The Elementary Theory of the Category of Sets

James Baxter Dustin Bryant

March 17, 2025

Abstract

Category theory presents a formulation of mathematical structures in terms of common properties of those structures. A particular formulation of interest is the Elementary Theory of the Category of Sets (ETCS), which is an axiomatization of set theory in category theory terms. This axiomatization provides an unusual view of sets, where the functions between sets are regarded as more important than the elements of the sets. We formalise an axiomatization of ETCS on top of HOL, following the presentation given by Halvorson [1]. We also build some other set theoretic results on top of the axiomatization, including Cantor's diagonalization theorem and mathematical induction. We additionally define a system of quantified predicate logic within the ETCS axiomatization.

Contents

1 Basic Types and Operators for the Category of Sets

theory *Cfunc*
 imports *Main HOL−Eisbach.Eisbach*
begin

typedecl *cset*
typedecl *cfunc*

We declare *cset* and *cfunc* as types to represent the sets and functions within ETCS, as distinct from HOL sets and functions. The "c" prefix here is intended to stand for "category", and emphasises that these are category-theoretic objects.

The axiomatization below corresponds to Axiom 1 (Sets Is a Category) in Halvorson.

axiomatization
 domain :: *cfunc* ⇒ *cset* **and**
 codomain :: *cfunc* ⇒ *cset* **and**
 comp :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (**infixr** $\circ_c$ *55*) **and**
 id :: *cset* ⇒ *cfunc* (id_c)
where
 domain-comp: $domain\ g = codomain\ f \Longrightarrow domain\ (g \circ_c f) = domain\ f$ **and**
 codomain-comp: $domain\ g = codomain\ f \Longrightarrow codomain\ (g \circ_c f) = codomain\ g$ **and**
 comp-associative: $domain\ h = codomain\ g \Longrightarrow domain\ g = codomain\ f \Longrightarrow h \circ_c (g \circ_c f) = (h \circ_c g) \circ_c f$ **and**
 id-domain: $domain\ (id\ X) = X$ **and**
 id-codomain: $codomain\ (id\ X) = X$ **and**
 id-right-unit: $f \circ_c id\ (domain\ f) = f$ **and**
 id-left-unit: $id\ (codomain\ f) \circ_c f = f$

We define a neater way of stating types and lift the type axioms into lemmas using it.

definition *cfunc-type* :: *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool* (- : - → - [*50*, *50*, *50*]*50*)
where
($f : X \to Y$) ⟷ (*domain* $f = X \wedge$ *codomain* $f = Y$)

lemma *comp-type*:
$f : X \to Y \Longrightarrow g : Y \to Z \Longrightarrow g \circ_c f : X \to Z$
by (*simp add*: *cfunc-type-def codomain-comp domain-comp*)

lemma *comp-associative2*:
$f : X \to Y \Longrightarrow g : Y \to Z \Longrightarrow h : Z \to W \Longrightarrow h \circ_c (g \circ_c f) = (h \circ_c g) \circ_c f$
by (*simp add*: *cfunc-type-def comp-associative*)

lemma *id-type*: *id* $X : X \to X$
unfolding *cfunc-type-def* **using** *id-domain id-codomain* **by** *auto*

lemma *id-right-unit2*: $f : X \to Y \Longrightarrow f \circ_c$ *id* $X = f$
unfolding *cfunc-type-def* **using** *id-right-unit* **by** *auto*

lemma *id-left-unit2*: $f : X \to Y \Longrightarrow$ *id* $Y \circ_c f = f$
unfolding *cfunc-type-def* **using** *id-left-unit* **by** *auto*

1.1 Tactics for Applying Typing Rules

ETCS lemmas often have assumptions on its ETCS type, which can often be cumbersome to prove. To simplify proofs involving ETCS types, we provide proof methods that apply type rules in a structured way to prove facts about ETCS function types. The type rules state the types of the basic constants and operators of ETCS and are declared as a named set of theorems called *type_rule*.

named-theorems *type-rule*

declare *id-type*[*type-rule*]
declare *comp-type*[*type-rule*]

ML-file ‹*typecheck.ml*›

1.1.1 typecheck_cfuncs: Tactic to Construct Type Facts

method-setup *typecheck-cfuncs* =
‹*Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *typecheck-cfuncs-method*›
Check types of cfuncs in current goal and add as assumptions of the current goal

method-setup *typecheck-cfuncs-all* =
‹*Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *typecheck-cfuncs-all-method*›
Check types of cfuncs in all subgoals and add as assumptions of the current goal

method-setup *typecheck-cfuncs-prems* =
‹*Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *typecheck-cfuncs-prems-method*›
Check types of cfuncs in assumptions of the current goal and add as assumptions of the current goal

1.1.2 etcs_rule: Tactic to Apply Rules with ETCS Typechecking

method-setup *etcs-rule* =
‹*Scan.repeats* (*Scan.unless* (*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) *Attrib.multi-thm*)
−− *Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *ETCS-resolve-method*›
apply rule with ETCS type checking

1.1.3 etcs_subst: Tactic to Apply Substitutions with ETCS Typechecking

method-setup *etcs-subst* =
‹*Scan.repeats* (*Scan.unless* (*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) *Attrib.multi-thm*)
−− *Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *ETCS-subst-method*›
apply substitution with ETCS type checking

method *etcs-assocl* **declares** *type-rule* = (*etcs-subst comp-associative2*)+
method *etcs-assocr* **declares** *type-rule* = (*etcs-subst sym*[*OF comp-associative2*])+

method-setup *etcs-subst-asm* =
‹*Runtime.exn-trace* (*fn* - => *Scan.repeats* (*Scan.unless* (*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) *Attrib.multi-thm*)
−− *Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *ETCS-subst-asm-method*)›
apply substitution to assumptions of the goal, with ETCS type checking

method *etcs-assocl-asm* **declares** *type-rule* = (*etcs-subst-asm comp-associative2*)+
method *etcs-assocr-asm* **declares** *type-rule* = (*etcs-subst-asm sym*[*OF comp-associative2*])+

1.1.4 etcs_erule: Tactic to Apply Elimination Rules with ETCS Typechecking

method-setup *etcs-erule* =
‹*Scan.repeats* (*Scan.unless* (*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) *Attrib.multi-thm*)
−− *Scan.option* ((*Scan.lift* (*Args.$$$ type-rule* −− *Args.colon*)) |−− *Attrib.thms*)
>> *ETCS-eresolve-method*›
apply erule with ETCS type checking

1.2 Monomorphisms, Epimorphisms and Isomorphisms

1.2.1 Monomorphisms

definition *monomorphism* :: *cfunc* ⇒ *bool* **where**
monomorphism f ⟷ (∀ *g h*.
(*codomain g* = *domain f* ∧ *codomain h* = *domain f*) ⟶ (*f* $\circ_c$ *g* = *f* $\circ_c$ *h* ⟶ *g* = *h*))

lemma *monomorphism-def2*:
monomorphism f ⟷ (∀ *g h A X Y*. *g* : *A* → *X* ∧ *h* : *A* → *X* ∧ *f* : *X* → *Y* ⟶ (*f* $\circ_c$ *g* = *f* $\circ_c$ *h* ⟶ *g* = *h*))
unfolding *monomorphism-def* **by** (*smt cfunc-type-def domain-comp*)

lemma *monomorphism-def3*:
assumes *f* : *X* → *Y*
shows *monomorphism f* ⟷ (∀ *g h A*. *g* : *A* → *X* ∧ *h* : *A* → *X* ⟶ (*f* $\circ_c$ *g* = *f* $\circ_c$ *h* ⟶ *g* = *h*))
unfolding *monomorphism-def2* **using** *assms cfunc-type-def* **by** *auto*

The lemma below corresponds to Exercise 2.1.7a in Halvorson.

lemma *comp-monic-imp-monic*:
assumes *domain g* = *codomain f*
shows *monomorphism* (*g* $\circ_c$ *f*) ⟹ *monomorphism f*
unfolding *monomorphism-def*
proof *clarify*
fix *s t*
assume *gf-monic*: ∀ *s*. ∀ *t*.
codomain s = *domain* (*g* $\circ_c$ *f*) ∧ *codomain t* = *domain* (*g* $\circ_c$ *f*) ⟶
(*g* $\circ_c$ *f*) $\circ_c$ *s* = (*g* $\circ_c$ *f*) $\circ_c$ *t* ⟶ *s* = *t*
assume *codomain-s*: *codomain s* = *domain f*
assume *codomain-t*: *codomain t* = *domain f*
assume *f* $\circ_c$ *s* = *f* $\circ_c$ *t*

then have (*g* $\circ_c$ *f*) $\circ_c$ *s* = (*g* $\circ_c$ *f*) $\circ_c$ *t*
by (*metis assms codomain-s codomain-t comp-associative*)
then show *s* = *t*
using *gf-monic codomain-s codomain-t domain-comp* **by** (*simp add*: *assms*)
qed

lemma *comp-monic-imp-monic'*:
assumes *f* : *X* → *Y* *g* : *Y* → *Z*
shows *monomorphism* (*g* $\circ_c$ *f*) ⟹ *monomorphism f*
by (*metis assms cfunc-type-def comp-monic-imp-monic*)

The lemma below corresponds to Exercise 2.1.7c in Halvorson.

lemma *composition-of-monic-pair-is-monic*:
assumes *codomain f* = *domain g*
shows *monomorphism f* ⟹ *monomorphism g* ⟹ *monomorphism* (*g* $\circ_c$ *f*)
unfolding *monomorphism-def*

proof *clarify*
fix *h k*
assume *f-mono*: $\forall\, s\ t.$
codomain s = *domain* $f \wedge$ *codomain* t = *domain* $f \longrightarrow f \circ_c s = f \circ_c t \longrightarrow s = t$
assume *g-mono*: $\forall\, s.\ \forall\, t.$
codomain s = *domain* $g \wedge$ *codomain* t = *domain* $g \longrightarrow g \circ_c s = g \circ_c t \longrightarrow s = t$
assume *codomain-k*: *codomain* k = *domain* $(g \circ_c f)$
assume *codomain-h*: *codomain* h = *domain* $(g \circ_c f)$
assume *gfh-eq-gfk*: $(g \circ_c f) \circ_c k = (g \circ_c f) \circ_c h$

have $g \circ_c (f \circ_c h) = (g \circ_c f) \circ_c h$
by (*simp add*: *assms codomain-h comp-associative domain-comp*)
also have ... $= (g \circ_c f) \circ_c k$
by (*simp add*: *gfh-eq-gfk*)
also have ... $= g \circ_c (f \circ_c k)$
by (*simp add*: *assms codomain-k comp-associative domain-comp*)
ultimately have $f \circ_c h = f \circ_c k$
using *assms cfunc-type-def codomain-h codomain-k comp-type domain-comp g-mono* **by** *auto*
then show $k = h$
by (*simp add*: *codomain-h codomain-k domain-comp f-mono assms*)
qed

1.2.2 Epimorphisms

definition *epimorphism* :: *cfunc* $\Rightarrow$ *bool* **where**
epimorphism $f \longleftrightarrow (\forall\ g\ h.$
(*domain* g = *codomain* $f \wedge$ *domain* h = *codomain* f) $\longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$

lemma *epimorphism-def2*:
epimorphism $f \longleftrightarrow (\forall\ g\ h\ A\ X\ Y.\ f : X \rightarrow Y \wedge g : Y \rightarrow A \wedge h : Y \rightarrow A \longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$
unfolding *epimorphism-def* **by** (*smt cfunc-type-def codomain-comp*)

lemma *epimorphism-def3*:
assumes $f : X \rightarrow Y$
shows *epimorphism* $f \longleftrightarrow (\forall\ g\ h\ A.\ g : Y \rightarrow A \wedge h : Y \rightarrow A \longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$
unfolding *epimorphism-def2* **using** *assms cfunc-type-def* **by** *auto*

The lemma below corresponds to Exercise 2.1.7b in Halvorson.

lemma *comp-epi-imp-epi*:
assumes *domain* g = *codomain* f
shows *epimorphism* $(g \circ_c f) \Longrightarrow$ *epimorphism* g
unfolding *epimorphism-def*
proof *clarify*
fix *s t*

assume *gf-epi*: $\forall s. \forall t.$
domain $s =$ *codomain* $(g \circ_c f) \wedge$ *domain* $t =$ *codomain* $(g \circ_c f) \longrightarrow$
$s \circ_c g \circ_c f = t \circ_c g \circ_c f \longrightarrow s = t$
assume *domain-s*: *domain* $s =$ *codomain* g
assume *domain-t*: *domain* $t =$ *codomain* g
assume *sf-eq-tf*: $s \circ_c g = t \circ_c g$

from *sf-eq-tf* **have** $s \circ_c (g \circ_c f) = t \circ_c (g \circ_c f)$
by (*simp add*: *assms comp-associative domain-s domain-t*)
then show $s = t$
using *gf-epi codomain-comp domain-s domain-t* **by** (*simp add*: *assms*)
qed

The lemma below corresponds to Exercise 2.1.7d in Halvorson.

lemma *composition-of-epi-pair-is-epi*:
assumes *codomain* $f =$ *domain* g
shows *epimorphism* $f \Longrightarrow$ *epimorphism* $g \Longrightarrow$ *epimorphism* $(g \circ_c f)$
unfolding *epimorphism-def*
proof *clarify*
fix h k
assume *f-epi* :$\forall$ s h.
(*domain* $s =$ *codomain* $f \wedge$ *domain* $h =$ *codomain* f) $\longrightarrow$ $(s \circ_c f = h \circ_c f \longrightarrow s = h)$
assume *g-epi* :$\forall$ s h.
(*domain* $s =$ *codomain* $g \wedge$ *domain* $h =$ *codomain* g) $\longrightarrow$ $(s \circ_c g = h \circ_c g \longrightarrow s = h)$
assume *domain-k*: *domain* $k =$ *codomain* $(g \circ_c f)$
assume *domain-h*: *domain* $h =$ *codomain* $(g \circ_c f)$
assume *hgf-eq-kgf*: $h \circ_c (g \circ_c f) = k \circ_c (g \circ_c f)$

have $(h \circ_c g) \circ_c f = h \circ_c (g \circ_c f)$
by (*simp add*: *assms codomain-comp comp-associative domain-h*)
also have ... $= k \circ_c (g \circ_c f)$
by (*simp add*: *hgf-eq-kgf*)
also have ... $=(k \circ_c g) \circ_c f$
by (*simp add*: *assms codomain-comp comp-associative domain-k*)
ultimately have $h \circ_c g = k \circ_c g$
by (*simp add*: *assms codomain-comp domain-comp domain-h domain-k f-epi*)
then show $h = k$
by (*simp add*: *codomain-comp domain-h domain-k g-epi assms*)
qed

1.2.3 Isomorphisms

definition *isomorphism* :: *cfunc* $\Rightarrow$ *bool* **where**
isomorphism $f \longleftrightarrow (\exists$ g. *domain* $g =$ *codomain* $f \wedge$ *codomain* $g =$ *domain* $f \wedge$
$g \circ_c f = id($*domain* $f) \wedge f \circ_c g = id($*domain* $g))$

lemma *isomorphism-def2*:

isomorphism f $\longleftrightarrow$ ($\exists$ *g X Y*. *f* : *X* $\rightarrow$ *Y* $\wedge$ *g* : *Y* $\rightarrow$ *X* $\wedge$ *g* $\circ_c$ *f* = *id X* $\wedge$ *f* $\circ_c$ *g* = *id Y*)
unfolding *isomorphism-def cfunc-type-def* **by** *auto*

lemma *isomorphism-def3*:
assumes *f* : *X* $\rightarrow$ *Y*
shows *isomorphism f* $\longleftrightarrow$ ($\exists$ *g*. *g* : *Y* $\rightarrow$ *X* $\wedge$ *g* $\circ_c$ *f* = *id X* $\wedge$ *f* $\circ_c$ *g* = *id Y*)
using *assms* **unfolding** *isomorphism-def2 cfunc-type-def* **by** *auto*

definition *inverse* :: *cfunc* $\Rightarrow$ *cfunc* (-$^{\mathbf{-1}}$ [*1000*] *999*) **where**
inverse f = (*THE g*. *g* : *codomain f* $\rightarrow$ *domain f* $\wedge$ *g* $\circ_c$ *f* = *id*(*domain f*) $\wedge$ *f* $\circ_c$ *g* = *id*(*codomain f*))

lemma *inverse-def2*:
assumes *isomorphism f*
shows $f^{\mathbf{-1}}$: *codomain f* $\rightarrow$ *domain f* $\wedge$ $f^{\mathbf{-1}}$ $\circ_c$ *f* = *id*(*domain f*) $\wedge$ *f* $\circ_c$ $f^{\mathbf{-1}}$ = *id*(*codomain f*)
unfolding *inverse-def*
proof (*rule theI'*, *safe*)
show $\exists$ *g*. *g* : *codomain f* $\rightarrow$ *domain f* $\wedge$ *g* $\circ_c$ *f* = id_c (*domain f*) $\wedge$ *f* $\circ_c$ *g* = id_c (*codomain f*)
using *assms* **unfolding** *isomorphism-def cfunc-type-def* **by** *auto*
next
fix *g1 g2*
assume *g1-f*: *g1* $\circ_c$ *f* = id_c (*domain f*) **and** *f-g1*: *f* $\circ_c$ *g1* = id_c (*codomain f*)
assume *g2-f*: *g2* $\circ_c$ *f* = id_c (*domain f*) **and** *f-g2*: *f* $\circ_c$ *g2* = id_c (*codomain f*)
assume *g1* : *codomain f* $\rightarrow$ *domain f g2* : *codomain f* $\rightarrow$ *domain f*
then have *codomain g1* = *domain f domain g2* = *codomain f*
unfolding *cfunc-type-def* **by** *auto*
then show *g1* = *g2*
by (*metis comp-associative f-g1 g2-f id-left-unit id-right-unit*)
qed

lemma *inverse-type*[*type-rule*]:
assumes *isomorphism f f* : *X* $\rightarrow$ *Y*
shows $f^{\mathbf{-1}}$: *Y* $\rightarrow$ *X*
using *assms inverse-def2* **unfolding** *cfunc-type-def* **by** *auto*

lemma *inv-left*:
assumes *isomorphism f f* : *X* $\rightarrow$ *Y*
shows $f^{\mathbf{-1}}$ $\circ_c$ *f* = *id X*
using *assms inverse-def2* **unfolding** *cfunc-type-def* **by** *auto*

lemma *inv-right*:
assumes *isomorphism f f* : *X* $\rightarrow$ *Y*
shows *f* $\circ_c$ $f^{\mathbf{-1}}$ = *id Y*
using *assms inverse-def2* **unfolding** *cfunc-type-def* **by** *auto*

lemma *inv-iso*:

assumes *isomorphism f*
shows *isomorphism*(f^{-1})
using *assms inverse-def2* **unfolding** *isomorphism-def cfunc-type-def* **by** (*intro exI*[**where** *x*=*f*], *auto*)

lemma *inv-idempotent*:
assumes *isomorphism f*
shows $(f^{-1})^{-1} = f$
by (*smt assms cfunc-type-def comp-associative id-left-unit inv-iso inverse-def2*)

definition *is-isomorphic* :: *cset* ⇒ *cset* ⇒ *bool* (**infix** ≅ *50*) **where**
$X \cong Y \longleftrightarrow (\exists\ f.\ f : X \to Y \land$ *isomorphism f*)

lemma *id-isomorphism*: *isomorphism* (*id X*)
unfolding *isomorphism-def*
by (*intro exI*[**where** *x*= *id X*], *auto simp add*: *id-codomain id-domain*, *metis id-domain id-right-unit*)

lemma *isomorphic-is-reflexive*: $X \cong X$
unfolding *is-isomorphic-def*
by (*intro exI*[**where** *x*= *id X*], *auto simp add*: *id-domain id-codomain id-isomorphism id-type*)

lemma *isomorphic-is-symmetric*: $X \cong Y \longrightarrow Y \cong X$
unfolding *is-isomorphic-def isomorphism-def*
by (*auto*, *metis cfunc-type-def*)

lemma *isomorphism-comp*:
domain f = *codomain g* ⟹ *isomorphism f* ⟹ *isomorphism g* ⟹ *isomorphism* $(f \circ_c g)$
unfolding *isomorphism-def* **by** (*auto*, *smt codomain-comp comp-associative domain-comp id-right-unit*)

lemma *isomorphism-comp*′:
assumes $f : Y \to Z$ $g : X \to Y$
shows *isomorphism f* ⟹ *isomorphism g* ⟹ *isomorphism* $(f \circ_c g)$
using *assms cfunc-type-def isomorphism-comp* **by** *auto*

lemma *isomorphic-is-transitive*: $(X \cong Y \land Y \cong Z) \longrightarrow X \cong Z$
unfolding *is-isomorphic-def* **by** (*auto*, *metis cfunc-type-def comp-type isomorphism-comp*)

lemma *is-isomorphic-equiv*:
equiv UNIV $\{(X,\ Y).\ X \cong Y\}$
unfolding *equiv-def*
proof *safe*
show *refl* $\{(x,\ y).\ x \cong y\}$
unfolding *refl-on-def* **using** *isomorphic-is-reflexive* **by** *auto*
next

```
  show sym {(x, y). x ≅ y}
    unfolding sym-def using isomorphic-is-symmetric by blast
next
  show trans {(x, y). x ≅ y}
    unfolding trans-def using isomorphic-is-transitive by blast
qed
```

The lemma below corresponds to Exercise 2.1.7e in Halvorson.

```
lemma iso-imp-epi-and-monic:
  isomorphism f ⟹ epimorphism f ∧ monomorphism f
  unfolding isomorphism-def epimorphism-def monomorphism-def
proof safe
  fix g s t
  assume domain-g: domain g = codomain f
  assume codomain-g: codomain g = domain f
  assume gf-id: g ∘c f = id (domain f)
  assume fg-id: f ∘c g = id (domain g)
  assume domain-s: domain s = codomain f
  assume domain-t: domain t = codomain f
  assume sf-eq-tf: s ∘c f = t ∘c f

  have s = s ∘c id(codomain(f))
    by (metis domain-s id-right-unit)
  also have ... = s ∘c (f ∘c g)
    by (simp add: domain-g fg-id)
  also have ... = (s ∘c f) ∘c g
    by (simp add: codomain-g comp-associative domain-s)
  also have ... = (t ∘c f) ∘c g
    by (simp add: sf-eq-tf)
  also have ... = t ∘c (f ∘c g)
    by (simp add: codomain-g comp-associative domain-t)
  also have ... = t ∘c id(codomain f)
    by (simp add: domain-g fg-id)
  also have ... = t
    by (metis domain-t id-right-unit)
  finally show s = t.
next
  fix g h k
  assume domain-g: domain g = codomain f
  assume codomain-g: codomain g = domain f
  assume gf-id: g ∘c f = id (domain f)
  assume fg-id: f ∘c g = id (domain g)
  assume codomain-h: codomain h = domain f
  assume codomain-k: codomain k = domain f
  assume fk-eq-fh: f ∘c k = f ∘c h

  have h = id(domain f) ∘c h
    by (metis codomain-h id-left-unit)
  also have ... = (g ∘c f) ∘c h
```

using *gf-id* **by** *auto*
also have ... = $g \circ_c (f \circ_c h)$
by (*simp add*: *codomain-h comp-associative domain-g*)
also have ... = $g \circ_c (f \circ_c k)$
by (*simp add*: *fk-eq-fh*)
also have ... = $(g \circ_c f) \circ_c k$
by (*simp add*: *codomain-k comp-associative domain-g*)
also have ... = $id(domain\ f) \circ_c k$
by (*simp add*: *gf-id*)
also have ... = k
by (*metis codomain-k id-left-unit*)
ultimately show $k = h$
by *simp*
qed

lemma *isomorphism-sandwich*:
assumes *f-type*: $f : A \to B$ **and** *g-type*: $g : B \to C$ **and** *h-type*: $h: C \to D$
assumes *f-iso*: *isomorphism f*
assumes *h-iso*: *isomorphism h*
assumes *hgf-iso*: *isomorphism*$(h \circ_c g \circ_c f)$
shows *isomorphism g*
proof −
have *isomorphism*$(h^{-1} \circ_c (h \circ_c g \circ_c f) \circ_c f^{-1})$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *f-iso h-iso hgf-iso inv-iso isomorphism-comp'*)
then show *isomorphism g*
using *assms* **by** (*typecheck-cfuncs-prems*, *smt comp-associative2 id-left-unit2 id-right-unit2 inv-left inv-right*)
qed

end

2 Cartesian Products of Sets

theory *Product*
imports *Cfunc*
begin

The axiomatization below corresponds to Axiom 2 (Cartesian Products) in Halvorson.

axiomatization
cart-prod :: *cset* ⇒ *cset* ⇒ *cset* (**infixr** $\times_c$ *65*) **and**
left-cart-proj :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
right-cart-proj :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
cfunc-prod :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (⟨-,-⟩)
where
left-cart-proj-type[*type-rule*]: *left-cart-proj X Y* : $X \times_c Y \to X$ **and**
right-cart-proj-type[*type-rule*]: *right-cart-proj X Y* : $X \times_c Y \to Y$ **and**

cfunc-prod-type[*type-rule*]: $f : Z \to X \Longrightarrow g : Z \to Y \Longrightarrow \langle f,g\rangle : Z \to X \times_c Y$
and
left-cart-proj-cfunc-prod: $f : Z \to X \Longrightarrow g : Z \to Y \Longrightarrow$ *left-cart-proj* $X\ Y \circ_c \langle f,g\rangle = f$ **and**
right-cart-proj-cfunc-prod: $f : Z \to X \Longrightarrow g : Z \to Y \Longrightarrow$ *right-cart-proj* $X\ Y \circ_c \langle f,g\rangle = g$ **and**
cfunc-prod-unique: $f : Z \to X \Longrightarrow g : Z \to Y \Longrightarrow h : Z \to X \times_c Y \Longrightarrow$
left-cart-proj $X\ Y \circ_c h = f \Longrightarrow$ *right-cart-proj* $X\ Y \circ_c h = g \Longrightarrow h = \langle f,g\rangle$

definition *is-cart-prod* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool* **where**
is-cart-prod $W\ \pi_0\ \pi_1\ X\ Y \longleftrightarrow$
$(\pi_0 : W \to X \wedge \pi_1 : W \to Y \wedge$
$(\forall\ f\ g\ Z.\ (f : Z \to X \wedge g : Z \to Y) \longrightarrow$
$(\exists\ h.\ h : Z \to W \wedge \pi_0 \circ_c h = f \wedge \pi_1 \circ_c h = g \wedge$
$(\forall\ h2.\ (h2 : Z \to W \wedge \pi_0 \circ_c h2 = f \wedge \pi_1 \circ_c h2 = g) \longrightarrow h2 = h))))$

lemma *is-cart-prod-def2*:
assumes $\pi_0 : W \to X\ \pi_1 : W \to Y$
shows *is-cart-prod* $W\ \pi_0\ \pi_1\ X\ Y \longleftrightarrow$
$(\forall\ f\ g\ Z.\ (f : Z \to X \wedge g : Z \to Y) \longrightarrow$
$(\exists\ h.\ h : Z \to W \wedge \pi_0 \circ_c h = f \wedge \pi_1 \circ_c h = g \wedge$
$(\forall\ h2.\ (h2 : Z \to W \wedge \pi_0 \circ_c h2 = f \wedge \pi_1 \circ_c h2 = g) \longrightarrow h2 = h)))$
unfolding *is-cart-prod-def* **using** *assms* **by** *auto*

abbreviation *is-cart-prod-triple* :: *cset* × *cfunc* × *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool*
where
is-cart-prod-triple $W\pi\ X\ Y \equiv$ *is-cart-prod* (*fst* $W\pi$) (*fst* (*snd* $W\pi$)) (*snd* (*snd* $W\pi$)) $X\ Y$

lemma *canonical-cart-prod-is-cart-prod*:
is-cart-prod $(X \times_c Y)$ (*left-cart-proj* $X\ Y$) (*right-cart-proj* $X\ Y$) $X\ Y$
unfolding *is-cart-prod-def*
proof (*typecheck-cfuncs*)
fix $f\ g\ Z$
assume *f-type*: f: $Z \to X$
assume *g-type*: g: $Z \to Y$
show $\exists h.\ h : Z \to X \times_c Y \wedge$
left-cart-proj $X\ Y \circ_c h = f \wedge$
right-cart-proj $X\ Y \circ_c h = g \wedge$
$(\forall h2.\ h2 : Z \to X \times_c Y \wedge$
left-cart-proj $X\ Y \circ_c h2 = f \wedge$ *right-cart-proj* $X\ Y \circ_c h2 = g \longrightarrow$
$h2 = h)$
using *f-type g-type left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod cfunc-prod-unique*
by (*intro exI*[**where** $x=\langle f,g\rangle$], *simp add*: *cfunc-prod-type*)
qed

The lemma below corresponds to Proposition 2.1.8 in Halvorson.

lemma *cart-prods-isomorphic*:
assumes *W-cart-prod*: *is-cart-prod-triple* $(W, \pi_0, \pi_1)\ X\ Y$

assumes W'*-cart-prod*: *is-cart-prod-triple* (W', π'_0, π'_1) X Y
shows $\exists f.\ f : W \to W' \wedge$ *isomorphism* $f \wedge \pi'_0 \circ_c f = \pi_0 \wedge \pi'_1 \circ_c f = \pi_1$
proof −
obtain f **where** *f-def*: $f : W \to W' \wedge \pi'_0 \circ_c f = \pi_0 \wedge \pi'_1 \circ_c f = \pi_1$
using W'*-cart-prod* *W-cart-prod* **unfolding** *is-cart-prod-def* **by** (*metis fstI sndI*)

obtain g **where** *g-def*: $g : W' \to W \wedge \pi_0 \circ_c g = \pi'_0 \wedge \pi_1 \circ_c g = \pi'_1$
using W'*-cart-prod* *W-cart-prod* **unfolding** *is-cart-prod-def* **by** (*metis fstI sndI*)

have *fg0*: $\pi'_0 \circ_c (f \circ_c g) = \pi'_0$
using W'*-cart-prod* *comp-associative2* *f-def* *g-def* *is-cart-prod-def* **by** *auto*
have *fg1*: $\pi'_1 \circ_c (f \circ_c g) = \pi'_1$
using W'*-cart-prod* *comp-associative2* *f-def* *g-def* *is-cart-prod-def* **by** *auto*

obtain idW' **where** idW' : $W' \to W' \wedge (\forall\ h2.\ (h2 : W' \to W' \wedge \pi'_0 \circ_c h2 = \pi'_0 \wedge \pi'_1 \circ_c h2 = \pi'_1) \longrightarrow h2 = idW')$
using W'*-cart-prod* **unfolding** *is-cart-prod-def* **by** (*metis fst-conv snd-conv*)
then have *fg*: $f \circ_c g = id\ W'$
proof *clarify*
assume idW'*-unique*: $\forall h2.\ h2 : W' \to W' \wedge \pi'_0 \circ_c h2 = \pi'_0 \wedge \pi'_1 \circ_c h2 = \pi'_1 \longrightarrow h2 = idW'$
have *1*: $f \circ_c g = idW'$
using *comp-type* *f-def* *fg0* *fg1* *g-def* idW'*-unique* **by** *blast*
have *2*: $id\ W' = idW'$
using W'*-cart-prod* idW'*-unique* *id-right-unit2* *id-type* *is-cart-prod-def* **by** *auto*
from *1 2* **show** $f \circ_c g = id\ W'$
by *auto*
qed

have *gf0*: $\pi_0 \circ_c (g \circ_c f) = \pi_0$
using *W-cart-prod* *comp-associative2* *f-def* *g-def* *is-cart-prod-def* **by** *auto*
have *gf1*: $\pi_1 \circ_c (g \circ_c f) = \pi_1$
using *W-cart-prod* *comp-associative2* *f-def* *g-def* *is-cart-prod-def* **by** *auto*

obtain idW **where** idW : $W \to W \wedge (\forall\ h2.\ (h2 : W \to W \wedge \pi_0 \circ_c h2 = \pi_0 \wedge \pi_1 \circ_c h2 = \pi_1) \longrightarrow h2 = idW)$
using *W-cart-prod* **unfolding** *is-cart-prod-def* **by** (*metis fst-conv snd-conv*)
then have *gf*: $g \circ_c f = id\ W$
proof *clarify*
assume *idW-unique*: $\forall h2.\ h2 : W \to W \wedge \pi_0 \circ_c h2 = \pi_0 \wedge \pi_1 \circ_c h2 = \pi_1 \longrightarrow h2 = idW$
have *1*: $g \circ_c f = idW$
using *idW-unique* *cfunc-type-def* *codomain-comp* *domain-comp* *f-def* *gf0* *gf1* *g-def* **by** *auto*
have *2*: $id\ W = idW$
using *idW-unique* *W-cart-prod* *id-right-unit2* *id-type* *is-cart-prod-def* **by** *auto*
from *1 2* **show** $g \circ_c f = id\ W$

by *auto*
qed

have *f-iso*: *isomorphism f*
using *f-def fg g-def gf isomorphism-def3* **by** *blast*
from *f-iso f-def* **show** $\exists f.\ f : W \to W' \wedge$ *isomorphism f* $\wedge \pi'_0 \circ_c f = \pi_0 \wedge \pi'_1 \circ_c f = \pi_1$
by *auto*
qed

lemma *product-commutes*:
$A \times_c B \cong B \times_c A$
proof −
have *id-AB*: ⟨*right-cart-proj* B A, *left-cart-proj* B A⟩ $\circ_c$ ⟨*right-cart-proj* A B, *left-cart-proj* A B⟩ = *id*$(A \times_c B)$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-unique comp-associative2 id-right-unit2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
have *id-BA*: ⟨*right-cart-proj* A B, *left-cart-proj* A B⟩ $\circ_c$ ⟨*right-cart-proj* B A, *left-cart-proj* B A⟩ = *id*$(B \times_c A)$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-unique comp-associative2 id-right-unit2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
show $A \times_c B \cong B \times_c A$
by (*smt* (*verit*, *ccfv-threshold*) *canonical-cart-prod-is-cart-prod cfunc-prod-unique cfunc-type-def id-AB id-BA is-cart-prod-def is-isomorphic-def isomorphism-def*)
qed

lemma *cart-prod-eq*:
assumes $a : Z \to X \times_c Y$ $b : Z \to X \times_c Y$
shows $a = b \longleftrightarrow$
(*left-cart-proj* X Y $\circ_c a =$ *left-cart-proj* X Y $\circ_c b$
$\wedge$ *right-cart-proj* X Y $\circ_c a =$ *right-cart-proj* X Y $\circ_c b$)
by (*metis* (*full-types*) *assms cfunc-prod-unique comp-type left-cart-proj-type right-cart-proj-type*)

lemma *cart-prod-eqI*:
assumes $a : Z \to X \times_c Y$ $b : Z \to X \times_c Y$
assumes (*left-cart-proj* X Y $\circ_c a =$ *left-cart-proj* X Y $\circ_c b$
$\wedge$ *right-cart-proj* X Y $\circ_c a =$ *right-cart-proj* X Y $\circ_c b$)
shows $a = b$
using *assms cart-prod-eq* **by** *blast*

lemma *cart-prod-eq2*:
assumes $a : Z \to X$ $b : Z \to Y$ $c : Z \to X$ $d : Z \to Y$
shows $\langle a, b\rangle = \langle c,d\rangle \longleftrightarrow (a = c \wedge b = d)$
by (*metis assms left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)

lemma *cart-prod-decomp*:
assumes $a : A \to X \times_c Y$
shows $\exists\ x\ y.\ a = \langle x, y\rangle \wedge x : A \to X \wedge y : A \to Y$
proof (*rule exI*[**where** *x*=*left-cart-proj* X Y $\circ_c a$], *intro exI* [**where** *x*=*right-cart-proj*

X Y $\circ_c$ *a*], *safe*)
show $a = \langle$*left-cart-proj X Y* $\circ_c$ *a*,*right-cart-proj X Y* $\circ_c$ *a*$\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-unique*)
show *left-cart-proj X Y* $\circ_c$ *a* : *A* $\rightarrow$ *X*
using *assms* **by** *typecheck-cfuncs*
show *right-cart-proj X Y* $\circ_c$ *a* : *A* $\rightarrow$ *Y*
using *assms* **by** *typecheck-cfuncs*
qed

2.1 Diagonal Functions

The definition below corresponds to Definition 2.1.9 in Halvorson.

definition *diagonal* :: *cset* $\Rightarrow$ *cfunc* **where**
diagonal X = $\langle$*id X*,*id X*$\rangle$

lemma *diagonal-type*[*type-rule*]:
diagonal X : *X* $\rightarrow$ *X* $\times_c$ *X*
unfolding *diagonal-def* **by** (*simp add*: *cfunc-prod-type id-type*)

lemma *diag-mono*:
monomorphism(*diagonal X*)
proof −
have *left-cart-proj X X* $\circ_c$ *diagonal X* = *id X*
by (*metis diagonal-def id-type left-cart-proj-cfunc-prod*)
then show *monomorphism*(*diagonal X*)
by (*metis cfunc-type-def comp-monic-imp-monic diagonal-type id-isomorphism iso-imp-epi-and-monic left-cart-proj-type*)
qed

2.2 Products of Functions

The definition below corresponds to Definition 2.1.10 in Halvorson.

definition *cfunc-cross-prod* :: *cfunc* $\Rightarrow$ *cfunc* $\Rightarrow$ *cfunc* (**infixr** $\times_f$ *55*) **where**
f $\times_f$ *g* = $\langle$*f* $\circ_c$ *left-cart-proj* (*domain f*) (*domain g*), *g* $\circ_c$ *right-cart-proj* (*domain f*) (*domain g*)$\rangle$

lemma *cfunc-cross-prod-def2*:
assumes *f* : *X* $\rightarrow$ *Y g* : *V* $\rightarrow$ *W*
shows *f* $\times_f$ *g* = $\langle$*f* $\circ_c$ *left-cart-proj X V*, *g* $\circ_c$ *right-cart-proj X V*$\rangle$
using *assms cfunc-cross-prod-def cfunc-type-def* **by** *auto*

lemma *cfunc-cross-prod-type*[*type-rule*]:
f : *W* $\rightarrow$ *Y* $\Longrightarrow$ *g* : *X* $\rightarrow$ *Z* $\Longrightarrow$ *f* $\times_f$ *g* : *W* $\times_c$ *X* $\rightarrow$ *Y* $\times_c$ *Z*
unfolding *cfunc-cross-prod-def*
using *cfunc-prod-type cfunc-type-def comp-type left-cart-proj-type right-cart-proj-type*
by *auto*

lemma *left-cart-proj-cfunc-cross-prod*:

$f : W \to Y \Longrightarrow g : X \to Z \Longrightarrow$ *left-cart-proj* $Y\ Z \circ_c f \times_f g = f \circ_c$ *left-cart-proj* $W\ X$
unfolding *cfunc-cross-prod-def*
using *cfunc-type-def comp-type left-cart-proj-cfunc-prod left-cart-proj-type right-cart-proj-type*
by (*smt* (*verit*))

lemma *right-cart-proj-cfunc-cross-prod*:
$f : W \to Y \Longrightarrow g : X \to Z \Longrightarrow$ *right-cart-proj* $Y\ Z \circ_c f \times_f g = g \circ_c$ *right-cart-proj* $W\ X$
unfolding *cfunc-cross-prod-def*
using *cfunc-type-def comp-type right-cart-proj-cfunc-prod left-cart-proj-type right-cart-proj-type*
by (*smt* (*verit*))

lemma *cfunc-cross-prod-unique*: $f : W \to Y \Longrightarrow g : X \to Z \Longrightarrow h : W \times_c X \to Y \times_c Z \Longrightarrow$
left-cart-proj $Y\ Z \circ_c h = f \circ_c$ *left-cart-proj* $W\ X \Longrightarrow$
right-cart-proj $Y\ Z \circ_c h = g \circ_c$ *right-cart-proj* $W\ X \Longrightarrow h = f \times_f g$
unfolding *cfunc-cross-prod-def*
using *cfunc-prod-unique cfunc-type-def comp-type left-cart-proj-type right-cart-proj-type*
by *auto*

The lemma below corresponds to Proposition 2.1.11 in Halvorson.

lemma *identity-distributes-across-composition*:
assumes *f-type*: $f : A \to B$ **and** *g-type*: $g : B \to C$
shows *id* $X \times_f (g \circ_c f) = ($*id* $X \times_f g) \circ_c ($*id* $X \times_f f)$
proof −
from *cfunc-cross-prod-unique*
have *uniqueness*: $\forall h.\ h : X \times_c A \to X \times_c C \wedge$
left-cart-proj $X\ C \circ_c h = id_c\ X \circ_c$ *left-cart-proj* $X\ A \wedge$
right-cart-proj $X\ C \circ_c h = (g \circ_c f) \circ_c$ *right-cart-proj* $X\ A \longrightarrow$
$h = id_c\ X \times_f (g \circ_c f)$
by (*meson comp-type f-type g-type id-type*)

have *left-eq*: *left-cart-proj* $X\ C \circ_c (id_c\ X \times_f g) \circ_c (id_c\ X \times_f f) = id_c\ X \circ_c$ *left-cart-proj* $X\ A$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 id-left-unit2 left-cart-proj-cfunc-cross-prod left-cart-proj-type*)
have *right-eq*: *right-cart-proj* $X\ C \circ_c (id_c\ X \times_f g) \circ_c (id_c\ X \times_f f) = (g \circ_c f) \circ_c$ *right-cart-proj* $X\ A$
using *assms* **by**(*typecheck-cfuncs*, *smt comp-associative2 right-cart-proj-cfunc-cross-prod right-cart-proj-type*)
show $id_c\ X \times_f g \circ_c f = (id_c\ X \times_f g) \circ_c id_c\ X \times_f f$
using *assms left-eq right-eq uniqueness* **by** (*typecheck-cfuncs*, *auto*)
qed

lemma *cfunc-cross-prod-comp-cfunc-prod*:
assumes *a-type*: $a : A \to W$ **and** *b-type*: $b : A \to X$
assumes *f-type*: $f : W \to Y$ **and** *g-type*: $g : X \to Z$
shows $(f \times_f g) \circ_c \langle a, b\rangle = \langle f \circ_c a, g \circ_c b\rangle$

proof −
from *cfunc-prod-unique* **have** *uniqueness*:
$\forall h.\ h : A \to Y \times_c Z \land$ *left-cart-proj* $Y\ Z \circ_c h = f \circ_c a \land$ *right-cart-proj* $Y\ Z$ $\circ_c h = g \circ_c b \longrightarrow$
$h = \langle f \circ_c a,\ g \circ_c b\rangle$
using *assms comp-type* **by** *blast*

have *left-cart-proj* $Y\ Z \circ_c (f \times_f g) \circ_c \langle a,\ b\rangle = f \circ_c$ *left-cart-proj* $W\ X \circ_c \langle a,\ b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-cart-proj-cfunc-cross-prod*)
then have *left-eq*: *left-cart-proj* $Y\ Z \circ_c (f \times_f g) \circ_c \langle a,\ b\rangle = f \circ_c a$
using *a-type b-type left-cart-proj-cfunc-prod* **by** *auto*

have *right-cart-proj* $Y\ Z \circ_c (f \times_f g) \circ_c \langle a,\ b\rangle = g \circ_c$ *right-cart-proj* $W\ X \circ_c \langle a,$ $b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-cart-proj-cfunc-cross-prod*)
then have *right-eq*: *right-cart-proj* $Y\ Z \circ_c (f \times_f g) \circ_c \langle a,\ b\rangle = g \circ_c b$
using *a-type b-type right-cart-proj-cfunc-prod* **by** *auto*

show $(f \times_f g) \circ_c \langle a,b\rangle = \langle f \circ_c a, g \circ_c b\rangle$
using *uniqueness left-eq right-eq assms* **by** (*meson cfunc-cross-prod-type cfunc-prod-type comp-type uniqueness*)
qed

lemma *cfunc-prod-comp*:
assumes *f-type*: $f : X \to Y$
assumes *a-type*: $a : Y \to A$ **and** *b-type*: $b : Y \to B$
shows $\langle a,\ b\rangle \circ_c f = \langle a \circ_c f,\ b \circ_c f\rangle$
proof −
have *same-left-proj*: *left-cart-proj* $A\ B \circ_c \langle a,\ b\rangle \circ_c f = a \circ_c f$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-cart-proj-cfunc-prod*)
have *same-right-proj*: *right-cart-proj* $A\ B \circ_c \langle a,\ b\rangle \circ_c f = b \circ_c f$
using *assms comp-associative2 right-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *auto*)
show $\langle a,b\rangle \circ_c f = \langle a \circ_c f,\ b \circ_c f\rangle$
by (*typecheck-cfuncs*, *metis a-type b-type cfunc-prod-unique f-type same-left-proj same-right-proj*)
qed

The lemma below corresponds to Exercise 2.1.12 in Halvorson.

lemma *id-cross-prod*: $id(X) \times_f id(Y) = id(X \times_c Y)$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-unique id-left-unit2 id-right-unit2 left-cart-proj-type right-cart-proj-type*)

The lemma below corresponds to Exercise 2.1.14 in Halvorson.

lemma *cfunc-cross-prod-comp-diagonal*:
assumes f: $X \to Y$
shows $(f \times_f f) \circ_c diagonal(X) = diagonal(Y) \circ_c f$
unfolding *diagonal-def*
proof −

have $(f \times_f f) \circ_c \langle id_c\ X,\ id_c\ X\rangle = \langle f \circ_c id_c\ X,\ f \circ_c id_c\ X\rangle$
  using *assms cfunc-cross-prod-comp-cfunc-prod id-type* **by** *blast*
also have ... $= \langle f,\ f\rangle$
  using *assms cfunc-type-def id-right-unit* **by** *auto*
also have ... $= \langle id_c\ Y \circ_c f,\ id_c\ Y \circ_c f\rangle$
  using *assms cfunc-type-def id-left-unit* **by** *auto*
also have ... $= \langle id_c\ Y,\ id_c\ Y\rangle \circ_c f$
  using *assms cfunc-prod-comp id-type* **by** *fastforce*
finally show $(f \times_f f) \circ_c \langle id_c\ X, id_c\ X\rangle = \langle id_c\ Y, id_c\ Y\rangle \circ_c f$**.**
qed

lemma *cfunc-cross-prod-comp-cfunc-cross-prod*:
assumes $a : A \to X\ b : B \to Y\ x : X \to Z\ y : Y \to W$
shows $(x \times_f y) \circ_c (a \times_f b) = (x \circ_c a) \times_f (y \circ_c b)$
proof −
have $(x \times_f y) \circ_c \langle a \circ_c$ *left-cart-proj* $A\ B$, $b \circ_c$ *right-cart-proj* $A\ B\rangle$
  $= \langle x \circ_c a \circ_c$ *left-cart-proj* $A\ B$, $y \circ_c b \circ_c$ *right-cart-proj* $A\ B\rangle$
by (*meson assms cfunc-cross-prod-comp-cfunc-prod comp-type left-cart-proj-type right-cart-proj-type*)
then show $(x \times_f y) \circ_c a \times_f b = (x \circ_c a) \times_f y \circ_c b$
  by (*typecheck-cfuncs,smt* (*z3*) *assms cfunc-cross-prod-def2 comp-associative2 left-cart-proj-type right-cart-proj-type*)
qed

lemma *cfunc-cross-prod-mono*:
assumes *type-assms*: $f : X \to Y\ g : Z \to W$
assumes *f-mono*: *monomorphism f* **and** *g-mono*: *monomorphism g*
shows *monomorphism* $(f \times_f g)$
using *type-assms*
proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
fix $x\ y\ A$
assume *x-type*: $x : A \to X \times_c Z$
assume *y-type*: $y : A \to X \times_c Z$

obtain *x1 x2* **where** *x-expand*: $x = \langle x1,\ x2\rangle$ **and** *x1-x2-type*: $x1 : A \to X\ x2 : A \to Z$
  using *cfunc-prod-unique comp-type left-cart-proj-type right-cart-proj-type x-type* **by** *blast*
obtain *y1 y2* **where** *y-expand*: $y = \langle y1,\ y2\rangle$ **and** *y1-y2-type*: $y1 : A \to X\ y2 : A \to Z$
  using *cfunc-prod-unique comp-type left-cart-proj-type right-cart-proj-type y-type* **by** *blast*

assume $(f \times_f g) \circ_c x = (f \times_f g) \circ_c y$
then have $(f \times_f g) \circ_c \langle x1,\ x2\rangle = (f \times_f g) \circ_c \langle y1,\ y2\rangle$
  using *x-expand y-expand* **by** *blast*
then have $\langle f \circ_c x1,\ g \circ_c x2\rangle = \langle f \circ_c y1,\ g \circ_c y2\rangle$
  using *cfunc-cross-prod-comp-cfunc-prod type-assms x1-x2-type y1-y2-type* **by** *auto*

then have $f \circ_c x1 = f \circ_c y1 \wedge g \circ_c x2 = g \circ_c y2$
by (*meson cart-prod-eq2 comp-type type-assms x1-x2-type y1-y2-type*)
then have $x1 = y1 \wedge x2 = y2$
using *cfunc-type-def f-mono g-mono monomorphism-def type-assms x1-x2-type y1-y2-type* **by** *auto*
then have $\langle x1, x2\rangle = \langle y1, y2\rangle$
by *blast*
then show $x = y$
by (*simp add*: *x-expand y-expand*)
qed

2.3 Useful Cartesian Product Permuting Functions

2.3.1 Swapping a Cartesian Product

definition *swap* :: *cset* ⇒ *cset* ⇒ *cfunc* **where**
swap X Y = ⟨*right-cart-proj* X Y, *left-cart-proj* X Y⟩

lemma *swap-type*[*type-rule*]: *swap* X Y : $X \times_c Y \to Y \times_c X$
unfolding *swap-def* **by** (*simp add*: *cfunc-prod-type left-cart-proj-type right-cart-proj-type*)

lemma *swap-ap*:
assumes $x : A \to X$ $y : A \to Y$
shows *swap* X Y $\circ_c \langle x, y\rangle = \langle y, x\rangle$
proof −
have *swap* X Y $\circ_c \langle x, y\rangle$ = ⟨*right-cart-proj* X Y,*left-cart-proj* X Y⟩ $\circ_c \langle x,y\rangle$
unfolding *swap-def* **by** *auto*
also have ... = ⟨*right-cart-proj* X Y $\circ_c \langle x,y\rangle$, *left-cart-proj* X Y $\circ_c \langle x,y\rangle$⟩
by (*meson assms cfunc-prod-comp cfunc-prod-type left-cart-proj-type right-cart-proj-type*)
also have ... = $\langle y, x\rangle$
using *assms left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod* **by** *auto*
finally show *?thesis***.**
qed

lemma *swap-cross-prod*:
assumes $x : A \to X$ $y : B \to Y$
shows *swap* X Y $\circ_c (x \times_f y) = (y \times_f x) \circ_c$ *swap* A B
proof −
have *swap* X Y $\circ_c (x \times_f y)$ = *swap* X Y $\circ_c$ ⟨x $\circ_c$ *left-cart-proj* A B, y $\circ_c$ *right-cart-proj* A B⟩
using *assms* **unfolding** *cfunc-cross-prod-def cfunc-type-def* **by** *auto*
also have ... = ⟨y $\circ_c$ *right-cart-proj* A B, x $\circ_c$ *left-cart-proj* A B⟩
by (*meson assms comp-type left-cart-proj-type right-cart-proj-type swap-ap*)
also have ... = $(y \times_f x) \circ_c$ ⟨*right-cart-proj* A B, *left-cart-proj* A B⟩
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = $(y \times_f x) \circ_c$ *swap* A B
unfolding *swap-def* **by** *auto*
finally show *?thesis***.**
qed

lemma *swap-idempotent*:
swap Y X $\circ_c$ *swap X Y* = *id* (*X* $\times_c$ *Y*)
by (*metis swap-def cfunc-prod-unique id-right-unit2 id-type left-cart-proj-type right-cart-proj-type swap-ap*)

lemma *swap-mono*:
monomorphism(*swap X Y*)
by (*metis cfunc-type-def iso-imp-epi-and-monic isomorphism-def swap-idempotent swap-type*)

2.3.2 Permuting a Cartesian Product to Associate to the Right

definition *associate-right* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
associate-right X Y Z =
⟨
left-cart-proj X Y $\circ_c$ *left-cart-proj* (*X* $\times_c$ *Y*) *Z*,
⟨
right-cart-proj X Y $\circ_c$ *left-cart-proj* (*X* $\times_c$ *Y*) *Z*,
right-cart-proj (*X* $\times_c$ *Y*) *Z*
⟩
⟩

lemma *associate-right-type*[*type-rule*]: *associate-right X Y Z* : (*X* $\times_c$ *Y*) $\times_c$ *Z* → *X* $\times_c$ (*Y* $\times_c$ *Z*)
unfolding *associate-right-def* **by** (*meson cfunc-prod-type comp-type left-cart-proj-type right-cart-proj-type*)

lemma *associate-right-ap*:
assumes *x* : *A* → *X* *y* : *A* → *Y* *z* : *A* → *Z*
shows *associate-right X Y Z* $\circ_c$ ⟨⟨*x*, *y*⟩, *z*⟩ = ⟨*x*, ⟨*y*, *z*⟩⟩
proof −
have *associate-right X Y Z* $\circ_c$ ⟨⟨*x*, *y*⟩, *z*⟩ = ⟨(*left-cart-proj X Y* $\circ_c$ *left-cart-proj* (*X* $\times_c$ *Y*) *Z*) $\circ_c$ ⟨⟨*x*,*y*⟩,*z*⟩,
⟨(*right-cart-proj X Y* $\circ_c$ *left-cart-proj* (*X* $\times_c$ *Y*) *Z*) $\circ_c$ ⟨⟨*x*,*y*⟩,*z*⟩, *right-cart-proj* (*X* $\times_c$ *Y*) *Z* $\circ_c$ ⟨⟨*x*,*y*⟩,*z*⟩⟩⟩
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *assms associate-right-def cfunc-prod-comp cfunc-prod-type*)
also have ... = ⟨*left-cart-proj X Y* $\circ_c$ ⟨*x*,*y*⟩, ⟨*right-cart-proj X Y* $\circ_c$ ⟨*x*,*y*⟩, *z*⟩⟩
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
also have ... =⟨*x*, ⟨*y*, *z*⟩⟩
using *assms left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod* **by** *auto*
finally show *?thesis***.**
qed

lemma *associate-right-crossprod-ap*:
assumes *x* : *A* → *X* *y* : *B* → *Y* *z* : *C* → *Z*
shows *associate-right X Y Z* $\circ_c$ ((*x* $\times_f$ *y*) $\times_f$ *z*) = (*x* $\times_f$ (*y* $\times_f$ *z*)) $\circ_c$ *associate-right A B C*

proof −
have *associate-right X Y Z* $\circ_c$ $((x \times_f y) \times_f z)$ =
associate-right X Y Z $\circ_c$ $\langle\langle x$ $\circ_c$ *left-cart-proj A B*, y $\circ_c$ *right-cart-proj A B*$\rangle$ $\circ_c$ *left-cart-proj* $(A\times_c B)$ C, z $\circ_c$ *right-cart-proj* $(A \times_c B)$ $C\rangle$
using *assms* **unfolding** *cfunc-cross-prod-def2* **by**(*typecheck-cfuncs*, *unfold cfunc-cross-prod-def2*, *auto*)
also have ... = *associate-right X Y Z* $\circ_c$ $\langle\langle x$ $\circ_c$ *left-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C, y $\circ_c$ *right-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ $C\rangle$, z $\circ_c$ *right-cart-proj* $(A \times_c B)$ $C\rangle$
using *assms* *cfunc-prod-comp comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle x$ $\circ_c$ *left-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C, $\langle y$ $\circ_c$ *right-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C, z $\circ_c$ *right-cart-proj* $(A \times_c B)$ $C\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *associate-right-ap*)
also have ... = $\langle x$ $\circ_c$ *left-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C, $(y \times_f z)\circ_c$ $\langle$*right-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C,*right-cart-proj* $(A \times_c B)$ $C\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = $(x \times_f (y\times_f z))$ $\circ_c$ $\langle\langle$*left-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C,$\langle$*right-cart-proj A B* $\circ_c$ *left-cart-proj* $(A\times_c B)$ C,*right-cart-proj* $(A \times_c B)$ $C\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = $(x \times_f (y\times_f z))$ $\circ_c$ *associate-right A B C*
unfolding *associate-right-def* **by** *auto*
finally show *?thesis*.
qed

2.3.3 Permuting a Cartesian Product to Associate to the Left

definition *associate-left* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
associate-left X Y Z =
$\langle$
$\langle$
left-cart-proj X $(Y \times_c Z)$,
left-cart-proj Y Z $\circ_c$ *right-cart-proj X* $(Y \times_c Z)$
$\rangle$,
right-cart-proj Y Z $\circ_c$ *right-cart-proj X* $(Y \times_c Z)$
$\rangle$

lemma *associate-left-type*[*type-rule*]: *associate-left X Y Z* : $X \times_c (Y \times_c Z) \to (X \times_c Y) \times_c Z$
unfolding *associate-left-def*
by (*meson cfunc-prod-type comp-type left-cart-proj-type right-cart-proj-type*)

lemma *associate-left-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *associate-left X Y Z* $\circ_c$ $\langle x, \langle y, z\rangle\rangle = \langle\langle x, y\rangle, z\rangle$
proof −
have *associate-left X Y Z* $\circ_c$ $\langle x, \langle y, z\rangle\rangle$ = $\langle\langle$*left-cart-proj X* $(Y \times_c Z)$,
left-cart-proj Y Z $\circ_c$ *right-cart-proj X* $(Y \times_c Z)\rangle$ $\circ_c$ $\langle x, \langle y, z\rangle\rangle$,
right-cart-proj Y Z $\circ_c$ *right-cart-proj X* $(Y \times_c Z)$ $\circ_c$ $\langle x, \langle y, z\rangle\rangle\rangle$
using *assms associate-left-def cfunc-prod-comp cfunc-type-def comp-associative*

comp-type **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle$ $\langle$*left-cart-proj X* (*Y* $\times_c$ *Z*) $\circ_c$ $\langle$*x*, $\langle$*y*, *z*$\rangle\rangle$,
left-cart-proj Y Z $\circ_c$ *right-cart-proj X* (*Y* $\times_c$ *Z*) $\circ_c$ $\langle$*x*, $\langle$*y*, *z*$\rangle\rangle\rangle$,
right-cart-proj Y Z $\circ_c$ *right-cart-proj X* (*Y* $\times_c$ *Z*) $\circ_c$ $\langle$*x*, $\langle$*y*, *z*$\rangle\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... = $\langle\langle$*x*, *left-cart-proj Y Z* $\circ_c$ $\langle$*y*, *z*$\rangle\rangle$, *right-cart-proj Y Z* $\circ_c$ $\langle$*y*, *z*$\rangle\rangle$
using *assms left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle\langle$*x*, *y*$\rangle$, *z*$\rangle$
using *assms left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod* **by** *auto*
finally show *?thesis*.
qed

lemma *right-left*:
associate-right A B C $\circ_c$ *associate-left A B C* = *id* (*A* $\times_c$ (*B* $\times_c$ *C*))
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *associate-left-def associate-right-ap cfunc-prod-unique comp-type id-right-unit2 left-cart-proj-type right-cart-proj-type*)

lemma *left-right*:
associate-left A B C $\circ_c$ *associate-right A B C* = *id* ((*A* $\times_c$ *B*) $\times_c$ *C*)
by (*smt associate-left-ap associate-right-def cfunc-cross-prod-def cfunc-prod-unique comp-type id-cross-prod id-domain id-left-unit2 left-cart-proj-type right-cart-proj-type*)

lemma *product-associates*:
A $\times_c$ (*B* $\times_c$ *C*) $\cong$ (*A* $\times_c$ *B*) $\times_c$ *C*
by (*metis associate-left-type associate-right-type cfunc-type-def is-isomorphic-def isomorphism-def left-right right-left*)

lemma *associate-left-crossprod-ap*:
assumes *x* : *A* $\rightarrow$ *X y* : *B* $\rightarrow$ *Y z* : *C* $\rightarrow$ *Z*
shows *associate-left X Y Z* $\circ_c$ (*x* $\times_f$ (*y*$\times_f$ *z*)) = ((*x* $\times_f$ *y*)$\times_f$ *z*) $\circ_c$ *associate-left A B C*
proof−
have *associate-left X Y Z* $\circ_c$ (*x* $\times_f$ (*y*$\times_f$ *z*)) =
associate-left X Y Z $\circ_c$ $\langle$*x* $\circ_c$ *left-cart-proj A* (*B*$\times_c$*C*), $\langle$*y* $\circ_c$ *left-cart-proj B C*, *z* $\circ_c$ *right-cart-proj B C*$\rangle$ $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*)$\rangle$
using *assms* **unfolding** *cfunc-cross-prod-def2* **by**(*typecheck-cfuncs*, *unfold cfunc-cross-prod-def2*, *auto*)
also have ... = *associate-left X Y Z* $\circ_c$ $\langle$*x* $\circ_c$ *left-cart-proj A* (*B*$\times_c$*C*), $\langle$*y* $\circ_c$ *left-cart-proj B C* $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*), *z* $\circ_c$ *right-cart-proj B C* $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*)$\rangle\rangle$
using *assms cfunc-prod-comp comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle\langle$*x* $\circ_c$ *left-cart-proj A* (*B*$\times_c$*C*), *y* $\circ_c$ *left-cart-proj B C* $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*)$\rangle$,*z* $\circ_c$ *right-cart-proj B C* $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*)$\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *associate-left-ap*)
also have ... = $\langle$(*x* $\times_f$ *y*)$\circ_c$ $\langle$ *left-cart-proj A* (*B*$\times_c$*C*), *left-cart-proj B C* $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*)$\rangle$,*z* $\circ_c$ *right-cart-proj B C* $\circ_c$ *right-cart-proj A* (*B*$\times_c$*C*)$\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = ((*x* $\times_f$ *y*)$\times_f$ *z*) $\circ_c$ $\langle\langle$*left-cart-proj A* (*B*$\times_c$*C*), *left-cart-proj B C*

$\circ_c$ *right-cart-proj* A $(B\times_c C)\rangle$,*right-cart-proj* B C $\circ_c$ *right-cart-proj* A $(B\times_c C)\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... $= ((x \times_f y)\times_f z) \circ_c$ *associate-left* A B C
unfolding *associate-left-def* **by** *auto*
finally show *?thesis*.
qed

2.3.4 Distributing over a Cartesian Product from the Right

definition *distribute-right-left* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
distribute-right-left X Y Z =
⟨*left-cart-proj* X Y $\circ_c$ *left-cart-proj* $(X \times_c Y)$ Z, *right-cart-proj* $(X \times_c Y)$ Z⟩

lemma *distribute-right-left-type*[*type-rule*]:
distribute-right-left X Y Z : $(X \times_c Y) \times_c Z \to X \times_c Z$
unfolding *distribute-right-left-def*
using *cfunc-prod-type comp-type left-cart-proj-type right-cart-proj-type* **by** *blast*

lemma *distribute-right-left-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-right-left* X Y Z $\circ_c$ $\langle\langle x, y\rangle, z\rangle = \langle x, z\rangle$
unfolding *distribute-right-left-def*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *assms cfunc-prod-comp comp-associative2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)

definition *distribute-right-right* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
distribute-right-right X Y Z =
⟨*right-cart-proj* X Y $\circ_c$ *left-cart-proj* $(X \times_c Y)$ Z, *right-cart-proj* $(X \times_c Y)$ Z⟩

lemma *distribute-right-right-type*[*type-rule*]:
distribute-right-right X Y Z : $(X \times_c Y) \times_c Z \to Y \times_c Z$
unfolding *distribute-right-right-def*
using *cfunc-prod-type comp-type left-cart-proj-type right-cart-proj-type* **by** *blast*

lemma *distribute-right-right-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-right-right* X Y Z $\circ_c$ $\langle\langle x, y\rangle, z\rangle = \langle y, z\rangle$
unfolding *distribute-right-right-def*
by (*typecheck-cfuncs*, *smt* (*z3*) *assms cfunc-prod-comp comp-associative2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)

definition *distribute-right* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
distribute-right X Y Z = ⟨*distribute-right-left* X Y Z, *distribute-right-right* X Y Z⟩

lemma *distribute-right-type*[*type-rule*]:
distribute-right X Y Z : $(X \times_c Y) \times_c Z \to (X \times_c Z) \times_c (Y \times_c Z)$
unfolding *distribute-right-def*
by (*simp add*: *cfunc-prod-type distribute-right-left-type distribute-right-right-type*)

lemma *distribute-right-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-right* $X\ Y\ Z \circ_c \langle\langle x, y\rangle, z\rangle = \langle\langle x, z\rangle, \langle y, z\rangle\rangle$
using *assms* **unfolding** *distribute-right-def*
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp distribute-right-left-ap distribute-right-right-ap*)

lemma *distribute-right-mono*:
monomorphism (*distribute-right* $X\ Y\ Z$)
proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
fix *g h A*
assume $g : A \to (X \times_c Y) \times_c Z$
then obtain *g1 g2 g3* **where** *g-expand*: $g = \langle\langle g1, g2\rangle, g3\rangle$
and *g1-g2-g3-types*: $g1 : A \to X$ $g2 : A \to Y$ $g3 : A \to Z$
using *cart-prod-decomp* **by** *blast*
assume $h : A \to (X \times_c Y) \times_c Z$
then obtain *h1 h2 h3* **where** *h-expand*: $h = \langle\langle h1, h2\rangle, h3\rangle$
and *h1-h2-h3-types*: $h1 : A \to X$ $h2 : A \to Y$ $h3 : A \to Z$
using *cart-prod-decomp* **by** *blast*

assume *distribute-right* $X\ Y\ Z \circ_c g =$ *distribute-right* $X\ Y\ Z \circ_c h$
then have *distribute-right* $X\ Y\ Z \circ_c \langle\langle g1, g2\rangle, g3\rangle =$ *distribute-right* $X\ Y\ Z \circ_c$
$\langle\langle h1, h2\rangle, h3\rangle$
using *g-expand h-expand* **by** *auto*
then have $\langle\langle g1, g3\rangle, \langle g2, g3\rangle\rangle = \langle\langle h1, h3\rangle, \langle h2, h3\rangle\rangle$
using *distribute-right-ap g1-g2-g3-types h1-h2-h3-types* **by** *auto*
then have $\langle g1, g3\rangle = \langle h1, h3\rangle \wedge \langle g2, g3\rangle = \langle h2, h3\rangle$
using *g1-g2-g3-types h1-h2-h3-types cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*)
then have $g1 = h1 \wedge g2 = h2 \wedge g3 = h3$
using *g1-g2-g3-types h1-h2-h3-types cart-prod-eq2* **by** *auto*
then have $\langle\langle g1, g2\rangle, g3\rangle = \langle\langle h1, h2\rangle, h3\rangle$
by *simp*
then show $g = h$
by (*simp add*: *g-expand h-expand*)
qed

