

  show T: T : A ↦_{smc-Par α} B by (rule assms(1))
```

```
  fix S R A'
  assume S: S : A' ↦_smc-Par α A
    and R: R : A' ↦_smc-Par α A
    and TS-TR: T ∘_A smc-Par α S = T ∘_A smc-Par α R
  from assms(3) T Par-Rel.subsemicategory-axioms have T : A ↦_mon smc-Rel α B
    by (intro smc-Rel-is-monic-arrI)
      (auto dest: v11.v11-vimage-vpsubset-neq elim!: smc-sub-fw-cs-intros)
  moreover from S Par-Rel.subsemicategory-axioms have S : A' ↦_smc-Rel α A
    by (cs-concl cs-shallow cs-intro: smc-sub-fw-cs-intros)
  moreover from R Par-Rel.subsemicategory-axioms have R : A' ↦_smc-Rel α A
    by (cs-concl cs-shallow cs-intro: smc-sub-fw-cs-intros)
  moreover from T S R TS-TR Par-Rel.subsemicategory-axioms have
    T ∘_A smc-Rel α S = T ∘_A smc-Rel α R
    by (auto simp: smc-sub-bw-cs-simps)
  ultimately show S = R by (rule is-monic-arrD(2))
qed

lemma smc-Par-is-monic-arrD:
  assumes T : A ↦_mon smc-Par α B
  shows T : A ↦_smc-Par α B and v11 (T⦇ArrVal⦈) and 𝒟∘ (T⦇ArrVal⦈) = A
proof–

  from assms show T: T : A ↦_smc-Par α B by auto
  interpret T: arr-Par α T
    rewrites [simp]: T⦇ArrDom⦈ = A and [simp]: T⦇ArrCod⦈ = B
    using T by (auto dest: smc-Par-is-arrD)

  show v11 (T⦇ArrVal⦈)
  proof(intro v11I)

    show vsv ((T⦇ArrVal⦈)⁻¹∘)
    proof(intro vsvI)

      fix a b c assume ⟨a, b⟩ ∈∘ (T⦇ArrVal⦈)⁻¹∘ and ⟨a, c⟩ ∈∘ (T⦇ArrVal⦈)⁻¹∘

      then have bar: ⟨b, a⟩ ∈∘ T⦇ArrVal⦈ and car: ⟨c, a⟩ ∈∘ T⦇ArrVal⦈ by auto
      with T.arr-Rel-ArrVal-vdomain have [intro]: b ∈∘ A c ∈∘ A by auto

      define R where R = [set {⟨0, b⟩}, set {0}, A]∘
      define S where S = [set {⟨0, c⟩}, set {0}, A]∘

      have R-components:
        R⦇ArrVal⦈ = set {⟨0, b⟩} R⦇ArrDom⦈ = set {0} R⦇ArrCod⦈ = A
        unfolding R-def by (simp-all add: arr-Rel-components)

      have S-components:
        S⦇ArrVal⦈ = set {⟨0, c⟩} S⦇ArrDom⦈ = set {0} S⦇ArrCod⦈ = A
        unfolding S-def by (simp-all add: arr-Rel-components)

      have R: R : set {0} ↦_smc-Par α A
      proof(rule smc-Par-is-arrI)
        show arr-Par α R
          unfolding R-def
          by (rule T.arr-Par-vfsequenceI) (auto simp: T.arr-Rel-ArrDom-in-Vset)
      qed (simp-all add: R-components)

      have S: S : set {0} ↦_smc-Par α A
      proof(rule smc-Par-is-arrI)
```

```
    show arr-Par α S
      unfolding S-def
      by (rule T.arr-Par-vfsequenceI) (auto simp: T.arr-Rel-ArrDom-in-Vset)
  qed (simp-all add: S-components)

  have T ∘_{A smc-Par α} R = [set {⟨0, a⟩}, set {0}, B]∘
    unfolding smc-Par-Comp-app[OF T R]
  proof
    (
      rule arr-Par-eqI[of α],
      unfold comp-Rel-components arr-Rel-components R-components
    )
    from R T show arr-Par α (T ∘_{Rel} R)
      by (intro arr-Par-comp-Par) (auto elim!: smc-Par-is-arrE)
    show arr-Par α [set {⟨0, a⟩}, set {0}, B]∘
    proof(rule T.arr-Par-vfsequenceI)
      from T.arr-Rel-ArrVal-vrange bar show R∘ (set {⟨0, a⟩}) ⊆∘ B by auto
    qed (auto simp: T.arr-Rel-ArrCod-in-Vset T.Axiom-of-Powers)
    show T(|ArrVal|) ∘∘ set {⟨0, b⟩} = set {⟨0, a⟩}
    proof(rule vsv-eqI, unfold vdomain-vsingleton)
      from bar show D∘ (T(|ArrVal|) ∘∘ set {⟨0, b⟩}) = set {0} by auto
      with bar show
        a' ∈∘ D∘ (T(|ArrVal|) ∘∘ set {⟨0, b⟩}) ⟹
          (T(|ArrVal|) ∘∘ set {⟨0, b⟩})(|a'|) = set {⟨0, a⟩}(|a'|)
        for a'
        by auto
    qed (auto intro: vsv-vcomp)
  qed simp-all

  moreover have T ∘_{A smc-Par α} S = [set {⟨0, a⟩}, set {0}, B]∘
    unfolding smc-Par-Comp-app[OF T S]
  proof
    (
      rule arr-Par-eqI[of α],
      unfold comp-Rel-components arr-Rel-components S-components
    )
    from T S show arr-Par α (T ∘_{Rel} S)
      by (intro arr-Par-comp-Par) (auto elim!: smc-Par-is-arrE)
    show arr-Par α [set {⟨0, a⟩}, set {0}, B]∘
    proof(rule T.arr-Par-vfsequenceI)
      from T.arr-Rel-ArrVal-vrange bar show R∘ (set {⟨0, a⟩}) ⊆∘ B by auto
    qed (auto simp: T.arr-Rel-ArrCod-in-Vset T.Axiom-of-Powers)
    show T(|ArrVal|) ∘∘ set {⟨0, c⟩} = set {⟨0, a⟩}
    proof(rule vsv-eqI, unfold vdomain-vsingleton)
      from car show D∘ (T(|ArrVal|) ∘∘ set {⟨0, c⟩}) = set {0} by auto
      with car show a' ∈∘ D∘ (T(|ArrVal|) ∘∘ set {⟨0, c⟩}) ⟹
        (T(|ArrVal|) ∘∘ set {⟨0, c⟩})(|a'|) = set {⟨0, a⟩}(|a'|)
        for a'
        by auto
    qed (auto intro: vsv-vcomp)
  qed simp-all
  ultimately have T ∘_{A smc-Par α} R = T ∘_{A smc-Par α} S by simp
  from assms R S this have R = S by blast
  with R-components(1) S-components(1) show b = c by simp
 qed auto

qed auto
```

```
  show 𝒟∘ (T(|ArrVal|)) = A
  proof(intro vsubset-antisym vsubsetI)
    from T.arr-Rel-ArrVal-vdomain show x ∈∘ 𝒟∘ (T(|ArrVal|)) ⟹ x ∈∘ A for x
      by auto
    fix a assume [simp]: a ∈∘ A show a ∈∘ 𝒟∘ (T(|ArrVal|))
    proof(rule ccontr)
      assume a: a ∉∘ 𝒟∘ (T(|ArrVal|))
      define R where R = [set {⟨0, a⟩}, set {0, 1}, A]∘
      define S where S = [set {⟨1, a⟩}, set {0, 1}, A]∘
      have R: R : set {0, 1} ↦_smc-Par α A
      proof(rule smc-Par-is-arrI)
        show arr-Par α R
          unfolding R-def
        proof(rule T.arr-Par-vfsequenceI)
          from T.Axiom-of-Infinity vone-in-omega show set {0, 1} ∈∘ Vset α
            by blast
        qed (auto simp: T.arr-Rel-ArrDom-in-Vset)
      qed (auto simp: R-def arr-Rel-components)
      have S: S : set {0, 1} ↦_smc-Par α A
      proof(rule smc-Par-is-arrI)
        show arr-Par α S
          unfolding S-def
        proof(rule T.arr-Par-vfsequenceI)
          from T.Axiom-of-Infinity vone-in-omega show set {0, 1} ∈∘ Vset α
            by blast
        qed (auto simp: T.arr-Rel-ArrDom-in-Vset)
      qed (auto simp: S-def arr-Rel-components)
      with a have T(|ArrVal|) ∘∘ R(|ArrVal|) = 0
        unfolding R-def arr-Rel-components
        by (intro vsubset-antisym vsubsetI) auto
      moreover with a have T(|ArrVal|) ∘∘ S(|ArrVal|) = 0
        unfolding S-def arr-Rel-components
        by (intro vsubset-antisym vsubsetI) auto
      ultimately have T ∘_A smc-Par α R = T ∘_A smc-Par α S
        using R T S
        by
          (
            intro arr-Par-eqI[of α ‹T ∘_A smc-Par α R› ‹T ∘_A smc-Par α S›];
            elim smc-Par-is-arrE
          )
          (
            auto simp:
              dg-Par-cs-intros
              smc-Par-Comp-app[OF T R]
              smc-Par-Comp-app[OF T S]
              comp-Rel-components
          )
      from R S this assms have R = S by blast
      then show False unfolding R-def S-def by simp
    qed
  qed

qed

lemma smc-Par-is-monic-arr:
  T : A ↦_mon smc-Par α B ⟷
    T : A ↦_smc-Par α B ∧ v11 (T(|ArrVal|)) ∧ 𝒟∘ (T(|ArrVal|)) = A
  by (intro iffI) (auto simp: smc-Par-is-monic-arrD smc-Par-is-monic-arrI)
```

context
begin

private lemma *smc-Par-is-epic-arr-vsubset*:
assumes $T : A \mapsto_{smc\text{-}Par\ \alpha} B$
and $\mathcal{R}_\circ\ (T(\!|ArrVal|\!)) = B$
and $R : B \mapsto_{smc\text{-}Par\ \alpha} C$
and $S : B \mapsto_{smc\text{-}Par\ \alpha} C$
and $R \circ_{A smc\text{-}Par\ \alpha} T = S \circ_{A smc\text{-}Par\ \alpha} T$
shows $R(\!|ArrVal|\!) \subseteq_\circ S(\!|ArrVal|\!)$
proof
interpret *T*: *arr-Par* α *T*
rewrites [*simp*]: $T(\!|ArrDom|\!) = A$ **and** [*simp*]: $T(\!|ArrCod|\!) = B$
using *assms smc-Par-is-arrD* **by** *auto*
interpret *R*: *arr-Par* α *R*
rewrites [*simp*]: $R(\!|ArrDom|\!) = B$ **and** [*simp*]: $R(\!|ArrCod|\!) = C$
using *assms smc-Par-is-arrD* **by** *auto*
from *assms*(5) **have** $(R \circ_{A smc\text{-}Par\ \alpha} T)(\!|ArrVal|\!) = (S \circ_{A smc\text{-}Par\ \alpha} T)(\!|ArrVal|\!)$
by *simp*
then have *eq*: $R(\!|ArrVal|\!) \circ_\circ T(\!|ArrVal|\!) = S(\!|ArrVal|\!) \circ_\circ T(\!|ArrVal|\!)$
unfolding
smc-Par-Comp-app[*OF assms*(3,1)]
smc-Par-Comp-app[*OF assms*(4,1)]
comp-Rel-components
by *simp*
fix *bc* **assume** *prems*: $bc \in_\circ R(\!|ArrVal|\!)$
moreover with *R.ArrVal.vbrelation* **obtain** *b c* **where** *bc-def*: $bc = \langle b, c\rangle$ **by** *auto*
ultimately have [*simp*]: $b \in_\circ B$ **and** $c \in_\circ C$
using *R.arr-Rel-ArrVal-vdomain R.arr-Rel-ArrVal-vrange* **by** *auto*
note [*intro*] = *prems*[*unfolded bc-def*]
have $b \in_\circ \mathcal{R}_\circ\ (T(\!|ArrVal|\!))$ **by** (*simp add*: *assms*(2))
then obtain *a* **where** *ab*: $\langle a, b\rangle \in_\circ T(\!|ArrVal|\!)$ **by** *auto*
then have $\langle a, c\rangle \in_\circ S(\!|ArrVal|\!) \circ_\circ T(\!|ArrVal|\!)$ **unfolding** *eq*[*symmetric*] **by** *auto*
then obtain b' **where** *ab'*: $\langle b', c\rangle \in_\circ S(\!|ArrVal|\!)$ **and** $\langle a, b'\rangle \in_\circ T(\!|ArrVal|\!)$
by *clarsimp*
with *ab ab' T.vsv T.ArrVal.vsv* **show** $bc \in_\circ S(\!|ArrVal|\!)$ **unfolding** *bc-def* **by** *blast*
qed

lemma *smc-Par-is-epic-arrI*:
assumes $T : A \mapsto_{smc\text{-}Par\ \alpha} B$ **and** $\mathcal{R}_\circ\ (T(\!|ArrVal|\!)) = B$
shows $T : A \mapsto_{epi smc\text{-}Par\ \alpha} B$
unfolding *is-epic-arr-def*
proof
(
intro is-monic-arrI[
of ‹*op-smc* (*smc-Par* α)›, *unfolded smc-op-simps*, *OF assms*(1)
]
)

interpret *T*: *arr-Par* α *T*
rewrites [*simp*]: $T(\!|ArrDom|\!) = A$ **and** [*simp*]: $T(\!|ArrCod|\!) = B$
using *assms smc-Par-is-arrD* **by** *auto*

interpret *semicategory* α ‹*smc-Par* α› **by** (*rule T.semicategory-smc-Par*)

fix *R S a*
assume *prems*:

```
    R : B ↦_{smc-Par α} a
    S : B ↦_{smc-Par α} a
    T ∘_{A op-smc (smc-Par α)} R = T ∘_{A op-smc (smc-Par α)} S

  from prems(3) have RT-ST: R ∘_{A smc-Par α} T = S ∘_{A smc-Par α} T
    unfolding
      op-smc-Comp[OF prems(1) assms(1)]
      op-smc-Comp[OF prems(2) assms(1)]
    by simp
  from smc-Par-is-epic-arr-vsubset[OF assms(1,2) prems(1,2) this]
  have RS: R(|ArrVal|) ⊆_∘ S(|ArrVal|).

  from smc-Par-is-epic-arr-vsubset[OF assms(1,2) prems(2,1) RT-ST[symmetric]]
  have SR: S(|ArrVal|) ⊆_∘ R(|ArrVal|).

  from prems show R = S
    by (intro arr-Par-eqI[of α R S])
      (auto simp: RS SR vsubset-antisym elim!: smc-Par-is-arrE)

qed

lemma smc-Par-is-epic-arrD:
  assumes T : A ↦_{epi smc-Par α} B
  shows T : A ↦_{smc-Par α} B and R_∘ (T(|ArrVal|)) = B
proof–

  from assms show T: T : A ↦_{smc-Par α} B
    unfolding is-epic-arr-def by (auto simp: op-smc-is-arr)

  interpret T: arr-Par α T
    rewrites [simp]: T(|ArrDom|) = A and [simp]: T(|ArrCod|) = B
    using T by (auto elim: smc-Par-is-arrE)

  interpret semicategory α ‹smc-Par α› by (rule T.semicategory-smc-Par)

  show R_∘ (T(|ArrVal|)) = B
  proof(intro vsubset-antisym vsubsetI)
    from T.arr-Rel-ArrVal-vrange show y ∈_∘ R_∘ (T(|ArrVal|)) ⟹ y ∈_∘ B for y
      by auto
    fix b assume [intro]: b ∈_∘ B show b ∈_∘ R_∘ (T(|ArrVal|))
    proof(rule ccontr)
      assume prems: b ∉_∘ R_∘ (T(|ArrVal|))
      define R where R = [set {⟨b, 0⟩}, B, set {0, 1}]_∘
      define S where S = [set {⟨b, 1⟩}, B, set {0, 1}]_∘
      have R: R : B ↦_{smc-Par α} set {0, 1}
        unfolding R-def
      proof
        (
          intro smc-Par-is-arrI T.arr-Par-vfsequenceI,
          unfold arr-Rel-components
        )
        from T.Axiom-of-Infinity vone-in-omega show set {0, 1} ∈_∘ Vset α
          by blast
      qed (auto simp: T.arr-Rel-ArrCod-in-Vset)
      have S: S : B ↦_{smc-Par α} set {0, 1}
        unfolding S-def
      proof
        (
```

```
        intro smc-Par-is-arrI T.arr-Par-vfsequenceI,
        unfold arr-Rel-components
      )
      from T.Axiom-of-Infinity vone-in-omega show set {0, 1} ∈∘ Vset α
        by blast
    qed (auto simp: T.arr-Rel-ArrCod-in-Vset)
    from prems have R(|ArrVal|) ∘∘ T(|ArrVal|) = 0
      unfolding R-def arr-Rel-components
      by (auto intro!: vsubset-antisym vsubsetI)
    moreover from prems have S(|ArrVal|) ∘∘ T(|ArrVal|) = 0
      unfolding S-def arr-Rel-components
      by (auto intro!: vsubset-antisym vsubsetI)
    ultimately have R ∘A smc-Par α T = S ∘A smc-Par α T
      unfolding smc-Par-Comp-app[OF R T] smc-Par-Comp-app[OF S T]
      by (simp add: R-def S-def arr-Rel-components comp-Rel-def)
    from is-epic-arrD(2)[OF assms R S this] show False
      unfolding R-def S-def by simp
  qed
 qed

qed

end

lemma smc-Par-is-epic-arr:
 T : A ↦epi smc-Par α B ⟷ T : A ↦smc-Par α B ∧ ℛ∘ (T(|ArrVal|)) = B
 by (intro iffI) (simp-all add: smc-Par-is-epic-arrD smc-Par-is-epic-arrI)
```

4.12.4 Terminal object, initial object and null object