2.3.5 Distributing over a Cartesian Product from the Left

definition *distribute-left-left* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
distribute-left-left $X\ Y\ Z =$
$\langle$*left-cart-proj* $X\ (Y \times_c Z)$, *left-cart-proj* $Y\ Z \circ_c$ *right-cart-proj* $X\ (Y \times_c Z)\rangle$

lemma *distribute-left-left-type*[*type-rule*]:
distribute-left-left $X\ Y\ Z : X \times_c (Y \times_c Z) \to X \times_c Y$
unfolding *distribute-left-left-def*
using *cfunc-prod-type comp-type left-cart-proj-type right-cart-proj-type* **by** *blast*

lemma *distribute-left-left-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$

shows *distribute-left-left X Y Z* $\circ_c$ ⟨*x*, ⟨*y*, *z*⟩⟩ = ⟨*x*, *y*⟩
using *assms distribute-left-left-def*
by (*typecheck-cfuncs*, *smt* (*z3*) *associate-left-ap associate-left-def cart-prod-decomp cart-prod-eq2 cfunc-prod-comp comp-associative2 distribute-left-left-def right-cart-proj-cfunc-prod right-cart-proj-type*)

definition *distribute-left-right* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
distribute-left-right X Y Z =
⟨*left-cart-proj X* (*Y* $\times_c$ *Z*), *right-cart-proj Y Z* $\circ_c$ *right-cart-proj X* (*Y* $\times_c$ *Z*)⟩

lemma *distribute-left-right-type*[*type-rule*]:
distribute-left-right X Y Z : *X* $\times_c$ (*Y* $\times_c$ *Z*) → *X* $\times_c$ *Z*
unfolding *distribute-left-right-def*
using *cfunc-prod-type comp-type left-cart-proj-type right-cart-proj-type* **by** *blast*

lemma *distribute-left-right-ap*:
assumes *x* : *A* → *X* *y* : *A* → *Y* *z* : *A* → *Z*
shows *distribute-left-right X Y Z* $\circ_c$ ⟨*x*, ⟨*y*, *z*⟩⟩ = ⟨*x*, *z*⟩
using *assms* **unfolding** *distribute-left-right-def*
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)

definition *distribute-left* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
distribute-left X Y Z = ⟨*distribute-left-left X Y Z*, *distribute-left-right X Y Z*⟩

lemma *distribute-left-type*[*type-rule*]:
distribute-left X Y Z : *X* $\times_c$ (*Y* $\times_c$ *Z*) → (*X* $\times_c$ *Y*) $\times_c$ (*X* $\times_c$ *Z*)
unfolding *distribute-left-def*
by (*simp add*: *cfunc-prod-type distribute-left-left-type distribute-left-right-type*)

lemma *distribute-left-ap*:
assumes *x* : *A* → *X* *y* : *A* → *Y* *z* : *A* → *Z*
shows *distribute-left X Y Z* $\circ_c$ ⟨*x*, ⟨*y*, *z*⟩⟩ = ⟨⟨*x*, *y*⟩, ⟨*x*, *z*⟩⟩
using *assms* **unfolding** *distribute-left-def*
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp distribute-left-left-ap distribute-left-right-ap*)

lemma *distribute-left-mono*:
monomorphism (*distribute-left X Y Z*)
proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
fix *g h A*
assume *g-type*: *g* : *A* → *X* $\times_c$ (*Y* $\times_c$ *Z*)
then obtain *g1 g2 g3* **where** *g-expand*: *g* = ⟨*g1*, ⟨*g2*, *g3*⟩⟩
and *g1-g2-g3-types*: *g1* : *A* → *X* *g2* : *A* → *Y* *g3* : *A* → *Z*
using *cart-prod-decomp* **by** *blast*
assume *h-type*: *h* : *A* → *X* $\times_c$ (*Y* $\times_c$ *Z*)
then obtain *h1 h2 h3* **where** *h-expand*: *h* = ⟨*h1*, ⟨*h2*, *h3*⟩⟩
and *h1-h2-h3-types*: *h1* : *A* → *X* *h2* : *A* → *Y* *h3* : *A* → *Z*
using *cart-prod-decomp* **by** *blast*

assume *distribute-left X Y Z* $\circ_c$ *g* = *distribute-left X Y Z* $\circ_c$ *h*
then have *distribute-left X Y Z* $\circ_c$ $\langle$*g1*, $\langle$*g2*, *g3*$\rangle\rangle$ = *distribute-left X Y Z* $\circ_c$ $\langle$*h1*, $\langle$*h2*, *h3*$\rangle\rangle$
using *g-expand h-expand* **by** *auto*
then have $\langle\langle$*g1*, *g2*$\rangle$, $\langle$*g1*, *g3*$\rangle\rangle$ = $\langle\langle$*h1*, *h2*$\rangle$, $\langle$*h1*, *h3*$\rangle\rangle$
using *distribute-left-ap g1-g2-g3-types h1-h2-h3-types* **by** *auto*
then have $\langle$*g1*, *g2*$\rangle$ = $\langle$*h1*, *h2*$\rangle$ $\wedge$ $\langle$*g1*, *g3*$\rangle$ = $\langle$*h1*, *h3*$\rangle$
using *g1-g2-g3-types h1-h2-h3-types cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*)
then have *g1* = *h1* $\wedge$ *g2* = *h2* $\wedge$ *g3* = *h3*
using *g1-g2-g3-types h1-h2-h3-types cart-prod-eq2* **by** *auto*
then have $\langle$*g1*, $\langle$*g2*, *g3*$\rangle\rangle$ = $\langle$*h1*, $\langle$*h2*, *h3*$\rangle\rangle$
by *simp*
then show *g* = *h*
by (*simp add*: *g-expand h-expand*)
qed

2.3.6 Selecting Pairs from a Pair of Pairs

definition *outers* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
outers A B C D = $\langle$
left-cart-proj A B $\circ_c$ *left-cart-proj* (*A* $\times_c$ *B*) (*C* $\times_c$ *D*),
right-cart-proj C D $\circ_c$ *right-cart-proj* (*A* $\times_c$ *B*) (*C* $\times_c$ *D*)
$\rangle$

lemma *outers-type*[*type-rule*]: *outers A B C D* : (*A* $\times_c$ *B*) $\times_c$ (*C* $\times_c$ *D*) $\rightarrow$ (*A* $\times_c$ *D*)
unfolding *outers-def* **by** *typecheck-cfuncs*

lemma *outers-apply*:
assumes *a* : *Z* $\rightarrow$ *A* *b* : *Z* $\rightarrow$ *B* *c* : *Z* $\rightarrow$ *C* *d* : *Z* $\rightarrow$ *D*
shows *outers A B C D* $\circ_c$ $\langle\langle$*a*,*b*$\rangle$, $\langle$*c*,*d*$\rangle\rangle$ = $\langle$*a*,*d*$\rangle$
proof –
have *outers A B C D* $\circ_c$ $\langle\langle$*a*,*b*$\rangle$, $\langle$*c*,*d*$\rangle\rangle$ = $\langle$
left-cart-proj A B $\circ_c$ *left-cart-proj* (*A* $\times_c$ *B*) (*C* $\times_c$ *D*) $\circ_c$ $\langle\langle$*a*,*b*$\rangle$, $\langle$*c*, *d*$\rangle\rangle$,
right-cart-proj C D $\circ_c$ *right-cart-proj* (*A* $\times_c$ *B*) (*C* $\times_c$ *D*) $\circ_c$ $\langle\langle$*a*,*b*$\rangle$, $\langle$*c*, *d*$\rangle\rangle$
$\rangle$
unfolding *outers-def* **using** *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... = $\langle$*left-cart-proj A B* $\circ_c$ $\langle$*a*,*b*$\rangle$, *right-cart-proj C D* $\circ_c$ $\langle$*c*,*d*$\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
also have ... = $\langle$*a*, *d*$\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
finally show *?thesis*.
qed

definition *inners* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
inners A B C D = $\langle$
right-cart-proj A B $\circ_c$ *left-cart-proj* (*A* $\times_c$ *B*) (*C* $\times_c$ *D*),
left-cart-proj C D $\circ_c$ *right-cart-proj* (*A* $\times_c$ *B*) (*C* $\times_c$ *D*)

```
  ⟩

lemma inners-type[type-rule]: inners A B C D : (A ×c B) ×c (C ×c D) → (B ×c
C)
  unfolding inners-def by typecheck-cfuncs

lemma inners-apply:
  assumes a : Z → A b : Z → B c : Z → C d : Z → D
  shows inners A B C D ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩ = ⟨b,c⟩
proof –
  have inners A B C D ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩ = ⟨
      right-cart-proj A B ∘c left-cart-proj (A ×c B) (C ×c D) ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩,
      left-cart-proj C D ∘c right-cart-proj (A ×c B) (C ×c D) ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩⟩
   unfolding inners-def using assms by (typecheck-cfuncs, simp add: cfunc-prod-comp
comp-associative2)
  also have ... = ⟨right-cart-proj A B ∘c ⟨a,b⟩, left-cart-proj C D ∘c ⟨c,d⟩⟩
   using assms by (typecheck-cfuncs, simp add: left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod)
  also have ... = ⟨b, c⟩
   using assms by (typecheck-cfuncs, simp add: left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod)
  finally show ?thesis.
qed

definition lefts :: cset ⇒ cset ⇒ cset ⇒ cset ⇒ cfunc where
  lefts A B C D = ⟨
      left-cart-proj A B ∘c left-cart-proj (A ×c B) (C ×c D),
      left-cart-proj C D ∘c right-cart-proj (A ×c B) (C ×c D)
  ⟩

lemma lefts-type[type-rule]: lefts A B C D : (A ×c B) ×c (C ×c D) → (A ×c C)
  unfolding lefts-def by typecheck-cfuncs

lemma lefts-apply:
  assumes a : Z → A b : Z → B c : Z → C d : Z → D
  shows lefts A B C D ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩ = ⟨a,c⟩
proof –
  have lefts A B C D ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩ = ⟨left-cart-proj A B ∘c left-cart-proj (A
×c B) (C ×c D) ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩, left-cart-proj C D ∘c right-cart-proj (A ×c B)
(C ×c D) ∘c ⟨⟨a,b⟩, ⟨c, d⟩⟩⟩
   unfolding lefts-def using assms by (typecheck-cfuncs, simp add: cfunc-prod-comp
comp-associative2)
  also have ... = ⟨left-cart-proj A B ∘c ⟨a,b⟩, left-cart-proj C D ∘c ⟨c,d⟩⟩
   using assms by (typecheck-cfuncs, simp add: left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod)
  also have ... = ⟨a, c⟩
    using assms by (typecheck-cfuncs, simp add: left-cart-proj-cfunc-prod)
  finally show ?thesis.
qed

definition rights :: cset ⇒ cset ⇒ cset ⇒ cset ⇒ cfunc where
  rights A B C D = ⟨
```

right-cart-proj A B $\circ_c$ *left-cart-proj* $(A \times_c B)$ $(C \times_c D)$,
right-cart-proj C D $\circ_c$ *right-cart-proj* $(A \times_c B)$ $(C \times_c D)$
$\rangle$

lemma *rights-type*[*type-rule*]: *rights* A B C D : $(A \times_c B) \times_c (C \times_c D) \to (B \times_c D)$
unfolding *rights-def* **by** *typecheck-cfuncs*

lemma *rights-apply*:
assumes $a : Z \to A$ $b : Z \to B$ $c : Z \to C$ $d : Z \to D$
shows *rights* A B C D $\circ_c$ $\langle\langle a,b\rangle, \langle c, d\rangle\rangle = \langle b,d\rangle$
proof −
have *rights* A B C D $\circ_c$ $\langle\langle a,b\rangle, \langle c, d\rangle\rangle = \langle$*right-cart-proj* A B $\circ_c$ *left-cart-proj* $(A \times_c B)$ $(C \times_c D)$ $\circ_c$ $\langle\langle a,b\rangle, \langle c, d\rangle\rangle$, *right-cart-proj* C D $\circ_c$ *right-cart-proj* $(A \times_c B)$ $(C \times_c D)$ $\circ_c$ $\langle\langle a,b\rangle, \langle c, d\rangle\rangle\rangle$
unfolding *rights-def* **using** *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... = $\langle$*right-cart-proj* A B $\circ_c$ $\langle a,b\rangle$, *right-cart-proj* C D $\circ_c$ $\langle c,d\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
also have ... = $\langle b, d\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *right-cart-proj-cfunc-prod*)
finally show *?thesis*.
qed

end

3 Terminal Objects and Elements

theory *Terminal*
imports *Cfunc Product*
begin

The axiomatization below corresponds to Axiom 3 (Terminal Object) in Halvorson.

axiomatization
terminal-func :: *cset* ⇒ *cfunc* (β- *100*) **and**
one-set :: *cset* (**1**)
where
terminal-func-type[*type-rule*]: $\beta_X : X \to \mathbf{1}$ **and**
terminal-func-unique: $h : X \to \mathbf{1} \Longrightarrow h = \beta_X$ **and**
one-separator: $f : X \to Y \Longrightarrow g : X \to Y \Longrightarrow (\bigwedge x.\ x : \mathbf{1} \to X \Longrightarrow f \circ_c x = g \circ_c x) \Longrightarrow f = g$

lemma *one-separator-contrapos*:
assumes $f : X \to Y$ $g : X \to Y$
shows $f \neq g \Longrightarrow \exists\ x.\ x : \mathbf{1} \to X \wedge f \circ_c x \neq g \circ_c x$
using *assms one-separator* **by** (*typecheck-cfuncs*, *blast*)

lemma *terminal-func-comp*:

$x : X \to Y \Longrightarrow \beta_Y \circ_c x = \beta_X$
by (*simp add*: *comp-type terminal-func-type terminal-func-unique*)

lemma *terminal-func-comp-elem*:
$x : \mathbf{1} \to X \Longrightarrow \beta_X \circ_c x = id\ \mathbf{1}$
by (*metis id-type terminal-func-comp terminal-func-unique*)

3.1 Set Membership and Emptiness

The abbreviation below captures Definition 2.1.16 in Halvorson.

abbreviation *member* :: *cfunc* ⇒ *cset* ⇒ *bool* (**infix** $\in_c$ *50*) **where**
$x \in_c X \equiv (x : \mathbf{1} \to X)$

definition *nonempty* :: *cset* ⇒ *bool* **where**
nonempty $X \equiv (\exists x.\ x \in_c X)$

definition *is-empty* :: *cset* ⇒ *bool* **where**
is-empty $X \equiv \neg(\exists x.\ x \in_c X)$

The lemma below corresponds to Exercise 2.1.18 in Halvorson.

lemma *element-monomorphism*:
$x \in_c X \Longrightarrow$ *monomorphism* x
unfolding *monomorphism-def*
by (*metis cfunc-type-def domain-comp terminal-func-unique*)

lemma *one-unique-element*:
$\exists!\ x.\ x \in_c \mathbf{1}$
using *terminal-func-type terminal-func-unique* **by** *blast*

lemma *prod-with-empty-is-empty1*:
assumes *is-empty* (A)
shows *is-empty*($A \times_c B$)
by (*meson assms comp-type left-cart-proj-type is-empty-def*)

lemma *prod-with-empty-is-empty2*:
assumes *is-empty* (B)
shows *is-empty* ($A \times_c B$)
using *assms cart-prod-decomp is-empty-def* **by** *blast*

3.2 Terminal Objects (sets with one element)

definition *terminal-object* :: *cset* ⇒ *bool* **where**
terminal-object $X \longleftrightarrow (\forall\ Y.\ \exists!\ f.\ f : Y \to X)$

lemma *one-terminal-object*: *terminal-object*($\mathbf{1}$)
unfolding *terminal-object-def* **using** *terminal-func-type terminal-func-unique* **by** *blast*

The lemma below is a generalisation of $?x \in_c\ ?X \Longrightarrow$ *monomorphism* $?x$

lemma *terminal-el-monomorphism*:
assumes $x : T \to X$
assumes *terminal-object* T
shows *monomorphism* x
unfolding *monomorphism-def*
by (*metis assms cfunc-type-def domain-comp terminal-object-def*)

The lemma below corresponds to Exercise 2.1.15 in Halvorson.

lemma *terminal-objects-isomorphic*:
assumes *terminal-object* X *terminal-object* Y
shows $X \cong Y$
unfolding *is-isomorphic-def*
proof −
obtain f **where** *f-type*: $f : X \to Y$ **and** *f-unique*: $\forall g.\ g : X \to Y \longrightarrow f = g$
using *assms*(*2*) *terminal-object-def* **by** *force*

obtain g **where** *g-type*: $g : Y \to X$ **and** *g-unique*: $\forall f.\ f : Y \to X \longrightarrow g = f$
using *assms*(*1*) *terminal-object-def* **by** *force*

have *g-f-is-id*: $g \circ_c f = id\ X$
using *assms*(*1*) *comp-type f-type g-type id-type terminal-object-def* **by** *blast*

have *f-g-is-id*: $f \circ_c g = id\ Y$
using *assms*(*2*) *comp-type f-type g-type id-type terminal-object-def* **by** *blast*

have *f-isomorphism*: *isomorphism f*
unfolding *isomorphism-def*
using *cfunc-type-def f-type g-type g-f-is-id f-g-is-id*
by (*intro exI*[**where** *x*=*g*], *auto*)

show $\exists f.\ f : X \to Y \wedge$ *isomorphism f*
using *f-isomorphism f-type* **by** *auto*
qed

The two lemmas below show the converse to Exercise 2.1.15 in Halvorson.

lemma *iso-to1-is-term*:
assumes $X \cong \mathbf{1}$
shows *terminal-object* X
unfolding *terminal-object-def*
proof
fix Y
obtain x **where** *x-type*[*type-rule*]: $x : \mathbf{1} \to X$ **and** *x-unique*: $\forall\ y.\ y : \mathbf{1} \to X \longrightarrow x = y$
by (*smt assms is-isomorphic-def iso-imp-epi-and-monic isomorphic-is-symmetric monomorphism-def2 terminal-func-comp terminal-func-unique*)
show $\exists !f.\ f : Y \to X$
proof (*rule ex1I*[**where** $a{=}x \circ_c \beta_Y$], *typecheck-cfuncs*)
fix *xa*
assume *xa-type*: $xa : Y \to X$

show $xa = x \circ_c \beta_Y$
proof (*rule ccontr*)
assume $xa \neq x \circ_c \beta_Y$
then obtain y **where** *elems-neq*: $xa \circ_c y \neq (x \circ_c \beta_Y) \circ_c y$ **and** *y-type*: $y : \mathbf{1} \to Y$
using *one-separator-contrapos comp-type terminal-func-type x-type xa-type* **by** *blast*
then show *False*
by (*smt* (*z3*) *comp-type elems-neq terminal-func-type x-unique xa-type y-type*)

qed
qed
qed

lemma *iso-to-term-is-term*:
assumes $X \cong Y$
assumes *terminal-object* Y
shows *terminal-object* X
by (*meson assms iso-to1-is-term isomorphic-is-transitive one-terminal-object terminal-objects-isomorphic*)

The lemma below corresponds to Proposition 2.1.19 in Halvorson.

lemma *single-elem-iso-one*:
$(\exists!\ x.\ x \in_c X) \longleftrightarrow X \cong \mathbf{1}$
proof
assume *X-iso-one*: $X \cong \mathbf{1}$
then have $\mathbf{1} \cong X$
by (*simp add*: *isomorphic-is-symmetric*)
then obtain f **where** *f-type*: $f : \mathbf{1} \to X$ **and** *f-iso*: *isomorphism* f
using *is-isomorphic-def* **by** *blast*
show $\exists!x.\ x \in_c X$
proof(*safe*)
show $\exists x.\ x \in_c X$
by (*meson f-type*)
next
fix x y
assume *x-type*[*type-rule*]: $x \in_c X$
assume *y-type*[*type-rule*]: $y \in_c X$
have *βx-eq-βy*: $\beta_X \circ_c x = \beta_X \circ_c y$
using *one-unique-element* **by** (*typecheck-cfuncs*, *blast*)
have *isomorphism* (β_X)
using *X-iso-one is-isomorphic-def terminal-func-unique* **by** *blast*
then have *monomorphism* (β_X)
by (*simp add*: *iso-imp-epi-and-monic*)
then show $x = y$
using *βx-eq-βy monomorphism-def2 terminal-func-type* **by** (*typecheck-cfuncs*, *blast*)
qed
next

assume $\exists!x.\ x \in_c X$
then obtain x **where** *x-type*: $x : \mathbf{1} \to X$ **and** *x-unique*: $\forall\ y.\ y : \mathbf{1} \to X \longrightarrow x = y$
by *blast*
have *terminal-object X*
unfolding *terminal-object-def*
proof
fix Y
show $\exists!f.\ f : Y \to X$
proof (*rule ex1I* [**where** $a{=}x \circ_c \beta_Y$])
show $x \circ_c \beta_Y : Y \to X$
using *comp-type terminal-func-type x-type* **by** *blast*
next
fix *xa*
assume *xa-type*: $xa : Y \to X$
show $xa = x \circ_c \beta_Y$
proof (*rule ccontr*)
assume $xa \neq x \circ_c \beta_Y$
then obtain y **where** *elems-neq*: $xa \circ_c y \neq (x \circ_c \beta_Y) \circ_c y$ **and** *y-type*: $y : \mathbf{1} \to Y$
using *one-separator-contrapos*[**where** $f{=}xa$, **where** $g{=}x \circ_c \beta_Y$, **where** $X{=}Y$, **where** $Y{=}X$]
using *comp-type terminal-func-type x-type xa-type* **by** *blast*
have *elem1*: $xa \circ_c y \in_c X$
using *comp-type xa-type y-type* **by** *auto*
have *elem2*: $(x \circ_c \beta_Y) \circ_c y \in_c X$
using *comp-type terminal-func-type x-type y-type* **by** *blast*
show *False*
using *elem1 elem2 elems-neq x-unique* **by** *blast*
qed
qed
qed
then show $X \cong \mathbf{1}$
by (*simp add*: *one-terminal-object terminal-objects-isomorphic*)
qed

3.3 Injectivity

The definition below corresponds to Definition 2.1.24 in Halvorson.

definition *injective* :: *cfunc* $\Rightarrow$ *bool* **where**
injective $f \longleftrightarrow (\forall\ x\ y.\ (x \in_c \mathit{domain}\ f \wedge y \in_c \mathit{domain}\ f \wedge f \circ_c x = f \circ_c y) \longrightarrow x = y)$

lemma *injective-def2*:
assumes $f : X \to Y$
shows *injective* $f \longleftrightarrow (\forall\ x\ y.\ (x \in_c X \wedge y \in_c X \wedge f \circ_c x = f \circ_c y) \longrightarrow x = y)$
using *assms cfunc-type-def injective-def* **by** *force*

The lemma below corresponds to Exercise 2.1.26 in Halvorson.

lemma *monomorphism-imp-injective*:
monomorphism f $\Longrightarrow$ *injective f*
by (*simp add*: *cfunc-type-def injective-def monomorphism-def*)

The lemma below corresponds to Proposition 2.1.27 in Halvorson.

lemma *injective-imp-monomorphism*:
injective f $\Longrightarrow$ *monomorphism f*
unfolding *monomorphism-def injective-def*
proof *clarify*
fix *g h*
assume *f-inj*: $\forall x\ y.\ x \in_c$ *domain f* $\wedge$ $y \in_c$ *domain f* $\wedge$ $f \circ_c x = f \circ_c y \longrightarrow x = y$
assume *cd-g-eq-d-f*: *codomain g* = *domain f*
assume *cd-h-eq-d-f*: *codomain h* = *domain f*
assume *fg-eq-fh*: $f \circ_c g = f \circ_c h$

obtain *X Y* **where** *f-type*: $f : X \to Y$
using *cfunc-type-def* **by** *auto*
obtain *A* **where** *g-type*: $g : A \to X$ **and** *h-type*: $h : A \to X$
by (*metis cd-g-eq-d-f cd-h-eq-d-f cfunc-type-def domain-comp f-type fg-eq-fh*)

have $\forall x.\ x \in_c A \longrightarrow g \circ_c x = h \circ_c x$
proof *clarify*
fix *x*
assume *x-in-A*: $x \in_c A$

have $f \circ_c g \circ_c x = f \circ_c h \circ_c x$
using *g-type h-type x-in-A f-type comp-associative2 fg-eq-fh* **by** (*typecheck-cfuncs*, *auto*)
then show $g \circ_c x = h \circ_c x$
using *cd-h-eq-d-f cfunc-type-def comp-type f-inj g-type h-type x-in-A* **by** *presburger*
qed
then show $g = h$
using *g-type h-type one-separator* **by** *auto*
qed

lemma *cfunc-cross-prod-inj*:
assumes *type-assms*: $f : X \to Y$ $g : Z \to W$
assumes *injective f* $\wedge$ *injective g*
shows *injective* $(f \times_f g)$
by (*typecheck-cfuncs*, *metis assms cfunc-cross-prod-mono injective-imp-monomorphism monomorphism-imp-injective*)

lemma *cfunc-cross-prod-mono-converse*:
assumes *type-assms*: $f : X \to Y$ $g : Z \to W$
assumes *fg-inject*: *injective* $(f \times_f g)$
assumes *nonempty*: *nonempty X nonempty Z*
shows *injective f* $\wedge$ *injective g*

```
  unfolding injective-def
proof safe
  fix x y
  assume x-type: x ∈c domain f
  assume y-type: y ∈c domain f
  assume equals: f ∘c x = f ∘c y
  have fg-type: f ×f g : X ×c Z → Y ×c W
    using assms by typecheck-cfuncs
  have x-type2: x ∈c X
    using cfunc-type-def type-assms(1) x-type by auto
  have y-type2: y ∈c X
    using cfunc-type-def type-assms(1) y-type by auto
  show x = y
  proof -
    obtain b where b-def: b ∈c Z
      using nonempty(2) nonempty-def by blast

    have xb-type: ⟨x,b⟩ ∈c X ×c Z
      by (simp add: b-def cfunc-prod-type x-type2)
    have yb-type: ⟨y,b⟩ ∈c X ×c Z
      by (simp add: b-def cfunc-prod-type y-type2)
    have (f ×f g) ∘c ⟨x,b⟩ = ⟨f ∘c x,g ∘c b⟩
      using b-def cfunc-cross-prod-comp-cfunc-prod type-assms x-type2 by blast
    also have ... = ⟨f ∘c y,g ∘c b⟩
      by (simp add: equals)
    also have ... = (f ×f g) ∘c ⟨y,b⟩
      using b-def cfunc-cross-prod-comp-cfunc-prod type-assms y-type2 by auto
    finally have ⟨x,b⟩ = ⟨y,b⟩
      by (metis cfunc-type-def fg-inject fg-type injective-def xb-type yb-type)
    then show x = y
      using b-def cart-prod-eq2 x-type2 y-type2 by auto
  qed
next
  fix x y
  assume x-type: x ∈c domain g
  assume y-type: y ∈c domain g
  assume equals: g ∘c x = g ∘c y
  have fg-type: f ×f g : X ×c Z → Y ×c W
    using assms by typecheck-cfuncs
  have x-type2: x ∈c Z
    using cfunc-type-def type-assms(2) x-type by auto
  have y-type2: y ∈c Z
    using cfunc-type-def type-assms(2) y-type by auto
  show x = y
  proof -
    obtain b where b-def: b ∈c X
      using nonempty(1) nonempty-def by blast
    have xb-type: ⟨b,x⟩ ∈c X ×c Z
      by (simp add: b-def cfunc-prod-type x-type2)
```

have *yb-type*: $\langle b,y\rangle \in_c X \times_c Z$
by (*simp add*: *b-def cfunc-prod-type y-type2*)
have $(f \times_f g) \circ_c \langle b,x\rangle = \langle f \circ_c b, g \circ_c x\rangle$
using *b-def cfunc-cross-prod-comp-cfunc-prod type-assms*(*1*) *type-assms*(*2*) *x-type2* **by** *blast*
also have ... $= (f \times_f g) \circ_c \langle b,y\rangle$
using *b-def cfunc-cross-prod-comp-cfunc-prod equals type-assms*(*1*) *type-assms*(*2*) *y-type2* **by** *auto*
finally have $\langle b,x\rangle = \langle b,y\rangle$
using *fg-inject fg-type injective-def2 xb-type yb-type* **by** *blast*
then show $x = y$
using *b-def cart-prod-eq2 x-type2 y-type2* **by** *blast*
qed
qed

The next lemma shows that unless both domains are nonempty we gain no new information. That is, it will be the case that $f \times g$ is injective, and we cannot infer from this that f or g are injective since $f \times g$ will be injective no matter what.

lemma *the-nonempty-assumption-above-is-always-required*:
assumes $f : X \to Y$ $g : Z \to W$
assumes $\neg$(*nonempty* X) $\vee$ $\neg$(*nonempty* Z)
shows *injective* $(f \times_f g)$
unfolding *injective-def*
proof(*cases nonempty*(X), *safe*)
fix x y
assume *nonempty*: *nonempty* X
assume *x-type*: $x \in_c$ *domain* $(f \times_f g)$
assume $y \in_c$ *domain* $(f \times_f g)$
then have $\neg$(*nonempty* Z)
using *nonempty assms*(*3*) **by** *blast*
have *fg-type*: $f \times_f g : X \times_c Z \to Y \times_c W$
by (*typecheck-cfuncs*, *simp add*: *assms*(*1*,*2*))
then have $x \in_c X \times_c Z$
using *x-type cfunc-type-def* **by** *auto*
then have $\exists z.\ z \in_c Z$
using *cart-prod-decomp* **by** *blast*
then have *False*
using *assms*(*3*) *nonempty nonempty-def* **by** *blast*
then show $x=y$
by *auto*
next
fix x y
assume *X-is-empty*: $\neg$ *nonempty* X
assume *x-type*: $x \in_c$ *domain* $(f \times_f g)$
assume $y \in_c$ *domain*$(f \times_f g)$
have *fg-type*: $f \times_f g : X \times_c Z \to Y \times_c W$
by (*typecheck-cfuncs*, *simp add*: *assms*(*1*,*2*))
then have $x \in_c X \times_c Z$

using *x-type cfunc-type-def* **by** *auto*
then have $\exists z.\ z \in_c X$
using *cart-prod-decomp* **by** *blast*
then have *False*
using *assms(3) X-is-empty nonempty-def* **by** *blast*
then show *x=y*
by *auto*
qed

3.4 Surjectivity

The definition below corresponds to Definition 2.1.28 in Halvorson.

definition *surjective* :: *cfunc* ⇒ *bool* **where**
surjective f $\longleftrightarrow (\forall y.\ y \in_c \textit{codomain}\ f \longrightarrow (\exists x.\ x \in_c \textit{domain}\ f \wedge f \circ_c x = y))$

lemma *surjective-def2*:
assumes $f : X \to Y$
shows *surjective f* $\longleftrightarrow (\forall y.\ y \in_c Y \longrightarrow (\exists x.\ x \in_c X \wedge f \circ_c x = y))$
using *assms* **unfolding** *surjective-def cfunc-type-def* **by** *auto*

The lemma below corresponds to Exercise 2.1.30 in Halvorson.

lemma *surjective-is-epimorphism*:
surjective f ⟹ *epimorphism f*
unfolding *surjective-def epimorphism-def*
proof (*cases nonempty* (*codomain f*), *safe*)
fix *g h*
assume *f-surj*: $\forall y.\ y \in_c \textit{codomain}\ f \longrightarrow (\exists x.\ x \in_c \textit{domain}\ f \wedge f \circ_c x = y)$
assume *d-g-eq-cd-f*: *domain g* = *codomain f*
assume *d-h-eq-cd-f*: *domain h* = *codomain f*
assume *gf-eq-hf*: $g \circ_c f = h \circ_c f$
assume *nonempty*: *nonempty* (*codomain f*)

obtain *X Y* **where** *f-type*: $f : X \to Y$
using *nonempty cfunc-type-def f-surj nonempty-def* **by** *auto*
obtain *A* **where** *g-type*: $g : Y \to A$ **and** *h-type*: $h : Y \to A$
by (*metis cfunc-type-def codomain-comp d-g-eq-cd-f d-h-eq-cd-f f-type gf-eq-hf*)
show $g = h$
proof (*rule ccontr*)
assume $g \neq h$
then obtain *y* **where** *y-in-X*: $y \in_c Y$ **and** *gy-neq-hy*: $g \circ_c y \neq h \circ_c y$
using *g-type h-type one-separator* **by** *blast*
then obtain *x* **where** $x \in_c X$ **and** $f \circ_c x = y$
using *cfunc-type-def f-surj f-type* **by** *auto*
then have $g \circ_c f \neq h \circ_c f$
using *comp-associative2 f-type g-type gy-neq-hy h-type* **by** *auto*
then show *False*
using *gf-eq-hf* **by** *auto*
qed
next

fix *g h*
assume *empty*: $\neg$ *nonempty* (*codomain f*)
assume *domain g* = *codomain f domain h* = *codomain f*
then show $g \circ_c f = h \circ_c f \Longrightarrow g = h$
by (*metis empty cfunc-type-def codomain-comp nonempty-def one-separator*)
qed

The lemma below corresponds to Proposition 2.2.10 in Halvorson.

lemma *cfunc-cross-prod-surj*:
assumes *type-assms*: $f : A \rightarrow C$ $g : B \rightarrow D$
assumes *f-surj*: *surjective f* **and** *g-surj*: *surjective g*
shows *surjective* $(f \times_f g)$
unfolding *surjective-def*
proof(*clarify*)
fix *y*
assume *y-type*: $y \in_c$ *codomain* $(f \times_f g)$
have *fg-type*: $f \times_f g$: $A \times_c B \rightarrow C \times_c D$
using *assms* **by** *typecheck-cfuncs*
then have $y \in_c C \times_c D$
using *cfunc-type-def y-type* **by** *auto*
then have $\exists\ c\ d.\ c \in_c C \wedge d \in_c D \wedge y = \langle c,d \rangle$
using *cart-prod-decomp* **by** *blast*
then obtain *c d* **where** *y-def*: $c \in_c C \wedge d \in_c D \wedge y = \langle c,d \rangle$
by *blast*
then have $\exists\ a\ b.\ a \in_c A \wedge b \in_c B \wedge f \circ_c a = c \wedge g \circ_c b = d$
by (*metis cfunc-type-def f-surj g-surj surjective-def type-assms*)
then obtain *a b* **where** *ab-def*: $a \in_c A \wedge b \in_c B \wedge f \circ_c a = c \wedge g \circ_c b = d$
by *blast*
then obtain *x* **where** *x-def*: $x = \langle a,b \rangle$
by *auto*
have *x-type*: $x \in_c$ *domain* $(f \times_f g)$
using *ab-def cfunc-prod-type cfunc-type-def fg-type x-def* **by** *auto*
have $(f \times_f g) \circ_c x = y$
using *ab-def cfunc-cross-prod-comp-cfunc-prod type-assms*(*1*) *type-assms*(*2*) *x-def y-def* **by** *blast*
then show $\exists x.\ x \in_c$ *domain* $(f \times_f g) \wedge (f \times_f g) \circ_c x = y$
using *x-type* **by** *blast*
qed

lemma *cfunc-cross-prod-surj-converse*:
assumes *type-assms*: $f : A \rightarrow C$ $g : B \rightarrow D$
assumes *nonempty*: *nonempty* $C \wedge$ *nonempty* D
assumes *surjective* $(f \times_f g)$
shows *surjective* $f \wedge$ *surjective* g
unfolding *surjective-def*
proof(*safe*)
fix *c*
assume *c-type*[*type-rule*]: $c \in_c$ *codomain f*
then have *c-type2*: $c \in_c C$

using *cfunc-type-def type-assms*(*1*) **by** *auto*
obtain *d* **where** *d-type*[*type-rule*]: $d \in_c D$
using *nonempty nonempty-def* **by** *blast*
then obtain *ab* **where** *ab-type*[*type-rule*]: $ab \in_c A \times_c B$ **and** *ab-def*: $(f \times_f g) \circ_c ab = \langle c, d\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis assms*(*4*) *cfunc-type-def surjective-def2*)
then obtain *a b* **where** *a-type*[*type-rule*]: $a \in_c A$ **and** *b-type*[*type-rule*]: $b \in_c B$ **and** *ab-def2*: $ab = \langle a,b\rangle$
using *cart-prod-decomp* **by** *blast*
have $a \in_c domain\ f \wedge f \circ_c a = c$
using *ab-def ab-def2 b-type cfunc-cross-prod-comp-cfunc-prod cfunc-type-def comp-type d-type cart-prod-eq2 type-assms* **by** (*typecheck-cfuncs*, *auto*)
then show $\exists x.\ x \in_c domain\ f \wedge f \circ_c x = c$
by *blast*
next
fix *d*
assume *d-type*[*type-rule*]: $d \in_c codomain\ g$
then have *y-type2*: $d \in_c D$
using *cfunc-type-def type-assms*(*2*) **by** *auto*
obtain *c* **where** *d-type*[*type-rule*]: $c \in_c C$
using *nonempty nonempty-def* **by** *blast*
then obtain *ab* **where** *ab-type*[*type-rule*]: $ab \in_c A \times_c B$ **and** *ab-def*: $(f \times_f g) \circ_c ab = \langle c, d\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis assms*(*4*) *cfunc-type-def surjective-def2*)
then obtain *a b* **where** *a-type*[*type-rule*]: $a \in_c A$ **and** *b-type*[*type-rule*]: $b \in_c B$ **and** *ab-def2*: $ab = \langle a,b\rangle$
using *cart-prod-decomp* **by** *blast*
then obtain *a b* **where** *a-type*[*type-rule*]: $a \in_c A$ **and** *b-type*[*type-rule*]: $b \in_c B$ **and** *ab-def2*: $ab = \langle a,b\rangle$
using *cart-prod-decomp* **by** *blast*
have $b \in_c domain\ g \wedge g \circ_c b = d$
using *a-type ab-def ab-def2 cfunc-cross-prod-comp-cfunc-prod cfunc-type-def comp-type d-type cart-prod-eq2 type-assms* **by**(*typecheck-cfuncs*, *force*)
then show $\exists x.\ x \in_c domain\ g \wedge g \circ_c x = d$
by *blast*
qed

3.5 Interactions of Cartesian Products with Terminal Objects

lemma *diag-on-elements*:
assumes $x \in_c X$
shows $diagonal\ X \circ_c x = \langle x,x\rangle$
using *assms cfunc-prod-comp cfunc-type-def diagonal-def id-left-unit id-type* **by** *auto*

lemma *one-cross-one-unique-element*:
$\exists!\ x.\ x \in_c \mathbf{1} \times_c \mathbf{1}$
proof (*rule ex1I*[**where** *a*=*diagonal* **1**])

show *diagonal* **1** $\in_c$ **1** $\times_c$ **1**
  by (*simp add*: *cfunc-prod-type diagonal-def id-type*)
next
  fix *x*
  assume *x-type*: $x \in_c$ **1** $\times_c$ **1**

  have *left-eq*: *left-cart-proj* **1 1** $\circ_c$ x = *id* **1**
    using *x-type one-unique-element* **by** (*typecheck-cfuncs*, *blast*)
  have *right-eq*: *right-cart-proj* **1 1** $\circ_c$ x = *id* **1**
    using *x-type one-unique-element* **by** (*typecheck-cfuncs*, *blast*)

  then show x = *diagonal* **1**
    unfolding *diagonal-def* **using** *cfunc-prod-unique id-type left-eq x-type* **by** *blast*
qed

The lemma below corresponds to Proposition 2.1.20 in Halvorson.

lemma *X-is-cart-prod1*:
  is-cart-prod X (*id X*) (β_X) *X* **1**
  unfolding *is-cart-prod-def*
proof *safe*
  show id_c $X : X \to X$
    by *typecheck-cfuncs*
next
  show $\beta_X : X \to$ **1**
    by *typecheck-cfuncs*
next
  fix *f g Y*
  assume *f-type*: $f : Y \to X$ **and** *g-type*: $g : Y \to$ **1**
  then show $\exists h.\ h : Y \to X \wedge$
      $id_c\ X \circ_c h = f \wedge \beta_X \circ_c h = g \wedge (\forall h2.\ h2 : Y \to X \wedge id_c\ X \circ_c h2 = f$
$\wedge\ \beta_X \circ_c h2 = g \longrightarrow h2 = h)$
  proof (*intro exI*[**where** *x*=*f*], *safe*)
    show *id* $X \circ_c f = f$
      using *cfunc-type-def f-type id-left-unit* **by** *auto*
    show $\beta_X \circ_c f = g$
      by (*metis comp-type f-type g-type terminal-func-type terminal-func-unique*)
    show $\bigwedge h2.\ h2 : Y \to X \Longrightarrow h2 = id_c\ X \circ_c h2$
      using *cfunc-type-def id-left-unit* **by** *auto*
  qed
qed

lemma *X-is-cart-prod2*:
  is-cart-prod X (β_X) (*id X*) **1** *X*
  unfolding *is-cart-prod-def*
proof *safe*
  show id_c $X : X \to X$
    by *typecheck-cfuncs*
next
  show $\beta_X : X \to$ **1**

by *typecheck-cfuncs*
next
fix *f g Z*
assume *f-type*: $f : Z \to \mathbf{1}$ **and** *g-type*: $g : Z \to X$
then show $\exists h.\ h : Z \to X \wedge$
$\beta_X \circ_c h = f \wedge id_c\ X \circ_c h = g \wedge (\forall h2.\ h2 : Z \to X \wedge \beta_X \circ_c h2 = f \wedge id_c\ X \circ_c h2 = g \longrightarrow h2 = h)$
proof (*intro exI*[**where** *x*=*g*], *safe*)
show $id_c\ X \circ_c g = g$
using *cfunc-type-def g-type id-left-unit* **by** *auto*
show $\beta_X \circ_c g = f$
by (*metis comp-type f-type g-type terminal-func-type terminal-func-unique*)
show $\bigwedge h2.\ h2 : Z \to X \Longrightarrow h2 = id_c\ X \circ_c h2$
using *cfunc-type-def id-left-unit* **by** *auto*
qed
qed

lemma *A-x-one-iso-A*:
$X \times_c \mathbf{1} \cong X$
by (*metis X-is-cart-prod1 canonical-cart-prod-is-cart-prod cart-prods-isomorphic fst-conv is-isomorphic-def snd-conv*)

lemma *one-x-A-iso-A*:
$\mathbf{1} \times_c X \cong X$
by (*meson A-x-one-iso-A isomorphic-is-transitive product-commutes*)

The following four lemmas provide some concrete examples of the above isomorphisms

lemma *left-cart-proj-one-left-inverse*:
$\langle id\ X, \beta_X \rangle \circ_c$ *left-cart-proj* $X\ \mathbf{1} = id\ (X \times_c \mathbf{1})$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp cfunc-prod-unique id-left-unit2 id-right-unit2 right-cart-proj-type terminal-func-comp terminal-func-unique*)

lemma *left-cart-proj-one-right-inverse*:
left-cart-proj $X\ \mathbf{1} \circ_c \langle id\ X, \beta_X \rangle = id\ X$
using *left-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *blast*)

lemma *right-cart-proj-one-left-inverse*:
$\langle \beta_X, id\ X \rangle \circ_c$ *right-cart-proj* $\mathbf{1}\ X = id\ (\mathbf{1} \times_c X)$
by (*typecheck-cfuncs*, *smt* (*z3*) *cart-prod-decomp cfunc-prod-comp id-left-unit2 id-right-unit2 right-cart-proj-cfunc-prod terminal-func-comp terminal-func-unique*)

lemma *right-cart-proj-one-right-inverse*:
right-cart-proj $\mathbf{1}\ X \circ_c \langle \beta_X, id\ X \rangle = id\ X$
using *right-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *blast*)

lemma *cfunc-cross-prod-right-terminal-decomp*:
assumes $f : X \to Y\ x : \mathbf{1} \to Z$
shows $f \times_f x = \langle f,\ x \circ_c \beta_X \rangle \circ_c$ *left-cart-proj* $X\ \mathbf{1}$

using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-def cfunc-prod-comp cfunc-type-def*
comp-associative2 right-cart-proj-type terminal-func-comp terminal-func-unique)

The lemma below corresponds to Proposition 2.1.21 in Halvorson.

lemma *cart-prod-elem-eq*:
assumes $a \in_c X \times_c Y$ $b \in_c X \times_c Y$
shows $a = b \longleftrightarrow$
($\textit{left-cart-proj}\ X\ Y \circ_c a = \textit{left-cart-proj}\ X\ Y \circ_c b$
$\wedge$ $\textit{right-cart-proj}\ X\ Y \circ_c a = \textit{right-cart-proj}\ X\ Y \circ_c b$)
by (*metis* (*full-types*) *assms cfunc-prod-unique comp-type left-cart-proj-type right-cart-proj-type*)

The lemma below corresponds to Note 2.1.22 in Halvorson.

lemma *element-pair-eq*:
assumes $x \in_c X$ $x' \in_c X$ $y \in_c Y$ $y' \in_c Y$
shows $\langle x, y\rangle = \langle x', y'\rangle \longleftrightarrow x = x' \wedge y = y'$
by (*metis assms left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)

The lemma below corresponds to Proposition 2.1.23 in Halvorson.

lemma *nonempty-right-imp-left-proj-epimorphism*:
nonempty $Y \Longrightarrow$ *epimorphism* (*left-cart-proj* X Y)
proof −
assume *nonempty* Y
then obtain y **where** *y-in-Y*: $y : \mathbf{1} \to Y$
using *nonempty-def* **by** *blast*
then have *id-eq*: $(\textit{left-cart-proj}\ X\ Y) \circ_c \langle \textit{id}\ X, y \circ_c \beta_X\rangle = \textit{id}\ X$
using *comp-type id-type left-cart-proj-cfunc-prod terminal-func-type* **by** *blast*
then show *epimorphism* (*left-cart-proj* X Y)
unfolding *epimorphism-def*
proof *clarify*
fix g h
assume *domain-g*: *domain* g = *codomain* (*left-cart-proj* X Y)
assume *domain-h*: *domain* h = *codomain* (*left-cart-proj* X Y)
assume $g \circ_c \textit{left-cart-proj}\ X\ Y = h \circ_c \textit{left-cart-proj}\ X\ Y$
then have $g \circ_c \textit{left-cart-proj}\ X\ Y \circ_c \langle \textit{id}\ X, y \circ_c \beta_X\rangle = h \circ_c \textit{left-cart-proj}\ X\ Y \circ_c \langle \textit{id}\ X, y \circ_c \beta_X\rangle$
using *y-in-Y* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-type-def comp-associative domain-g domain-h*)
then show $g = h$
by (*metis cfunc-type-def domain-g domain-h id-eq id-right-unit left-cart-proj-type*)
qed
qed

The lemma below is the dual of Proposition 2.1.23 in Halvorson.

lemma *nonempty-left-imp-right-proj-epimorphism*:
nonempty $X \Longrightarrow$ *epimorphism* (*right-cart-proj* X Y)
proof −
assume *nonempty* X
then obtain y **where** *y-in-Y*: y: $\mathbf{1} \to X$

using *nonempty-def* **by** *blast*
then have *id-eq*: (*right-cart-proj X Y*) $\circ_c$ $\langle y \circ_c \beta_Y, id\ Y\rangle$ = *id Y*
using *comp-type id-type right-cart-proj-cfunc-prod terminal-func-type* **by** *blast*
then show *epimorphism* (*right-cart-proj X Y*)
unfolding *epimorphism-def*
proof *clarify*
fix *g h*
assume *domain-g*: *domain g* = *codomain* (*right-cart-proj X Y*)
assume *domain-h*: *domain h* = *codomain* (*right-cart-proj X Y*)
assume *g* $\circ_c$ *right-cart-proj X Y* = *h* $\circ_c$ *right-cart-proj X Y*
then have *g* $\circ_c$ *right-cart-proj X Y* $\circ_c$ $\langle y \circ_c \beta_Y, id\ Y\rangle$ = *h* $\circ_c$ *right-cart-proj X Y* $\circ_c$ $\langle y \circ_c \beta_Y, id\ Y\rangle$
using *y-in-Y* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-type-def comp-associative domain-g domain-h*)
then show *g* = *h*
by (*metis cfunc-type-def domain-g domain-h id-eq id-right-unit right-cart-proj-type*)
qed
qed

lemma *cart-prod-extract-left*:
assumes $f : \mathbf{1} \to X$ $g : \mathbf{1} \to Y$
shows $\langle f, g\rangle = \langle id\ X, g \circ_c \beta_X\rangle \circ_c f$
proof −
have $\langle f, g\rangle = \langle id\ X \circ_c f, g \circ_c \beta_X \circ_c f\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis id-left-unit2 id-right-unit2 id-type one-unique-element*)
also have ... = $\langle id\ X, g \circ_c \beta_X\rangle \circ_c f$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
finally show *?thesis*.
qed

lemma *cart-prod-extract-right*:
assumes $f : \mathbf{1} \to X$ $g : \mathbf{1} \to Y$
shows $\langle f, g\rangle = \langle f \circ_c \beta_Y, id\ Y\rangle \circ_c g$
proof −
have $\langle f, g\rangle = \langle f \circ_c \beta_Y \circ_c g, id\ Y \circ_c g\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis id-left-unit2 id-right-unit2 id-type one-unique-element*)
also have ... = $\langle f \circ_c \beta_Y, id\ Y\rangle \circ_c g$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
finally show *?thesis*.
qed

3.5.1 Cartesian Products as Pullbacks

The definition below corresponds to a definition stated between Definition 2.1.42 and Definition 2.1.43 in Halvorson.

definition *is-pullback* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cfunc* ⇒ *bool* **where**

is-pullback A B C D *ab bd ac cd* $\longleftrightarrow$
  $(ab : A \to B \wedge bd : B \to D \wedge ac : A \to C \wedge cd : C \to D \wedge bd \circ_c ab = cd \circ_c ac \wedge$
  $(\forall\ Z\ k\ h.\ (k : Z \to B \wedge h : Z \to C \wedge bd \circ_c k = cd \circ_c h) \longrightarrow$
   $(\exists!\ j.\ j : Z \to A \wedge ab \circ_c j = k \wedge ac \circ_c j = h)))$

lemma *pullback-unique*:
 assumes $ab : A \to B$ $bd : B \to D$ $ac : A \to C$ $cd : C \to D$
 assumes $k : Z \to B$ $h : Z \to C$
 assumes *is-pullback* A B C D *ab bd ac cd*
 shows $bd \circ_c k = cd \circ_c h \Longrightarrow (\exists!\ j.\ j : Z \to A \wedge ab \circ_c j = k \wedge ac \circ_c j = h)$
 using *assms* **unfolding** *is-pullback-def* **by** *simp*

lemma *pullback-iff-product*:
 assumes *terminal-object*(T)
 assumes *f-type*[*type-rule*]: $f : Y \to T$
 assumes *g-type*[*type-rule*]: $g : X \to T$
 shows (*is-pullback* P Y X T (pY) f (pX) g) = (*is-cart-prod* P pX pY X Y)
proof(*safe*)
 assume *pullback*: *is-pullback* P Y X T pY f pX g
 have *f-type*[*type-rule*]: $f : Y \to T$
  using *is-pullback-def pullback* **by** *force*
 have *g-type*[*type-rule*]: $g : X \to T$
  using *is-pullback-def pullback* **by** *force*
 show *is-cart-prod* P pX pY X Y
  unfolding *is-cart-prod-def*
 proof(*safe*)
  show *pX-type*[*type-rule*]: $pX : P \to X$
   using *pullback is-pullback-def* **by** *force*
  show *pY-type*[*type-rule*]: $pY : P \to Y$
   using *pullback is-pullback-def* **by** *force*
  show $\bigwedge x\ y\ Z.$
    $x : Z \to X \Longrightarrow$
    $y : Z \to Y \Longrightarrow$
    $\exists h.\ h : Z \to P \wedge$
      $pX \circ_c h = x \wedge pY \circ_c h = y \wedge (\forall h2.\ h2 : Z \to P \wedge pX \circ_c h2 = x \wedge pY \circ_c h2 = y \longrightarrow h2 = h)$
  proof –
   fix x y Z
   assume *x-type*[*type-rule*]: $x : Z \to X$
   assume *y-type*[*type-rule*]: $y : Z \to Y$
   have $\bigwedge Z\ k\ h.\ k : Z \to Y \Longrightarrow h : Z \to X \Longrightarrow f \circ_c k = g \circ_c h \Longrightarrow \exists j.\ j : Z \to P \wedge pY \circ_c j = k \wedge pX \circ_c j = h$
    using *is-pullback-def pullback* **by** *blast*
   then have $\exists h.\ h : Z \to P \wedge$
     $pX \circ_c h = x \wedge pY \circ_c h = y$
     by (*smt* (*verit*, *ccfv-threshold*) *assms cfunc-type-def codomain-comp domain-comp f-type g-type terminal-object-def x-type y-type*)
   then show $\exists h.\ h : Z \to P \wedge$

$pX \circ_c h = x \wedge pY \circ_c h = y \wedge (\forall h2.\ h2 : Z \to P \wedge pX \circ_c h2 = x \wedge pY \circ_c h2 = y \longrightarrow h2 = h)$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *comp-associative2 is-pullback-def pullback*)
qed
qed
next
assume *prod*: *is-cart-prod P pX pY X Y*
then show *is-pullback P Y X T pY f pX g*
unfolding *is-cart-prod-def is-pullback-def*
proof(*typecheck-cfuncs*, *safe*)
assume *pX-type*[*type-rule*]: $pX : P \to X$
assume *pY-type*[*type-rule*]: $pY : P \to Y$
show $f \circ_c pY = g \circ_c pX$
using *assms*(*1*) *terminal-object-def* **by** (*typecheck-cfuncs*, *auto*)
show $\bigwedge Z\ k\ h.\ k : Z \to Y \Longrightarrow h : Z \to X \Longrightarrow f \circ_c k = g \circ_c h \Longrightarrow \exists j.\ j : Z \to P \wedge pY \circ_c j = k \wedge pX \circ_c j = h$
using *is-cart-prod-def prod* **by** *blast*
show $\bigwedge Z\ j\ y.$
$pY \circ_c j : Z \to Y \Longrightarrow$
$pX \circ_c j : Z \to X \Longrightarrow$
$f \circ_c pY \circ_c j = g \circ_c pX \circ_c j \Longrightarrow j : Z \to P \Longrightarrow y : Z \to P \Longrightarrow pY \circ_c y = pY \circ_c j \Longrightarrow pX \circ_c y = pX \circ_c j \Longrightarrow j = y$
using *is-cart-prod-def prod* **by** *blast*
qed
qed

end

4 Equalizers and Subobjects

theory *Equalizer*
imports *Terminal*
begin

4.1 Equalizers

definition *equalizer* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cfunc* ⇒ *bool* **where**
equalizer E m f g $\longleftrightarrow (\exists\ X\ Y.\ (f : X \to Y) \wedge (g : X \to Y) \wedge (m : E \to X)$
$\wedge (f \circ_c m = g \circ_c m)$
$\wedge (\forall\ h\ F.\ ((h : F \to X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k.\ (k : F \to E) \wedge m \circ_c k = h)))$

lemma *equalizer-def2*:
assumes $f : X \to Y$ $g : X \to Y$ $m : E \to X$
shows *equalizer E m f g* $\longleftrightarrow ((f \circ_c m = g \circ_c m)$
$\wedge (\forall\ h\ F.\ ((h : F \to X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k.\ (k : F \to E) \wedge m \circ_c k = h)))$
using *assms* **unfolding** *equalizer-def* **by** (*auto simp add*: *cfunc-type-def*)

lemma *equalizer-eq*:
  assumes $f : X \to Y$ $g : X \to Y$ $m : E \to X$
  assumes *equalizer* E m f g
  shows $f \circ_c m = g \circ_c m$
  using *assms equalizer-def2* **by** *auto*

lemma *similar-equalizers*:
  assumes $f : X \to Y$ $g : X \to Y$ $m : E \to X$
  assumes *equalizer* E m f g
  assumes $h : F \to X$ $f \circ_c h = g \circ_c h$
  shows $\exists!\ k.\ k : F \to E \wedge m \circ_c k = h$
  using *assms equalizer-def2* **by** *auto*

The definition above and the axiomatization below correspond to Axiom 4 (Equalizers) in Halvorson.

axiomatization where
  equalizer-exists: $f : X \to Y \Longrightarrow g : X \to Y \Longrightarrow \exists\ E\ m.$ *equalizer* E m f g

lemma *equalizer-exists2*:
  assumes $f : X \to Y$ $g : X \to Y$
  shows $\exists\ E\ m.\ m : E \to X \wedge f \circ_c m = g \circ_c m \wedge (\forall\ h\ F.\ ((h : F \to X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k.\ (k : F \to E) \wedge m \circ_c k = h))$
proof −
  obtain E m **where** *equalizer* E m f g
    using *assms equalizer-exists* **by** *blast*
  then show *?thesis*
    unfolding *equalizer-def*
  proof (*intro exI*[**where** x=E], *intro exI*[**where** x=m], *safe*)
    fix X' Y'
    assume *f-type2*: $f : X' \to Y'$
    assume *g-type2*: $g : X' \to Y'$
    assume *m-type*: $m : E \to X'$
    assume *fm-eq-gm*: $f \circ_c m = g \circ_c m$
    assume *equalizer-unique*: $\forall h\ F.\ h : F \to X' \wedge f \circ_c h = g \circ_c h \longrightarrow (\exists!k.\ k : F \to E \wedge m \circ_c k = h)$

    show *m-type2*: $m : E \to X$
      using *assms*(*2*) *cfunc-type-def g-type2 m-type* **by** *auto*

    show $\bigwedge h\ F.\ h : F \to X \Longrightarrow f \circ_c h = g \circ_c h \Longrightarrow \exists k.\ k : F \to E \wedge m \circ_c k = h$
      by (*metis m-type2 cfunc-type-def equalizer-unique m-type*)

    show $\bigwedge F\ k\ y.\ m \circ_c k : F \to X \Longrightarrow f \circ_c m \circ_c k = g \circ_c m \circ_c k \Longrightarrow k : F \to E \Longrightarrow y : F \to E$
        $\Longrightarrow m \circ_c y = m \circ_c k \Longrightarrow k = y$
      using *comp-type equalizer-unique m-type* **by** *blast*
  qed
qed

The lemma below corresponds to Exercise 2.1.31 in Halvorson.

lemma *equalizers-isomorphic*:
assumes *equalizer* E m f g *equalizer* E' m' f g
shows $\exists\ k.\ k : E \to E' \wedge$ *isomorphism* $k \wedge m = m' \circ_c k$
proof −
have *fm-eq-gm*: $f \circ_c m = g \circ_c m$
using *assms*(*1*) *equalizer-def* **by** *blast*
have *fm'-eq-gm'*: $f \circ_c m' = g \circ_c m'$
using *assms*(*2*) *equalizer-def* **by** *blast*

obtain X Y **where** *f-type*: $f : X \to Y$ **and** *g-type*: $g : X \to Y$ **and** *m-type*: $m : E \to X$
using *assms*(*1*) **unfolding** *equalizer-def* **by** *auto*

obtain k **where** *k-type*: $k : E' \to E$ **and** *mk-eq-m'*: $m \circ_c k = m'$
by (*metis assms cfunc-type-def equalizer-def*)
obtain k' **where** *k'-type*: $k' : E \to E'$ **and** *m'k-eq-m*: $m' \circ_c k' = m$
by (*metis assms cfunc-type-def equalizer-def*)

have $f \circ_c m \circ_c k \circ_c k' = g \circ_c m \circ_c k \circ_c k'$
using *comp-associative2 m-type fm-eq-gm k'-type k-type m'k-eq-m mk-eq-m'* **by** *auto*

have $k \circ_c k' : E \to E \wedge m \circ_c k \circ_c k' = m$
using *comp-associative2 comp-type k'-type k-type m-type m'k-eq-m mk-eq-m'* **by** *auto*
then have *kk'-eq-id*: $k \circ_c k' =$ *id* E
using *assms*(*1*) *equalizer-def id-right-unit2 id-type* **by** *blast*

have $k' \circ_c k : E' \to E' \wedge m' \circ_c k' \circ_c k = m'$
by (*smt comp-associative2 comp-type k'-type k-type m'k-eq-m m-type mk-eq-m'*)
then have *k'k-eq-id*: $k' \circ_c k =$ *id* E'
using *assms*(*2*) *equalizer-def id-right-unit2 id-type* **by** *blast*

show $\exists k.\ k : E \to E' \wedge$ *isomorphism* $k \wedge m = m' \circ_c k$
using *cfunc-type-def isomorphism-def k'-type k'k-eq-id k-type kk'-eq-id m'k-eq-m* **by** (*intro exI*[**where** *x*=*k'*], *auto*)
qed

lemma *isomorphic-to-equalizer-is-equalizer*:
assumes φ: $E' \to E$
assumes *isomorphism* φ
assumes *equalizer* E m f g
assumes $f : X \to Y$
assumes $g : X \to Y$
assumes $m : E \to X$
shows *equalizer* E' $(m \circ_c \varphi)$ f g
proof −
obtain φ-*inv* **where** φ-*inv-type*[*type-rule*]: φ-*inv* $: E \to E'$ **and** φ-*inv*-φ: φ-*inv*

$\circ_c$ φ = *id*(E') **and** *φφ-inv*: $\varphi \circ_c \varphi$*-inv* = *id*(E)
    using *assms(1,2) cfunc-type-def isomorphism-def* **by** *auto*

  have *equalizes*: $f \circ_c m \circ_c \varphi = g \circ_c m \circ_c \varphi$
    using *assms comp-associative2 equalizer-def* **by** *force*
  have $\forall h\ F.\ h : F \to X \wedge f \circ_c h = g \circ_c h \longrightarrow (\exists!k.\ k : F \to E' \wedge (m \circ_c \varphi) \circ_c k = h)$
  proof(*safe*)
    fix *h F*
    assume *h-type*[*type-rule*]: $h : F \to X$
    assume *h-equalizes*: $f \circ_c h = g \circ_c h$
    have *k-exists-uniquely*: $\exists!\ k.\ k: F \to E \wedge m \circ_c k = h$
      using *assms equalizer-def2 h-equalizes* **by** (*typecheck-cfuncs*, *auto*)
    then obtain *k* **where** *k-type*[*type-rule*]: $k: F \to E$ **and** *k-def*: $m \circ_c k = h$
      by *blast*
    then show $\exists k.\ k : F \to E' \wedge (m \circ_c \varphi) \circ_c k = h$
     using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *φφ-inv φ-inv-type comp-associative2 comp-type id-right-unit2 k-exists-uniquely*)
  next
    fix *F k y*
    assume $(m \circ_c \varphi) \circ_c k : F \to X$
    assume $f \circ_c (m \circ_c \varphi) \circ_c k = g \circ_c (m \circ_c \varphi) \circ_c k$
    assume *k-type*[*type-rule*]: $k : F \to E'$
    assume *y-type*[*type-rule*]: $y : F \to E'$
    assume $(m \circ_c \varphi) \circ_c y = (m \circ_c \varphi) \circ_c k$
    then show $k = y$
      by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *assms(1,2,3) cfunc-type-def comp-associative comp-type equalizer-def id-left-unit2 isomorphism-def*)
  qed
  then show *?thesis*
    by (*smt* (*verit*, *best*) *assms(1,4,5,6) comp-type equalizer-def equalizes*)
qed

The lemma below corresponds to Exercise 2.1.34 in Halvorson.

lemma *equalizer-is-monomorphism*:
  equalizer E m f g $\Longrightarrow$ *monomorphism*(m)
  unfolding *equalizer-def monomorphism-def*
proof *clarify*
  fix *h1 h2 X Y*
  assume *f-type*: $f : X \to Y$
  assume *g-type*: $g : X \to Y$
  assume *m-type*: $m : E \to X$
  assume *fm-gm*: $f \circ_c m = g \circ_c m$
  assume *uniqueness*: $\forall h\ F.\ h : F \to X \wedge f \circ_c h = g \circ_c h \longrightarrow (\exists!k.\ k : F \to E \wedge m \circ_c k = h)$
  assume *relation-ga*: *codomain h1* = *domain m*
  assume *relation-h*: *codomain h2* = *domain m*
  assume *m-ga-mh*: $m \circ_c h1 = m \circ_c h2$
  have $f \circ_c m \circ_c h1 = g \circ_c m \circ_c h2$

using *cfunc-type-def comp-associative f-type fm-gm g-type m-ga-mh m-type relation-h* **by** *auto*
then obtain z **where** z: $domain(h1) \to E \wedge m \circ_c z = m \circ_c h1 \wedge$
$(\forall\ j.\ j{:}domain(h1) \to E \wedge\ m \circ_c j = m \circ_c h1 \longrightarrow j = z)$
using *uniqueness* **by** (*smt cfunc-type-def codomain-comp domain-comp m-ga-mh m-type relation-ga*)
then show $h1 = h2$
by (*metis cfunc-type-def domain-comp m-ga-mh m-type relation-ga relation-h*)
qed

The definition below corresponds to Definition 2.1.35 in Halvorson.

definition *regular-monomorphism* :: *cfunc* ⇒ *bool*
where *regular-monomorphism f* $\longleftrightarrow$
$(\exists\ g\ h.\ domain\ g = codomain\ f \wedge domain\ h = codomain\ f \wedge equalizer\ (domain\ f)\ f\ g\ h)$

The lemma below corresponds to Exercise 2.1.36 in Halvorson.

lemma *epi-regmon-is-iso*:
assumes *epimorphism f regular-monomorphism f*
shows *isomorphism f*
proof −
obtain *g h* **where** *g-type*: $domain\ g = codomain\ f$ **and**
h-type: $domain\ h = codomain\ f$ **and**
f-equalizer: $equalizer\ (domain\ f)\ f\ g\ h$
using *assms*(*2*) *regular-monomorphism-def* **by** *auto*
then have $g \circ_c f = h \circ_c f$
using *equalizer-def* **by** *blast*
then have $g = h$
using *assms*(*1*) *cfunc-type-def epimorphism-def equalizer-def f-equalizer* **by** *auto*
then have $g \circ_c id(codomain\ f) = h \circ_c id(codomain\ f)$
by *simp*
then obtain k **where** *k-type*: $f \circ_c k = id(codomain(f)) \wedge codomain\ k = domain\ f$
by (*metis cfunc-type-def equalizer-def f-equalizer id-type*)
then have $f \circ_c id(domain(f)) = f \circ_c (k \circ_c f)$
by (*metis comp-associative domain-comp id-domain id-left-unit id-right-unit*)
then have *monomorphism f* $\Longrightarrow k \circ_c f = id(domain\ f)$
by (*metis* (*mono-tags*) *codomain-comp domain-comp id-codomain id-domain k-type monomorphism-def*)
then have $k \circ_c f = id(domain\ f)$
using *equalizer-is-monomorphism f-equalizer* **by** *blast*
then show *isomorphism f*
by (*metis domain-comp id-domain isomorphism-def k-type*)
qed

4.2 Subobjects

The definition below corresponds to Definition 2.1.32 in Halvorson.

definition *factors-through* :: *cfunc* ⇒ *cfunc* ⇒ *bool* (**infix** *factorsthru 90*)

where *g factorsthru f* ⟷ (∃ *h*. (*h*: *domain*(*g*)→ *domain*(*f*)) ∧ *f* $\circ_c$ *h* = *g*)

lemma *factors-through-def2*:
assumes *g* : *X* → *Z f* : *Y* → *Z*
shows *g factorsthru f* ⟷ (∃ *h*. *h*: *X* → *Y* ∧ *f* $\circ_c$ *h* = *g*)
unfolding *factors-through-def* **using** *assms* **by** (*simp add*: *cfunc-type-def*)

The lemma below corresponds to Exercise 2.1.33 in Halvorson.

lemma *xfactorthru-equalizer-iff-fx-eq-gx*:
assumes *f*: *X*→ *Y g*:*X* → *Y equalizer E m f g x*$\in_c$ *X*
shows *x factorsthru m* ⟷ *f* $\circ_c$ *x* = *g* $\circ_c$ *x*
proof *safe*
assume *LHS*: *x factorsthru m*
then show *f* $\circ_c$ *x* = *g* $\circ_c$ *x*
using *assms*(*3*) *cfunc-type-def comp-associative equalizer-def factors-through-def*
by *auto*
next
assume *RHS*: *f* $\circ_c$ *x* = *g* $\circ_c$ *x*
then show *x factorsthru m*
unfolding *cfunc-type-def factors-through-def*
by (*metis RHS assms*(*1*,*3*,*4*) *cfunc-type-def equalizer-def*)
qed

The definition below corresponds to Definition 2.1.37 in Halvorson.

definition *subobject-of* :: *cset* × *cfunc* ⇒ *cset* ⇒ *bool* (**infix** $\subseteq_c$ *50*)
where *B* $\subseteq_c$ *X* ⟷ (*snd B* : *fst B* → *X* ∧ *monomorphism* (*snd B*))

lemma *subobject-of-def2*:
(*B*,*m*) $\subseteq_c$ *X* = (*m* : *B* → *X* ∧ *monomorphism m*)
by (*simp add*: *subobject-of-def*)

definition *relative-subset* :: *cset* × *cfunc* ⇒ *cset* ⇒ *cset* × *cfunc* ⇒ *bool* (-$\subseteq_-$-[*51*,*50*,*51*]*50*)
where *B* $\subseteq_X$ *A* ⟷
(*snd B* : *fst B* → *X* ∧ *monomorphism* (*snd B*) ∧ *snd A* : *fst A* → *X* ∧ *monomorphism* (*snd A*)
∧ (∃ *k*. *k*: *fst B* → *fst A* ∧ *snd A* $\circ_c$ *k* = *snd B*))

lemma *relative-subset-def2*:
(*B*,*m*) $\subseteq_X$ (*A*,*n*) = (*m* : *B* → *X* ∧ *monomorphism m* ∧ *n* : *A* → *X* ∧ *monomorphism n*
∧ (∃ *k*. *k*: *B* → *A* ∧ *n* $\circ_c$ *k* = *m*))
unfolding *relative-subset-def* **by** *auto*

lemma *subobject-is-relative-subset*: (*B*,*m*) $\subseteq_c$ *A* ⟷ (*B*,*m*) $\subseteq_A$ (*A*, *id*(*A*))
unfolding *relative-subset-def2 subobject-of-def2*
using *cfunc-type-def id-isomorphism id-left-unit id-type iso-imp-epi-and-monic*
by *auto*

The definition below corresponds to Definition 2.1.39 in Halvorson.

definition *relative-member* :: *cfunc* ⇒ *cset* ⇒ *cset* × *cfunc* ⇒ *bool* (- $\in_{-}$ - [*51*,*50*,*51*]*50*)
where
$x \in_X B \longleftrightarrow (x \in_c X \wedge$ *monomorphism* (*snd* B) $\wedge$ *snd* B : *fst* $B \rightarrow X \wedge x$ *factorsthru* (*snd* B))

lemma *relative-member-def2*:
$x \in_X (B, m) = (x \in_c X \wedge$ *monomorphism* $m \wedge m : B \rightarrow X \wedge x$ *factorsthru* m)
unfolding *relative-member-def* **by** *auto*

The lemma below corresponds to Proposition 2.1.40 in Halvorson.

lemma *relative-subobject-member*:
assumes $(A,n) \subseteq_X (B,m)$ $x \in_c X$
shows $x \in_X (A,n) \Longrightarrow x \in_X (B,m)$
using *assms* **unfolding** *relative-member-def2 relative-subset-def2*
proof *clarify*
fix k
assume *m-type*: $m : B \rightarrow X$
assume *k-type*: $k : A \rightarrow B$
assume *m-monomorphism*: *monomorphism* m
assume *mk-monomorphism*: *monomorphism* $(m \circ_c k)$
assume *n-eq-mk*: $n = m \circ_c k$
assume *factorsthru-mk*: x *factorsthru* $(m \circ_c k)$

obtain a **where** *a-assms*: $a \in_c A \wedge (m \circ_c k) \circ_c a = x$
using *assms*(*2*) *cfunc-type-def domain-comp factors-through-def factorsthru-mk k-type m-type* **by** *auto*
then show x *factorsthru* m
unfolding *factors-through-def*
using *cfunc-type-def comp-type k-type m-type comp-associative*
by (*intro exI*[**where** $x{=}k \circ_c a$], *auto*)
qed

4.3 Inverse Image

The definition below corresponds to a definition given by a diagram between Definition 2.1.37 and Proposition 2.1.38 in Halvorson.

definition *inverse-image* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cset* ($-^{-1}(\!|-|\!)_{-}$ [*101*,*0*,*0*]*100*)
where
inverse-image f B m = (*SOME* A. $\exists$ X Y k. $f : X \rightarrow Y \wedge m : B \rightarrow Y \wedge$ *monomorphism* $m \wedge$
equalizer A k $(f \circ_c$ *left-cart-proj* X B) $(m \circ_c$ *right-cart-proj* X B))

lemma *inverse-image-is-equalizer*:
assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism* m
shows $\exists k$. *equalizer* $(f^{-1}(\!|B|\!)_m)$ k $(f \circ_c$ *left-cart-proj* X B) $(m \circ_c$ *right-cart-proj* X B)
proof −
obtain A k **where** *equalizer* A k $(f \circ_c$ *left-cart-proj* X B) $(m \circ_c$ *right-cart-proj* X B)

by (*meson assms*(*1*,*2*) *comp-type equalizer-exists left-cart-proj-type right-cart-proj-type*)
then show ∃ *k*. *equalizer* (*inverse-image f B m*) *k* (*f* $\circ_c$ *left-cart-proj X B*) (*m* $\circ_c$ *right-cart-proj X B*)
unfolding *inverse-image-def* **using** *assms cfunc-type-def* **by** (*subst someI2-ex*, *auto*)
qed

definition *inverse-image-mapping* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cfunc* **where**
inverse-image-mapping f B m = (*SOME k*. ∃ *X Y*. *f* : *X* → *Y* ∧ *m* : *B* → *Y* ∧ *monomorphism m* ∧
equalizer (*inverse-image f B m*) *k* (*f* $\circ_c$ *left-cart-proj X B*) (*m* $\circ_c$ *right-cart-proj X B*))

lemma *inverse-image-is-equalizer2*:
assumes *m* : *B* → *Y f* : *X* → *Y monomorphism m*
shows *equalizer* (*inverse-image f B m*) (*inverse-image-mapping f B m*) (*f* $\circ_c$ *left-cart-proj X B*) (*m* $\circ_c$ *right-cart-proj X B*)
proof −
obtain *k* **where** *equalizer* (*inverse-image f B m*) *k* (*f* $\circ_c$ *left-cart-proj X B*) (*m* $\circ_c$ *right-cart-proj X B*)
using *assms inverse-image-is-equalizer* **by** *blast*
then have ∃ *X Y*. *f* : *X* → *Y* ∧ *m* : *B* → *Y* ∧ *monomorphism m* ∧
equalizer (*inverse-image f B m*) (*inverse-image-mapping f B m*) (*f* $\circ_c$ *left-cart-proj X B*) (*m* $\circ_c$ *right-cart-proj X B*)
unfolding *inverse-image-mapping-def* **using** *assms* **by** (*subst someI-ex*, *auto*)
then show *equalizer* (*inverse-image f B m*) (*inverse-image-mapping f B m*) (*f* $\circ_c$ *left-cart-proj X B*) (*m* $\circ_c$ *right-cart-proj X B*)
using *assms*(*2*) *cfunc-type-def* **by** *auto*
qed

lemma *inverse-image-mapping-type*[*type-rule*]:
assumes *m* : *B* → *Y f* : *X* → *Y monomorphism m*
shows *inverse-image-mapping f B m* : (*inverse-image f B m*) → *X* $\times_c$ *B*
using *assms cfunc-type-def domain-comp equalizer-def inverse-image-is-equalizer2 left-cart-proj-type* **by** *auto*

lemma *inverse-image-mapping-eq*:
assumes *m* : *B* → *Y f* : *X* → *Y monomorphism m*
shows *f* $\circ_c$ *left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*
= *m* $\circ_c$ *right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*
using *assms cfunc-type-def comp-associative equalizer-def inverse-image-is-equalizer2*
by (*typecheck-cfuncs*, *smt* (*verit*))

lemma *inverse-image-mapping-monomorphism*:
assumes *m* : *B* → *Y f* : *X* → *Y monomorphism m*
shows *monomorphism* (*inverse-image-mapping f B m*)
using *assms equalizer-is-monomorphism inverse-image-is-equalizer2* **by** *blast*

The lemma below is the dual of Proposition 2.1.38 in Halvorson.

lemma *inverse-image-monomorphism*:
assumes $m : B \to Y$ $f : X \to Y$ *monomorphism m*
shows *monomorphism* (*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*)
using *assms*
proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
fix *g h A*
assume *g-type*: $g : A \to (f^{-1}(\!|B|\!)_m)$
assume *h-type*: $h : A \to (f^{-1}(\!|B|\!)_m)$
assume *left-eq*: (*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *g*
= (*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *h*
then have *f* $\circ_c$ (*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *g*
= *f* $\circ_c$ (*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *h*
by *auto*
then have *m* $\circ_c$ (*right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *g*
= *m* $\circ_c$ (*right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *h*
using *assms g-type h-type*
by (*typecheck-cfuncs*, *smt cfunc-type-def codomain-comp comp-associative domain-comp inverse-image-mapping-eq left-cart-proj-type*)
then have *right-eq*: (*right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *g*
= (*right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*) $\circ_c$ *h*
using *assms g-type h-type monomorphism-def3* **by** (*typecheck-cfuncs*, *auto*)
then have *inverse-image-mapping f B m* $\circ_c$ *g* = *inverse-image-mapping f B m* $\circ_c$ *h*
using *assms g-type h-type cfunc-type-def comp-associative left-eq left-cart-proj-type right-cart-proj-type*
by (*typecheck-cfuncs*, *subst cart-prod-eq*, *auto*)
then show *g* = *h*
using *assms g-type h-type inverse-image-mapping-monomorphism inverse-image-mapping-type monomorphism-def3*
by *blast*
qed

definition *inverse-image-subobject-mapping* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cfunc*
([-$^{-1}$(\!|-|\!)-]*map* [*101*,*0*,*0*]*100*) **where**
[$f^{-1}(\!|B|\!)_m$]*map* = *left-cart-proj* (*domain f*) *B* $\circ_c$ *inverse-image-mapping f B m*

lemma *inverse-image-subobject-mapping-def2*:
assumes $f : X \to Y$
shows [$f^{-1}(\!|B|\!)_m$]*map* = *left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*
using *assms* **unfolding** *inverse-image-subobject-mapping-def cfunc-type-def* **by** *auto*

lemma *inverse-image-subobject-mapping-type*[*type-rule*]:
assumes $f : X \to Y$ $m : B \to Y$ *monomorphism m*
shows [$f^{-1}(\!|B|\!)_m$]*map* : $f^{-1}(\!|B|\!)_m \to X$
by (*smt* (*verit*, *best*) *assms comp-type inverse-image-mapping-type inverse-image-subobject-mapping-def2 left-cart-proj-type*)

lemma *inverse-image-subobject-mapping-mono*:

assumes $f : X \to Y$ $m : B \to Y$ *monomorphism m*
shows *monomorphism* $([f^{-1}(\!|B|\!)_m]map)$
using *assms cfunc-type-def inverse-image-monomorphism inverse-image-subobject-mapping-def*
by *fastforce*

lemma *inverse-image-subobject*:
assumes $m : B \to Y$ $f : X \to Y$ *monomorphism m*
shows $(f^{-1}(\!|B|\!)_m, [f^{-1}(\!|B|\!)_m]map) \subseteq_c X$
unfolding *subobject-of-def2*
using *assms inverse-image-subobject-mapping-mono inverse-image-subobject-mapping-type*
by *force*

lemma *inverse-image-pullback*:
assumes $m : B \to Y$ $f : X \to Y$ *monomorphism m*
shows *is-pullback* $(f^{-1}(\!|B|\!)_m)$ B X Y
(*right-cart-proj* X B $\circ_c$ *inverse-image-mapping* f B m) m
(*left-cart-proj* X B $\circ_c$ *inverse-image-mapping* f B m) f
unfolding *is-pullback-def* **using** *assms*
proof *safe*
show *right-type*: *right-cart-proj* X B $\circ_c$ *inverse-image-mapping* f B m : $f^{-1}(\!|B|\!)_m$
$\to B$
using *assms cfunc-type-def codomain-comp domain-comp inverse-image-mapping-type right-cart-proj-type* **by** *auto*
show *left-type*: *left-cart-proj* X B $\circ_c$ *inverse-image-mapping* f B m : $f^{-1}(\!|B|\!)_m \to$
X
using *assms fst-conv inverse-image-subobject subobject-of-def* **by** (*typecheck-cfuncs*)

show $m \circ_c$ *right-cart-proj* X B $\circ_c$ *inverse-image-mapping* f B m =
$f \circ_c$ *left-cart-proj* X B $\circ_c$ *inverse-image-mapping* f B m
using *assms inverse-image-mapping-eq* **by** *auto*
next
fix Z k h
assume *k-type*: $k : Z \to B$ **and** *h-type*: $h : Z \to X$
assume *mk-eq-fh*: $m \circ_c k = f \circ_c h$

have *equalizer* $(f^{-1}(\!|B|\!)_m)$ (*inverse-image-mapping* f B m) ($f \circ_c$ *left-cart-proj* X
B) ($m \circ_c$ *right-cart-proj* X B)
using *assms inverse-image-is-equalizer2* **by** *blast*
then have $\forall h\ F.\ h : F \to (X \times_c B)$
$\wedge$ ($f \circ_c$ *left-cart-proj* X B) $\circ_c h$ = ($m \circ_c$ *right-cart-proj* X B) $\circ_c h \longrightarrow$
$(\exists!u.\ u : F \to (f^{-1}(\!|B|\!)_m) \wedge$ *inverse-image-mapping* f B $m \circ_c u = h)$
unfolding *equalizer-def* **using** *assms*(2) *cfunc-type-def domain-comp left-cart-proj-type*
by *auto*
then have $\langle h,k\rangle : Z \to X \times_c B \implies$
($f \circ_c$ *left-cart-proj* X B) $\circ_c \langle h,k\rangle$ = ($m \circ_c$ *right-cart-proj* X B) $\circ_c \langle h,k\rangle \implies$
$(\exists!u.\ u : Z \to (f^{-1}(\!|B|\!)_m) \wedge$ *inverse-image-mapping* f B $m \circ_c u = \langle h,k\rangle)$
by *auto*
then have $\exists!u.\ u : Z \to (f^{-1}(\!|B|\!)_m) \wedge$ *inverse-image-mapping* f B $m \circ_c u =$
$\langle h,k\rangle$

using *k-type h-type assms*
by (*typecheck-cfuncs*, *smt comp-associative2 left-cart-proj-cfunc-prod left-cart-proj-type mk-eq-fh right-cart-proj-cfunc-prod right-cart-proj-type*)
then show $\exists j.\ j : Z \rightarrow (f^{-1}(\!|B|\!)_m) \wedge$
$(\textit{right-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j = k \wedge$
$(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j = h$
proof (*clarify*)
fix *u*
assume *u-type*[*type-rule*]: $u : Z \rightarrow (f^{-1}(\!|B|\!)_m)$
assume *u-eq*: $\textit{inverse-image-mapping}\ f\ B\ m \circ_c u = \langle h,k\rangle$

show $\exists j.\ j : Z \rightarrow f^{-1}(\!|B|\!)_m \wedge$
$(\textit{right-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j = k \wedge$
$(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j = h$
proof (*rule exI*[**where** *x*=*u*], *typecheck-cfuncs*, *safe*)

show $(\textit{right-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c u = k$
using *assms u-type h-type k-type u-eq*
by (*typecheck-cfuncs*, *metis* (*full-types*) *comp-associative2 right-cart-proj-cfunc-prod*)

show $(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c u = h$
using *assms u-type h-type k-type u-eq*
by (*typecheck-cfuncs*, *metis* (*full-types*) *comp-associative2 left-cart-proj-cfunc-prod*)
qed
qed
next
fix *Z j y*
assume *j-type*: $j : Z \rightarrow (f^{-1}(\!|B|\!)_m)$
assume *y-type*: $y : Z \rightarrow (f^{-1}(\!|B|\!)_m)$
assume $(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c y =$
$(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j$
then show $j = y$
using *assms j-type y-type inverse-image-mapping-type comp-type*
by (*smt* (*verit*, *ccfv-threshold*) *inverse-image-monomorphism left-cart-proj-type monomorphism-def3*)
qed

The lemma below corresponds to Proposition 2.1.41 in Halvorson.

lemma *in-inverse-image*:
assumes $f : X \rightarrow Y\ (B,m) \subseteq_c Y\ x \in_c X$
shows $(x \in_X (f^{-1}(\!|B|\!)_m, \textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m)) = (f \circ_c x \in_Y (B,m))$
proof
have *m-type*: $m : B \rightarrow Y$ *monomorphism m*
using *assms*(*2*) **unfolding** *subobject-of-def2* **by** *auto*

assume $x \in_X (f^{-1}(\!|B|\!)_m, \textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m)$
then obtain *h* **where** *h-type*: $h \in_c (f^{-1}(\!|B|\!)_m)$
and *h-def*: $(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c h = x$

unfolding *relative-member-def2 factors-through-def* **by** (*auto simp add*: *cfunc-type-def*)
then have $f \circ_c x = f \circ_c$ *left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m* $\circ_c h$
using *assms m-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 h-def*)
then have $f \circ_c x = (f \circ_c$ *left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*$) \circ_c$
h
using *assms m-type h-type h-def comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
then have $f \circ_c x = (m \circ_c$ *right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*$)$
$\circ_c h$
using *assms h-type m-type* **by** (*typecheck-cfuncs*, *simp add*: *inverse-image-mapping-eq m-type*)
then have $f \circ_c x = m \circ_c$ *right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*
$\circ_c h$
using *assms m-type h-type* **by** (*typecheck-cfuncs*, *smt cfunc-type-def comp-associative domain-comp*)
then have $(f \circ_c x)$ *factorsthru m*
unfolding *factors-through-def* **using** *assms h-type m-type*
by (*intro exI*[**where** *x=right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m* $\circ_c$
h],
typecheck-cfuncs, *auto simp add*: *cfunc-type-def*)
then show $f \circ_c x \in_Y (B, m)$
unfolding *relative-member-def2* **using** *assms m-type* **by** (*typecheck-cfuncs*, *auto*)
next
have *m-type*: $m : B \to Y$ *monomorphism m*
using *assms*(*2*) **unfolding** *subobject-of-def2* **by** *auto*

assume $f \circ_c x \in_Y (B, m)$
then have $\exists h.\ h :$ *domain* $(f \circ_c x) \to$ *domain* $m \land m \circ_c h = f \circ_c x$
unfolding *relative-member-def2 factors-through-def* **by** *auto*
then obtain h **where** *h-type*: $h \in_c B$ **and** *h-def*: $m \circ_c h = f \circ_c x$
unfolding *relative-member-def2 factors-through-def*
using *assms cfunc-type-def domain-comp m-type* **by** *auto*
then have $\exists j.\ j \in_c (f^{-1}(\!|B|\!)_m) \land$
(*right-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*$) \circ_c j = h \land$
(*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*$) \circ_c j = x$
using *inverse-image-pullback assms m-type* **unfolding** *is-pullback-def* **by** *blast*
then have *x factorsthru* (*left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*)
using *m-type assms cfunc-type-def* **by** (*typecheck-cfuncs*, *unfold factors-through-def*, *auto*)
then show $x \in_X (f^{-1}(\!|B|\!)_m$, *left-cart-proj X B* $\circ_c$ *inverse-image-mapping f B m*)
unfolding *relative-member-def2* **using** *m-type assms*
by (*typecheck-cfuncs*, *simp add*: *inverse-image-monomorphism*)
qed

4.4 Fibered Products

The definition below corresponds to Definition 2.1.42 in Halvorson.

definition *fibered-product* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cset* (- $_{-}\times_{c-}$ - [*66*,*50*,*50*,*65*]*65*) **where**

$X \ {}_f{\times}_c{}_g \ Y = (SOME\ E.\ \exists\ Z\ m.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
equalizer E m $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$))$

lemma *fibered-product-equalizer*:
assumes $f : X \rightarrow Z$ $g : Y \rightarrow Z$
shows $\exists$ *m. equalizer* $(X \ {}_f{\times}_c{}_g \ Y)$ *m* $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
proof –
obtain *E m* **where** *equalizer E m* $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
using *assms equalizer-exists* **by** (*typecheck-cfuncs*, *blast*)
then have $\exists\, x\ Z\ m.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
equalizer x m $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
using *assms* **by** *blast*
then have $\exists\ Z\ m.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
equalizer $(X \ {}_f{\times}_c{}_g \ Y)$ *m* $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
unfolding *fibered-product-def* **by** (*rule someI-ex*)
then show $\exists$ *m. equalizer* $(X \ {}_f{\times}_c{}_g \ Y)$ *m* $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
by *auto*
qed

definition *fibered-product-morphism* :: *cset* $\Rightarrow$ *cfunc* $\Rightarrow$ *cfunc* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc*
where
fibered-product-morphism X f g Y $= (SOME\ m.\ \exists\ Z.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
equalizer $(X \ {}_f{\times}_c{}_g \ Y)$ *m* $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$))$

lemma *fibered-product-morphism-equalizer*:
assumes $f : X \rightarrow Z$ $g : Y \rightarrow Z$
shows *equalizer* $(X \ {}_f{\times}_c{}_g \ Y)$ (*fibered-product-morphism X f g Y*) $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
proof –
have $\exists\, x\ Z.\ f : X \rightarrow Z \wedge$
$g : Y \rightarrow Z \wedge$ *equalizer* $(X \ {}_f{\times}_c{}_g \ Y)$ *x* $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
using *assms fibered-product-equalizer* **by** *blast*
then have $\exists\, Z.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
equalizer $(X \ {}_f{\times}_c{}_g \ Y)$ (*fibered-product-morphism X f g Y*) $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
unfolding *fibered-product-morphism-def* **by** (*rule someI-ex*)
then show *equalizer* $(X \ {}_f{\times}_c{}_g \ Y)$ (*fibered-product-morphism X f g Y*) $(f \circ_c$ *left-cart-proj X Y*$)$ $(g \circ_c$ *right-cart-proj X Y*$)$
by *auto*
qed

lemma *fibered-product-morphism-type*[*type-rule*]:
assumes $f : X \rightarrow Z$ $g : Y \rightarrow Z$
shows *fibered-product-morphism X f g Y* : $X \ {}_f{\times}_c{}_g \ Y \rightarrow X \times_c Y$
using *assms cfunc-type-def domain-comp equalizer-def fibered-product-morphism-equalizer*

left-cart-proj-type **by** *auto*

lemma *fibered-product-morphism-monomorphism*:
assumes $f : X \to Z$ $g : Y \to Z$
shows *monomorphism* (*fibered-product-morphism* X f g Y)
using *assms equalizer-is-monomorphism fibered-product-morphism-equalizer* **by** *blast*

definition *fibered-product-left-proj* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cfunc* **where**
fibered-product-left-proj X f g Y = (*left-cart-proj* X Y) $\circ_c$ (*fibered-product-morphism* X f g Y)

lemma *fibered-product-left-proj-type*[*type-rule*]:
assumes $f : X \to Z$ $g : Y \to Z$
shows *fibered-product-left-proj* X f g Y : $X\ {}_f{\times}_{cg}\ Y \to X$
by (*metis assms comp-type fibered-product-left-proj-def fibered-product-morphism-type left-cart-proj-type*)

definition *fibered-product-right-proj* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cfunc* **where**
fibered-product-right-proj X f g Y = (*right-cart-proj* X Y) $\circ_c$ (*fibered-product-morphism* X f g Y)

lemma *fibered-product-right-proj-type*[*type-rule*]:
assumes $f : X \to Z$ $g : Y \to Z$
shows *fibered-product-right-proj* X f g Y : $X\ {}_f{\times}_{cg}\ Y \to Y$
by (*metis assms comp-type fibered-product-right-proj-def fibered-product-morphism-type right-cart-proj-type*)

lemma *pair-factorsthru-fibered-product-morphism*:
assumes $f : X \to Z$ $g : Y \to Z$ $x : A \to X$ $y : A \to Y$
shows $f \circ_c x = g \circ_c y \Longrightarrow \langle x,y \rangle$ *factorsthru fibered-product-morphism* X f g Y
unfolding *factors-through-def*
proof −
have *equalizer*: *equalizer* ($X\ {}_f{\times}_{cg}\ Y$) (*fibered-product-morphism* X f g Y) ($f \circ_c$ *left-cart-proj* X Y) ($g \circ_c$ *right-cart-proj* X Y)
using *fibered-product-morphism-equalizer assms* **by** (*typecheck-cfuncs*, *auto*)

assume $f \circ_c x = g \circ_c y$
then have ($f \circ_c$ *left-cart-proj* X Y) $\circ_c \langle x,y \rangle$ = ($g \circ_c$ *right-cart-proj* X Y) $\circ_c \langle x,y \rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
then have $\exists!\ h.\ h : A \to X\ {}_f{\times}_{cg}\ Y \wedge$ *fibered-product-morphism* X f g Y $\circ_c h = \langle x,y \rangle$
using *assms similar-equalizers* **by** (*typecheck-cfuncs*, *smt* (*verit*, *del-insts*) *cfunc-type-def equalizer equalizer-def*)
then show $\exists h.\ h :$ *domain* $\langle x,y \rangle \to$ *domain* (*fibered-product-morphism* X f g Y) $\wedge$

fibered-product-morphism X f g Y $\circ_c$ *h* = ⟨*x*,*y*⟩
by (*metis assms*(*1*,*2*) *cfunc-type-def domain-comp fibered-product-morphism-type*)
qed

lemma *fibered-product-is-pullback*:
assumes *f-type*[*type-rule*]: *f* : *X* → *Z* **and** *g-type*[*type-rule*]: *g* : *Y* → *Z*
shows *is-pullback* (*X* $_f\times_{cg}$ *Y*) *Y X Z* (*fibered-product-right-proj X f g Y*) *g* (*fibered-product-left-proj X f g Y*) *f*
unfolding *is-pullback-def*
using *assms fibered-product-left-proj-type fibered-product-right-proj-type*
proof *safe*
show *g* $\circ_c$ *fibered-product-right-proj X f g Y* = *f* $\circ_c$ *fibered-product-left-proj X f g Y*
unfolding *fibered-product-right-proj-def fibered-product-left-proj-def*
using *cfunc-type-def comp-associative2 equalizer-def fibered-product-morphism-equalizer*
by (*typecheck-cfuncs*, *auto*)
next
fix *A k h*
assume *k-type*: *k* : *A* → *Y* **and** *h-type*: *h* : *A* → *X*
assume *k-h-commutes*: *g* $\circ_c$ *k* = *f* $\circ_c$ *h*

have ⟨*h*,*k*⟩ *factorsthru fibered-product-morphism X f g Y*
using *assms h-type k-h-commutes k-type pair-factorsthru-fibered-product-morphism*
by *auto*
then have *f1*: ∃ *j*. *j* : *A* → *X* $_f\times_{cg}$ *Y* ∧ *fibered-product-morphism X f g Y* $\circ_c$ *j* = ⟨*h*,*k*⟩
by (*meson assms cfunc-prod-type factors-through-def2 fibered-product-morphism-type h-type k-type*)
then show ∃ *j*. *j* : *A* → *X* $_f\times_{cg}$ *Y* ∧
fibered-product-right-proj X f g Y $\circ_c$ *j* = *k* ∧ *fibered-product-left-proj X f g Y* $\circ_c$ *j* = *h*
unfolding *fibered-product-right-proj-def fibered-product-left-proj-def*
proof (*clarify*, *safe*)
fix *j*
assume *j-type*: *j* : *A* → *X* $_f\times_{cg}$ *Y*

show ∃ *j*. *j* : *A* → *X* $_f\times_{cg}$ *Y* ∧
(*right-cart-proj X Y* $\circ_c$ *fibered-product-morphism X f g Y*) $\circ_c$ *j* = *k* ∧ (*left-cart-proj X Y* $\circ_c$ *fibered-product-morphism X f g Y*) $\circ_c$ *j* = *h*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *f1 comp-associative2 h-type k-type left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod*)
qed
next
fix *A j y*
assume *j-type*: *j* : *A* → *X* $_f\times_{cg}$ *Y* **and** *y-type*: *y* : *A* → *X* $_f\times_{cg}$ *Y*
assume *fibered-product-right-proj X f g Y* $\circ_c$ *y* = *fibered-product-right-proj X f g Y* $\circ_c$ *j*
then have *right-eq*: *right-cart-proj X Y* $\circ_c$ (*fibered-product-morphism X f g Y* $\circ_c$ *y*) =

right-cart-proj X Y $\circ_c$ (*fibered-product-morphism X f g Y* $\circ_c$ *j*)
unfolding *fibered-product-right-proj-def* **using** *assms j-type y-type*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
assume *fibered-product-left-proj X f g Y* $\circ_c$ *y* = *fibered-product-left-proj X f g Y* $\circ_c$ *j*
then have *left-eq*: *left-cart-proj X Y* $\circ_c$ (*fibered-product-morphism X f g Y* $\circ_c$ *y*) =
left-cart-proj X Y $\circ_c$ (*fibered-product-morphism X f g Y* $\circ_c$ *j*)
unfolding *fibered-product-left-proj-def* **using** *assms j-type y-type*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)

have *mono*: *monomorphism* (*fibered-product-morphism X f g Y*)
using *assms fibered-product-morphism-monomorphism* **by** *auto*

have *fibered-product-morphism X f g Y* $\circ_c$ *y* = *fibered-product-morphism X f g Y* $\circ_c$ *j*
using *right-eq left-eq cart-prod-eq fibered-product-morphism-type y-type j-type assms comp-type*
by (*subst cart-prod-eq*[**where** *Z*=*A*, **where** *X*=*X*, **where** *Y*=*Y*], *auto*)
then show *j* = *y*
using *mono assms cfunc-type-def fibered-product-morphism-type j-type y-type*
unfolding *monomorphism-def*
by *auto*
qed

lemma *fibered-product-proj-eq*:
assumes *f* : *X* → *Z* *g* : *Y* → *Z*
shows *f* $\circ_c$ *fibered-product-left-proj X f g Y* = *g* $\circ_c$ *fibered-product-right-proj X f g Y*
using *fibered-product-is-pullback assms*
unfolding *is-pullback-def* **by** *auto*

lemma *fibered-product-pair-member*:
assumes *f* : *X* → *Z* *g* : *Y* → *Z* *x* $\in_c$ *X* *y* $\in_c$ *Y*
shows ($\langle x, y\rangle \in_{X \times_c Y}$ ($X_f{\times_c}_g Y$, *fibered-product-morphism X f g Y*)) = (*f* $\circ_c$ *x* = *g* $\circ_c$ *y*)
proof
assume $\langle x,y\rangle \in_{X \times_c Y}$ ($X\ _f{\times_c}_g\ Y$, *fibered-product-morphism X f g Y*)
then obtain *h* **where**
h-type: *h* $\in_c$ $X_f{\times_c}_g Y$ **and** *h-eq*: *fibered-product-morphism X f g Y* $\circ_c$ *h* = $\langle x,y\rangle$
unfolding *relative-member-def2 factors-through-def*
using *assms*(*3*,*4*) *cfunc-prod-type cfunc-type-def* **by** *auto*

have *left-eq*: *fibered-product-left-proj X f g Y* $\circ_c$ *h* = *x*
unfolding *fibered-product-left-proj-def*
using *assms h-type*
by (*typecheck-cfuncs*, *smt comp-associative2 h-eq left-cart-proj-cfunc-prod*)

have *right-eq*: *fibered-product-right-proj X f g Y* $\circ_c$ *h* = *y*

unfolding *fibered-product-right-proj-def*
using *assms h-type*
by (*typecheck-cfuncs*, *smt comp-associative2 h-eq right-cart-proj-cfunc-prod*)

have $f \circ_c$ *fibered-product-left-proj* $X\ f\ g\ Y \circ_c h = g \circ_c$ *fibered-product-right-proj* $X\ f\ g\ Y \circ_c h$
using *assms h-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 fibered-product-proj-eq*)
then show $f \circ_c x = g \circ_c y$
using *left-eq right-eq* **by** *auto*
next
assume *f-g-eq*: $f \circ_c x = g \circ_c y$
show $\langle x,y\rangle \in_{X \times_c Y} (X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y)$
unfolding *relative-member-def factors-through-def*
proof (*safe*)
show $\langle x,y\rangle \in_c X \times_c Y$
using *assms* **by** *typecheck-cfuncs*
show *monomorphism* (*snd* $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y))$
using *assms*(*1*,*2*) *fibered-product-morphism-monomorphism* **by** *auto*
show *snd* $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y)$: *fst* $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y) \to X \times_c Y$
using *assms*(*1*,*2*) *fibered-product-morphism-type* **by** *force*
have *j-exists*: $\bigwedge Z\ k\ h.\ k : Z \to Y \Longrightarrow h : Z \to X \Longrightarrow g \circ_c k = f \circ_c h \Longrightarrow$
$(\exists!j.\ j : Z \to X\ {}_f\times_{cg}\ Y \wedge$
fibered-product-right-proj $X\ f\ g\ Y \circ_c j = k \wedge$
fibered-product-left-proj $X\ f\ g\ Y \circ_c j = h)$
using *fibered-product-is-pullback assms* **unfolding** *is-pullback-def* **by** *auto*

obtain j **where** *j-type*: $j \in_c X\ {}_f\times_{cg}\ Y$ **and**
j-projs: *fibered-product-right-proj* $X\ f\ g\ Y \circ_c j = y$ *fibered-product-left-proj* $X\ f\ g\ Y \circ_c j = x$
using *j-exists*[**where** Z=**1**, **where** k=y, **where** h=x] *assms f-g-eq* **by** *auto*
show $\exists\, h.\ h :$ *domain* $\langle x,y\rangle \to$ *domain* (*snd* $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y)) \wedge$
snd $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y) \circ_c h = \langle x,y\rangle$
proof (*intro exI*[**where** x=j], *safe*)
show j : *domain* $\langle x,y\rangle \to$ *domain* (*snd* $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y))$
using *assms j-type cfunc-type-def* **by** (*typecheck-cfuncs*, *auto*)

have *left-eq*: *left-cart-proj* $X\ Y \circ_c$ *fibered-product-morphism* $X\ f\ g\ Y \circ_c j = x$
using *j-projs assms j-type comp-associative2*
unfolding *fibered-product-left-proj-def* **by** (*typecheck-cfuncs*, *auto*)

have *right-eq*: *right-cart-proj* $X\ Y \circ_c$ *fibered-product-morphism* $X\ f\ g\ Y \circ_c j = y$
using *j-projs assms j-type comp-associative2*
unfolding *fibered-product-right-proj-def* **by** (*typecheck-cfuncs*, *auto*)

show *snd* $(X\ {}_f\times_{cg}\ Y$, *fibered-product-morphism* $X\ f\ g\ Y) \circ_c j = \langle x,y\rangle$

using *left-eq right-eq assms j-type* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-unique*)
qed
qed
qed

lemma *fibered-product-pair-member2*:
assumes $f : X \to Y$ $g : X \to E$ $x \in_c X$ $y \in_c X$
assumes $g \circ_c$ *fibered-product-left-proj* $X\, f\, f\, X = g \circ_c$ *fibered-product-right-proj* X $f\, f\, X$
shows $\forall\, x\, y.\ x \in_c X \longrightarrow y \in_c X \longrightarrow \langle x,y\rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\, f\, f\, X) \longrightarrow g \circ_c x = g \circ_c y$
proof(*clarify*)
fix x y
assume *x-type*[*type-rule*]: $x \in_c X$
assume *y-type*[*type-rule*]: $y \in_c X$
assume *a3*: $\langle x,y\rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\, f\, f\, X)$
then obtain h **where**
h-type: $h \in_c X_f{\times_c}_fX$ **and** *h-eq*: *fibered-product-morphism* $X\, f\, f\, X \circ_c h = \langle x,y\rangle$
by (*meson factors-through-def2 relative-member-def2*)

have *left-eq*: *fibered-product-left-proj* $X\, f\, f\, X \circ_c h = x$
unfolding *fibered-product-left-proj-def*
by (*typecheck-cfuncs*, *smt* (*z3*) *assms*(*1*) *comp-associative2 h-eq h-type left-cart-proj-cfunc-prod y-type*)

have *right-eq*: *fibered-product-right-proj* $X\, f\, f\, X \circ_c h = y$
unfolding *fibered-product-right-proj-def*
by (*typecheck-cfuncs*, *metis* (*full-types*) *a3 comp-associative2 h-eq h-type relative-member-def2 right-cart-proj-cfunc-prod x-type*)

then show $g \circ_c x = g \circ_c y$
using *assms*(*1*,*2*,*5*) *cfunc-type-def comp-associative fibered-product-left-proj-type fibered-product-right-proj-type h-type left-eq right-eq* **by** *fastforce*
qed

lemma *kernel-pair-subset*:
assumes f: $X \to Y$
shows $(X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\, f\, f\, X) \subseteq_c X \times_c X$
using *assms fibered-product-morphism-monomorphism fibered-product-morphism-type subobject-of-def2* **by** *auto*

The three lemmas below correspond to Exercise 2.1.44 in Halvorson.

lemma *kern-pair-proj-iso-TFAE1*:
assumes f: $X \to Y$ *monomorphism* f
shows (*fibered-product-left-proj* $X\, f\, f\, X$) = (*fibered-product-right-proj* $X\, f\, f\, X$)
proof (*cases* $\exists\, x.\ x \in_c X_f{\times_c}_fX$, *clarify*)
fix x
assume *x-type*: $x \in_c X_f{\times_c}_fX$
then have $(f \circ_c$ (*fibered-product-left-proj* $X\, f\, f\, X$)) $\circ_c x = (f \circ_c$ (*fibered-product-right-proj*

X f f X))$\circ_c$ *x*
  using *assms cfunc-type-def comp-associative equalizer-def fibered-product-morphism-equalizer*
   unfolding *fibered-product-right-proj-def fibered-product-left-proj-def*
   by (*typecheck-cfuncs*, *smt* (*verit*))
 then have $f \circ_c$ (*fibered-product-left-proj X f f X*) $= f \circ_c$ (*fibered-product-right-proj X f f X*)
   using *assms fibered-product-is-pullback is-pullback-def* **by** *auto*
 then show (*fibered-product-left-proj X f f X*) = (*fibered-product-right-proj X f f X*)
  using *assms cfunc-type-def fibered-product-left-proj-type fibered-product-right-proj-type monomorphism-def* **by** *auto*
next
 assume $\nexists x.\ x \in_c X\ {}_f\times_{cf}\ X$
 then show *fibered-product-left-proj X f f X* = *fibered-product-right-proj X f f X*
  using *assms fibered-product-left-proj-type fibered-product-right-proj-type one-separator* **by** *blast*
qed

lemma *kern-pair-proj-iso-TFAE2*:
 assumes f: $X \to Y$ *fibered-product-left-proj X f f X* = *fibered-product-right-proj X f f X*
  shows *monomorphism f* $\wedge$ *isomorphism* (*fibered-product-left-proj X f f X*) $\wedge$ *isomorphism* (*fibered-product-right-proj X f f X*)
 using *assms*
proof *safe*
 have *injective f*
   unfolding *injective-def*
 proof *clarify*
   fix *x y*
   assume *x-type*: $x \in_c$ *domain f* **and** *y-type*: $y \in_c$ *domain f*
   then have *x-type2*: $x \in_c X$ **and** *y-type2*: $y \in_c X$
     using *assms*(*1*) *cfunc-type-def* **by** *auto*

   have *x-y-type*: $\langle x,y \rangle : \mathbf{1} \to X \times_c X$
     using *x-type2 y-type2* **by** (*typecheck-cfuncs*)
   have *fibered-product-type*: *fibered-product-morphism X f f X* : $X\ {}_f\times_{cf}\ X \to X \times_c X$
     using *assms* **by** *typecheck-cfuncs*

   assume $f \circ_c x = f \circ_c y$
   then have *factorsthru*: $\langle x,y \rangle$ *factorsthru fibered-product-morphism X f f X*
     using *assms*(*1*) *pair-factorsthru-fibered-product-morphism x-type2 y-type2* **by** *auto*
   then obtain *xy* **where** *xy-assms*: $xy : \mathbf{1} \to X\ {}_f\times_{cf}\ X$ *fibered-product-morphism X f f X* $\circ_c$ $xy = \langle x,y \rangle$
     using *factors-through-def2 fibered-product-type x-y-type* **by** *blast*

   have *left-proj*: *fibered-product-left-proj X f f X* $\circ_c$ $xy = x$
     unfolding *fibered-product-left-proj-def* **using** *assms xy-assms*

by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative left-cart-proj-cfunc-prod x-type2 xy-assms*(*2*) *y-type2*)
have *right-proj*: *fibered-product-right-proj X f f X* $\circ_c$ *xy* = *y*
unfolding *fibered-product-right-proj-def* **using** *assms xy-assms*
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative right-cart-proj-cfunc-prod x-type2 xy-assms*(*2*) *y-type2*)

show *x* = *y*
using *assms*(*2*) *left-proj right-proj* **by** *auto*
qed
then show *monomorphism f*
using *injective-imp-monomorphism* **by** *blast*
next
have *diagonal X factorsthru fibered-product-morphism X f f X*
using *assms*(*1*) *diagonal-def id-type pair-factorsthru-fibered-product-morphism* **by** *fastforce*
then obtain *xx* **where** *xx-assms*: *xx* : *X* → *X* ${}_f\times_{cf}$ *X diagonal X* = *fibered-product-morphism X f f X* $\circ_c$ *xx*
using *assms*(*1*) *cfunc-type-def diagonal-type factors-through-def fibered-product-morphism-type* **by** *fastforce*
have *eq1*: *fibered-product-right-proj X f f X* $\circ_c$ *xx* = *id X*
by (*smt assms*(*1*) *comp-associative2 diagonal-def fibered-product-morphism-type fibered-product-right-proj-def id-type right-cart-proj-cfunc-prod right-cart-proj-type xx-assms*)

have *eq2*: *xx* $\circ_c$ *fibered-product-right-proj X f f X* = *id* (*X* ${}_f\times_{cf}$ *X*)
proof (*rule one-separator*[**where** *X*=*X* ${}_f\times_{cf}$ *X*, **where** *Y*=*X* ${}_f\times_{cf}$ *X*])
show *xx* $\circ_c$ *fibered-product-right-proj X f f X* : *X* ${}_f\times_{cf}$ *X* → *X* ${}_f\times_{cf}$ *X*
using *assms*(*1*) *comp-type fibered-product-right-proj-type xx-assms* **by** *blast*
show id_c (*X* ${}_f\times_{cf}$ *X*) : *X* ${}_f\times_{cf}$ *X* → *X* ${}_f\times_{cf}$ *X*
by (*simp add*: *id-type*)
next
fix *x*
assume *x-type*: *x* $\in_c$ *X* ${}_f\times_{cf}$ *X*
then obtain *a* **where** *a-assms*: ⟨*a*,*a*⟩ = *fibered-product-morphism X f f X* $\circ_c$ *x* *a* $\in_c$ *X*
by (*smt assms cfunc-prod-comp cfunc-prod-unique comp-type fibered-product-left-proj-def fibered-product-morphism-type fibered-product-right-proj-def fibered-product-right-proj-type*)

have (*xx* $\circ_c$ *fibered-product-right-proj X f f X*) $\circ_c$ *x* = *xx* $\circ_c$ *right-cart-proj X X* $\circ_c$ ⟨*a*,*a*⟩
using *xx-assms x-type a-assms assms comp-associative2*
unfolding *fibered-product-right-proj-def*
by (*typecheck-cfuncs*, *auto*)
also have ... = *xx* $\circ_c$ *a*
using *a-assms*(*2*) *right-cart-proj-cfunc-prod* **by** *auto*
also have ... = *x*
proof −
have *f2*: ∀ *c*. *c* : **1** → *X* ⟶ *fibered-product-morphism X f f X* $\circ_c$ *xx* $\circ_c$ *c* =

diagonal X $\circ_c$ *c*
  proof *safe*
    fix *c*
    assume $c \in_c X$
    then show *fibered-product-morphism X f f X* $\circ_c$ *xx* $\circ_c$ *c* = *diagonal X* $\circ_c$ *c*
      using *assms xx-assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 xx-assms*(*2*))
  qed
  have *f4*: *xx* : $X \to$ *codomain xx*
    using *cfunc-type-def xx-assms* **by** *presburger*
  have *f5*: *diagonal X* $\circ_c$ *a* = $\langle a,a \rangle$
    using *a-assms diag-on-elements* **by** *blast*
  have *f6*: *codomain* (*xx* $\circ_c$ *a*) = *codomain xx*
    using *f4* **by** (*meson a-assms cfunc-type-def comp-type*)
  then have *f9*: *x* : *domain x* $\to$ *codomain xx*
    using *cfunc-type-def x-type xx-assms* **by** *auto*
  have *f10*: $\forall$ *c ca. domain* (*ca* $\circ_c$ *a*) = **1** $\vee$ $\neg$ *ca* : $X \to c$
    by (*meson a-assms cfunc-type-def comp-type*)
  then have *domain* $\langle a,a \rangle$ = **1**
    using *diagonal-type f5* **by** *force*
  then have *f11*: *domain x* = **1**
    using *cfunc-type-def x-type* **by** *blast*
  have *xx* $\circ_c$ *a* $\in_c$ *codomain xx*
    using *a-assms comp-type f4* **by** *auto*
  then show *?thesis*
  using *f11 f9 f5 f2 a-assms assms*(*1*) *cfunc-type-def fibered-product-morphism-monomorphism*

      fibered-product-morphism-type monomorphism-def x-type
    by *auto*
  qed
  also have ... = id_c ($X\ {}_f\times_{cf}\ X$) $\circ_c$ *x*
    by (*metis cfunc-type-def id-left-unit x-type*)
  finally show (*xx* $\circ_c$ *fibered-product-right-proj X f f X*) $\circ_c$ *x* = id_c ($X\ {}_f\times_{cf}\ X$) $\circ_c$ *x*.
qed

show *isomorphism* (*fibered-product-left-proj X f f X*)
  unfolding *isomorphism-def*
  by (*metis assms cfunc-type-def eq1 eq2 fibered-product-right-proj-type xx-assms*(*1*))

then show *isomorphism* (*fibered-product-right-proj X f f X*)
  unfolding *isomorphism-def*
  using *assms*(*2*) *isomorphism-def* **by** *auto*
qed

lemma *kern-pair-proj-iso-TFAE3*:
  assumes *f*: $X \to Y$
  assumes *isomorphism* (*fibered-product-left-proj X f f X*) *isomorphism* (*fibered-product-right-proj X f f X*)

shows *fibered-product-left-proj X f f X* = *fibered-product-right-proj X f f X*
proof −
obtain *q0* **where**
q0-assms: *q0* : $X \to X\ {}_f{\times}_{cf}\ X$
fibered-product-left-proj X f f X $\circ_c$ *q0* = *id X*
q0 $\circ_c$ *fibered-product-left-proj X f f X* = *id* ($X\ {}_f{\times}_{cf}\ X$)
using *assms*(*1*,*2*) *cfunc-type-def isomorphism-def* **by** (*typecheck-cfuncs*, *force*)

obtain *q1* **where**
q1-assms: *q1* : $X \to X\ {}_f{\times}_{cf}\ X$
fibered-product-right-proj X f f X $\circ_c$ *q1* = *id X*
q1 $\circ_c$ *fibered-product-right-proj X f f X* = *id* ($X\ {}_f{\times}_{cf}\ X$)
using *assms*(*1*,*3*) *cfunc-type-def isomorphism-def* **by** (*typecheck-cfuncs*, *force*)

have $\bigwedge x.\ x \in_c$ *domain f* $\Longrightarrow$ *q0* $\circ_c$ *x* = *q1* $\circ_c$ *x*
proof −
fix *x*
have *fxfx*: *f* $\circ_c$ *x* = *f* $\circ_c$ *x*
by *simp*
assume *x-type*: $x \in_c$ *domain f*
have *factorsthru*: ⟨*x*,*x*⟩ *factorsthru fibered-product-morphism X f f X*
using *assms*(*1*) *cfunc-type-def fxfx pair-factorsthru-fibered-product-morphism x-type* **by** *auto*
then obtain *xx* **where** *xx-assms*: *xx* : $\mathbf{1} \to X\ {}_f{\times}_{cf}\ X$ ⟨*x*,*x*⟩ = *fibered-product-morphism X f f X* $\circ_c$ *xx*
by (*smt assms*(*1*) *cfunc-type-def diag-on-elements diagonal-type domain-comp factors-through-def factorsthru fibered-product-morphism-type x-type*)

have *projection-prop*: *q0* $\circ_c$ ((*fibered-product-left-proj X f f X*) $\circ_c$ *xx*) =
q1 $\circ_c$ ((*fibered-product-right-proj X f f X*) $\circ_c$ *xx*)
using *q0-assms q1-assms xx-assms assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have *fun-fact*: *x* = ((*fibered-product-left-proj X f f X*) $\circ_c$ *q1*) $\circ_c$ (((*fibered-product-left-proj X f f X*) $\circ_c$ *xx*))
by (*smt assms*(*1*) *cfunc-type-def comp-associative2 fibered-product-left-proj-def fibered-product-left-proj-type fibered-product-morphism-type fibered-product-right-proj-def fibered-product-right-proj-type id-left-unit2 left-cart-proj-cfunc-prod left-cart-proj-type q1-assms right-cart-proj-cfunc-prod right-cart-proj-type x-type xx-assms*)
then have *q1* $\circ_c$ ((*fibered-product-left-proj X f f X*) $\circ_c$ *xx*) =
q0 $\circ_c$ ((*fibered-product-left-proj X f f X*) $\circ_c$ *xx*)
using *q0-assms q1-assms xx-assms assms*
by (*typecheck-cfuncs*, *smt cfunc-type-def comp-associative2 fibered-product-left-proj-def fibered-product-morphism-type fibered-product-right-proj-def left-cart-proj-cfunc-prod left-cart-proj-type projection-prop right-cart-proj-cfunc-prod right-cart-proj-type x-type xx-assms*(*2*))
then show *q0* $\circ_c$ *x* = *q1* $\circ_c$ *x*
by (*smt assms*(*1*) *cfunc-type-def codomain-comp comp-associative fibered-product-left-proj-type fun-fact id-left-unit2 q0-assms q1-assms xx-assms*)
qed

then have $q0 = q1$
by (*metis assms*(*1*) *cfunc-type-def one-separator-contrapos q0-assms*(*1*) *q1-assms*(*1*))
then show *fibered-product-left-proj X f f X* = *fibered-product-right-proj X f f X*
by (*smt assms*(*1*) *comp-associative2 fibered-product-left-proj-type fibered-product-right-proj-type id-left-unit2 id-right-unit2 q0-assms q1-assms*)
qed

lemma *terminal-fib-prod-iso*:
assumes *terminal-object*(T)
assumes *f-type*: $f : Y \to T$
assumes *g-type*: $g : X \to T$
shows $(X\ {}_{g}\times_{cf}\ Y) \cong X \times_c Y$
proof −
have (*is-pullback* $(X\ {}_{g}\times_{cf}\ Y)$ Y X T (*fibered-product-right-proj* X g f Y) f (*fibered-product-left-proj* X g f Y) g)
using *assms pullback-iff-product fibered-product-is-pullback* **by** (*typecheck-cfuncs*, *blast*)
then have (*is-cart-prod* $(X\ {}_{g}\times_{cf}\ Y)$ (*fibered-product-left-proj* X g f Y) (*fibered-product-right-proj* X g f Y) X Y)
using *assms* **by** (*meson one-terminal-object pullback-iff-product terminal-func-type*)
then show *?thesis*
using *assms* **by** (*metis canonical-cart-prod-is-cart-prod cart-prods-isomorphic fst-conv is-isomorphic-def snd-conv*)
qed

end

5 Truth Values and Characteristic Functions

theory *Truth*
imports *Equalizer*
begin

The axiomatization below corresponds to Axiom 5 (Truth-Value Object) in Halvorson.

axiomatization
true-func :: *cfunc* (t) **and**
false-func :: *cfunc* (f) **and**
truth-value-set :: *cset* (Ω)
where
true-func-type[*type-rule*]: $\mathrm{t} \in_c \Omega$ **and**
false-func-type[*type-rule*]: $\mathrm{f} \in_c \Omega$ **and**
true-false-distinct: $\mathrm{t} \neq \mathrm{f}$ **and**
true-false-only-truth-values: $x \in_c \Omega \Longrightarrow x = \mathrm{f} \vee x = \mathrm{t}$ **and**
characteristic-function-exists:
$m : B \to X \Longrightarrow$ *monomorphism* $m \Longrightarrow \exists!\ \chi.$ *is-pullback* B $\mathbf{1}$ X Ω (β_B) t m χ

definition *characteristic-func* :: *cfunc* ⇒ *cfunc* **where**
characteristic-func m =

(*THE* χ. *monomorphism m* $\longrightarrow$ *is-pullback* (*domain m*) **1** (*codomain m*) Ω ($\beta_{domain\ m}$) t *m* χ)

lemma *characteristic-func-is-pullback*:
 assumes $m : B \to X$ *monomorphism m*
 shows *is-pullback B* **1** *X* Ω (β_B) t *m* (*characteristic-func m*)
proof −
 obtain χ **where** *chi-is-pullback*: *is-pullback B* **1** *X* Ω (β_B) t *m* χ
 using *assms characteristic-function-exists* **by** *blast*

 have *monomorphism m* $\longrightarrow$ *is-pullback* (*domain m*) **1** (*codomain m*) Ω ($\beta_{domain\ m}$) t *m* (*characteristic-func m*)
 unfolding *characteristic-func-def*
 proof (*rule theI'*, *rule ex1I*[**where** *a*= χ], *clarify*)
 show *is-pullback* (*domain m*) **1** (*codomain m*) Ω ($\beta_{domain\ m}$) t *m* χ
 using *assms*(*1*) *cfunc-type-def chi-is-pullback* **by** *auto*
 show $\bigwedge x$. *monomorphism m* $\longrightarrow$ *is-pullback* (*domain m*) **1** (*codomain m*) Ω ($\beta_{domain\ m}$) t *m x* $\Longrightarrow x = \chi$
 using *assms cfunc-type-def characteristic-function-exists chi-is-pullback* **by** *fastforce*
 qed
 then show *is-pullback B* **1** *X* Ω (β_B) t *m* (*characteristic-func m*)
 using *assms cfunc-type-def* **by** *auto*
qed

lemma *characteristic-func-type*[*type-rule*]:
 assumes $m : B \to X$ *monomorphism m*
 shows *characteristic-func m* : $X \to \Omega$
proof −
 have *is-pullback B* **1** *X* Ω (β_B) t *m* (*characteristic-func m*)
 using *assms* **by** (*rule characteristic-func-is-pullback*)
 then show *characteristic-func m* : $X \to \Omega$
 unfolding *is-pullback-def* **by** *auto*
qed

lemma *characteristic-func-eq*:
 assumes $m : B \to X$ *monomorphism m*
 shows *characteristic-func m* $\circ_c$ *m* = t $\circ_c$ β_B
 using *assms characteristic-func-is-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *monomorphism-equalizes-char-func*:
 assumes *m-type*[*type-rule*]: $m : B \to X$ **and** *m-mono*[*type-rule*]: *monomorphism m*
 shows *equalizer B m* (*characteristic-func m*) (t $\circ_c$ β_X)
 unfolding *equalizer-def*
proof (*rule exI*[**where** *x*=*X*], *rule exI*[**where** *x*=Ω], *safe*)
 show *characteristic-func m* : $X \to \Omega$
 by *typecheck-cfuncs*
 show t $\circ_c$ β_X : $X \to \Omega$

by *typecheck-cfuncs*
show $m : B \to X$
by *typecheck-cfuncs*
have *comm*: $\mathrm{t} \circ_c \beta_B = \textit{characteristic-func}\ m \circ_c m$
using *characteristic-func-eq m-mono m-type* **by** *auto*
then have $\beta_B = \beta_X \circ_c m$
using *m-type terminal-func-comp* **by** *auto*
then show $\textit{characteristic-func}\ m \circ_c m = (\mathrm{t} \circ_c \beta_X) \circ_c m$
using *comm comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
next
show $\bigwedge h\ F.\ h : F \to X \Longrightarrow \textit{characteristic-func}\ m \circ_c h = (\mathrm{t} \circ_c \beta_X) \circ_c h \Longrightarrow \exists k.\ k : F \to B \wedge m \circ_c k = h$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *cfunc-type-def characteristic-func-is-pullback comp-associative comp-type is-pullback-def m-mono*)
next
show $\bigwedge F\ k\ y.\ \textit{characteristic-func}\ m \circ_c m \circ_c k = (\mathrm{t} \circ_c \beta_X) \circ_c m \circ_c k \Longrightarrow k : F \to B \Longrightarrow y : F \to B \Longrightarrow m \circ_c y = m \circ_c k \Longrightarrow k = y$
by (*typecheck-cfuncs*, *smt m-mono monomorphism-def2*)
qed

lemma *characteristic-func-true-relative-member*:
assumes $m : B \to X$ *monomorphism m* $x \in_c X$
assumes *characteristic-func-true*: $\textit{characteristic-func}\ m \circ_c x = \mathrm{t}$
shows $x \in_X (B,m)$
unfolding *relative-member-def2 factors-through-def*
proof (*insert assms*, *clarify*)
have *is-pullback* $B\ \mathbf{1}\ X\ \Omega\ (\beta_B)\ \mathrm{t}\ m$ (*characteristic-func m*)
by (*simp add*: *assms characteristic-func-is-pullback*)
then have $\exists j.\ j : \mathbf{1} \to B \wedge \beta_B \circ_c j = \textit{id}\ \mathbf{1} \wedge m \circ_c j = x$
unfolding *is-pullback-def* **using** *assms* **by** (*metis id-right-unit2 id-type true-func-type*)
then show $\exists j.\ j : \textit{domain}\ x \to \textit{domain}\ m \wedge m \circ_c j = x$
using *assms*(*1*,*3*) *cfunc-type-def* **by** *auto*
qed

lemma *characteristic-func-false-not-relative-member*:
assumes $m : B \to X$ *monomorphism m* $x \in_c X$
assumes *characteristic-func-true*: $\textit{characteristic-func}\ m \circ_c x = \mathrm{f}$
shows $\neg\ (x \in_X (B,m))$
unfolding *relative-member-def2 factors-through-def*
proof (*insert assms*, *clarify*)
fix *h*
assume *x-def*: $x = m \circ_c h$
assume $h : \textit{domain}\ (m \circ_c h) \to \textit{domain}\ m$
then have *h-type*: $h \in_c B$
using *assms*(*1*,*3*) *cfunc-type-def x-def* **by** *auto*

have *is-pullback* $B\ \mathbf{1}\ X\ \Omega\ (\beta_B)\ \mathrm{t}\ m$ (*characteristic-func m*)
by (*simp add*: *assms characteristic-func-is-pullback*)
then have *char-m-true*: $\textit{characteristic-func}\ m \circ_c m = \mathrm{t} \circ_c \beta_B$

unfolding *is-pullback-def* **by** *auto*

then have *characteristic-func* $m \circ_c m \circ_c h = \mathrm{f}$
using *x-def characteristic-func-true* **by** *auto*
then have (*characteristic-func* $m \circ_c m) \circ_c h = \mathrm{f}$
using *assms h-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have $(\mathrm{t} \circ_c \beta_B) \circ_c h = \mathrm{f}$
using *char-m-true* **by** *auto*
then have $\mathrm{t} = \mathrm{f}$
by (*metis cfunc-type-def comp-associative h-type id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type true-func-type*)
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *rel-mem-char-func-true*:
assumes $m : B \to X$ *monomorphism* m $x \in_c X$
assumes $x \in_X (B,m)$
shows *characteristic-func* $m \circ_c x = \mathrm{t}$
by (*meson assms*(*4*) *characteristic-func-false-not-relative-member characteristic-func-type comp-type relative-member-def2 true-false-only-truth-values*)

lemma *not-rel-mem-char-func-false*:
assumes $m : B \to X$ *monomorphism* m $x \in_c X$
assumes $\neg (x \in_X (B,m))$
shows *characteristic-func* $m \circ_c x = \mathrm{f}$
by (*meson assms characteristic-func-true-relative-member characteristic-func-type comp-type true-false-only-truth-values*)

The lemma below corresponds to Proposition 2.2.2 in Halvorson.

lemma *card* $\{x.\ x \in_c \Omega \times_c \Omega\} = 4$
proof −
have $\{x.\ x \in_c \Omega \times_c \Omega\} = \{\langle \mathrm{t},\mathrm{t}\rangle, \langle \mathrm{t},\mathrm{f}\rangle, \langle \mathrm{f},\mathrm{t}\rangle, \langle \mathrm{f},\mathrm{f}\rangle\}$
by (*auto simp add*: *cfunc-prod-type true-func-type false-func-type*,
smt cfunc-prod-unique comp-type left-cart-proj-type right-cart-proj-type true-false-only-truth-values)
then show *card* $\{x.\ x \in_c \Omega \times_c \Omega\} = 4$
using *element-pair-eq false-func-type true-false-distinct true-func-type* **by** *auto*
qed

5.1 Equality Predicate

definition *eq-pred* :: *cset* ⇒ *cfunc* **where**
eq-pred $X = (THE\ \chi.$ *is-pullback* X $\mathbf{1}$ $(X \times_c X)$ Ω (β_X) t (*diagonal* X) $\chi)$

lemma *eq-pred-pullback*: *is-pullback* X $\mathbf{1}$ $(X \times_c X)$ Ω (β_X) t (*diagonal* X) (*eq-pred* X)
unfolding *eq-pred-def*
by (*rule the1I2*, *simp-all add*: *characteristic-function-exists diag-mono diagonal-type*)

lemma *eq-pred-type*[*type-rule*]:
  eq-pred $X : X \times_c X \rightarrow \Omega$
  using *eq-pred-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *eq-pred-square*: *eq-pred* $X \circ_c$ *diagonal* $X = \mathrm{t} \circ_c \beta_X$
  using *eq-pred-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *eq-pred-iff-eq*:
  assumes $x : \mathbf{1} \rightarrow X$ $y : \mathbf{1} \rightarrow X$
  shows $(x = y) = ($*eq-pred* $X \circ_c \langle x, y\rangle = \mathrm{t})$
proof *safe*
  assume *x-eq-y*: $x = y$

  have (*eq-pred* $X \circ_c \langle id_c\ X, id_c\ X\rangle) \circ_c y = (\mathrm{t} \circ_c \beta_X) \circ_c y$
    using *eq-pred-square* **unfolding** *diagonal-def* **by** *auto*
  then have *eq-pred* $X \circ_c \langle y, y\rangle = (\mathrm{t} \circ_c \beta_X) \circ_c y$
    using *assms diagonal-type id-type*
   by (*typecheck-cfuncs*, *smt cfunc-prod-comp comp-associative2 diagonal-def id-left-unit2*)
  then show *eq-pred* $X \circ_c \langle y, y\rangle = \mathrm{t}$
    using *assms id-type*
   by (*typecheck-cfuncs*, *smt comp-associative2 terminal-func-comp terminal-func-type terminal-func-unique id-right-unit2*)
next
  assume *eq-pred* $X \circ_c \langle x,y\rangle = \mathrm{t}$
  then have *eq-pred* $X \circ_c \langle x,y\rangle = \mathrm{t} \circ_c$ *id* $\mathbf{1}$
    using *id-right-unit2 true-func-type* **by** *auto*
  then obtain j **where** *j-type*: $j : \mathbf{1} \rightarrow X$ **and** *diagonal* $X \circ_c j = \langle x,y\rangle$
   using *eq-pred-pullback assms* **unfolding** *is-pullback-def* **by** (*metis cfunc-prod-type id-type*)
  then have $\langle j,j\rangle = \langle x,y\rangle$
    using *diag-on-elements* **by** *auto*
  then show $x = y$
    using *assms element-pair-eq j-type* **by** *auto*
qed

lemma *eq-pred-iff-eq-conv*:
  assumes $x : \mathbf{1} \rightarrow X$ $y : \mathbf{1} \rightarrow X$
  shows $(x \neq y) = ($*eq-pred* $X \circ_c \langle x, y\rangle = \mathrm{f})$
proof(*safe*)
  assume $x \neq y$
  then show *eq-pred* $X \circ_c \langle x,y\rangle = \mathrm{f}$
     using *assms eq-pred-iff-eq true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
next
  show *eq-pred* $X \circ_c \langle y,y\rangle = \mathrm{f} \Longrightarrow x = y \Longrightarrow$ *False*
    by (*metis assms*(*1*) *eq-pred-iff-eq true-false-distinct*)
qed

```
lemma eq-pred-iff-eq-conv2:
  assumes x : 1 → X y : 1 → X
  shows (x ≠ y) = (eq-pred X ∘c ⟨x, y⟩ ≠ t)
  using assms eq-pred-iff-eq by presburger

lemma eq-pred-of-monomorphism:
  assumes m-type[type-rule]: m : X → Y and m-mono: monomorphism m
  shows eq-pred Y ∘c (m ×f m) = eq-pred X
proof (rule one-separator[where X=X ×c X, where Y=Ω])
  show eq-pred Y ∘c m ×f m : X ×c X → Ω
    by typecheck-cfuncs
  show eq-pred X : X ×c X → Ω
    by typecheck-cfuncs
next
  fix x
  assume x ∈c X ×c X
  then obtain x1 x2 where x-def: x = ⟨x1, x2⟩ and x1-type[type-rule]: x1 ∈c X
and x2-type[type-rule]: x2 ∈c X
    using cart-prod-decomp by blast
  show (eq-pred Y ∘c m ×f m) ∘c x = eq-pred X ∘c x
    unfolding x-def
  proof (cases (eq-pred Y ∘c m ×f m) ∘c ⟨x1,x2⟩ = t)
    assume LHS: (eq-pred Y ∘c m ×f m) ∘c ⟨x1,x2⟩ = t
    then have eq-pred Y ∘c (m ×f m) ∘c ⟨x1,x2⟩ = t
      by (typecheck-cfuncs, simp add: comp-associative2)
    then have eq-pred Y ∘c ⟨m ∘c x1, m ∘c x2⟩ = t
      by (typecheck-cfuncs, auto simp add: cfunc-cross-prod-comp-cfunc-prod)
    then have m ∘c x1 = m ∘c x2
      by (typecheck-cfuncs-prems, simp add: eq-pred-iff-eq)
    then have x1 = x2
      using m-mono m-type monomorphism-def3 x1-type x2-type by blast
    then have RHS: eq-pred X ∘c ⟨x1,x2⟩ = t
      by (typecheck-cfuncs, insert eq-pred-iff-eq, blast)
    show (eq-pred Y ∘c m ×f m) ∘c ⟨x1,x2⟩ = eq-pred X ∘c ⟨x1,x2⟩
      using LHS RHS by auto
  next
    assume (eq-pred Y ∘c m ×f m) ∘c ⟨x1,x2⟩ ≠ t
    then have LHS: (eq-pred Y ∘c m ×f m) ∘c ⟨x1,x2⟩ = f
      by (typecheck-cfuncs, meson true-false-only-truth-values)
    then have eq-pred Y ∘c (m ×f m) ∘c ⟨x1,x2⟩ = f
      by (typecheck-cfuncs, simp add: comp-associative2)
    then have eq-pred Y ∘c ⟨m ∘c x1, m ∘c x2⟩ = f
      by (typecheck-cfuncs, auto simp add: cfunc-cross-prod-comp-cfunc-prod)
    then have m ∘c x1 ≠ m ∘c x2
      using eq-pred-iff-eq-conv by (typecheck-cfuncs-prems, blast)
    then have x1 ≠ x2
      by auto
    then have RHS: eq-pred X ∘c ⟨x1,x2⟩ = f
      using eq-pred-iff-eq-conv by (typecheck-cfuncs, blast)
```

show (*eq-pred* $Y \circ_c m \times_f m$) $\circ_c$ ⟨*x1*,*x2*⟩ = *eq-pred* $X \circ_c$ ⟨*x1*,*x2*⟩
using *LHS RHS* **by** *auto*
qed
qed

lemma *eq-pred-true-extract-right*:
assumes $x \in_c X$
shows *eq-pred* $X \circ_c \langle x \circ_c \beta_X, id\ X\rangle \circ_c x = \mathrm{t}$
using *assms cart-prod-extract-right eq-pred-iff-eq* **by** *fastforce*

lemma *eq-pred-false-extract-right*:
assumes $x \in_c X$ $y \in_c X$ $x \neq y$
shows *eq-pred* $X \circ_c \langle x \circ_c \beta_X, id\ X\rangle \circ_c y = \mathrm{f}$
using *assms cart-prod-extract-right eq-pred-iff-eq true-false-only-truth-values* **by** (*typecheck-cfuncs*, *fastforce*)

5.2 Properties of Monomorphisms and Epimorphisms

The lemma below corresponds to Exercise 2.2.3 in Halvorson.

lemma *regmono-is-mono*: *regular-monomorphism m* ⟹ *monomorphism m*
using *equalizer-is-monomorphism regular-monomorphism-def* **by** *blast*

The lemma below corresponds to Proposition 2.2.4 in Halvorson.

lemma *mono-is-regmono*:
shows *monomorphism m* ⟹ *regular-monomorphism m*
unfolding *regular-monomorphism-def*
by (*rule exI*[**where** *x*=*characteristic-func m*],
rule exI[**where** $x = \mathrm{t} \circ_c \beta_{codomain(m)}$],
typecheck-cfuncs, *auto simp add*: *cfunc-type-def monomorphism-equalizes-char-func*)

The lemma below corresponds to Proposition 2.2.5 in Halvorson.

lemma *epi-mon-is-iso*:
assumes *epimorphism f monomorphism f*
shows *isomorphism f*
using *assms epi-regmon-is-iso mono-is-regmono* **by** *auto*

The lemma below corresponds to Proposition 2.2.8 in Halvorson.

lemma *epi-is-surj*:
assumes $p: X \to Y$ *epimorphism p*
shows *surjective p*
unfolding *surjective-def*
proof(*rule ccontr*)
assume *a1*: $\neg (\forall y.\ y \in_c codomain\ p \longrightarrow (\exists x.\ x \in_c domain\ p \wedge p \circ_c x = y))$
have $\exists y.\ y \in_c Y \wedge \neg(\exists x.\ x \in_c X \wedge p \circ_c x = y)$
using *a1 assms*(*1*) *cfunc-type-def* **by** *auto*
then obtain *y0* **where** *y-def*: $y0 \in_c Y \wedge (\forall x.\ x \in_c X \longrightarrow p \circ_c x \neq y0)$
by *auto*
have *mono*: *monomorphism y0*
using *element-monomorphism y-def* **by** *blast*

```
  obtain g where g-def: g = eq-pred Y ∘c ⟨y0 ∘c β_Y, id Y⟩
    by simp
  have g-right-arg-type: ⟨y0 ∘c β_Y, id Y⟩ : Y → Y ×c Y
    by (meson cfunc-prod-type comp-type id-type terminal-func-type y-def)
  then have g-type[type-rule]: g: Y → Ω
    using comp-type eq-pred-type g-def by blast

  have gpx-Eqs-f: ∀x. x ∈c X ⟶ g ∘c p ∘c x = f
  proof(rule ccontr)
    assume ¬ (∀x. x ∈c X ⟶ g ∘c p ∘c x = f)
    then obtain x where x-type: x ∈c X and bwoc: g ∘c p ∘c x ≠ f
      by blast

    show False
        by (smt (verit) assms(1) bwoc cfunc-type-def comp-associative comp-type
eq-pred-false-extract-right eq-pred-type g-def g-right-arg-type x-type y-def)
  qed
  obtain h where h-def: h = f ∘c β_Y and h-type[type-rule]:h: Y → Ω
    by (typecheck-cfuncs, simp)
  have hpx-eqs-f: ∀x. x ∈c X ⟶ h ∘c p ∘c x = f
   by (smt assms(1) cfunc-type-def codomain-comp comp-associative false-func-type
h-def id-right-unit2 id-type terminal-func-comp terminal-func-type terminal-func-unique)
  have gp-eqs-hp: g ∘c p = h ∘c p
  proof(rule one-separator[where X=X,where Y=Ω])
    show g ∘c p : X → Ω
      using assms by typecheck-cfuncs
    show h ∘c p : X → Ω
      using assms by typecheck-cfuncs
    show ⋀x. x ∈c X ⟹ (g ∘c p) ∘c x = (h ∘c p) ∘c x
      using assms(1) comp-associative2 g-type gpx-Eqs-f h-type hpx-eqs-f by auto
  qed
  have g-not-h: g ≠ h
  proof –
   have f1: ∀c. β_(codomain c) ∘c c = β_(domain c)
    by (simp add: cfunc-type-def terminal-func-comp)
   have f2: domain ⟨y0 ∘c β_Y,id_c Y⟩ = Y
    using cfunc-type-def g-right-arg-type by blast
  have f3: codomain ⟨y0 ∘c β_Y,id_c Y⟩ = Y ×c Y
    using cfunc-type-def g-right-arg-type by blast
  have f4: codomain y0 = Y
    using cfunc-type-def y-def by presburger
  have ∀c. domain (eq-pred c) = c ×c c
    using cfunc-type-def eq-pred-type by auto
  then have g ∘c y0 ≠ f
    using f4 f3 f2 by (metis (no-types) eq-pred-true-extract-right comp-associative
g-def true-false-distinct y-def)
  then show ?thesis
     using f1 by (metis (no-types) cfunc-type-def comp-associative false-func-type
h-def id-right-unit2 id-type one-unique-element terminal-func-type y-def)
```

qed
  then show *False*
    using *gp-eqs-hp assms cfunc-type-def epimorphism-def g-type h-type* **by** *auto*
qed