```
lemma (in 𝒵) smc-Par-obj-terminal: obj-terminal (smc-Par α) A ⟷ A = 0
proof–

  interpret semicategory α ‹smc-Par α› by (rule semicategory-smc-Par)

  have (∀ A∈∘Vset α. ∃!T. T : A ↦smc-Par α B) ⟷ B = 0 for B
  proof(intro iffI allI ballI)

    assume prems[rule-format]: ∀ A∈∘Vset α. ∃!T. T : A ↦smc-Par α B

    then obtain T where T : 0 ↦smc-Par α B by (meson vempty-is-zet)
    then have [simp]: B ∈∘ Vset α by (fastforce simp: smc-Par-components(1))

    show B = 0
    proof(rule ccontr)
      assume B ≠ 0
      then obtain b where b ∈∘ B using trad-foundation by auto
      have [set {⟨0, b⟩}, set {0}, B]∘ : set {0} ↦smc-Par α B
        by (intro smc-Par-is-arrI arr-Par-vfsequenceI, unfold arr-Rel-components)
          (auto simp: ‹b ∈∘ B› vsubset-vsingleton-leftI)
      moreover have [0, set {0}, B]∘ : set {0} ↦smc-Par α B
        by (intro smc-Par-is-arrI arr-Par-vfsequenceI, unfold arr-Rel-components)
          (auto simp: ‹b ∈∘ B› vsubset-vsingleton-leftI)
      moreover have [set {⟨0, b⟩}, set {0}, B]∘ ≠ [0, set {0}, B]∘ by simp
      ultimately show False
        by (metis prems smc-is-arrE smc-Par-components(1))
    qed
```

```
  next

    fix A assume [simp]: B = 0 A ∈ₒ Vset α
    show ∃!T. T : A ↦_smc-Par α B
    proof(intro ex1I [of - ‹[0, A, 0]ₒ›])
      show zAz: [0, A, 0]ₒ : A ↦_smc-Par α B
        by
          (
            intro smc-Par-is-arrI arr-Par-vfsequenceI,
            unfold arr-Rel-components
          )
          simp-all
      show T = [0, A, 0]ₒ if T : A ↦_smc-Par α B for T
      proof(rule arr-Par-eqI[of α], unfold arr-Rel-components)
        interpret arr-Par α T using that by (simp add: smc-Par-is-arrD(1))
        from zAz show arr-Par α [0, A, 0]ₒ by (auto elim: smc-Par-is-arrE)
        from arr-Par-axioms that show T(|ArrVal|) = 0
          by (clarsimp simp: vsv.vsv-vrange-vempty smc-Par-is-arrD(3))
      qed (use that in ‹auto dest: smc-Par-is-arrD›)
    qed

  qed

  then show ?thesis
    apply(intro iffI obj-terminalI)
    subgoal by (metis smc-is-arrD(2) obj-terminalE)
    subgoal by blast
    subgoal by (metis smc-Par-components(1))
    done

qed

lemma (in 𝒵) smc-Par-obj-initial: obj-initial (smc-Par α) A ⟷ A = 0
proof–

  interpret Par: semicategory α ‹smc-Par α› by (rule semicategory-smc-Par)

  have (∀ B∈ₒ Vset α. ∃!T. T : A ↦_smc-Par α B) ⟷ (A = 0) for A
  proof(intro iffI allI ballI)

    assume prems[rule-format]: ∀ B∈ₒ Vset α. ∃!T. T : A ↦_smc-Par α B

    then obtain T where T : A ↦_smc-Par α 0
      by (meson vempty-is-zet)
    then have [simp]: A ∈ₒ Vset α by (fastforce simp: smc-Par-components(1))

    show A = 0
    proof(rule ccontr)
      assume A ≠ 0
      then obtain a where a ∈ₒ A using trad-foundation by auto
      have [set {⟨a, 0⟩}, A, set {0}]ₒ : A ↦_smc-Par α set {0}
        by (intro smc-Par-is-arrI arr-Par-vfsequenceI, unfold arr-Rel-components)
          (auto simp: ‹a ∈ₒ A› vsubset-vsingleton-leftI)
      moreover have [0, A, set {0}]ₒ : A ↦_smc-Par α set {0}
        by (intro smc-Par-is-arrI arr-Par-vfsequenceI, unfold arr-Rel-components)
          (auto simp: ‹a ∈ₒ A› vsubset-vsingleton-leftI)
      moreover have [set {⟨a, 0⟩}, A, set {0}]ₒ ≠ [0, A, set {0}]ₒ by simp
      ultimately show False
```

```
      by (metis prems Par.smc-is-arrE smc-Par-components(1))
    qed

  next

    fix B assume prems[simp]: A = 0 B ∈∘ Vset α

    show ∃!T. T : A ↦_smc-Par α B
    proof(intro ex1I[of - ‹[0, 0, B]∘›])
      show zzB: [0, 0, B]∘ : A ↦_smc-Par α B
        by
          (
            intro smc-Par-is-arrI arr-Par-vfsequenceI,
            unfold arr-Rel-components
          )
          simp-all
      show T = [0, 0, B]∘ if T : A ↦_smc-Par α B for T
      proof(rule arr-Par-eqI[of α], unfold arr-Rel-components)
        interpret arr-Par α T using that by (simp add: smc-Par-is-arrD(1))
        show arr-Par α T by (rule arr-Par-axioms)
        from zzB show arr-Par α [0, 0, B]∘ by (auto elim: smc-Par-is-arrE)
        from arr-Par-axioms that show T(|ArrVal|) = 0
          by (elim smc-Par-is-arrE arr-ParE)
            (
              auto
                intro: ArrVal.vsv-vrange-vempty
                simp: ArrVal.vdomain-vrange-is-vempty
            )
      qed (use that in ‹auto dest: smc-Par-is-arrD›)
    qed
  qed

  then show ?thesis
    unfolding obj-initial-def
    apply(intro iffI obj-terminalI, elim obj-terminalE, unfold smc-op-simps)
    subgoal by (metis smc-Par-components(1))
    subgoal by (simp add: smc-Par-components(1))
    subgoal by (metis smc-Par-components(1))
    done

qed

lemma (in 𝒵) smc-Par-obj-terminal-obj-initial:
  obj-initial (smc-Par α) A ⟷ obj-terminal (smc-Par α) A
  unfolding smc-Par-obj-initial smc-Par-obj-terminal by simp

lemma (in 𝒵) smc-Par-obj-null: obj-null (smc-Par α) A ⟷ A = 0
  unfolding obj-null-def smc-Par-obj-terminal smc-Par-obj-initial by simp
```

4.12.5 Zero arrow

```
lemma (in 𝒵) smc-Par-is-zero-arr:
  assumes A ∈∘ smc-Par α(|Obj|) and B ∈∘ smc-Par α(|Obj|)
  shows T : A ↦_0 smc-Par α B ⟷ T = [0, A, B]∘
proof(intro HOL.ext iffI)
  interpret Par: semicategory α ‹smc-Par α› by (rule semicategory-smc-Par)
  fix T A B assume T : A ↦_0 smc-Par α B
  with smc-Par-is-arrD(1) obtain R S
```

```
    where T-def: T = R ∘_A smc-Par α S
     and S: S : A ↦_smc-Par α 0
     and R: R : 0 ↦_smc-Par α B
    by (auto simp: arr-Par-def 𝒵.smc-Par-obj-initial obj-null-def)
  interpret S: arr-Par α S
    rewrites [simp]: S(|ArrDom|) = A and [simp]: S(|ArrCod|) = 0
    using S smc-Par-is-arrD by auto
  interpret R: arr-Par α R
    rewrites [simp]: R(|ArrDom|) = 0 and [simp]: R(|ArrCod|) = B
    using R smc-Par-is-arrD by auto
  have S-def: S = [0, A, 0]∘
    by
      (
        rule arr-Rel-eqI[of α],
        unfold arr-Rel-components,
        insert S.arr-Rel-ArrVal-vrange S.ArrVal.vbrelation-vintersection-vrange
      )
      (
        auto simp:
        S.arr-Rel-axioms
        S.arr-Rel-ArrDom-in-Vset
        arr-Rel-vfsequenceI vbrelationI
      )
  show T = [0, A, B]∘
    unfolding T-def smc-Par-Comp-app[OF R S]
    by (rule arr-Rel-eqI[of α], unfold comp-Rel-components)
      (
        auto simp:
        𝒵-axioms
        S-def
        R.arr-Rel-axioms
        S.arr-Rel-axioms
        arr-Rel-comp-Rel
        arr-Rel-components
        R.arr-Rel-ArrCod-in-Vset
        S.arr-Rel-ArrDom-in-Vset
        𝒵.arr-Rel-vfsequenceI
        vbrelation-vempty
      )
next
  fix T assume prems: T = [0, A, B]∘
  let ?S = ‹[0, A, 0]∘› and ?R = ‹[0, 0, B]∘›
  have S: arr-Par α ?S and R: arr-Par α ?R
    by (all‹intro arr-Par-vfsequenceI›)
      (simp-all add: assms[unfolded smc-Par-components])
  have SA0: ?S : A ↦_smc-Par α 0
    by (intro smc-Par-is-arrI) (simp-all add: S arr-Rel-components)
  moreover have R0B: ?R : 0 ↦_smc-Par α B
    by (intro smc-Par-is-arrI) (simp-all add: R arr-Rel-components)
  moreover have T = ?R ∘_A smc-Par α ?S
    unfolding smc-Par-Comp-app[OF R0B SA0]
  proof
    (
      rule arr-Par-eqI[of α],
      unfold comp-Rel-components arr-Rel-components prems
    )
    show arr-Par α [0, A, B]∘
      unfolding prems
```

```
    by (intro arr-Par-vfsequenceI)
      (auto simp: assms[unfolded smc-Par-components])
  qed (use R S in ‹auto simp: smc-Par-cs-intros›)
  ultimately show T : A ↦₀_{smc-Par α} B
    by (simp add: is-zero-arrI smc-Par-obj-null)
qed
```

4.13 *Set* as a semicategory

4.13.1 Background

The methodology chosen for the exposition of *Set* as a semicategory is analogous to the one used in the previous chapter for the exposition of *Set* as a digraph.

named-theorems *smc-Set-cs-simps*
named-theorems *smc-Set-cs-intros*

lemmas (**in** *arr-Set*) [*smc-Set-cs-simps*] =
dg-Rel-shared-cs-simps

lemmas (**in** *arr-Set*) [*smc-cs-intros*, *smc-Set-cs-intros*] =
arr-Set-axioms'

lemmas [*smc-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 [*smc-Set-cs-intros*] =
dg-Rel-shared-cs-intros
arr-Set-comp-Set

4.13.2 *Set* as a semicategory

Definition and elementary properties

definition *smc-Set* :: $V \Rightarrow V$
where *smc-Set* α =
[
Vset α,
set $\{T.\ \textit{arr-Set}\ \alpha\ T\}$,
$(\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$,
$(\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$,
$(\lambda ST \in_\circ \textit{composable-arrs}\ (\textit{dg-Set}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$
]$_\circ$

Components.

lemma *smc-Set-components*:
shows *smc-Set* $\alpha(\!|\textit{Obj}|\!)$ = *Vset* α
and *smc-Set* $\alpha(\!|\textit{Arr}|\!) = \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}$
and *smc-Set* $\alpha(\!|\textit{Dom}|\!) = (\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrDom}|\!))$
and *smc-Set* $\alpha(\!|\textit{Cod}|\!) = (\lambda T \in_\circ \textit{set}\ \{T.\ \textit{arr-Set}\ \alpha\ T\}.\ T(\!|\textit{ArrCod}|\!))$
and *smc-Set* $\alpha(\!|\textit{Comp}|\!) = (\lambda ST \in_\circ \textit{composable-arrs}\ (\textit{dg-Set}\ \alpha).\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbb{N}}|\!))$
unfolding *smc-Set-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-smc-Set*: *smc-dg* (*smc-Set* α) = *dg-Set* α
proof(*rule vsv-eqI*)
have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-Set* α)) = $4_{\mathbb{N}}$
unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-Set* α) = $4_{\mathbb{N}}$
unfolding *dg-Set-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{D}_\circ$ (*smc-dg* (*smc-Set* α)) = $\mathcal{D}_\circ$ (*dg-Set* α)
unfolding *dom-lhs dom-rhs* **by** *simp*
show $a \in_\circ \mathcal{D}_\circ$ (*smc-dg* (*smc-Set* α)) $\Longrightarrow$ *smc-dg* (*smc-Set* α)$(\!|a|\!)$ = *dg-Set* $\alpha(\!|a|\!)$

for *a*
by
(
unfold dom-lhs,
elim-in-numeral,
unfold smc-dg-def dg-field-simps smc-Set-def dg-Set-def
)
(*auto simp*: *nat-omega-simps*)
qed (*auto simp*: *smc-dg-def dg-Set-def*)

lemmas-with [*folded smc-dg-smc-Set*, *unfolded slicing-simps*]:
smc-Set-Obj-iff = *dg-Set-Obj-iff*
and *smc-Set-Arr-iff*[*smc-Set-cs-simps*] = *dg-Set-Arr-iff*
and *smc-Set-Dom-vsv*[*smc-Set-cs-intros*] = *dg-Set-Dom-vsv*
and *smc-Set-Dom-vdomain*[*smc-Set-cs-simps*] = *dg-Set-Dom-vdomain*
and *smc-Set-Dom-vrange* = *dg-Set-Dom-vrange*
and *smc-Set-Dom-app*[*smc-Set-cs-simps*] = *dg-Set-Dom-app*
and *smc-Set-Cod-vsv*[*smc-Set-cs-intros*] = *dg-Set-Cod-vsv*
and *smc-Set-Cod-vdomain*[*smc-Set-cs-simps*] = *dg-Set-Cod-vdomain*
and *smc-Set-Cod-vrange* = *dg-Set-Cod-vrange*
and *smc-Set-Cod-app*[*smc-Set-cs-simps*] = *dg-Set-Cod-app*
and *smc-Set-is-arrI* = *dg-Set-is-arrI*
and *smc-Set-is-arrD* = *dg-Set-is-arrD*
and *smc-Set-is-arrE* = *dg-Set-is-arrE*
and *smc-Set-ArrVal-vdomain*[*smc-Set-cs-simps*] = *dg-Set-ArrVal-vdomain*
and *smc-Set-ArrVal-app-vrange*[*smc-Set-cs-intros*] = *dg-Set-ArrVal-app-vrange*

lemmas [*smc-cs-simps*] = *smc-Set-is-arrD*(2,3)

lemmas-with (in $\mathcal{Z}$**)** [*folded smc-dg-smc-Set*, *unfolded slicing-simps*]:
smc-Set-Hom-vifunion-in-Vset = *dg-Set-Hom-vifunion-in-Vset*
and *smc-Set-incl-Set-is-arr* = *dg-Set-incl-Set-is-arr*

lemmas [*smc-Set-cs-intros*] =
smc-Set-is-arrI

lemma (in $\mathcal{Z}$**)** *smc-Set-incl-Set-is-arr'*[*smc-cs-intros*, *smc-Set-cs-intros*]:
assumes $A \in_{\circ}$ *smc-Set* $\alpha(\!|Obj|\!)$
and $B \in_{\circ}$ *smc-Set* $\alpha(\!|Obj|\!)$
and $A \subseteq_{\circ} B$
and $A' = A$
and $B' = B$
and $\mathfrak{C}' =$ *smc-Set* α
shows *incl-Set* A $B : A' \mapsto_{\mathfrak{C}'} B'$
using *assms*(1–3) **unfolding** *assms*(4–6) **by** (*rule smc-Set-incl-Set-is-arr*)

lemmas [*smc-Set-cs-intros*] = $\mathcal{Z}$*.smc-Set-incl-Set-is-arr'*

Composable arrows

lemma *smc-Set-composable-arrs-dg-Set*:
composable-arrs (*dg-Set* α) = *composable-arrs* (*smc-Set* α)
unfolding *composable-arrs-def smc-dg-smc-Set*[*symmetric*] *slicing-simps* **by** *simp*

lemma *smc-Set-Comp*:
smc-Set $\alpha(\!|Comp|\!)$ =
VLambda (*composable-arrs* (*smc-Set* α)) ($\lambda ST.\ ST(\!|0|\!) \circ_{Rel} ST(\!|1_{\mathbf{N}}|\!)$)
unfolding *smc-Set-components smc-Set-composable-arrs-dg-Set* **..**

Composition

lemma *smc-Set-Comp-app*[*smc-Set-cs-simps*]:
assumes $S : b \mapsto_{smc\text{-}Set\ \alpha} c$ **and** $T : a \mapsto_{smc\text{-}Set\ \alpha} b$
shows $S \circ_{A\ smc\text{-}Set\ \alpha} T = S \circ_{Set} T$
proof–
from *assms* **have** $[S, T]_\circ \in_\circ composable\text{-}arrs\ (smc\text{-}Set\ \alpha)$
by (*auto simp*: *smc-cs-intros*)
then show $S \circ_{A\ smc\text{-}Set\ \alpha} T = S \circ_{Set} T$
unfolding *smc-Set-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-Set-Comp-vdomain*: $\mathcal{D}_\circ\ (smc\text{-}Set\ \alpha(\!|Comp|\!)) = composable\text{-}arrs\ (smc\text{-}Set\ \alpha)$
unfolding *smc-Set-Comp* **by** *simp*

lemma (**in** $\mathcal{Z}$) *smc-Set-Comp-vrange*:
$\mathcal{R}_\circ\ (smc\text{-}Set\ \alpha(\!|Comp|\!)) \subseteq_\circ set\ \{T.\ arr\text{-}Set\ \alpha\ T\}$
proof(*rule vsubsetI*)
interpret *digraph* α ‹*smc-dg* (*smc-Set* α)›
unfolding *smc-dg-smc-Set* **by** (*simp add*: *digraph-dg-Set*)
fix R **assume** $R \in_\circ \mathcal{R}_\circ\ (smc\text{-}Set\ \alpha(\!|Comp|\!))$
then obtain ST
where *R-def*: $R = smc\text{-}Set\ \alpha(\!|Comp|\!)(\!|ST|\!)$
and $ST \in_\circ \mathcal{D}_\circ\ (smc\text{-}Set\ \alpha(\!|Comp|\!))$
unfolding *smc-Set-components* **by** (*blast dest*: *rel-VLambda.vrange-atD*)
then obtain $S\ T\ a\ b\ c$
where $ST = [S, T]_\circ$
and *S*: $S : b \mapsto_{smc\text{-}Set\ \alpha} c$
and *T*: $T : a \mapsto_{smc\text{-}Set\ \alpha} b$
by (*auto simp*: *smc-Set-Comp-vdomain*)
with *R-def* **have** *R-def'*: $R = S \circ_{A\ smc\text{-}Set\ \alpha} T$ **by** *simp*
interpret *S*: *arr-Set* α S + *T*: *arr-Set* α T
rewrites [*simp*]: $S(\!|ArrDom|\!) = b$
and [*simp*]: $S(\!|ArrCod|\!) = c$
and [*simp*]: $T(\!|ArrDom|\!) = a$
and [*simp*]: $T(\!|ArrCod|\!) = b$
using $S\ T$ **by** (*auto elim*!: *smc-Set-is-arrD*)
have $\mathcal{R}_\circ\ (T(\!|ArrVal|\!)) \subseteq_\circ \mathcal{D}_\circ\ (S(\!|ArrVal|\!))$
proof(*intro vsubsetI*)
fix y **assume** *prems*: $y \in_\circ \mathcal{R}_\circ\ (T(\!|ArrVal|\!))$
with *T.ArrVal.vrange-atD* **obtain** x
where *y-def*: $y = T(\!|ArrVal|\!)(\!|x|\!)$ **and** *x*: $x \in_\circ \mathcal{D}_\circ\ (T(\!|ArrVal|\!))$
by *metis*
from *prems x T.arr-Set-ArrVal-vrange* **show** $y \in_\circ \mathcal{D}_\circ\ (S(\!|ArrVal|\!))$
unfolding *y-def* **by** (*auto simp*: *smc-Set-cs-simps*)
qed
with *S.arr-Set-axioms T.arr-Set-axioms* **have** *arr-Set* α $(S \circ_{Set} T)$
by (*simp add*: *arr-Set-comp-Set*)
from *this* **show** $R \in_\circ set\ \{T.\ arr\text{-}Set\ \alpha\ T\}$
unfolding *R-def' smc-Set-Comp-app*[*OF S T*] **by** *simp*
qed

lemma *smc-Set-composable-vrange-vdomain*[*smc-Set-cs-intros*]:
assumes $g : b \mapsto_{smc\text{-}Set\ \alpha} c$ **and** $f : a \mapsto_{smc\text{-}Set\ \alpha} b$
shows $\mathcal{R}_\circ\ (f(\!|ArrVal|\!)) \subseteq_\circ \mathcal{D}_\circ\ (g(\!|ArrVal|\!))$
proof(*intro vsubsetI*)
from *assms* **have** *g*: *arr-Set* α g **and** *f*: *arr-Set* α f
by (*auto simp*: *smc-Set-is-arrD*)