The lemma below corresponds to Proposition 2.2.9 in Halvorson.

lemma *pullback-of-epi-is-epi1*:
assumes *f*: $Y \to Z$ *epimorphism f is-pullback A Y X Z q1 f q0 g*
shows *epimorphism q0*
proof −
  have *surj-f*: *surjective f*
    using *assms*(*1*,*2*) *epi-is-surj* **by** *auto*
  have *surjective* (*q0*)
    unfolding *surjective-def*
  proof(*clarify*)
    fix *y*
    assume *y-type*: $y \in_c$ *codomain q0*
    then have *codomain-gy*: $g \circ_c y \in_c Z$
      using *assms*(*3*) *cfunc-type-def is-pullback-def* **by** (*typecheck-cfuncs*, *auto*)
    then have *z-exists*: $\exists\ z.\ z \in_c Y \wedge f \circ_c z = g \circ_c y$
      using *assms*(*1*) *cfunc-type-def surj-f surjective-def* **by** *auto*
    then obtain *z* **where** *z-def*: $z \in_c Y \wedge f \circ_c z = g \circ_c y$
      by *blast*
    then have $\exists!\ k.\ k: \mathbf{1} \to A \wedge q0 \circ_c k = y \wedge q1 \circ_c k = z$
      by (*smt* (*verit*, *ccfv-threshold*) *assms*(*3*) *cfunc-type-def is-pullback-def y-type*)
    then show $\exists x.\ x \in_c$ *domain q0* $\wedge\ q0 \circ_c x = y$
      using *assms*(*3*) *cfunc-type-def is-pullback-def* **by** *auto*
  qed
  then show *?thesis*
    using *surjective-is-epimorphism* **by** *blast*
qed

The lemma below corresponds to Proposition 2.2.9b in Halvorson.

lemma *pullback-of-epi-is-epi2*:
assumes *g*: $X \to Z$ *epimorphism g is-pullback A Y X Z q1 f q0 g*
shows *epimorphism q1*
proof −
  have *surj-g*: *surjective g*
    using *assms*(*1*) *assms*(*2*) *epi-is-surj* **by** *auto*
  have *surjective q1*
    unfolding *surjective-def*
  proof(*clarify*)
    fix *y*
    assume *y-type*: $y \in_c$ *codomain q1*
    then have *codomain-gy*: $f \circ_c y \in_c Z$
      using *assms*(*3*) *cfunc-type-def comp-type is-pullback-def* **by** *auto*
    then have *z-exists*: $\exists\ z.\ z \in_c X \wedge g \circ_c z = f \circ_c y$
      using *assms*(*1*) *cfunc-type-def surj-g surjective-def* **by** *auto*
    then obtain *z* **where** *z-def*: $z \in_c X \wedge g \circ_c z = f \circ_c y$

by *blast*
then have $\exists!$ *k*. *k*: **1** $\rightarrow$ *A* $\wedge$ *q0* $\circ_c$ *k* = *z* $\wedge$ *q1* $\circ_c$ *k* =*y*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*3*) *cfunc-type-def is-pullback-def y-type*)

then show $\exists$ *x*. *x* $\in_c$ *domain q1* $\wedge$ *q1* $\circ_c$ *x* = *y*
using *assms*(*3*) *cfunc-type-def is-pullback-def* **by** *auto*
qed
then show *?thesis*
using *surjective-is-epimorphism* **by** *blast*
qed

The lemma below corresponds to Proposition 2.2.9c in Halvorson.

lemma *pullback-of-mono-is-mono1*:
assumes *g*: *X* $\rightarrow$ *Z monomorphism f is-pullback A Y X Z q1 f q0 g*
shows *monomorphism q0*
unfolding *monomorphism-def2*
proof(*clarify*)
fix *u v Q a x*
assume *u-type*: *u* : *Q* $\rightarrow$ *a*
assume *v-type*: *v* : *Q* $\rightarrow$ *a*
assume *q0-type*: *q0* : *a* $\rightarrow$ *x*
assume *equals*: *q0* $\circ_c$ *u* = *q0* $\circ_c$ *v*

have *a-is-A*: *a* = *A*
using *assms*(*3*) *cfunc-type-def is-pullback-def q0-type* **by** *force*
have *x-is-X*: *x* = *X*
using *assms*(*3*) *cfunc-type-def is-pullback-def q0-type* **by** *fastforce*
have *u-type2*[*type-rule*]: *u* : *Q* $\rightarrow$ *A*
using *a-is-A u-type* **by** *blast*
have *v-type2*[*type-rule*]: *v* : *Q* $\rightarrow$ *A*
using *a-is-A v-type* **by** *blast*
have *q1-type2*[*type-rule*]: *q0* : *A* $\rightarrow$ *X*
using *a-is-A q0-type x-is-X* **by** *blast*

have *eqn1*: *g* $\circ_c$ (*q0* $\circ_c$ *u*) = *f* $\circ_c$ (*q1* $\circ_c$ *v*)
proof –
have *g* $\circ_c$ (*q0* $\circ_c$ *u*) = *g* $\circ_c$ *q0* $\circ_c$ *v*
by (*simp add*: *equals*)
also have ... = *f* $\circ_c$ (*q1* $\circ_c$ *v*)
using *assms*(*3*) *cfunc-type-def comp-associative is-pullback-def* **by** (*typecheck-cfuncs*, *force*)
finally show *?thesis***.**
qed

have *eqn2*: *q1* $\circ_c$ *u* = *q1* $\circ_c$ *v*
proof –
have *f1*: *f* $\circ_c$ *q1* $\circ_c$ *u* = *g* $\circ_c$ *q0* $\circ_c$ *u*
using *assms*(*3*) *comp-associative2 is-pullback-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *g* $\circ_c$ *q0* $\circ_c$ *v*

```
      by (simp add: equals)
    also have ... = f ∘c q1 ∘c v
      using eqn1 equals by fastforce
    then show ?thesis
     by (typecheck-cfuncs, smt (verit, ccfv-threshold) f1 assms(2,3) eqn1 is-pullback-def
monomorphism-def3)
  qed

  have uniqueness: ∃! j. (j : Q → A ∧ q1 ∘c j = q1 ∘c v ∧ q0 ∘c j = q0 ∘c u)
   by (typecheck-cfuncs, smt (verit, ccfv-threshold) assms(3) eqn1 is-pullback-def)
  then show u = v
    using eqn2 equals uniqueness by (typecheck-cfuncs, auto)
qed
```

The lemma below corresponds to Proposition 2.2.9d in Halvorson.

```
lemma pullback-of-mono-is-mono2:
assumes g: X → Z monomorphism g is-pullback A Y X Z q1 f q0 g
shows monomorphism q1
  unfolding monomorphism-def2
proof(clarify)
  fix u v Q a y
  assume u-type: u : Q → a
  assume v-type: v : Q → a
  assume q1-type: q1 : a → y
  assume equals: q1 ∘c u = q1 ∘c v

  have a-is-A: a = A
    using assms(3) cfunc-type-def is-pullback-def q1-type by force
  have y-is-Y: y = Y
    using assms(3) cfunc-type-def is-pullback-def q1-type by fastforce
  have u-type2[type-rule]: u : Q → A
    using a-is-A u-type by blast
  have v-type2[type-rule]: v : Q → A
    using a-is-A v-type by blast
  have q1-type2[type-rule]: q1 : A → Y
    using a-is-A q1-type y-is-Y by blast

  have eqn1: f ∘c (q1 ∘c u) = g ∘c (q0 ∘c v)
  proof -
    have f ∘c (q1 ∘c u) = f ∘c q1 ∘c v
      by (simp add: equals)
    also have ... = g ∘c (q0 ∘c v)
     using assms(3) cfunc-type-def comp-associative is-pullback-def by (typecheck-cfuncs,
force)
    finally show ?thesis.
  qed

  have eqn2: q0 ∘c u = q0 ∘c v
  proof -
```

have *f1*: $g \circ_c q0 \circ_c u = f \circ_c q1 \circ_c u$
using *assms*(*3*) *comp-associative2 is-pullback-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= f \circ_c q1 \circ_c v$
by (*simp add*: *equals*)
also have ... $= g \circ_c q0 \circ_c v$
using *eqn1 equals* **by** *fastforce*
then show *?thesis*
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *f1 assms*(*2*,*3*) *eqn1 is-pullback-def monomorphism-def3*)
qed
have *uniqueness*: $\exists!\ j.\ (j : Q \to A \wedge q0 \circ_c j = q0 \circ_c v \wedge q1 \circ_c j = q1 \circ_c u)$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *assms*(*3*) *eqn1 is-pullback-def*)
then show $u = v$
using *eqn2 equals uniqueness* **by** (*typecheck-cfuncs*, *auto*)
qed

5.3 Fiber Over an Element and its Connection to the Fibered Product

The definition below corresponds to Definition 2.2.6 in Halvorson.

definition *fiber* :: *cfunc* ⇒ *cfunc* ⇒ *cset* ($\text{-}^{-1}\{\text{-}\}$ [*100*,*100*]*100*) **where**
$f^{-1}\{y\} = (f^{-1}(\!|\mathbf{1}|\!)_y)$

definition *fiber-morphism* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* **where**
fiber-morphism f y = *left-cart-proj* (*domain f*) $\mathbf{1} \circ_c$ *inverse-image-mapping f* $\mathbf{1}$ *y*

lemma *fiber-morphism-type*[*type-rule*]:
assumes $f : X \to Y$ $y \in_c Y$
shows *fiber-morphism f y* : $f^{-1}\{y\} \to X$
unfolding *fiber-def fiber-morphism-def*
using *assms cfunc-type-def element-monomorphism inverse-image-subobject subobject-of-def2*
by (*typecheck-cfuncs*, *auto*)

lemma *fiber-subset*:
assumes $f : X \to Y$ $y \in_c Y$
shows ($f^{-1}\{y\}$, *fiber-morphism f y*) $\subseteq_c X$
unfolding *fiber-def fiber-morphism-def*
using *assms cfunc-type-def element-monomorphism inverse-image-subobject inverse-image-subobject-mapping-def*
by (*typecheck-cfuncs*, *auto*)

lemma *fiber-morphism-monomorphism*:
assumes $f : X \to Y$ $y \in_c Y$
shows *monomorphism* (*fiber-morphism f y*)
using *assms cfunc-type-def element-monomorphism fiber-morphism-def inverse-image-monomorphism*
by *auto*

lemma *fiber-morphism-eq*:

assumes $f : X \to Y$ $y \in_c Y$
shows $f \circ_c$ *fiber-morphism f y* $= y \circ_c \beta_{f^{-1}\{y\}}$
proof −
have $f \circ_c$ *fiber-morphism f y* $= f \circ_c$ *left-cart-proj* (*domain f*) $\mathbf{1} \circ_c$ *inverse-image-mapping f* $\mathbf{1}$ *y*
unfolding *fiber-morphism-def* **by** *auto*
also have ... $= y \circ_c$ *right-cart-proj X* $\mathbf{1} \circ_c$ *inverse-image-mapping f* $\mathbf{1}$ *y*
using *assms cfunc-type-def element-monomorphism inverse-image-mapping-eq* **by** *auto*
also have ... $= y \circ_c \beta_{f^{-1}(\!|\mathbf{1}|\!)y}$
using *assms* **by** (*typecheck-cfuncs*, *metis element-monomorphism terminal-func-unique*)
also have ... $= y \circ_c \beta_{f^{-1}\{y\}}$
unfolding *fiber-def* **by** *auto*
finally show *?thesis*.
qed

The lemma below corresponds to Proposition 2.2.7 in Halvorson.

lemma *not-surjective-has-some-empty-preimage*:
assumes *p-type*[*type-rule*]: p: $X \to Y$ **and** *p-not-surj*: $\neg$ *surjective p*
shows $\exists$ y. $y \in_c Y \wedge$ *is-empty*($p^{-1}\{y\}$)
proof −
have *nonempty*: *nonempty*(Y)
using *assms cfunc-type-def nonempty-def surjective-def* **by** *auto*
obtain *y0* **where** *y0-type*[*type-rule*]: $y0 \in_c Y$ $\forall$ x. $x \in_c X \longrightarrow p \circ_c x \neq y0$
using *assms cfunc-type-def surjective-def* **by** *auto*

have $\neg$*nonempty*($p^{-1}\{y0\}$)
proof (*rule ccontr*, *clarify*)
assume *a1*: *nonempty*($p^{-1}\{y0\}$)
obtain *z* **where** *z-type*[*type-rule*]: $z \in_c p^{-1}\{y0\}$
using *a1 nonempty-def* **by** *blast*
have *fiber-z-type*: *fiber-morphism p y0* $\circ_c z \in_c X$
using *assms*(*1*) *comp-type fiber-morphism-type y0-type z-type* **by** *auto*
have *contradiction*: $p \circ_c$ *fiber-morphism p y0* $\circ_c z = y0$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 fiber-morphism-eq id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type*)
have $p \circ_c$ (*fiber-morphism p y0* $\circ_c z$) $\neq y0$
by (*simp add*: *fiber-z-type y0-type*)
then show *False*
using *contradiction* **by** *blast*
qed
then show *?thesis*
using *is-empty-def nonempty-def y0-type* **by** *blast*
qed

lemma *fiber-iso-fibered-prod*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
assumes *y-type*[*type-rule*]: $y : \mathbf{1} \to Y$
shows $f^{-1}\{y\} \cong X \; {}_{f}\!\times_{c y} \mathbf{1}$

using *element-monomorphism equalizers-isomorphic f-type fiber-def fibered-product-equalizer inverse-image-is-equalizer is-isomorphic-def y-type* **by** *moura*

lemma *fib-prod-left-id-iso*:
assumes $g : Y \to X$
shows $(X\ _{id(X)}{\times_c}_g\ Y) \cong Y$
proof −
have *is-pullback*: *is-pullback* $(X\ _{id(X)}{\times_c}_g\ Y)$ $Y\ X\ X$ (*fibered-product-right-proj* $X\ (id(X))\ g\ Y)$ g (*fibered-product-left-proj* $X\ (id(X))\ g\ Y)$ $(id(X))$
using *assms fibered-product-is-pullback* **by** (*typecheck-cfuncs*, *blast*)
then have *mono*: *monomorphism*(*fibered-product-right-proj* $X\ (id(X))\ g\ Y$)
using *assms* **by** (*typecheck-cfuncs*, *meson id-isomorphism iso-imp-epi-and-monic pullback-of-mono-is-mono2*)
have *epimorphism*(*fibered-product-right-proj* $X\ (id(X))\ g\ Y$)
by (*meson id-isomorphism id-type is-pullback iso-imp-epi-and-monic pullback-of-epi-is-epi2*)
then have *isomorphism*(*fibered-product-right-proj* $X\ (id(X))\ g\ Y$)
by (*simp add*: *epi-mon-is-iso mono*)
then show *?thesis*
using *assms fibered-product-right-proj-type id-type is-isomorphic-def* **by** *blast*
qed

lemma *fib-prod-right-id-iso*:
assumes $f : X \to Y$
shows $(X\ _f{\times_c}_{id(Y)}\ Y) \cong X$
proof −
have *is-pullback*: *is-pullback* $(X\ _f{\times_c}_{id(Y)}\ Y)$ $Y\ X\ Y$ (*fibered-product-right-proj* $X\ f\ (id(Y))\ Y)$ $(id(Y))$ (*fibered-product-left-proj* $X\ f\ (id(Y))\ Y)$ f
using *assms fibered-product-is-pullback* **by** (*typecheck-cfuncs*, *blast*)

then have *mono*: *monomorphism*(*fibered-product-left-proj* $X\ f\ (id(Y))\ Y$)
using *assms* **by** (*typecheck-cfuncs*, *meson id-isomorphism is-pullback iso-imp-epi-and-monic pullback-of-mono-is-mono1*)
have *epimorphism*(*fibered-product-left-proj* $X\ f\ (id(Y))\ Y$)
by (*meson id-isomorphism id-type is-pullback iso-imp-epi-and-monic pullback-of-epi-is-epi1*)
then have *isomorphism*(*fibered-product-left-proj* $X\ f\ (id(Y))\ Y$)
by (*simp add*: *epi-mon-is-iso mono*)
then show *?thesis*
using *assms fibered-product-left-proj-type id-type is-isomorphic-def* **by** *blast*
qed

The lemma below corresponds to the discussion at the top of page 42 in Halvorson.

lemma *kernel-pair-connection*:
assumes *f-type*[*type-rule*]: $f : X \to Y$ **and** *g-type*[*type-rule*]: $g : X \to E$
assumes *g-epi*: *epimorphism* g
assumes *h-g-eq-f*: $h \circ_c g = f$
assumes *g-eq*: $g \circ_c$ *fibered-product-left-proj* $X\ f\ f\ X = g \circ_c$ *fibered-product-right-proj* $X\ f\ f\ X$
assumes *h-type*[*type-rule*]: $h : E \to Y$

shows $\exists!$ b. $b : X\ {}_f{\times}_{cf}\ X \to E\ {}_h{\times}_{ch}\ E$ $\wedge$
fibered-product-left-proj E h h E $\circ_c$ b = g $\circ_c$ *fibered-product-left-proj X f f X* $\wedge$
fibered-product-right-proj E h h E $\circ_c$ b = g $\circ_c$ *fibered-product-right-proj X f f X*
$\wedge$
epimorphism b
proof −
have *gxg-fpmorph-eq*: (h $\circ_c$ *left-cart-proj E E*) $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X*
= (h $\circ_c$ *right-cart-proj E E*) $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X*
proof −
have (h $\circ_c$ *left-cart-proj E E*) $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X*
= h $\circ_c$ (*left-cart-proj E E* $\circ_c$ ($g \times_f g$)) $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = h $\circ_c$ (g $\circ_c$ *left-cart-proj X X*) $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-cart-proj-cfunc-cross-prod*)
also have ... = (h $\circ_c$ g) $\circ_c$ *left-cart-proj X X* $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = f $\circ_c$ *left-cart-proj X X* $\circ_c$ *fibered-product-morphism X f f X*
by (*simp add*: *h-g-eq-f*)
also have ... = f $\circ_c$ *right-cart-proj X X* $\circ_c$ *fibered-product-morphism X f f X*
using *f-type fibered-product-left-proj-def fibered-product-proj-eq fibered-product-right-proj-def* **by** *auto*
also have ... = (h $\circ_c$ g) $\circ_c$ *right-cart-proj X X* $\circ_c$ *fibered-product-morphism X f f X*
by (*simp add*: *h-g-eq-f*)
also have ... = h $\circ_c$ (g $\circ_c$ *right-cart-proj X X*) $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = h $\circ_c$ *right-cart-proj E E* $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-cart-proj-cfunc-cross-prod*)
also have ... = (h $\circ_c$ *right-cart-proj E E*) $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *smt comp-associative2*)
finally show *?thesis*.
qed
have *h-equalizer*: *equalizer* ($E\ {}_h{\times}_{ch}\ E$) (*fibered-product-morphism E h h E*) (h $\circ_c$ *left-cart-proj E E*) (h $\circ_c$ *right-cart-proj E E*)
using *fibered-product-morphism-equalizer h-type* **by** *auto*
then have $\forall j\ F.\ j : F \to E \times_c E$ $\wedge$ (h $\circ_c$ *left-cart-proj E E*) $\circ_c$ j = (h $\circ_c$ *right-cart-proj E E*) $\circ_c$ j $\longrightarrow$
($\exists!k$. $k : F \to E\ {}_h{\times}_{ch}\ E$ $\wedge$ *fibered-product-morphism E h h E* $\circ_c$ $k = j$)
unfolding *equalizer-def* **using** *cfunc-type-def fibered-product-morphism-type h-type* **by** (*smt* (*verit*))
then have ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X* : $X\ {}_f{\times}_{cf}\ X \to E \times_c E$ $\wedge$ (h $\circ_c$ *left-cart-proj E E*) $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X* = (h $\circ_c$ *right-cart-proj E E*) $\circ_c$ ($g \times_f g$) $\circ_c$ *fibered-product-morphism X f f X* $\longrightarrow$

($\exists$!*k*. *k* : *X* ${}_f\times_{cf}$ *X* $\rightarrow$ *E* ${}_h\times_{ch}$ *E* $\wedge$ *fibered-product-morphism E h h E* $\circ_c$ *k* = (*g* $\times_f$ *g*) $\circ_c$ *fibered-product-morphism X f f X*)
by *auto*
then obtain *b* **where** *b-type*[*type-rule*]: *b* : *X* ${}_f\times_{cf}$ *X* $\rightarrow$ *E* ${}_h\times_{ch}$ *E*
and *b-eq*: *fibered-product-morphism E h h E* $\circ_c$ *b* = (*g* $\times_f$ *g*) $\circ_c$ *fibered-product-morphism X f f X*
by (*meson cfunc-cross-prod-type comp-type f-type fibered-product-morphism-type g-type gxg-fpmorph-eq*)

have *is-pullback* (*X* ${}_f\times_{cf}$ *X*) (*X* $\times_c$ *X*) (*E* ${}_h\times_{ch}$ *E*) (*E* $\times_c$ *E*)
(*fibered-product-morphism X f f X*) (*g* $\times_f$ *g*) *b* (*fibered-product-morphism E h h E*)
unfolding *is-pullback-def*
proof (*typecheck-cfuncs*, *safe*, *metis b-eq*)
fix *Z k j*
assume *k-type*[*type-rule*]: *k* : *Z* $\rightarrow$ *X* $\times_c$ *X* **and** *h-type*[*type-rule*]: *j* : *Z* $\rightarrow$ *E* ${}_h\times_{ch}$ *E*
assume *k-h-eq*: (*g* $\times_f$ *g*) $\circ_c$ *k* = *fibered-product-morphism E h h E* $\circ_c$ *j*

have *left-k-right-k-eq*: *f* $\circ_c$ *left-cart-proj X X* $\circ_c$ *k* = *f* $\circ_c$ *right-cart-proj X X* $\circ_c$ *k*
proof –
have *f* $\circ_c$ *left-cart-proj X X* $\circ_c$ *k* = *h* $\circ_c$ *g* $\circ_c$ *left-cart-proj X X* $\circ_c$ *k*
by (*smt* (*z3*) *assms*(*6*) *comp-associative2 comp-type g-type h-g-eq-f k-type left-cart-proj-type*)
also have ... = *h* $\circ_c$ *left-cart-proj E E* $\circ_c$ (*g* $\times_f$ *g*) $\circ_c$ *k*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-cart-proj-cfunc-cross-prod*)
also have ... = *h* $\circ_c$ *left-cart-proj E E* $\circ_c$ *fibered-product-morphism E h h E* $\circ_c$ *j*
by (*simp add*: *k-h-eq*)
also have ... = ((*h* $\circ_c$ *left-cart-proj E E*) $\circ_c$ *fibered-product-morphism E h h E*) $\circ_c$ *j*
by (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = ((*h* $\circ_c$ *right-cart-proj E E*) $\circ_c$ *fibered-product-morphism E h h E*) $\circ_c$ *j*
using *equalizer-def h-equalizer* **by** *auto*
also have ... = *h* $\circ_c$ *right-cart-proj E E* $\circ_c$ *fibered-product-morphism E h h E* $\circ_c$ *j*
by (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = *h* $\circ_c$ *right-cart-proj E E* $\circ_c$ (*g* $\times_f$ *g*) $\circ_c$ *k*
by (*simp add*: *k-h-eq*)
also have ... = *h* $\circ_c$ *g* $\circ_c$ *right-cart-proj X X* $\circ_c$ *k*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-cart-proj-cfunc-cross-prod*)
also have ... = *f* $\circ_c$ *right-cart-proj X X* $\circ_c$ *k*
using *assms*(*6*) *comp-associative2 comp-type g-type h-g-eq-f k-type right-cart-proj-type* **by** *blast*
finally show *?thesis*.
qed

have *is-pullback* $(X\ {}_f{\times}_{cf}\ X)$ *X X Y*
(*fibered-product-right-proj X f f X*) *f* (*fibered-product-left-proj X f f X*) *f*
by (*simp add*: *f-type fibered-product-is-pullback*)
then have *right-cart-proj X X* $\circ_c$ *k* : *Z* → *X* ⟹ *left-cart-proj X X* $\circ_c$ *k* : *Z* → *X* ⟹ *f* $\circ_c$ *right-cart-proj X X* $\circ_c$ *k* = *f* $\circ_c$ *left-cart-proj X X* $\circ_c$ *k* ⟹
(∃!*j*. *j* : *Z* → $X\ {}_f{\times}_{cf}\ X$ ∧
fibered-product-right-proj X f f X $\circ_c$ *j* = *right-cart-proj X X* $\circ_c$ *k*
∧ *fibered-product-left-proj X f f X* $\circ_c$ *j* = *left-cart-proj X X* $\circ_c$ *k*)
unfolding *is-pullback-def* **by** *auto*
then obtain *z* **where** *z-type*[*type-rule*]: *z* : *Z* → $X\ {}_f{\times}_{cf}\ X$
and *k-right-eq*: *fibered-product-right-proj X f f X* $\circ_c$ *z* = *right-cart-proj X X* $\circ_c$ *k*
and *k-left-eq*: *fibered-product-left-proj X f f X* $\circ_c$ *z* = *left-cart-proj X X* $\circ_c$ *k*
and *z-unique*: ⋀*j*. *j* : *Z* → $X\ {}_f{\times}_{cf}\ X$
∧ *fibered-product-right-proj X f f X* $\circ_c$ *j* = *right-cart-proj X X* $\circ_c$ *k*
∧ *fibered-product-left-proj X f f X* $\circ_c$ *j* = *left-cart-proj X X* $\circ_c$ *k* ⟹ *z* = *j*
using *left-k-right-k-eq* **by** (*typecheck-cfuncs*, *auto*)

have *k-eq*: *fibered-product-morphism X f f X* $\circ_c$ *z* = *k*
using *k-right-eq k-left-eq*
unfolding *fibered-product-right-proj-def fibered-product-left-proj-def*
by (*typecheck-cfuncs-prems*, *smt cfunc-prod-comp cfunc-prod-unique*)

then show ∃ *l*. *l* : *Z* → $X\ {}_f{\times}_{cf}\ X$ ∧ *fibered-product-morphism X f f X* $\circ_c$ *l* = *k* ∧ *b* $\circ_c$ *l* = *j*
proof (*intro exI*[**where** *x*=*z*], *clarify*)
assume *k-def*: *k* = *fibered-product-morphism X f f X* $\circ_c$ *z*
have *fibered-product-morphism E h h E* $\circ_c$ *j* = $(g \times_f g)$ $\circ_c$ *k*
by (*simp add*: *k-h-eq*)
also have ... = $(g \times_f g)$ $\circ_c$ *fibered-product-morphism X f f X* $\circ_c$ *z*
by (*simp add*: *k-eq*)
also have ... = *fibered-product-morphism E h h E* $\circ_c$ *b* $\circ_c$ *z*
by (*typecheck-cfuncs*, *simp add*: *b-eq comp-associative2*)
then show *z* : *Z* → $X\ {}_f{\times}_{cf}\ X$ ∧ *fibered-product-morphism X f f X* $\circ_c$ *z* = *fibered-product-morphism X f f X* $\circ_c$ *z* ∧ *b* $\circ_c$ *z* = *j*
by (*typecheck-cfuncs*, *metis assms*(*6*) *fibered-product-morphism-monomorphism fibered-product-morphism-type k-def k-h-eq monomorphism-def3*)
qed

show ⋀ *j y*. *j* : *Z* → $X\ {}_f{\times}_{cf}\ X$ ⟹ *y* : *Z* → $X\ {}_f{\times}_{cf}\ X$ ⟹
fibered-product-morphism X f f X $\circ_c$ *y* = *fibered-product-morphism X f f X* $\circ_c$ *j* ⟹
j = *y*
using *fibered-product-morphism-monomorphism monomorphism-def2* **by** (*typecheck-cfuncs-prems*, *metis*)
qed
then have *b-epi*: *epimorphism b*
using *g-epi g-type cfunc-cross-prod-type cfunc-cross-prod-surj pullback-of-epi-is-epi1 h-type*

by (*meson epi-is-surj surjective-is-epimorphism*)

have *existence*: $\exists\, b.\ b : X\ {}_f{\times}_{cf}\ X \to E\ {}_h{\times}_{ch}\ E\ \wedge$
fibered-product-left-proj E h h E $\circ_c$ $b = g$ $\circ_c$ *fibered-product-left-proj X f f X* $\wedge$
fibered-product-right-proj E h h E $\circ_c$ $b = g$ $\circ_c$ *fibered-product-right-proj X f f X* $\wedge$
epimorphism b
proof (*intro exI*[**where** *x=b*], *safe*)
show $b : X\ {}_f{\times}_{cf}\ X \to E\ {}_h{\times}_{ch}\ E$
by *typecheck-cfuncs*
show *fibered-product-left-proj E h h E* $\circ_c$ $b = g$ $\circ_c$ *fibered-product-left-proj X f f X*
proof −
have *fibered-product-left-proj E h h E* $\circ_c$ *b*
= *left-cart-proj E E* $\circ_c$ *fibered-product-morphism E h h E* $\circ_c$ *b*
unfolding *fibered-product-left-proj-def* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *left-cart-proj E E* $\circ_c$ $(g \times_f g)$ $\circ_c$ *fibered-product-morphism X f f X*
by (*simp add*: *b-eq*)
also have ... = g $\circ_c$ *left-cart-proj X X* $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-cart-proj-cfunc-cross-prod*)
also have ... = g $\circ_c$ *fibered-product-left-proj X f f X*
unfolding *fibered-product-left-proj-def* **by** (*typecheck-cfuncs*)
finally show *?thesis*.
qed
show *fibered-product-right-proj E h h E* $\circ_c$ $b = g$ $\circ_c$ *fibered-product-right-proj X f f X*
proof −
have *fibered-product-right-proj E h h E* $\circ_c$ *b*
= *right-cart-proj E E* $\circ_c$ *fibered-product-morphism E h h E* $\circ_c$ *b*
unfolding *fibered-product-right-proj-def* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *right-cart-proj E E* $\circ_c$ $(g \times_f g)$ $\circ_c$ *fibered-product-morphism X f f X*
by (*simp add*: *b-eq*)
also have ... = g $\circ_c$ *right-cart-proj X X* $\circ_c$ *fibered-product-morphism X f f X*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-cart-proj-cfunc-cross-prod*)
also have ... = g $\circ_c$ *fibered-product-right-proj X f f X*
unfolding *fibered-product-right-proj-def* **by** (*typecheck-cfuncs*)
finally show *?thesis*.
qed
show *epimorphism b*
by (*simp add*: *b-epi*)
qed
show $\exists!b.\ b : X\ {}_f{\times}_{cf}\ X \to E\ {}_h{\times}_{ch}\ E\ \wedge$
fibered-product-left-proj E h h E $\circ_c$ $b = g$ $\circ_c$ *fibered-product-left-proj X f f X* $\wedge$

fibered-product-right-proj E h h E $\circ_c$ *b* = *g* $\circ_c$ *fibered-product-right-proj X f f X* ∧
epimorphism b
by (*typecheck-cfuncs*, *metis epimorphism-def2 existence g-eq iso-imp-epi-and-monic kern-pair-proj-iso-TFAE2 monomorphism-def3*)
qed

6 Set Subtraction

definition *set-subtraction* :: *cset* ⇒ *cset* × *cfunc* ⇒ *cset* (**infix** \ *60*) **where**
Y \ *X* = (*SOME E*. ∃ *m′*. *equalizer E m′* (*characteristic-func* (*snd X*)) (f $\circ_c$ β_Y))

lemma *set-subtraction-equalizer*:
assumes *m* : *X* → *Y monomorphism m*
shows ∃ *m′*. *equalizer* (*Y* \ (*X*,*m*)) *m′* (*characteristic-func m*) (f $\circ_c$ β_Y)
proof −
have ∃ *E m′*. *equalizer E m′* (*characteristic-func m*) (f $\circ_c$ β_Y)
using *assms equalizer-exists* **by** (*typecheck-cfuncs*, *auto*)
then have ∃ *m′*. *equalizer* (*Y* \ (*X*,*m*)) *m′* (*characteristic-func* (*snd* (*X*,*m*))) (f $\circ_c$ β_Y)
unfolding *set-subtraction-def* **by** (*subst someI-ex*, *auto*)
then show ∃ *m′*. *equalizer* (*Y* \ (*X*,*m*)) *m′* (*characteristic-func m*) (f $\circ_c$ β_Y)
by *auto*
qed

definition *complement-morphism* :: *cfunc* ⇒ *cfunc* (-^{c} [*1000*]) **where**
m^c = (*SOME m′*. *equalizer* (*codomain m* \ (*domain m*, *m*)) *m′* (*characteristic-func m*) (f $\circ_c$ $\beta_{codomain\ m}$))

lemma *complement-morphism-equalizer*:
assumes *m* : *X* → *Y monomorphism m*
shows *equalizer* (*Y* \ (*X*,*m*)) m^c (*characteristic-func m*) (f $\circ_c$ β_Y)
proof −
have ∃ *m′*. *equalizer* (*codomain m* \ (*domain m*, *m*)) *m′* (*characteristic-func m*) (f $\circ_c$ $\beta_{codomain\ m}$)
by (*simp add*: *assms cfunc-type-def set-subtraction-equalizer*)
then have *equalizer* (*codomain m* \ (*domain m*, *m*)) m^c (*characteristic-func m*) (f $\circ_c$ $\beta_{codomain\ m}$)
unfolding *complement-morphism-def* **by** (*subst someI-ex*, *auto*)
then show *equalizer* (*Y* \ (*X*, *m*)) m^c (*characteristic-func m*) (f $\circ_c$ β_Y)
using *assms* **unfolding** *cfunc-type-def* **by** *auto*
qed

lemma *complement-morphism-type*[*type-rule*]:
assumes *m* : *X* → *Y monomorphism m*
shows m^c : *Y* \ (*X*,*m*) → *Y*
using *assms cfunc-type-def characteristic-func-type complement-morphism-equalizer equalizer-def* **by** *auto*

lemma *complement-morphism-mono*:
  assumes $m : X \to Y$ *monomorphism m*
  shows *monomorphism* m^c
  using *assms complement-morphism-equalizer equalizer-is-monomorphism* **by** *blast*

lemma *complement-morphism-eq*:
  assumes $m : X \to Y$ *monomorphism m*
  shows *characteristic-func* $m \circ_c m^c = (\mathrm{f} \circ_c \beta_Y) \circ_c m^c$
  using *assms complement-morphism-equalizer* **unfolding** *equalizer-def* **by** *auto*

lemma *characteristic-func-true-not-complement-member*:
  assumes $m : B \to X$ *monomorphism* m $x \in_c X$
  assumes *characteristic-func-true*: *characteristic-func* $m \circ_c x = \mathrm{t}$
  shows $\neg\ x \in_X (X \setminus (B,\ m), m^c)$
proof
  assume *in-complement*: $x \in_X (X \setminus (B,\ m),\ m^c)$
  then obtain x' **where** x'*-type*: $x' \in_c X \setminus (B,m)$ **and** x'*-def*: $m^c \circ_c x' = x$
    using *assms cfunc-type-def complement-morphism-type factors-through-def relative-member-def2*
    by *auto*
  then have *characteristic-func* $m \circ_c m^c = (\mathrm{f} \circ_c \beta_X) \circ_c m^c$
    using *assms complement-morphism-equalizer equalizer-def* **by** *blast*
  then have *characteristic-func* $m \circ_c x = \mathrm{f} \circ_c \beta_X \circ_c x$
    using *assms* x'*-type complement-morphism-type*
      by (*typecheck-cfuncs*, *smt* x'*-def assms cfunc-type-def comp-associative domain-comp*)
  then have *characteristic-func* $m \circ_c x = \mathrm{f}$
   using *assms* **by** (*typecheck-cfuncs*, *metis id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type*)
  then show *False*
    using *characteristic-func-true true-false-distinct* **by** *auto*
qed

lemma *characteristic-func-false-complement-member*:
  assumes $m : B \to X$ *monomorphism* m $x \in_c X$
  assumes *characteristic-func-false*: *characteristic-func* $m \circ_c x = \mathrm{f}$
  shows $x \in_X (X \setminus (B,\ m), m^c)$
proof –
  have *x-equalizes*: *characteristic-func* $m \circ_c x = \mathrm{f} \circ_c \beta_X \circ_c x$
   by (*metis assms*(*3*) *characteristic-func-false false-func-type id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type*)
  have $\bigwedge h\ F.\ h : F \to X \wedge$ *characteristic-func* $m \circ_c h = (\mathrm{f} \circ_c \beta_X) \circ_c h \longrightarrow$
          $(\exists!k.\ k : F \to X \setminus (B,\ m) \wedge m^c \circ_c k = h)$
    using *assms complement-morphism-equalizer* **unfolding** *equalizer-def*
    by (*smt cfunc-type-def characteristic-func-type*)
  then obtain x' **where** x'*-type*: $x' \in_c X \setminus (B,\ m)$ **and** x'*-def*: $m^c \circ_c x' = x$
   by (*metis assms*(*3*) *cfunc-type-def comp-associative false-func-type terminal-func-type x-equalizes*)

then show $x \in_X (X \setminus (B,\ m),m^c)$
unfolding *relative-member-def factors-through-def*
using *assms complement-morphism-mono complement-morphism-type cfunc-type-def*
by *auto*
qed

lemma *in-complement-not-in-subset*:
assumes $m : X \to Y$ *monomorphism* m $x \in_c Y$
assumes $x \in_Y (Y \setminus (X,m),\ m^c)$
shows $\neg\ x \in_Y (X,\ m)$
using *assms characteristic-func-false-not-relative-member characteristic-func-true-not-complement-member characteristic-func-type comp-type true-false-only-truth-values* **by** *blast*

lemma *not-in-subset-in-complement*:
assumes $m : X \to Y$ *monomorphism* m $x \in_c Y$
assumes $\neg\ x \in_Y (X,\ m)$
shows $x \in_Y (Y \setminus (X,m),\ m^c)$
using *assms characteristic-func-false-complement-member characteristic-func-true-relative-member characteristic-func-type comp-type true-false-only-truth-values* **by** *blast*

lemma *complement-disjoint*:
assumes $m : X \to Y$ *monomorphism* m
assumes $x \in_c X$ $x' \in_c Y \setminus (X,m)$
shows $m \circ_c x \neq m^c \circ_c x'$
proof
assume $m \circ_c x = m^c \circ_c x'$
then have *characteristic-func* $m \circ_c m \circ_c x =$ *characteristic-func* $m \circ_c m^c \circ_c x'$
by *auto*
then have $($*characteristic-func* $m \circ_c m) \circ_c x = ($*characteristic-func* $m \circ_c m^c) \circ_c x'$
using *assms comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have $(\mathrm{t} \circ_c \beta_X) \circ_c x = ((\mathrm{f} \circ_c \beta_Y) \circ_c m^c) \circ_c x'$
using *assms characteristic-func-eq complement-morphism-eq* **by** *auto*
then have $\mathrm{t} \circ_c \beta_X \circ_c x = \mathrm{f} \circ_c \beta_Y \circ_c m^c \circ_c x'$
using *assms comp-associative2* **by** (*typecheck-cfuncs*, *smt terminal-func-comp terminal-func-type*)
then have $\mathrm{t} \circ_c id\ \mathbf{1} = \mathrm{f} \circ_c id\ \mathbf{1}$
using *assms* **by** (*smt cfunc-type-def comp-associative complement-morphism-type id-type one-unique-element terminal-func-comp terminal-func-type*)
then have $\mathrm{t} = \mathrm{f}$
using *false-func-type id-right-unit2 true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *set-subtraction-right-iso*:
assumes *m-type*[*type-rule*]: $m : A \to C$ **and** *m-mono*[*type-rule*]: *monomorphism* m

assumes *i-type*[*type-rule*]: $i : B \to A$ **and** *i-iso*: *isomorphism i*
shows $C \setminus (A,m) = C \setminus (B,\ m \circ_c i)$
proof −
have *mi-mono*[*type-rule*]: *monomorphism* $(m \circ_c i)$
using *cfunc-type-def composition-of-monic-pair-is-monic i-iso i-type iso-imp-epi-and-monic m-mono m-type* **by** *presburger*
obtain χm **where** χm*-type*[*type-rule*]: $\chi m : C \to \Omega$ **and** χm*-def*: $\chi m =$ *characteristic-func m*
using *characteristic-func-type m-mono m-type* **by** *blast*
obtain χmi **where** χmi*-type*[*type-rule*]: $\chi mi : C \to \Omega$ **and** χmi*-def*: $\chi mi =$ *characteristic-func* $(m \circ_c i)$
by (*typecheck-cfuncs*, *simp*)
have $\bigwedge c.\ c \in_c C \Longrightarrow (\chi m \circ_c c = \mathrm{t}) = (\chi mi \circ_c c = \mathrm{t})$
proof −
fix c
assume *c-type*[*type-rule*]: $c \in_c C$
have $(\chi m \circ_c c = \mathrm{t}) = (c \in_C (A,m))$
by (*typecheck-cfuncs*, *metis* χm*-def m-mono not-rel-mem-char-func-false rel-mem-char-func-true true-false-distinct*)
also have ... $= (\exists\ a.\ a \in_c A \wedge c = m \circ_c a)$
using *cfunc-type-def factors-through-def m-mono relative-member-def2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\exists\ b.\ b \in_c B \wedge c = m \circ_c i \circ_c b)$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-type-def comp-type epi-is-surj i-iso iso-imp-epi-and-monic surjective-def*)
also have ... $= (c \in_C (B, m \circ_c i))$
using *cfunc-type-def comp-associative2 composition-of-monic-pair-is-monic factors-through-def2 i-iso iso-imp-epi-and-monic m-mono relative-member-def2*
by (*typecheck-cfuncs*, *auto*)
also have ... $= (\chi mi \circ_c c = \mathrm{t})$
by (*typecheck-cfuncs*, *metis* χmi*-def mi-mono not-rel-mem-char-func-false rel-mem-char-func-true true-false-distinct*)
finally show $(\chi m \circ_c c = \mathrm{t}) = (\chi mi \circ_c c = \mathrm{t})$**.**
qed
then have $\chi m = \chi mi$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-type one-separator true-false-only-truth-values*)

then show $C \setminus (A,m) = C \setminus (B,\ m \circ_c i)$
using χm*-def* χmi*-def isomorphic-is-reflexive set-subtraction-def* **by** *auto*
qed

lemma *set-subtraction-left-iso*:
assumes *m-type*[*type-rule*]: $m : C \to A$ **and** *m-mono*[*type-rule*]: *monomorphism m*
assumes *i-type*[*type-rule*]: $i : A \to B$ **and** *i-iso*: *isomorphism i*
shows $A \setminus (C,m) \cong B \setminus (C,\ i \circ_c m)$
proof −
have *im-mono*[*type-rule*]: *monomorphism* $(i \circ_c m)$
using *cfunc-type-def composition-of-monic-pair-is-monic i-iso i-type iso-imp-epi-and-monic*

m-mono m-type **by** *presburger*
obtain χm **where** χm*-type*[*type-rule*]: $\chi m : A \rightarrow \Omega$ **and** χm*-def*: χm = *characteristic-func m*
using *characteristic-func-type m-mono m-type* **by** *blast*
obtain χim **where** χim*-type*[*type-rule*]: $\chi im : B \rightarrow \Omega$ **and** χim*-def*: χim = *characteristic-func* $(i \circ_c m)$
by (*typecheck-cfuncs*, *simp*)
have χim*-pullback*: *is-pullback* C **1** B Ω (β_C) t $(i \circ_c m)$ χim
using χim*-def characteristic-func-is-pullback comp-type i-type im-mono m-type* **by** *blast*
have *is-pullback* C **1** A Ω (β_C) t m $(\chi im \circ_c i)$
unfolding *is-pullback-def*
proof (*typecheck-cfuncs*, *safe*)
show t $\circ_c \beta_C = (\chi im \circ_c i) \circ_c m$
by (*typecheck-cfuncs*, *etcs-assocr*, *metis* χim*-def characteristic-func-eq comp-type im-mono*)
next
fix Z k h
assume *k-type*[*type-rule*]: $k : Z \rightarrow$ **1** **and** *h-type*[*type-rule*]: $h : Z \rightarrow A$
assume *eq*: t $\circ_c k = (\chi im \circ_c i) \circ_c h$
then obtain j **where** *j-type*[*type-rule*]: $j : Z \rightarrow C$ **and** *j-def*: $i \circ_c h = (i \circ_c m) \circ_c j$
using χim*-pullback* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *comp-associative2 k-type*)
then show $\exists j.\ j : Z \rightarrow C \wedge \beta_C \circ_c j = k \wedge m \circ_c j = h$
by (*intro exI*[**where** *x=j*], *typecheck-cfuncs*, *smt comp-associative2 i-iso iso-imp-epi-and-monic monomorphism-def2 terminal-func-unique*)
next
fix Z j y
assume *j-type*[*type-rule*]: $j : Z \rightarrow C$ **and** *y-type*[*type-rule*]: $y : Z \rightarrow C$
assume t $\circ_c \beta_C \circ_c j = (\chi im \circ_c i) \circ_c m \circ_c j$ $\beta_C \circ_c y = \beta_C \circ_c j$ $m \circ_c y = m \circ_c j$
then show $j = y$
using *m-mono monomorphism-def2* **by** (*typecheck-cfuncs-prems*, *blast*)
qed
then have χim*-i-eq-*χm: $\chi im \circ_c i = \chi m$
using χm*-def characteristic-func-is-pullback characteristic-function-exists m-mono m-type* **by** *blast*
then have $\chi im \circ_c (i \circ_c m^c) = \mathrm{f} \circ_c \beta_B \circ_c (i \circ_c m^c)$
by (*etcs-assocl*, *typecheck-cfuncs*, *smt* (*verit*, *best*) χm*-def comp-associative2 complement-morphism-eq m-mono terminal-func-comp*)
then obtain i' **where** i'*-type*[*type-rule*]: $i' : A \setminus (C, m) \rightarrow B \setminus (C, i \circ_c m)$ **and** i'*-def*: $i \circ_c m^c = (i \circ_c m)^c \circ_c i'$
using *complement-morphism-equalizer* **unfolding** *equalizer-def*
by (−, *typecheck-cfuncs*, *smt* χim*-def cfunc-type-def comp-associative2 im-mono*)

have $\chi m \circ_c (i^{-1} \circ_c (i \circ_c m)^c) = \mathrm{f} \circ_c \beta_A \circ_c (i^{-1} \circ_c (i \circ_c m)^c)$
proof −
have $\chi m \circ_c (i^{-1} \circ_c (i \circ_c m)^c) = \chi im \circ_c (i \circ_c i^{-1}) \circ_c (i \circ_c m)^c$

by (*typecheck-cfuncs*, *simp add*: χ*im-i-eq-*χ*m cfunc-type-def comp-associative i-iso*)
also have ... = $\chi im \circ_c (i \circ_c m)^c$
using *i-iso id-left-unit2 inv-right* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\mathrm{f} \circ_c \beta_B \circ_c (i \circ_c m)^c$
by (*typecheck-cfuncs*, *simp add*: χ*im-def comp-associative2 complement-morphism-eq im-mono*)
also have ... = $\mathrm{f} \circ_c \beta_A \circ_c (i^{-1} \circ_c (i \circ_c m)^c)$
by (*typecheck-cfuncs*, *metis i-iso terminal-func-unique*)
finally show *?thesis*.
qed
then obtain i'*-inv* **where** i'*-inv-type*[*type-rule*]: $i'\text{-}inv : B \setminus (C, i \circ_c m) \to A \setminus (C, m)$
and i'*-inv-def*: $(i \circ_c m)^c = (i \circ_c m^c) \circ_c i'\text{-}inv$
using *complement-morphism-equalizer*[**where** m=m, **where** X=C, **where** Y=A] **unfolding** *equalizer-def*
by (−, *typecheck-cfuncs*, *smt* (*z3*) χ*m-def cfunc-type-def comp-associative2 i-iso id-left-unit2 inv-right m-mono*)

have *isomorphism* i'
proof (*etcs-subst isomorphism-def3*, *intro exI*[**where** $x = i'\text{-}inv$], *safe*)
show $i'\text{-}inv : B \setminus (C, i \circ_c m) \to A \setminus (C, m)$
by *typecheck-cfuncs*
have $i \circ_c m^c = (i \circ_c m^c) \circ_c i'\text{-}inv \circ_c i'$
using i'*-inv-def* **by** (*etcs-subst* i'*-def*, *etcs-assocl*, *auto*)
then show $i'\text{-}inv \circ_c i' = id_c (A \setminus (C, m))$
by (*typecheck-cfuncs-prems*, *smt* (*verit*, *best*) *cfunc-type-def complement-morphism-mono composition-of-monic-pair-is-monic i-iso id-right-unit2 id-type iso-imp-epi-and-monic m-mono monomorphism-def3*)
next
have $(i \circ_c m)^c = (i \circ_c m)^c \circ_c i' \circ_c i'\text{-}inv$
using i'*-def* **by** (*etcs-subst* i'*-inv-def*, *etcs-assocl*, *auto*)
then show $i' \circ_c i'\text{-}inv = id_c (B \setminus (C, i \circ_c m))$
by (*typecheck-cfuncs-prems*, *metis complement-morphism-mono id-right-unit2 id-type im-mono monomorphism-def3*)
qed
then show $A \setminus (C, m) \cong B \setminus (C, i \circ_c m)$
using i'*-type is-isomorphic-def* **by** *blast*
qed

7 Graphs

definition *functional-on* :: *cset* ⇒ *cset* ⇒ *cset* × *cfunc* ⇒ *bool* **where**
$functional\text{-}on\ X\ Y\ R = (R \subseteq_c X \times_c Y \wedge$
$(\forall x.\ x \in_c X \longrightarrow (\exists!\ y.\ y \in_c Y \wedge$
$\langle x,y \rangle \in_{X \times_c Y} R)))$

The definition below corresponds to Definition 2.3.12 in Halvorson.

definition *graph* :: *cfunc* ⇒ *cset* **where**

graph f = (*SOME E*. ∃ *m*. *equalizer E m* (*f* $\circ_c$ *left-cart-proj* (*domain f*) (*codomain f*)) (*right-cart-proj* (*domain f*) (*codomain f*)))

lemma *graph-equalizer*:
∃ *m*. *equalizer* (*graph f*) *m* (*f* $\circ_c$ *left-cart-proj* (*domain f*) (*codomain f*)) (*right-cart-proj* (*domain f*) (*codomain f*))
unfolding *graph-def*
by (*typecheck-cfuncs*, *rule someI-ex*, *simp add*: *cfunc-type-def equalizer-exists*)

lemma *graph-equalizer2*:
assumes $f : X \to Y$
shows ∃ *m*. *equalizer* (*graph f*) *m* (*f* $\circ_c$ *left-cart-proj X Y*) (*right-cart-proj X Y*)
using *assms* **by** (*typecheck-cfuncs*, *metis cfunc-type-def graph-equalizer*)

definition *graph-morph* :: *cfunc* ⇒ *cfunc* **where**
graph-morph f = (*SOME m*. *equalizer* (*graph f*) *m* (*f* $\circ_c$ *left-cart-proj* (*domain f*) (*codomain f*)) (*right-cart-proj* (*domain f*) (*codomain f*)))

lemma *graph-equalizer3*:
equalizer (*graph f*) (*graph-morph f*) (*f* $\circ_c$ *left-cart-proj* (*domain f*) (*codomain f*)) (*right-cart-proj* (*domain f*) (*codomain f*))
unfolding *graph-morph-def* **by** (*rule someI-ex*, *simp add*: *graph-equalizer*)

lemma *graph-equalizer4*:
assumes $f : X \to Y$
shows *equalizer* (*graph f*) (*graph-morph f*) (*f* $\circ_c$ *left-cart-proj X Y*) (*right-cart-proj X Y*)
using *assms cfunc-type-def graph-equalizer3* **by** *auto*

lemma *graph-subobject*:
assumes $f : X \to Y$
shows (*graph f*, *graph-morph f*) $\subseteq_c$ ($X \times_c Y$)
by (*metis assms cfunc-type-def equalizer-def equalizer-is-monomorphism graph-equalizer3 right-cart-proj-type subobject-of-def2*)

lemma *graph-morph-type*[*type-rule*]:
assumes $f : X \to Y$
shows *graph-morph*(*f*) : *graph f* → $X \times_c Y$
using *graph-subobject subobject-of-def2 assms* **by** *auto*

The lemma below corresponds to Exercise 2.3.13 in Halvorson.

lemma *graphs-are-functional*:
assumes $f : X \to Y$
shows *functional-on X Y* (*graph f*, *graph-morph f*)
unfolding *functional-on-def*
proof(*safe*)
show *graph-subobj*: (*graph f*, *graph-morph f*) $\subseteq_c$ ($X \times_c Y$)
by (*simp add*: *assms graph-subobject*)
show $\bigwedge x.\ x \in_c X \Longrightarrow \exists y.\ y \in_c Y \wedge \langle x,y \rangle \in_{X \times_c Y}$ (*graph f*, *graph-morph f*)

proof –
fix *x*
assume *x-type*[*type-rule*]: $x \in_c X$
obtain *y* **where** *y-def*: $y = f \circ_c x$
by *simp*
then have *y-type*[*type-rule*]: $y \in_c Y$
using *assms comp-type x-type y-def* **by** *blast*

have $\langle x,y \rangle \in_{X \times_c Y}$ (*graph f*, *graph-morph f*)
unfolding *relative-member-def*
proof(*typecheck-cfuncs*, *safe*)
show *monomorphism* (*snd* (*graph f*, *graph-morph f*))
using *graph-subobj subobject-of-def* **by** *auto*
show *snd* (*graph f*, *graph-morph f*) : *fst* (*graph f*, *graph-morph f*) $\to X \times_c$
Y
by (*simp add*: *assms graph-morph-type*)
have $\langle x,y \rangle$ *factorsthru graph-morph f*
proof(*subst xfactorthru-equalizer-iff-fx-eq-gx*[**where** E = *graph f*, **where** m
= *graph-morph f*,
where f = ($f \circ_c$ *left-cart-proj X Y*),
where g = *right-cart-proj X Y*, **where** $X = X \times_c Y$, **where** $Y = Y$,
where $x = \langle x,y \rangle$])
show $f \circ_c$ *left-cart-proj X Y* : $X \times_c Y \to Y$
using *assms* **by** *typecheck-cfuncs*
show *right-cart-proj X Y* : $X \times_c Y \to Y$
by *typecheck-cfuncs*
show *equalizer* (*graph f*) (*graph-morph f*) ($f \circ_c$ *left-cart-proj X Y*) (*right-cart-proj*
X Y)
by (*simp add*: *assms graph-equalizer4*)
show $\langle x,y \rangle \in_c X \times_c Y$
by *typecheck-cfuncs*
show ($f \circ_c$ *left-cart-proj X Y*) $\circ_c \langle x,y \rangle$ = *right-cart-proj X Y* $\circ_c \langle x,y \rangle$
using *assms*
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 left-cart-proj-cfunc-prod*
right-cart-proj-cfunc-prod y-def)
qed
then show $\langle x,y \rangle$ *factorsthru snd* (*graph f*, *graph-morph f*)
by *simp*
qed
then show $\exists y.\ y \in_c Y \wedge \langle x,y \rangle \in_{X \times_c Y}$ (*graph f*, *graph-morph f*)
using *y-type* **by** *blast*
qed
show $\bigwedge x\ y\ ya.$
$x \in_c X \Longrightarrow$
$y \in_c Y \Longrightarrow$
$\langle x,y \rangle \in_{X \times_c Y}$ (*graph f*, *graph-morph f*) $\Longrightarrow$
$ya \in_c Y \Longrightarrow$
$\langle x,ya \rangle \in_{X \times_c Y}$ (*graph f*, *graph-morph f*)
$\Longrightarrow y = ya$

using *assms*
by (*smt* (*z3*) *comp-associative2 equalizer-def factors-through-def2 graph-equalizer4 left-cart-proj-cfunc-prod left-cart-proj-type relative-member-def2 right-cart-proj-cfunc-prod*)
qed

lemma *functional-on-isomorphism*:
assumes *functional-on* X Y (R,m)
shows *isomorphism*(*left-cart-proj* X Y $\circ_c$ m)
proof −
have *m-mono*: *monomorphism*(m)
using *assms functional-on-def subobject-of-def2* **by** *blast*
have *pi0-m-type*[*type-rule*]: *left-cart-proj* X Y $\circ_c$ $m : R \to X$
using *assms functional-on-def subobject-of-def2* **by** (*typecheck-cfuncs*, *blast*)
have *surj*: *surjective*(*left-cart-proj* X Y $\circ_c$ m)
unfolding *surjective-def*
proof(*clarify*)
fix x
assume $x \in_c$ *codomain* (*left-cart-proj* X Y $\circ_c$ m)
then have [*type-rule*]: $x \in_c X$
using *cfunc-type-def pi0-m-type* **by** *force*
then have $\exists!\ y.\ (y \in_c Y \wedge \langle x,y\rangle \in_{X \times_c Y} (R,m))$
using *assms functional-on-def* **by** *force*
then show $\exists z.\ z \in_c$ *domain* (*left-cart-proj* X Y $\circ_c$ m) $\wedge$ (*left-cart-proj* X Y $\circ_c$ m) $\circ_c$ $z = x$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *cfunc-type-def comp-associative factors-through-def2 left-cart-proj-cfunc-prod relative-member-def2*)
qed
have *inj*: *injective*(*left-cart-proj* X Y $\circ_c$ m)
proof(*unfold injective-def*, *clarify*)
fix *r1 r2*
assume *r1* $\in_c$ *domain* (*left-cart-proj* X Y $\circ_c$ m) **then have** *r1-type*[*type-rule*]: *r1* $\in_c R$
by (*metis cfunc-type-def pi0-m-type*)
assume *r2* $\in_c$ *domain* (*left-cart-proj* X Y $\circ_c$ m) **then have** *r2-type*[*type-rule*]: *r2* $\in_c R$
by (*metis cfunc-type-def pi0-m-type*)
assume (*left-cart-proj* X Y $\circ_c$ m) $\circ_c$ *r1* = (*left-cart-proj* X Y $\circ_c$ m) $\circ_c$ *r2*
then have *eq*: *left-cart-proj* X Y $\circ_c$ m $\circ_c$ *r1* = *left-cart-proj* X Y $\circ_c$ m $\circ_c$ *r2*
using *assms cfunc-type-def comp-associative functional-on-def subobject-of-def2* **by** (*typecheck-cfuncs*, *auto*)
have *mx-type*[*type-rule*]: m $\circ_c$ *r1* $\in_c X \times_c Y$
using *assms functional-on-def subobject-of-def2* **by** (*typecheck-cfuncs*, *blast*)
then obtain *x1* **and** *y1* **where** *m1r1-eqs*: m $\circ_c$ *r1* = $\langle$*x1*, *y1*$\rangle$ $\wedge$ *x1* $\in_c X$ $\wedge$ *y1* $\in_c Y$
using *cart-prod-decomp* **by** *presburger*
have *my-type*[*type-rule*]: m $\circ_c$ *r2* $\in_c X \times_c Y$
using *assms functional-on-def subobject-of-def2* **by** (*typecheck-cfuncs*, *blast*)
then obtain *x2* **and** *y2* **where** *m2r2-eqs*:m $\circ_c$ *r2* = $\langle$*x2*, *y2*$\rangle$ $\wedge$ *x2* $\in_c X$ $\wedge$ *y2* $\in_c Y$

using *cart-prod-decomp* **by** *presburger*
have *x-equal*: $x1 = x2$
using *eq left-cart-proj-cfunc-prod m1r1-eqs m2r2-eqs* **by** *force*
have *functional*: $\exists!\ y.\ (y \in_c Y \wedge \langle x1, y\rangle \in_{X \times_c Y} (R,m))$
using *assms functional-on-def m1r1-eqs* **by** *force*
then have *y-equal*: $y1 = y2$
by (*metis prod.sel factors-through-def2 m1r1-eqs m2r2-eqs mx-type my-type r1-type r2-type relative-member-def x-equal*)
then show $r1 = r2$
by (*metis functional cfunc-type-def m1r1-eqs m2r2-eqs monomorphism-def r1-type r2-type relative-member-def2 x-equal*)
qed
show *isomorphism*(*left-cart-proj X Y* $\circ_c$ *m*)
by (*metis epi-mon-is-iso inj injective-imp-monomorphism surj surjective-is-epimorphism*)
qed

The lemma below corresponds to Proposition 2.3.14 in Halvorson.

lemma *functional-relations-are-graphs*:
assumes *functional-on X Y* (*R*,*m*)
shows $\exists!\ f.\ f : X \to Y \wedge$
($\exists$ *i*. *i* : *R* $\to$ *graph*(*f*) $\wedge$ *isomorphism*(*i*) $\wedge$ *m* = *graph-morph*(*f*) $\circ_c$ *i*)
proof *safe*
have *m-type*[*type-rule*]: $m : R \to X \times_c Y$
using *assms* **unfolding** *functional-on-def subobject-of-def2* **by** *auto*
have *m-mono*[*type-rule*]: *monomorphism*(*m*)
using *assms functional-on-def subobject-of-def2* **by** *blast*
have *isomorphism*[*type-rule*]: *isomorphism*(*left-cart-proj X Y* $\circ_c$ *m*)
using *assms functional-on-isomorphism* **by** *force*

obtain *h* **where** *h-type*[*type-rule*]: $h: X \to R$ **and** *h-def*: *h* = (*left-cart-proj X Y* $\circ_c$ *m*)$^{-1}$
by (*typecheck-cfuncs*, *simp*)
obtain *f* **where** *f-def*: *f* = (*right-cart-proj X Y*) $\circ_c$ *m* $\circ_c$ *h*
by *auto*
then have *f-type*[*type-rule*]: $f : X \to Y$
by (*metis assms comp-type f-def functional-on-def h-type right-cart-proj-type subobject-of-def2*)

have *eq*: *f* $\circ_c$ *left-cart-proj X Y* $\circ_c$ *m* = *right-cart-proj X Y* $\circ_c$ *m*
unfolding *f-def h-def* **by** (*typecheck-cfuncs*, *smt comp-associative2 id-right-unit2 inv-left isomorphism*)

show $\exists$*f*. $f : X \to Y \wedge$ ($\exists$*i*. *i* : *R* $\to$ *graph f* $\wedge$ *isomorphism i* $\wedge$ *m* = *graph-morph f* $\circ_c$ *i*)
proof (*intro exI*[**where** *x*=*f*], *safe*, *typecheck-cfuncs*)
have *graph-equalizer*: *equalizer* (*graph f*) (*graph-morph f*) (*f* $\circ_c$ *left-cart-proj X Y*) (*right-cart-proj X Y*)
by (*simp add*: *f-type graph-equalizer4*)
then have $\forall$ *h F*. $h : F \to X \times_c Y \wedge$ (*f* $\circ_c$ *left-cart-proj X Y*) $\circ_c$ *h* =

right-cart-proj X Y $\circ_c$ h $\longrightarrow$
($\exists!k$. k : $F \rightarrow$ *graph* f $\wedge$ *graph-morph* f $\circ_c$ $k = h$)
unfolding *equalizer-def* **using** *cfunc-type-def* **by** (*typecheck-cfuncs*, *auto*)
then obtain i **where** *i-type*[*type-rule*]: i : $R \rightarrow$ *graph* f **and** *i-eq*: *graph-morph* f $\circ_c$ $i = m$
by (*typecheck-cfuncs*, *smt comp-associative2 eq left-cart-proj-type*)
have *surjective* i
proof (*etcs-subst surjective-def2*, *clarify*)
fix y'
assume y'*-type*[*type-rule*]: $y' \in_c$ *graph* f

define x **where** x = *left-cart-proj* X Y $\circ_c$ *graph-morph*(f) $\circ_c$ y'
then have *x-type*[*type-rule*]: $x \in_c X$
unfolding *x-def* **by** *typecheck-cfuncs*

obtain y **where** *y-type*[*type-rule*]: $y \in_c Y$ **and** *x-y-in-R*: $\langle x,y\rangle \in_{X \times_c Y}$ (R, m)
and *y-unique*: $\forall$ z. ($z \in_c Y \wedge \langle x,z\rangle \in_{X \times_c Y}$ (R, m)) $\longrightarrow z = y$
by (*metis assms functional-on-def x-type*)

obtain x' **where** x'*-type*[*type-rule*]: $x' \in_c R$ **and** x'*-eq*: $m \circ_c x' = \langle x, y\rangle$
using *x-y-in-R* **unfolding** *relative-member-def2* **by** (−, *etcs-subst-asm factors-through-def2*, *auto*)

have *graph-morph*(f) $\circ_c$ i $\circ_c$ x' = *graph-morph*(f) $\circ_c$ y'
proof (*typecheck-cfuncs*, *rule cart-prod-eqI*, *safe*)
show *left*: *left-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ i $\circ_c$ x' = *left-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ y'
proof −
have *left-cart-proj* X Y $\circ_c$ *graph-morph*(f) $\circ_c$ i $\circ_c$ x' = *left-cart-proj* X Y $\circ_c$ m $\circ_c$ x'
by (*typecheck-cfuncs*, *smt comp-associative2 i-eq*)
also have ... = x
unfolding x'*-eq* **using** *left-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *blast*)
also have ... = *left-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ y'
unfolding *x-def* **by** *auto*
finally show *?thesis*.
qed

show *right-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ i $\circ_c$ x' = *right-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ y'
proof −
have *right-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ i $\circ_c$ x' = f $\circ_c$ *left-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ i $\circ_c$ x'
by (*etcs-assocl*, *typecheck-cfuncs*, *metis graph-equalizer equalizer-eq*)
also have ... = f $\circ_c$ *left-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ y'
by (*subst left*, *simp*)
also have ... = *right-cart-proj* X Y $\circ_c$ *graph-morph* f $\circ_c$ y'

by (*etcs-assocl*, *typecheck-cfuncs*, *metis graph-equalizer equalizer-eq*)
finally show *?thesis*.
qed
qed
then have $i \circ_c x' = y'$
using *equalizer-is-monomorphism graph-equalizer monomorphism-def2* **by** (*typecheck-cfuncs-prems*, *blast*)
then show $\exists x'. x' \in_c R \wedge i \circ_c x' = y'$
by (*intro exI*[**where** $x=x'$], *simp add*: *x'-type*)
qed
then have *isomorphism i*
by (*metis comp-monic-imp-monic' epi-mon-is-iso f-type graph-morph-type i-eq i-type m-mono surjective-is-epimorphism*)
then show $\exists i.\ i : R \to graph\ f \wedge isomorphism\ i \wedge m = graph\text{-}morph\ f \circ_c i$
by (*intro exI*[**where** $x=i$], *simp add*: *i-type i-eq*)
qed
next
fix *f1 f2 i1 i2*
assume *f1-type*[*type-rule*]: $f1 : X \to Y$
assume *f2-type*[*type-rule*]: $f2 : X \to Y$
assume *i1-type*[*type-rule*]: $i1 : R \to graph\ f1$
assume *i2-type*[*type-rule*]: $i2 : R \to graph\ f2$
assume *i1-iso*: *isomorphism i1*
assume *i2-iso*: *isomorphism i2*
assume *eq1*: $m = graph\text{-}morph\ f1 \circ_c i1$
assume *eq2*: $graph\text{-}morph\ f1 \circ_c i1 = graph\text{-}morph\ f2 \circ_c i2$

have *m-type*[*type-rule*]: $m : R \to X \times_c Y$
using *assms* **unfolding** *functional-on-def subobject-of-def2* **by** *auto*
have *isomorphism*[*type-rule*]: *isomorphism*($left\text{-}cart\text{-}proj\ X\ Y \circ_c m$)
using *assms functional-on-isomorphism* **by** *force*
obtain *h* **where** *h-type*[*type-rule*]: $h: X \to R$ **and** *h-def*: $h = (left\text{-}cart\text{-}proj\ X\ Y \circ_c m)^{-1}$
by (*typecheck-cfuncs*, *simp*)
have $f1 \circ_c left\text{-}cart\text{-}proj\ X\ Y \circ_c m = f2 \circ_c left\text{-}cart\text{-}proj\ X\ Y \circ_c m$
proof –
have $f1 \circ_c left\text{-}cart\text{-}proj\ X\ Y \circ_c m = (f1 \circ_c left\text{-}cart\text{-}proj\ X\ Y) \circ_c graph\text{-}morph\ f1 \circ_c i1$
using *comp-associative2 eq1 eq2* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (right\text{-}cart\text{-}proj\ X\ Y) \circ_c graph\text{-}morph\ f1 \circ_c i1$
by (*typecheck-cfuncs*, *smt comp-associative2 equalizer-def graph-equalizer4*)
also have ... $= (right\text{-}cart\text{-}proj\ X\ Y) \circ_c graph\text{-}morph\ f2 \circ_c i2$
by (*simp add*: *eq2*)
also have ... $= (f2 \circ_c left\text{-}cart\text{-}proj\ X\ Y) \circ_c graph\text{-}morph\ f2 \circ_c i2$
by (*typecheck-cfuncs*, *smt comp-associative2 equalizer-eq graph-equalizer4*)
also have ... $= f2 \circ_c left\text{-}cart\text{-}proj\ X\ Y \circ_c m$
by (*typecheck-cfuncs*, *metis comp-associative2 eq1 eq2*)
finally show *?thesis*.
qed

then show $f1 = f2$
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative h-def h-type id-right-unit2 inverse-def2 isomorphism*)
qed

end

8 Equivalence Classes and Coequalizers

theory *Equivalence*
imports *Truth*
begin

definition *reflexive-on* :: $cset \Rightarrow cset \times cfunc \Rightarrow bool$ **where**
reflexive-on X $R = (R \subseteq_c X \times_c X \wedge$
$(\forall x.\ x \in_c X \longrightarrow (\langle x,x\rangle \in_{X \times_c X} R)))$

definition *symmetric-on* :: $cset \Rightarrow cset \times cfunc \Rightarrow bool$ **where**
symmetric-on X $R = (R \subseteq_c X \times_c X \wedge$
$(\forall x\ y.\ x \in_c X \wedge\ y \in_c X \longrightarrow$
$(\langle x,y\rangle \in_{X \times_c X} R \longrightarrow \langle y,x\rangle \in_{X \times_c X} R)))$

definition *transitive-on* :: $cset \Rightarrow cset \times cfunc \Rightarrow bool$ **where**
transitive-on X $R = (R \subseteq_c X \times_c X \wedge$
$(\forall x\ y\ z.\ x \in_c X \wedge\ y \in_c X \wedge z \in_c X \longrightarrow$
$(\langle x,y\rangle \in_{X \times_c X} R \wedge \langle y,z\rangle \in_{X \times_c X} R \longrightarrow \langle x,z\rangle \in_{X \times_c X} R)))$

definition *equiv-rel-on* :: $cset \Rightarrow cset \times cfunc \Rightarrow bool$ **where**
equiv-rel-on X $R \longleftrightarrow$ (*reflexive-on* X R $\wedge$ *symmetric-on* X R $\wedge$ *transitive-on* X R)

definition *const-on-rel* :: $cset \Rightarrow cset \times cfunc \Rightarrow cfunc \Rightarrow bool$ **where**
const-on-rel X R $f = (\forall x\ y.\ x \in_c X \longrightarrow y \in_c X \longrightarrow \langle x,\ y\rangle \in_{X \times_c X} R \longrightarrow f \circ_c x = f \circ_c y)$

lemma *reflexive-def2*:
assumes *reflexive-Y*: *reflexive-on* X $(Y,\ m)$
assumes *x-type*: $x \in_c X$
shows $\exists y.\ y \in_c Y \wedge\ m \circ_c y = \langle x,x\rangle$
using *assms* **unfolding** *reflexive-on-def relative-member-def factors-through-def2*
proof –
assume *a1*: $(Y,\ m) \subseteq_c X \times_c X \wedge (\forall x.\ x \in_c X \longrightarrow \langle x,x\rangle \in_c X \times_c X \wedge$ *monomorphism* (*snd* $(Y,\ m)$) $\wedge$ *snd* $(Y,\ m)$: *fst* $(Y,\ m) \rightarrow X \times_c X \wedge \langle x,x\rangle$ *factorsthru snd* $(Y,\ m)$)
have *xx-type*: $\langle x,x\rangle \in_c X \times_c X$
by (*typecheck-cfuncs*, *simp add*: *x-type*)
have $\langle x,x\rangle$ *factorsthru* m
using *a1 x-type* **by** *auto*
then show *?thesis*

using *a1 xx-type cfunc-type-def factors-through-def subobject-of-def2* **by** *force*
qed

lemma *symmetric-def2*:
assumes *symmetric-Y*: *symmetric-on* X (Y, m)
assumes *x-type*: $x \in_c X$
assumes *y-type*: $y \in_c X$
assumes *relation*: $\exists\, v.\ v \in_c Y \wedge\ m \circ_c v = \langle x,y\rangle$
shows $\exists\, w.\ w \in_c Y \wedge\ m \circ_c w = \langle y,x\rangle$
using *assms* **unfolding** *symmetric-on-def relative-member-def factors-through-def2*
by (*metis cfunc-prod-type factors-through-def2 fst-conv snd-conv subobject-of-def2*)

lemma *transitive-def2*:
assumes *transitive-Y*: *transitive-on* X (Y, m)
assumes *x-type*: $x \in_c X$
assumes *y-type*: $y \in_c X$
assumes *z-type*: $z \in_c X$
assumes *relation1*: $\exists\, v.\ v \in_c Y \wedge\ m \circ_c v = \langle x,y\rangle$
assumes *relation2*: $\exists\, w.\ w \in_c Y \wedge\ m \circ_c w = \langle y,z\rangle$
shows $\exists\, u.\ u \in_c Y \wedge\ m \circ_c u = \langle x,z\rangle$
using *assms* **unfolding** *transitive-on-def relative-member-def factors-through-def2*
by (*metis cfunc-prod-type factors-through-def2 fst-conv snd-conv subobject-of-def2*)

The lemma below corresponds to Exercise 2.3.3 in Halvorson.

lemma *kernel-pair-equiv-rel*:
assumes $f : X \to Y$
shows *equiv-rel-on* X ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$)
proof (*unfold equiv-rel-on-def*, *safe*)
show *reflexive-on* X ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$)
proof (*unfold reflexive-on-def*, *safe*)
show ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$) $\subseteq_c X \times_c X$
using *assms kernel-pair-subset* **by** *auto*
next
fix x
assume *x-type*: $x \in_c X$
then show $\langle x,x\rangle \in_{X \times_c X}$ ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$)
by (*smt assms comp-type diag-on-elements diagonal-type fibered-product-morphism-monomorphism fibered-product-morphism-type pair-factorsthru-fibered-product-morphism relative-member-def2*)
qed

show *symmetric-on* X ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$)
proof (*unfold symmetric-on-def*, *safe*)
show ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$) $\subseteq_c X \times_c X$
using *assms kernel-pair-subset* **by** *auto*
next
fix $x\ y$
assume *x-type*: $x \in_c X$ **and** *y-type*: $y \in_c X$
assume *xy-in*: $\langle x,y\rangle \in_{X \times_c X}$ ($X\ {}_f\!\times_{cf}\ X$, *fibered-product-morphism* $X\ f\ f\ X$)

then have $f \circ_c x = f \circ_c y$
  using *assms fibered-product-pair-member x-type y-type* **by** *blast*

then show $\langle y,x\rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\ f\ f\ X)$
  using *assms fibered-product-pair-member x-type y-type* **by** *auto*
qed

show *transitive-on* $X\ (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\ f\ f\ X)$
proof (*unfold transitive-on-def*, *safe*)
  show $(X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\ f\ f\ X) \subseteq_c X \times_c X$
    using *assms kernel-pair-subset* **by** *auto*
next
  fix $x\ y\ z$
  assume *x-type*: $x \in_c X$ **and** *y-type*: $y \in_c X$ **and** *z-type*: $z \in_c X$
  assume *xy-in*: $\langle x,y\rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\ f\ f\ X)$
  assume *yz-in*: $\langle y,z\rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\ f\ f\ X)$

  have *eqn1*: $f \circ_c x = f \circ_c y$
    using *assms fibered-product-pair-member x-type xy-in y-type* **by** *blast*

  have *eqn2*: $f \circ_c y = f \circ_c z$
    using *assms fibered-product-pair-member y-type yz-in z-type* **by** *blast*

  show $\langle x,z\rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X\ f\ f\ X)$
    using *assms eqn1 eqn2 fibered-product-pair-member x-type z-type* **by** *auto*
qed
qed

The axiomatization below corresponds to Axiom 6 (Equivalence Classes) in Halvorson.

axiomatization
quotient-set :: *cset* ⇒ (*cset* × *cfunc*) ⇒ *cset* (**infix** // *50*) **and**
equiv-class :: *cset* × *cfunc* ⇒ *cfunc* **and**
quotient-func :: *cfunc* ⇒ *cset* × *cfunc* ⇒ *cfunc*
where
equiv-class-type[*type-rule*]: *equiv-rel-on* $X\ R \Longrightarrow$ *equiv-class* $R : X \to$ *quotient-set* $X\ R$ **and**
equiv-class-eq: *equiv-rel-on* $X\ R \Longrightarrow \langle x,\ y\rangle \in_c X{\times_c}X \Longrightarrow$
$\langle x,\ y\rangle \in_{X{\times_c}X} R \longleftrightarrow$ *equiv-class* $R \circ_c x =$ *equiv-class* $R \circ_c y$ **and**
quotient-func-type[*type-rule*]:
equiv-rel-on $X\ R \Longrightarrow f : X \to Y \Longrightarrow$ (*const-on-rel* $X\ R\ f$) $\Longrightarrow$
quotient-func $f\ R$: *quotient-set* $X\ R \to Y$ **and**
quotient-func-eq: *equiv-rel-on* $X\ R \Longrightarrow f : X \to Y \Longrightarrow$ (*const-on-rel* $X\ R\ f$) $\Longrightarrow$
quotient-func $f\ R \circ_c$ *equiv-class* $R = f$ **and**
quotient-func-unique: *equiv-rel-on* $X\ R \Longrightarrow f : X \to Y \Longrightarrow$ (*const-on-rel* $X\ R\ f$) $\Longrightarrow$
h : *quotient-set* $X\ R \to Y \Longrightarrow h \circ_c$ *equiv-class* $R = f \Longrightarrow h =$ *quotient-func* f R

Note that (//) corresponds to X/R, *equiv-class* corresponds to the canonical quotient mapping q, and *quotient-func* corresponds to $\bar{f}$ in Halvorson's formulation of this axiom.

abbreviation *equiv-class'* :: *cfunc* ⇒ *cset* × *cfunc* ⇒ *cfunc* ([-]₋) **where**
$[x]_R \equiv$ *equiv-class R* $\circ_c$ *x*

8.1 Coequalizers

The definition below corresponds to a comment after Axiom 6 (Equivalence Classes) in Halvorson.

definition *coequalizer* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cfunc* ⇒ *bool* **where**
coequalizer E m f g ⟷ (∃ *X Y*. (*f* : *Y* → *X*) ∧ (*g* : *Y* → *X*) ∧ (*m* : *X* → *E*)
∧ ($m \circ_c f = m \circ_c g$)
∧ (∀ *h F*. ((*h* : *X* → *F*) ∧ ($h \circ_c f = h \circ_c g$)) ⟶ (∃! *k*. (*k* : *E* → *F*) ∧ $k \circ_c m = h$)))

lemma *coequalizer-def2*:
assumes *f* : *Y* → *X* *g* : *Y* → *X* *m* : *X* → *E*
shows *coequalizer E m f g* ⟷
($m \circ_c f = m \circ_c g$)
∧ (∀ *h F*. ((*h* : *X* → *F*) ∧ ($h \circ_c f = h \circ_c g$)) ⟶ (∃! *k*. (*k* : *E* → *F*) ∧ $k \circ_c m = h$))
using *assms* **unfolding** *coequalizer-def cfunc-type-def* **by** *auto*

The lemma below corresponds to Exercise 2.3.1 in Halvorson.

lemma *coequalizer-unique*:
assumes *coequalizer E m f g coequalizer F n f g*
shows *E* ≅ *F*
proof −
obtain *k* **where** *k-def*: *k*: *E* → *F* ∧ $k \circ_c m = n$
by (*typecheck-cfuncs*, *metis assms cfunc-type-def coequalizer-def*)
obtain *k'* **where** *k'-def*: *k'*: *F* → *E* ∧ $k' \circ_c n = m$
by (*typecheck-cfuncs*, *metis assms cfunc-type-def coequalizer-def*)
obtain *k''* **where** *k''-def*: *k''*: *F* → *F* ∧ $k'' \circ_c n = n$
by (*typecheck-cfuncs*, *smt* (*verit*) *assms*(*2*) *cfunc-type-def coequalizer-def*)

have *k''-def2*: *k''* = *id F*
using *assms*(*2*) *coequalizer-def id-left-unit2 k''-def* **by** (*typecheck-cfuncs*, *blast*)
have *kk'-idF*: $k \circ_c k'$ = *id F*
by (*typecheck-cfuncs*, *smt* (*verit*) *assms*(*2*) *cfunc-type-def coequalizer-def comp-associative k''-def k''-def2 k'-def k-def*)
have *k'k-idE*: $k' \circ_c k$ = *id E*
by (*typecheck-cfuncs*, *smt* (*verit*) *assms*(*1*) *coequalizer-def comp-associative2 id-left-unit2 k'-def k-def*)

show *E* ≅ *F*
using *cfunc-type-def is-isomorphic-def isomorphism-def k'-def k'k-idE k-def kk'-idF* **by** *fastforce*
qed

The lemma below corresponds to Exercise 2.3.2 in Halvorson.

lemma *coequalizer-is-epimorphism*:
  coequalizer E m f g $\Longrightarrow$ *epimorphism*(m)
  unfolding *coequalizer-def epimorphism-def*
proof *clarify*
  fix $k\ h\ X\ Y$
  assume *f-type*: $f : Y \rightarrow X$
  assume *g-type*: $g : Y \rightarrow X$
  assume *m-type*: $m : X \rightarrow E$
  assume *fm-gm*: $m \circ_c f = m \circ_c g$
  assume *uniqueness*: $\forall h\ F.\ h : X \rightarrow F \wedge h \circ_c f = h \circ_c g \longrightarrow (\exists !k.\ k : E \rightarrow F \wedge k \circ_c m = h)$
  assume *relation-k*: *domain* k =*codomain* m
  assume *relation-h*: *domain* h = *codomain* m
  assume *m-k-mh*: $k \circ_c m = h \circ_c m$

  have $k \circ_c m \circ_c f = h \circ_c m \circ_c g$
    using *cfunc-type-def comp-associative fm-gm g-type m-k-mh m-type relation-k relation-h* **by** *auto*

  then obtain z **where** z: $E \rightarrow$ *codomain*(k) $\wedge z \circ_c m = k \circ_c m \wedge$
   ($\forall$ j. j:$E \rightarrow$ *codomain*(k) $\wedge$ $j \circ_c m = k \circ_c m \longrightarrow j = z$)
   using *uniqueness* **by** (*smt cfunc-type-def codomain-comp comp-associative domain-comp f-type g-type m-k-mh m-type relation-k relation-h*)

  then show $k = h$
   by (*metis cfunc-type-def codomain-comp m-k-mh m-type relation-k relation-h*)
qed

lemma *canonical-quotient-map-is-coequalizer*:
  assumes *equiv-rel-on* X (R,m)
  shows *coequalizer* (X // (R,m)) (*equiv-class* (R,m))
          (*left-cart-proj* $X\ X \circ_c m$) (*right-cart-proj* $X\ X \circ_c m$)
  unfolding *coequalizer-def*
proof(*rule exI*[**where** x=X], *intro exI*[**where** x= R], *safe*)
  have *m-type*: m: $R \rightarrow X \times_c X$
   using *assms equiv-rel-on-def subobject-of-def2 transitive-on-def* **by** *blast*
  show *left-cart-proj* $X\ X \circ_c m : R \rightarrow X$
   using *m-type* **by** *typecheck-cfuncs*
  show *right-cart-proj* $X\ X \circ_c m : R \rightarrow X$
   using *m-type* **by** *typecheck-cfuncs*
  show *equiv-class* (R, m) : $X \rightarrow X$ // (R, m)
   by (*simp add*: *assms equiv-class-type*)
  show [*left-cart-proj* $X\ X \circ_c m$]$_{(R,\ m)}$ = [*right-cart-proj* $X\ X \circ_c m$]$_{(R,\ m)}$
  proof(*rule one-separator*[**where** X=R, **where** Y = X // (R,m)], *typecheck-cfuncs*)
   show [*left-cart-proj* $X\ X \circ_c m$]$_{(R,\ m)}$: $R \rightarrow X$ // (R, m)
    using *m-type assms* **by** *typecheck-cfuncs*
   show [*right-cart-proj* $X\ X \circ_c m$]$_{(R,\ m)}$: $R \rightarrow X$ // (R, m)

using *m-type assms* **by** *typecheck-cfuncs*
next
fix *x*
assume *x-type*: $x \in_c R$
then have *m-x-type*: $m \circ_c x \in_c X \times_c X$
using *m-type* **by** *typecheck-cfuncs*
then obtain *a b* **where** *a-type*: $a \in_c X$ **and** *b-type*: $b \in_c X$ **and** *m-x-eq*: $m \circ_c x = \langle a,b \rangle$
using *cart-prod-decomp* **by** *blast*
then have *ab-inR-relXX*: $\langle a,b \rangle \in_{X \times_c X} (R,m)$
using *assms cfunc-type-def equiv-rel-on-def factors-through-def m-x-type reflexive-on-def relative-member-def2 x-type* **by** *auto*
then have *equiv-class* $(R, m) \circ_c a =$ *equiv-class* $(R, m) \circ_c b$
using *equiv-class-eq assms relative-member-def* **by** *blast*
then have *equiv-class* $(R, m) \circ_c$ *left-cart-proj X X* $\circ_c \langle a,b \rangle =$ *equiv-class* $(R, m) \circ_c$ *right-cart-proj X X* $\circ_c \langle a,b \rangle$
using *a-type b-type left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod* **by** *auto*
then have *equiv-class* $(R, m) \circ_c$ *left-cart-proj X X* $\circ_c m \circ_c x =$ *equiv-class* $(R, m) \circ_c$ *right-cart-proj X X* $\circ_c m \circ_c x$
by (*simp add*: *m-x-eq*)
then show $[$*left-cart-proj X X* $\circ_c m]_{(R,\ m)} \circ_c x = [$*right-cart-proj X X* $\circ_c m]_{(R,\ m)} \circ_c x$
using *x-type m-type assms* **by** (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative m-x-eq*)
qed
next
fix *h F*
assume *h-type*: $h : X \to F$
assume *h-proj1-eqs-h-proj2*: $h \circ_c$ *left-cart-proj X X* $\circ_c m = h \circ_c$ *right-cart-proj X X* $\circ_c m$

have *m-type*: $m : R \to X \times_c X$
using *assms equiv-rel-on-def reflexive-on-def subobject-of-def2* **by** *blast*
have *const-on-rel X* (R, m) *h*
proof (*unfold const-on-rel-def*, *clarify*)
fix *x y*
assume *x-type*: $x \in_c X$ **and** *y-type*: $y \in_c X$
assume $\langle x,y \rangle \in_{X \times_c X} (R, m)$
then obtain *xy* **where** *xy-type*: $xy \in_c R$ **and** *m-h-eq*: $m \circ_c xy = \langle x,y \rangle$
unfolding *relative-member-def2 factors-through-def* **using** *cfunc-type-def* **by** *auto*

have $h \circ_c$ *left-cart-proj X X* $\circ_c m \circ_c xy = h \circ_c$ *right-cart-proj X X* $\circ_c m \circ_c xy$
using *h-type m-type xy-type* **by** (*typecheck-cfuncs*, *smt comp-associative2 comp-type h-proj1-eqs-h-proj2*)
then have $h \circ_c$ *left-cart-proj X X* $\circ_c \langle x,y \rangle = h \circ_c$ *right-cart-proj X X* $\circ_c \langle x,y \rangle$
using *m-h-eq* **by** *auto*
then show $h \circ_c x = h \circ_c y$
using *left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod x-type y-type* **by** *auto*

qed
then show $\exists k.\ k : X \mathbin{/\!/} (R, m) \rightarrow F \wedge k \circ_c$ *equiv-class* $(R, m) = h$
using *assms h-type quotient-func-type quotient-func-eq*
by (*intro exI*[**where** *x=quotient-func h* (*R*, *m*)], *safe*)
next
fix *F k y*
assume *k-type*[*type-rule*]: $k : X \mathbin{/\!/} (R, m) \rightarrow F$
assume *y-type*[*type-rule*]: $y : X \mathbin{/\!/} (R, m) \rightarrow F$
assume *k-equiv-class-type*[*type-rule*]: $k \circ_c$ *equiv-class* $(R, m) : X \rightarrow F$
assume *k-equiv-class-eq*: $(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *left-cart-proj X X* $\circ_c m =$
$(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *right-cart-proj X X* $\circ_c m$
assume *y-k-eq*: $y \circ_c$ *equiv-class* $(R, m) = k \circ_c$ *equiv-class* (R, m)

have *m-type*[*type-rule*]: $m : R \rightarrow X \times_c X$
using *assms equiv-rel-on-def reflexive-on-def subobject-of-def2* **by** *blast*

have *y-eq*: $y =$ *quotient-func* $(y \circ_c$ *equiv-class* $(R, m))\ (R, m)$
using *assms y-k-eq*
proof (*etcs-rule quotient-func-unique*[**where** *X=X*, **where** *Y=F*], *unfold const-on-rel-def*, *safe*)
fix *a b*
assume *a-type*[*type-rule*]: $a \in_c X$ **and** *b-type*[*type-rule*]: $b \in_c X$
assume *ab-in-R*: $\langle a,b\rangle \in_{X \times_c X} (R, m)$
then obtain *h* **where** *h-type*[*type-rule*]: $h \in_c R$ **and** *m-h-eq*: $m \circ_c h = \langle a, b\rangle$
unfolding *relative-member-def factors-through-def* **using** *cfunc-type-def* **by** *auto*

have $(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *left-cart-proj X X* $\circ_c m \circ_c h =$
$(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *right-cart-proj X X* $\circ_c m \circ_c h$
using *assms*
by (*typecheck-cfuncs*, *smt comp-associative2 comp-type k-equiv-class-eq*)
then have $(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *left-cart-proj X X* $\circ_c \langle a, b\rangle =$
$(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *right-cart-proj X X* $\circ_c \langle a, b\rangle$
by (*simp add*: *m-h-eq*)
then show $(y \circ_c$ *equiv-class* $(R, m)) \circ_c a = (y \circ_c$ *equiv-class* $(R, m)) \circ_c b$
using *a-type b-type left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod y-k-eq* **by** *auto*
qed

have *k-eq*: $k =$ *quotient-func* $(y \circ_c$ *equiv-class* $(R, m))\ (R, m)$
using *assms sym*[*OF y-k-eq*]
proof (*etcs-rule quotient-func-unique*[**where** *X=X*, **where** *Y=F*], *unfold const-on-rel-def*, *safe*)
fix *a b*
assume *a-type*: $a \in_c X$ **and** *b-type*: $b \in_c X$
assume *ab-in-R*: $\langle a,b\rangle \in_{X \times_c X} (R, m)$
then obtain *h* **where** *h-type*: $h \in_c R$ **and** *m-h-eq*: $m \circ_c h = \langle a, b\rangle$
unfolding *relative-member-def factors-through-def* **using** *cfunc-type-def* **by** *auto*

have $(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *left-cart-proj* $X\ X \circ_c m \circ_c h =$
$(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *right-cart-proj* $X\ X \circ_c m \circ_c h$
using *k-type m-type h-type assms*
by (*typecheck-cfuncs*, *smt comp-associative2 comp-type k-equiv-class-eq*)
then have $(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *left-cart-proj* $X\ X \circ_c \langle a, b\rangle =$
$(k \circ_c$ *equiv-class* $(R, m)) \circ_c$ *right-cart-proj* $X\ X \circ_c \langle a, b\rangle$
by (*simp add*: *m-h-eq*)
then show $(y \circ_c$ *equiv-class* $(R, m)) \circ_c a = (y \circ_c$ *equiv-class* $(R, m)) \circ_c b$
using *a-type b-type left-cart-proj-cfunc-prod right-cart-proj-cfunc-prod y-k-eq*
by *auto*
qed
show $k = y$
using *y-eq k-eq* **by** *auto*
qed

lemma *canonical-quot-map-is-epi*:
assumes *equiv-rel-on* X (R,m)
shows *epimorphism*((*equiv-class* (R,m)))
by (*meson assms canonical-quotient-map-is-coequalizer coequalizer-is-epimorphism*)

8.2 Regular Epimorphisms

The definition below corresponds to Definition 2.3.4 in Halvorson.

definition *regular-epimorphism* :: *cfunc* ⇒ *bool* **where**
regular-epimorphism $f = (\exists\ g\ h.$ *coequalizer* (*codomain* f) $f\ g\ h)$

The lemma below corresponds to Exercise 2.3.5 in Halvorson.

lemma *reg-epi-and-mono-is-iso*:
assumes $f : X \to Y$ *regular-epimorphism* f *monomorphism* f
shows *isomorphism* f
proof −
obtain $g\ h$ **where** *gh-def*: *coequalizer* (*codomain* f) $f\ g\ h$
using *assms*(*2*) *regular-epimorphism-def* **by** *auto*
obtain W **where** *W-def*: $(g\colon W \to X) \wedge (h\colon W \to X) \wedge$ (*coequalizer* $Y\ f\ g\ h$)
using *assms*(*1*) *cfunc-type-def coequalizer-def gh-def* **by** *fastforce*
have *fg-eqs-fh*: $f \circ_c g = f \circ_c h$
using *coequalizer-def gh-def* **by** *blast*
then have $id(X) \circ_c g = id(X) \circ_c h$
using *W-def assms*(*1*,*3*) *monomorphism-def2* **by** *blast*
then obtain j **where** *j-def*: $j\colon Y \to X \wedge j \circ_c f = id(X)$
using *assms*(*1*) *W-def coequalizer-def2* **by** (*typecheck-cfuncs*, *blast*)
have $id(Y) \circ_c f = f \circ_c id(X)$
using *assms*(*1*) *id-left-unit2 id-right-unit2* **by** *auto*
also have ... $= (f \circ_c j) \circ_c f$
using *assms*(*1*) *comp-associative2 j-def* **by** *fastforce*
ultimately have $id(Y) = f \circ_c j$
by (*typecheck-cfuncs*, *metis W-def assms*(*1*) *coequalizer-is-epimorphism epimorphism-def3 j-def*)

then show *isomorphism f*
using *assms(1) cfunc-type-def isomorphism-def j-def* **by** *fastforce*
qed

The two lemmas below correspond to Proposition 2.3.6 in Halvorson.

lemma *epimorphism-coequalizer-kernel-pair*:
assumes $f : X \to Y$ *epimorphism f*
shows *coequalizer Y f (fibered-product-left-proj X f f X) (fibered-product-right-proj X f f X)*
unfolding *coequalizer-def*
proof (*rule exI*[**where** $x = X$], *rule exI*[**where** $x{=}X\ {}_f{\times}_{cf}\ X$], *safe*)
show *fibered-product-left-proj X f f X* : $X\ {}_f{\times}_{cf}\ X \to X$
using *assms* **by** *typecheck-cfuncs*
show *fibered-product-right-proj X f f X* : $X\ {}_f{\times}_{cf}\ X \to X$
using *assms* **by** *typecheck-cfuncs*
show $f : X \to Y$
using *assms* **by** *typecheck-cfuncs*
show $f \circ_c$ *fibered-product-left-proj X f f X* = $f \circ_c$ *fibered-product-right-proj X f f X*
using *fibered-product-is-pullback assms* **unfolding** *is-pullback-def* **by** *auto*
next
fix *g E*
assume *g-type*: $g : X \to E$
assume *g-eq*: $g \circ_c$ *fibered-product-left-proj X f f X* = $g \circ_c$ *fibered-product-right-proj X f f X*

define *F* **where** *F-def*: *F = quotient-set X* ($X\ {}_f{\times}_{cf}\ X$, *fibered-product-morphism X f f X*)
obtain *q* **where** *q-def*: *q = equiv-class* ($X\ {}_f{\times}_{cf}\ X$, *fibered-product-morphism X f f X*) **and**
q-type[*type-rule*]: $q : X \to F$
using *F-def assms(1) equiv-class-type kernel-pair-equiv-rel* **by** *auto*
obtain *f-bar* **where** *f-bar-def*: *f-bar = quotient-func f* ($X\ {}_f{\times}_{cf}\ X$, *fibered-product-morphism X f f X*) **and**
f-bar-type[*type-rule*]: *f-bar* : $F \to Y$
using *F-def assms(1) const-on-rel-def fibered-product-pair-member kernel-pair-equiv-rel quotient-func-type* **by** *auto*
have *fibr-proj-left-type*[*type-rule*]: *fibered-product-left-proj F (f-bar) (f-bar) F* : $F\ {}_{(f\text{-}bar)}{\times}_{c(f\text{-}bar)}\ F \to F$
by *typecheck-cfuncs*
have *fibr-proj-right-type*[*type-rule*]: *fibered-product-right-proj F (f-bar) (f-bar) F* : $F\ {}_{(f\text{-}bar)}{\times}_{c(f\text{-}bar)}\ F \to F$
by *typecheck-cfuncs*

have *f-eqs*: *f-bar* $\circ_c$ *q* = *f*
  proof −
    have *fact1*: *equiv-rel-on X* (*X* $_f\times_{cf}$ *X*, *fibered-product-morphism X f f X*)
      by (*meson assms*(*1*) *kernel-pair-equiv-rel*)
    have *fact2*: *const-on-rel X* (*X* $_f\times_{cf}$ *X*, *fibered-product-morphism X f f X*) *f*
      using *assms*(*1*) *const-on-rel-def fibered-product-pair-member* **by** *presburger*
    show *?thesis*
      using *assms*(*1*) *f-bar-def fact1 fact2 q-def quotient-func-eq* **by** *blast*
qed

have ∃! *b*. *b* : *X* $_f\times_{cf}$ *X* → *F* $_{(f\text{-}bar)}\times_{c(f\text{-}bar)}$ *F* ∧
  fibered-product-left-proj F (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-left-proj X f f X* ∧
  fibered-product-right-proj F (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-right-proj X f f X* ∧
  epimorphism b
proof(*rule kernel-pair-connection*[**where** *Y* = *Y*])
  show *f* : *X* → *Y*
    using *assms* **by** *typecheck-cfuncs*
  show *q* : *X* → *F*
    by *typecheck-cfuncs*
  show *epimorphism q*
    using *assms*(*1*) *canonical-quot-map-is-epi kernel-pair-equiv-rel q-def* **by** *blast*
  show *f-bar* $\circ_c$ *q* = *f*
    by (*simp add*: *f-eqs*)
  show *q* $\circ_c$ *fibered-product-left-proj X f f X* = *q* $\circ_c$ *fibered-product-right-proj X f f X*
    by (*metis assms*(*1*) *canonical-quotient-map-is-coequalizer coequalizer-def fibered-product-left-proj-def fibered-product-right-proj-def kernel-pair-equiv-rel q-def*)
  show *f-bar* : *F* → *Y*
    by *typecheck-cfuncs*
qed

then obtain *b* **where** *b-type*[*type-rule*]: *b* : *X* $_f\times_{cf}$ *X* → *F* $_{(f\text{-}bar)}\times_{c(f\text{-}bar)}$ *F* **and**
  left-b-eqs: *fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-left-proj X f f X* **and**
  right-b-eqs: *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-right-proj X f f X* **and**
  epi-b: *epimorphism b*
  by *auto*

have *fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F* = *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F*
proof −
have (*fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F*) $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-left-proj X f f X*
by (*simp add*: *left-b-eqs*)
also have ... = *q* $\circ_c$ *fibered-product-right-proj X f f X*
using *assms*(*1*) *canonical-quotient-map-is-coequalizer coequalizer-def fibered-product-left-proj-def fibered-product-right-proj-def kernel-pair-equiv-rel q-def* **by** *fastforce*
also have ... = *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b*
by (*simp add*: *right-b-eqs*)
finally have *fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b*.
then show *?thesis*
using *b-type epi-b epimorphism-def2 fibr-proj-left-type fibr-proj-right-type* **by** *blast*
qed

then obtain *b* **where** *b-type*[*type-rule*]: $b : X\ {}_{f}\times_{cf}\ X \rightarrow F\ {}_{(f\text{-}bar)}\times_{c(f\text{-}bar)}\ F$ **and**
left-b-eqs: *fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-left-proj X f f X* **and**
right-b-eqs: *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-right-proj X f f X* **and**
epi-b: *epimorphism b*
using *b-type epi-b left-b-eqs right-b-eqs* **by** *blast*

have *fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F* = *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F*
proof −
have (*fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F*) $\circ_c$ *b* = *q* $\circ_c$ *fibered-product-left-proj X f f X*
by (*simp add*: *left-b-eqs*)
also have ... = *q* $\circ_c$ *fibered-product-right-proj X f f X*
using *assms*(*1*) *canonical-quotient-map-is-coequalizer coequalizer-def fibered-product-left-proj-def fibered-product-right-proj-def kernel-pair-equiv-rel q-def* **by** *fastforce*
also have ... = *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b*
by (*simp add*: *right-b-eqs*)
finally have *fibered-product-left-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b* = *fibered-product-right-proj F* (*f-bar*) (*f-bar*) *F* $\circ_c$ *b*.
then show *?thesis*
using *b-type epi-b epimorphism-def2 fibr-proj-left-type fibr-proj-right-type* **by** *blast*
qed

then have *mono-fbar*: *monomorphism*(*f-bar*)

by (*typecheck-cfuncs*, *simp add*: *kern-pair-proj-iso-TFAE2*)

have *epimorphism*(*f-bar*)
by (*typecheck-cfuncs*, *metis assms*(*2*) *cfunc-type-def comp-epi-imp-epi f-eqs q-type*)

then have *isomorphism*(*f-bar*)
by (*simp add*: *epi-mon-is-iso mono-fbar*)

obtain *f-bar-inv* **where** *f-bar-inv-type*[*type-rule*]: *f-bar-inv*: $Y \to F$ **and**
f-bar-inv-eq1: *f-bar-inv* $\circ_c$ *f-bar* = *id*(F) **and**
f-bar-inv-eq2: *f-bar* $\circ_c$ *f-bar-inv* = *id*(Y)
using ‹*isomorphism f-bar*› *cfunc-type-def isomorphism-def* **by** (*typecheck-cfuncs*, *force*)

obtain *g-bar* **where** *g-bar-def*: *g-bar* = *quotient-func g* ($X \;{}_f\!\times_{cf}\; X$, *fibered-product-morphism* $X\ f\ f\ X$)
by *auto*
have *const-on-rel* X ($X \;{}_f\!\times_{cf}\; X$, *fibered-product-morphism* $X\ f\ f\ X$) g
unfolding *const-on-rel-def*
by (*meson assms*(*1*) *fibered-product-pair-member2 g-eq g-type*)
then have *g-bar-type*[*type-rule*]: *g-bar* : $F \to E$
using *F-def assms*(*1*) *g-bar-def g-type kernel-pair-equiv-rel quotient-func-type* **by** *blast*
obtain k **where** *k-def*: k = *g-bar* $\circ_c$ *f-bar-inv* **and** *k-type*[*type-rule*]: $k : Y \to E$
by (*typecheck-cfuncs*, *simp*)
then show $\exists\, k.\ k : Y \to E \wedge k \circ_c f = g$
by (*smt* (*z3*) ‹*const-on-rel* X ($X \;{}_f\!\times_{cf}\; X$, *fibered-product-morphism* $X\ f\ f\ X$) g› *assms*(*1*) *comp-associative2 f-bar-inv-eq1 f-bar-inv-type f-bar-type f-eqs g-bar-def g-bar-type g-type id-left-unit2 kernel-pair-equiv-rel q-def q-type quotient-func-eq*)
next
show $\bigwedge F\ k\ y.$
$k \circ_c f : X \to F \Longrightarrow$
$(k \circ_c f) \circ_c$ *fibered-product-left-proj* $X\ f\ f\ X = (k \circ_c f) \circ_c$ *fibered-product-right-proj* $X\ f\ f\ X \Longrightarrow$
$k : Y \to F \Longrightarrow y : Y \to F \Longrightarrow y \circ_c f = k \circ_c f \Longrightarrow k = y$
using *assms epimorphism-def2* **by** *blast*
qed

lemma *epimorphisms-are-regular*:
assumes $f : X \to Y$ *epimorphism f*
shows *regular-epimorphism f*
by (*meson assms*(*2*) *cfunc-type-def epimorphism-coequalizer-kernel-pair regular-epimorphism-def*)

8.3 Epi-monic Factorization

lemma *epi-monic-factorization*:
  assumes *f-type*[*type-rule*]: $f : X \to Y$
  shows $\exists\ g\ m\ E.\ g : X \to E \wedge m : E \to Y$
    $\wedge$ *coequalizer E g* (*fibered-product-left-proj X f f X*) (*fibered-product-right-proj X f f X*)
    $\wedge$ *monomorphism m* $\wedge f = m \circ_c g$
    $\wedge (\forall x.\ x : E \to Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
proof −
  obtain *q* **where** *q-def*: $q =$ *equiv-class* ($X\ {}_f\times_{cf}\ X$, *fibered-product-morphism X f f X*)
    by *auto*
  obtain *E* **where** *E-def*: $E =$ *quotient-set X* ($X\ {}_f\times_{cf}\ X$, *fibered-product-morphism X f f X*)
    by *auto*
  obtain *m* **where** *m-def*: $m =$ *quotient-func f* ($X\ {}_f\times_{cf}\ X$, *fibered-product-morphism X f f X*)
    by *auto*
  show $\exists\ g\ m\ E.\ g : X \to E \wedge m : E \to Y$
    $\wedge$ *coequalizer E g* (*fibered-product-left-proj X f f X*) (*fibered-product-right-proj X f f X*)
    $\wedge$ *monomorphism m* $\wedge f = m \circ_c g$
    $\wedge (\forall x.\ x : E \to Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
  proof (*rule exI*[**where** *x*=*q*], *rule exI* [**where** *x*=*m*], *rule exI*[**where** *x*=*E*], *safe*)
    show *q-type*[*type-rule*]: $q : X \to E$
      unfolding *q-def E-def* **using** *kernel-pair-equiv-rel* **by** (*typecheck-cfuncs*, *blast*)

    have *f-const*: *const-on-rel X* ($X\ {}_f\times_{cf}\ X$, *fibered-product-morphism X f f X*) *f*
      unfolding *const-on-rel-def* **using** *assms fibered-product-pair-member* **by** *auto*
    then show *m-type*[*type-rule*]: $m : E \to Y$
      unfolding *m-def E-def* **using** *kernel-pair-equiv-rel* **by** (*typecheck-cfuncs*, *blast*)

    show *q-coequalizer*: *coequalizer E q* (*fibered-product-left-proj X f f X*) (*fibered-product-right-proj X f f X*)
      unfolding *q-def fibered-product-left-proj-def fibered-product-right-proj-def E-def*
        using *canonical-quotient-map-is-coequalizer f-type kernel-pair-equiv-rel* **by** *auto*
    then have *q-epi*: *epimorphism q*
      using *coequalizer-is-epimorphism* **by** *auto*

    show *m-mono*: *monomorphism m*
    proof −
      have *q-eq*: $q \circ_c$ *fibered-product-left-proj X f f X* $= q \circ_c$ *fibered-product-right-proj X f f X*
        using *canonical-quotient-map-is-coequalizer coequalizer-def f-type fibered-product-left-proj-def fibered-product-right-proj-def kernel-pair-equiv-rel q-def* **by** *fastforce*
      then have $\exists!b.\ b : X\ {}_f\times_{cf}\ X \to E\ {}_m\times_{cm}\ E \wedge$
        fibered-product-left-proj E m m E $\circ_c b = q \circ_c$ *fibered-product-left-proj X f f*

$X \wedge$
fibered-product-right-proj E m m E $\circ_c$ $b = q \circ_c$ *fibered-product-right-proj X f f X* $\wedge$
epimorphism b
by (*typecheck-cfuncs*, *intro kernel-pair-connection*,
simp-all add: *q-epi*, *metis f-const kernel-pair-equiv-rel m-def q-def quotient-func-eq*)
then obtain *b* **where** *b-type*[*type-rule*]: $b : X\ {}_f{\times_c}_f\ X \to E\ {}_m{\times_c}_m\ E$ **and**
b-left-eq: *fibered-product-left-proj E m m E* $\circ_c$ $b = q \circ_c$ *fibered-product-left-proj X f f X* **and**
b-right-eq: *fibered-product-right-proj E m m E* $\circ_c$ $b = q \circ_c$ *fibered-product-right-proj X f f X* **and**
b-epi: *epimorphism b*
by *auto*

have *fibered-product-left-proj E m m E* $\circ_c$ $b =$ *fibered-product-right-proj E m m E* $\circ_c$ b
using *b-left-eq b-right-eq q-eq* **by** *force*
then have *fibered-product-left-proj E m m E* = *fibered-product-right-proj E m m E*
using *b-epi cfunc-type-def epimorphism-def* **by** (*typecheck-cfuncs-prems*, *auto*)
then show *monomorphism m*
using *kern-pair-proj-iso-TFAE2 m-type* **by** *auto*
qed

show *f-eq-m-q*: $f = m \circ_c q$
using *f-const f-type kernel-pair-equiv-rel m-def q-def quotient-func-eq* **by** *fastforce*

show $\bigwedge x.\ x : E \to Y \Longrightarrow f = x \circ_c q \Longrightarrow x = m$
proof –
fix *x*
assume *x-type*[*type-rule*]: $x : E \to Y$
assume *f-eq-x-q*: $f = x \circ_c q$
have $x \circ_c q = m \circ_c q$
using *f-eq-m-q f-eq-x-q* **by** *auto*
then show $x = m$
using *epimorphism-def2 m-type q-epi q-type x-type* **by** *blast*
qed
qed
qed

lemma *epi-monic-factorization2*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
shows $\exists\ g\ m\ E.\ g : X \to E \wedge m : E \to Y$
$\wedge$ *epimorphism g* $\wedge$ *monomorphism m* $\wedge\ f = m \circ_c g$
$\wedge\ (\forall x.\ x : E \to Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
using *epi-monic-factorization coequalizer-is-epimorphism* **by** (*meson f-type*)

8.3.1 Image of a Function

The definition below corresponds to Definition 2.3.7 in Halvorson.

definition *image-of* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cset* (-⦇-⦈- [*101*,*0*,*0*]*100*) **where**
image-of f A n = (*SOME fA*. ∃ *g m*.
g : *A* → *fA* ∧
m : *fA* → *codomain f* ∧
coequalizer fA g (*fibered-product-left-proj A* ($f \circ_c n$) ($f \circ_c n$) *A*) (*fibered-product-right-proj A* ($f \circ_c n$) ($f \circ_c n$) *A*) ∧
monomorphism m ∧ $f \circ_c n = m \circ_c g$ ∧ (∀ *x*. *x* : *fA* → *codomain f* ⟶ $f \circ_c n = x \circ_c g$ ⟶ *x* = *m*))

lemma *image-of-def2*:
assumes *f* : *X* → *Y* *n* : *A* → *X*
shows ∃ *g m*.
g : *A* → $f⦇A⦈_n$ ∧
m : $f⦇A⦈_n$ → *Y* ∧
coequalizer ($f⦇A⦈_n$) *g* (*fibered-product-left-proj A* ($f \circ_c n$) ($f \circ_c n$) *A*) (*fibered-product-right-proj A* ($f \circ_c n$) ($f \circ_c n$) *A*) ∧
monomorphism m ∧ $f \circ_c n = m \circ_c g$ ∧ (∀ *x*. *x* : $f⦇A⦈_n$ → *Y* ⟶ $f \circ_c n = x \circ_c g$ ⟶ *x* = *m*)
proof −
have ∃ *g m*.
g : *A* → $f⦇A⦈_n$ ∧
m : $f⦇A⦈_n$ → *codomain f* ∧
coequalizer ($f⦇A⦈_n$) *g* (*fibered-product-left-proj A* ($f \circ_c n$) ($f \circ_c n$) *A*) (*fibered-product-right-proj A* ($f \circ_c n$) ($f \circ_c n$) *A*) ∧
monomorphism m ∧ $f \circ_c n = m \circ_c g$ ∧ (∀ *x*. *x* : $f⦇A⦈_n$ → *codomain f* ⟶ $f \circ_c n = x \circ_c g$ ⟶ *x* = *m*)
using *assms cfunc-type-def comp-type epi-monic-factorization*[**where** $f=f \circ_c n$, **where** *X*=*A*, **where** *Y*=*codomain f*]
by (*unfold image-of-def*, *subst someI-ex*, *auto*)
then show *?thesis*
using *assms*(*1*) *cfunc-type-def* **by** *auto*
qed

definition *image-restriction-mapping* :: *cfunc* ⇒ *cset* × *cfunc* ⇒ *cfunc* (-↾- [*101*,*0*]*100*)
where
image-restriction-mapping f An = (*SOME g*. ∃ *m*. *g* : *fst An* → $f⦇fst\ An⦈_{snd\ An}$ ∧ *m* : $f⦇fst\ An⦈_{snd\ An}$ → *codomain f* ∧
coequalizer ($f⦇fst\ An⦈_{snd\ An}$) *g* (*fibered-product-left-proj* (*fst An*) ($f \circ_c snd\ An$) ($f \circ_c snd\ An$) (*fst An*)) (*fibered-product-right-proj* (*fst An*) ($f \circ_c snd\ An$) ($f \circ_c snd\ An$) (*fst An*)) ∧
monomorphism m ∧ $f \circ_c snd\ An = m \circ_c g$ ∧ (∀ *x*. *x* : $f⦇fst\ An⦈_{snd\ An}$ → *codomain f* ⟶ $f \circ_c snd\ An = x \circ_c g$ ⟶ *x* = *m*))

lemma *image-restriction-mapping-def2*:
assumes *f* : *X* → *Y* *n* : *A* → *X*
shows ∃ *m*. $f{\restriction}_{(A,\ n)}$: *A* → $f⦇A⦈_n$ ∧ *m* : $f⦇A⦈_n$ → *Y* ∧

coequalizer $(f(\!|A|\!)_n)$ $(f\restriction_{(A,\ n)})$ (*fibered-product-left-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) (*fibered-product-right-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) $\wedge$
monomorphism $m \wedge f \circ_c n = m \circ_c (f\restriction_{(A,\ n)}) \wedge (\forall x.\ x : f(\!|A|\!)_n \to Y \longrightarrow f \circ_c n = x \circ_c (f\restriction_{(A,\ n)}) \longrightarrow x = m)$

proof −
have *codom-f*: *codomain* $f = Y$
using *assms*(*1*) *cfunc-type-def* **by** *auto*
have $\exists\ m.\ f\restriction_{(A,\ n)} : \mathit{fst}\ (A,\ n) \to f(\!|\mathit{fst}\ (A,\ n)|\!)_{\mathit{snd}\ (A,\ n)} \wedge m : f(\!|\mathit{fst}\ (A,\ n)|\!)_{\mathit{snd}\ (A,\ n)} \to \mathit{codomain}\ f\ \wedge$
coequalizer $(f(\!|\mathit{fst}\ (A,\ n)|\!)_{\mathit{snd}\ (A,\ n)})$ $(f\restriction_{(A,\ n)})$ (*fibered-product-left-proj* (*fst* $(A,\ n)$) $(f \circ_c \mathit{snd}\ (A,\ n))$ $(f \circ_c \mathit{snd}\ (A,\ n))$ (*fst* $(A,\ n)$)) (*fibered-product-right-proj* (*fst* $(A,\ n)$) $(f \circ_c \mathit{snd}\ (A,\ n))$ $(f \circ_c \mathit{snd}\ (A,\ n))$ (*fst* $(A,\ n)$)) $\wedge$
monomorphism $m \wedge f \circ_c \mathit{snd}\ (A,\ n) = m \circ_c (f\restriction_{(A,\ n)}) \wedge (\forall x.\ x : f(\!|\mathit{fst}\ (A,\ n)|\!)_{\mathit{snd}\ (A,\ n)} \to \mathit{codomain}\ f \longrightarrow f \circ_c \mathit{snd}\ (A,\ n) = x \circ_c (f\restriction_{(A,\ n)}) \longrightarrow x = m)$
unfolding *image-restriction-mapping-def* **by** (*rule someI-ex*, *insert assms image-of-def2 codom-f*, *auto*)
then show *?thesis*
using *codom-f* **by** *simp*
qed

definition *image-subobject-mapping* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cfunc* ([-(\|-\|)-]*map* [*101*,*0*,*0*]*100*) **where**
$[f(\!|A|\!)_n]\mathit{map} = (\mathit{THE}\ m.\ f\restriction_{(A,\ n)} : A \to f(\!|A|\!)_n \wedge m : f(\!|A|\!)_n \to \mathit{codomain}\ f\ \wedge$
coequalizer $(f(\!|A|\!)_n)$ $(f\restriction_{(A,\ n)})$ (*fibered-product-left-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) (*fibered-product-right-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) $\wedge$
monomorphism $m \wedge f \circ_c n = m \circ_c (f\restriction_{(A,\ n)}) \wedge (\forall x.\ x : (f(\!|A|\!)_n) \to \mathit{codomain}\ f \longrightarrow f \circ_c n = x \circ_c (f\restriction_{(A,\ n)}) \longrightarrow x = m))$

lemma *image-subobject-mapping-def2*:
assumes $f : X \to Y$ $n : A \to X$
shows $f\restriction_{(A,\ n)} : A \to f(\!|A|\!)_n \wedge [f(\!|A|\!)_n]\mathit{map} : f(\!|A|\!)_n \to Y\ \wedge$
coequalizer $(f(\!|A|\!)_n)$ $(f\restriction_{(A,\ n)})$ (*fibered-product-left-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) (*fibered-product-right-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) $\wedge$
monomorphism $([f(\!|A|\!)_n]\mathit{map}) \wedge f \circ_c n = [f(\!|A|\!)_n]\mathit{map} \circ_c (f\restriction_{(A,\ n)}) \wedge (\forall x.\ x : f(\!|A|\!)_n \to Y \longrightarrow f \circ_c n = x \circ_c (f\restriction_{(A,\ n)}) \longrightarrow x = [f(\!|A|\!)_n]\mathit{map})$

proof −
have *codom-f*: *codomain* $f = Y$
using *assms*(*1*) *cfunc-type-def* **by** *auto*
have $f\restriction_{(A,\ n)} : A \to f(\!|A|\!)_n \wedge ([f(\!|A|\!)_n]\mathit{map}) : f(\!|A|\!)_n \to \mathit{codomain}\ f\ \wedge$
coequalizer $(f(\!|A|\!)_n)$ $(f\restriction_{(A,\ n)})$ (*fibered-product-left-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) (*fibered-product-right-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) $\wedge$
monomorphism $([f(\!|A|\!)_n]\mathit{map}) \wedge f \circ_c n = ([f(\!|A|\!)_n]\mathit{map}) \circ_c (f\restriction_{(A,\ n)})\ \wedge$
$(\forall x.\ x : (f(\!|A|\!)_n) \to \mathit{codomain}\ f \longrightarrow f \circ_c n = x \circ_c (f\restriction_{(A,\ n)}) \longrightarrow x = ([f(\!|A|\!)_n]\mathit{map}))$
unfolding *image-subobject-mapping-def*
by (*rule theI'*, *insert assms codom-f image-restriction-mapping-def2*, *blast*)
then show *?thesis*

using *codom-f* **by** *fastforce*
qed

lemma *image-rest-map-type*[*type-rule*]:
assumes $f : X \to Y$ $n : A \to X$
shows $f\restriction_{(A,\ n)} : A \to f(\!|A|\!)_n$
using *assms image-restriction-mapping-def2* **by** *blast*

lemma *image-rest-map-coequalizer*:
assumes $f : X \to Y$ $n : A \to X$
shows *coequalizer* $(f(\!|A|\!)_n)$ $(f\restriction_{(A,\ n)})$ (*fibered-product-left-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A) (*fibered-product-right-proj* A $(f \circ_c n)$ $(f \circ_c n)$ A)
using *assms image-restriction-mapping-def2* **by** *blast*

lemma *image-rest-map-epi*:
assumes $f : X \to Y$ $n : A \to X$
shows *epimorphism* $(f\restriction_{(A,\ n)})$
using *assms image-rest-map-coequalizer coequalizer-is-epimorphism* **by** *blast*

lemma *image-subobj-map-type*[*type-rule*]:
assumes $f : X \to Y$ $n : A \to X$
shows $[f(\!|A|\!)_n]map : f(\!|A|\!)_n \to Y$
using *assms image-subobject-mapping-def2* **by** *blast*

lemma *image-subobj-map-mono*:
assumes $f : X \to Y$ $n : A \to X$
shows *monomorphism* $([f(\!|A|\!)_n]map)$
using *assms image-subobject-mapping-def2* **by** *blast*

lemma *image-subobj-comp-image-rest*:
assumes $f : X \to Y$ $n : A \to X$
shows $[f(\!|A|\!)_n]map \circ_c (f\restriction_{(A,\ n)}) = f \circ_c n$
using *assms image-subobject-mapping-def2* **by** *auto*

lemma *image-subobj-map-unique*:
assumes $f : X \to Y$ $n : A \to X$
shows $x : f(\!|A|\!)_n \to Y \Longrightarrow f \circ_c n = x \circ_c (f\restriction_{(A,\ n)}) \Longrightarrow x = [f(\!|A|\!)_n]map$
using *assms image-subobject-mapping-def2* **by** *blast*

lemma *image-self*:
assumes $f : X \to Y$ **and** *monomorphism* f
assumes $a : A \to X$ **and** *monomorphism* a
shows $f(\!|A|\!)_a \cong A$
proof –
have *monomorphism* $(f \circ_c a)$
using *assms cfunc-type-def composition-of-monic-pair-is-monic* **by** *auto*
then have *monomorphism* $([f(\!|A|\!)_a]map \circ_c (f\restriction_{(A,\ a)}))$
using *assms image-subobj-comp-image-rest* **by** *auto*

then have *monomorphism* $(f{\restriction}_{(A,\ a)})$
by (*meson assms comp-monic-imp-monic′ image-rest-map-type image-subobj-map-type*)
then have *isomorphism* $(f{\restriction}_{(A,\ a)})$
using *assms epi-mon-is-iso image-rest-map-epi* **by** *blast*
then have $A \cong f(\!|A|\!)_a$
using *assms* **unfolding** *is-isomorphic-def* **by** (*intro exI*[**where** $x=f{\restriction}_{(A,\ a)}$], *typecheck-cfuncs*)
then show *?thesis*
by (*simp add*: *isomorphic-is-symmetric*)
qed

The lemma below corresponds to Proposition 2.3.8 in Halvorson.

lemma *image-smallest-subobject*:
assumes *f-type*[*type-rule*]: $f : X \to Y$ **and** *a-type*[*type-rule*]: $a : A \to X$
shows $(B,\ n) \subseteq_c Y \Longrightarrow f$ *factorsthru* $n \Longrightarrow (f(\!|A|\!)_a,\ [f(\!|A|\!)_a]map) \subseteq_Y (B,\ n)$
proof −
assume $(B,\ n) \subseteq_c Y$
then have *n-type*[*type-rule*]: $n : B \to Y$ **and** *n-mono*: *monomorphism* n
unfolding *subobject-of-def2* **by** *auto*
assume f *factorsthru* n
then obtain g **where** *g-type*[*type-rule*]: $g : X \to B$ **and** *f-eq-ng*: $n \circ_c g = f$
using *factors-through-def2* **by** (*typecheck-cfuncs*, *auto*)

have *fa-type*[*type-rule*]: $f \circ_c a : A \to Y$
by (*typecheck-cfuncs*)

obtain *p0* **where** *p0-def*[*simp*]: $p0 =$ *fibered-product-left-proj* $A\ (f \circ_c a)\ (f \circ_c a)\ A$
by *auto*
obtain *p1* **where** *p1-def*[*simp*]: $p1 =$ *fibered-product-right-proj* $A\ (f \circ_c a)\ (f \circ_c a)$ A
by *auto*
obtain E **where** *E-def*[*simp*]: $E = A\ {}_{f \circ_c a}{\times_c}_{f \circ_c a}\ A$
by *auto*

have *fa-coequalizes*: $(f \circ_c a) \circ_c p0 = (f \circ_c a) \circ_c p1$
using *fa-type fibered-product-proj-eq* **by** *auto*
have *ga-coequalizes*: $(g \circ_c a) \circ_c p0 = (g \circ_c a) \circ_c p1$
proof −
from *fa-coequalizes* **have** $n \circ_c ((g \circ_c a) \circ_c p0) = n \circ_c ((g \circ_c a) \circ_c p1)$
by (*auto*, *typecheck-cfuncs*, *auto simp add*: *f-eq-ng comp-associative2*)
then show $(g \circ_c a) \circ_c p0 = (g \circ_c a) \circ_c p1$
using *n-mono* **unfolding** *monomorphism-def2* **by** (*auto*, *typecheck-cfuncs-prems*, *meson*)
qed

have $\forall h\ F.\ h : A \to F \land h \circ_c p0 = h \circ_c p1 \longrightarrow (\exists!k.\ k : f(\!|A|\!)_a \to F \land k \circ_c f{\restriction}_{(A,\ a)} = h)$
using *image-rest-map-coequalizer*[**where** $n=a$] **unfolding** *coequalizer-def*
by (*simp*, *typecheck-cfuncs*, *auto simp add*: *cfunc-type-def*)

then obtain *k* **where** *k-type*[*type-rule*]: $k : f(|A|)_a \rightarrow B$ **and** *k-e-eq-g*: $k \circ_c f\restriction_{(A,\ a)} = g \circ_c a$
using *ga-coequalizes* **by** (*typecheck-cfuncs*, *blast*)

then have $n \circ_c k = [f(|A|)_a]map$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 f-eq-ng g-type image-rest-map-type image-subobj-map-unique k-e-eq-g*)
then show $(f(|A|)_a,\ [f(|A|)_a]map) \subseteq_Y (B,\ n)$
unfolding *relative-subset-def2*
using *image-subobj-map-mono k-type n-mono* **by** (*typecheck-cfuncs*, *blast*)
qed

lemma *images-iso*:
assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$
assumes *m-type*[*type-rule*]: $m : Z \rightarrow X$ **and** *n-type*[*type-rule*]: $n : A \rightarrow Z$
shows $(f \circ_c m)(|A|)_n \cong f(|A|)_{m \circ_c n}$
proof −
have *f-m-image-coequalizer*:
coequalizer $((f \circ_c m)(|A|)_n)$ $((f \circ_c m)\restriction_{(A,\ n)})$
(*fibered-product-left-proj* A $(f \circ_c m \circ_c n)$ $(f \circ_c m \circ_c n)$ A)
(*fibered-product-right-proj* A $(f \circ_c m \circ_c n)$ $(f \circ_c m \circ_c n)$ A)
by (*typecheck-cfuncs*, *smt comp-associative2 image-restriction-mapping-def2*)

have *f-image-coequalizer*:
coequalizer $(f(|A|)_{m \circ_c n})$ $(f\restriction_{(A,\ m \circ_c n)})$
(*fibered-product-left-proj* A $(f \circ_c m \circ_c n)$ $(f \circ_c m \circ_c n)$ A)
(*fibered-product-right-proj* A $(f \circ_c m \circ_c n)$ $(f \circ_c m \circ_c n)$ A)
by (*typecheck-cfuncs*, *smt comp-associative2 image-restriction-mapping-def2*)

from *f-m-image-coequalizer f-image-coequalizer*
show $(f \circ_c m)(|A|)_n \cong f(|A|)_{m \circ_c n}$
by (*meson coequalizer-unique*)
qed

lemma *image-subset-conv*:
assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$
assumes *m-type*[*type-rule*]: $m : Z \rightarrow X$ **and** *n-type*[*type-rule*]: $n : A \rightarrow Z$
shows $\exists\, i.\ ((f \circ_c m)(|A|)_n,\ i) \subseteq_c B \Longrightarrow \exists\, j.\ (f(|A|)_{m \circ_c n},\ j) \subseteq_c B$
proof −
assume $\exists\, i.\ ((f \circ_c m)(|A|)_n,\ i) \subseteq_c B$
then obtain *i* **where**
i-type[*type-rule*]: $i : (f \circ_c m)(|A|)_n \rightarrow B$ **and**
i-mono: *monomorphism i*
unfolding *subobject-of-def* **by** *force*

have $(f \circ_c m)(|A|)_n \cong f(|A|)_{m \circ_c n}$
using *f-type images-iso m-type n-type* **by** *blast*
then obtain *k* **where**
k-type[*type-rule*]: $k : f(|A|)_{m \circ_c n} \rightarrow (f \circ_c m)(|A|)_n$ **and**

k-mono: *monomorphism k*
by (*meson is-isomorphic-def iso-imp-epi-and-monic isomorphic-is-symmetric*)
then show $\exists j.\ (f(\!|A|\!)_{m \circ_c n},\ j) \subseteq_c B$
unfolding *subobject-of-def* **using** *composition-of-monic-pair-is-monic i-mono*
by (*intro exI*[**where** $x{=}i \circ_c k$], *typecheck-cfuncs*, *simp add*: *cfunc-type-def*)
qed

lemma *image-rel-subset-conv*:
assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$
assumes *m-type*[*type-rule*]: $m : Z \rightarrow X$ **and** *n-type*[*type-rule*]: $n : A \rightarrow Z$
assumes *rel-sub1*: $((f \circ_c m)(\!|A|\!)_n,\ [(f \circ_c m)(\!|A|\!)_n]map) \subseteq_Y (B,b)$
shows $(f(\!|A|\!)_{m \circ_c n},\ [f(\!|A|\!)_{m \circ_c n}]map) \subseteq_Y (B,b)$
using *rel-sub1 image-subobj-map-mono*
unfolding *relative-subset-def2*
proof (*typecheck-cfuncs*, *safe*)
fix *k*
assume *k-type*[*type-rule*]: $k : (f \circ_c m)(\!|A|\!)_n \rightarrow B$
assume *b-type*[*type-rule*]: $b : B \rightarrow Y$
assume *b-mono*: *monomorphism b*
assume *b-k-eq-map*: $b \circ_c k = [(f \circ_c m)(\!|A|\!)_n]map$

have *f-m-image-coequalizer*:
coequalizer $((f \circ_c m)(\!|A|\!)_n)\ ((f \circ_c m)\restriction_{(A,\ n)})$
(*fibered-product-left-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A$)
(*fibered-product-right-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A$)
by (*typecheck-cfuncs*, *smt comp-associative2 image-restriction-mapping-def2*)
then have *f-m-image-coequalises*:
$(f \circ_c m)\restriction_{(A,\ n)} \circ_c$ *fibered-product-left-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A$
$= (f \circ_c m)\restriction_{(A,\ n)} \circ_c$ *fibered-product-right-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A$
by (*typecheck-cfuncs-prems*, *unfold coequalizer-def2*, *auto*)

have *f-image-coequalizer*:
coequalizer $(f(\!|A|\!)_{m \circ_c n})\ (f\restriction_{(A,\ m \circ_c n)})$
(*fibered-product-left-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A$)
(*fibered-product-right-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A$)
by (*typecheck-cfuncs*, *smt comp-associative2 image-restriction-mapping-def2*)
then have $\bigwedge h\ F.\ h : A \rightarrow F \Longrightarrow$
$h \circ_c$ *fibered-product-left-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A =$
$h \circ_c$ *fibered-product-right-proj* $A\ (f \circ_c m \circ_c n)\ (f \circ_c m \circ_c n)\ A \Longrightarrow$
$(\exists!k.\ k : f(\!|A|\!)_{m \circ_c n} \rightarrow F \wedge k \circ_c f\restriction_{(A,\ m \circ_c n)} = h)$
by (*typecheck-cfuncs-prems*, *unfold coequalizer-def2*, *auto*)
then have $\exists!k.\ k : f(\!|A|\!)_{m \circ_c n} \rightarrow (f \circ_c m)(\!|A|\!)_n \wedge k \circ_c f\restriction_{(A,\ m \circ_c n)} = (f \circ_c m)\restriction_{(A,\ n)}$
using *f-m-image-coequalises* **by** (*typecheck-cfuncs*, *presburger*)
then obtain k' **where**
k'*-type*[*type-rule*]: $k' : f(\!|A|\!)_{m \circ_c n} \rightarrow (f \circ_c m)(\!|A|\!)_n$ **and**
k'*-eq*: $k' \circ_c f\restriction_{(A,\ m \circ_c n)} = (f \circ_c m)\restriction_{(A,\ n)}$

by *auto*

have *k′-maps-eq*: $[f(\!|A|\!)_m \circ_c n]map = [(f \circ_c m)(\!|A|\!)_n]map \circ_c k'$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 image-subobject-mapping-def2 k′-eq*)
have *k-mono*: *monomorphism k*
by (*metis b-k-eq-map cfunc-type-def comp-monic-imp-monic k-type rel-sub1 relative-subset-def2*)
have *k′-mono*: *monomorphism k′*
by (*smt* (*verit*, *ccfv-SIG*) *cfunc-type-def comp-monic-imp-monic comp-type f-type image-subobject-mapping-def2 k′-maps-eq k′-type m-type n-type*)

show $\exists k.\ k : f(\!|A|\!)_m \circ_c n \to B \wedge b \circ_c k = [f(\!|A|\!)_m \circ_c n]map$
by (*intro exI*[**where** $x{=}k \circ_c k'$], *typecheck-cfuncs*, *simp add*: *b-k-eq-map comp-associative2 k′-maps-eq*)
qed

The lemma below corresponds to Proposition 2.3.9 in Halvorson.