```
  fix y assume y ∈∘ ℛ∘ (f(|ArrVal|))
  with assms f have y ∈∘ b by (force simp: smc-Set-is-arrD(3))
  with assms g show y ∈∘ 𝒟∘ (g(|ArrVal|))
    by (simp add: smc-Set-is-arrD(2) arr-SetD(5))
qed

lemma smc-Set-Comp-ArrVal[smc-cs-simps]:
  assumes S : y ↦_smc-Set α z and T : x ↦_smc-Set α y and a ∈∘ x
  shows (S ∘_A smc-Set α T)(|ArrVal|)(|a|) = S(|ArrVal|)(|T(|ArrVal|)(|a|)|)
proof–
  interpret S: arr-Set α S + T: arr-Set α T
    using assms by (auto simp: smc-Set-is-arrD)
  have Ta: T(|ArrVal|)(|a|) ∈∘ y
  proof–
    from assms have a ∈∘ T(|ArrDom|) by (auto simp: smc-Set-is-arrD)
    with assms T.arr-Set-ArrVal-vrange show ?thesis
      by
        (
          simp add:
            T.ArrVal.vsv-vimageI2 vsubset-iff smc-Set-is-arrD smc-Set-cs-simps
        )
  qed
  from Ta assms S.arr-Set-axioms T.arr-Set-axioms show ?thesis
    by ((cs-concl-step smc-Set-cs-simps)+, intro arr-Set-comp-Set-ArrVal-app[of α])
      (simp-all add: smc-Set-is-arrD smc-Set-cs-simps)
qed
```

Set is a semicategory

```
lemma (in 𝒵) semicategory-smc-Set: semicategory α (smc-Set α)
proof(rule semicategoryI, unfold smc-dg-smc-Set)

  interpret wide-subdigraph α ‹dg-Set α› ‹dg-Par α›
    by (rule wide-subdigraph-dg-Set-dg-Par)
  interpret smc-Par: semicategory α ‹smc-Par α› by (rule semicategory-smc-Par)

  show vfsequence (smc-Set α) unfolding smc-Set-def by simp
  show vcard (smc-Set α) = 5ℕ
    unfolding smc-Set-def by (simp add: nat-omega-simps)

  show (gf ∈∘ 𝒟∘ (smc-Set α(|Comp|))) ⟷
    (∃ g f b c a. gf = [g, f]∘ ∧ g : b ↦_smc-Set α c ∧ f : a ↦_smc-Set α b)
    for gf
    unfolding smc-Set-Comp-vdomain by (auto intro: composable-arrsI)

  show [intro]: g ∘_A smc-Set α f : a ↦_smc-Set α c
    if g : b ↦_smc-Set α c f : a ↦_smc-Set α b for g b c f a
  proof–
    from that have g: arr-Set α g and f: arr-Set α f
      by (auto simp: smc-Set-is-arrD)
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: smc-cs-simps smc-Set-cs-simps
            cs-intro: smc-Set-cs-intros
        )
  qed
```

```
  show h ∘_A smc-Set α g ∘_A smc-Set α f = h ∘_A smc-Set α (g ∘_A smc-Set α f)
    if h : c ↦_smc-Set α d
      and g : b ↦_smc-Set α c
      and f : a ↦_smc-Set α b
    for h c d g b f a
  proof–
    from that have arr-Set α h arr-Set α g arr-Set α f
      by (auto simp: smc-Set-is-arrD)
    with that show ?thesis
      by
        (
          cs-concl cs-shallow
            cs-simp: smc-cs-simps smc-Set-cs-simps
            cs-intro: smc-Set-cs-intros
        )
  qed

qed (auto simp: digraph-dg-Set smc-Set-components)
```

Set is a wide subsemicategory of *Par*

```
lemma (in Z) wide-subsemicategory-smc-Set-smc-Par:
  smc-Set α ⊆_SMC.wide α smc-Par α
proof–
  interpret Par: semicategory α ‹smc-Par α› by (rule semicategory-smc-Par)
  interpret Set: semicategory α ‹smc-Set α› by (rule semicategory-smc-Set)
  show ?thesis
  proof
    (
      intro wide-subsemicategoryI subsemicategoryI,
      unfold smc-dg-smc-Par smc-dg-smc-Set
    )
    from wide-subdigraph-dg-Set-dg-Par show wsd:
      dg-Set α ⊆_DG α dg-Par α
      dg-Set α ⊆_DG.wide α dg-Par α
      by auto
    interpret wide-subdigraph α ‹dg-Set α› ‹dg-Par α› by (rule wsd(2))
    show g ∘_A smc-Set α f = g ∘_A smc-Par α f
      if g : b ↦_smc-Set α c and f : a ↦_smc-Set α b for g b c f a
    proof–
      from that have g : b ↦_dg-Set α c and f : a ↦_dg-Set α b
        by
          (
            cs-concl cs-shallow
              cs-simp: smc-dg-smc-Set[symmetric] cs-intro: slicing-intros
          )+
      then have g : b ↦_dg-Par α c and f : a ↦_dg-Par α b
        by (cs-concl cs-shallow cs-intro: dg-sub-fw-cs-intros)+
      then have g : b ↦_smc-Par α c and f : a ↦_smc-Par α b
        unfolding smc-dg-smc-Par[symmetric] slicing-simps by simp-all
      from that this show g ∘_A smc-Set α f = g ∘_A smc-Par α f
        by (cs-concl cs-shallow cs-simp: smc-Set-cs-simps smc-Par-cs-simps)
    qed
  qed (auto simp: smc-cs-intros)
qed
```

4.13.3 Monic arrow and epic arrow

lemma *smc-Set-is-monic-arrI*:
— See Chapter I-5 in [39]).
assumes $T : A \mapsto_{smc\text{-}Set\ \alpha} B$ **and** $v11\ (T(\!|ArrVal|\!))$ **and** $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = A$
shows $T : A \mapsto_{mon\,smc\text{-}Set\ \alpha} B$
proof(*rule is-monic-arrI*)

interpret T: *arr-Set* α T **by** (*intro smc-Set-is-arrD*[*OF assms*(1)])+
interpret *wide-subsemicategory* α ‹*smc-Set* α› ‹*smc-Par* α›
by (*rule T.wide-subsemicategory-smc-Set-smc-Par*)
interpret *v11* ‹$T(\!|ArrVal|\!)$› **by** (*rule assms*(2))

show T: $T : A \mapsto_{smc\text{-}Set\ \alpha} B$ **by** (*rule assms*(1))
fix $S\ R\ A'$
assume S: $S : A' \mapsto_{smc\text{-}Set\ \alpha} A$
and R: $R : A' \mapsto_{smc\text{-}Set\ \alpha} A$
and *TS-TR*: $T \circ_{A\,smc\text{-}Set\ \alpha} S = T \circ_{A\,smc\text{-}Set\ \alpha} R$
from *assms*(3) T **have** $T : A \mapsto_{mon\,smc\text{-}Par\ \alpha} B$
by (*intro smc-Par-is-monic-arrI*)
(*auto simp*: *v11-axioms dest*: *subsmc-is-arrD*)
moreover from S *subsemicategory-axioms* **have** $S : A' \mapsto_{smc\text{-}Par\ \alpha} A$
by (*cs-concl* **cs-shallow cs-intro**: *smc-sub-fw-cs-intros*)
moreover from R *subsemicategory-axioms* **have** $R : A' \mapsto_{smc\text{-}Par\ \alpha} A$
by (*cs-concl* **cs-shallow cs-intro**: *smc-sub-fw-cs-intros*)
moreover from $T\ S\ R$ *TS-TR subsemicategory-axioms* **have**
$T \circ_{A\,smc\text{-}Par\ \alpha} S = T \circ_{A\,smc\text{-}Par\ \alpha} R$
by (*auto simp*: *smc-sub-bw-cs-simps*)
ultimately show $S = R$ **by** (*rule is-monic-arrD*(2))

qed

lemma *smc-Set-is-monic-arrD*:
assumes $T : A \mapsto_{mon\,smc\text{-}Set\ \alpha} B$
shows $T : A \mapsto_{smc\text{-}Set\ \alpha} B$ **and** $v11\ (T(\!|ArrVal|\!))$ **and** $\mathcal{D}_\circ\ (T(\!|ArrVal|\!)) = A$
proof–

from *assms* **show** T: $T : A \mapsto_{smc\text{-}Set\ \alpha} B$ **by** *auto*
interpret T: *arr-Set* α T
rewrites [*simp*]: $T(\!|ArrDom|\!) = A$ **and** [*simp*]: $T(\!|ArrCod|\!) = B$
by (*intro smc-Set-is-arrD*[*OF T*])+

interpret *wide-subdigraph* α ‹*dg-Set* α› ‹*dg-Par* α›
by (*rule T.wide-subdigraph-dg-Set-dg-Par*)
interpret *Par*: *semicategory* α ‹*smc-Par* α› **by** (*rule T.semicategory-smc-Par*)

show $v11\ (T(\!|ArrVal|\!))$
proof(*rule v11I*)

show *vsv* $((T(\!|ArrVal|\!))^{-1}{}_\circ)$
proof(*rule vsvI*)

fix $a\ b\ c$ **assume** $\langle a, b\rangle \in_\circ (T(\!|ArrVal|\!))^{-1}{}_\circ$ **and** $\langle a, c\rangle \in_\circ (T(\!|ArrVal|\!))^{-1}{}_\circ$

then have *bar*: $\langle b, a\rangle \in_\circ T(\!|ArrVal|\!)$ **and** *car*: $\langle c, a\rangle \in_\circ T(\!|ArrVal|\!)$
by *auto*
with *T.arr-Set-ArrVal-vdomain* **have** [*intro*]: $b \in_\circ A$ $c \in_\circ A$ **by** *blast*+

```
define R where R = [set {⟨0, b⟩}, set {0}, A]∘
define S where S = [set {⟨0, c⟩}, set {0}, A]∘

have R: R : set {0} ↦_{smc-Set α} A
proof(rule smc-Set-is-arrI)
  show arr-Set α R
    unfolding R-def
    by (rule T.arr-Set-vfsequenceI) (auto simp: T.arr-Rel-ArrDom-in-Vset)
qed (simp-all add: R-def arr-Rel-components)
interpret R: arr-Set α R
  rewrites [simp]: R(|ArrDom|) = set {0} and [simp]: R(|ArrCod|) = A
  by (intro smc-Set-is-arrD[OF R])+

have S: S : set {0} ↦_{smc-Set α} A
proof(rule smc-Set-is-arrI)
  show arr-Set α S
    unfolding S-def
    by (rule T.arr-Set-vfsequenceI) (auto simp: T.arr-Rel-ArrDom-in-Vset)
qed (simp-all add: S-def arr-Rel-components)
interpret S: arr-Set α S
  rewrites [simp]: S(|ArrDom|) = set {0} and [simp]: S(|ArrCod|) = A
  by (intro smc-Set-is-arrD[OF S])+

have T ∘_{A smc-Set α} R = [set {⟨0, a⟩}, set {0}, B]∘
  unfolding smc-Set-Comp-app[OF T R]
proof
  (
    rule arr-Set-eqI[of α],
    unfold comp-Rel-components arr-Rel-components
  )
  from R T show arr-Set α (T ∘_{Set} R)
    by (intro arr-Set-comp-Set)
      (auto elim!: smc-Set-is-arrE simp: smc-Set-cs-simps)
  show arr-Set α [set {⟨0, a⟩}, set {0}, B]∘
  proof(rule T.arr-Set-vfsequenceI)
    from T.arr-Rel-ArrVal-vrange bar show R∘ (set {⟨0, a⟩}) ⊆∘ B by auto
  qed (auto simp: T.arr-Rel-ArrCod-in-Vset T.Axiom-of-Powers)
  show T(|ArrVal|) ∘∘ R(|ArrVal|) = set {⟨0, a⟩}
    unfolding R-def arr-Rel-components
  proof(rule vsv-eqI, unfold vdomain-vsingleton)
    from bar show D∘ (T(|ArrVal|) ∘∘ set {⟨0, b⟩}) = set {0} by auto
    with bar show a' ∈∘ D∘ (T(|ArrVal|) ∘∘ set {⟨0, b⟩}) ⟹
      (T(|ArrVal|) ∘∘ set {⟨0, b⟩})(|a'|) = set {⟨0, a⟩}(|a'|)
      for a'
      by auto
  qed (auto intro: vsv-vcomp)
qed (simp-all add: R-def arr-Rel-components)
moreover have T ∘_{A smc-Set α} S = [set {⟨0, a⟩}, set {0}, B]∘
  unfolding smc-Set-Comp-app[OF T S]
proof
  (
    rule arr-Set-eqI[of α],
    unfold comp-Rel-components arr-Rel-components
  )
  from T S show arr-Set α (T ∘_{Set} S)
    by (intro arr-Set-comp-Set)
      (
        auto simp:
```

```
        T.arr-Set-axioms
        smc-Set-is-arrD
        S.arr-Set-ArrVal-vrange
        smc-Set-cs-simps
      )
    show arr-Set α [set {⟨0, a⟩}, set {0}, B]∘
    proof(rule T.arr-Set-vfsequenceI)
      from T.arr-Rel-ArrVal-vrange bar show ℛ∘ (set {⟨0, a⟩}) ⊆∘ B by auto
    qed (auto simp: T.arr-Rel-ArrCod-in-Vset T.Axiom-of-Powers)
    show T(|ArrVal|) ∘∘ S(|ArrVal|) = set {⟨0, a⟩}
      unfolding S-def arr-Rel-components
    proof(rule vsv-eqI, unfold vdomain-vsingleton)
      from car show 𝒟∘ (T(|ArrVal|) ∘∘ set {⟨0, c⟩}) = set {0} by auto
      with car show a' ∈∘ 𝒟∘ (T(|ArrVal|) ∘∘ set {⟨0, c⟩}) ⟹
        (T(|ArrVal|) ∘∘ set {⟨0, c⟩})(|a'|) = set {⟨0, a⟩}(|a'|)
        for a'
        by auto
    qed (auto intro: vsv-vcomp)
  qed (simp-all add: S-def arr-Rel-components)
  ultimately have T ∘A smc-Set α R = T ∘A smc-Set α S by simp
  from R S assms this have R = S by clarsimp
  then have R(|ArrVal|) = S(|ArrVal|) by simp
  then show b = c unfolding R-def S-def arr-Rel-components by simp
qed clarsimp

qed auto

show 𝒟∘ (T(|ArrVal|)) = A by (simp add: smc-Set-cs-simps)

qed

lemma smc-Set-is-monic-arr:
  T : A ↦mon smc-Set α B ⟷
    T : A ↦smc-Set α B ∧ v11 (T(|ArrVal|)) ∧ 𝒟∘ (T(|ArrVal|)) = A
  by (rule iffI) (auto simp: smc-Set-is-monic-arrD smc-Set-is-monic-arrI)
```

An epic arrow in *Set* is a total surjective function (see Chapter I-5 in [39]).