lemma *subset-inv-image-iff-image-subset*:
assumes $(A,a) \subseteq_c X$ $(B,m) \subseteq_c Y$
assumes[*type-rule*]: $f : X \to Y$
shows $((A,\ a) \subseteq_X (f^{-1}(\!|B|\!)_m,[f^{-1}(\!|B|\!)_m]map)) = ((f(\!|A|\!)_a,\ [f(\!|A|\!)_a]map) \subseteq_Y (B,m))$
proof *safe*
have *b-mono*: *monomorphism*(m)
using *assms*(*2*) *subobject-of-def2* **by** *blast*
have *b-type*[*type-rule*]: $m : B \to Y$
using *assms*(*2*) *subobject-of-def2* **by** *blast*
obtain m' **where** *m′-def*: $m' = [f^{-1}(\!|B|\!)_m]map$
by *blast*
then have *m′-type*[*type-rule*]: $m' : f^{-1}(\!|B|\!)_m \to X$
using *assms*(*3*) *b-mono inverse-image-subobject-mapping-type m′-def* **by** (*typecheck-cfuncs*, *force*)

assume $(A,\ a) \subseteq_X (f^{-1}(\!|B|\!)_m,\ [f^{-1}(\!|B|\!)_m]map)$
then have *a-type*[*type-rule*]: $a : A \to X$ **and**
a-mono: *monomorphism a* **and**
k-exists: $\exists k.\ k : A \to f^{-1}(\!|B|\!)_m \wedge [f^{-1}(\!|B|\!)_m]map \circ_c k = a$
unfolding *relative-subset-def2* **by** *auto*
then obtain k **where** *k-type*[*type-rule*]: $k : A \to f^{-1}(\!|B|\!)_m$ **and** *k-a-eq*: $[f^{-1}(\!|B|\!)_m]map \circ_c k = a$
by *auto*

obtain d **where** *d-def*: $d = m' \circ_c k$
by *simp*

obtain j **where** *j-def*: $j = [f(\!|A|\!)_d]map$
by *simp*
then have *j-type*[*type-rule*]: $j : f(\!|A|\!)_d \to Y$

using *assms*(*3*) *comp-type d-def m′-type image-subobj-map-type k-type* **by** *presburger*

obtain *e* **where** *e-def*: $e = f\restriction_{(A,\ d)}$
by *simp*
then have *e-type*[*type-rule*]: $e : A \to f(\!|A|\!)_d$
using *assms*(*3*) *comp-type d-def image-rest-map-type k-type m′-type* **by** *blast*

have *je-equals*: $j \circ_c e = f \circ_c m' \circ_c k$
by (*typecheck-cfuncs*, *simp add*: *d-def e-def image-subobj-comp-image-rest j-def*)

have ($f \circ_c m' \circ_c k$) *factorsthru m*
proof(*typecheck-cfuncs*, *unfold factors-through-def2*)

obtain *middle-arrow* **where** *middle-arrow-def*:
middle-arrow = (*right-cart-proj X B*) $\circ_c$ (*inverse-image-mapping f B m*)
by *simp*

then have *middle-arrow-type*[*type-rule*]: *middle-arrow* : $f^{-1}(\!|B|\!)_m \to B$
unfolding *middle-arrow-def* **using** *b-mono* **by** (*typecheck-cfuncs*)

show $\exists\, h.\ h : A \to B \wedge m \circ_c h = f \circ_c m' \circ_c k$
by (*intro exI*[**where** *x*=*middle-arrow* $\circ_c$ *k*], *typecheck-cfuncs*,
simp add: *b-mono cfunc-type-def comp-associative2 inverse-image-mapping-eq inverse-image-subobject-mapping-def m′-def middle-arrow-def*)
qed

then have $((f \circ_c m' \circ_c k)(\!|A|\!)_{id_c\ A}, [(f \circ_c m' \circ_c k)(\!|A|\!)_{id_c\ A}]map) \subseteq_Y (B, m)$
by (*typecheck-cfuncs*, *meson assms*(*2*) *image-smallest-subobject*)
then have $((f \circ_c a)(\!|A|\!)_{id_c\ A}, [(f \circ_c a)(\!|A|\!)_{id_c\ A}]map) \subseteq_Y (B, m)$
by (*simp add*: *k-a-eq m′-def*)
then show $(f(\!|A|\!)_a, [f(\!|A|\!)_a]map) \subseteq_Y (B, m)$
by (*typecheck-cfuncs*, *metis id-right-unit2 id-type image-rel-subset-conv*)
next
have *m-mono*: *monomorphism*(*m*)
using *assms*(*2*) *subobject-of-def2* **by** *blast*
have *m-type*[*type-rule*]: $m : B \to Y$
using *assms*(*2*) *subobject-of-def2* **by** *blast*

assume $(f(\!|A|\!)_a, [f(\!|A|\!)_a]map) \subseteq_Y (B, m)$
then obtain *s* **where**
s-type[*type-rule*]: $s : f(\!|A|\!)_a \to B$ **and**
m-s-eq-subobj-map: $m \circ_c s = [f(\!|A|\!)_a]map$
unfolding *relative-subset-def2* **by** *auto*

have *a-mono*: *monomorphism a*
using *assms*(*1*) **unfolding** *subobject-of-def2* **by** *auto*

have *pullback-map1-type*[*type-rule*]: $s \circ_c f\restriction_{(A,\ a)} : A \to B$

using *assms*(*1*) **unfolding** *subobject-of-def2* **by** (*auto*, *typecheck-cfuncs*)
have *pullback-map2-type*[*type-rule*]: $a : A \to X$
using *assms*(*1*) **unfolding** *subobject-of-def2* **by** *auto*
have *pullback-maps-commute*: $m \circ_c s \circ_c f{\restriction}_{(A,\ a)} = f \circ_c a$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 image-subobj-comp-image-rest m-s-eq-subobj-map*)

have $\bigwedge Z\ k\ h.\ k : Z \to B \Longrightarrow h : Z \to X \Longrightarrow m \circ_c k = f \circ_c h \Longrightarrow$
$(\exists!j.\ j : Z \to f^{-1}(\!|B|\!)_m \wedge$
$(\textit{right-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j = k \wedge$
$(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c j = h)$
using *inverse-image-pullback assms*(*3*) *m-mono m-type* **unfolding** *is-pullback-def* **by** *simp*
then obtain k **where** *k-type*[*type-rule*]: $k : A \to f^{-1}(\!|B|\!)_m$ **and**
k-right-eq: $(\textit{right-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c k = s \circ_c f{\restriction}_{(A,\ a)}$ **and**
k-left-eq: $(\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c k = a$
using *pullback-map1-type pullback-map2-type pullback-maps-commute* **by** *blast*

have *monomorphism* $((\textit{left-cart-proj}\ X\ B \circ_c \textit{inverse-image-mapping}\ f\ B\ m) \circ_c k) \Longrightarrow$ *monomorphism* k
using *comp-monic-imp-monic′ m-mono* **by** (*typecheck-cfuncs*, *blast*)
then have *monomorphism* k
by (*simp add*: *a-mono k-left-eq*)
then show $(A,\ a) \subseteq_X (f^{-1}(\!|B|\!)_m,\ [f^{-1}(\!|B|\!)_m]map)$
unfolding *relative-subset-def2*
using *assms a-mono m-mono inverse-image-subobject-mapping-mono*
proof (*typecheck-cfuncs*, *safe*)
assume *monomorphism* k
then show $\exists\, k.\ k : A \to f^{-1}(\!|B|\!)_m \wedge [f^{-1}(\!|B|\!)_m]map \circ_c k = a$
using *assms*(*3*) *inverse-image-subobject-mapping-def2 k-left-eq k-type*
by (*intro exI*[**where** x=k], *force*)
qed
qed

The lemma below corresponds to Exercise 2.3.10 in Halvorson.

lemma *in-inv-image-of-image*:
assumes $(A,m) \subseteq_c X$
assumes[*type-rule*]: $f : X \to Y$
shows $(A,m) \subseteq_X (f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map},\ [f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}]map)$
proof –
have *m-type*[*type-rule*]: $m : A \to X$
using *assms*(*1*) **unfolding** *subobject-of-def2* **by** *auto*
have *m-mono*: *monomorphism* m
using *assms*(*1*) **unfolding** *subobject-of-def2* **by** *auto*

have $((f(\!|A|\!)_m,\ [f(\!|A|\!)_m]map) \subseteq_Y (f(\!|A|\!)_m,\ [f(\!|A|\!)_m]map))$
unfolding *relative-subset-def2*
using *m-mono image-subobj-map-mono id-right-unit2 id-type* **by** (*typecheck-cfuncs*,

blast)
then show $(A,m) \subseteq_X (f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}, [f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}]map)$
by (*meson assms relative-subset-def2 subobject-of-def2 subset-inv-image-iff-image-subset*)
qed

8.4 *distribute-left* and *distribute-right* as Equivalence Relations

lemma *left-pair-subset*:
assumes $m : Y \rightarrow X \times_c X$ *monomorphism* m
shows $(Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)) \subseteq_c (X \times_c Z) \times_c (X \times_c Z)$
unfolding *subobject-of-def2* **using** *assms*
proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
fix *g h A*
assume *g-type*: $g : A \rightarrow Y \times_c Z$
assume *h-type*: $h : A \rightarrow Y \times_c Z$
assume (*distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)) \circ_c g =$ (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c h$
then have *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z) \circ_c g =$ *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z) \circ_c h$
using *assms g-type h-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have $(m \times_f id_c\ Z) \circ_c g = (m \times_f id_c\ Z) \circ_c h$
using *assms g-type h-type distribute-right-mono distribute-right-type monomorphism-def2*
by (*typecheck-cfuncs*, *blast*)
then show $g = h$
proof −
have *monomorphism* $(m \times_f id_c\ Z)$
using *assms cfunc-cross-prod-mono id-isomorphism iso-imp-epi-and-monic* **by** (*typecheck-cfuncs*, *blast*)
then show $(m \times_f id_c\ Z) \circ_c g = (m \times_f id_c\ Z) \circ_c h \Longrightarrow g = h$
using *assms g-type h-type* **unfolding** *monomorphism-def2* **by** (*typecheck-cfuncs*, *blast*)
qed
qed

lemma *right-pair-subset*:
assumes $m : Y \rightarrow X \times_c X$ *monomorphism* m
shows $(Z \times_c Y,$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m)) \subseteq_c (Z \times_c X) \times_c (Z \times_c X)$
unfolding *subobject-of-def2* **using** *assms*
proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
fix *g h A*
assume *g-type*: $g : A \rightarrow Z \times_c Y$
assume *h-type*: $h : A \rightarrow Z \times_c Y$
assume (*distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m)) \circ_c g =$ (*distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m)) \circ_c h$
then have *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m) \circ_c g =$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m) \circ_c h$

using *assms g-type h-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have $(id_c\ Z \times_f m) \circ_c g = (id_c\ Z \times_f m) \circ_c h$
using *assms g-type h-type distribute-left-mono distribute-left-type monomorphism-def2*
by (*typecheck-cfuncs*, *blast*)
then show $g = h$
proof −
have *monomorphism* $(id_c\ Z \times_f m)$
using *assms cfunc-cross-prod-mono id-isomorphism id-type iso-imp-epi-and-monic* **by** *blast*
then show $(id_c\ Z \times_f m) \circ_c g = (id_c\ Z \times_f m) \circ_c h \Longrightarrow g = h$
using *assms g-type h-type* **unfolding** *monomorphism-def2* **by** (*typecheck-cfuncs*, *blast*)
qed
qed

lemma *left-pair-reflexive*:
assumes *reflexive-on* X (Y, m)
shows *reflexive-on* $(X \times_c Z)$ $(Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z))$
proof (*unfold reflexive-on-def*, *safe*)
have $m : Y \to X \times_c X \wedge$ *monomorphism* m
using *assms* **unfolding** *reflexive-on-def subobject-of-def2* **by** *auto*
then show $(Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \subseteq_c (X \times_c Z) \times_c X \times_c Z$
by (*simp add*: *left-pair-subset*)
next
fix *xz*
have *m-type*: $m : Y \to X \times_c X$
using *assms* **unfolding** *reflexive-on-def subobject-of-def2* **by** *auto*
assume *xz-type*: $xz \in_c X \times_c Z$
then obtain $x\ z$ **where** *x-type*: $x \in_c X$ **and** *z-type*: $z \in_c Z$ **and** *xz-def*: $xz = \langle x, z\rangle$
using *cart-prod-decomp* **by** *blast*
then show $\langle xz,xz\rangle \in_{(X \times_c Z) \times_c X \times_c Z} (Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z)$
using *m-type*
proof (*clarify*, *typecheck-cfuncs*, *unfold relative-member-def2*, *safe*)
have *monomorphism* m
using *assms* **unfolding** *reflexive-on-def subobject-of-def2* **by** *auto*
then show *monomorphism* (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z$)
using *cfunc-cross-prod-mono cfunc-type-def composition-of-monic-pair-is-monic distribute-right-mono id-isomorphism iso-imp-epi-and-monic m-type* **by** (*typecheck-cfuncs*, *auto*)
next
have *xzxz-type*: $\langle\langle x,z\rangle,\langle x,z\rangle\rangle \in_c (X \times_c Z) \times_c X \times_c Z$
using *xz-type cfunc-prod-type xz-def* **by** *blast*
obtain y **where** *y-def*: $y \in_c Y\ m \circ_c y = \langle x, x\rangle$
using *assms reflexive-def2 x-type* **by** *blast*
have *mid-type*: $m \times_f id_c\ Z : Y \times_c Z \to (X \times_c X) \times_c Z$

by (*simp add*: *cfunc-cross-prod-type id-type m-type*)
have *dist-mid-type*:*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z : Y \times_c Z \rightarrow (X \times_c Z) \times_c X \times_c Z$
using *comp-type distribute-right-type mid-type* **by** *force*
have *yz-type*: $\langle y,z\rangle \in_c Y \times_c Z$
by (*typecheck-cfuncs*, *simp add*: ‹$z \in_c Z$› *y-def*)
have (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c \langle y,z\rangle$ = *distribute-right* $X\ X\ Z \circ_c (m \times_f id(Z)) \circ_c \langle y,z\rangle$
using *comp-associative2 mid-type yz-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *distribute-right* $X\ X\ Z \circ_c \langle m \circ_c y, id(Z) \circ_c z\rangle$
using *z-type cfunc-cross-prod-comp-cfunc-prod m-type y-def* **by** (*typecheck-cfuncs*, *auto*)
also have *distxxz*: ... = *distribute-right* $X\ X\ Z \circ_c \langle \langle x, x\rangle, z\rangle$
using *z-type id-left-unit2 y-def* **by** *auto*
also have ... = $\langle\langle x,z\rangle,\langle x,z\rangle\rangle$
by (*meson z-type distribute-right-ap x-type*)
ultimately show $\langle\langle x,z\rangle,\langle x,z\rangle\rangle$ *factorsthru* (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z$)
using *dist-mid-type distxxz factors-through-def2 xzxz-type yz-type* **by** (*typecheck-cfuncs*, *auto*)
qed
qed

lemma *right-pair-reflexive*:
assumes *reflexive-on* $X\ (Y, m)$
shows *reflexive-on* $(Z \times_c X)\ (Z \times_c Y,$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m))$
proof (*unfold reflexive-on-def*, *safe*)
have $m : Y \rightarrow X \times_c X \wedge$ *monomorphism* m
using *assms* **unfolding** *reflexive-on-def subobject-of-def2* **by** *auto*
then show $(Z \times_c Y,$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m)) \subseteq_c (Z \times_c X) \times_c Z \times_c X$
by (*simp add*: *right-pair-subset*)
next
fix *zx*
have *m-type*: $m : Y \rightarrow X \times_c X$
using *assms* **unfolding** *reflexive-on-def subobject-of-def2* **by** *auto*
assume *zx-type*: $zx \in_c Z \times_c X$
then obtain $z\ x$ **where** *x-type*: $x \in_c X$ **and** *z-type*: $z \in_c Z$ **and** *zx-def*: $zx = \langle z, x\rangle$
using *cart-prod-decomp* **by** *blast*
then show $\langle zx,zx\rangle \in_{(Z \times_c X) \times_c Z \times_c X} (Z \times_c Y,$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m))$
using *m-type*
proof (*clarify*, *typecheck-cfuncs*, *unfold relative-member-def2*, *safe*)
have *monomorphism m*
using *assms* **unfolding** *reflexive-on-def subobject-of-def2* **by** *auto*
then show *monomorphism* (*distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m)$)
using *cfunc-cross-prod-mono cfunc-type-def composition-of-monic-pair-is-monic distribute-left-mono id-isomorphism iso-imp-epi-and-monic m-type* **by** (*typecheck-cfuncs*,

auto)
  next
    have *zxzx-type*: $\langle\langle z,x\rangle,\langle z,x\rangle\rangle \in_c (Z \times_c X) \times_c Z \times_c X$
      using *zx-type cfunc-prod-type zx-def* **by** *blast*
    obtain *y* **where** *y-def*: $y \in_c Y$ $m \circ_c y = \langle x, x\rangle$
      using *assms reflexive-def2 x-type* **by** *blast*
        have *mid-type*: $(id_c\ Z \times_f m) : Z \times_c Y \rightarrow\ \ Z \times_c (X \times_c X)$
      by (*simp add*: *cfunc-cross-prod-type id-type m-type*)
    have *dist-mid-type*:*distribute-left* $Z\ X\ X\ \circ_c (id_c\ Z \times_f m) : Z \times_c Y \rightarrow (Z \times_c X) \times_c Z \times_c X$
      using *comp-type distribute-left-type mid-type* **by** *force*
    have *yz-type*: $\langle z,y\rangle \in_c Z \times_c Y$
      by (*typecheck-cfuncs*, *simp add*: ‹$z \in_c Z$› *y-def*)
    have (*distribute-left* $Z\ X\ X\ \circ_c (id_c\ Z \times_f m)) \circ_c \langle z,y\rangle$ = *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m) \circ_c \langle z,y\rangle$
      using *comp-associative2 mid-type yz-type* **by** (*typecheck-cfuncs*, *auto*)
    also have ... = *distribute-left* $Z\ X\ X\ \circ_c\ \langle id_c\ Z \circ_c z\ ,\ m \circ_c y\ \rangle$
    using *z-type cfunc-cross-prod-comp-cfunc-prod m-type y-def* **by** (*typecheck-cfuncs*, *auto*)
    also have *distxxz*: ... = *distribute-left* $Z\ X\ X\ \circ_c\ \langle z, \langle x, x\rangle\rangle$
      using *z-type id-left-unit2 y-def* **by** *auto*
    also have ... = $\langle\langle z,x\rangle,\langle z,x\rangle\rangle$
      by (*meson z-type distribute-left-ap x-type*)
    ultimately show $\langle\langle z,x\rangle,\langle z,x\rangle\rangle$ *factorsthru* (*distribute-left* $Z\ X\ X\ \circ_c (id_c\ Z \times_f m)$)
      using *z-type distribute-left-ap x-type dist-mid-type factors-through-def2 yz-type zxzx-type* **by** *auto*
  qed
qed

lemma *left-pair-symmetric*:
  assumes *symmetric-on* X (Y, m)
  shows *symmetric-on* $(X \times_c Z)$ $(Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z))$
proof (*unfold symmetric-on-def*, *safe*)
  have $m : Y \rightarrow X \times_c X$ *monomorphism* m
    using *assms subobject-of-def2 symmetric-on-def* **by** *auto*
  then show $(Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \subseteq_c (X \times_c Z) \times_c X \times_c Z$
    by (*simp add*: *left-pair-subset*)
next
  have *m-def*[*type-rule*]: $m : Y \rightarrow X \times_c X$ *monomorphism* m
    using *assms subobject-of-def2 symmetric-on-def* **by** *auto*
  fix *s t*
  assume *s-type*[*type-rule*]: $s \in_c X \times_c Z$
  assume *t-type*[*type-rule*]: $t \in_c X \times_c Z$
  assume *st-relation*: $\langle s,t\rangle \in_{(X \times_c Z) \times_c X \times_c Z} (Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z)$

obtain *sx sz* **where** *s-def*[*type-rule*]: $sx \in_c X$ $sz \in_c Z$ $s = \langle sx,sz\rangle$
  using *cart-prod-decomp s-type* **by** *blast*
obtain *tx tz* **where** *t-def*[*type-rule*]: $tx \in_c X$ $tz \in_c Z$ $t = \langle tx,tz\rangle$
  using *cart-prod-decomp t-type* **by** *blast*

show $\langle t,s\rangle \in_{(X \times_c Z) \times_c (X \times_c Z)} (Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z))$
  using *s-def t-def m-def*
proof (*typecheck-cfuncs*, *clarify*, *unfold relative-member-def2*, *safe*)
  show *monomorphism* (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z$)
    using *relative-member-def2 st-relation* **by** *blast*

  have $\langle\langle sx,sz\rangle, \langle tx,tz\rangle\rangle$ *factorsthru* (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z$)
    using *st-relation s-def t-def* **unfolding** *relative-member-def2* **by** *auto*
  then obtain *yz* **where** *yz-type*[*type-rule*]: $yz \in_c Y \times_c Z$
   and *yz-def*: (*distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)) \circ_c yz = \langle\langle sx,sz\rangle, \langle tx,tz\rangle\rangle$
     using *s-def t-def m-def* **by** (*typecheck-cfuncs*, *unfold factors-through-def2*, *auto*)
  then obtain *y z* **where**
    y-type[*type-rule*]: $y \in_c Y$ **and** *z-type*[*type-rule*]: $z \in_c Z$ **and** *yz-pair*: $yz = \langle y, z\rangle$
    using *cart-prod-decomp* **by** *blast*
  then obtain *my1 my2* **where** *my-types*[*type-rule*]: $my1 \in_c X$ $my2 \in_c X$ **and** *my-def*: $m \circ_c y = \langle my1,my2\rangle$
   by (*metis cart-prod-decomp cfunc-type-def codomain-comp domain-comp m-def*(*1*))
   then obtain y' **where** y'*-type*[*type-rule*]: $y' \in_c Y$ **and** y'*-def*: $m \circ_c y' = \langle my2,my1\rangle$
    using *assms symmetric-def2 y-type* **by** *blast*

  have (*distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)) \circ_c yz = \langle\langle my1,z\rangle, \langle my2,z\rangle\rangle$
  proof −
    have (*distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)) \circ_c yz =$ *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z) \circ_c \langle y, z\rangle$
      unfolding *yz-pair* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
    also have ... = *distribute-right* $X\ X\ Z \circ_c \langle m \circ_c y, id_c\ Z \circ_c z\rangle$
      by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
    also have ... = *distribute-right* $X\ X\ Z \circ_c \langle\langle my1,my2\rangle, z\rangle$
      unfolding *my-def* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)
    also have ... = $\langle\langle my1,z\rangle, \langle my2,z\rangle\rangle$
      using *distribute-right-ap* **by** (*typecheck-cfuncs*, *auto*)
    finally show *?thesis*.
  qed
  then have $\langle\langle sx,sz\rangle,\langle tx,tz\rangle\rangle = \langle\langle my1,z\rangle,\langle my2,z\rangle\rangle$
    using *yz-def* **by** *auto*
  then have $\langle sx,sz\rangle = \langle my1,z\rangle \wedge \langle tx,tz\rangle = \langle my2,z\rangle$
    using *element-pair-eq* **by** (*typecheck-cfuncs*, *auto*)
  then have *eqs*: $sx = my1 \wedge sz = z \wedge tx = my2 \wedge tz = z$
    using *element-pair-eq* **by** (*typecheck-cfuncs*, *auto*)

have (*distribute-right X X Z* $\circ_c$ (*m* $\times_f$ *id*$_c$ *Z*)) $\circ_c$ $\langle y',z\rangle$ = $\langle\langle tx,tz\rangle, \langle sx,sz\rangle\rangle$
proof –
have (*distribute-right X X Z* $\circ_c$ (*m* $\times_f$ *id*$_c$ *Z*)) $\circ_c$ $\langle y',z\rangle$ = *distribute-right X X Z* $\circ_c$ (*m* $\times_f$ *id*$_c$ *Z*) $\circ_c$ $\langle y',z\rangle$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *distribute-right X X Z* $\circ_c$ $\langle m \circ_c y', id_c\ Z \circ_c z\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = *distribute-right X X Z* $\circ_c$ $\langle\langle my2,my1\rangle, z\rangle$
unfolding *y'-def* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)
also have ... = $\langle\langle my2,z\rangle, \langle my1,z\rangle\rangle$
using *distribute-right-ap* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle\langle tx,tz\rangle, \langle sx,sz\rangle\rangle$
using *eqs* **by** *auto*
finally show *?thesis*.
qed
then show $\langle\langle tx,tz\rangle,\langle sx,sz\rangle\rangle$ *factorsthru* (*distribute-right X X Z* $\circ_c$ *m* $\times_f$ *id*$_c$ *Z*)
by (*typecheck-cfuncs*, *metis cfunc-prod-type eqs factors-through-def2 y'-type*)
qed
qed

lemma *right-pair-symmetric*:
assumes *symmetric-on X* (*Y*, *m*)
shows *symmetric-on* (*Z* $\times_c$ *X*) (*Z* $\times_c$ *Y*, *distribute-left Z X X* $\circ_c$ (*id*$_c$ *Z* $\times_f$ *m*))
proof (*unfold symmetric-on-def*, *safe*)
have *m* : *Y* $\rightarrow$ *X* $\times_c$ *X monomorphism m*
using *assms subobject-of-def2 symmetric-on-def* **by** *auto*
then show (*Z* $\times_c$ *Y*, *distribute-left Z X X* $\circ_c$ (*id*$_c$ *Z* $\times_f$ *m*)) $\subseteq_c$ (*Z* $\times_c$ *X*) $\times_c$ *Z* $\times_c$ *X*
by (*simp add*: *right-pair-subset*)
next
have *m-def*[*type-rule*]: *m* : *Y* $\rightarrow$ *X* $\times_c$ *X monomorphism m*
using *assms subobject-of-def2 symmetric-on-def* **by** *auto*

fix *s t*
assume *s-type*[*type-rule*]: *s* $\in_c$ *Z* $\times_c$ *X*
assume *t-type*[*type-rule*]: *t* $\in_c$ *Z* $\times_c$ *X*
assume *st-relation*: $\langle s,t\rangle \in_{(Z \times_c X) \times_c Z \times_c X}$ (*Z* $\times_c$ *Y*, *distribute-left Z X X* $\circ_c$ (*id*$_c$ *Z* $\times_f$ *m*))

obtain *xs zs* **where** *s-def*[*type-rule*]: *xs* $\in_c$ *Z zs* $\in_c$ *X s* = $\langle xs,zs\rangle$
using *cart-prod-decomp s-type* **by** *blast*
obtain *xt zt* **where** *t-def*[*type-rule*]: *xt* $\in_c$ *Z zt* $\in_c$ *X t* = $\langle xt,zt\rangle$
using *cart-prod-decomp t-type* **by** *blast*

show $\langle t,s\rangle \in_{(Z \times_c X) \times_c (Z \times_c X)}$ (*Z* $\times_c$ *Y*, *distribute-left Z X X* $\circ_c$ (*id*$_c$ *Z* $\times_f$ *m*))
using *s-def t-def m-def*
proof (*typecheck-cfuncs*, *clarify*, *unfold relative-member-def2*, *safe*)

show *monomorphism* (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*))
  using *relative-member-def2 st-relation* **by** *blast*

have ⟨⟨*xs*,*zs*⟩, ⟨*xt*,*zt*⟩⟩ *factorsthru* (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*))
  using *st-relation s-def t-def* **unfolding** *relative-member-def2* **by** *auto*
then obtain *zy* **where** *zy-type*[*type-rule*]: *zy* $\in_c$ *Z* $\times_c$ *Y*
  and *zy-def*: (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*)) $\circ_c$ *zy* = ⟨⟨*xs*,*zs*⟩, ⟨*xt*,*zt*⟩⟩
    using *s-def t-def m-def* **by** (*typecheck-cfuncs*, *unfold factors-through-def2*, *auto*)
then obtain *y z* **where**
  y-type[*type-rule*]: *y* $\in_c$ *Y* **and** *z-type*[*type-rule*]: *z* $\in_c$ *Z* **and** *yz-pair*: *zy* = ⟨*z*, *y*⟩
  using *cart-prod-decomp* **by** *blast*
then obtain *my1 my2* **where** *my-types*[*type-rule*]: *my1* $\in_c$ *X* *my2* $\in_c$ *X* **and** *my-def*: *m* $\circ_c$ *y* = ⟨*my2*,*my1*⟩
  by (*metis cart-prod-decomp cfunc-type-def codomain-comp domain-comp m-def*(*1*))
  then obtain *y'* **where** *y'-type*[*type-rule*]: *y'* $\in_c$ *Y* **and** *y'-def*: *m* $\circ_c$ *y'* = ⟨*my1*,*my2*⟩
  using *assms symmetric-def2 y-type* **by** *blast*

have (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*)) $\circ_c$ *zy* = ⟨⟨*z*,*my2*⟩, ⟨*z*,*my1*⟩⟩
proof −
  have (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*)) $\circ_c$ *zy* = *distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*) $\circ_c$ *zy*
    unfolding *yz-pair* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
  also have ... = *distribute-left Z X X* $\circ_c$ ⟨id_c *Z* $\circ_c$ *z* , *m* $\circ_c$ *y*⟩
    by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod yz-pair*)
  also have ... = *distribute-left Z X X* $\circ_c$ ⟨*z* , ⟨*my2*,*my1*⟩⟩
    unfolding *my-def* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)
  also have ... = ⟨⟨*z*,*my2*⟩, ⟨*z*,*my1*⟩⟩
    using *distribute-left-ap* **by** (*typecheck-cfuncs*, *auto*)
  finally show *?thesis*.
qed
then have ⟨⟨*xs*,*zs*⟩,⟨*xt*,*zt*⟩⟩ = ⟨⟨*z*,*my2*⟩,⟨*z*,*my1*⟩⟩
  using *zy-def* **by** *auto*
then have ⟨*xs*,*zs*⟩ = ⟨*z*,*my2*⟩ ∧ ⟨*xt*,*zt*⟩ = ⟨*z*, *my1*⟩
  using *element-pair-eq* **by** (*typecheck-cfuncs*, *auto*)
then have *eqs*: *xs* = *z* ∧ *zs* = *my2* ∧ *xt* = *z* ∧ *zt* = *my1*
  using *element-pair-eq* **by** (*typecheck-cfuncs*, *auto*)

have (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*)) $\circ_c$ ⟨*z*,*y'*⟩ = ⟨⟨*xt*,*zt*⟩, ⟨*xs*,*zs*⟩⟩
proof −
  have (*distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*)) $\circ_c$ ⟨*z*,*y'*⟩ = *distribute-left Z X X* $\circ_c$ (id_c *Z* $\times_f$ *m*) $\circ_c$ ⟨*z*,*y'*⟩
    by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
  also have ... = *distribute-left Z X X* $\circ_c$ ⟨id_c *Z* $\circ_c$ *z*, *m* $\circ_c$ *y'*⟩
    by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
  also have ... = *distribute-left Z X X* $\circ_c$ ⟨*z*, ⟨*my1*,*my2*⟩⟩
    unfolding *y'-def* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)

also have ... = $\langle\langle z,my1\rangle, \langle z,my2\rangle\rangle$
using *distribute-left-ap* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle\langle xt,zt\rangle, \langle xs,zs\rangle\rangle$
using *eqs* **by** *auto*
finally show *?thesis*.
qed
then show $\langle\langle xt,zt\rangle,\langle xs,zs\rangle\rangle$ *factorsthru* (*distribute-left* $Z\ X\ X\ \circ_c$ ($id_c\ Z \times_f m$))
by (*typecheck-cfuncs*, *metis cfunc-prod-type eqs factors-through-def2 y′-type*)
qed
qed

lemma *left-pair-transitive*:
assumes *transitive-on* X (Y, m)
shows *transitive-on* ($X \times_c Z$) ($Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)$)
proof (*unfold transitive-on-def*, *safe*)
have $m : Y \to X \times_c X$ *monomorphism* m
using *assms subobject-of-def2 transitive-on-def* **by** *auto*
then show ($Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z$) $\subseteq_c (X \times_c Z) \times_c X \times_c Z$
by (*simp add*: *left-pair-subset*)
next
have *m-def*[*type-rule*]: $m : Y \to X \times_c X$ *monomorphism* m
using *assms subobject-of-def2 transitive-on-def* **by** *auto*

fix $s\ t\ u$
assume *s-type*[*type-rule*]: $s \in_c X \times_c Z$
assume *t-type*[*type-rule*]: $t \in_c X \times_c Z$
assume *u-type*[*type-rule*]: $u \in_c X \times_c Z$

assume *st-relation*: $\langle s,t\rangle \in_{(X \times_c Z) \times_c X \times_c Z}$ ($Y \times_c Z$, *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z$)
then obtain h **where** *h-type*[*type-rule*]: $h \in_c Y \times_c Z$ **and** *h-def*: (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c h = \langle s,t\rangle$
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
then obtain *hy hz* **where** *h-part-types*[*type-rule*]: $hy \in_c Y$ $hz \in_c Z$ **and** *h-decomp*: $h = \langle hy, hz\rangle$
using *cart-prod-decomp* **by** *blast*
then obtain *mhy1 mhy2* **where** *mhy-types*[*type-rule*]: $mhy1 \in_c X$ $mhy2 \in_c X$ **and** *mhy-decomp*: $m \circ_c hy = \langle mhy1, mhy2\rangle$
using *cart-prod-decomp* **by** (*typecheck-cfuncs*, *blast*)

have $\langle s,t\rangle = \langle\langle mhy1, hz\rangle, \langle mhy2, hz\rangle\rangle$
proof −
have $\langle s,t\rangle$ = (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c \langle hy, hz\rangle$
using *h-decomp h-def* **by** *auto*
also have ... = *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z) \circ_c \langle hy, hz\rangle$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *distribute-right* $X\ X\ Z \circ_c \langle m \circ_c hy, hz\rangle$

by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = ⟨⟨*mhy1*, *hz*⟩, ⟨*mhy2*, *hz*⟩⟩
unfolding *mhy-decomp* **by** (*typecheck-cfuncs*, *simp add*: *distribute-right-ap*)
finally show *?thesis***.**
qed
then have *s-def*: $s = \langle mhy1, hz\rangle$ **and** *t-def*: $t = \langle mhy2, hz\rangle$
using *cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*, *presburger*)

assume *tu-relation*: $\langle t,u\rangle \in_{(X \times_c Z) \times_c X \times_c Z} (Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z)$
then obtain *g* **where** *g-type*[*type-rule*]: $g \in_c Y \times_c Z$ **and** *g-def*: (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c g = \langle t,u\rangle$
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
then obtain *gy gz* **where** *g-part-types*[*type-rule*]: $gy \in_c Y\ gz \in_c Z$ **and** *g-decomp*: $g = \langle gy, gz\rangle$
using *cart-prod-decomp* **by** *blast*
then obtain *mgy1 mgy2* **where** *mgy-types*[*type-rule*]: $mgy1 \in_c X\ mgy2 \in_c X$ **and** *mgy-decomp*: $m \circ_c gy = \langle mgy1, mgy2\rangle$
using *cart-prod-decomp* **by** (*typecheck-cfuncs*, *blast*)

have $\langle t,u\rangle = \langle\langle mgy1, gz\rangle, \langle mgy2, gz\rangle\rangle$
proof −
have $\langle t,u\rangle =$ (*distribute-right* $X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c \langle gy, gz\rangle$
using *g-decomp g-def* **by** *auto*
also have ... = *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z) \circ_c \langle gy, gz\rangle$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *distribute-right* $X\ X\ Z \circ_c \langle m \circ_c gy, gz\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = $\langle\langle mgy1, gz\rangle, \langle mgy2, gz\rangle\rangle$
unfolding *mgy-decomp* **by** (*typecheck-cfuncs*, *simp add*: *distribute-right-ap*)
finally show *?thesis***.**
qed
then have *t-def2*: $t = \langle mgy1, gz\rangle$ **and** *u-def*: $u = \langle mgy2, gz\rangle$
using *cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*, *presburger*)

have *mhy2-eq-mgy1*: $mhy2 = mgy1$
using *t-def2 t-def cart-prod-eq2* **by** (*typecheck-cfuncs-prems*, *auto*)
have *gy-eq-gz*: $hz = gz$
using *t-def2 t-def cart-prod-eq2* **by** (*typecheck-cfuncs-prems*, *auto*)

have *mhy-in-Y*: $\langle mhy1, mhy2\rangle \in_{X \times_c X} (Y, m)$
using *m-def h-part-types mhy-decomp*
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
have *mgy-in-Y*: $\langle mhy2, mgy2\rangle \in_{X \times_c X} (Y, m)$
using *m-def g-part-types mgy-decomp mhy2-eq-mgy1*
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)

have $\langle mhy1, mgy2\rangle \in_{X \times_c X} (Y, m)$
using *assms mhy-in-Y mgy-in-Y mgy-types mhy2-eq-mgy1* **unfolding** *transi-*

tive-on-def
  by (*typecheck-cfuncs*, *blast*)
 then obtain *y* **where** *y-type*[*type-rule*]: $y \in_c Y$ **and** *y-def*: $m \circ_c y = \langle mhy1, mgy2\rangle$
  by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)

 show $\langle s,u\rangle \in_{(X \times_c Z) \times_c X \times_c Z} (Y \times_c Z,\ distribute\text{-}right\ X\ X\ Z \circ_c (m \times_f id_c\ Z))$
 proof (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *safe*)
  show *monomorphism* $(distribute\text{-}right\ X\ X\ Z \circ_c m \times_f id_c\ Z)$
   using *relative-member-def2 st-relation* **by** *blast*

  show $\exists h.\ h \in_c Y \times_c Z \wedge (distribute\text{-}right\ X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c h = \langle s,u\rangle$
   unfolding *s-def u-def gy-eq-gz*
  proof (*intro exI*[**where** $x=\langle y,gz\rangle$], *safe*, *typecheck-cfuncs*)
   have $(distribute\text{-}right\ X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c \langle y,gz\rangle = distribute\text{-}right\ X\ X\ Z \circ_c (m \times_f id_c\ Z) \circ_c \langle y,gz\rangle$
    by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
   also have ... $= distribute\text{-}right\ X\ X\ Z \circ_c \langle m \circ_c y,\ gz\rangle$
   by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
   also have ... $= \langle\langle mhy1,gz\rangle,\langle mgy2,gz\rangle\rangle$
    unfolding *y-def* **by** (*typecheck-cfuncs*, *simp add*: *distribute-right-ap*)
  finally show $(distribute\text{-}right\ X\ X\ Z \circ_c m \times_f id_c\ Z) \circ_c \langle y,gz\rangle = \langle\langle mhy1,gz\rangle,\langle mgy2,gz\rangle\rangle$.
  qed
 qed
qed

lemma *right-pair-transitive*:
 assumes *transitive-on* $X\ (Y,\ m)$
 shows *transitive-on* $(Z \times_c X)\ (Z \times_c Y,\ distribute\text{-}left\ Z\ X\ X \circ_c (id_c\ Z \times_f m))$
proof (*unfold transitive-on-def*, *safe*)
 have $m : Y \rightarrow X \times_c X$ *monomorphism* m
  using *assms subobject-of-def2 transitive-on-def* **by** *auto*
 then show $(Z \times_c Y,\ distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m) \subseteq_c (Z \times_c X) \times_c Z \times_c X$
  by (*simp add*: *right-pair-subset*)
next
 have *m-def*[*type-rule*]: $m : Y \rightarrow X \times_c X$ *monomorphism* m
  using *assms subobject-of-def2 transitive-on-def* **by** *auto*

 fix $s\ t\ u$
 assume *s-type*[*type-rule*]: $s \in_c Z \times_c X$
 assume *t-type*[*type-rule*]: $t \in_c Z \times_c X$
 assume *u-type*[*type-rule*]: $u \in_c Z \times_c X$
 assume *st-relation*: $\langle s,t\rangle \in_{(Z \times_c X) \times_c Z \times_c X} (Z \times_c Y,\ distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m)$
 then obtain *h* **where** *h-type*[*type-rule*]: $h \in_c Z \times_c Y$ **and** *h-def*: $(distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m) \circ_c h = \langle s,t\rangle$
  by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)

then obtain *hy hz* **where** *h-part-types*[*type-rule*]: $hy \in_c Y$ $hz \in_c Z$ **and** *h-decomp*: $h = \langle hz, hy \rangle$
using *cart-prod-decomp* **by** *blast*
then obtain *mhy1 mhy2* **where** *mhy-types*[*type-rule*]: $mhy1 \in_c X$ $mhy2 \in_c X$ **and** *mhy-decomp*: $m \circ_c hy = \langle mhy1, mhy2 \rangle$
using *cart-prod-decomp* **by** (*typecheck-cfuncs*, *blast*)

have $\langle s,t \rangle = \langle \langle hz, mhy1 \rangle, \langle hz, mhy2 \rangle \rangle$
proof −
have $\langle s,t \rangle = (distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m) \circ_c \langle hz, hy \rangle$
using *h-decomp h-def* **by** *auto*
also have ... $= distribute\text{-}left\ Z\ X\ X \circ_c (id_c\ Z \times_f m) \circ_c \langle hz, hy \rangle$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... $= distribute\text{-}left\ Z\ X\ X \circ_c \langle hz, m \circ_c hy \rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... $= \langle \langle hz, mhy1 \rangle, \langle hz, mhy2 \rangle \rangle$
unfolding *mhy-decomp* **by** (*typecheck-cfuncs*, *simp add*: *distribute-left-ap*)
finally show *?thesis*.
qed
then have *s-def*: $s = \langle hz, mhy1 \rangle$ **and** *t-def*: $t = \langle hz, mhy2 \rangle$
using *cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*, *presburger*)

assume *tu-relation*: $\langle t,u \rangle \in_{(Z \times_c X) \times_c Z \times_c X} (Z \times_c Y, distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m)$
then obtain *g* **where** *g-type*[*type-rule*]: $g \in_c Z \times_c Y$ **and** *g-def*: $(distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m) \circ_c g = \langle t,u \rangle$
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
then obtain *gy gz* **where** *g-part-types*[*type-rule*]: $gy \in_c Y$ $gz \in_c Z$ **and** *g-decomp*: $g = \langle gz, gy \rangle$
using *cart-prod-decomp* **by** *blast*
then obtain *mgy1 mgy2* **where** *mgy-types*[*type-rule*]: $mgy1 \in_c X$ $mgy2 \in_c X$ **and** *mgy-decomp*: $m \circ_c gy = \langle mgy2, mgy1 \rangle$
using *cart-prod-decomp* **by** (*typecheck-cfuncs*, *blast*)

have $\langle t,u \rangle = \langle \langle gz, mgy2 \rangle, \langle gz, mgy1 \rangle \rangle$
proof −
have $\langle t,u \rangle = (distribute\text{-}left\ Z\ X\ X \circ_c id_c\ Z \times_f m) \circ_c \langle gz, gy \rangle$
using *g-decomp g-def* **by** *auto*
also have ... $= distribute\text{-}left\ Z\ X\ X \circ_c (id_c\ Z \times_f m) \circ_c \langle gz, gy \rangle$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... $= distribute\text{-}left\ Z\ X\ X \circ_c \langle gz, m \circ_c gy \rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... $= \langle \langle gz, mgy2 \rangle, \langle gz, mgy1 \rangle \rangle$
unfolding *mgy-decomp* **by** (*typecheck-cfuncs*, *simp add*: *distribute-left-ap*)
finally show *?thesis*.
qed
then have *t-def2*: $t = \langle gz, mgy2 \rangle$ **and** *u-def*: $u = \langle gz, mgy1 \rangle$
using *cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*, *presburger*)
have *mhy2-eq-mgy2*: $mhy2 = mgy2$

using *t-def2 t-def cart-prod-eq2* **by** (*typecheck-cfuncs-prems*, *auto*)
have *gy-eq-gz*: $hz = gz$
using *t-def2 t-def cart-prod-eq2* **by** (*typecheck-cfuncs-prems*, *auto*)
have *mhy-in-Y*: $\langle mhy1,\ mhy2\rangle \in_{X \times_c X} (Y,\ m)$
using *m-def h-part-types mhy-decomp*
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
have *mgy-in-Y*: $\langle mhy2,\ mgy1\rangle \in_{X \times_c X} (Y,\ m)$
using *m-def g-part-types mgy-decomp mhy2-eq-mgy2*
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
have $\langle mhy1,\ mgy1\rangle \in_{X \times_c X} (Y,\ m)$
using *assms mhy-in-Y mgy-in-Y mgy-types mhy2-eq-mgy2* **unfolding** *transitive-on-def*
by (*typecheck-cfuncs*, *blast*)
then obtain *y* **where** *y-type*[*type-rule*]: $y \in_c Y$ **and** *y-def*: $m \circ_c y = \langle mhy1,\ mgy1\rangle$
by (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *auto*)
show $\langle s,u\rangle \in_{(Z \times_c X) \times_c Z \times_c X} (Z \times_c Y,\ \textit{distribute-left}\ Z\ X\ X \circ_c id_c\ Z \times_f m)$
proof (*typecheck-cfuncs*, *unfold relative-member-def2 factors-through-def2*, *safe*)
show *monomorphism* $(\textit{distribute-left}\ Z\ X\ X \circ_c id_c\ Z \times_f m)$
using *relative-member-def2 st-relation* **by** *blast*
show $\exists h.\ h \in_c Z \times_c Y \wedge (\textit{distribute-left}\ Z\ X\ X \circ_c id_c\ Z \times_f m) \circ_c h = \langle s,u\rangle$
unfolding *s-def u-def gy-eq-gz*
proof (*intro exI*[**where** $x=\langle gz,y\rangle$], *safe*, *typecheck-cfuncs*)
have $(\textit{distribute-left}\ Z\ X\ X\ \circ_c (id_c\ Z \times_f m)) \circ_c \langle gz,y\rangle = \textit{distribute-left}\ Z\ X\ X\ \circ_c (id_c\ Z \times_f m) \circ_c \langle gz,y\rangle$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have $\ldots = \textit{distribute-left}\ Z\ X\ X\ \circ_c \langle gz,\ m \circ_c y\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have $\ldots = \langle\langle gz,mhy1\rangle,\langle gz,mgy1\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *distribute-left-ap y-def*)
finally show $(\textit{distribute-left}\ Z\ X\ X \circ_c id_c\ Z \times_f m) \circ_c \langle gz,y\rangle = \langle\langle gz,mhy1\rangle,\langle gz,mgy1\rangle\rangle$.
qed
qed
qed

lemma *left-pair-equiv-rel*:
assumes *equiv-rel-on* $X\ (Y,\ m)$
shows *equiv-rel-on* $(X \times_c Z)\ (Y \times_c Z,\ \textit{distribute-right}\ X\ X\ Z \circ_c (m \times_f id\ Z))$
using *assms left-pair-reflexive left-pair-symmetric left-pair-transitive*
by (*unfold equiv-rel-on-def*, *auto*)

lemma *right-pair-equiv-rel*:
assumes *equiv-rel-on* $X\ (Y,\ m)$
shows *equiv-rel-on* $(Z \times_c X)\ (Z \times_c Y,\ \textit{distribute-left}\ Z\ X\ X\ \circ_c (id\ Z \times_f m))$
using *assms right-pair-reflexive right-pair-symmetric right-pair-transitive*
by (*unfold equiv-rel-on-def*, *auto*)

end

9 Coproducts

theory *Coproduct*
imports *Equivalence*
begin

hide-const *case-bool*

The axiomatization below corresponds to Axiom 7 (Coproducts) in Halvorson.

axiomatization
coprod :: *cset* ⇒ *cset* ⇒ *cset* (**infixr** $\coprod$ *65*) **and**
left-coproj :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
right-coproj :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
cfunc-coprod :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (**infixr** ⨿ *65*)
where
left-proj-type[*type-rule*]: *left-coproj X Y* : $X \to X \coprod Y$ **and**
right-proj-type[*type-rule*]: *right-coproj X Y* : $Y \to X \coprod Y$ **and**
cfunc-coprod-type[*type-rule*]: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f ⨿ g : X \coprod Y \to Z$ **and**
left-coproj-cfunc-coprod: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f ⨿ g \circ_c$ (*left-coproj X Y*) $= f$ **and**
right-coproj-cfunc-coprod: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f ⨿ g \circ_c$ (*right-coproj X Y*) $= g$ **and**
cfunc-coprod-unique: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow h : X \coprod Y \to Z \Longrightarrow$
$h \circ_c$ *left-coproj X Y* $= f \Longrightarrow h \circ_c$ *right-coproj X Y* $= g \Longrightarrow h = f ⨿ g$

definition *is-coprod* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool* **where**
is-coprod $W\ i_0\ i_1\ X\ Y \longleftrightarrow$
$(i_0 : X \to W \wedge i_1 : Y \to W \wedge$
$(\forall\ f\ g\ Z.\ (f : X \to Z \wedge g : Y \to Z) \longrightarrow$
$(\exists\ h.\ h : W \to Z \wedge h \circ_c i_0 = f \wedge h \circ_c i_1 = g \wedge$
$(\forall\ h2.\ (h2 : W \to Z \wedge h2 \circ_c i_0 = f \wedge h2 \circ_c i_1 = g) \longrightarrow h2 = h))))$

lemma *is-coprod-def2*:
assumes $i_0 : X \to W\ i_1 : Y \to W$
shows *is-coprod* $W\ i_0\ i_1\ X\ Y \longleftrightarrow$
$(\forall\ f\ g\ Z.\ (f : X \to Z \wedge g : Y \to Z) \longrightarrow$
$(\exists\ h.\ h : W \to Z \wedge h \circ_c i_0 = f \wedge h \circ_c i_1 = g \wedge$
$(\forall\ h2.\ (h2 : W \to Z \wedge h2 \circ_c i_0 = f \wedge h2 \circ_c i_1 = g) \longrightarrow h2 = h)))$
unfolding *is-coprod-def* **using** *assms* **by** *auto*

abbreviation *is-coprod-triple* :: *cset* × *cfunc* × *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool*
where
is-coprod-triple Wi X Y ≡ *is-coprod* (*fst Wi*) (*fst* (*snd Wi*)) (*snd* (*snd Wi*)) *X Y*

lemma *canonical-coprod-is-coprod*:
is-coprod $(X \coprod Y)$ (*left-coproj X Y*) (*right-coproj X Y*) *X Y*

unfolding *is-coprod-def*
proof (*typecheck-cfuncs*)
fix *f g Z*
assume *f-type*: $f : X \to Z$
assume *g-type*: $g : Y \to Z$
show $\exists h.\ h : X \coprod Y \to Z \wedge$
$h \circ_c$ *left-coproj* $X\ Y = f \wedge$
$h \circ_c$ *right-coproj* $X\ Y = g \wedge (\forall h2.\ h2 : X \coprod Y \to Z \wedge h2 \circ_c$ *left-coproj* $X\ Y = f \wedge h2 \circ_c$ *right-coproj* $X\ Y = g \longrightarrow h2 = h)$
using *cfunc-coprod-type cfunc-coprod-unique f-type g-type left-coproj-cfunc-coprod right-coproj-cfunc-coprod*
by(*intro exI*[**where** $x = f \amalg g$], *auto*)
qed

The lemma below is dual to Proposition 2.1.8 in Halvorson.

lemma *coprods-isomorphic*:
assumes *W-coprod*: *is-coprod-triple* $(W, i_0, i_1)\ X\ Y$
assumes *W′-coprod*: *is-coprod-triple* $(W', i'_0, i'_1)\ X\ Y$
shows $\exists\ g.\ g : W \to W' \wedge$ *isomorphism* $g \wedge g \circ_c i_0 = i'_0 \wedge g \circ_c i_1 = i'_1$
proof −
obtain *f* **where** *f-def*: $f : W' \to W \wedge f \circ_c i'_0 = i_0 \wedge f \circ_c i'_1 = i_1$
using *W-coprod W′-coprod* **unfolding** *is-coprod-def*
by (*metis split-pairs*)

obtain *g* **where** *g-def*: $g : W \to W' \wedge g \circ_c i_0 = i'_0 \wedge g \circ_c i_1 = i'_1$
using *W-coprod W′-coprod* **unfolding** *is-coprod-def*
by (*metis split-pairs*)

have *fg0*: $(f \circ_c g) \circ_c i_0 = i_0$
by (*metis W-coprod comp-associative2 f-def g-def is-coprod-def split-pairs*)
have *fg1*: $(f \circ_c g) \circ_c i_1 = i_1$
by (*metis W-coprod comp-associative2 f-def g-def is-coprod-def split-pairs*)

obtain *idW* **where** $idW : W \to W \wedge (\forall\ h2.\ (h2 : W \to W \wedge h2 \circ_c i_0 = i_0 \wedge h2 \circ_c i_1 = i_1) \longrightarrow h2 = idW)$
by (*smt* (*verit*, *best*) *W-coprod is-coprod-def prod.sel*)
then have *fg*: $f \circ_c g = id\ W$
proof *clarify*
assume *idW-unique*: $\forall h2.\ h2 : W \to W \wedge h2 \circ_c i_0 = i_0 \wedge h2 \circ_c i_1 = i_1 \longrightarrow h2 = idW$
have *1*: $f \circ_c g = idW$
using *comp-type f-def fg0 fg1 g-def idW-unique* **by** *blast*
have *2*: $id\ W = idW$
using *W-coprod idW-unique id-left-unit2 id-type is-coprod-def* **by** *auto*
from *1 2* **show** $f \circ_c g = id\ W$
by *auto*
qed

have *gf0*: $(g \circ_c f) \circ_c i'_0 = i'_0$

using *W′-coprod comp-associative2 f-def g-def is-coprod-def* **by** *auto*
have *gf1*: $(g \circ_c f) \circ_c i'_1 = i'_1$
using *W′-coprod comp-associative2 f-def g-def is-coprod-def* **by** *auto*

obtain *idW′* **where** *idW′*: $W' \to W' \wedge (\forall\ h2.\ (h2 : W' \to W' \wedge\ h2 \circ_c i'_0 = i'_0 \wedge h2 \circ_c i'_1 = i'_1) \longrightarrow h2 = idW')$
by (*smt* (*verit*, *best*) *W′-coprod is-coprod-def prod.sel*)
then have *gf*: $g \circ_c f = id\ W'$
proof *clarify*
assume *idW′-unique*: $\forall h2.\ h2 : W' \to W' \wedge h2 \circ_c i'_0 = i'_0 \wedge h2 \circ_c i'_1 = i'_1 \longrightarrow h2 = idW'$
have *1*: $g \circ_c f = idW'$
using *comp-type f-def g-def gf0 gf1 idW′-unique* **by** *blast*
have *2*: $id\ W' = idW'$
using *W′-coprod idW′-unique id-left-unit2 id-type is-coprod-def* **by** *auto*
from *1 2* **show** $g \circ_c f = id\ W'$
by *auto*
qed

have *g-iso*: *isomorphism g*
using *f-def fg g-def gf isomorphism-def3* **by** *blast*
from *g-iso g-def* **show** $\exists\ g.\ g : W \to W' \wedge$ *isomorphism* $g \wedge g \circ_c i_0 = i'_0 \wedge g \circ_c i_1 = i'_1$
by *blast*
qed

9.1 Coproduct Function Properities

lemma *cfunc-coprod-comp*:
assumes $a : Y \to Z$ $b : X \to Y$ $c : W \to Y$
shows $(a \circ_c b) \amalg (a \circ_c c) = a \circ_c (b \amalg c)$
proof −
have $((a \circ_c b) \amalg (a \circ_c c)) \circ_c$ (*left-coproj X W*) $= a \circ_c (b \amalg c) \circ_c$ (*left-coproj X W*)
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
then have *left-coproj-eq*: $((a \circ_c b) \amalg (a \circ_c c)) \circ_c$ (*left-coproj X W*) $= (a \circ_c (b \amalg c)) \circ_c$ (*left-coproj X W*)
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
have $((a \circ_c b) \amalg (a \circ_c c)) \circ_c$ (*right-coproj X W*) $= a \circ_c (b \amalg c) \circ_c$ (*right-coproj X W*)
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
then have *right-coproj-eq*: $((a \circ_c b) \amalg (a \circ_c c)) \circ_c$ (*right-coproj X W*) $= (a \circ_c (b \amalg c)) \circ_c$ (*right-coproj X W*)
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)

show $(a \circ_c b) \amalg (a \circ_c c) = a \circ_c (b \amalg c)$
using *assms left-coproj-eq right-coproj-eq*
by (*typecheck-cfuncs*, *smt cfunc-coprod-unique left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
qed

lemma *id-coprod*:
id($A \coprod B$) = (*left-coproj* A B) II (*right-coproj* A B)
by (*typecheck-cfuncs*, *simp add*: *cfunc-coprod-unique id-left-unit2*)

The lemma below corresponds to Proposition 2.4.1 in Halvorson.

lemma *coproducts-disjoint*:
$x \in_c X \Longrightarrow y \in_c Y \Longrightarrow$ (*left-coproj* X Y) $\circ_c$ $x \neq$ (*right-coproj* X Y) $\circ_c$ y
proof (*rule ccontr*, *clarify*)
assume *x-type*[*type-rule*]: $x \in_c X$
assume *y-type*[*type-rule*]: $y \in_c Y$
assume *BWOC*: ((*left-coproj* X Y) $\circ_c$ x = (*right-coproj* X Y) $\circ_c$ y)
obtain g **where** *g-def*: g *factorsthru* t **and** *g-type*[*type-rule*]: g: $X \to \Omega$
by (*typecheck-cfuncs*, *meson comp-type factors-through-def2 terminal-func-type*)
then have *fact1*: t = $g \circ_c x$
by (*metis cfunc-type-def comp-associative factors-through-def id-right-unit2 id-type*
terminal-func-comp terminal-func-unique true-func-type x-type)

obtain h **where** *h-def*: h *factorsthru* f **and** *h-type*[*type-rule*]: h: $Y \to \Omega$
by (*typecheck-cfuncs*, *meson comp-type factors-through-def2 one-terminal-object terminal-object-def*)
then have *gUh-type*[*type-rule*]: g II h: $X \coprod Y \to \Omega$ **and**
gUh-def: (g II h) $\circ_c$ (*left-coproj* X Y) = g $\wedge$ (g II h) $\circ_c$ (*right-coproj* X Y) = h
using *left-coproj-cfunc-coprod right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have *fact2*: f = ((g II h) $\circ_c$ (*right-coproj* X Y)) $\circ_c$ y
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *comp-associative2 factors-through-def2 gUh-def h-def id-right-unit2 terminal-func-comp-elem terminal-func-unique*)
also have ... = ((g II h) $\circ_c$ (*left-coproj* X Y)) $\circ_c$ x
by (*smt BWOC comp-associative2 gUh-type left-proj-type right-proj-type x-type y-type*)
also have ... = t
by (*simp add*: *fact1 gUh-def*)
ultimately show *False*
using *true-false-distinct* **by** *auto*
qed

The lemma below corresponds to Proposition 2.4.2 in Halvorson.

lemma *left-coproj-are-monomorphisms*:
monomorphism(*left-coproj* X Y)
proof (*cases* $\exists x.\ x \in_c X$)
assume *X-nonempty*: $\exists x.\ x \in_c X$
then obtain x **where** *x-type*[*type-rule*]: $x \in_c X$
by *auto*
then have (*id* X II ($x \circ_c \beta_Y$)) $\circ_c$ *left-coproj* X Y = *id* X
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
then show *monomorphism* (*left-coproj* X Y)

by (*typecheck-cfuncs*, *metis* (*mono-tags*) *cfunc-coprod-type comp-monic-imp-monic′*
comp-type id-isomorphism id-type iso-imp-epi-and-monic terminal-func-type
x-type)
next
show $\nexists x.\ x \in_c X \Longrightarrow$ *monomorphism* (*left-coproj X Y*)
by (*typecheck-cfuncs*, *metis cfunc-type-def injective-def injective-imp-monomorphism*)
qed

lemma *right-coproj-are-monomorphisms*:
monomorphism(*right-coproj X Y*)
proof (*cases* $\exists y.\ y \in_c Y$)
assume *Y-nonempty*: $\exists y.\ y \in_c Y$
then obtain y **where** *y-type*[*type-rule*]: $y \in_c Y$
by *auto*
have $((y \circ_c \beta_X) \amalg id\ Y) \circ_c$ *right-coproj X Y* = *id Y*
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
then show *monomorphism* (*right-coproj X Y*)
by (*typecheck-cfuncs*, *metis* (*mono-tags*) *cfunc-coprod-type comp-monic-imp-monic′*
comp-type id-isomorphism id-type iso-imp-epi-and-monic terminal-func-type
y-type)
next
show $\nexists y.\ y \in_c Y \Longrightarrow$ *monomorphism* (*right-coproj X Y*)
by (*typecheck-cfuncs*, *metis cfunc-type-def injective-def injective-imp-monomorphism*)
qed

The lemma below corresponds to Exercise 2.4.3 in Halvorson.

lemma *coprojs-jointly-surj*:
assumes *z-type*[*type-rule*]: $z \in_c X \coprod Y$
shows ($\exists\ x.\ (x \in_c X \land z =$ (*left-coproj X Y*) $\circ_c x$))
$\lor$ ($\exists\ y.\ (y \in_c Y \land z =$ (*right-coproj X Y*) $\circ_c y$))
proof (*clarify*, *rule ccontr*)
assume *not-in-right-image*: $\nexists y.\ y \in_c Y \land z =$ *right-coproj X Y* $\circ_c y$
assume *not-in-left-image*: $\nexists x.\ x \in_c X \land z =$ *left-coproj X Y* $\circ_c x$

obtain h **where** *h-def*: $h = \mathrm{f} \circ_c \beta_{X \coprod Y}$ **and** *h-type*[*type-rule*]: $h : X \coprod Y \to$
Ω
by (*typecheck-cfuncs*, *simp*)

have *fact1*: (*eq-pred* $(X \coprod Y)) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id\ (X \coprod Y)\rangle) \circ_c$ *left-coproj*
X Y = $h \circ_c$ *left-coproj X Y*
proof(*etcs-rule one-separator*[**where** *X*=*X*, **where** $Y = \Omega$])
show $\bigwedge x.\ x \in_c X \Longrightarrow$ ((*eq-pred* $(X \coprod Y)) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle)$
$\circ_c$ *left-coproj X Y*) $\circ_c x$ =
($h \circ_c$ *left-coproj X Y*) $\circ_c x$
proof −
fix x
assume *x-type*: $x \in_c X$
have ((*eq-pred* $(X \coprod Y)) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle) \circ_c$ *left-coproj X*
Y) $\circ_c x$ =

eq-pred $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle \circ_c$ (*left-coproj X Y* $\circ_c$ *x*)
using *x-type* **by** (*typecheck-cfuncs*, *metis assms cfunc-type-def comp-associative*)
also have ... = f
using *assms eq-pred-false-extract-right not-in-left-image x-type* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = $h \circ_c$ (*left-coproj X Y* $\circ_c$ *x*)
using *x-type* **by** (*typecheck-cfuncs*, *smt comp-associative2 h-def id-right-unit2 id-type terminal-func-comp terminal-func-type terminal-func-unique*)
also have ... = ($h \circ_c$ *left-coproj X Y*) $\circ_c$ *x*
using *x-type cfunc-type-def comp-associative comp-type false-func-type h-def terminal-func-type* **by** (*typecheck-cfuncs*, *force*)
finally show ((*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle) \circ_c$ *left-coproj X Y*) $\circ_c$ *x* = ($h \circ_c$ *left-coproj X Y*) $\circ_c$ *x*.
qed
qed

have *fact2*: (*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id\ (X \coprod Y)\rangle) \circ_c$ *right-coproj X Y* = $h \circ_c$ *right-coproj X Y*
proof(*etcs-rule one-separator*[**where** $X = Y$, **where** $Y = \Omega$])
show $\bigwedge x.\ x \in_c Y \Longrightarrow$
((*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle) \circ_c$ *right-coproj X Y*) $\circ_c x =$
($h \circ_c$ *right-coproj X Y*) $\circ_c x$
proof −
fix *x*
assume *x-type*[*type-rule*]: $x \in_c Y$
have ((*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle) \circ_c$ *right-coproj X Y*) $\circ_c x$ = f
by (*typecheck-cfuncs*, *smt* (*verit*) *assms cfunc-type-def eq-pred-false-extract-right comp-associative comp-type not-in-right-image*)
also have ... = ($h \circ_c$ *right-coproj X Y*) $\circ_c$ *x*
by (*etcs-assocr*,*typecheck-cfuncs*, *metis cfunc-type-def comp-associative h-def id-right-unit2 terminal-func-comp-elem terminal-func-type*)
finally show ((*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle) \circ_c$ *right-coproj X Y*) $\circ_c$ *x* = ($h \circ_c$ *right-coproj X Y*) $\circ_c$ *x*.
qed
qed
have *indicator-is-false*:*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id\ (X \coprod Y)\rangle = h$
proof(*etcs-rule one-separator*[**where** $X = X \coprod Y$, **where** $Y = \Omega$])
show $\bigwedge x.\ x \in_c X \coprod Y \Longrightarrow$ (*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id_c\ (X \coprod Y)\rangle) \circ_c x = h \circ_c x$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-comp fact1 fact2 id-coprod id-right-unit2 left-proj-type right-proj-type*)
qed
have *hz-gives-false*: $h \circ_c z$ = f
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 h-def id-right-unit2 id-type terminal-func-comp terminal-func-type terminal-func-unique*)

then have *indicator-z-gives-false*: (*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id\ (X \coprod Y)\rangle) \circ_c z = \mathrm{f}$
using *assms indicator-is-false* **by** (*typecheck-cfuncs*, *blast*)
then have *indicator-z-gives-true*: (*eq-pred* $(X \coprod Y) \circ_c \langle z \circ_c \beta_{X \coprod Y}, id\ (X \coprod Y)\rangle) \circ_c z = \mathrm{t}$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*verit*, *del-insts*) *comp-associative2 eq-pred-true-extract-right*)
then show *False*
using *indicator-z-gives-false true-false-distinct* **by** *auto*
qed

lemma *maps-into-1u1*:
assumes *x-type*: $x \in_c (\mathbf{1} \coprod \mathbf{1})$
shows $(x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1}) \vee (x = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1})$
using *assms* **by** (*typecheck-cfuncs*, *metis coprojs-jointly-surj terminal-func-unique*)

lemma *coprod-preserves-left-epi*:
assumes $f: X \to Z$ $g: Y \to Z$
assumes *surjective*(f)
shows *surjective*($f \amalg g$)
unfolding *surjective-def*
proof(*clarify*)
fix z
assume *y-type*[*type-rule*]: $z \in_c$ *codomain* $(f \amalg g)$
then obtain x **where** *x-def*: $x \in_c X \wedge f \circ_c x = z$
using *assms cfunc-coprod-type cfunc-type-def cfunc-type-def surjective-def* **by** *auto*
have $(f \amalg g) \circ_c$ (*left-coproj* $X\ Y \circ_c x) = z$
by (*typecheck-cfuncs*, *smt assms comp-associative2 left-coproj-cfunc-coprod x-def*)
then show $\exists x.\ x \in_c$ *domain*$(f \amalg g) \wedge f \amalg g \circ_c x = z$
by (*typecheck-cfuncs*, *metis assms*(*1*,*2*) *cfunc-type-def codomain-comp domain-comp left-proj-type x-def*)
qed

lemma *coprod-preserves-right-epi*:
assumes $f: X \to Z$ $g: Y \to Z$
assumes *surjective*(g)
shows *surjective*($f \amalg g$)
unfolding *surjective-def*
proof(*clarify*)
fix z
assume *y-type*: $z \in_c$ *codomain* $(f \amalg g)$
have *fug-type*: $(f \amalg g) : (X \coprod Y) \to Z$
by (*typecheck-cfuncs*, *simp add*: *assms*)
then have *y-type2*: $z \in_c Z$
using *cfunc-type-def y-type* **by** *auto*
then have $\exists\ y.\ y \in_c Y \wedge g \circ_c y = z$
using *assms*(*2*,*3*) *cfunc-type-def surjective-def* **by** *auto*
then obtain y **where** *y-def*: $y \in_c Y \wedge g \circ_c y = z$

by *blast*
have *coproj-x-type*: *right-coproj X Y* $\circ_c$ *y* $\in_c$ *X* $\coprod$ *Y*
using *comp-type right-proj-type y-def* **by** *blast*
have $(f \amalg g) \circ_c$ (*right-coproj X Y* $\circ_c$ $y) = z$
using *assms(1) assms(2) cfunc-type-def comp-associative fug-type right-coproj-cfunc-coprod right-proj-type y-def* **by** *auto*
then show $\exists y.\ y \in_c$ *domain*$(f \amalg g) \wedge f \amalg g \circ_c y = z$
using *cfunc-type-def coproj-x-type fug-type* **by** *auto*
qed

lemma *coprod-eq*:
assumes $a : X \coprod Y \to Z$ $b : X \coprod Y \to Z$
shows $a = b \longleftrightarrow$
($a \circ_c$ *left-coproj X Y* $= b \circ_c$ *left-coproj X Y*
$\wedge\ a \circ_c$ *right-coproj X Y* $= b \circ_c$ *right-coproj X Y*)
by (*smt assms cfunc-coprod-unique cfunc-type-def codomain-comp domain-comp left-proj-type right-proj-type*)

lemma *coprod-eqI*:
assumes $a : X \coprod Y \to Z$ $b : X \coprod Y \to Z$
assumes ($a \circ_c$ *left-coproj X Y* $= b \circ_c$ *left-coproj X Y*
$\wedge\ a \circ_c$ *right-coproj X Y* $= b \circ_c$ *right-coproj X Y*)
shows $a = b$
using *assms coprod-eq* **by** *blast*

lemma *coprod-eq2*:
assumes $a : X \to Z$ $b : Y \to Z$ $c : X \to Z$ $d : Y \to Z$
shows $(a \amalg b) = (c \amalg d) \longleftrightarrow (a = c \wedge b = d)$
by (*metis assms left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)

lemma *coprod-decomp*:
assumes $a : X \coprod Y \to A$
shows $\exists\ x\ y.\ a = (x \amalg y) \wedge x : X \to A \wedge y : Y \to A$
proof (*rule exI*[**where** x=$a \circ_c$ *left-coproj X Y*], *intro exI*[**where** x=$a \circ_c$ *right-coproj X Y*], *safe*)
show $a = (a \circ_c$ *left-coproj X Y*$) \amalg (a \circ_c$ *right-coproj X Y*$)$
using *assms cfunc-coprod-unique cfunc-type-def codomain-comp domain-comp left-proj-type right-proj-type* **by** *auto*
show $a \circ_c$ *left-coproj X Y* $: X \to A$
by (*meson assms comp-type left-proj-type*)
show $a \circ_c$ *right-coproj X Y* $: Y \to A$
by (*meson assms comp-type right-proj-type*)
qed

The lemma below corresponds to Proposition 2.4.4 in Halvorson.

lemma *truth-value-set-iso-1u1*:
isomorphism(t⨿f)
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *CollectI epi-mon-is-iso injective-def2 injective-imp-monomorphism left-coproj-cfunc-coprod left-proj-type maps-into-1u1*

right-coproj-cfunc-coprod right-proj-type surjective-def2 surjective-is-epimorphism

true-false-distinct true-false-only-truth-values)

9.1.1 Equality Predicate with Coproduct Properities

lemma *eq-pred-left-coproj*:
assumes *u-type*[*type-rule*]: $u \in_c X \coprod Y$ **and** *x-type*[*type-rule*]: $x \in_c X$
shows *eq-pred* $(X \coprod Y) \circ_c \langle u,$ *left-coproj* $X\ Y \circ_c x\rangle = ((\textit{eq-pred}\ X \circ_c \langle \textit{id}\ X,\ x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y)) \circ_c u$
proof (*cases eq-pred* $(X \coprod Y) \circ_c \langle u,$ *left-coproj* $X\ Y \circ_c x\rangle = \mathrm{t}$)
assume *case1*: *eq-pred* $(X \coprod Y) \circ_c \langle u,$ *left-coproj* $X\ Y \circ_c x\rangle = \mathrm{t}$
then have *u-is-left-coproj*: $u =$ *left-coproj* $X\ Y \circ_c x$
using *eq-pred-iff-eq* **by** (*typecheck-cfuncs-prems*, *presburger*)
show *eq-pred* $(X \coprod Y) \circ_c \langle u,$*left-coproj* $X\ Y \circ_c x\rangle = (\textit{eq-pred}\ X \circ_c \langle \textit{id}_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c u$
proof –
have $((\textit{eq-pred}\ X \circ_c \langle \textit{id}\ X,\ x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y)) \circ_c u$
$= ((\textit{eq-pred}\ X \circ_c \langle \textit{id}\ X,\ x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y)) \circ_c$ *left-coproj* $X\ Y \circ_c x$
using *u-is-left-coproj* **by** *auto*
also have ... $= (\textit{eq-pred}\ X \circ_c \langle \textit{id}\ X,\ x \circ_c \beta_X\rangle) \circ_c x$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod*)
also have ... $=$ *eq-pred* $X \circ_c \langle x,\ x\rangle$
by (*typecheck-cfuncs*, *metis cart-prod-extract-left cfunc-type-def comp-associative*)
also have ... $= \mathrm{t}$
using *eq-pred-iff-eq* **by** (*typecheck-cfuncs*, *blast*)
ultimately show *?thesis*
by (*simp add*: *case1*)
qed
next
assume *eq-pred* $(X \coprod Y) \circ_c \langle u,$*left-coproj* $X\ Y \circ_c x\rangle \neq \mathrm{t}$
then have *case2*: *eq-pred* $(X \coprod Y) \circ_c \langle u,$*left-coproj* $X\ Y \circ_c x\rangle = \mathrm{f}$
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have *u-not-left-coproj-x*: $u \neq$ *left-coproj* $X\ Y \circ_c x$
using *eq-pred-iff-eq-conv* **by** (*typecheck-cfuncs-prems*, *blast*)
show *eq-pred* $(X \coprod Y) \circ_c \langle u,$*left-coproj* $X\ Y \circ_c x\rangle = (\textit{eq-pred}\ X \circ_c \langle \textit{id}_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c u$
proof (*cases* $\exists\ g.\ g : \mathbf{1} \to X \wedge u =$ *left-coproj* $X\ Y \circ_c g$)
assume $\exists g.\ g \in_c X \wedge u =$ *left-coproj* $X\ Y \circ_c g$
then obtain g **where** *g-type*[*type-rule*]: $g \in_c X$ **and** *g-def*: $u =$ *left-coproj* $X\ Y \circ_c g$
by *auto*
then have *x-not-g*: $x \neq g$
using *u-not-left-coproj-x* **by** *auto*
show *eq-pred* $(X \coprod Y) \circ_c \langle u,$*left-coproj* $X\ Y \circ_c x\rangle = (\textit{eq-pred}\ X \circ_c \langle \textit{id}_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c u$
proof –
have $(\textit{eq-pred}\ X \circ_c \langle \textit{id}_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c$ *left-coproj* $X\ Y \circ_c g$
$= (\textit{eq-pred}\ X \circ_c \langle \textit{id}_c\ X, x \circ_c \beta_X\rangle) \circ_c g$

using *comp-associative2 left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *force*)
also have ... = *eq-pred* $X \circ_c \langle g,x \rangle$
by (*typecheck-cfuncs*, *simp add*: *cart-prod-extract-left comp-associative2*)
also have ... = f
using *eq-pred-iff-eq-conv x-not-g* **by** (*typecheck-cfuncs*, *blast*)
ultimately show *?thesis*
using *case2 g-def* **by** *argo*
qed
next
assume $\nexists g.\ g \in_c X \wedge u = $ *left-coproj* $X\ Y \circ_c g$
then obtain g **where** *g-type*[*type-rule*]: $g \in_c Y$ **and** *g-def*: $u =$ *right-coproj* $X\ Y \circ_c g$
by (*meson coprojs-jointly-surj u-type*)

show *eq-pred* $(X \coprod Y) \circ_c \langle u,$ *left-coproj* $X\ Y \circ_c x\rangle = ($*eq-pred* $X \circ_c \langle id_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c u$
proof −
have $($*eq-pred* $X \circ_c \langle id_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c u$
$= ($*eq-pred* $X \circ_c \langle id_c\ X, x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y) \circ_c$ *right-coproj* $X\ Y \circ_c g$
using *g-def* **by** *auto*
also have ... $= (\mathrm{f} \circ_c \beta_Y) \circ_c g$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-coproj-cfunc-coprod*)
also have ... = f
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 id-right-unit2 id-type terminal-func-comp terminal-func-unique*)
ultimately show *?thesis*
using *case2* **by** *argo*
qed
qed
qed

lemma *eq-pred-right-coproj*:
assumes *u-type*[*type-rule*]: $u \in_c X \coprod Y$ **and** *y-type*[*type-rule*]: $y \in_c Y$
shows *eq-pred* $(X \coprod Y) \circ_c \langle u,$ *right-coproj* $X\ Y \circ_c y\rangle = ((\mathrm{f} \circ_c \beta_X) \amalg ($*eq-pred* $Y \circ_c \langle id\ Y, y \circ_c \beta_Y\rangle)) \circ_c u$
proof (*cases eq-pred* $(X \coprod Y) \circ_c \langle u,$ *right-coproj* $X\ Y \circ_c y\rangle = \mathrm{t}$)
assume *case1*: *eq-pred* $(X \coprod Y) \circ_c \langle u,$*right-coproj* $X\ Y \circ_c y\rangle = \mathrm{t}$
then have *u-is-right-coproj*: $u =$ *right-coproj* $X\ Y \circ_c y$
using *eq-pred-iff-eq* **by** (*typecheck-cfuncs-prems*, *presburger*)
show *eq-pred* $(X \coprod Y) \circ_c \langle u,$*right-coproj* $X\ Y \circ_c y\rangle = (\mathrm{f} \circ_c \beta_X) \amalg ($*eq-pred* $Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c u$
proof −
have $(\mathrm{f} \circ_c \beta_X) \amalg ($*eq-pred* $Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c u$
$= (\mathrm{f} \circ_c \beta_X) \amalg ($*eq-pred* $Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c$ *right-coproj* $X\ Y \circ_c y$
using *u-is-right-coproj* **by** *auto*
also have ... $= ($*eq-pred* $Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c y$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-coproj-cfunc-coprod*)
also have ... = *eq-pred* $Y \circ_c \langle y,y\rangle$
by (*typecheck-cfuncs*, *smt cart-prod-extract-left comp-associative2*)

also have ... = t
using *eq-pred-iff-eq y-type* **by** *auto*
ultimately show *?thesis*
using *case1* **by** *argo*
qed
next
assume *eq-pred* $(X \coprod Y) \circ_c \langle u, right\text{-}coproj\ X\ Y \circ_c y\rangle \neq$ t
then have *eq-pred-false*: *eq-pred* $(X \coprod Y) \circ_c \langle u, right\text{-}coproj\ X\ Y \circ_c y\rangle =$ f
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have *u-not-right-coproj-y*: $u \neq right\text{-}coproj\ X\ Y \circ_c y$
using *eq-pred-iff-eq-conv* **by** (*typecheck-cfuncs-prems*, *blast*)

show *eq-pred* $(X \coprod Y) \circ_c \langle u, right\text{-}coproj\ X\ Y \circ_c y\rangle = (\mathrm{f} \circ_c \beta_X) \amalg (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c u$
proof (*cases* $\exists\ g.\ g : \mathbf{1} \to Y \wedge u = right\text{-}coproj\ X\ Y \circ_c g$)
assume $\exists g.\ g \in_c Y \wedge u = right\text{-}coproj\ X\ Y \circ_c g$
then obtain g **where** *g-type*[*type-rule*]: $g \in_c Y$ **and** *g-def*: $u = right\text{-}coproj\ X\ Y \circ_c g$
by *auto*
then have *y-not-g*: $y \neq g$
using *u-not-right-coproj-y* **by** *auto*

show *eq-pred* $(X \coprod Y) \circ_c \langle u, right\text{-}coproj\ X\ Y \circ_c y\rangle = (\mathrm{f} \circ_c \beta_X) \amalg (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c u$
proof −
have $(\mathrm{f} \circ_c \beta_X) \amalg (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c right\text{-}coproj\ X\ Y \circ_c g$
$= (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c g$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-coproj-cfunc-coprod*)
also have ... = *eq-pred* $Y \circ_c \langle g, y\rangle$
using *cart-prod-extract-left comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... = f
using *eq-pred-iff-eq-conv y-not-g y-type g-type* **by** *blast*
ultimately show *?thesis*
using *eq-pred-false g-def* **by** *argo*
qed
next
assume $\nexists g.\ g \in_c Y \wedge u = right\text{-}coproj\ X\ Y \circ_c g$
then obtain g **where** *g-type*[*type-rule*]: $g \in_c X$ **and** *g-def*: $u = left\text{-}coproj\ X\ Y \circ_c g$
by (*meson coprojs-jointly-surj u-type*)
show *eq-pred* $(X \coprod Y) \circ_c \langle u, right\text{-}coproj\ X\ Y \circ_c y\rangle = (\mathrm{f} \circ_c \beta_X) \amalg (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c u$
proof −
have $(\mathrm{f} \circ_c \beta_X) \amalg (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c u$
$= (\mathrm{f} \circ_c \beta_X) \amalg (eq\text{-}pred\ Y \circ_c \langle id_c\ Y, y \circ_c \beta_Y\rangle) \circ_c left\text{-}coproj\ X\ Y \circ_c g$
using *g-def* **by** *auto*
also have ... = $(\mathrm{f} \circ_c \beta_X) \circ_c g$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod*)
also have ... = f

by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 id-right-unit2 id-type terminal-func-comp terminal-func-unique*)
ultimately show *?thesis*
using *eq-pred-false* **by** *auto*
qed
qed
qed

9.2 Bowtie Product

definition *cfunc-bowtie-prod* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (**infixr** $\bowtie_f$ *55*) **where**
$f \bowtie_f g$ = ((*left-coproj* (*codomain f*) (*codomain g*)) $\circ_c$ *f*) ⨿ ((*right-coproj* (*codomain f*) (*codomain g*)) $\circ_c$ *g*)

lemma *cfunc-bowtie-prod-def2*:
assumes $f : X \to Y$ $g : V \to W$
shows $f \bowtie_f g$ = (*left-coproj Y W* $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)
using *assms cfunc-bowtie-prod-def cfunc-type-def* **by** *auto*

lemma *cfunc-bowtie-prod-type*[*type-rule*]:
$f : X \to Y \Longrightarrow g : V \to W \Longrightarrow f \bowtie_f g : X \coprod V \to Y \coprod W$
unfolding *cfunc-bowtie-prod-def*
using *cfunc-coprod-type cfunc-type-def comp-type left-proj-type right-proj-type* **by** *auto*

lemma *left-coproj-cfunc-bowtie-prod*:
$f : X \to Y \Longrightarrow g : V \to W \Longrightarrow (f \bowtie_f g) \circ_c$ *left-coproj X V* = *left-coproj Y W* $\circ_c f$
unfolding *cfunc-bowtie-prod-def2*
by (*meson comp-type left-coproj-cfunc-coprod left-proj-type right-proj-type*)

lemma *right-coproj-cfunc-bowtie-prod*:
$f : X \to Y \Longrightarrow g : V \to W \Longrightarrow (f \bowtie_f g) \circ_c$ *right-coproj X V* = *right-coproj Y W* $\circ_c g$
unfolding *cfunc-bowtie-prod-def2*
by (*meson comp-type right-coproj-cfunc-coprod right-proj-type left-proj-type*)

lemma *cfunc-bowtie-prod-unique*: $f : X \to Y \Longrightarrow g : V \to W \Longrightarrow h : X \coprod V \to Y \coprod W \Longrightarrow$
$h \circ_c$ *left-coproj X V* = *left-coproj Y W* $\circ_c f \Longrightarrow$
$h \circ_c$ *right-coproj X V* = *right-coproj Y W* $\circ_c g \Longrightarrow h = f \bowtie_f g$
unfolding *cfunc-bowtie-prod-def*
using *cfunc-coprod-unique cfunc-type-def codomain-comp domain-comp left-proj-type right-proj-type* **by** *auto*

The lemma below is dual to Proposition 2.1.11 in Halvorson.

lemma *identity-distributes-across-composition-dual*:
assumes *f-type*: $f : A \to B$ **and** *g-type*: $g : B \to C$
shows $(g \circ_c f) \bowtie_f id\ X = (g \bowtie_f id\ X) \circ_c (f \bowtie_f id\ X)$
proof –

from *cfunc-bowtie-prod-unique*
have *uniqueness*: $\forall h.\ h : A \coprod X \to C \coprod X \wedge$
$h \circ_c$ *left-coproj* $A\ X = $ *left-coproj* $C\ X \circ_c (g \circ_c f) \wedge$
$h \circ_c$ *right-coproj* $A\ X =$ *right-coproj* $C\ X \circ_c id(X) \longrightarrow$
$h = (g \circ_c f) \bowtie_f id_c\ X$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-bowtie-prod-unique*)

have *left-eq*: $((g \bowtie_f id_c\ X) \circ_c (f \bowtie_f id_c\ X)) \circ_c$ *left-coproj* $A\ X =$ *left-coproj* $C\ X \circ_c (g \circ_c f)$
by (*typecheck-cfuncs*, *smt comp-associative2 left-coproj-cfunc-bowtie-prod left-proj-type assms*)
have *right-eq*: $((g \bowtie_f id_c\ X) \circ_c (f \bowtie_f id_c\ X)) \circ_c$ *right-coproj* $A\ X =$ *right-coproj* $C\ X \circ_c id\ X$
by(*typecheck-cfuncs*, *smt comp-associative2 id-right-unit2 right-coproj-cfunc-bowtie-prod right-proj-type assms*)

show *?thesis*
using *assms left-eq right-eq uniqueness* **by** (*typecheck-cfuncs*, *auto*)
qed

lemma *coproduct-of-beta*:
$\beta_X \amalg \beta_Y = \beta_{X \coprod Y}$
by (*metis* (*full-types*) *cfunc-coprod-unique left-proj-type right-proj-type terminal-func-comp terminal-func-type*)

lemma *cfunc-bowtieprod-comp-cfunc-coprod*:
assumes *a-type*: $a : Y \to Z$ **and** *b-type*: $b : W \to Z$
assumes *f-type*: $f : X \to Y$ **and** *g-type*: $g : V \to W$
shows $(a \amalg b) \circ_c (f \bowtie_f g) = (a \circ_c f) \amalg (b \circ_c g)$
proof −
from *cfunc-bowtie-prod-unique* **have** *uniqueness*:
$\forall h.\ h : X \coprod V \to Z \wedge h \circ_c$ *left-coproj* $X\ V = a \circ_c f \wedge h \circ_c$ *right-coproj* $X\ V = b \circ_c g \longrightarrow$
$h = (a \circ_c f) \amalg (b \circ_c g)$
using *assms comp-type* **by** (*metis* (*full-types*) *cfunc-coprod-unique*)

have *left-eq*: $(a \amalg b \circ_c f \bowtie_f g) \circ_c$ *left-coproj* $X\ V = (a \circ_c f)$
proof −
have $(a \amalg b \circ_c f \bowtie_f g) \circ_c$ *left-coproj* $X\ V = (a \amalg b) \circ_c (f \bowtie_f g) \circ_c$ *left-coproj* $X\ V$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... $= (a \amalg b) \circ_c$ *left-coproj* $Y\ W \circ_c f$
using *f-type g-type left-coproj-cfunc-bowtie-prod* **by** *auto*
also have ... $= ((a \amalg b) \circ_c$ *left-coproj* $Y\ W) \circ_c f$
using *a-type assms*(*2*) *cfunc-type-def comp-associative f-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (a \circ_c f)$
using *a-type b-type left-coproj-cfunc-coprod* **by** *presburger*
finally show $(a \amalg b \circ_c f \bowtie_f g) \circ_c$ *left-coproj* $X\ V = (a \circ_c f)$.

qed

have *right-eq*: $(a \amalg b \circ_c f \bowtie_f g) \circ_c$ *right-coproj* $X\ V = (b \circ_c g)$
proof −
have $(a \amalg b \circ_c f \bowtie_f g) \circ_c$ *right-coproj* $X\ V = (a \amalg b) \circ_c (f \bowtie_f g) \circ_c$ *right-coproj* $X\ V$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = $(a \amalg b) \circ_c$ *right-coproj* $Y\ W \circ_c g$
using *f-type g-type right-coproj-cfunc-bowtie-prod* **by** *auto*
also have ... = $((a \amalg b) \circ_c$ *right-coproj* $Y\ W) \circ_c g$
using *a-type assms*(*2*) *cfunc-type-def comp-associative g-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $(b \circ_c g)$
using *a-type b-type right-coproj-cfunc-coprod* **by** *auto*
finally show $(a \amalg b \circ_c f \bowtie_f g) \circ_c$ *right-coproj* $X\ V = (b \circ_c g)$.
qed

show $(a \amalg b) \circ_c (f \bowtie_f g) = (a \circ_c f) \amalg (b \circ_c g)$
using *uniqueness left-eq right-eq assms*
by (*typecheck-cfuncs*, *auto*)
qed

lemma *id-bowtie-prod*: $id(X) \bowtie_f id(Y) = id(X \coprod Y)$
by (*metis cfunc-bowtie-prod-def id-codomain id-coprod id-right-unit2 left-proj-type right-proj-type*)

lemma *cfunc-bowtie-prod-comp-cfunc-bowtie-prod*:
assumes $f : X \to Y$ $g : V \to W$ $x : Y \to S$ $y : W \to T$
shows $(x \bowtie_f y) \circ_c (f \bowtie_f g) = (x \circ_c f) \bowtie_f (y \circ_c g)$
proof−
have $(x \bowtie_f y) \circ_c ((\textit{left-coproj}\ Y\ W \circ_c f) \amalg (\textit{right-coproj}\ Y\ W \circ_c g))$
$= ((x \bowtie_f y) \circ_c \textit{left-coproj}\ Y\ W \circ_c f) \amalg ((x \bowtie_f y) \circ_c \textit{right-coproj}\ Y\ W \circ_c g)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-coprod-comp*)
also have ... = $(((x \bowtie_f y) \circ_c \textit{left-coproj}\ Y\ W) \circ_c f) \amalg (((x \bowtie_f y) \circ_c \textit{right-coproj}\ Y\ W) \circ_c g)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = $((\textit{left-coproj}\ S\ T \circ_c x) \circ_c f) \amalg ((\textit{right-coproj}\ S\ T \circ_c y) \circ_c g)$
using *assms*(*3*,*4*) *left-coproj-cfunc-bowtie-prod right-coproj-cfunc-bowtie-prod* **by** *auto*
also have ... = $(\textit{left-coproj}\ S\ T \circ_c x \circ_c f) \amalg (\textit{right-coproj}\ S\ T \circ_c y \circ_c g)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = $(x \circ_c f) \bowtie_f (y \circ_c g)$
using *assms cfunc-bowtie-prod-def cfunc-type-def codomain-comp* **by** *auto*
ultimately show $(x \bowtie_f y) \circ_c (f \bowtie_f g) = (x \circ_c f) \bowtie_f (y \circ_c g)$
using *assms*(*1*,*2*) *cfunc-bowtie-prod-def2* **by** *auto*
qed

lemma *cfunc-bowtieprod-epi*:
assumes *f-type*[*type-rule*]: $f : X \to Y$ **and** *g-type*[*type-rule*]: $g : V \to W$

assumes *f-epi*: *epimorphism f* **and** *g-epi*: *epimorphism g*
shows *epimorphism* $(f \bowtie_f g)$
proof (*typecheck-cfuncs*, *unfold epimorphism-def3*, *clarify*)
fix *x y A*
assume *x-type*: x: $Y \coprod W \to A$
assume *y-type*: y: $Y \coprod W \to A$
assume *eqs*: $x \circ_c f \bowtie_f g = y \circ_c f \bowtie_f g$

obtain *x1 x2* **where** *x-expand*: $x = x1 \amalg x2$ **and** *x1-x2-type*: $x1 : Y \to A$ $x2 : W \to A$
using *coprod-decomp x-type* **by** *blast*
obtain *y1 y2* **where** *y-expand*: $y = y1 \amalg y2$ **and** *y1-y2-type*: $y1 : Y \to A$ $y2 : W \to A$
using *coprod-decomp y-type* **by** *blast*

have $(x1 = y1) \wedge (x2 = y2)$
proof
have $x1 \circ_c f = ((x1 \amalg x2) \circ_c$ *left-coproj Y W*$) \circ_c f$
using *x1-x2-type left-coproj-cfunc-coprod* **by** *auto*
also have ... $= (x1 \amalg x2) \circ_c$ *left-coproj Y W* $\circ_c f$
using *assms comp-associative2 x-expand x-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (x1 \amalg x2) \circ_c (f \bowtie_f g) \circ_c$ *left-coproj X V*
using *left-coproj-cfunc-bowtie-prod* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (y1 \amalg y2) \circ_c (f \bowtie_f g) \circ_c$ *left-coproj X V*
using *assms cfunc-type-def comp-associative eqs x-expand x-type y-expand y-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (y1 \amalg y2) \circ_c$ *left-coproj Y W* $\circ_c f$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-bowtie-prod*)
also have ... $= ((y1 \amalg y2) \circ_c$ *left-coproj Y W*$) \circ_c f$
using *assms comp-associative2 y-expand y-type* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= y1 \circ_c f$
using *y1-y2-type left-coproj-cfunc-coprod* **by** *auto*
ultimately show $x1 = y1$
using *epimorphism-def3 f-epi f-type x1-x2-type*(*1*) *y1-y2-type*(*1*) **by** *fastforce*
next
have $x2 \circ_c g = ((x1 \amalg x2) \circ_c$ *right-coproj Y W*$) \circ_c g$
using *x1-x2-type right-coproj-cfunc-coprod* **by** *auto*
also have ... $= (x1 \amalg x2) \circ_c$ *right-coproj Y W* $\circ_c g$
using *assms comp-associative2 x-expand x-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (x1 \amalg x2) \circ_c (f \bowtie_f g) \circ_c$ *right-coproj X V*
using *right-coproj-cfunc-bowtie-prod* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (y1 \amalg y2) \circ_c (f \bowtie_f g) \circ_c$ *right-coproj X V*
using *assms cfunc-type-def comp-associative eqs x-expand x-type y-expand y-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (y1 \amalg y2) \circ_c$ *right-coproj Y W* $\circ_c g$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-bowtie-prod*)
also have ... $= ((y1 \amalg y2) \circ_c$ *right-coproj Y W*$) \circ_c g$
using *assms comp-associative2 y-expand y-type* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= y2 \circ_c g$

using *right-coproj-cfunc-coprod y1-y2-type*(*1*) *y1-y2-type*(*2*) **by** *auto*
ultimately show $x2 = y2$
using *epimorphism-def3 g-epi g-type x1-x2-type*(*2*) *y1-y2-type*(*2*) **by** *fastforce*
qed
then show $x = y$
by (*simp add*: *x-expand y-expand*)
qed

lemma *cfunc-bowtieprod-inj*:
assumes *type-assms*: $f : X \to Y$ $g : V \to W$
assumes *f-epi*: *injective f* **and** *g-epi*: *injective g*
shows *injective* $(f \bowtie_f g)$
unfolding *injective-def*
proof(*clarify*)
fix *z1 z2*
assume *x-type*: $z1 \in_c domain\ (f \bowtie_f g)$
assume *y-type*: $z2 \in_c domain\ (f \bowtie_f g)$
assume *eqs*: $(f \bowtie_f g) \circ_c z1 = (f \bowtie_f g) \circ_c z2$

have *f-bowtie-g-type*: $(f \bowtie_f g) : X \coprod V \to Y \coprod W$
by (*simp add*: *cfunc-bowtie-prod-type type-assms*(*1*) *type-assms*(*2*))

have *x-type2*: $z1 \in_c X \coprod V$
using *cfunc-type-def f-bowtie-g-type x-type* **by** *auto*
have *y-type2*: $z2 \in_c X \coprod V$
using *cfunc-type-def f-bowtie-g-type y-type* **by** *auto*

have *z1-decomp*: $(\exists\ x1.\ (x1 \in_c X \wedge z1 = left\text{-}coproj\ X\ V \circ_c x1))$
$\vee\ (\exists\ y1.\ (y1 \in_c V \wedge z1 = right\text{-}coproj\ X\ V \circ_c y1))$
by (*simp add*: *coprojs-jointly-surj x-type2*)

have *z2-decomp*: $(\exists\ x2.\ (x2 \in_c X \wedge z2 = left\text{-}coproj\ X\ V \circ_c x2))$
$\vee\ (\exists\ y2.\ (y2 \in_c V \wedge z2 = right\text{-}coproj\ X\ V \circ_c y2))$
by (*simp add*: *coprojs-jointly-surj y-type2*)

show $z1 = z2$
proof(*cases* $\exists\ x1.\ x1 \in_c X \wedge z1 = left\text{-}coproj\ X\ V \circ_c x1$)
assume *case1*: $\exists x1.\ x1 \in_c X \wedge z1 = left\text{-}coproj\ X\ V \circ_c x1$
obtain *x1* **where** *x1-def*: $x1 \in_c X \wedge z1 = left\text{-}coproj\ X\ V \circ_c x1$
using *case1* **by** *blast*
show $z1 = z2$
proof(*cases* $\exists\ x2.\ x2 \in_c X \wedge z2 = left\text{-}coproj\ X\ V \circ_c x2$)
assume *caseA*: $\exists x2.\ x2 \in_c X \wedge z2 = left\text{-}coproj\ X\ V \circ_c x2$
show $z1 = z2$
proof –
obtain *x2* **where** *x2-def*: $x2 \in_c X \wedge z2 = left\text{-}coproj\ X\ V \circ_c x2$
using *caseA* **by** *blast*
have $x1 = x2$
proof –

have *left-coproj Y W* $\circ_c$ *f* $\circ_c$ *x1* = (*left-coproj Y W* $\circ_c$ *f*) $\circ_c$ *x1*
using *cfunc-type-def comp-associative left-proj-type type-assms*(*1*) *x1-def*
by *auto*
also have ... =
(((*left-coproj Y W* $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *left-coproj X*
V) $\circ_c$ *x1*
using *cfunc-bowtie-prod-def2 left-coproj-cfunc-bowtie-prod type-assms* **by**
auto
also have ... = ((*left-coproj Y W* $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$
left-coproj X V $\circ_c$ *x1*
using *comp-associative2 type-assms x1-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *z1*
using *cfunc-bowtie-prod-def2 type-assms x1-def* **by** *auto*
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *z2*
by (*meson eqs*)
also have ... = ((*left-coproj Y W* $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$
left-coproj X V $\circ_c$ *x2*
using *cfunc-bowtie-prod-def2 type-assms*(*1*) *type-assms*(*2*) *x2-def* **by** *auto*
also have ... = ((((*left-coproj Y W*) $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$
left-coproj X V) $\circ_c$ *x2*
by (*typecheck-cfuncs*, *meson comp-associative2 type-assms*(*1*) *type-assms*(*2*)
x2-def)
also have ... = (*left-coproj Y W* $\circ_c$ *f*) $\circ_c$ *x2*
using *cfunc-bowtie-prod-def2 left-coproj-cfunc-bowtie-prod type-assms* **by**
auto
also have ... = *left-coproj Y W* $\circ_c$ *f* $\circ_c$ *x2*
by (*metis comp-associative2 left-proj-type type-assms*(*1*) *x2-def*)
ultimately have *f* $\circ_c$ *x1* = *f* $\circ_c$ *x2*
using *cfunc-type-def left-coproj-are-monomorphisms*
left-proj-type monomorphism-def type-assms(*1*) *x1-def x2-def* **by** (*typecheck-cfuncs*,*auto*)
then show *x1* = *x2*
by (*metis cfunc-type-def f-epi injective-def type-assms*(*1*) *x1-def x2-def*)
qed
then show *z1* = *z2*
by (*simp add*: *x1-def x2-def*)
qed
next
assume *caseB*: ∄ *x2*. *x2* $\in_c$ *X* ∧ *z2* = *left-coproj X V* $\circ_c$ *x2*
then obtain *y2* **where** *y2-def*: (*y2* $\in_c$ *V* ∧ *z2* = *right-coproj X V* $\circ_c$ *y2*)
using *z2-decomp* **by** *blast*
have *left-coproj Y W* $\circ_c$ *f* $\circ_c$ *x1* = (*left-coproj Y W* $\circ_c$ *f*) $\circ_c$ *x1*
using *cfunc-type-def comp-associative left-proj-type type-assms*(*1*) *x1-def*
by *auto*
also have ... =
(((*left-coproj Y W* $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *left-coproj X V*)
$\circ_c$ *x1*
using *cfunc-bowtie-prod-def2 left-coproj-cfunc-bowtie-prod type-assms*(*1*)
type-assms(*2*) **by** *auto*
also have ... = ((*left-coproj Y W* $\circ_c$ *f*) ⨿ (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *left-coproj*

X V $\circ_c$ *x1*
using *comp-associative2 type-assms*(*1*,*2*) *x1-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *z1*
using *cfunc-bowtie-prod-def2 type-assms x1-def* **by** *auto*
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *z2*
by (*meson eqs*)
also have ... = ((*left-coproj Y W* $\circ_c$ *f*) II (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *right-coproj X V* $\circ_c$ *y2*
using *cfunc-bowtie-prod-def2 type-assms y2-def* **by** *auto*
also have ... = (((*left-coproj Y W* $\circ_c$ *f*) II (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *right-coproj X V*) $\circ_c$ *y2*
by (*typecheck-cfuncs*, *meson comp-associative2 type-assms y2-def*)
also have ... = (*right-coproj Y W* $\circ_c$ *g*) $\circ_c$ *y2*
using *right-coproj-cfunc-coprod type-assms* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = *right-coproj Y W* $\circ_c$ *g* $\circ_c$ *y2*
using *comp-associative2 type-assms*(*2*) *y2-def* **by** (*typecheck-cfuncs*, *auto*)
ultimately have *False*
using *comp-type coproducts-disjoint type-assms x1-def y2-def* **by** *auto*
then show *z1* = *z2*
by *simp*
qed
next
assume *case2*: $\nexists$*x1*. *x1* $\in_c$ *X* $\wedge$ *z1* = *left-coproj X V* $\circ_c$ *x1*
then obtain *y1* **where** *y1-def*: *y1* $\in_c$ *V* $\wedge$ *z1* = *right-coproj X V* $\circ_c$ *y1*
using *z1-decomp* **by** *blast*
show *z1* = *z2*
proof(*cases* $\exists$ *x2*. *x2* $\in_c$ *X* $\wedge$ *z2* = *left-coproj X V* $\circ_c$ *x2*)
assume *caseA*: $\exists$*x2*. *x2* $\in_c$ *X* $\wedge$ *z2* = *left-coproj X V* $\circ_c$ *x2*
show *z1* = *z2*
proof −
obtain *x2* **where** *x2-def*: *x2* $\in_c$ *X* $\wedge$ *z2* = *left-coproj X V* $\circ_c$ *x2*
using *caseA* **by** *blast*
have *left-coproj Y W* $\circ_c$ *f* $\circ_c$ *x2* = (*left-coproj Y W* $\circ_c$ *f*) $\circ_c$ *x2*
using *comp-associative2 type-assms*(*1*) *x2-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... =
(((*left-coproj Y W* $\circ_c$ *f*) II (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *left-coproj X V*) $\circ_c$ *x2*
using *cfunc-bowtie-prod-def2 left-coproj-cfunc-bowtie-prod type-assms* **by** *auto*
also have ... = ((*left-coproj Y W* $\circ_c$ *f*) II (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *left-coproj X V* $\circ_c$ *x2*
using *comp-associative2 type-assms x2-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *z2*
using *cfunc-bowtie-prod-def2 type-assms x2-def* **by** *auto*
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *z1*
by (*simp add*: *eqs*)
also have ... = ((*left-coproj Y W* $\circ_c$ *f*) II (*right-coproj Y W* $\circ_c$ *g*)) $\circ_c$ *right-coproj X V* $\circ_c$ *y1*

```
        using cfunc-bowtie-prod-def2 type-assms y1-def by auto
        also have ... = (((left-coproj Y W ∘c f) ⨿ (right-coproj Y W ∘c g)) ∘c
right-coproj X V) ∘c y1
        by (typecheck-cfuncs, meson comp-associative2 type-assms y1-def)
      also have ... = (right-coproj Y W ∘c g) ∘c y1
       using right-coproj-cfunc-coprod type-assms by (typecheck-cfuncs, fastforce)
      also have ... = right-coproj Y W ∘c g ∘c y1
        using comp-associative2 type-assms(2) y1-def by (typecheck-cfuncs, auto)
      ultimately have False
        using comp-type coproducts-disjoint type-assms x2-def y1-def by auto
      then show z1 = z2
        by simp
    qed
  next
    assume caseB: ∄x2. x2 ∈c X ∧ z2 = left-coproj X V ∘c x2
    then obtain y2 where y2-def: (y2 ∈c V ∧ z2 = right-coproj X V ∘c y2)
      using z2-decomp by blast
      have y1 = y2
      proof −
        have right-coproj Y W ∘c g ∘c y1 = (right-coproj Y W ∘c g) ∘c y1
        using comp-associative2 type-assms(2) y1-def by (typecheck-cfuncs, auto)
        also have ... =
             (((left-coproj Y W ∘c f) ⨿ (right-coproj Y W ∘c g)) ∘c right-coproj X
V) ∘c y1
        using right-coproj-cfunc-coprod type-assms by (typecheck-cfuncs, fastforce)
          also have ... = ((left-coproj Y W ∘c f) ⨿ (right-coproj Y W ∘c g)) ∘c
right-coproj X V ∘c y1
       using comp-associative2 type-assms y1-def by (typecheck-cfuncs, fastforce)
        also have ... = (f ⋈f g) ∘c z1
          using cfunc-bowtie-prod-def2 type-assms y1-def by auto
        also have ... = (f ⋈f g) ∘c z2
          by (meson eqs)
          also have ... = ((left-coproj Y W ∘c f) ⨿ (right-coproj Y W ∘c g)) ∘c
right-coproj X V ∘c y2
          using cfunc-bowtie-prod-def2 type-assms y2-def by auto
         also have ... = (((left-coproj Y W ∘c f) ⨿ (right-coproj Y W ∘c g)) ∘c
right-coproj X V) ∘c y2
          by (typecheck-cfuncs, meson comp-associative2 type-assms y2-def)
        also have ... = (right-coproj Y W ∘c g) ∘c y2
        using right-coproj-cfunc-coprod type-assms by (typecheck-cfuncs, fastforce)
        also have ... = right-coproj Y W ∘c g ∘c y2
        using comp-associative2 type-assms(2) y2-def by (typecheck-cfuncs, auto)
        ultimately have g ∘c y1 = g ∘c y2
          using cfunc-type-def right-coproj-are-monomorphisms
               right-proj-type monomorphism-def type-assms(2) y1-def y2-def by
(typecheck-cfuncs,auto)
        then show y1 = y2
          by (metis cfunc-type-def g-epi injective-def type-assms(2) y1-def y2-def)
      qed
```

then show $z1 = z2$
  by (*simp add*: *y1-def y2-def*)
qed
qed
qed

lemma *cfunc-bowtieprod-inj-converse*:
  assumes *type-assms*: $f : X \to Y$ $g : Z \to W$
  assumes *inj-f-bowtie-g*: *injective* $(f \bowtie_f g)$
  shows *injective* $f \land$ *injective* g
  unfolding *injective-def*
proof(*safe*)
  fix x y
  assume *x-type*: $x \in_c$ *domain* f
  assume *y-type*: $y \in_c$ *domain* f
  assume *eqs*: $f \circ_c x = f \circ_c y$

  have *x-type2*: $x \in_c X$
    using *cfunc-type-def type-assms*(*1*) *x-type* **by** *auto*
  have *y-type2*: $y \in_c X$
    using *cfunc-type-def type-assms*(*1*) *y-type* **by** *auto*
  have *fg-bowtie-tyepe*: $(f \bowtie_f g) : X \coprod Z \to Y \coprod W$
    using *assms* **by** *typecheck-cfuncs*
  have *lift*: $(f \bowtie_f g) \circ_c$ *left-coproj* X Z $\circ_c x = (f \bowtie_f g) \circ_c$ *left-coproj* X Z $\circ_c y$
  proof −
    have $(f \bowtie_f g) \circ_c$ *left-coproj* X Z $\circ_c x = ((f \bowtie_f g) \circ_c$ *left-coproj* X $Z) \circ_c x$
      using *x-type2 comp-associative2 fg-bowtie-tyepe* **by** (*typecheck-cfuncs*, *auto*)
    also have ... = (*left-coproj* Y W $\circ_c f) \circ_c x$
      using *left-coproj-cfunc-bowtie-prod type-assms* **by** *auto*
    also have ... = *left-coproj* Y W $\circ_c f \circ_c x$
      using *x-type2 comp-associative2 type-assms*(*1*) **by** (*typecheck-cfuncs*, *auto*)
    also have ... = *left-coproj* Y W $\circ_c f \circ_c y$
      by (*simp add*: *eqs*)
    also have ... = (*left-coproj* Y W $\circ_c f) \circ_c y$
      using *y-type2 comp-associative2 type-assms*(*1*) **by** (*typecheck-cfuncs*, *auto*)
    also have ... = $((f \bowtie_f g) \circ_c$ *left-coproj* X $Z) \circ_c y$
      using *left-coproj-cfunc-bowtie-prod type-assms*(*1*) *type-assms*(*2*) **by** *auto*
    also have ... = $(f \bowtie_f g) \circ_c$ *left-coproj* X Z $\circ_c y$
      using *y-type2 comp-associative2 fg-bowtie-tyepe* **by** (*typecheck-cfuncs*, *auto*)
    finally show *?thesis*.
  qed
  then have *monomorphism* $(f \bowtie_f g)$
    using *inj-f-bowtie-g injective-imp-monomorphism* **by** *auto*
  then have *left-coproj* X Z $\circ_c x =$ *left-coproj* X Z $\circ_c y$
    by (*typecheck-cfuncs*, *metis cfunc-type-def fg-bowtie-tyepe inj-f-bowtie-g injective-def lift x-type2 y-type2*)
  then show $x = y$
    using *x-type2 y-type2 cfunc-type-def left-coproj-are-monomorphisms left-proj-type monomorphism-def* **by** *auto*

next
  fix *x y*
  assume *x-type*: $x \in_c$ *domain g*
  assume *y-type*: $y \in_c$ *domain g*
  assume *eqs*: $g \circ_c x = g \circ_c y$

  have *x-type2*: $x \in_c Z$
    using *cfunc-type-def type-assms*(*2*) *x-type* **by** *auto*
  have *y-type2*: $y \in_c Z$
    using *cfunc-type-def type-assms*(*2*) *y-type* **by** *auto*
  have *fg-bowtie-tyepe*: $f \bowtie_f g : X \coprod Z \to Y \coprod W$
    using *assms* **by** *typecheck-cfuncs*
  have *lift*: $(f \bowtie_f g) \circ_c$ *right-coproj X Z* $\circ_c x = (f \bowtie_f g) \circ_c$ *right-coproj X Z* $\circ_c y$
  proof −
    have $(f \bowtie_f g) \circ_c$ *right-coproj X Z* $\circ_c x = ((f \bowtie_f g) \circ_c$ *right-coproj X Z*$) \circ_c x$
      using *x-type2 comp-associative2 fg-bowtie-tyepe* **by** (*typecheck-cfuncs*, *auto*)
    also have ... = (*right-coproj Y W* $\circ_c g) \circ_c x$
      using *right-coproj-cfunc-bowtie-prod type-assms* **by** *auto*
    also have ... = *right-coproj Y W* $\circ_c g \circ_c x$
      using *x-type2 comp-associative2 type-assms*(*2*) **by** (*typecheck-cfuncs*, *auto*)
    also have ... = *right-coproj Y W* $\circ_c g \circ_c y$
      by (*simp add*: *eqs*)
    also have ... = (*right-coproj Y W* $\circ_c g) \circ_c y$
      using *y-type2 comp-associative2 type-assms*(*2*) **by** (*typecheck-cfuncs*, *auto*)
    also have ... = $((f \bowtie_f g) \circ_c$ *right-coproj X Z*$) \circ_c y$
      using *right-coproj-cfunc-bowtie-prod type-assms*(*1*) *type-assms*(*2*) **by** *auto*
    also have ... = $(f \bowtie_f g) \circ_c$ *right-coproj X Z* $\circ_c y$
      using *y-type2 comp-associative2 fg-bowtie-tyepe* **by** (*typecheck-cfuncs*, *auto*)
    finally show *?thesis***.**
  qed
  then have *monomorphism* $(f \bowtie_f g)$
    using *inj-f-bowtie-g injective-imp-monomorphism* **by** *auto*
  then have *right-coproj X Z* $\circ_c x =$ *right-coproj X Z* $\circ_c y$
    by (*typecheck-cfuncs*, *metis cfunc-type-def fg-bowtie-tyepe inj-f-bowtie-g injective-def lift x-type2 y-type2*)
  then show $x = y$
    using *x-type2 y-type2 cfunc-type-def right-coproj-are-monomorphisms right-proj-type monomorphism-def* **by** *auto*
qed

lemma *cfunc-bowtieprod-iso*:
  assumes *type-assms*: $f : X \to Y$ $g : V \to W$
  assumes *f-iso*: *isomorphism f* **and** *g-iso*: *isomorphism g*
  shows *isomorphism* $(f \bowtie_f g)$
  by (*typecheck-cfuncs*, *meson cfunc-bowtieprod-epi cfunc-bowtieprod-inj epi-mon-is-iso f-iso g-iso injective-imp-monomorphism iso-imp-epi-and-monic monomorphism-imp-injective singletonI assms*)

lemma *cfunc-bowtieprod-surj-converse*:

assumes *type-assms*: $f : X \to Y$ $g : Z \to W$
assumes *inj-f-bowtie-g*: *surjective* $(f \bowtie_f g)$
shows *surjective* $f \wedge$ *surjective* g
unfolding *surjective-def*
proof(*safe*)
fix y
assume *y-type*: $y \in_c$ *codomain* f
then have *y-type2*: $y \in_c Y$
using *cfunc-type-def type-assms*(*1*) **by** *auto*
then have *coproj-y-type*: *left-coproj* Y $W \circ_c y \in_c Y \coprod W$
by *typecheck-cfuncs*
have *fg-type*: $(f \bowtie_f g) : X \coprod Z \to Y \coprod W$
using *assms* **by** *typecheck-cfuncs*
obtain *xz* **where** *xz-def*: $xz \in_c X \coprod Z \wedge (f \bowtie_f g) \circ_c xz =$ *left-coproj* Y $W \circ_c$
y
using *fg-type y-type2 cfunc-type-def inj-f-bowtie-g surjective-def* **by** (*typecheck-cfuncs*, *auto*)
then have *xz-form*: $(\exists\ x.\ x \in_c X \wedge$ *left-coproj* X $Z \circ_c x = xz) \vee$
$(\exists\ z.\ z \in_c Z \wedge$ *right-coproj* X $Z \circ_c z = xz)$
using *coprojs-jointly-surj xz-def* **by** (*typecheck-cfuncs*, *blast*)
show $\exists\ x.\ x \in_c$ *domain* $f \wedge f \circ_c x = y$
proof(*cases* $\exists\ x.\ x \in_c X \wedge$ *left-coproj* X $Z \circ_c x = xz$)
assume $\exists\ x.\ x \in_c X \wedge$ *left-coproj* X $Z \circ_c x = xz$
then obtain x **where** *x-def*: $x \in_c X \wedge$ *left-coproj* X $Z \circ_c x = xz$
by *blast*
have $f \circ_c x = y$
proof –
have *left-coproj* Y $W \circ_c y = (f \bowtie_f g) \circ_c xz$
by (*simp add*: *xz-def*)
also have ... $= (f \bowtie_f g) \circ_c$ *left-coproj* X $Z \circ_c x$
by (*simp add*: *x-def*)
also have ... $= ((f \bowtie_f g) \circ_c$ *left-coproj* X $Z) \circ_c x$
using *comp-associative2 fg-type x-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... $=$ (*left-coproj* Y $W \circ_c f) \circ_c x$
using *left-coproj-cfunc-bowtie-prod type-assms* **by** *auto*
also have ... $=$ *left-coproj* Y $W \circ_c f \circ_c x$
using *comp-associative2 type-assms*(*1*) *x-def* **by** (*typecheck-cfuncs*, *auto*)
ultimately show $f \circ_c x = y$
using *type-assms*(*1*) *x-def y-type2*
by (*typecheck-cfuncs*, *metis cfunc-type-def left-coproj-are-monomorphisms left-proj-type monomorphism-def x-def*)
qed
then show *?thesis*
using *cfunc-type-def type-assms*(*1*) *x-def* **by** *auto*
next
assume $\nexists x.\ x \in_c X \wedge$ *left-coproj* X $Z \circ_c x = xz$
then obtain z **where** *z-def*: $z \in_c Z \wedge$ *right-coproj* X $Z \circ_c z = xz$
using *xz-form* **by** *blast*
have *False*

proof −
have *left-coproj Y W* $\circ_c$ *y* = (*f* $\bowtie_f$ *g*) $\circ_c$ *xz*
by (*simp add*: *xz-def*)
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *right-coproj X Z* $\circ_c$ *z*
by (*simp add*: *z-def*)
also have ... = ((*f* $\bowtie_f$ *g*) $\circ_c$ *right-coproj X Z*) $\circ_c$ *z*
using *comp-associative2 fg-type z-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*right-coproj Y W* $\circ_c$ *g*) $\circ_c$ *z*
using *right-coproj-cfunc-bowtie-prod type-assms* **by** *auto*
also have ... = *right-coproj Y W* $\circ_c$ *g* $\circ_c$ *z*
using *comp-associative2 type-assms*(*2*) *z-def* **by** (*typecheck-cfuncs*, *auto*)
ultimately show *False*
using *comp-type coproducts-disjoint type-assms*(*2*) *y-type2 z-def* **by** *auto*
qed
then show *?thesis*
by *simp*
qed
next
fix *y*
assume *y-type*: *y* $\in_c$ *codomain g*
then have *y-type2*: *y* $\in_c$ *W*
using *cfunc-type-def type-assms*(*2*) **by** *auto*
then have *coproj-y-type*: (*right-coproj Y W*) $\circ_c$ *y* $\in_c$ (*Y* $\coprod$ *W*)
using *cfunc-type-def comp-type right-proj-type type-assms*(*2*) **by** *auto*
have *fg-type*: (*f* $\bowtie_f$ *g*) : *X* $\coprod$ *Z* → *Y* $\coprod$ *W*
by (*simp add*: *cfunc-bowtie-prod-type type-assms*)
obtain *xz* **where** *xz-def*: *xz* $\in_c$ *X* $\coprod$ *Z* ∧ (*f* $\bowtie_f$ *g*) $\circ_c$ *xz* = *right-coproj Y W* $\circ_c$
y
using *fg-type y-type2 cfunc-type-def inj-f-bowtie-g surjective-def* **by** (*typecheck-cfuncs*,
auto)
then have *xz-form*: (∃ *x*. *x* $\in_c$ *X* ∧ *left-coproj X Z* $\circ_c$ *x* = *xz*) ∨
(∃ *z*. *z* $\in_c$ *Z* ∧ *right-coproj X Z* $\circ_c$ *z* = *xz*)
using *coprojs-jointly-surj xz-def* **by** (*typecheck-cfuncs*, *blast*)
show ∃ *x*. *x* $\in_c$ *domain g* ∧ *g* $\circ_c$ *x* = *y*
proof(*cases* ∃ *x*. *x* $\in_c$ *X* ∧ *left-coproj X Z* $\circ_c$ *x* = *xz*)
assume ∃ *x*. *x* $\in_c$ *X* ∧ *left-coproj X Z* $\circ_c$ *x* = *xz*
then obtain *x* **where** *x-def*: *x* $\in_c$ *X* ∧ *left-coproj X Z* $\circ_c$ *x* = *xz*
by *blast*
have *False*
proof −
have *right-coproj Y W* $\circ_c$ *y* = (*f* $\bowtie_f$ *g*) $\circ_c$ *xz*
by (*simp add*: *xz-def*)
also have ... = (*f* $\bowtie_f$ *g*) $\circ_c$ *left-coproj X Z* $\circ_c$ *x*
by (*simp add*: *x-def*)
also have ... = ((*f* $\bowtie_f$ *g*) $\circ_c$ *left-coproj X Z*) $\circ_c$ *x*
using *comp-associative2 fg-type x-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*left-coproj Y W* $\circ_c$ *f*) $\circ_c$ *x*
using *left-coproj-cfunc-bowtie-prod type-assms* **by** *auto*
also have ... = *left-coproj Y W* $\circ_c$ *f* $\circ_c$ *x*

```
        using comp-associative2 type-assms(1) x-def by (typecheck-cfuncs, auto)
      ultimately show False
        by (metis comp-type coproducts-disjoint type-assms(1) x-def y-type2)
    qed
    then show ?thesis
      by simp
next
  assume ∄x. x ∈c X ∧ left-coproj X Z ∘c x = xz
  then obtain z where z-def: z ∈c Z ∧ right-coproj X Z ∘c z = xz
    using xz-form by blast
  have g ∘c z = y
    proof −
      have right-coproj Y W ∘c y = (f ⋈f g) ∘c xz
        by (simp add: xz-def)
      also have ... = (f ⋈f g) ∘c right-coproj X Z ∘c z
        by (simp add: z-def)
      also have ... = ((f ⋈f g) ∘c right-coproj X Z) ∘c z
        using comp-associative2 fg-type z-def by (typecheck-cfuncs, auto)
      also have ... = (right-coproj Y W ∘c g) ∘c z
        using right-coproj-cfunc-bowtie-prod type-assms by auto
      also have ... = right-coproj Y W ∘c g ∘c z
        using comp-associative2 type-assms(2) z-def by (typecheck-cfuncs, auto)
      ultimately show ?thesis
        by (metis cfunc-type-def codomain-comp monomorphism-def
              right-coproj-are-monomorphisms right-proj-type type-assms(2) y-type2
z-def)
    qed
    then show ?thesis
      using cfunc-type-def type-assms(2) z-def by auto
 qed
qed
```

9.3 Boolean Cases

```
definition case-bool :: cfunc where
  case-bool = (THE f. f : Ω → (1 ∐ 1) ∧
    (t ⨿ f) ∘c f = id Ω ∧ f ∘c (t ⨿ f) = id (1 ∐ 1))

lemma case-bool-def2:
  case-bool : Ω → (1 ∐ 1) ∧
    (t ⨿ f) ∘c case-bool = id Ω ∧ case-bool ∘c (t ⨿ f) = id (1 ∐ 1)
  unfolding case-bool-def
proof (rule theI', safe)
  show ∃x. x : Ω → 1 ∐ 1 ∧ t ⨿ f ∘c x = idc Ω ∧ x ∘c t ⨿ f = idc (1 ∐ 1)
    unfolding isomorphism-def
    using isomorphism-def3 truth-value-set-iso-1u1 by (typecheck-cfuncs, blast)
next
  fix x y
  assume x-type[type-rule]: x : Ω → 1 ∐ 1 and y-type[type-rule]: y : Ω → 1 ∐ 1
```

assume *x-left-inv*: t II f $\circ_c$ x = id_c Ω
assume x $\circ_c$ t II f = id_c (**1** ∐ **1**) y $\circ_c$ t II f = id_c (**1** ∐ **1**)
then have x $\circ_c$ t II f = y $\circ_c$ t II f
by *auto*
then have x $\circ_c$ t II f $\circ_c$ x = y $\circ_c$ t II f $\circ_c$ x
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
then show $x = y$
using *id-right-unit2 x-left-inv* **by** (*typecheck-cfuncs-prems*, *auto*)
qed

lemma *case-bool-type*[*type-rule*]:
case-bool : Ω → **1** ∐ **1**
using *case-bool-def2* **by** *auto*

lemma *case-bool-true-coprod-false*:
case-bool $\circ_c$ (t II f) = *id* (**1** ∐ **1**)
using *case-bool-def2* **by** *auto*

lemma *true-coprod-false-case-bool*:
(t II f) $\circ_c$ *case-bool* = *id* Ω
using *case-bool-def2* **by** *auto*

lemma *case-bool-iso*:
isomorphism case-bool
using *case-bool-def2* **unfolding** *isomorphism-def*
by (*intro exI*[**where** x=t II f], *typecheck-cfuncs*, *auto simp add*: *cfunc-type-def*)

lemma *case-bool-true-and-false*:
(*case-bool* $\circ_c$ t = *left-coproj* **1 1**) ∧ (*case-bool* $\circ_c$ f = *right-coproj* **1 1**)
proof −
have (*left-coproj* **1 1**) II (*right-coproj* **1 1**) = *id*(**1** ∐ **1**)
by (*simp add*: *id-coprod*)
also have ... = *case-bool* $\circ_c$ (t II f)
by (*simp add*: *case-bool-def2*)
also have ... = (*case-bool* $\circ_c$ t) II (*case-bool* $\circ_c$ f)
using *case-bool-def2 cfunc-coprod-comp false-func-type true-func-type* **by** *auto*
ultimately show *?thesis*
using *coprod-eq2* **by** (*typecheck-cfuncs*, *auto*)
qed

lemma *case-bool-true*:
case-bool $\circ_c$ t = *left-coproj* **1 1**
by (*simp add*: *case-bool-true-and-false*)

lemma *case-bool-false*:
case-bool $\circ_c$ f = *right-coproj* **1 1**
by (*simp add*: *case-bool-true-and-false*)

lemma *coprod-case-bool-true*:

assumes *x1* $\in_c$ *X*
assumes *x2* $\in_c$ *X*
shows (*x1* II *x2* $\circ_c$ *case-bool*) $\circ_c$ t = *x1*
proof −
have (*x1* II *x2* $\circ_c$ *case-bool*) $\circ_c$ t = (*x1* II *x2*) $\circ_c$ *case-bool* $\circ_c$ t
using *assms* **by** (*typecheck-cfuncs* , *simp add*: *comp-associative2*)
also have ... = (*x1* II *x2*) $\circ_c$ *left-coproj* **1 1**
using *assms case-bool-true* **by** *presburger*
also have ... = *x1*
using *assms left-coproj-cfunc-coprod* **by** *force*
finally show *?thesis***.**
qed

lemma *coprod-case-bool-false*:
assumes *x1* $\in_c$ *X*
assumes *x2* $\in_c$ *X*
shows (*x1* II *x2* $\circ_c$ *case-bool*) $\circ_c$ f = *x2*
proof −
have (*x1* II *x2* $\circ_c$ *case-bool*) $\circ_c$ f = (*x1* II *x2*) $\circ_c$ *case-bool* $\circ_c$ f
using *assms* **by** (*typecheck-cfuncs* , *simp add*: *comp-associative2*)
also have ... = (*x1* II *x2*) $\circ_c$ *right-coproj* **1 1**
using *assms case-bool-false* **by** *presburger*
also have ... = *x2*
using *assms right-coproj-cfunc-coprod* **by** *force*
finally show *?thesis***.**
qed

9.4 Distribution of Products over Coproducts

9.4.1 Factor Product over Coproduct on Left

definition *factor-prod-coprod-left* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
factor-prod-coprod-left A B C = (*id A* $\times_f$ *left-coproj B C*) II (*id A* $\times_f$ *right-coproj B C*)

lemma *factor-prod-coprod-left-type*[*type-rule*]:
factor-prod-coprod-left A B C : (*A* $\times_c$ *B*) ∐ (*A* $\times_c$ *C*) → *A* $\times_c$ (*B* ∐ *C*)
unfolding *factor-prod-coprod-left-def* **by** *typecheck-cfuncs*

lemma *factor-prod-coprod-left-ap-left*:
assumes *a* $\in_c$ *A* *b* $\in_c$ *B*
shows *factor-prod-coprod-left A B C* $\circ_c$ *left-coproj* (*A* $\times_c$ *B*) (*A* $\times_c$ *C*) $\circ_c$ ⟨*a*, *b*⟩ = ⟨*a*, *left-coproj B C* $\circ_c$ *b*⟩
unfolding *factor-prod-coprod-left-def* **using** *assms*
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod comp-associative2 id-left-unit2 left-coproj-cfunc-coprod*)

lemma *factor-prod-coprod-left-ap-right*:
assumes *a* $\in_c$ *A* *c* $\in_c$ *C*
shows *factor-prod-coprod-left A B C* $\circ_c$ *right-coproj* (*A* $\times_c$ *B*) (*A* $\times_c$ *C*) $\circ_c$ ⟨*a*,

$c\rangle = \langle a,$ *right-coproj* $B\ C \circ_c c\rangle$
unfolding *factor-prod-coprod-left-def* **using** *assms*
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod comp-associative2 id-left-unit2 right-coproj-cfunc-coprod*)

lemma *factor-prod-coprod-left-mono*:
monomorphism (*factor-prod-coprod-left* $A\ B\ C$)
proof −
obtain φ **where** φ-*def*: $\varphi = (id\ A\ \times_f$ *left-coproj* $B\ C) \amalg (id\ A \times_f$ *right-coproj* $B\ C)$ **and**
φ-*type*[*type-rule*]: $\varphi : (A \times_c B) \coprod (A \times_c C) \to A \times_c (B \coprod C)$
by (*typecheck-cfuncs*, *simp*)

have *injective*: *injective*(φ)
unfolding *injective-def*
proof(*clarify*)
fix $x\ y$
assume x-*type*: $x \in_c$ *domain* φ
assume y-*type*: $y \in_c$ *domain* φ
assume *equal*: $\varphi \circ_c x = \varphi \circ_c y$

have x-*type*[*type-rule*]: $x \in_c (A \times_c B) \coprod (A \times_c C)$
using *cfunc-type-def* φ-*type* x-*type* **by** *auto*
then have x-*form*: $(\exists\ x'.\ x' \in_c A \times_c B \land x = ($*left-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c x')$
$\lor\ (\exists\ x'.\ x' \in_c A \times_c C \land x = ($*right-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c x')$
by (*simp add*: *coprojs-jointly-surj*)
have y-*type*[*type-rule*]: $y \in_c (A \times_c B) \coprod (A \times_c C)$
using *cfunc-type-def* φ-*type* y-*type* **by** *auto*
then have y-*form*: $(\exists\ y'.\ y' \in_c A \times_c B \land y = ($*left-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c y')$
$\lor\ (\exists\ y'.\ y' \in_c A \times_c C \land y = ($*right-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c y')$
by (*simp add*: *coprojs-jointly-surj*)

show $x = y$
proof(*cases* $(\exists\ x'.\ x' \in_c A \times_c B \land x = ($*left-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c x'))$
assume $\exists\ x'.\ x' \in_c A \times_c B \land x = ($*left-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c x'$
then obtain x' **where** x'-*def*[*type-rule*]: $x' \in_c A \times_c B\ x =$ *left-coproj* $(A \times_c B)\ (A \times_c C) \circ_c x'$
by *blast*
then have *ab-exists*: $\exists\ a\ b.\ a \in_c A \land b \in_c B \land x' = \langle a,b\rangle$
using *cart-prod-decomp* **by** *blast*
then obtain $a\ b$ **where** *ab-def*[*type-rule*]: $a \in_c A\ b \in_c B\ \ x' = \langle a,b\rangle$
by *blast*
show $x = y$
proof(*cases* $\exists\ y'.\ y' \in_c A \times_c B \land y = ($*left-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c y')$
assume $\exists\ y'.\ y' \in_c A \times_c B \land y = ($*left-coproj* $(A \times_c B)\ (A \times_c C)) \circ_c y'$

then obtain y' **where** y'*-def*: $y' \in_c A \times_c B$ $y = $ *left-coproj* $(A \times_c B)$ $(A \times_c C) \circ_c y'$
by *blast*
then have *ab-exists*: $\exists\ a'\ b'.\ a' \in_c A \wedge b' \in_c B \wedge y' = \langle a', b' \rangle$
using *cart-prod-decomp* **by** *blast*
then obtain $a'\ b'$ **where** $a'b'$*-def*[*type-rule*]: $a' \in_c A$ $b' \in_c B$ $y' = \langle a', b' \rangle$
by *blast*
have *equal-pair*: $\langle a,$ *left-coproj* $B\ C \circ_c b \rangle = \langle a',$ *left-coproj* $B\ C \circ_c b' \rangle$
proof −
have $\langle a,$ *left-coproj* $B\ C \circ_c b \rangle = \langle id\ A \circ_c a,$ *left-coproj* $B\ C \circ_c b \rangle$
using *ab-def id-left-unit2* **by** *force*
also have ... = $(id\ A \times_f$ *left-coproj* $B\ C) \circ_c \langle a, b \rangle$
by (*smt ab-def cfunc-cross-prod-comp-cfunc-prod id-type left-proj-type*)
also have ... = $(\varphi \circ_c$ *left-coproj* $(A \times_c B)$ $(A \times_c C)) \circ_c \langle a, b \rangle$
unfolding φ*-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\varphi \circ_c x$
using *ab-def comp-associative2* x'*-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = $\varphi \circ_c y$
by (*simp add*: *local.equal*)
also have ... = $(\varphi \circ_c$ *left-coproj* $(A \times_c B)$ $(A \times_c C)) \circ_c \langle a', b' \rangle$
using $a'b'$*-def comp-associative2* φ*-type* y'*-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $(id\ A \times_f$ *left-coproj* $B\ C) \circ_c \langle a', b' \rangle$
unfolding φ*-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle id\ A \circ_c a',$ *left-coproj* $B\ C \circ_c b' \rangle$
using $a'b'$*-def cfunc-cross-prod-comp-cfunc-prod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle a',$ *left-coproj* $B\ C \circ_c b' \rangle$
using $a'b'$*-def id-left-unit2* **by** *force*
finally show $\langle a,$ *left-coproj* $B\ C \circ_c b \rangle = \langle a',$ *left-coproj* $B\ C \circ_c b' \rangle$.
qed
then have *a-equal*: $a = a' \wedge$ *left-coproj* $B\ C \circ_c b = $ *left-coproj* $B\ C \circ_c b'$
using $a'b'$*-def ab-def cart-prod-eq2 equal-pair* **by** (*typecheck-cfuncs*, *blast*)
then have *b-equal*: $b = b'$
using $a'b'$*-def a-equal ab-def left-coproj-are-monomorphisms left-proj-type monomorphism-def3* **by** *blast*
then show $x = y$
by (*simp add*: $a'b'$*-def a-equal ab-def* x'*-def* y'*-def*)
next
assume $\nexists y'.\ y' \in_c A \times_c B \wedge y = $ *left-coproj* $(A \times_c B)$ $(A \times_c C) \circ_c y'$
then obtain y' **where** y'*-def*: $y' \in_c A \times_c C$ $y = $ *right-coproj* $(A \times_c B)$ $(A \times_c C) \circ_c y'$
using *y-form* **by** *blast*
then obtain $a'\ c'$ **where** $a'c'$*-def*: $a' \in_c A$ $c' \in_c C$ $y' = \langle a', c' \rangle$
by (*meson cart-prod-decomp*)
have *equal-pair*: $\langle a,$ (*left-coproj* $B\ C) \circ_c b \rangle = \langle a',$ *right-coproj* $B\ C \circ_c c' \rangle$
proof −

have ⟨a, *left-coproj* B C $\circ_c$ b⟩ = ⟨*id* A $\circ_c$ a, *left-coproj* B C $\circ_c$ b⟩
using *ab-def id-left-unit2* **by** *force*
also have ... = (*id* A $\times_f$ *left-coproj* B C) $\circ_c$ ⟨a, b⟩
by (*smt ab-def cfunc-cross-prod-comp-cfunc-prod id-type left-proj-type*)
also have ... = (φ $\circ_c$ *left-coproj* (A $\times_c$ B) (A $\times_c$ C)) $\circ_c$ ⟨a, b⟩
unfolding φ*-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = φ $\circ_c$ x
using *ab-def comp-associative2* φ*-type* x'*-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = φ $\circ_c$ y
by (*simp add*: *local.equal*)
also have ... = (φ $\circ_c$ *right-coproj* (A $\times_c$ B) (A $\times_c$ C)) $\circ_c$ ⟨a', c'⟩
using $a'c'$*-def comp-associative2* y'*-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = (*id* A $\times_f$ *right-coproj* B C) $\circ_c$ ⟨a', c'⟩
unfolding φ*-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = ⟨*id* A $\circ_c$ a', *right-coproj* B C $\circ_c$ c'⟩
using $a'c'$*-def cfunc-cross-prod-comp-cfunc-prod* **by** (*typecheck-cfuncs*,*auto*)
also have ... = ⟨a', *right-coproj* B C $\circ_c$ c'⟩
using $a'c'$*-def id-left-unit2* **by** *force*
finally show ⟨a, *left-coproj* B C $\circ_c$ b⟩ = ⟨a', *right-coproj* B C $\circ_c$ c'⟩**.**
qed
then have *impossible*: *left-coproj* B C $\circ_c$ b = *right-coproj* B C $\circ_c$ c'
using $a'c'$*-def ab-def element-pair-eq equal-pair* **by** (*typecheck-cfuncs*, *blast*)
then show $x = y$
using $a'c'$*-def ab-def coproducts-disjoint* **by** *blast*
qed
next
assume $\nexists x'$. $x' \in_c A \times_c B \wedge x =$ *left-coproj* ($A \times_c B$) ($A \times_c C$) $\circ_c x'$
then obtain x' **where** x'*-def*: $x' \in_c A \times_c C$ x = *right-coproj* ($A \times_c B$) ($A \times_c C$) $\circ_c x'$
using *x-form* **by** *blast*
then have *ac-exists*: $\exists$ a c. $a \in_c A \wedge c \in_c C \wedge x' =$⟨$a$,$c$⟩
using *cart-prod-decomp* **by** *blast*
then obtain a c **where** *ac-def*: $a \in_c A$ $c \in_c C$ $x' =$⟨a,c⟩
by *blast*
show $x = y$
proof(*cases* $\exists$ y'. $y' \in_c A \times_c B \wedge y =$ *left-coproj* ($A \times_c B$) ($A \times_c C$) $\circ_c y'$)
assume $\exists$ y'. $y' \in_c A \times_c B \wedge y =$ *left-coproj* ($A \times_c B$) ($A \times_c C$) $\circ_c y'$
then obtain y' **where** y'*-def*: $y' \in_c A \times_c B \wedge y =$ *left-coproj* ($A \times_c B$) ($A \times_c C$) $\circ_c y'$
by *blast*
then obtain a' b' **where** $a'b'$*-def*: $a' \in_c A \wedge b' \in_c B \wedge y' =$⟨$a'$,$b'$⟩
using *cart-prod-decomp* y'*-def* **by** *blast*
have *equal-pair*: ⟨a, *right-coproj* B C $\circ_c$ c⟩ = ⟨a', *left-coproj* B C $\circ_c$ b'⟩
proof –
have ⟨a, *right-coproj* B C $\circ_c$ c⟩ = ⟨*id*(A) $\circ_c$ a, *right-coproj* B C $\circ_c$ c⟩
using *ac-def id-left-unit2* **by** *force*
also have ... = (*id* A $\times_f$ *right-coproj* B C) $\circ_c$ ⟨a, c⟩
by (*smt ac-def cfunc-cross-prod-comp-cfunc-prod id-type right-proj-type*)

also have ... = $(\varphi \circ_c$ *right-coproj* $(A \times_c B)$ $(A \times_c C)) \circ_c \langle a, c\rangle$
unfolding *φ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\varphi \circ_c x$
using *ac-def comp-associative2 φ-type x′-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = $\varphi \circ_c y$
by (*simp add*: *local.equal*)
also have ... = $(\varphi \circ_c$ *left-coproj* $(A \times_c B)$ $(A \times_c C)) \circ_c \langle a', b'\rangle$
using *a′b′-def comp-associative2 φ-type y′-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $(id\ A \times_f$ *left-coproj* $B\ C) \circ_c \langle a', b'\rangle$
unfolding *φ-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle id\ A \circ_c a'$, *left-coproj* $B\ C \circ_c b'\rangle$
using *a′b′-def cfunc-cross-prod-comp-cfunc-prod* **by** (*typecheck-cfuncs*,*auto*)
also have ... = $\langle a'$, *left-coproj* $B\ C \circ_c b'\rangle$
using *a′b′-def id-left-unit2* **by** *force*
finally show $\langle a$, *right-coproj* $B\ C \circ_c c\rangle = \langle a'$, *left-coproj* $B\ C \circ_c b'\rangle$**.**
qed
then have *impossible*: *right-coproj* $B\ C \circ_c c =$ *left-coproj* $B\ C \circ_c b'$
using *a′b′-def ac-def cart-prod-eq2 equal-pair* **by** (*typecheck-cfuncs*, *blast*)
then show $x = y$
using *a′b′-def ac-def coproducts-disjoint* **by** *force*
next
assume $\nexists y'.\ y' \in_c A \times_c B \land y =$ *left-coproj* $(A \times_c B)$ $(A \times_c C) \circ_c y'$
then obtain y' **where** *y′-def*: $y' \in_c (A \times_c C) \land y =$ *right-coproj* $(A \times_c B)$ $(A \times_c C) \circ_c y'$
using *y-form* **by** *blast*
then obtain $a'\ c'$ **where** *a′c′-def*: $a' \in_c A\ c' \in_c C\ y' = \langle a', c'\rangle$
using *cart-prod-decomp* **by** *blast*
have *equal-pair*: $\langle a$, *right-coproj* $B\ C \circ_c c\rangle = \langle a'$, *right-coproj* $B\ C \circ_c c'\rangle$
proof −
have $\langle a$, *right-coproj* $B\ C \circ_c c\rangle = \langle id\ A \circ_c a$, *right-coproj* $B\ C \circ_c c\rangle$
using *ac-def id-left-unit2* **by** *force*
also have ... = $(id\ A \times_f$ *right-coproj* $B\ C) \circ_c \langle a, c\rangle$
by (*smt ac-def cfunc-cross-prod-comp-cfunc-prod id-type right-proj-type*)
also have ... = $(\varphi \circ_c$ *right-coproj* $(A \times_c B)$ $(A \times_c C)) \circ_c \langle a, c\rangle$
unfolding *φ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\varphi \circ_c x$
using *ac-def comp-associative2 φ-type x′-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = $\varphi \circ_c y$
by (*simp add*: *local.equal*)
also have ... = $(\varphi \circ_c$ *right-coproj* $(A \times_c B)$ $(A \times_c C)) \circ_c \langle a', c'\rangle$
using *a′c′-def comp-associative2 φ-type y′-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $(id\ A \times_f$ *right-coproj* $B\ C) \circ_c \langle a', c'\rangle$
unfolding *φ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $\langle id\ A \circ_c a'$, *right-coproj* $B\ C \circ_c c'\rangle$

using *a′c′-def cfunc-cross-prod-comp-cfunc-prod* **by** (*typecheck-cfuncs*,*auto*)
also have ... = ⟨*a*′, *right-coproj B C* $\circ_c$ *c*′⟩
using *a′c′-def id-left-unit2* **by** *force*
finally show ⟨*a*, *right-coproj B C* $\circ_c$ *c*⟩ = ⟨*a*′, *right-coproj B C* $\circ_c$ *c*′⟩**.**

qed
then have *a-equal*: *a* = *a*′ ∧ *right-coproj B C* $\circ_c$ *c* = *right-coproj B C* $\circ_c$ *c*′
using *a′c′-def ac-def element-pair-eq equal-pair* **by** (*typecheck-cfuncs*, *blast*)
then have *c-equal*: *c* = *c*′
using *a′c′-def a-equal ac-def right-coproj-are-monomorphisms right-proj-type monomorphism-def3* **by** *blast*
then show *x* = *y*
by (*simp add*: *a′c′-def a-equal ac-def x′-def y′-def*)
qed
qed
qed
then show *monomorphism* (*factor-prod-coprod-left A B C*)
using *φ-def factor-prod-coprod-left-def injective-imp-monomorphism* **by** *fast-force*
qed

lemma *factor-prod-coprod-left-epi*:
epimorphism (*factor-prod-coprod-left A B C*)
proof −
obtain *φ* **where** *φ-def*: *φ* = (*id A* $\times_f$ *left-coproj B C*) ⨿ (*id A* $\times_f$ *right-coproj B C*) **and**
φ-type[*type-rule*]: *φ* : (*A* $\times_c$ *B*) ∐ (*A* $\times_c$ *C*) → *A* $\times_c$ (*B* ∐ *C*)
by (*typecheck-cfuncs*, *simp*)
have *surjective*: *surjective*((*id A* $\times_f$ *left-coproj B C*) ⨿ (*id A* $\times_f$ *right-coproj B C*))
unfolding *surjective-def*
proof(*clarify*)
fix *y*
assume *y-type*: *y* $\in_c$ *codomain* ((*id*$_c$ *A* $\times_f$ *left-coproj B C*) ⨿ (*id*$_c$ *A* $\times_f$ *right-coproj B C*))
then have *y-type2*: *y* $\in_c$ *A* $\times_c$ (*B* ∐ *C*)
using *φ-def φ-type cfunc-type-def* **by** *auto*
then obtain *a* **where** *a-def*: ∃ *bc*. *a* $\in_c$ *A* ∧ *bc* $\in_c$ *B* ∐ *C* ∧ *y* = ⟨*a*,*bc*⟩
by (*meson cart-prod-decomp*)
then obtain *bc* **where** *bc-def*: *bc* $\in_c$ (*B* ∐ *C*) ∧ *y* = ⟨*a*,*bc*⟩
by *blast*
have *bc-form*: (∃ *b*. *b* $\in_c$ *B* ∧ *bc* = *left-coproj B C* $\circ_c$ *b*) ∨ (∃ *c*. *c* $\in_c$ *C* ∧ *bc* = *right-coproj B C* $\circ_c$ *c*)
by (*simp add*: *bc-def coprojs-jointly-surj*)
have *domain-is*: (*A* $\times_c$ *B*) ∐ (*A* $\times_c$ *C*) = *domain* ((*id*$_c$ *A* $\times_f$ *left-coproj B C*) ⨿ (*id*$_c$ *A* $\times_f$ *right-coproj B C*))
by (*typecheck-cfuncs*, *simp add*: *cfunc-type-def*)
show ∃*x*. *x* $\in_c$ *domain* ((*id*$_c$ *A* $\times_f$ *left-coproj B C*) ⨿ (*id*$_c$ *A* $\times_f$ *right-coproj B C*)) ∧