```
lemma smc-Set-is-epic-arrI:
  assumes T : A ↦smc-Set α B and ℛ∘ (T(|ArrVal|)) = B
  shows T : A ↦epi smc-Set α B
proof–

  interpret T: arr-Set α T
    rewrites [simp]: T(|ArrDom|) = A and [simp]: T(|ArrCod|) = B
    by (intro smc-Set-is-arrD[OF assms(1)])+

  interpret wide-subsemicategory α ‹smc-Set α› ‹smc-Par α›
    by (rule T.wide-subsemicategory-smc-Set-smc-Par)
  have epi-T: T : A ↦epi smc-Par α B
    using assms by (meson smc-Par-is-epic-arr subsmc-is-arrD)

  show ?thesis
  proof(rule sdg.is-epic-arrI)
    show T: T : A ↦smc-Set α B by (rule assms(1))
    fix f g a
    assume prems:
      f : B ↦smc-Set α a
      g : B ↦smc-Set α a
```

```
    f ∘_A smc-Set α T = g ∘_A smc-Set α T
  from prems(1) subsemicategory-axioms have f : B ↦_smc-Par α a
    by (cs-concl cs-shallow cs-intro: smc-sub-fw-cs-intros)
  moreover from prems(2) subsemicategory-axioms have g : B ↦_smc-Par α a
    by (cs-concl cs-shallow cs-intro: smc-sub-fw-cs-intros)
  moreover from prems T subsemicategory-axioms have
    f ∘_A smc-Par α T = g ∘_A smc-Par α T
    by (auto simp: smc-sub-bw-cs-simps)
  ultimately show f = g
    by (rule dg.is-epic-arrD(2)[OF epi-T])
 qed

qed

lemma smc-Set-is-epic-arrD:
 assumes T : A ↦_epi smc-Set α B
 shows T : A ↦_smc-Set α B and ℛ∘ (T(|ArrVal|)) = B
proof–

 from assms show T: T : A ↦_smc-Set α B by auto
 interpret T: arr-Set α T
  rewrites T(|ArrDom|) = A and T(|ArrCod|) = B
  by (intro smc-Set-is-arrD[OF T])+

 interpret semicategory α ‹smc-Set α› by (rule T.semicategory-smc-Set)

 show ℛ∘ (T(|ArrVal|)) = B
 proof(intro vsubset-antisym vsubsetI)
  fix b assume [intro]: b ∈∘ B
  show b ∈∘ ℛ∘ (T(|ArrVal|))
  proof(rule ccontr)
   assume b: b ∉∘ ℛ∘ (T(|ArrVal|))
   define R
    where R = [vinsert ⟨b, 0⟩ ((B −∘ set {b}) ×∘ set {1}), B, set {0, 1}]∘
   define S where S = [B ×∘ set {1}, B, set {0, 1}]∘
   have R: R : B ↦_smc-Set α set {0, 1}
    unfolding R-def
   proof
    (
     intro smc-Set-is-arrI T.arr-Set-vfsequenceI,
     unfold arr-Rel-components
    )
    from T.Axiom-of-Infinity vone-in-omega show set {0, 1} ∈∘ Vset α
     by blast
   qed (auto simp: T.arr-Rel-ArrCod-in-Vset)
   have S: S : B ↦_smc-Set α set {0, 1}
    unfolding S-def
   proof
    (
     intro smc-Set-is-arrI T.arr-Set-vfsequenceI,
     unfold arr-Rel-components
    )
    from T.Axiom-of-Infinity vone-in-omega show set {0, 1} ∈∘ Vset α
     by blast
   qed (auto simp: T.arr-Rel-ArrCod-in-Vset)
   from b have R(|ArrVal|) ∘∘ T(|ArrVal|) = S(|ArrVal|) ∘∘ T(|ArrVal|)
    unfolding S-def R-def arr-Rel-components
    by (auto intro!: vsubset-antisym vsubsetI)
```

```
    then have R ∘_A smc-Set α T = S ∘_A smc-Set α T
      unfolding smc-Set-Comp-app[OF R T] smc-Set-Comp-app[OF S T]
      by (simp add: R-def S-def arr-Rel-components comp-Rel-def)
    from R S this have R = S by (rule is-epic-arrD(2)[OF assms])
    with zero-neq-one show False unfolding R-def S-def by blast
  qed
qed (use T.arr-Set-ArrVal-vrange in auto)

qed

lemma smc-Set-is-epic-arr:
  T : A ↦_epi smc-Set α B ⟷ T : A ↦_smc-Set α B ∧ ℛ∘ (T(|ArrVal|)) = B
  by (rule iffI) (simp-all add: smc-Set-is-epic-arrD smc-Set-is-epic-arrI)
```

4.13.4 Terminal object, initial object and null object

An object in *Set* is terminal if and only if it is a singleton set (see Chapter I-5 in [39]).

```
lemma (in 𝒵) smc-Set-obj-terminal:
  obj-terminal (smc-Set α) A ⟷ (∃B∈∘Vset α. A = set {B})
proof–

  interpret semicategory α ‹smc-Set α› by (rule semicategory-smc-Set)

  have (∀A∈∘Vset α. ∃!T. T : A ↦_smc-Set α B) ⟷ (∃C∈∘Vset α. B = set {C})
    for B
  proof(intro iffI ballI)

    assume prems[rule-format]: ∀A∈∘Vset α. ∃!T. T : A ↦_smc-Set α B

    then obtain T where T0B: T : 0 ↦_smc-Set α B by (meson vempty-is-zet)
    then have B[simp]: B ∈∘ Vset α by (fastforce simp: smc-Set-components(1))

    show ∃C∈∘Vset α. B = set {C}
    proof(rule ccontr, cases ‹B = 0›)
      case True
      from prems have ∃!T. T : A ↦_smc-Set α 0 if A ∈∘ Vset α for A
        using that unfolding True by simp
      then obtain T where T: T : set {0} ↦_smc-Set α 0
        by (metis Axiom-of-Pairing insert-absorb2 vempty-is-zet)
      interpret T: arr-Set α T
        rewrites T(|ArrDom|) = set {0} and T(|ArrCod|) = 0
        by (intro smc-Set-is-arrD[OF T])+
      from
        T.vdomain-vrange-is-vempty
        T.ArrVal.vdomain-vrange-is-vempty
        T.arr-Set-ArrVal-vrange
      show False
        by (auto simp: smc-Set-cs-simps)
    next
      case False
      assume ¬(∃C∈∘Vset α. B = set {C})
      with B have ∄C. B = set {C} by blast
      with False obtain a b where ab: a ≠ b a ∈∘ B b ∈∘ B
        by (metis V-equalityI vemptyE vintersection-vsingleton vsingletonD)
      have [set {⟨0, a⟩}, set {0}, B]∘ : set {0} ↦_smc-Set α B
        by (intro smc-Set-is-arrI arr-SetI, unfold arr-Rel-components)
          (auto simp: ab(2) nat-omega-simps)
```

```
    moreover from ab have
      [set {⟨0, b⟩}, set {0}, B]∘ : set {0} ↦_{smc-Set α} B
      by (intro smc-Set-is-arrI arr-SetI, unfold arr-Rel-components)
        (auto simp: ab(2) nat-omega-simps)
    moreover with ab have
      [set {⟨0, a⟩}, set {0}, B]∘ ≠ [set {⟨0, b⟩}, set {0}, B]∘
      by simp
    ultimately show False
      by (metis prems smc-is-arrE smc-Set-components(1))
  qed

next

  fix A assume prems: ∃ b∈∘ Vset α. B = set {b} A ∈∘ Vset α
  then obtain b where B-def: B = set {b} and b: b ∈∘ Vset α by blast

  have vconst-on A b = A ×∘ set {b} by (simp add: vconst-on-eq-vtimes)

  show ∃!T. T : A ↦_{smc-Set α} B
    unfolding B-def
  proof(rule ex1I[of - ‹[A ×∘ set {b}, A, set {b}]∘›])

    show [A ×∘ set {b}, A, set {b}]∘ : A ↦_{smc-Set α} set {b}
      using b
      by
        (
          intro smc-Set-is-arrI arr-Set-vfsequenceI,
          unfold arr-Rel-components
        )
        (auto simp: prems(2) vconst-on-eq-vtimes[symmetric])

    fix T assume prems: T : A ↦_{smc-Set α} set {b}

    interpret T: arr-Set α T
      rewrites [simp]: T(|ArrDom|) = A and [simp]: T(|ArrCod|) = set {b}
      by (intro smc-Set-is-arrD[OF prems])+

    have [simp]: T(|ArrVal|) = A ×∘ set {b}
    proof(intro vsubset-antisym vsubsetI)
      fix x assume prems: x ∈∘ T(|ArrVal|)
      with T.vbrelation-axioms app-vdomainI obtain a b′
        where x = ⟨a, b′⟩ and a ∈∘ A
        by (metis T.ArrVal.vbrelation-vinE T.arr-Set-ArrVal-vdomain)
      with prems T.arr-Set-ArrVal-vrange show x ∈∘ A ×∘ set {b} by auto
    next
      fix x assume x ∈∘ A ×∘ set {b}
      then obtain a where x-def: x = ⟨a, b⟩ and a ∈∘ A by clarsimp
      have 𝒟∘ (T(|ArrVal|)) = A by (simp add: smc-Set-cs-simps)
      moreover with
        T.arr-Set-ArrVal-vrange T.ArrVal.vdomain-vrange-is-vempty ‹a ∈∘ A›
      have ℛ∘ (T(|ArrVal|)) = set {b}
        by auto
      ultimately show x ∈∘ T(|ArrVal|)
        using ‹a ∈∘ A›
        unfolding x-def
        by
          (
            metis
```

```
        T.ArrVal.vsv-ex1-app1
        T.ArrVal.vsv-vimageI1
        vimage-vdomain
        vsingletonD
      )
  qed

  show T = [A ×∘ set {b}, A, set {b}]∘
  proof(rule arr-Set-eqI[of α], unfold arr-Rel-components)
    show arr-Set α [A ×∘ set {b}, A, set {b}]∘
      using T.arr-Rel-def T.arr-Set-axioms by auto
  qed (auto simp: T.arr-Set-axioms)

 qed
qed

 then show ?thesis
  apply(intro iffI obj-terminalI)
  subgoal by (metis smc-is-arrD(2) obj-terminalE)
  subgoal by blast
  subgoal by (metis smc-Set-components(1))
  done

qed
```

An object in *Set* is initial if and only if it is the empty set (see Chapter I-5 in [39]).

```
lemma (in Z) smc-Set-obj-initial: obj-initial (smc-Set α) A ⟷ A = 0
proof–

 interpret semicategory α ‹smc-Set α› by (rule semicategory-smc-Set)

 have (∀ B∈∘Vset α. ∃!T. T : A ↦_smc-Set α B) ⟷ A = 0 for A
 proof(intro iffI ballI)

  assume prems[rule-format]: ∀ B∈∘Vset α. ∃!T. T : A ↦_smc-Set α B

  then obtain T where T0B: T : A ↦_smc-Set α 0 by (meson vempty-is-zet)
  then have A[simp]: A ∈∘ Vset α by (fastforce simp: smc-Set-components(1))

  show A = 0
  proof(rule ccontr)
   assume A ≠ 0
   then obtain a where a: a ∈∘ A by (auto dest: trad-foundation)
   from Axiom-of-Powers a have A0:
    [A ×∘ set {0}, A, set {0}]∘ : A ↦_smc-Set α set {0}
    by
     (
      intro smc-Set-is-arrI arr-Set-vfsequenceI,
      unfold arr-Rel-components
     )
     auto
   have A1: [A ×∘ set {1}, A, set {1}]∘ : A ↦_smc-Set α set {1}
   proof
     (
      intro smc-Set-is-arrI arr-Set-vfsequenceI,
      unfold arr-Rel-components
     )
    show set {1} ∈∘ Vset α by (blast intro: vone-in-omega Axiom-of-Infinity)
```

```
    qed auto
    have [A ×∘ set {0}, A, set {0, 1}]∘ : A ↦_{smc-Set α} set {0, 1}
    proof
      (
        intro smc-Set-is-arrI arr-Set-vfsequenceI,
        unfold arr-Rel-components
      )
      show set {0, 1} ∈∘ Vset α
        by (intro Limit-vdoubleton-in-VsetI) (force simp: nat-omega-simps)+
    qed auto
    moreover have
      [A ×∘ set {1}, A, set {0, 1}]∘ : A ↦_{smc-Set α} set {0, 1}
    proof
      (
        intro smc-Set-is-arrI arr-Set-vfsequenceI,
        unfold arr-Rel-components
      )
      show set {0, 1} ∈∘ Vset α
        by (intro Limit-vdoubleton-in-VsetI) (force simp: nat-omega-simps)+
    qed auto
    moreover from ‹A ≠ 0› one-neq-zero have
      [A ×∘ set {0}, A, set {0, 1}]∘ ≠ [A ×∘ set {1}, A, set {0, 1}]∘
      by (blast intro!: vsubset-antisym)
    ultimately show False
      by (metis prems smc-is-arrE smc-Set-components(1))
  qed
next
  fix B assume prems: A = 0 B ∈∘ Vset α
  show ∃!T. T : A ↦_{smc-Set α} B
  proof(rule ex1I[of - ‹[0, 0, B]∘›], unfold prems(1))
    show zzB: [0, 0, B]∘ : 0 ↦_{smc-Set α} B
      by
        (
          intro smc-Set-is-arrI arr-Set-vfsequenceI,
          unfold arr-Rel-components
        )
        (simp-all add: prems)
    fix T assume prems': T : 0 ↦_{smc-Set α} B
    interpret T: arr-Set α T
      rewrites [simp]: T(|ArrDom|) = 0 and [simp]: T(|ArrCod|) = B
      by (intro smc-Set-is-arrD[OF prems'])+
    show T = [0, 0, B]∘
    proof(rule arr-Set-eqI[of α], unfold arr-Rel-components)
      show arr-Set α T by (rule T.arr-Set-axioms)
      from zzB show arr-Set α [0, 0, B]∘ by (meson smc-Set-is-arrE)
      from T.ArrVal.vdomain-vrange-is-vempty show T(|ArrVal|) = 0
        by (auto intro: T.ArrVal.vsv-vrange-vempty simp: smc-Set-cs-simps)
    qed simp-all
  qed
qed

then show ?thesis
  apply(intro iffI obj-initialI, elim obj-initialE)
  subgoal by (metis smc-Set-components(1))
  subgoal by (simp add: smc-Set-components(1))
  subgoal by (metis smc-Set-components(1))
  done
```

qed

There are no null objects in *Set* (this is a trivial corollary of the above).

lemma (in $\mathcal{Z}$**)** *smc-Set-obj-null*: *obj-null* (*smc-Set* α) $A \longleftrightarrow$ *False*
unfolding *obj-null-def smc-Set-obj-terminal smc-Set-obj-initial* **by** *simp*

4.13.5 Zero arrow

There are no zero arrows in *Set* (this result is a trivial corollary of the absence of null objects).

lemma (in $\mathcal{Z}$**)** *smc-Set-is-zero-arr*: $T : A \mapsto_{0 smc\text{-}Set\ \alpha} B \longleftrightarrow$ *False*
using *smc-Set-obj-null* **unfolding** *is-zero-arr-def* **by** *auto*

4.14 *GRPH* as a semicategory

4.14.1 Background

The methodology for the exposition of *GRPH* as a semicategory is analogous to the one used in the previous chapter for the exposition of *GRPH* as a digraph.

named-theorems *smc-GRPH-cs-simps*
named-theorems *smc-GRPH-cs-intros*

4.14.2 Definition and elementary properties

definition *smc-GRPH* :: $V \Rightarrow V$
  where *smc-GRPH* α =
    [
      set $\{\mathfrak{C}.$ *digraph* $\alpha\ \mathfrak{C}\}$,
      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). $\mathfrak{GF}(\!|0|\!) \circ_{DGHM} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
    ]$_\circ$

Components.

lemma *smc-GRPH-components*:
  shows *smc-GRPH* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *digraph* $\alpha\ \mathfrak{C}\}$
    and *smc-GRPH* $\alpha(\!|$*Arr*$|\!)$ = *all-dghms* α
    and *smc-GRPH* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-dghms* $\alpha.\ \mathfrak{F}(\!|$*HomDom*$|\!))$
    and *smc-GRPH* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-dghms* $\alpha.\ \mathfrak{F}(\!|$*HomCod*$|\!))$
    and *smc-GRPH* $\alpha(\!|$*Comp*$|\!)$ =
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-GRPH* α). $\mathfrak{GF}(\!|0|\!) \circ_{DGHM} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
  unfolding *smc-GRPH-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-GRPH*: *smc-dg* (*smc-GRPH* α) = *dg-GRPH* α
proof(*rule vsv-eqI*)
  show *vsv* (*smc-dg* (*smc-GRPH* α)) **unfolding** *smc-dg-def* **by** *auto*
  show *vsv* (*dg-GRPH* α) **unfolding** *dg-GRPH-def* **by** *auto*
  have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-GRPH* α)) = $4_{\mathbb{N}}$
    unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-GRPH* α) = $4_{\mathbb{N}}$
    unfolding *dg-GRPH-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*smc-dg* (*smc-GRPH* α)) = $\mathcal{D}_\circ$ (*dg-GRPH* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show $a \in_\circ \mathcal{D}_\circ$ (*smc-dg* (*smc-GRPH* α)) $\Longrightarrow$ *smc-dg* (*smc-GRPH* α)$(\!|a|\!)$ = *dg-GRPH* $\alpha(\!|a|\!)$
    for *a*
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold smc-dg-def dg-field-simps smc-GRPH-def dg-GRPH-def
)
      (*auto simp*: *nat-omega-simps*)
qed

lemmas-with [*folded smc-dg-GRPH*, *unfolded slicing-simps*]:
  smc-GRPH-ObjI = *dg-GRPH-ObjI*
  and *smc-GRPH-ObjD* = *dg-GRPH-ObjD*
  and *smc-GRPH-ObjE* = *dg-GRPH-ObjE*

and *smc-GRPH-Obj-iff*[*smc-GRPH-cs-simps*] = *dg-GRPH-Obj-iff*
and *smc-GRPH-Dom-app*[*smc-GRPH-cs-simps*] = *dg-GRPH-Dom-app*
and *smc-GRPH-Cod-app*[*smc-GRPH-cs-simps*] = *dg-GRPH-Cod-app*
and *smc-GRPH-is-arrI* = *dg-GRPH-is-arrI*
and *smc-GRPH-is-arrD* = *dg-GRPH-is-arrD*
and *smc-GRPH-is-arrE* = *dg-GRPH-is-arrE*
and *smc-GRPH-is-arr-iff*[*smc-GRPH-cs-simps*] = *dg-GRPH-is-arr-iff*

4.14.3 Composable arrows

lemma *smc-GRPH-composable-arrs-dg-GRPH*:
composable-arrs (*dg-GRPH* α) = *composable-arrs* (*smc-GRPH* α)
unfolding *composable-arrs-def smc-dg-GRPH*[*symmetric*] *slicing-simps* **by** *auto*

lemma *smc-GRPH-Comp*:
smc-GRPH $\alpha(\!|Comp|\!)$ = $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*smc-GRPH* α). $\mathfrak{GF}(\!|0|\!) \circ_{DGHM} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
unfolding *smc-GRPH-components smc-GRPH-composable-arrs-dg-GRPH* **..**

4.14.4 Composition

lemma *smc-GRPH-Comp-app*:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto_{smc\text{-}GRPH\ \alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto_{smc\text{-}GRPH\ \alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{A\,smc\text{-}GRPH\ \alpha} \mathfrak{F} = \mathfrak{G} \circ_{DGHM} \mathfrak{F}$
proof–
from *assms* **have** $[\mathfrak{G}, \mathfrak{F}]_\circ \in_\circ$ *composable-arrs* (*smc-GRPH* α)
by (*auto intro*: *smc-cs-intros*)
then show $\mathfrak{G} \circ_{A\,smc\text{-}GRPH\ \alpha} \mathfrak{F} = \mathfrak{G} \circ_{DGHM} \mathfrak{F}$
unfolding *smc-GRPH-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-GRPH-Comp-vdomain*:
$\mathcal{D}_\circ$ (*smc-GRPH* $\alpha(\!|Comp|\!)$) = *composable-arrs* (*smc-GRPH* α)
unfolding *smc-GRPH-Comp* **by** *auto*

4.14.5 *GRPH* is a semicategory

lemma (**in** $\mathcal{Z}$) *tiny-semicategory-smc-GRPH*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$
shows *tiny-semicategory* β (*smc-GRPH* α)
proof(*intro tiny-semicategoryI*, *unfold smc-GRPH-is-arr-iff*)
show *vfsequence* (*smc-GRPH* α) **unfolding** *smc-GRPH-def* **by** *auto*
show *vcard* (*smc-GRPH* α) = $5_{\mathbb{N}}$
unfolding *smc-GRPH-def* **by** (*simp add*: *nat-omega-simps*)
show (*gf* $\in_\circ \mathcal{D}_\circ$ (*smc-GRPH* $\alpha(\!|Comp|\!)$)) $\longleftrightarrow$
($\exists\, g\ f\ b\ c\ a.$ *gf* = $[g, f]_\circ \wedge g : b \mapsto\mapsto_{DG\alpha} c \wedge f : a \mapsto\mapsto_{DG\alpha} b$)
for *gf*
unfolding *smc-GRPH-Comp-vdomain*
proof
show *gf* $\in_\circ$ *composable-arrs* (*smc-GRPH* α) $\Longrightarrow$
$\exists\, g\ f\ b\ c\ a.$ *gf* = $[g, f]_\circ \wedge g : b \mapsto\mapsto_{DG\alpha} c \wedge f : a \mapsto\mapsto_{DG\alpha} b$
by (*elim composable-arrsE*) (*auto simp*: *smc-GRPH-is-arr-iff*)
next
assume $\exists\, g\ f\ b\ c\ a.$ *gf* = $[g, f]_\circ \wedge g : b \mapsto\mapsto_{DG\alpha} c \wedge f : a \mapsto\mapsto_{DG\alpha} b$
with *smc-GRPH-is-arr-iff* **show** *gf* $\in_\circ$ *composable-arrs* (*smc-GRPH* α)
unfolding *smc-GRPH-Comp-vdomain* **by** (*auto intro*: *smc-cs-intros*)
qed
show $[\![\ g : b \mapsto\mapsto_{DG\alpha} c; f : a \mapsto\mapsto_{DG\alpha} b\ ]\!] \Longrightarrow$
$g \circ_{A\,smc\text{-}GRPH\ \alpha} f : a \mapsto\mapsto_{DG\alpha} c$

```
    for g b c f a
    by (auto simp: smc-GRPH-Comp-app dghm-comp-is-dghm smc-GRPH-cs-simps)
  fix h c d g b f a
  assume h : c ↦↦_{DGα} d g : b ↦↦_{DGα} c f : a ↦↦_{DGα} b
  moreover then have g ∘_{DGHM} f : a ↦↦_{DGα} c h ∘_{DGHM} g : b ↦↦_{DGα} d
    by (auto simp: dghm-comp-is-dghm smc-GRPH-cs-simps)
  ultimately show
    h ∘_{Asmc-GRPH α} g ∘_{Asmc-GRPH α} f =
      h ∘_{Asmc-GRPH α} (g ∘_{Asmc-GRPH α} f)
    by (simp add: smc-GRPH-is-arr-iff smc-GRPH-Comp-app dghm-comp-assoc)
qed (simp-all add: assms smc-dg-GRPH tiny-digraph-dg-GRPH smc-GRPH-components)
```

4.14.6 Initial object

```
lemma (in Z) smc-GRPH-obj-initialI: obj-initial (smc-GRPH α) dg-0
  unfolding obj-initial-def
proof
  (
    intro obj-terminalI,
    unfold smc-op-simps smc-GRPH-is-arr-iff smc-GRPH-Obj-iff
  )
  show digraph α dg-0 by (intro digraph-dg-0)
  fix 𝔄 assume digraph α 𝔄
  then interpret digraph α 𝔄 .
  show ∃!f. f : dg-0 ↦↦_{DGα} 𝔄
  proof
    show dghm-0: dghm-0 𝔄 : dg-0 ↦↦_{DGα} 𝔄
      by (simp add: dghm-0-is-ft-dghm digraph-axioms is-ft-dghm.axioms(1))
    fix 𝔉 assume prems: 𝔉 : dg-0 ↦↦_{DGα} 𝔄
    then interpret 𝔉: is-dghm α dg-0 𝔄 𝔉 .
    show 𝔉 = dghm-0 𝔄
    proof(rule dghm-eqI)
      from dghm-0 show dghm-0 𝔄 : dg-0 ↦↦_{DGα} 𝔄
        unfolding smc-GRPH-is-arr-iff by simp
      have [simp]: 𝒟∘ (𝔉(|ObjMap|)) = 0 by (simp add: dg-cs-simps dg-0-components)
      with 𝔉.ObjMap.vdomain-vrange-is-vempty show 𝔉(|ObjMap|) = dghm-0 𝔄(|ObjMap|)
        by
          (
            auto
              intro: 𝔉.ObjMap.vsv-vrange-vempty
              simp: dg-0-components dghm-0-components
          )
      from 𝔉.dghm-ObjMap-vdomain have 𝒟∘ (𝔉(|ArrMap|)) = 0
        by
          (
            auto
              simp: 𝔉.dghm-ArrMap-vdomain
              intro: 𝔉.HomDom.dg-Arr-vempty-if-Obj-vempty
          )
      then show 𝔉(|ArrMap|) = dghm-0 𝔄(|ArrMap|)
        by
          (
            metis
              𝔉.ArrMap.vsv-axioms
              dghm-0-components(2)
              vsv.vdomain-vrange-is-vempty
              vsv.vsv-vrange-vempty
          )
```

```
    qed (auto simp: dghm-0-components prems)
  qed
qed

lemma (in 𝒵) smc-GRPH-obj-initialD:
  assumes obj-initial (smc-GRPH α) 𝔄
  shows 𝔄 = dg-0
  using assms unfolding obj-initial-def
proof
  (
    elim obj-terminalE,
    unfold smc-op-simps smc-GRPH-is-arr-iff smc-GRPH-Obj-iff
  )
  assume prems: digraph α 𝔄 digraph α 𝔅 ⟹ ∃!𝔉. 𝔉 : 𝔄 ↦↦_{DGα} 𝔅 for 𝔅
  from prems(2)[OF digraph-dg-0] obtain 𝔉 where 𝔉: 𝔉 : 𝔄 ↦↦_{DGα} dg-0
    by meson
  interpret 𝔉: is-dghm α 𝔄 dg-0 𝔉 by (rule 𝔉)
  have ℛ∘ (𝔉(|ObjMap|)) ⊆∘ 0
    unfolding dg-0-components(1)[symmetric] by (simp add: 𝔉.dghm-ObjMap-vrange)
  then have 𝔉(|ObjMap|) = 0 by (auto intro: 𝔉.ObjMap.vsv-vrange-vempty)
  with 𝔉.dghm-ObjMap-vdomain have Obj[simp]: 𝔄(|Obj|) = 0 by auto
  have ℛ∘ (𝔉(|ArrMap|)) ⊆∘ 0
    unfolding dg-0-components(2)[symmetric]
    by (simp add: 𝔉.dghm-ArrMap-vrange)
  then have 𝔉(|ArrMap|) = 0 by (auto intro: 𝔉.ArrMap.vsv-vrange-vempty)
  with 𝔉.dghm-ArrMap-vdomain have Arr[simp]: 𝔄(|Arr|) = 0 by auto
  from Arr 𝔉.HomDom.dg-Dom-vempty-if-Arr-vempty have [simp]: 𝔄(|Dom|) = 0
    by auto
  from Arr 𝔉.HomDom.dg-Cod-vempty-if-Arr-vempty have [simp]: 𝔄(|Cod|) = 0
    by auto
  show 𝔄 = dg-0
    by (rule dg-eqI[of α]) (simp-all add: prems(1) dg-0-components digraph-dg-0)
qed

lemma (in 𝒵) smc-GRPH-obj-initialE:
  assumes obj-initial (smc-GRPH α) 𝔄
  obtains 𝔄 = dg-0
  using assms by (auto dest: smc-GRPH-obj-initialD)

lemma (in 𝒵) smc-GRPH-obj-initial-iff[smc-GRPH-cs-simps]:
  obj-initial (smc-GRPH α) 𝔄 ⟷ 𝔄 = dg-0
  using smc-GRPH-obj-initialI smc-GRPH-obj-initialD by auto
```

4.14.7 Terminal object

```
lemma (in 𝒵) smc-GRPH-obj-terminalI[smc-GRPH-cs-intros]:
  assumes a ∈∘ Vset α and f ∈∘ Vset α
  shows obj-terminal (smc-GRPH α) (dg-1 a f)
proof
  (
    intro obj-terminalI,
    unfold smc-op-simps smc-GRPH-is-arr-iff smc-GRPH-Obj-iff
  )
  fix 𝔄 assume digraph α 𝔄
  then interpret digraph α 𝔄 .
  show ∃!𝔉'. 𝔉' : 𝔄 ↦↦_{DGα} dg-1 a f
  proof
    show dghm-1: dghm-const 𝔄 (dg-1 a f) a f : 𝔄 ↦↦_{DGα} dg-1 a f
```

```
  by
    (
      auto simp:
        assms
        dg-1-is-arr-iff
        dghm-const-is-dghm
        digraph-axioms'
        digraph-dg-1
    )
fix 𝔉' assume prems: 𝔉' : 𝔄 ↦↦_{DGα} dg-1 a f
then interpret 𝔉': is-dghm α 𝔄 ‹dg-1 a f› 𝔉' .
show 𝔉' = dghm-const 𝔄 (dg-1 a f) a f
proof(rule dghm-eqI, unfold dghm-const-components)
  show dghm-const 𝔄 (dg-1 a f) a f : 𝔄 ↦↦_{DGα} dg-1 a f by (rule dghm-1)
  show 𝔉'(|ObjMap|) = vconst-on (𝔄(|Obj|)) a
  proof(cases‹𝔄(|Obj|) = 0›)
    case True
    then have 𝔉'(|ObjMap|) = 0
      by
        (
          simp add:
            𝔉'.ObjMap.vdomain-vrange-is-vempty
            𝔉'.dghm-ObjMap-vsv
            vsv.vsv-vrange-vempty
        )
    with True show ?thesis by simp
  next
    case False
    then have 𝒟∘ (𝔉'(|ObjMap|)) ≠ 0 by (auto simp: 𝔉'.dghm-ObjMap-vdomain)
    with False have ℛ∘ (𝔉'(|ObjMap|)) ≠ 0 by fastforce
    moreover from 𝔉'.dghm-ObjMap-vrange have ℛ∘ (𝔉'(|ObjMap|)) ⊆∘ set {a}
      by (simp add: dg-1-components)
    ultimately have ℛ∘ (𝔉'(|ObjMap|)) = set {a} by auto
    with 𝔉'.dghm-ObjMap-vdomain show ?thesis
      by (intro vsv.vsv-is-vconst-onI) blast+
  qed
  show 𝔉'(|ArrMap|) = vconst-on (𝔄(|Arr|)) f
  proof(cases‹𝔄(|Arr|) = 0›)
    case True
    then have 𝔉'(|ArrMap|) = 0
      by
        (
          simp add:
            𝔉'.ArrMap.vdomain-vrange-is-vempty
            𝔉'.dghm-ArrMap-vsv
            vsv.vsv-vrange-vempty
        )
    with True show ?thesis by simp
  next
    case False
    then have 𝒟∘ (𝔉'(|ArrMap|)) ≠ 0 by (auto simp: 𝔉'.dghm-ArrMap-vdomain)
    with False have ℛ∘ (𝔉'(|ArrMap|)) ≠ 0
      by (force simp: 𝔉'.ArrMap.vdomain-vrange-is-vempty)
    moreover from 𝔉'.dghm-ArrMap-vrange have ℛ∘ (𝔉'(|ArrMap|)) ⊆∘ set {f}
      by (simp add: dg-1-components)
    ultimately have ℛ∘ (𝔉'(|ArrMap|)) = set {f} by auto
    then show ?thesis
      by (intro vsv.vsv-is-vconst-onI) (auto simp: 𝔉'.dghm-ArrMap-vdomain)
```

```
    qed
  qed (auto intro: prems)
 qed
qed (simp add: assms digraph-dg-1)

lemma (in 𝒵) smc-GRPH-obj-terminalE:
 assumes obj-terminal (smc-GRPH α) 𝔅
 obtains a f where a ∈∘ Vset α and f ∈∘ Vset α and 𝔅 = dg-1 a f
 using assms
proof
 (
  elim obj-terminalE;
  unfold smc-op-simps smc-GRPH-is-arr-iff smc-GRPH-Obj-iff
 )
 assume prems: digraph α 𝔅 digraph α 𝔄 ⟹ ∃!𝔉. 𝔉 : 𝔄 ↦↦DGα 𝔅 for 𝔄
 then interpret 𝔅: digraph α 𝔅 by simp
 obtain a where 𝔅-Obj: 𝔅(|Obj|) = set {a} and a: a ∈∘ Vset α
 proof–
  have dg-10: digraph α (dg-10 0) by (rule digraph-dg-10) auto
  from prems(2)[OF dg-10] obtain 𝔉
   where 𝔉: 𝔉 : dg-10 0 ↦↦DGα 𝔅
    and 𝔊𝔉: 𝔊 : dg-10 0 ↦↦DGα 𝔅 ⟹ 𝔊 = 𝔉 for 𝔊
   by fastforce
  interpret 𝔉: is-dghm α ‹dg-10 0› 𝔅 𝔉 by (rule 𝔉)
  have 𝒟∘ (𝔉(|ObjMap|)) = set {0}
   by (simp add: dg-cs-simps dg-10-components)
  then obtain a where vrange-𝔉[simp]: ℛ∘ (𝔉(|ObjMap|)) = set {a}
   by
    (
     auto
      simp: dg-cs-simps
      intro: 𝔉.ObjMap.vsv-vdomain-vsingleton-vrange-vsingleton
    )
  with 𝔅.dg-Obj-vsubset-Vset 𝔉.dghm-ObjMap-vrange have [simp]: a ∈∘ Vset α
   by auto
  from 𝔉.dghm-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
   define 𝔊 where 𝔊 = [set {⟨0, b⟩}, 0, dg-10 0, 𝔅]∘
   have 𝔊-components:
    𝔊(|ObjMap|) = set {⟨0, b⟩}
    𝔊(|ArrMap|) = 0
    𝔊(|HomDom|) = dg-10 0
    𝔊(|HomCod|) = 𝔅
    unfolding 𝔊-def dghm-field-simps by (simp-all add: nat-omega-simps)
   have 𝔊: 𝔊 : dg-10 0 ↦↦DGα 𝔅
    by (rule is-dghmI, unfold 𝔊-components dg-10-components)
     (
      auto simp:
       dg-cs-intros
       nat-omega-simps
       digraph-dg-10
       𝔊-def
       dg-10-is-arr-iff
       b
       vsubset-vsingleton-leftI
```

```
      )
    then have 𝔊-def: 𝔊 = 𝔉 by (rule 𝔊𝔉)
    have ℛ∘ (𝔊(|ObjMap|)) = set {b} unfolding 𝔊-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
obtain f where 𝔅-Arr: 𝔅(|Arr|) = set {f} and f: f ∈∘ Vset α
proof–
  from prems(2)[OF digraph-dg-1, of 0 0] obtain 𝔉
    where 𝔉: 𝔉 : dg-1 0 0 ↦↦DGα 𝔅
      and 𝔊𝔉: 𝔊 : dg-1 0 0 ↦↦DGα 𝔅 ⟹ 𝔊 = 𝔉 for 𝔊
    by fastforce
  interpret 𝔉: is-dghm α ‹dg-1 0 0› 𝔅 𝔉 by (rule 𝔉)
  have 𝒟∘ (𝔉(|ObjMap|)) = set {0}
    by (simp add: dg-cs-simps dg-1-components)
  then obtain a′ where ℛ∘ (𝔉(|ObjMap|)) = set {a′}
    by
      (
        auto
          simp: dg-cs-simps
          intro: 𝔉.ObjMap.vsv-vdomain-vsingleton-vrange-vsingleton
      )
  with 𝔅-Obj 𝔉.dghm-ObjMap-vrange have ℛ∘ (𝔉(|ObjMap|)) = set {a} by auto
  have 𝒟∘ (𝔉(|ArrMap|)) = set {0} by (simp add: dg-cs-simps dg-1-components)
  then obtain f where vrange-𝔉[simp]: ℛ∘ (𝔉(|ArrMap|)) = set {f}
    by
      (
        auto
          simp: dg-cs-simps
          intro: 𝔉.ArrMap.vsv-vdomain-vsingleton-vrange-vsingleton
      )
  with 𝔅.dg-Arr-vsubset-Vset 𝔉.dghm-ArrMap-vrange have [simp]: f ∈∘ Vset α
    by auto
  from 𝔉.dghm-ArrMap-vrange have set {f} ⊆∘ 𝔅(|Arr|) by simp
  moreover have 𝔅(|Arr|) ⊆∘ set {f}
  proof(rule ccontr)
    assume ¬𝔅(|Arr|) ⊆∘ set {f}
    then obtain g where gf: g ≠ f and g: g ∈∘ 𝔅(|Arr|) by force
    have g: g : a ↦𝔅 a
    proof(intro is-arrI)
      from g 𝔅-Obj show 𝔅(|Dom|)(|g|) = a
        by (metis 𝔅.dg-is-arrD(2) is-arr-def vsingleton-iff)
      from g 𝔅-Obj show 𝔅(|Cod|)(|g|) = a
        by (metis 𝔅.dg-is-arrD(3) is-arr-def vsingleton-iff)
    qed (auto simp: g)
    define 𝔊 where 𝔊 = [set {⟨0, a⟩}, set {⟨0, g⟩}, dg-1 0 0, 𝔅]∘
    have 𝔊-components:
      𝔊(|ObjMap|) = set {⟨0, a⟩}
      𝔊(|ArrMap|) = set {⟨0, g⟩}
      𝔊(|HomDom|) = dg-1 0 0
      𝔊(|HomCod|) = 𝔅
      unfolding 𝔊-def dghm-field-simps by (simp-all add: nat-omega-simps)
    have 𝔊: 𝔊 : dg-1 0 0 ↦↦DGα 𝔅
      by (rule is-dghmI, unfold 𝔊-components dg-1-components)
        (
          auto simp:
```

```
        dg-cs-intros nat-omega-simps 𝔊-def dg-1-is-arr-iff 𝔅-Obj g
      )
    then have 𝔊-def: 𝔊 = 𝔉 by (rule 𝔊𝔉)
    have ℛ∘ (𝔊(|ArrMap|)) = set {g} unfolding 𝔊-components by simp
    with vrange-𝔉 gf show False unfolding 𝔊-def by simp
  qed
  ultimately have 𝔅(|Arr|) = set {f} by simp
  with that show ?thesis by simp
qed
have 𝔅 = dg-1 a f
proof(rule dg-eqI[of α], unfold dg-1-components)
  show 𝔅(|Obj|) = set {a} by (simp add: 𝔅-Obj)
  moreover show 𝔅(|Arr|) = set {f} by (simp add: 𝔅-Arr)
  ultimately have 𝔅(|Dom|)(|f|) = a 𝔅(|Cod|)(|f|) = a
    by (metis 𝔅.dg-is-arrE is-arr-def vsingleton-iff)+
  have 𝒟∘ (𝔅(|Dom|)) = set {f} by (simp add: dg-cs-simps 𝔅-Arr)
  moreover from 𝔅.Dom.vsv-vrange-vempty 𝔅.dg-Dom-vdomain 𝔅.dg-Dom-vrange
  have ℛ∘ (𝔅(|Dom|)) = set {a} by (fastforce simp: 𝔅-Arr 𝔅-Obj)
  ultimately show 𝔅(|Dom|) = set {⟨f, a⟩}
    using 𝔅.Dom.vsv-vdomain-vrange-vsingleton by simp
  have 𝒟∘ (𝔅(|Cod|)) = set {f} by (simp add: dg-cs-simps 𝔅-Arr)
  moreover from 𝔅.Cod.vsv-vrange-vempty 𝔅.dg-Cod-vdomain 𝔅.dg-Cod-vrange
  have ℛ∘ (𝔅(|Cod|)) = set {a}
    by (fastforce simp: 𝔅-Arr 𝔅-Obj)
  ultimately show 𝔅(|Cod|) = set {⟨f, a⟩}
    using assms 𝔅.Cod.vsv-vdomain-vrange-vsingleton by simp
qed (auto simp: dg-cs-intros 𝔅-Obj digraph-dg-1 a f)
with a f that show ?thesis by auto
qed
```

4.15 *SemiCAT* as a digraph

4.15.1 Background

SemiCAT is usually defined as a category of semicategories and semifunctors (e.g., see [3][8]). However, there is little that can prevent one from exposing *SemiCAT* as a digraph and provide additional structure gradually in subsequent theories. Thus, in this section, α-*SemiCAT* is defined as a digraph of semicategories and semifunctors in V_α.

named-theorems *dg-SemiCAT-simps*
named-theorems *dg-SemiCAT-intros*

4.15.2 Definition and elementary properties

definition *dg-SemiCAT* :: $V \Rightarrow V$
  where *dg-SemiCAT* α =
    [
      set $\{\mathfrak{C}.$ *semicategory* α $\mathfrak{C}\}$,
      all-smcfs α,
      $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$,
      $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
    ]$_\circ$

Components.

lemma *dg-SemiCAT-components*:
  shows *dg-SemiCAT* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *semicategory* α $\mathfrak{C}\}$
    and *dg-SemiCAT* $\alpha(\!|$*Arr*$|\!)$ = *all-smcfs* α
    and *dg-SemiCAT* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$
    and *dg-SemiCAT* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
  unfolding *dg-SemiCAT-def dg-field-simps* **by** (*simp-all add*: *nat-omega-simps*)

4.15.3 Object

lemma *dg-SemiCAT-ObjI*:
  assumes *semicategory* α $\mathfrak{A}$
  shows $\mathfrak{A}$ $\in_\circ$ *dg-SemiCAT* $\alpha(\!|$*Obj*$|\!)$
  using *assms* **unfolding** *dg-SemiCAT-components* **by** *auto*

lemma *dg-SemiCAT-ObjD*:
  assumes $\mathfrak{A}$ $\in_\circ$ *dg-SemiCAT* $\alpha(\!|$*Obj*$|\!)$
  shows *semicategory* α $\mathfrak{A}$
  using *assms* **unfolding** *dg-SemiCAT-components* **by** *auto*

lemma *dg-SemiCAT-ObjE*:
  assumes $\mathfrak{A}$ $\in_\circ$ *dg-SemiCAT* $\alpha(\!|$*Obj*$|\!)$
  obtains *semicategory* α $\mathfrak{A}$
  using *assms* **unfolding** *dg-SemiCAT-components* **by** *auto*

lemma *dg-SemiCAT-Obj-iff*[*dg-SemiCAT-simps*]:
  $\mathfrak{A}$ $\in_\circ$ *dg-SemiCAT* $\alpha(\!|$*Obj*$|\!)$ $\longleftrightarrow$ *semicategory* α $\mathfrak{A}$
  unfolding *dg-SemiCAT-components* **by** *auto*

4.15.4 Domain and codomain

lemma [*dg-SemiCAT-simps*]:
  assumes $\mathfrak{F}$ $\in_\circ$ *all-smcfs* α
  shows *dg-SemiCAT-Dom-app*: *dg-SemiCAT* $\alpha(\!|$*Dom*$|\!)(\!|\mathfrak{F}|\!)$ = $\mathfrak{F}(\!|$*HomDom*$|\!)$

[8] https://ncatlab.org/nlab/show/semicategory

and *dg-SemiCAT-Cod-app*: $dg\text{-}SemiCAT\ \alpha(\!|Cod|\!)(\!|\mathfrak{F}|\!) = \mathfrak{F}(\!|HomCod|\!)$
using *assms* **unfolding** *dg-SemiCAT-components* **by** *auto*

4.15.5 *SemiCAT* is a digraph

lemma (in $\mathcal{Z}$**)** *tiny-digraph-dg-SemiCAT*:
assumes $\mathcal{Z}\ \beta$ **and** $\alpha \in_\circ \beta$
shows $tiny\text{-}digraph\ \beta\ (dg\text{-}SemiCAT\ \alpha)$
proof(*intro tiny-digraphI*)
show $vfsequence\ (dg\text{-}SemiCAT\ \alpha)$ **unfolding** *dg-SemiCAT-def* **by** *simp*
show $vcard\ (dg\text{-}SemiCAT\ \alpha) = 4_{\mathbb{N}}$
unfolding *dg-SemiCAT-def* **by** (*simp add*: *nat-omega-simps*)
show $\mathcal{R}_\circ\ (dg\text{-}SemiCAT\ \alpha(\!|Dom|\!)) \subseteq_\circ dg\text{-}SemiCAT\ \alpha(\!|Obj|\!)$
proof(*intro vsubsetI*)
fix $\mathfrak{A}$ **assume** $\mathfrak{A} \in_\circ \mathcal{R}_\circ\ (dg\text{-}SemiCAT\ \alpha(\!|Dom|\!))$
then obtain $\mathfrak{F}$
where $\mathfrak{F} \in_\circ all\text{-}smcfs\ \alpha$ **and** $\mathfrak{A}\text{-}def$: $\mathfrak{A} = \mathfrak{F}(\!|HomDom|\!)$
unfolding *dg-SemiCAT-components* **by** *auto*
then obtain $\mathfrak{B}\ \mathfrak{F}$ **where** $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
unfolding *dg-SemiCAT-components* **by** *auto*
then interpret *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **.**
show $\mathfrak{A} \in_\circ dg\text{-}SemiCAT\ \alpha(\!|Obj|\!)$
by (*simp add*: *dg-SemiCAT-components HomDom.semicategory-axioms*)
qed
show $\mathcal{R}_\circ\ (dg\text{-}SemiCAT\ \alpha(\!|Cod|\!)) \subseteq_\circ dg\text{-}SemiCAT\ \alpha(\!|Obj|\!)$
proof(*intro vsubsetI*)
fix $\mathfrak{B}$ **assume** $\mathfrak{B} \in_\circ \mathcal{R}_\circ\ (dg\text{-}SemiCAT\ \alpha(\!|Cod|\!))$
then obtain $\mathfrak{F}$ **where** $\mathfrak{F} \in_\circ \mathcal{D}_\circ\ (dg\text{-}SemiCAT\ \alpha(\!|Cod|\!))$ **and** $\mathfrak{B} = \mathfrak{F}(\!|HomCod|\!)$
unfolding *dg-SemiCAT-components* **by** *auto*
then obtain $\mathfrak{A}\ \mathfrak{F}$
where $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$ **and** $\mathfrak{A}\text{-}def$: $\mathfrak{B} = \mathfrak{F}(\!|HomCod|\!)$
unfolding *dg-SemiCAT-components* **by** *auto*
have $\mathfrak{B} = \mathfrak{F}(\!|HomCod|\!)$ **unfolding** $\mathfrak{A}\text{-}def$ **by** *simp*
interpret *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule* $\mathfrak{F}$)
show $\mathfrak{B} \in_\circ dg\text{-}SemiCAT\ \alpha(\!|Obj|\!)$
by (*simp add*: *HomCod.semicategory-axioms dg-SemiCAT-components*)
qed
show $dg\text{-}SemiCAT\ \alpha(\!|Obj|\!) \in_\circ Vset\ \beta$
unfolding *dg-SemiCAT-components* **by** (*rule semicategories-in-Vset*[*OF assms*])
show $dg\text{-}SemiCAT\ \alpha(\!|Arr|\!) \in_\circ Vset\ \beta$
unfolding *dg-SemiCAT-components* **by** (*rule all-smcfs-in-Vset*[*OF assms*])
qed (*simp-all add*: *assms dg-SemiCAT-components*)

4.15.6 Arrow with a domain and a codomain

lemma *dg-SemiCAT-is-arrI*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}SemiCAT\ \alpha} \mathfrak{B}$
proof(*intro is-arrI*, *unfold dg-SemiCAT-components*(2))
interpret *is-semifunctor* $\alpha\ \mathfrak{A}\ \mathfrak{B}\ \mathfrak{F}$ **by** (*rule assms*)
from *assms* **show** $\mathfrak{F} \in_\circ all\text{-}smcfs\ \alpha$ **by** *auto*
with *assms* **show** $dg\text{-}SemiCAT\ \alpha(\!|Dom|\!)(\!|\mathfrak{F}|\!) = \mathfrak{A}\ dg\text{-}SemiCAT\ \alpha(\!|Cod|\!)(\!|\mathfrak{F}|\!) = \mathfrak{B}$
by (*simp-all add*: *smc-cs-simps dg-SemiCAT-components*)
qed

lemma *dg-SemiCAT-is-arrD*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}SemiCAT\ \alpha} \mathfrak{B}$
shows $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$

using *assms* **by** (*elim is-arrE*) (*auto simp*: *dg-SemiCAT-components*)

lemma *dg-SemiCAT-is-arrE*:
assumes $\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}SemiCAT\ \alpha} \mathfrak{B}$
obtains $\mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
using *assms* **by** (*simp add*: *dg-SemiCAT-is-arrD*)

lemma *dg-SemiCAT-is-arr-iff*[*dg-SemiCAT-simps*]:
$\mathfrak{F} : \mathfrak{A} \mapsto_{dg\text{-}SemiCAT\ \alpha} \mathfrak{B} \longleftrightarrow \mathfrak{F} : \mathfrak{A} \mapsto\mapsto_{SMC\alpha} \mathfrak{B}$
by (*auto intro*: *dg-SemiCAT-is-arrI dest*: *dg-SemiCAT-is-arrD*)

4.16 *SemiCAT* as a semicategory

4.16.1 Background

The subsection presents the theory of the semicategories of α-semicategories. It continues the development that was initiated in section 4.15.

named-theorems *smc-SemiCAT-simps*
named-theorems *smc-SemiCAT-intros*

4.16.2 Definition and elementary properties

definition *smc-SemiCAT* :: $V \Rightarrow V$
  where *smc-SemiCAT* α =
    [
      set $\{\mathfrak{C}.$ *semicategory* α $\mathfrak{C}\}$,
      all-smcfs α,
      $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$,
      $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$,
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-SemiCAT* α)$.$ $\mathfrak{GF}(\!|0|\!) \circ_{SMCF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
    ]$_\circ$

Components.

lemma *smc-SemiCAT-components*:
  shows *smc-SemiCAT* $\alpha(\!|$*Obj*$|\!)$ = *set* $\{\mathfrak{C}.$ *semicategory* α $\mathfrak{C}\}$
    and *smc-SemiCAT* $\alpha(\!|$*Arr*$|\!)$ = *all-smcfs* α
    and *smc-SemiCAT* $\alpha(\!|$*Dom*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomDom*$|\!))$
    and *smc-SemiCAT* $\alpha(\!|$*Cod*$|\!)$ = $(\lambda\mathfrak{F}\in_\circ$*all-smcfs* $\alpha.$ $\mathfrak{F}(\!|$*HomCod*$|\!))$
    and *smc-SemiCAT* $\alpha(\!|$*Comp*$|\!)$ =
      $(\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*dg-SemiCAT* α)$.$ $\mathfrak{GF}(\!|0|\!) \circ_{SMCF} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!))$
  unfolding *smc-SemiCAT-def dg-field-simps*
  by (*simp-all add*: *nat-omega-simps*)

Slicing.

lemma *smc-dg-SemiCAT*[*smc-SemiCAT-simps*]: *smc-dg* (*smc-SemiCAT* α) = *dg-SemiCAT* α
proof(*rule vsv-eqI*)
  show *vsv* (*smc-dg* (*smc-SemiCAT* α)) **unfolding** *smc-dg-def* **by** *auto*
  show *vsv* (*dg-SemiCAT* α) **unfolding** *dg-SemiCAT-def* **by** *auto*
  have *dom-lhs*: $\mathcal{D}_\circ$ (*smc-dg* (*smc-SemiCAT* α)) = $4_{\mathbb{N}}$
    unfolding *smc-dg-def* **by** (*simp add*: *nat-omega-simps*)
  have *dom-rhs*: $\mathcal{D}_\circ$ (*dg-SemiCAT* α) = $4_{\mathbb{N}}$
    unfolding *dg-SemiCAT-def* **by** (*simp add*: *nat-omega-simps*)
  show $\mathcal{D}_\circ$ (*smc-dg* (*smc-SemiCAT* α)) = $\mathcal{D}_\circ$ (*dg-SemiCAT* α)
    unfolding *dom-lhs dom-rhs* **by** *simp*
  show *a* $\in_\circ$ $\mathcal{D}_\circ$ (*smc-dg* (*smc-SemiCAT* α)) $\Longrightarrow$
    smc-dg (*smc-SemiCAT* α)$(\!|a|\!)$ = *dg-SemiCAT* $\alpha(\!|a|\!)$
    for *a*
    by
      (
        unfold dom-lhs,
        elim-in-numeral,
        unfold smc-dg-def dg-field-simps smc-SemiCAT-def dg-SemiCAT-def
)
      (*auto simp*: *nat-omega-simps*)
qed

lemmas-with [*folded smc-dg-SemiCAT*, *unfolded slicing-simps*]:
  smc-SemiCAT-ObjI = *dg-SemiCAT-ObjI*

and *smc-SemiCAT-ObjD* = *dg-SemiCAT-ObjD*
and *smc-SemiCAT-ObjE* = *dg-SemiCAT-ObjE*
and *smc-SemiCAT-Obj-iff*[*smc-SemiCAT-simps*] = *dg-SemiCAT-Obj-iff*
and *smc-SemiCAT-Dom-app*[*smc-SemiCAT-simps*] = *dg-SemiCAT-Dom-app*
and *smc-SemiCAT-Cod-app*[*smc-SemiCAT-simps*] = *dg-SemiCAT-Cod-app*
and *smc-SemiCAT-is-arrI* = *dg-SemiCAT-is-arrI*
and *smc-SemiCAT-is-arrD* = *dg-SemiCAT-is-arrD*
and *smc-SemiCAT-is-arrE* = *dg-SemiCAT-is-arrE*
and *smc-SemiCAT-is-arr-iff*[*smc-SemiCAT-simps*] = *dg-SemiCAT-is-arr-iff*

4.16.3 Composable arrows

lemma *smc-SemiCAT-composable-arrs-dg-SemiCAT*:
composable-arrs (*dg-SemiCAT* α) = *composable-arrs* (*smc-SemiCAT* α)
unfolding *composable-arrs-def smc-dg-SemiCAT*[*symmetric*] *slicing-simps* **by** *auto*

lemma *smc-SemiCAT-Comp*:
smc-SemiCAT $\alpha(\!|Comp|\!)$ =
($\lambda\mathfrak{GF}\in_\circ$*composable-arrs* (*smc-SemiCAT* α). $\mathfrak{GF}(\!|0|\!) \circ_{DGHM} \mathfrak{GF}(\!|1_{\mathbb{N}}|\!)$)
unfolding *smc-SemiCAT-components smc-SemiCAT-composable-arrs-dg-SemiCAT* **..**

4.16.4 Composition

lemma *smc-SemiCAT-Comp-app*[*smc-SemiCAT-simps*]:
assumes $\mathfrak{G} : \mathfrak{B} \mapsto_{\textit{smc-SemiCAT } \alpha} \mathfrak{C}$ **and** $\mathfrak{F} : \mathfrak{A} \mapsto_{\textit{smc-SemiCAT } \alpha} \mathfrak{B}$
shows $\mathfrak{G} \circ_{A\,\textit{smc-SemiCAT } \alpha} \mathfrak{F} = \mathfrak{G} \circ_{SMCF} \mathfrak{F}$
proof–
from *assms* **have** $[\mathfrak{G}, \mathfrak{F}]_\circ \in_\circ$ *composable-arrs* (*smc-SemiCAT* α)
by (*auto simp*: *composable-arrsI*)
then show $\mathfrak{G} \circ_{A\,\textit{smc-SemiCAT } \alpha} \mathfrak{F} = \mathfrak{G} \circ_{SMCF} \mathfrak{F}$
unfolding *smc-SemiCAT-Comp* **by** (*simp add*: *nat-omega-simps*)
qed

lemma *smc-SemiCAT-Comp-vdomain*[*smc-SemiCAT-simps*]:
$\mathcal{D}_\circ$ (*smc-SemiCAT* $\alpha(\!|Comp|\!)$) = *composable-arrs* (*smc-SemiCAT* α)
unfolding *smc-SemiCAT-Comp* **by** *auto*

lemma *smc-SemiCAT-Comp-vrange*: $\mathcal{R}_\circ$ (*smc-SemiCAT* $\alpha(\!|Comp|\!)$) $\subseteq_\circ$ *all-smcfs* α
proof(*rule vsubsetI*)
fix $\mathfrak{H}$ **assume** $\mathfrak{H} \in_\circ \mathcal{R}_\circ$ (*smc-SemiCAT* $\alpha(\!|Comp|\!)$)
then obtain $\mathfrak{GF}$
where $\mathfrak{H}$-*def*: $\mathfrak{H}$ = *smc-SemiCAT* $\alpha(\!|Comp|\!)(\!|\mathfrak{GF}|\!)$
and $\mathfrak{GF} \in_\circ \mathcal{D}_\circ$ (*smc-SemiCAT* $\alpha(\!|Comp|\!)$)
unfolding *smc-SemiCAT-components* **by** (*auto intro*: *composable-arrsI*)
then obtain $\mathfrak{G}$ $\mathfrak{F}$ $\mathfrak{A}$ $\mathfrak{B}$ $\mathfrak{C}$
where $\mathfrak{GF} = [\mathfrak{G}, \mathfrak{F}]_\circ$
and $\mathfrak{G}$: $\mathfrak{G} : \mathfrak{B} \mapsto_{\textit{smc-SemiCAT } \alpha} \mathfrak{C}$
and $\mathfrak{F}$: $\mathfrak{F} : \mathfrak{A} \mapsto_{\textit{smc-SemiCAT } \alpha} \mathfrak{B}$
by (*auto simp*: *smc-SemiCAT-Comp-vdomain*)
with $\mathfrak{H}$-*def* **have** $\mathfrak{H}$-*def'*: $\mathfrak{H} = \mathfrak{G} \circ_{A\,\textit{smc-SemiCAT } \alpha} \mathfrak{F}$ **by** *simp*
from $\mathfrak{G}$ $\mathfrak{F}$ **show** $\mathfrak{H} \in_\circ$ *all-smcfs* α
unfolding $\mathfrak{H}$-*def'* **by** (*auto intro*: *smc-cs-intros simp*: *smc-SemiCAT-simps*)
qed

4.16.5 *SemiCAT* is a semicategory

lemma (**in** $\mathcal{Z}$) *tiny-semicategory-smc-SemiCAT*:
assumes $\mathcal{Z}$ β **and** $\alpha \in_\circ \beta$

```
  shows tiny-semicategory β (smc-SemiCAT α)
proof(intro tiny-semicategoryI, unfold smc-SemiCAT-is-arr-iff)
  show vfsequence (smc-SemiCAT α) unfolding smc-SemiCAT-def by auto
  show vcard (smc-SemiCAT α) = 5_ℕ
    unfolding smc-SemiCAT-def by (simp add: nat-omega-simps)
  show (𝔊𝔉 ∈₀ 𝒟₀ (smc-SemiCAT α(|Comp|))) ⟷
    (∃𝔊 𝔉 𝔅 ℭ 𝔄. 𝔊𝔉 = [𝔊, 𝔉]₀ ∧ 𝔊 : 𝔅 ↦↦_SMCα ℭ ∧ 𝔉 : 𝔄 ↦↦_SMCα 𝔅)
    for 𝔊𝔉
    unfolding smc-SemiCAT-Comp-vdomain
  proof
    show 𝔊𝔉 ∈₀ composable-arrs (smc-SemiCAT α) ⟹
      ∃𝔊 𝔉 𝔅 ℭ 𝔄. 𝔊𝔉 = [𝔊, 𝔉]₀ ∧ 𝔊 : 𝔅 ↦↦_SMCα ℭ ∧ 𝔉 : 𝔄 ↦↦_SMCα 𝔅
      by (elim composable-arrsE) (auto simp: smc-SemiCAT-is-arr-iff)
  next
    assume ∃𝔊 𝔉 𝔅 ℭ 𝔄. 𝔊𝔉 = [𝔊, 𝔉]₀ ∧ 𝔊 : 𝔅 ↦↦_SMCα ℭ ∧ 𝔉 : 𝔄 ↦↦_SMCα 𝔅
    with smc-SemiCAT-is-arr-iff show 𝔊𝔉 ∈₀ composable-arrs (smc-SemiCAT α)
      unfolding smc-SemiCAT-Comp-vdomain by (auto intro: smc-cs-intros)
  qed
  show [[ 𝔊 : 𝔅 ↦↦_SMCα ℭ; 𝔉 : 𝔄 ↦↦_SMCα 𝔅 ]] ⟹
    𝔊 ∘_A smc-SemiCAT α 𝔉 : 𝔄 ↦↦_SMCα ℭ
    for 𝔊 𝔅 ℭ 𝔉 𝔄
    by (auto simp: smc-SemiCAT-simps intro: smc-cs-intros)
  fix ℌ ℭ 𝔇 𝔊 𝔅 𝔉 𝔄
  assume ℌ : ℭ ↦↦_SMCα 𝔇 𝔊 : 𝔅 ↦↦_SMCα ℭ 𝔉 : 𝔄 ↦↦_SMCα 𝔅
  moreover then have 𝔊 ∘_SMCF 𝔉 : 𝔄 ↦↦_SMCα ℭ ℌ ∘_SMCF 𝔊 : 𝔅 ↦↦_SMCα 𝔇
    by (auto intro: smc-cs-intros)
  ultimately show ℌ ∘_A smc-SemiCAT α 𝔊 ∘_A smc-SemiCAT α 𝔉 =
    ℌ ∘_A smc-SemiCAT α (𝔊 ∘_A smc-SemiCAT α 𝔉)
    by
      (
        simp add:
          smc-SemiCAT-is-arr-iff smc-SemiCAT-Comp-app smcf-comp-assoc
      )
qed
  (
    auto simp:
      assms smc-dg-SemiCAT tiny-digraph-dg-SemiCAT smc-SemiCAT-components
  )
```

4.16.6 Initial object

```
lemma (in 𝒵) smc-SemiCAT-obj-initialI: obj-initial (smc-SemiCAT α) smc-0
  unfolding obj-initial-def
proof
  (
    intro obj-terminalI,
    unfold smc-op-simps smc-SemiCAT-is-arr-iff smc-SemiCAT-Obj-iff
  )
  show semicategory α smc-0 by (intro semicategory-smc-0)
  fix 𝔄 assume prems: semicategory α 𝔄
  interpret semicategory α 𝔄 using prems .
  show ∃!𝔉. 𝔉 : smc-0 ↦↦_SMCα 𝔄
  proof
    show smcf-0: smcf-0 𝔄 : smc-0 ↦↦_SMCα 𝔄
      by
        (
          simp add:
            smcf-0-is-ft-semifunctor semicategory-axioms is-ft-semifunctor.axioms(1)
```

```
      )
    fix 𝔉 assume prems: 𝔉 : smc-0 ↦↦_{SMCα} 𝔄
    then interpret 𝔉: is-semifunctor α smc-0 𝔄 𝔉 .
    show 𝔉 = smcf-0 𝔄
    proof(rule smcf-eqI)
      show 𝔉 : smc-0 ↦↦_{SMCα} 𝔄 by (auto simp: smc-cs-intros)
      from smcf-0 show smcf-0 𝔄 : smc-0 ↦↦_{SMCα} 𝔄
        unfolding smc-SemiCAT-is-arr-iff by simp
      have 𝒟∘ (𝔉(|ObjMap|)) = 0 by (auto simp: smc-0-components smc-cs-simps)
      with 𝔉.ObjMap.vdomain-vrange-is-vempty show 𝔉(|ObjMap|) = smcf-0 𝔄(|ObjMap|)
        unfolding smcf-0-components by (auto intro: 𝔉.ObjMap.vsv-vrange-vempty)
      have 𝒟∘ (𝔉(|ArrMap|)) = 0 by (auto simp: smc-0-components smc-cs-simps)
      with 𝔉.ArrMap.vdomain-vrange-is-vempty show 𝔉(|ArrMap|) = smcf-0 𝔄(|ArrMap|)
        unfolding smcf-0-components by (auto intro: 𝔉.ArrMap.vsv-vrange-vempty)
    qed (simp-all add: smcf-0-components)
  qed
qed

lemma (in 𝒵) smc-SemiCAT-obj-initialD:
  assumes obj-initial (smc-SemiCAT α) 𝔄
  shows 𝔄 = smc-0
  using assms unfolding obj-initial-def
proof
  (
    elim obj-terminalE,
    unfold smc-op-simps smc-SemiCAT-is-arr-iff smc-SemiCAT-Obj-iff
  )
  assume prems:
    semicategory α 𝔄
    semicategory α 𝔅 ⟹ ∃!𝔉. 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
    for 𝔅
  from prems(2)[OF semicategory-smc-0] obtain 𝔉 where 𝔉 : 𝔄 ↦↦_{SMCα} smc-0
    by meson
  then interpret 𝔉: is-semifunctor α 𝔄 smc-0 𝔉 .
  have ℛ∘ (𝔉(|ObjMap|)) ⊆∘ 0
    unfolding smc-0-components(1)[symmetric]
    by (simp add: 𝔉.smcf-ObjMap-vrange)
  then have 𝔉(|ObjMap|) = 0 by (auto intro: 𝔉.ObjMap.vsv-vrange-vempty)
  with 𝔉.smcf-ObjMap-vdomain have Obj[simp]: 𝔄(|Obj|) = 0 by auto
  have ℛ∘ (𝔉(|ArrMap|)) ⊆∘ 0
    unfolding smc-0-components(2)[symmetric]
    by (simp add: 𝔉.smcf-ArrMap-vrange)
  then have 𝔉(|ArrMap|) = 0 by (auto intro: 𝔉.ArrMap.vsv-vrange-vempty)
  with 𝔉.smcf-ArrMap-vdomain have Arr[simp]: 𝔄(|Arr|) = 0 by auto
  from 𝔉.HomDom.Dom.vdomain-vrange-is-vempty have [simp]: 𝔄(|Dom|) = 0
    by (auto simp: smc-cs-simps intro: 𝔉.HomDom.Dom.vsv-vrange-vempty)
  from 𝔉.HomDom.Cod.vdomain-vrange-is-vempty have [simp]: 𝔄(|Cod|) = 0
    by (auto simp: smc-cs-simps intro: 𝔉.HomDom.Cod.vsv-vrange-vempty)
  from Arr have 𝔄(|Arr|) ^× 2_ℕ = 0 by (simp add: vcpower-of-vempty)
  with 𝔉.HomDom.Comp.pnop-vdomain have 𝒟∘ (𝔄(|Comp|)) = 0 by simp
  with 𝔉.HomDom.Comp.vdomain-vrange-is-vempty have [simp]: 𝔄(|Comp|) = 0
    by (auto intro: 𝔉.HomDom.Comp.vsv-vrange-vempty)
  show 𝔄 = smc-0
    by (rule smc-eqI[of α])
      (simp-all add: prems(1) smc-0-components semicategory-smc-0)
qed

lemma (in 𝒵) smc-SemiCAT-obj-initialE:
```

```
  assumes obj-initial (smc-SemiCAT α) 𝔄
  obtains 𝔄 = smc-0
  using assms by (auto dest: smc-SemiCAT-obj-initialD)

lemma (in 𝒵) smc-SemiCAT-obj-initial-iff[smc-SemiCAT-simps]:
  obj-initial (smc-SemiCAT α) 𝔄 ⟷ 𝔄 = smc-0
  using smc-SemiCAT-obj-initialI smc-SemiCAT-obj-initialD by auto
```

4.16.7 Terminal object

```
lemma (in 𝒵) smc-SemiCAT-obj-terminalI[smc-SemiCAT-intros]:
  assumes a ∈∘ Vset α and f ∈∘ Vset α
  shows obj-terminal (smc-SemiCAT α) (smc-1 a f)
proof
  (
    intro obj-terminalI,
    unfold smc-op-simps smc-SemiCAT-is-arr-iff smc-SemiCAT-Obj-iff
  )
  fix 𝔄 assume semicategory α 𝔄
  then interpret semicategory α 𝔄 .
  show ∃!ℱ'. ℱ' : 𝔄 ↦↦SMCα smc-1 a f
  proof
    show smcf-1: smcf-const 𝔄 (smc-1 a f) a f : 𝔄 ↦↦SMCα smc-1 a f
      by
        (
          auto
            intro: smc-cs-intros smc-1-is-arrI smcf-const-is-semifunctor
            simp: assms semicategory-smc-1
        )
    fix ℱ' assume ℱ' : 𝔄 ↦↦SMCα smc-1 a f
    then interpret ℱ': is-semifunctor α 𝔄 ‹smc-1 a f› ℱ' .
    show ℱ' = smcf-const 𝔄 (smc-1 a f) a f
    proof(rule smcf-eqI, unfold dghm-const-components)
      show smcf-const 𝔄 (smc-1 a f) a f : 𝔄 ↦↦SMCα smc-1 a f
        by (rule smcf-1)
      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: smc-cs-simps)
        with True show ?thesis by simp
      next
        case False
        then have 𝒟∘ (ℱ'(|ObjMap|)) ≠ 0 by (auto simp: smc-cs-simps)
        then have ℛ∘ (ℱ'(|ObjMap|)) ≠ 0
          by (simp add: ℱ'.ObjMap.vsv-vdomain-vempty-vrange-vempty)
        moreover from ℱ'.smcf-ObjMap-vrange have ℛ∘ (ℱ'(|ObjMap|)) ⊆∘ set {a}
          by (simp add: smc-1-components)
        ultimately have ℛ∘ (ℱ'(|ObjMap|)) = set {a} by auto
        then show ?thesis
          by (intro vsv.vsv-is-vconst-onI) (auto simp: smc-cs-simps)
      qed
      show ℱ'(|ArrMap|) = vconst-on (𝔄(|Arr|)) f
      proof(cases‹𝔄(|Arr|) = 0›)
        case True
        with ℱ'.ArrMap.vdomain-vrange-is-vempty have ℱ'(|ArrMap|) = 0
          by (simp add: smc-cs-simps ℱ'.smcf-ArrMap-vsv vsv.vsv-vrange-vempty)
        with True show ?thesis by simp
```

```
      next
        case False
        then have 𝒟ₒ (𝔉′(⌊ArrMap⌋)) ≠ 0 by (auto simp: smc-cs-simps)
        then have ℛₒ (𝔉′(⌊ArrMap⌋)) ≠ 0
          by (simp add: 𝔉′.ArrMap.vsv-vdomain-vempty-vrange-vempty)
        moreover from 𝔉′.smcf-ArrMap-vrange have ℛₒ (𝔉′(⌊ArrMap⌋)) ⊆ₒ set {f}
          by (simp add: smc-1-components)
        ultimately have ℛₒ (𝔉′(⌊ArrMap⌋)) = set {f} by auto
        then show ?thesis
          by (intro vsv.vsv-is-vconst-onI) (auto simp: smc-cs-simps)
      qed
    qed (auto intro: smc-cs-intros)
  qed
qed (simp add: assms semicategory-smc-1)

lemma (in 𝒵) smc-SemiCAT-obj-terminalE:
  assumes obj-terminal (smc-SemiCAT α) 𝔅
  obtains a f where a ∈ₒ Vset α and f ∈ₒ Vset α and 𝔅 = smc-1 a f
  using assms
proof
  (
    elim obj-terminalE,
    unfold smc-op-simps smc-SemiCAT-is-arr-iff smc-SemiCAT-Obj-iff
  )

  assume prems:
    semicategory α 𝔅
    semicategory α 𝔄 ⟹ ∃!𝔉. 𝔉 : 𝔄 ↦↦_{SMCα} 𝔅
    for 𝔄
  interpret 𝔅: semicategory α 𝔅 by (rule prems(1))

  obtain a where 𝔅-Obj: 𝔅(⌊Obj⌋) = set {a} and a: a ∈ₒ Vset α
  proof–
    have semicategory-smc-10: semicategory α (smc-10 0)
      by (intro semicategory-smc-10) auto
    from prems(2)[OF semicategory-smc-10] obtain 𝔉
      where 𝔉: 𝔉 : smc-10 0 ↦↦_{SMCα} 𝔅
        and 𝔊𝔉: 𝔊 : smc-10 0 ↦↦_{SMCα} 𝔅 ⟹ 𝔊 = 𝔉 for 𝔊
      by fastforce
    interpret 𝔉: is-semifunctor α ‹smc-10 0› 𝔅 𝔉 by (rule 𝔉)
    have 𝒟ₒ (𝔉(⌊ObjMap⌋)) = set {0}
      by (auto simp add: smc-10-components smc-cs-simps)
    then obtain a where vrange-𝔉[simp]: ℛₒ (𝔉(⌊ObjMap⌋)) = set {a}
      by (auto intro: 𝔉.ObjMap.vsv-vdomain-vsingleton-vrange-vsingleton)
    with 𝔅.smc-Obj-vsubset-Vset 𝔉.smcf-ObjMap-vrange have [simp]: a ∈ₒ Vset α
      by auto
    from 𝔉.smcf-ObjMap-vrange have set {a} ⊆ₒ 𝔅(⌊Obj⌋) by simp
    moreover have 𝔅(⌊Obj⌋) ⊆ₒ set {a}
    proof(rule ccontr)
      assume ¬ 𝔅(⌊Obj⌋) ⊆ₒ set {a}
      then obtain b where ba: b ≠ a and b: b ∈ₒ 𝔅(⌊Obj⌋) by force
      define 𝔊 where 𝔊 = [set {⟨0, b⟩}, 0, smc-10 0, 𝔅]ₒ
      have 𝔊-components:
        𝔊(⌊ObjMap⌋) = set {⟨0, b⟩}
        𝔊(⌊ArrMap⌋) = 0
        𝔊(⌊HomDom⌋) = smc-10 0
        𝔊(⌊HomCod⌋) = 𝔅
        unfolding 𝔊-def dghm-field-simps by (simp-all add: nat-omega-simps)
```

```
    have 𝔊: 𝔊 : smc-10 0 ↦↦_{SMCα} 𝔙
    proof(rule is-semifunctorI, unfold 𝔊-components smc-10-components)
      show vfsequence 𝔊 unfolding 𝔊-def by auto
      show vcard 𝔊 = 4_ℕ
        unfolding 𝔊-def by (auto simp: nat-omega-simps)
      show smcf-dghm 𝔊 : smc-dg (smc-10 0) ↦↦_{DGα} smc-dg 𝔙
      proof(intro is-dghmI, unfold 𝔊-components dg-10-components smc-dg-smc-10)
        show vfsequence (smcf-dghm 𝔊) unfolding smcf-dghm-def by simp
        show vcard (smcf-dghm 𝔊) = 4_ℕ
          unfolding smcf-dghm-def by (simp add: nat-omega-simps)
      qed
        (
          auto simp:
            slicing-simps slicing-intros slicing-commute smc-dg-smc-10
            b 𝔊-components dg-10-is-arr-iff digraph-dg-10
        )
    qed (auto simp: smc-cs-intros smc-10-is-arr-iff b vsubset-vsingleton-leftI)
    then have 𝔊-def: 𝔊 = 𝔉 by (rule 𝔊𝔉)
    have ℛ∘ (𝔊(|ObjMap|)) = set {b} unfolding 𝔊-components by simp
    with vrange-𝔉 ba show False unfolding 𝔊-def by simp
  qed
  ultimately have 𝔙(|Obj|) = set {a} by simp
  with that show ?thesis by simp
qed

obtain f
  where 𝔙-Arr: 𝔙(|Arr|) = set {f}
    and f: f ∈∘ Vset α
    and ff-f: f ∘_{A𝔙} f = f
proof–
  from prems(2)[OF semicategory-smc-1, of 0 0] obtain 𝔉
    where 𝔉 : smc-1 0 0 ↦↦_{SMCα} 𝔙
      and 𝔊 : smc-1 0 0 ↦↦_{SMCα} 𝔙 ⟹ 𝔊 = 𝔉
    for 𝔊
    by fastforce
  then interpret 𝔉: is-semifunctor α ‹smc-1 0 0› 𝔙 𝔉 by force
  have 𝒟∘ (𝔉(|ObjMap|)) = set {0}
    by (simp add: smc-cs-simps smc-1-components)
  then obtain a' where ℛ∘ (𝔉(|ObjMap|)) = set {a'}
    by (auto intro: 𝔉.ObjMap.vsv-vdomain-vsingleton-vrange-vsingleton)
  with 𝔉.smcf-ObjMap-vrange have ℛ∘ (𝔉(|ObjMap|)) = set {a}
    by (auto simp: 𝔙-Obj)
  have vdomain-𝔉: 𝒟∘ (𝔉(|ArrMap|)) = set {0}
    by (simp add: smc-cs-simps smc-1-components)
  then obtain f where vrange-𝔉[simp]: ℛ∘ (𝔉(|ArrMap|)) = set {f}
    by (auto intro: 𝔉.ArrMap.vsv-vdomain-vsingleton-vrange-vsingleton)
  with 𝔙.smc-Arr-vsubset-Vset 𝔉.smcf-ArrMap-vrange have [simp]: f ∈∘ Vset α
    by auto
  from 𝔉.smcf-ArrMap-vrange have f-ss-𝔙: set {f} ⊆∘ 𝔙(|Arr|) by simp
  then have f ∈∘ 𝔙(|Arr|) by auto
  then have f: f : a ↦_𝔙 a
    by (metis 𝔙-Obj 𝔙.smc-is-arrD(2,3) is-arrI vsingleton-iff)
  from vdomain-𝔉 𝔉.ArrMap.vsv-value have [simp]: 𝔉(|ArrMap|)(|0|) = f by auto
  from 𝔉.smcf-is-arr-HomCod(2) have [simp]: 𝔉(|ObjMap|)(|0|) = a
    by (auto simp: smc-1-is-arr-iff 𝔙-Obj)
  have 𝔉(|ArrMap|)(|0|) ∘_{A𝔙} 𝔉(|ArrMap|)(|0|) = 𝔉(|ArrMap|)(|0|)
    by (metis smc-1-Comp-app 𝔉.smcf-ArrMap-Comp smc-1-is-arr-iff)
  then have ff-f[simp]: f ∘_{A𝔙} f = f by simp
```

```
  have id-𝔅: smcf-id 𝔅 : 𝔅 ↦↦_SMCα 𝔅
    by (simp add: 𝔅.smc-smcf-id-is-semifunctor)
  interpret id-𝔅: is-semifunctor α 𝔅 𝔅 ‹smcf-id 𝔅› by (rule id-𝔅)
  from prems(2)[OF 𝔅.semicategory-axioms] have
    𝔊 : 𝔅 ↦↦_SMCα 𝔅 ⟹ 𝔊 = smcf-id 𝔅 for 𝔊
    by (clarsimp simp: id-𝔅.is-semifunctor-axioms)
  moreover from f have smcf-const 𝔅 𝔅 a f : 𝔅 ↦↦_SMCα 𝔅
    by (intro smcf-const-is-semifunctor) (auto intro: smc-cs-intros)
  ultimately have const-eq-id: smcf-const 𝔅 𝔅 a f = smcf-id 𝔅 by simp
  have 𝔅(|Arr|) ⊆∘ set {f}
  proof(rule ccontr)
    assume ¬𝔅(|Arr|) ⊆∘ set {f}
    then obtain g where gf: g ≠ f and g: g ∈∘ 𝔅(|Arr|) by force
    have g: g : a ↦_𝔅 a
    proof(intro is-arrI)
      from g 𝔅-Obj show 𝔅(|Dom|)(|g|) = a
        by (metis 𝔅.smc-is-arrD(2) is-arr-def vsingleton-iff)
      from g 𝔅-Obj show 𝔅(|Cod|)(|g|) = a
        by (metis 𝔅.smc-is-arrD(3) is-arr-def vsingleton-iff)
    qed (auto simp: g)
    then have smcf-const 𝔅 𝔅 a f(|ArrMap|)(|g|) = f
      by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
    moreover from g have smcf-id 𝔅(|ArrMap|)(|g|) = g
      by (cs-concl cs-shallow cs-simp: smc-cs-simps cs-intro: smc-cs-intros)
    ultimately show False using const-eq-id by (simp add: gf)
  qed
  with f-ss-𝔅 have 𝔅(|Arr|) = set {f} by simp
  with that show ?thesis by simp
qed

have 𝔅 = smc-1 a f
proof(rule smc-eqI [of α], unfold smc-1-components)
  show 𝔅(|Obj|) = set {a} by (simp add: 𝔅-Obj)
  moreover show 𝔅(|Arr|) = set {f} by (simp add: 𝔅-Arr)
  ultimately have dom: 𝔅(|Dom|)(|f|) = a and cod: 𝔅(|Cod|)(|f|) = a
    by (metis 𝔅.smc-is-arrE is-arr-def vsingleton-iff)+
  have 𝒟∘ (𝔅(|Dom|)) = set {f} by (simp add: 𝔅-Arr smc-cs-simps)
  moreover from 𝔅.Dom.vsv-vrange-vempty 𝔅.smc-Dom-vdomain 𝔅.smc-Dom-vrange
  have ℛ∘ (𝔅(|Dom|)) = set {a}
    by (fastforce simp: 𝔅-Arr 𝔅-Obj)
  ultimately show 𝔅(|Dom|) = set {⟨f, a⟩}
    using assms 𝔅.Dom.vsv-vdomain-vrange-vsingleton by simp
  have 𝒟∘ (𝔅(|Cod|)) = set {f} by (simp add: 𝔅-Arr smc-cs-simps)
  moreover from 𝔅.Cod.vsv-vrange-vempty 𝔅.smc-Cod-vdomain 𝔅.smc-Cod-vrange
  have ℛ∘ (𝔅(|Cod|)) = set {a}
    by (fastforce simp: 𝔅-Arr 𝔅-Obj)
  ultimately show 𝔅(|Cod|) = set {⟨f, a⟩}
    using assms 𝔅.Cod.vsv-vdomain-vrange-vsingleton by simp
  show 𝔅(|Comp|) = set {⟨[f, f]∘, f⟩}
  proof(rule vsv-eqI)
    show [simp]: 𝒟∘ (𝔅(|Comp|)) = 𝒟∘ (set {⟨[f, f]∘, f⟩})
      unfolding vdomain-vsingleton
    proof(rule vsubset-antisym)
      from 𝔅.Comp.pnop-vdomain show 𝒟∘ (𝔅(|Comp|)) ⊆∘ set {[f, f]∘}
        by (auto simp: 𝔅-Arr intro: smc-cs-intros)
      from 𝔅-Arr dom cod is-arrI show set {[f, f]∘} ⊆∘ 𝒟∘ (𝔅(|Comp|))
        by (metis 𝔅.smc-Comp-vdomainI vsingletonI vsubset-vsingleton-leftI)
    qed
```

```
    from ff-f show a ∈∘ 𝒟∘ (𝔅(|Comp|)) ⟹ 𝔅(|Comp|)(|a|) = set {⟨[f, f]∘, f⟩}(|a|)
      for a
      by simp
  qed auto
qed (auto intro: smc-cs-intros a f semicategory-smc-1)
with a f that show ?thesis by auto

qed
```

Bibliography

[1] Association of Mizar Users. Mizar home page., . URL http://mizar.org/.

[2] Encyclopedia of Mathematics, . URL https://www.encyclopediaofmath.org/index.php/Main_Page.

[3] nLab, . URL https://ncatlab.org/nlab/show/HomePage.

[4] ProofWiki, . URL https://proofwiki.org/wiki/Main_Page.

[5] Wikipedia, 2001. URL https://www.wikipedia.org/.

[6] Isabelle/HOL Standard Library, 2020. URL https://isabelle.in.tum.de/website-Isabelle2020/dist/library/HOL/HOL/index.html.

[7] J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories - The Joy of Cats*. 2006.

[8] C. Ballarin. Locales and Locale Expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs*, volume 3085, pages 34–50. Springer, Heidelberg, 2004. ISBN 978-3-540-22164-7.

[9] C. Ballarin. Tutorial to Locales and Locale Interpretation. 2020. URL https://isabelle.in.tum.de/website-Isabelle2020/dist/Isabelle2020/doc/locales.pdf.

[10] G. Bancerek. Categorial Categories and Slice Categories. *Formalized Mathematics*, 5(2): 157–165, 1996.

[11] G. Bancerek. Indexed Category. *Formalized Mathematics*, 5(3):329–337, 1996.

[12] G. Bancerek. Concrete Categories. *Formalized Mathematics*, 9(3):605–621, 2001.

[13] G. Bancerek and A. Darmochwał. Comma Category. *Formalized Mathematics*, 2(5): 679–681, 1991.

[14] E. D. Bloch. *The Real Numbers and Real Analysis*. Springer Science + Business Media, Heidelberg, 2010. ISBN 978-0-387-72176-7.

[15] P. Bodo. *Categories and Functors*. Academic Press, New York, 1970.

[16] N. Bourbaki. *Elements of Mathematics, Theory of Sets*. Originally published as Éléments de Mathématique Théorie des Ensembles; Paris: N. Bourbaki. Reprint, Heidelberg: Springer-Verlag, 2004., 1970. ISBN 978-3-540-22525-6.

[17] C. E. Brown, C. Kaliszyk, and K. Pak. Higher-Order Tarski Grothendieck as a Foundation for Formal Proof. In J. Harrison, J. O'Leary, and A. Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, pages 9:1–9:16, Portland, USA, 2019.

[18] C. Byliński. Introduction to Categories and Functors. *Formalized Mathematics*, 1(2): 409–420, 1990.

[19] C. Byliński. Subcategories and Products of Categories. *Formalized Mathematics*, 1(4): 725–732, 1990.

[20] C. Byliński. Category Ens. *Formalized Mathematics*, 2(4):527–533, 1991.

[21] C. Byliński. Opposite Categories and Contravariant Functors. *Formalized Mathematics*, 2 (3):419–424, 1991.

[22] C. Byliński. Products and Coproducts in Categories. *Formalized Mathematics*, 2(5): 701–709, 1991.

[23] C. Byliński. Cartesian Categories. *Formalized Mathematics*, 3(2):161–169, 1992.

[24] M. Cáccamo and G. Winskel. A Higher-Order Calculus for Categories. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics*, pages 136–153, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-44755-9. doi: 10.1007/3-540-44755-5_11.

[25] M. J. Cáccamo and G. Winskel. A Higher-Order Calculus for Categories. Technical report, University of Aarhus, Aarhus, Denmark, 2001.

[26] J. Chen, K. Kappelmann, and A. Krauss. HOTG, 2021. URL https://bitbucket.org/cezaryka/tyset/src.

[27] M. Eberl. Syntax proposal: multiway if, 2021. URL https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2021-February/msg00034.html.

[28] 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.

[29] M. Goliński and A. Korniłowicz. Coproducts in Categories without Uniqueness of cod and dom. *Formalized Mathematics*, 21(4):235–239, 2013.

[30] A. Grabowski and Y. Shidama. *Preface*, volume 22. 2014.

[31] F. Haftmann. Sketch-and-Explore, 2021. URL https://isabelle.in.tum.de/library/HOL/HOL-ex/Sketch_and_Explore.html.

[32] T. W. Hungerford. *Algebra*. Springer, New York, 2003. ISBN 978-0-387-90518-1.

[33] F. Kammüller, M. Wenzel, and L. C. Paulson. Locales A Sectioning Concept for Isabelle. In Y. Bertot, G. Dowek, L. Théry, A. Hirschowitz, and C. Paulin-Mohring, editors, *Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics, TPHOLs'99, Nice, France, September, 1999*, Lecture Notes in Computer Science, pages 149–165, Heidelberg, Germany, 1999. Springer. ISBN 978-3-540-48256-7. doi: 10.1007/3-540-48256-3_11.

[34] A. Katovsky. Category Theory. *Archive of Formal Proofs*, 2010.

[35] J. L. Kelley. *General Topology*. Originally published as General Topology; New York, NY, USA: Van Nostrand Reinhold Company. Reprint, Mineola, NY, USA: Dover Publications, 2017, 1955. ISBN 978-0-486-81544-2.

[36] A. Korniłowicz. On the Categories Without Uniqueness of cod and dom. Some Properties of the Morphisms and the Functors. *Formalized Mathematics*, 6(4):475–481, 1997.

[37] A. Korniłowicz. The Composition of Functors and Transformations in Alternative Categories. *Formalized Mathematics*, 7(1):1–7, 1998.

[38] A. Korniłowicz. Products in Categories without Uniqueness of cod and dom. *Formalized Mathematics*, 20(4):303–307, 2012.

[39] 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.

[40] N. Megill and D. A. Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, Morrisville, North Carolina, 2019. ISBN 978-0-359-70223-7.

[41] M. Milehins. Category Theory for ZFC in HOL II: Elementary Theory of 1-Categories. *Archive of Formal Proofs*, 2021.

[42] B. Mitchell. The Dominion of Isbell. *Transactions of the American Mathematical Society*, 167, 1972.

[43] M. Muzalewski. Categories of Groups. *Formalized Mathematics*, 2(4):563–571, 1991.

[44] M. Muzalewski. Category of Rings. *Formalized Mathematics*, 2(5):643–648, 1991.

[45] M. Muzalewski. Category of Left Modules. *Formalized Mathematics*, 2(5):649–652, 1991.

[46] R. Nieszczerzewski. Category of Functors Between Alternative Categories. *Formalized Mathematics*, 6(3):371–375, 1997.

[47] 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.

[48] G. O'Keefe. Category Theory to Yoneda's Lemma. *Archive of Formal Proofs*, 2005.

[49] L. C. Paulson. Natural Deduction as Higher-Order Resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986. doi: 10.1016/0743-1066(86)90015-4.

[50] L. C. Paulson. The Hereditarily Finite Sets. *Archive of Formal Proofs*, 2013.

[51] L. C. Paulson. Zermelo Fraenkel Set Theory in Higher-Order Logic. *Archive of Formal Proofs*, 2019.

[52] V. K. Rao. On Doing Category Theory within Set Theoretic Foundations. In G. Sica, editor, *What is Category Theory?* Polimetrica s.a.s., 2006. ISBN 978-88-7699-031-1.

[53] M. Riccardi. Object-Free Definition of Categories. *Formalized Mathematics*, 21(3):193–205, 2013.

[54] M. Riccardi. Categorical Pullbacks. *Formalized Mathematics*, 23(1):1–14, 2015. doi: 10.2478/forma-2015-0001.

[55] M. Riccardi. Exponential Objects. *Formalized Mathematics*, 23(4):351–369, 2015.

[56] E. Riehl. *Category Theory in Context*. Emily Riehl, 2016.

[57] M. A. Shulman. Set Theory for Category Theory. *arXiv:0810.1279 [math]*, 2008. URL http://arxiv.org/abs/0810.1279.

[58] E. W. Stark. Category Theory with Adjunctions and Limits. *Archive of Formal Proofs*, 2016.

[59] G. Takeuti and W. M. Zaring. *Introduction to Axiomatic Set Theory*. Springer-Verlag, Heidelberg, 1971. ISBN 0-387-05302-6.

[60] A. Trybulec. Isomorphisms of Categories. *Formalized Mathematics*, 2(5):629–634, 1991.

[61] A. Trybulec. Natural Transformations. Discrete Categories. *Formalized Mathematics*, 2(4): 467–474, 1991.

[62] A. Trybulec. Some Isomorphisms Between Functor Categories. *Formalized Mathematics*, 3(1):33–40, 1992.

[63] A. Trybulec. Categories without Uniqueness of cod and dom. *Formalized Mathematics*, 5 (2):259–267, 1996.

[64] A. Trybulec. Functors for Alternative Categories. *Formalized Mathematics*, 5(4):595–608, 1996.