$(id_c\ A \times_f \mathit{left\text{-}coproj}\ B\ C) \amalg (id_c\ A \times_f \mathit{right\text{-}coproj}\ B\ C) \circ_c x = y$
proof(*cases* $\exists\ b.\ b \in_c B \wedge bc = \mathit{left\text{-}coproj}\ B\ C \circ_c b$)
assume *case1*: $\exists b.\ b \in_c B \wedge bc = \mathit{left\text{-}coproj}\ B\ C \circ_c b$
then obtain b **where** *b-def*: $b \in_c B \wedge bc = \mathit{left\text{-}coproj}\ B\ C \circ_c b$
by *blast*
then have *ab-type*: $\langle a,\ b\rangle \in_c (A \times_c B)$
using *a-def b-def* **by** (*typecheck-cfuncs*, *blast*)
obtain x **where** *x-def*: $x = \mathit{left\text{-}coproj}\ (A \times_c B)\ (A \times_c C) \circ_c \langle a,\ b\rangle$
by *simp*
have *x-type*: $x \in_c \mathit{domain}\ ((id_c\ A \times_f \mathit{left\text{-}coproj}\ B\ C) \amalg (id_c\ A \times_f \mathit{right\text{-}coproj}\ B\ C))$
using *ab-type cfunc-type-def codomain-comp domain-comp domain-is left-proj-type x-def* **by** *auto*
have *y-def2*: $y = \langle a, \mathit{left\text{-}coproj}\ B\ C \circ_c b\rangle$
by (*simp add*: *b-def bc-def*)
have $y = (id(A) \times_f \mathit{left\text{-}coproj}\ B\ C) \circ_c \langle a,b\rangle$
using *a-def b-def cfunc-cross-prod-comp-cfunc-prod id-left-unit2 y-def2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\varphi \circ_c \mathit{left\text{-}coproj}\ (A \times_c B)\ (A \times_c C)) \circ_c \langle a,\ b\rangle$
unfolding *φ-def* **by** (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
also have ... $= \varphi \circ_c x$
using *φ-type x-def ab-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
ultimately show $\exists x.\ x \in_c \mathit{domain}\ ((id_c\ A \times_f \mathit{left\text{-}coproj}\ B\ C) \amalg (id_c\ A \times_f \mathit{right\text{-}coproj}\ B\ C)) \wedge$
$(id_c\ A \times_f \mathit{left\text{-}coproj}\ B\ C) \amalg (id_c\ A \times_f \mathit{right\text{-}coproj}\ B\ C) \circ_c x = y$
using *φ-def x-type* **by** *auto*
next
assume $\nexists b.\ b \in_c B \wedge bc = \mathit{left\text{-}coproj}\ B\ C \circ_c b$
then have *case2*: $\exists\ c.\ c \in_c C \wedge bc = (\mathit{right\text{-}coproj}\ B\ C \circ_c c)$
using *bc-form* **by** *blast*
then obtain c **where** *c-def*: $c \in_c C \wedge bc = \mathit{right\text{-}coproj}\ B\ C \circ_c c$
by *blast*
then have *ac-type*: $\langle a,\ c\rangle \in_c (A \times_c C)$
using *a-def c-def* **by** (*typecheck-cfuncs*, *blast*)
obtain x **where** *x-def*: $x = \mathit{right\text{-}coproj}\ (A \times_c B)\ (A \times_c C) \circ_c \langle a,\ c\rangle$
by *simp*
have *x-type*: $x \in_c \mathit{domain}\ ((id_c\ A \times_f \mathit{left\text{-}coproj}\ B\ C) \amalg (id_c\ A \times_f \mathit{right\text{-}coproj}\ B\ C))$
using *ac-type cfunc-type-def codomain-comp domain-comp domain-is right-proj-type x-def* **by** *auto*
have *y-def2*: $y = \langle a, \mathit{right\text{-}coproj}\ B\ C \circ_c c\rangle$
by (*simp add*: *c-def bc-def*)
have $y = (id(A) \times_f \mathit{right\text{-}coproj}\ B\ C) \circ_c \langle a,c\rangle$
using *a-def c-def cfunc-cross-prod-comp-cfunc-prod id-left-unit2 y-def2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\varphi \circ_c \mathit{right\text{-}coproj}\ (A \times_c B)\ (A \times_c C)) \circ_c \langle a,\ c\rangle$
unfolding *φ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= \varphi \circ_c x$
using *φ-type x-def ac-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)

ultimately show $\exists x.\ x \in_c$ *domain* $((id_c\ A \times_f$ *left-coproj* $B\ C) \amalg (id_c\ A \times_f$ *right-coproj* $B\ C)) \wedge$
$(id_c\ A \times_f$ *left-coproj* $B\ C) \amalg (id_c\ A \times_f$ *right-coproj* $B\ C) \circ_c x = y$
using φ-*def* *x-type* **by** *auto*
qed
qed
then show *epimorphism* (*factor-prod-coprod-left* $A\ B\ C$)
by (*simp add*: *factor-prod-coprod-left-def surjective-is-epimorphism*)
qed

lemma *dist-prod-coprod-iso*:
isomorphism(*factor-prod-coprod-left* $A\ B\ C$)
by (*simp add*: *factor-prod-coprod-left-epi factor-prod-coprod-left-mono epi-mon-is-iso*)

The lemma below corresponds to Proposition 2.5.10 in Halvorson.

lemma *prod-distribute-coprod*:
$A \times_c (X \coprod Y) \cong (A \times_c X) \coprod (A \times_c Y)$
using *dist-prod-coprod-iso factor-prod-coprod-left-type is-isomorphic-def isomorphic-is-symmetric* **by** *blast*

9.4.2 Distribute Product over Coproduct on Left

definition *dist-prod-coprod-left* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
dist-prod-coprod-left $A\ B\ C = (THE\ f.\ f : A \times_c (B \coprod C) \to (A \times_c B) \coprod (A \times_c C)$
$\wedge f \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id\ ((A \times_c B) \coprod (A \times_c C))$
$\wedge$ *factor-prod-coprod-left* $A\ B\ C \circ_c f = id\ (A \times_c (B \coprod C)))$

lemma *dist-prod-coprod-left-def2*:
shows *dist-prod-coprod-left* $A\ B\ C : A \times_c (B \coprod C) \to (A \times_c B) \coprod (A \times_c C)$
$\wedge$ *dist-prod-coprod-left* $A\ B\ C \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id\ ((A \times_c B) \coprod (A \times_c C))$
$\wedge$ *factor-prod-coprod-left* $A\ B\ C \circ_c$ *dist-prod-coprod-left* $A\ B\ C = id\ (A \times_c (B \coprod C))$
unfolding *dist-prod-coprod-left-def*
proof (*rule theI'*, *safe*)
show $\exists x.\ x : A \times_c B \coprod C \to (A \times_c B) \coprod A \times_c C \wedge$
$x \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id_c\ ((A \times_c B) \coprod A \times_c C) \wedge$
factor-prod-coprod-left $A\ B\ C \circ_c x = id_c\ (A \times_c B \coprod C)$
using *dist-prod-coprod-iso*[**where** A=A, **where** B=B, **where** C=C] **unfolding** *isomorphism-def*
by (*typecheck-cfuncs*, *auto simp add*: *cfunc-type-def*)
then obtain *inv* **where** *inv-type*: $inv : A \times_c B \coprod C \to (A \times_c B) \coprod A \times_c C$ **and**
inv-left: $inv \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id_c\ ((A \times_c B) \coprod A \times_c C)$ **and**
inv-right: *factor-prod-coprod-left* $A\ B\ C \circ_c inv = id_c\ (A \times_c B \coprod C)$
by *auto*

fix $x\ y$

assume *x-type*: $x : A \times_c B \coprod C \rightarrow (A \times_c B) \coprod A \times_c C$
assume *y-type*: $y : A \times_c B \coprod C \rightarrow (A \times_c B) \coprod A \times_c C$

assume $x \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id_c\ ((A \times_c B) \coprod A \times_c C)$
and $y \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id_c\ ((A \times_c B) \coprod A \times_c C)$
then have $x \circ_c$ *factor-prod-coprod-left* $A\ B\ C = y \circ_c$ *factor-prod-coprod-left* A $B\ C$
by *auto*
then have $(x \circ_c$ *factor-prod-coprod-left* $A\ B\ C) \circ_c inv = (y \circ_c$ *factor-prod-coprod-left* $A\ B\ C) \circ_c inv$
by *auto*
then have $x \circ_c$ *factor-prod-coprod-left* $A\ B\ C \circ_c inv = y \circ_c$ *factor-prod-coprod-left* $A\ B\ C \circ_c inv$
using *inv-type x-type y-type* **by** (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
then have $x \circ_c id_c\ (A \times_c B \coprod C) = y \circ_c id_c\ (A \times_c B \coprod C)$
by (*simp add*: *inv-right*)
then show $x = y$
using *id-right-unit2 x-type y-type* **by** *auto*
qed

lemma *dist-prod-coprod-left-type*[*type-rule*]:
dist-prod-coprod-left $A\ B\ C : A \times_c (B \coprod C) \rightarrow (A \times_c B) \coprod (A \times_c C)$
by (*simp add*: *dist-prod-coprod-left-def2*)

lemma *dist-factor-prod-coprod-left*:
dist-prod-coprod-left $A\ B\ C \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id\ ((A \times_c B) \coprod (A \times_c C))$
by (*simp add*: *dist-prod-coprod-left-def2*)

lemma *factor-dist-prod-coprod-left*:
factor-prod-coprod-left $A\ B\ C \circ_c$ *dist-prod-coprod-left* $A\ B\ C = id\ (A \times_c (B \coprod C))$
by (*simp add*: *dist-prod-coprod-left-def2*)

lemma *dist-prod-coprod-left-iso*:
isomorphism(*dist-prod-coprod-left* $A\ B\ C$)
by (*metis factor-dist-prod-coprod-left dist-prod-coprod-left-type dist-prod-coprod-iso factor-prod-coprod-left-type id-isomorphism id-right-unit2 id-type isomorphism-sandwich*)

lemma *dist-prod-coprod-left-ap-left*:
assumes $a \in_c A\ b \in_c B$
shows *dist-prod-coprod-left* $A\ B\ C \circ_c \langle a,$*left-coproj* $B\ C \circ_c b\rangle =$ *left-coproj* $(A \times_c B)\ (A \times_c C) \circ_c \langle a,b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 dist-prod-coprod-left-def2 factor-prod-coprod-left-ap-left factor-prod-coprod-left-type id-left-unit2*)

lemma *dist-prod-coprod-left-ap-right*:
assumes $a \in_c A\ c \in_c C$
shows *dist-prod-coprod-left* $A\ B\ C \circ_c \langle a,$*right-coproj* $B\ C \circ_c c\rangle =$ *right-coproj* $(A$

$\times_c$ B) (A $\times_c$ C) $\circ_c$ $\langle a,c \rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 dist-prod-coprod-left-def2 factor-prod-coprod-left-ap-right factor-prod-coprod-left-type id-left-unit2*)

9.4.3 Factor Product over Coproduct on Right

definition *factor-prod-coprod-right* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
factor-prod-coprod-right A B C = *swap* C (A $\coprod$ B) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ (*swap* A C $\bowtie_f$ *swap* B C)

lemma *factor-prod-coprod-right-type*[*type-rule*]:
factor-prod-coprod-right A B C : (A $\times_c$ C) $\coprod$ (B $\times_c$ C) $\rightarrow$ (A $\coprod$ B) $\times_c$ C
unfolding *factor-prod-coprod-right-def* **by** *typecheck-cfuncs*

lemma *factor-prod-coprod-right-ap-left*:
assumes $a \in_c A$ $c \in_c C$
shows *factor-prod-coprod-right* A B C $\circ_c$ (*left-coproj* (A $\times_c$ C) (B $\times_c$ C) $\circ_c$ $\langle a, c \rangle$) = $\langle$*left-coproj* A B $\circ_c$ a, $c\rangle$
proof −
have *factor-prod-coprod-right* A B C $\circ_c$ (*left-coproj* (A $\times_c$ C) (B $\times_c$ C) $\circ_c$ $\langle a, c \rangle$)
= (*swap* C (A $\coprod$ B) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ (*swap* A C $\bowtie_f$ *swap* B C)) $\circ_c$ (*left-coproj* (A $\times_c$ C) (B $\times_c$ C) $\circ_c$ $\langle a, c \rangle$)
unfolding *factor-prod-coprod-right-def* **by** *auto*
also have ... = *swap* C (A $\coprod$ B) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ ((*swap* A C $\bowtie_f$ *swap* B C) $\circ_c$ *left-coproj* (A $\times_c$ C) (B $\times_c$ C)) $\circ_c$ $\langle a, c \rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = *swap* C (A $\coprod$ B) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ (*left-coproj* (C $\times_c$ A) (C $\times_c$ B) $\circ_c$ *swap* A C) $\circ_c$ $\langle a, c \rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *left-coproj-cfunc-bowtie-prod*)
also have ... = *swap* C (A $\coprod$ B) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ *left-coproj* (C $\times_c$ A) (C $\times_c$ B) $\circ_c$ *swap* A C $\circ_c$ $\langle a, c \rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *swap* C (A $\coprod$ B) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ *left-coproj* (C $\times_c$ A) (C $\times_c$ B) $\circ_c$ $\langle c, a \rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *swap* C (A $\coprod$ B) $\circ_c$ $\langle c$, *left-coproj* A B $\circ_c$ $a\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *factor-prod-coprod-left-ap-left*)
also have ... = $\langle$*left-coproj* A B $\circ_c$ a, $c\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
finally show *?thesis*.
qed

lemma *factor-prod-coprod-right-ap-right*:
assumes $b \in_c B$ $c \in_c C$
shows *factor-prod-coprod-right* A B C $\circ_c$ *right-coproj* (A $\times_c$ C) (B $\times_c$ C) $\circ_c$ $\langle b, c \rangle$ = $\langle$*right-coproj* A B $\circ_c$ b, $c\rangle$
proof −
have *factor-prod-coprod-right* A B C $\circ_c$ *right-coproj* (A $\times_c$ C) (B $\times_c$ C) $\circ_c$ $\langle b,$

$c\rangle$
= (*swap* C ($A \coprod B$) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ (*swap* A C $\bowtie_f$ *swap* B C)) $\circ_c$ (*right-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ $\langle b, c\rangle$)
unfolding *factor-prod-coprod-right-def* **by** *auto*
also have ... = *swap* C ($A \coprod B$) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ ((*swap* A C $\bowtie_f$ *swap* B C) $\circ_c$ *right-coproj* ($A \times_c C$) ($B \times_c C$)) $\circ_c$ $\langle b, c\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = *swap* C ($A \coprod B$) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ (*right-coproj* ($C \times_c A$) ($C \times_c B$) $\circ_c$ *swap* B C) $\circ_c$ $\langle b, c\rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *right-coproj-cfunc-bowtie-prod*)
also have ... = *swap* C ($A \coprod B$) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ *right-coproj* ($C \times_c A$) ($C \times_c B$) $\circ_c$ *swap* B C $\circ_c$ $\langle b, c\rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *swap* C ($A \coprod B$) $\circ_c$ *factor-prod-coprod-left* C A B $\circ_c$ *right-coproj* ($C \times_c A$) ($C \times_c B$) $\circ_c$ $\langle c, b\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *swap* C ($A \coprod B$) $\circ_c$ $\langle c$, *right-coproj* A B $\circ_c$ $b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *factor-prod-coprod-left-ap-right*)
also have ... = $\langle$*right-coproj* A B $\circ_c$ $b, c\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
finally show *?thesis*.
qed

9.4.4 Distribute Product over Coproduct on Right

definition *dist-prod-coprod-right* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
dist-prod-coprod-right A B C = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ *swap* ($A \coprod B$) C

lemma *dist-prod-coprod-right-type*[*type-rule*]:
dist-prod-coprod-right A B C : ($A \coprod B$) $\times_c C \to (A \times_c C) \coprod (B \times_c C)$
unfolding *dist-prod-coprod-right-def* **by** *typecheck-cfuncs*

lemma *dist-prod-coprod-right-ap-left*:
assumes $a \in_c A$ $c \in_c C$
shows *dist-prod-coprod-right* A B C $\circ_c$ $\langle$*left-coproj* A B $\circ_c$ $a, c\rangle$ = *left-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ $\langle a, c\rangle$
proof –
have *dist-prod-coprod-right* A B C $\circ_c$ $\langle$*left-coproj* A B $\circ_c$ $a, c\rangle$
= ((*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ *swap* ($A \coprod B$) C) $\circ_c$ $\langle$*left-coproj* A B $\circ_c$ $a, c\rangle$
unfolding *dist-prod-coprod-right-def* **by** *auto*
also have ... = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ *swap* ($A \coprod B$) C $\circ_c$ $\langle$*left-coproj* A B $\circ_c$ $a, c\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ $\langle c$, *left-coproj* A B $\circ_c$ $a\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *left-coproj* ($C \times_c A$) ($C \times_c B$) $\circ_c$

$\langle c, a\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *dist-prod-coprod-left-ap-left*)
also have ... = ((*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *left-coproj* ($C \times_c A$) ($C \times_c B$)) $\circ_c$ $\langle c, a\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = (*left-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ *swap* C A) $\circ_c$ $\langle c, a\rangle$
using *assms left-coproj-cfunc-bowtie-prod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *left-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ *swap* C A $\circ_c$ $\langle c, a\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = *left-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ $\langle a, c\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
finally show *?thesis*.
qed

lemma *dist-prod-coprod-right-ap-right*:
assumes $b \in_c B$ $c \in_c C$
shows *dist-prod-coprod-right* A B C $\circ_c$ $\langle$*right-coproj* A B $\circ_c$ b, $c\rangle$ = *right-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ $\langle b, c\rangle$
proof –
have *dist-prod-coprod-right* A B C $\circ_c$ $\langle$*right-coproj* A B $\circ_c$ b, $c\rangle$
= ((*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ *swap* ($A \coprod B$) C) $\circ_c$ $\langle$*right-coproj* A B $\circ_c$ b, $c\rangle$
unfolding *dist-prod-coprod-right-def* **by** *auto*
also have ... = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ *swap* ($A \coprod B$) C $\circ_c$ $\langle$*right-coproj* A B $\circ_c$ b, $c\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-left* C A B $\circ_c$ $\langle c$, *right-coproj* A B $\circ_c$ $b\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *right-coproj* ($C \times_c A$) ($C \times_c B$) $\circ_c$ $\langle c, b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *dist-prod-coprod-left-ap-right*)
also have ... = ((*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *right-coproj* ($C \times_c A$) ($C \times_c B$)) $\circ_c$ $\langle c, b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = (*right-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ *swap* C B) $\circ_c$ $\langle c, b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *right-coproj-cfunc-bowtie-prod*)
also have ... = *right-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ *swap* C B $\circ_c$ $\langle c, b\rangle$
using *assms* **by** (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *right-coproj* ($A \times_c C$) ($B \times_c C$) $\circ_c$ $\langle b, c\rangle$
using *assms swap-ap* **by** (*typecheck-cfuncs*, *auto*)
finally show *?thesis*.
qed

lemma *dist-prod-coprod-right-left-coproj*:
dist-prod-coprod-right X Y H $\circ_c$ (*left-coproj* X Y $\times_f$ *id* H) = *left-coproj* ($X \times_c H$) ($Y \times_c H$)
by (*typecheck-cfuncs*, *smt* (*z3*) *one-separator cart-prod-decomp cfunc-cross-prod-comp-cfunc-prod comp-associative2 dist-prod-coprod-right-ap-left id-left-unit2*)

lemma *dist-prod-coprod-right-right-coproj*:
dist-prod-coprod-right X Y H $\circ_c$ (*right-coproj* X Y $\times_f$ *id* H) = *right-coproj* (X $\times_c$ H) (Y $\times_c$ H)
by (*typecheck-cfuncs*, *smt* (*z3*) *one-separator cart-prod-decomp cfunc-cross-prod-comp-cfunc-prod comp-associative2 dist-prod-coprod-right-ap-right id-left-unit2*)

lemma *factor-dist-prod-coprod-right*:
factor-prod-coprod-right A B C $\circ_c$ *dist-prod-coprod-right* A B C = *id* ((A $\coprod$ B) $\times_c$ C)
unfolding *factor-prod-coprod-right-def dist-prod-coprod-right-def*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *cfunc-bowtie-prod-comp-cfunc-bowtie-prod comp-associative2 factor-dist-prod-coprod-left id-bowtie-prod id-left-unit2 swap-idempotent*)

lemma *dist-factor-prod-coprod-right*:
dist-prod-coprod-right A B C $\circ_c$ *factor-prod-coprod-right* A B C = *id* ((A $\times_c$ C) $\coprod$ (B $\times_c$ C))
unfolding *factor-prod-coprod-right-def dist-prod-coprod-right-def*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *cfunc-bowtie-prod-comp-cfunc-bowtie-prod comp-associative2 dist-factor-prod-coprod-left id-bowtie-prod id-left-unit2 swap-idempotent*)

lemma *factor-prod-coprod-right-iso*:
isomorphism(*factor-prod-coprod-right* A B C)
by (*metis cfunc-type-def dist-factor-prod-coprod-right factor-prod-coprod-right-type factor-dist-prod-coprod-right dist-prod-coprod-right-type isomorphism-def*)

9.5 Casting between Sets

9.5.1 Going from a Set or its Complement to the Superset

This subsection corresponds to Proposition 2.4.5 in Halvorson.

definition *into-super* :: *cfunc* ⇒ *cfunc* **where**
into-super m = m II m^c

lemma *into-super-type*[*type-rule*]:
monomorphism m ⟹ m : X → Y ⟹ *into-super* m : X $\coprod$ (Y \ (X,m)) → Y
unfolding *into-super-def* **by** *typecheck-cfuncs*

lemma *into-super-mono*:
assumes *monomorphism* m m : X → Y
shows *monomorphism* (*into-super* m)
proof (*rule injective-imp-monomorphism*, *unfold injective-def*, *clarify*)
fix x y
assume x $\in_c$ *domain* (*into-super* m) **then have** *x-type*: x $\in_c$ X $\coprod$ (Y \ (X,m))
using *assms cfunc-type-def into-super-type* **by** *auto*

assume y $\in_c$ *domain* (*into-super* m) **then have** *y-type*: y $\in_c$ X $\coprod$ (Y \ (X,m))
using *assms cfunc-type-def into-super-type* **by** *auto*

assume *into-super-eq*: *into-super m* $\circ_c$ *x* = *into-super m* $\circ_c$ *y*

have *x-cases*: ($\exists$ *x'*. *x'* $\in_c$ *X* $\wedge$ *x* = *left-coproj X* (*Y* \ (*X*,*m*)) $\circ_c$ *x'*)
$\vee$ ($\exists$ *x'*. *x'* $\in_c$ *Y* \ (*X*,*m*) $\wedge$ *x* = *right-coproj X* (*Y* \ (*X*,*m*)) $\circ_c$ *x'*)
by (*simp add*: *coprojs-jointly-surj x-type*)

have *y-cases*: ($\exists$ *y'*. *y'* $\in_c$ *X* $\wedge$ *y* = *left-coproj X* (*Y* \ (*X*,*m*)) $\circ_c$ *y'*)
$\vee$ ($\exists$ *y'*. *y'* $\in_c$ *Y* \ (*X*,*m*) $\wedge$ *y* = *right-coproj X* (*Y* \ (*X*,*m*)) $\circ_c$ *y'*)
by (*simp add*: *coprojs-jointly-surj y-type*)

show *x* = *y*
using *x-cases y-cases*
proof *safe*
fix *x' y'*
assume *x'-type*: *x'* $\in_c$ *X* **and** *x-def*: *x* = *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *x'*
assume *y'-type*: *y'* $\in_c$ *X* **and** *y-def*: *y* = *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *y'*

have *into-super m* $\circ_c$ *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *x'* = *into-super m* $\circ_c$ *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *y'*
using *into-super-eq* **unfolding** *x-def y-def* **by** *auto*
then have (*into-super m* $\circ_c$ *left-coproj X* (*Y* \ (*X*, *m*))) $\circ_c$ *x'* = (*into-super m* $\circ_c$ *left-coproj X* (*Y* \ (*X*, *m*))) $\circ_c$ *y'*
using *assms x'-type y'-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have *m* $\circ_c$ *x'* = *m* $\circ_c$ *y'*
using *assms* **unfolding** *into-super-def*
by (*simp add*: *complement-morphism-type left-coproj-cfunc-coprod*)
then have *x'* = *y'*
using *assms cfunc-type-def monomorphism-def x'-type y'-type* **by** *auto*
then show *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *x'* = *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *y'*
by *simp*
next
fix *x' y'*
assume *x'-type*: *x'* $\in_c$ *X* **and** *x-def*: *x* = *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *x'*
assume *y'-type*: *y'* $\in_c$ *Y* \ (*X*, *m*) **and** *y-def*: *y* = *right-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *y'*

have *into-super m* $\circ_c$ *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *x'* = *into-super m* $\circ_c$ *right-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *y'*
using *into-super-eq* **unfolding** *x-def y-def* **by** *auto*
then have (*into-super m* $\circ_c$ *left-coproj X* (*Y* \ (*X*, *m*))) $\circ_c$ *x'* = (*into-super m* $\circ_c$ *right-coproj X* (*Y* \ (*X*, *m*))) $\circ_c$ *y'*
using *assms x'-type y'-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have *m* $\circ_c$ *x'* = m^c $\circ_c$ *y'*
using *assms* **unfolding** *into-super-def*
by (*simp add*: *complement-morphism-type left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
then have *False*
using *assms complement-disjoint x'-type y'-type* **by** *blast*
then show *left-coproj X* (*Y* \ (*X*, *m*)) $\circ_c$ *x'* = *right-coproj X* (*Y* \ (*X*, *m*))

$\circ_c$ y'
by *auto*
next
fix x' y'
assume x'*-type*: $x' \in_c Y \setminus (X, m)$ **and** *x-def*: x = *right-coproj* X $(Y \setminus (X, m)) \circ_c x'$
assume y'*-type*: $y' \in_c X$ **and** *y-def*: y = *left-coproj* X $(Y \setminus (X, m)) \circ_c y'$

have *into-super* $m \circ_c$ *right-coproj* X $(Y \setminus (X, m)) \circ_c x'$ = *into-super* $m \circ_c$ *left-coproj* X $(Y \setminus (X, m)) \circ_c y'$
using *into-super-eq* **unfolding** *x-def y-def* **by** *auto*
then have (*into-super* $m \circ_c$ *right-coproj* X $(Y \setminus (X, m))) \circ_c x'$ = (*into-super* $m \circ_c$ *left-coproj* X $(Y \setminus (X, m))) \circ_c y'$
using *assms* x'*-type* y'*-type* *comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have $m^c \circ_c x' = m \circ_c y'$
using *assms* **unfolding** *into-super-def*
by (*simp add*: *complement-morphism-type left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
then have *False*
using *assms complement-disjoint* x'*-type* y'*-type* **by** *fastforce*
then show *right-coproj* X $(Y \setminus (X, m)) \circ_c x'$ = *left-coproj* X $(Y \setminus (X, m))$ $\circ_c$ y'
by *auto*
next
fix x' y'
assume x'*-type*: $x' \in_c Y \setminus (X, m)$ **and** *x-def*: x = *right-coproj* X $(Y \setminus (X, m)) \circ_c x'$
assume y'*-type*: $y' \in_c Y \setminus (X, m)$ **and** *y-def*: y = *right-coproj* X $(Y \setminus (X, m)) \circ_c y'$

have *into-super* $m \circ_c$ *right-coproj* X $(Y \setminus (X, m)) \circ_c x'$ = *into-super* $m \circ_c$ *right-coproj* X $(Y \setminus (X, m)) \circ_c y'$
using *into-super-eq* **unfolding** *x-def y-def* **by** *auto*
then have (*into-super* $m \circ_c$ *right-coproj* X $(Y \setminus (X, m))) \circ_c x'$ = (*into-super* $m \circ_c$ *right-coproj* X $(Y \setminus (X, m))) \circ_c y'$
using *assms* x'*-type* y'*-type* *comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have $m^c \circ_c x' = m^c \circ_c y'$
using *assms* **unfolding** *into-super-def*
by (*simp add*: *complement-morphism-type right-coproj-cfunc-coprod*)
then have $x' = y'$
using *assms complement-morphism-mono complement-morphism-type monomorphism-def2* x'*-type* y'*-type* **by** *blast*
then show *right-coproj* X $(Y \setminus (X, m)) \circ_c x'$ = *right-coproj* X $(Y \setminus (X, m))$ $\circ_c$ y'
by *simp*
qed
qed

lemma *into-super-epi*:
assumes *monomorphism* m $m : X \to Y$

shows *epimorphism* (*into-super* m)
proof (*rule surjective-is-epimorphism*, *unfold surjective-def*, *clarify*)
fix y
assume $y \in_c$ *codomain* (*into-super* m)
then have *y-type*: $y \in_c Y$
using *assms cfunc-type-def into-super-type* **by** *auto*

have *y-cases*: (*characteristic-func* $m \circ_c y = \mathrm{t}$) $\vee$ (*characteristic-func* $m \circ_c y = \mathrm{f}$)
using *y-type assms true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then show $\exists x.\ x \in_c$ *domain* (*into-super* m) $\wedge$ *into-super* $m \circ_c x = y$
proof *safe*
assume *characteristic-func* $m \circ_c y = \mathrm{t}$
then have $y \in_Y (X, m)$
by (*simp add*: *assms characteristic-func-true-relative-member y-type*)
then obtain x **where** *x-type*: $x \in_c X$ **and** *x-def*: $y = m \circ_c x$
unfolding *relative-member-def2* **by** (*auto*, *unfold factors-through-def2*, *auto*)
then show $\exists x.\ x \in_c$ *domain* (*into-super* m) $\wedge$ *into-super* $m \circ_c x = y$
unfolding *into-super-def* **using** *assms cfunc-type-def comp-associative left-coproj-cfunc-coprod*
by (*intro exI*[**where** x=*left-coproj* X ($Y \setminus (X, m)$) $\circ_c x$], *typecheck-cfuncs*, *metis*)
next
assume *characteristic-func* $m \circ_c y = \mathrm{f}$
then have $\neg\ y \in_Y (X, m)$
by (*simp add*: *assms characteristic-func-false-not-relative-member y-type*)
then have $y \in_Y (Y \setminus (X, m), m^c)$
by (*simp add*: *assms not-in-subset-in-complement y-type*)
then obtain x' **where** x'*-type*: $x' \in_c Y \setminus (X, m)$ **and** x'*-def*: $y = m^c \circ_c x'$
unfolding *relative-member-def2* **by** (*auto*, *unfold factors-through-def2*, *auto*)
then show $\exists x.\ x \in_c$ *domain* (*into-super* m) $\wedge$ *into-super* $m \circ_c x = y$
unfolding *into-super-def* **using** *assms cfunc-type-def comp-associative right-coproj-cfunc-coprod*
by (*intro exI*[**where** x=*right-coproj* X ($Y \setminus (X, m)$) $\circ_c x'$], *typecheck-cfuncs*, *metis*)
qed
qed

lemma *into-super-iso*:
assumes *monomorphism* m $m : X \to Y$
shows *isomorphism* (*into-super* m)
using *assms epi-mon-is-iso into-super-epi into-super-mono* **by** *auto*

9.5.2 Going from a Set to a Subset or its Complement

definition *try-cast* :: *cfunc* $\Rightarrow$ *cfunc* **where**
try-cast m = (*THE* m'. m' : *codomain* $m \to$ *domain* $m \coprod$ ((*codomain* m) $\setminus$ ((*domain* m),m))
$\wedge$ $m' \circ_c$ *into-super* m = *id* (*domain* $m \coprod$ (*codomain* $m \setminus$ ((*domain* m),m)))
$\wedge$ *into-super* $m \circ_c m'$ = *id* (*codomain* m))

lemma *try-cast-def2*:
assumes *monomorphism m m* : $X \to Y$
shows *try-cast m* : *codomain m* $\to$ (*domain m*) $\coprod$ ((*codomain m*) $\setminus$ ((*domain m*),*m*))
$\wedge$ *try-cast m* $\circ_c$ *into-super m* = *id* ((*domain m*) $\coprod$ ((*codomain m*) $\setminus$ ((*domain m*),*m*)))
$\wedge$ *into-super m* $\circ_c$ *try-cast m* = *id* (*codomain m*)
unfolding *try-cast-def*
proof (*rule theI'*, *safe*)
show $\exists$ *x*. *x* : *codomain m* $\to$ *domain m* $\coprod$ (*codomain m* $\setminus$ (*domain m*, *m*)) $\wedge$
x $\circ_c$ *into-super m* = id_c (*domain m* $\coprod$ (*codomain m* $\setminus$ (*domain m*, *m*))) $\wedge$
into-super m $\circ_c$ *x* = id_c (*codomain m*)
using *assms into-super-iso cfunc-type-def into-super-type* **unfolding** *isomorphism-def* **by** *fastforce*
next
fix *x y*
assume *x-type*: *x* : *codomain m* $\to$ *domain m* $\coprod$ (*codomain m* $\setminus$ (*domain m*, *m*))
assume *y-type*: *y* : *codomain m* $\to$ *domain m* $\coprod$ (*codomain m* $\setminus$ (*domain m*, *m*))
assume *into-super m* $\circ_c$ *x* = id_c (*codomain m*) **and** *into-super m* $\circ_c$ *y* = id_c (*codomain m*)
then have *into-super m* $\circ_c$ *x* = *into-super m* $\circ_c$ *y*
by *auto*
then show *x* = *y*
using *into-super-mono* **unfolding** *monomorphism-def*
by (*metis assms*(*1*) *cfunc-type-def into-super-type monomorphism-def x-type y-type*)
qed

lemma *try-cast-type*[*type-rule*]:
assumes *monomorphism m m* : $X \to Y$
shows *try-cast m* : $Y \to X \coprod (Y \setminus (X,m))$
using *assms cfunc-type-def try-cast-def2* **by** *auto*

lemma *try-cast-into-super*:
assumes *monomorphism m m* : $X \to Y$
shows *try-cast m* $\circ_c$ *into-super m* = *id* $(X \coprod (Y \setminus (X,m)))$
using *assms cfunc-type-def try-cast-def2* **by** *auto*

lemma *into-super-try-cast*:
assumes *monomorphism m m* : $X \to Y$
shows *into-super m* $\circ_c$ *try-cast m* = *id Y*
using *assms cfunc-type-def try-cast-def2* **by** *auto*

lemma *try-cast-in-X*:
assumes *m-type*: *monomorphism m m* : $X \to Y$
assumes *y-in-X*: $y \in_Y (X, m)$
shows $\exists$ *x*. $x \in_c X$ $\wedge$ *try-cast m* $\circ_c$ *y* = *left-coproj X* $(Y \setminus (X,m))$ $\circ_c$ *x*
proof –
have *y-type*: $y \in_c Y$

using *y-in-X* **unfolding** *relative-member-def2* **by** *auto*
obtain x **where** *x-type*: $x \in_c X$ **and** *x-def*: $y = m \circ_c x$
using *y-in-X* **unfolding** *relative-member-def2 factors-through-def* **by** (*auto simp add*: *cfunc-type-def*)
then have $y = (\textit{into-super } m \circ_c \textit{left-coproj } X \ (Y \setminus (X,m))) \circ_c x$
unfolding *into-super-def* **using** *complement-morphism-type left-coproj-cfunc-coprod m-type* **by** *auto*
then have $y = \textit{into-super } m \circ_c \textit{left-coproj } X \ (Y \setminus (X,m)) \circ_c x$
using *x-type m-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have $\textit{try-cast } m \circ_c y = (\textit{try-cast } m \circ_c \textit{into-super } m) \circ_c \textit{left-coproj } X \ (Y \setminus (X,m)) \circ_c x$
using *x-type m-type* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
then have $\textit{try-cast } m \circ_c y = \textit{left-coproj } X \ (Y \setminus (X,m)) \circ_c x$
using *m-type x-type* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2 try-cast-into-super*)
then show *?thesis*
using *x-type* **by** *blast*
qed

lemma *try-cast-not-in-X*:
assumes *m-type*: *monomorphism m* $m : X \to Y$
assumes *y-in-X*: $\neg\ y \in_Y (X, m)$ **and** *y-type*: $y \in_c Y$
shows $\exists\ x.\ x \in_c Y \setminus (X,m) \wedge \textit{try-cast } m \circ_c y = \textit{right-coproj } X \ (Y \setminus (X,m)) \circ_c x$
proof −
have *y-in-complement*: $y \in_Y (Y \setminus (X,m),\ m^c)$
by (*simp add*: *assms not-in-subset-in-complement*)
then obtain x **where** *x-type*: $x \in_c Y \setminus (X,m)$ **and** *x-def*: $y = m^c \circ_c x$
unfolding *relative-member-def2 factors-through-def* **by** (*auto simp add*: *cfunc-type-def*)
then have $y = (\textit{into-super } m \circ_c \textit{right-coproj } X \ (Y \setminus (X,m))) \circ_c x$
unfolding *into-super-def* **using** *complement-morphism-type m-type right-coproj-cfunc-coprod* **by** *auto*
then have $y = \textit{into-super } m \circ_c \textit{right-coproj } X \ (Y \setminus (X,m)) \circ_c x$
using *x-type m-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have $\textit{try-cast } m \circ_c y = (\textit{try-cast } m \circ_c \textit{into-super } m) \circ_c \textit{right-coproj } X \ (Y \setminus (X,m)) \circ_c x$
using *x-type m-type* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
then have $\textit{try-cast } m \circ_c y = \textit{right-coproj } X \ (Y \setminus (X,m)) \circ_c x$
using *m-type x-type* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2 try-cast-into-super*)
then show *?thesis*
using *x-type* **by** *blast*
qed

lemma *try-cast-m-m*:
assumes *m-type*: *monomorphism m* $m : X \to Y$
shows $(\textit{try-cast } m) \circ_c m = \textit{left-coproj } X \ (Y \setminus (X,m))$
by (*smt comp-associative2 complement-morphism-type id-left-unit2 into-super-def into-super-type left-coproj-cfunc-coprod left-proj-type m-type try-cast-into-super try-cast-type*)

lemma *try-cast-m-m'*:

assumes *m-type*: *monomorphism m* $m : X \to Y$
shows (*try-cast m*) $\circ_c$ m^c = *right-coproj X* ($Y \setminus (X,m)$)
by (*smt comp-associative2 complement-morphism-type id-left-unit2 into-super-def into-super-type m-type(1) m-type(2) right-coproj-cfunc-coprod right-proj-type try-cast-into-super try-cast-type*)

lemma *try-cast-mono*:
assumes *m-type*: *monomorphism m* $m : X \to Y$
shows *monomorphism*(*try-cast m*)
by (*smt cfunc-type-def comp-monic-imp-monic′ id-isomorphism into-super-type iso-imp-epi-and-monic try-cast-def2 assms*)

9.6 Cases

definition *cases* :: *cfunc* ⇒ *cfunc* **where**
cases(*f*) = ((*right-cart-proj* **1** (*domain f*)) $\bowtie_f$ (*right-cart-proj* **1** (*domain f*))) $\circ_c$ (*dist-prod-coprod-right* **1 1** (*domain f*)) $\circ_c$ ⟨*case-bool* $\circ_c$ *f*, *id*(*domain*(*f*))⟩

lemma *cases-def2*:
assumes $f : X \to \Omega$
shows *cases*(*f*) = ((*right-cart-proj* **1** *X*) $\bowtie_f$ (*right-cart-proj* **1** *X*)) $\circ_c$ (*dist-prod-coprod-right* **1 1** *X*) $\circ_c$ ⟨*case-bool* $\circ_c$ *f*, *id X*⟩
unfolding *cases-def*
using *assms cfunc-type-def* **by** *auto*

lemma *cases-type*[*type-rule*]:
assumes $f : X \to \Omega$
shows *cases*(*f*) : $X \to X \coprod X$
using *assms* **by**(*etcs-subst cases-def2*,
meson case-bool-def2 cfunc-bowtie-prod-type cfunc-prod-type comp-type dist-prod-coprod-right-type id-type right-cart-proj-type)

lemma *true-case*:
assumes *x-type*[*type-rule*]: $x \in_c X$
assumes *f-type*[*type-rule*]: $f : X \to \Omega$
assumes *true-case*: $f \circ_c x$ = t
shows *cases f* $\circ_c$ *x* = *left-coproj X X* $\circ_c$ *x*
proof (*etcs-subst cases-def2*)
have ((*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1 1** *X* $\circ_c$ ⟨*case-bool* $\circ_c$ *f*,id_c *X*⟩) $\circ_c$ *x*
= (*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1 1** *X* $\circ_c$ ⟨*case-bool* $\circ_c$ *f* $\circ_c$ *x*, *x*⟩
using *cfunc-prod-comp comp-associative2 id-left-unit2* **by** (*etcs-assocr*, *type-check-cfuncs*, *force*)
also have ... = (*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1 1** *X* $\circ_c$ ⟨*left-coproj* **1 1**, *x*⟩
using *true-case case-bool-true* **by** *argo*
also have ... = (*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *left-coproj* (**1** $\times_c$ *X*) (**1** $\times_c$ *X*) $\circ_c$ ⟨*id* **1** , *x*⟩

by (*typecheck-cfuncs*, *metis dist-prod-coprod-right-ap-left id-right-unit2*)
also have ... = *left-coproj X X* $\circ_c$ *right-cart-proj* **1** *X* $\circ_c$ $\langle$*id* **1**, *x*$\rangle$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-bowtie-prod*)
also have ... = *left-coproj X X* $\circ_c$ *x*
using *right-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *presburger*)
finally show ((*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1** **1** *X* $\circ_c$ $\langle$*case-bool* $\circ_c$ *f*,*id*$_c$ *X*$\rangle$) $\circ_c$ *x* = *left-coproj X X* $\circ_c$ *x*.
qed

lemma *false-case*:
assumes *x-type*[*type-rule*]: $x \in_c X$
assumes *f-type*[*type-rule*]: $f : X \to \Omega$
assumes *false-case*: $f \circ_c x = \mathrm{f}$
shows *cases f* $\circ_c$ *x* = *right-coproj X X* $\circ_c$ *x*
proof (*etcs-subst cases-def2*)
have ((*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$
dist-prod-coprod-right **1** **1** *X* $\circ_c$ $\langle$*case-bool* $\circ_c$ *f*,*id*$_c$ *X*$\rangle$) $\circ_c$ *x*
= (*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1** **1** *X* $\circ_c$ $\langle$*case-bool* $\circ_c$ *f* $\circ_c$ *x*, *x*$\rangle$
using *cfunc-prod-comp comp-associative2 id-left-unit2* **by** (*etcs-assocr*, *typecheck-cfuncs*, *force*)
also have ... = (*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1** **1** *X* $\circ_c$ $\langle$*right-coproj* **1** **1**, *x*$\rangle$
using *false-case case-bool-false* **by** *argo*
also have ... = (*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *right-coproj* (**1** $\times_c$ *X*) (**1** $\times_c$ *X*) $\circ_c$ $\langle$*id* **1** , *x*$\rangle$
by (*typecheck-cfuncs*, *metis dist-prod-coprod-right-ap-right id-right-unit2*)
also have ... = *right-coproj X X* $\circ_c$ *right-cart-proj* **1** *X* $\circ_c$ $\langle$*id* **1**, *x*$\rangle$
using *comp-associative2 right-coproj-cfunc-bowtie-prod* **by** (*typecheck-cfuncs*, *force*)
also have ... = *right-coproj X X* $\circ_c$ *x*
using *right-cart-proj-cfunc-prod* **by** (*typecheck-cfuncs*, *presburger*)
finally show ((*right-cart-proj* **1** *X* $\bowtie_f$ *right-cart-proj* **1** *X*) $\circ_c$ *dist-prod-coprod-right* **1** **1** *X* $\circ_c$ $\langle$*case-bool* $\circ_c$ *f*,*id*$_c$ *X*$\rangle$) $\circ_c$ *x* = *right-coproj X X* $\circ_c$ *x*.
qed

9.7 Coproduct Set Properities

lemma *coproduct-commutes*:
$A \coprod B \cong B \coprod A$
proof –
have *id-AB*: ((*right-coproj A B*) II (*left-coproj A B*)) $\circ_c$ ((*right-coproj B A*) II (*left-coproj B A*)) = *id*($A \coprod B$)
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-comp id-coprod left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
have *id-BA*: ((*right-coproj B A*) II (*left-coproj B A*)) $\circ_c$ ((*right-coproj A B*) II (*left-coproj A B*)) = *id*($B \coprod A$)
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-comp id-coprod right-coproj-cfunc-coprod left-coproj-cfunc-coprod*)

show $A \coprod B \cong B \coprod A$
by (*smt* (*verit*, *ccfv-threshold*) *cfunc-coprod-type cfunc-type-def id-AB id-BA is-isomorphic-def isomorphism-def left-proj-type right-proj-type*)
qed

lemma *coproduct-associates*:
$A \coprod (B \coprod C) \cong (A \coprod B) \coprod C$
proof −
obtain q **where** *q-def*: $q = (\textit{left-coproj}\ (A \coprod B)\ C\ ) \circ_c (\textit{right-coproj}\ A\ B)$ **and** *q-type*[*type-rule*]: q: $B \to (A \coprod B) \coprod C$
by (*typecheck-cfuncs*, *simp*)
obtain f **where** *f-def*: $f = q\ \amalg\ (\textit{right-coproj}\ (A \coprod B)\ C)$ **and** *f-type*[*type-rule*]: (f: $(B \coprod C) \to ((A \coprod B) \coprod C)$)
by (*typecheck-cfuncs*, *simp*)
have *f-prop*: $(f \circ_c \textit{left-coproj}\ B\ C = q) \wedge (f \circ_c \textit{right-coproj}\ B\ C = \textit{right-coproj}\ (A \coprod B)\ C)$
by (*typecheck-cfuncs*, *simp add*: *f-def left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
then have *f-unique*: $(\exists!f.\ (f: (B \coprod C) \to ((A \coprod B) \coprod C)) \wedge (f \circ_c \textit{left-coproj}\ B\ C = q) \wedge (f \circ_c \textit{right-coproj}\ B\ C = \textit{right-coproj}\ (A \coprod B)\ C))$
by (*typecheck-cfuncs*, *metis cfunc-coprod-unique f-prop f-type*)

obtain m **where** *m-def*: $m = (\textit{left-coproj}\ (A \coprod B)\ C\ ) \circ_c (\textit{left-coproj}\ A\ B)$ **and** *m-type*[*type-rule*]: $m : A \to (A \coprod B) \coprod C$
by (*typecheck-cfuncs*, *simp*)
obtain g **where** *g-def*: $g = m\ \amalg\ f$ **and** *g-type*[*type-rule*]: g: $A \coprod (B \coprod C) \to (A \coprod B) \coprod C$
by (*typecheck-cfuncs*, *simp*)
have *g-prop*: $(g \circ_c (\textit{left-coproj}\ A\ (B \coprod C)) = m) \wedge (g \circ_c (\textit{right-coproj}\ A\ (B \coprod C)) = f)$
by (*typecheck-cfuncs*, *simp add*: *g-def left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)

have *g-unique*: $\exists!\ g.\ ((g: A \coprod (B \coprod C) \to (A \coprod B) \coprod C) \wedge (g \circ_c (\textit{left-coproj}\ A\ (B \coprod C)) = m) \wedge (g \circ_c (\textit{right-coproj}\ A\ (B \coprod C)) = f))$
by (*typecheck-cfuncs*, *metis cfunc-coprod-unique g-prop g-type*)

obtain p **where** *p-def*: $p = (\textit{right-coproj}\ A\ (B \coprod C)) \circ_c (\textit{left-coproj}\ B\ C)$ **and** *p-type*[*type-rule*]: p: $B \to A \coprod (B \coprod C)$
by (*typecheck-cfuncs*, *simp*)
obtain h **where** *h-def*: $h = (\textit{left-coproj}\ A\ (B \coprod C))\ \amalg\ p$ **and** *h-type*[*type-rule*]: h: $(A \coprod B) \to A \coprod (B \coprod C)$
by (*typecheck-cfuncs*, *simp*)
have *h-prop1*: $h \circ_c (\textit{left-coproj}\ A\ B) = (\textit{left-coproj}\ A\ (B \coprod C))$
by (*typecheck-cfuncs*, *simp add*: *h-def left-coproj-cfunc-coprod p-type*)
have *h-prop2*: $h \circ_c (\textit{right-coproj}\ A\ B) = p$
using *h-def left-proj-type right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *blast*)
have *h-unique*: $\exists!\ h.\ ((h: (A \coprod B) \to A \coprod (B \coprod C)) \wedge (h \circ_c (\textit{left-coproj}\ A\ B) = (\textit{left-coproj}\ A\ (B \coprod C))) \wedge (h \circ_c (\textit{right-coproj}\ A\ B) = p))$
by (*typecheck-cfuncs*, *metis cfunc-coprod-unique h-prop1 h-prop2 h-type*)

obtain *j* **where** *j-def*: $j = (right\text{-}coproj\ A\ (B \coprod\ C)) \circ_c\ (right\text{-}coproj\ B\ C)$ **and** *j-type*[*type-rule*]: $j : C \rightarrow A \coprod\ (B \coprod\ C)$
by (*typecheck-cfuncs*, *simp*)
obtain *k* **where** *k-def*: $k = h \amalg j$ **and** *k-type*[*type-rule*]: *k*: $(A \coprod\ B) \coprod\ C \rightarrow A \coprod\ (B \coprod\ C)$
by (*typecheck-cfuncs*, *simp*)

have *fact1*: $(k \circ_c\ g) \circ_c\ (left\text{-}coproj\ A\ (B \coprod\ C)) = (left\text{-}coproj\ A\ (B \coprod\ C))$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 g-prop h-prop1 h-type j-type k-def left-coproj-cfunc-coprod left-proj-type m-def*)
have *fact2*: $(g \circ_c\ k) \circ_c\ (left\text{-}coproj\ (A \coprod\ B)\ C) = (left\text{-}coproj\ (A \coprod\ B)\ C)$
by (*typecheck-cfuncs*, *smt* (*verit*) *cfunc-coprod-comp cfunc-coprod-unique comp-associative2 comp-type f-prop g-prop g-type h-def h-type j-def k-def k-type left-coproj-cfunc-coprod left-proj-type m-def p-def p-type q-def right-proj-type*)
have *fact3*: $(g \circ_c\ k) \circ_c\ (right\text{-}coproj\ (A \coprod\ B)\ C) = (right\text{-}coproj\ (A \coprod\ B)\ C)$
by (*smt comp-associative2 comp-type f-def g-prop g-type h-type j-def k-def k-type q-type right-coproj-cfunc-coprod right-proj-type*)
have *fact4*: $(k \circ_c\ g) \circ_c\ (right\text{-}coproj\ A\ (B \coprod\ C)) = (right\text{-}coproj\ A\ (B \coprod\ C))$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *cfunc-coprod-unique cfunc-type-def comp-associative comp-type f-prop g-prop h-prop2 h-type j-def k-def left-coproj-cfunc-coprod left-proj-type p-def q-def right-coproj-cfunc-coprod right-proj-type*)
have *fact5*: $(k \circ_c\ g) = id(\ A \coprod\ (B \coprod\ C))$
by (*typecheck-cfuncs*, *metis cfunc-coprod-unique fact1 fact4 id-coprod left-proj-type right-proj-type*)
have *fact6*: $(g \circ_c\ k) = id((A \coprod\ B) \coprod\ C)$
by (*typecheck-cfuncs*, *metis cfunc-coprod-unique fact2 fact3 id-coprod left-proj-type right-proj-type*)
show *?thesis*
by (*metis cfunc-type-def fact5 fact6 g-type is-isomorphic-def isomorphism-def k-type*)
qed

The lemma below corresponds to Proposition 2.5.10.

lemma *product-distribute-over-coproduct-left*:
$A \times_c\ (X \coprod\ Y) \cong (A \times_c\ X) \coprod\ (A \times_c\ Y)$
using *factor-prod-coprod-left-type dist-prod-coprod-iso is-isomorphic-def isomorphic-is-symmetric* **by** *blast*

lemma *prod-pres-iso*:
assumes $A \cong\ C\ \ B \cong D$
shows $A \times_c B \cong\ C \times_c D$
proof −
obtain *f* **where** *f-def*: *f*: $A \rightarrow C \wedge isomorphism(f)$
using *assms*(*1*) *is-isomorphic-def* **by** *blast*
obtain *g* **where** *g-def*: *g*: $B \rightarrow D \wedge isomorphism(g)$
using *assms*(*2*) *is-isomorphic-def* **by** *blast*
have $isomorphism(f \times_f g)$
by (*meson cfunc-cross-prod-mono cfunc-cross-prod-surj epi-is-surj epi-mon-is-iso f-def g-def iso-imp-epi-and-monic surjective-is-epimorphism*)

then show $A \times_c B \cong\ C \times_c D$
by (*meson cfunc-cross-prod-type f-def g-def is-isomorphic-def*)
qed

lemma *coprod-pres-iso*:
assumes $A \cong\ C\ \ B \cong D$
shows $A \coprod\ B \cong\ C \coprod\ D$
proof−
obtain f **where** *f-def*: f: $A \to C$ *isomorphism*(f)
using *assms*(*1*) *is-isomorphic-def* **by** *blast*
obtain g **where** *g-def*: g: $B \to D$ *isomorphism*(g)
using *assms*(*2*) *is-isomorphic-def* **by** *blast*

have *surj-f*: *surjective*(f)
using *epi-is-surj f-def iso-imp-epi-and-monic* **by** *blast*
have *surj-g*: *surjective*(g)
using *epi-is-surj g-def iso-imp-epi-and-monic* **by** *blast*

have *coproj-f-inject*: *injective*(((*left-coproj C D*) $\circ_c$ f))
using *cfunc-type-def composition-of-monic-pair-is-monic f-def iso-imp-epi-and-monic left-coproj-are-monomorphisms left-proj-type monomorphism-imp-injective* **by** *auto*

have *coproj-g-inject*: *injective*(((*right-coproj C D*) $\circ_c$ g))
using *cfunc-type-def composition-of-monic-pair-is-monic g-def iso-imp-epi-and-monic right-coproj-are-monomorphisms right-proj-type monomorphism-imp-injective* **by** *auto*

obtain φ **where** φ*-def*: φ = (*left-coproj C D* $\circ_c$ f) II (*right-coproj C D* $\circ_c$ g)
by *simp*
then have φ*-type*: φ: $A \coprod\ B \to\ \ C \coprod\ D$
using *cfunc-coprod-type cfunc-type-def codomain-comp domain-comp f-def g-def left-proj-type right-proj-type* **by** *auto*

have *surjective*(φ)
unfolding *surjective-def*
proof(*clarify*)
fix y
assume *y-type*: $y \in_c$ *codomain* φ
then have *y-type2*: $y \in_c C \coprod\ D$
using φ*-type cfunc-type-def* **by** *auto*
then have *y-form*: ($\exists$ c. $c \in_c C \land y =$ *left-coproj C D* $\circ_c$ c)
$\lor$ ($\exists$ d. $d \in_c D \land y =$ *right-coproj C D* $\circ_c$ d)
using *coprojs-jointly-surj* **by** *auto*
show $\exists x$. $x \in_c$ *domain* $\varphi \land \varphi \circ_c x = y$
proof(*cases* $\exists$ c. $c \in_c C \land y =$ *left-coproj C D* $\circ_c$ c)
assume $\exists$ c. $c \in_c C \land y =$ *left-coproj C D* $\circ_c$ c
then obtain c **where** *c-def*: $c \in_c C \land y =$ *left-coproj C D* $\circ_c$ c
by *blast*
then have $\exists$ a. $a \in_c A \land f \circ_c a = c$
using *cfunc-type-def f-def surj-f surjective-def* **by** *auto*

```
      then obtain a where a-def: a ∈c A ∧ f ∘c a = c
        by blast
      obtain x where x-def: x = left-coproj A B ∘c a
        by blast
      have x-type: x ∈c A ∐ B
        using a-def comp-type left-proj-type x-def by blast
      have φ ∘c x = y
      using φ-def φ-type a-def c-def cfunc-type-def comp-associative comp-type f-def
g-def left-coproj-cfunc-coprod left-proj-type right-proj-type x-def by (smt (verit))
      then show ∃x. x ∈c domain φ ∧ φ ∘c x = y
        using φ-type cfunc-type-def x-type by auto
    next
      assume ∄c. c ∈c C ∧ y = left-coproj C D ∘c c
      then have y-def2: ∃ d. d ∈c D ∧ y = right-coproj C D ∘c d
        using y-form by blast
      then obtain d where d-def: d ∈c D y = right-coproj C D ∘c d
        by blast
      then have ∃ b. b ∈c B ∧ g ∘c b = d
        using cfunc-type-def g-def surj-g surjective-def by auto
      then obtain b where b-def: b ∈c B g ∘c b = d
        by blast
      obtain x where x-def: x = right-coproj A B ∘c b
        by blast
      have x-type: x ∈c A ∐ B
        using b-def comp-type right-proj-type x-def by blast
      have φ ∘c x = y
      using φ-def φ-type b-def cfunc-type-def comp-associative comp-type d-def f-def
g-def left-proj-type right-coproj-cfunc-coprod right-proj-type x-def by (smt (verit))
      then show ∃x. x ∈c domain φ ∧ φ ∘c x = y
        using φ-type cfunc-type-def x-type by auto
    qed
  qed

  have injective(φ)
    unfolding injective-def
  proof(clarify)
    fix x y
    assume x-type: x ∈c domain φ
    assume y-type: y ∈c domain φ
    assume equals: φ ∘c x = φ ∘c y
    have x-type2: x ∈c A ∐ B
      using φ-type cfunc-type-def x-type by auto
    have y-type2: y ∈c A ∐ B
      using φ-type cfunc-type-def y-type by auto

    have phix-type: φ ∘c x ∈c C ∐ D
      using φ-type comp-type x-type2 by blast
    have phiy-type: φ ∘c y ∈c C ∐ D
      using equals phix-type by auto
```

have *x-form*: $(\exists\ a.\ a \in_c A\ \ \wedge x = \textit{left-coproj}\ A\ B \circ_c a)$
$\vee\ (\exists\ b.\ b \in_c B \wedge x = \textit{right-coproj}\ A\ B \circ_c b)$
using *cfunc-type-def coprojs-jointly-surj x-type x-type2 y-type* **by** *auto*

have *y-form*: $(\exists\ a.\ a \in_c A\ \ \wedge y = \textit{left-coproj}\ A\ B \circ_c a)$
$\vee\ (\exists\ b.\ b \in_c B \wedge y = \textit{right-coproj}\ A\ B \circ_c b)$
using *cfunc-type-def coprojs-jointly-surj x-type x-type2 y-type* **by** *auto*

show $x=y$
proof(*cases* $\exists\ a.\ a \in_c A\ \ \wedge x = \textit{left-coproj}\ A\ B \circ_c a$)
assume $\exists\ a.\ a \in_c A\ \ \wedge x = \textit{left-coproj}\ A\ B \circ_c a$
then obtain a **where** *a-def*: $a \in_c A\ x = \textit{left-coproj}\ A\ B \circ_c a$
by *blast*
show $x = y$
proof(*cases* $\exists\ a.\ a \in_c A\ \ \wedge y = \textit{left-coproj}\ A\ B \circ_c a$)
assume $\exists\ a.\ a \in_c A\ \ \wedge y = \textit{left-coproj}\ A\ B \circ_c a$
then obtain a' **where** a'*-def*: $a' \in_c A\ y = \textit{left-coproj}\ A\ B \circ_c a'$
by *blast*
then have $a = a'$
proof –
have $(\textit{left-coproj}\ C\ D \circ_c f) \circ_c a = \varphi \circ_c x$
using φ*-def a-def cfunc-type-def comp-associative comp-type f-def g-def left-coproj-cfunc-coprod left-proj-type right-proj-type x-type* **by** (*smt* (*verit*))
also have ... $= \varphi \circ_c y$
by (*meson equals*)
also have ... $= (\varphi \circ_c \textit{left-coproj}\ A\ B) \circ_c a'$
using φ*-type* a'*-def comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= (\textit{left-coproj}\ C\ D \circ_c f) \circ_c a'$
unfolding φ*-def* **using** *f-def g-def* a'*-def left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
ultimately show $a = a'$
by (*smt* a'*-def a-def cfunc-type-def coproj-f-inject domain-comp f-def injective-def left-proj-type*)
qed
then show $x=y$
by (*simp add*: a'*-def*(*2*) *a-def*(*2*))
next
assume $\nexists a.\ a \in_c A \wedge y = \textit{left-coproj}\ A\ B \circ_c a$
then have $\exists\ b.\ b \in_c B \wedge y = \textit{right-coproj}\ A\ B \circ_c b$
using *y-form* **by** *blast*
then obtain b' **where** b'*-def*: $b' \in_c B\ y = \textit{right-coproj}\ A\ B \circ_c b'$
by *blast*
show $x = y$
proof –
have $\textit{left-coproj}\ C\ D \circ_c (f \circ_c a) = (\textit{left-coproj}\ C\ D \circ_c f) \circ_c a$
using *a-def cfunc-type-def comp-associative f-def left-proj-type* **by** *auto*
also have ... $= \varphi \circ_c x$
using φ*-def a-def cfunc-type-def comp-associative comp-type f-def g-def*

left-coproj-cfunc-coprod left-proj-type right-proj-type x-type **by** (*smt* (*verit*))
also have ... = $\varphi \circ_c y$
by (*meson equals*)
also have ... = ($\varphi \circ_c$ *right-coproj A B*) $\circ_c b'$
using φ*-type* b'*-def comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... = (*right-coproj C D* $\circ_c g$) $\circ_c b'$
unfolding φ*-def* **using** *f-def g-def* b'*-def right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *right-coproj C D* $\circ_c$ ($g \circ_c b'$)
using *g-def* b'*-def* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
ultimately show $x = y$
using *a-def*(*1*) b'*-def*(*1*) *comp-type coproducts-disjoint f-def*(*1*) *g-def*(*1*) **by** *auto*
qed
qed
next
assume $\nexists a.\ a \in_c A \wedge x =$ *left-coproj A B* $\circ_c a$
then have $\exists\ b.\ b \in_c B \wedge x =$ *right-coproj A B* $\circ_c b$
using *x-form* **by** *blast*
then obtain b **where** *b-def*: $b \in_c B \wedge x =$ *right-coproj A B* $\circ_c b$
by *blast*
show $x = y$
proof(*cases* $\exists\ a.\ a \in_c A\ \wedge y =$ *left-coproj A B* $\circ_c a$)
assume $\exists\ a.\ a \in_c A\ \wedge y =$ *left-coproj A B* $\circ_c a$
then obtain a' **where** a'*-def*: $a' \in_c A$ $y =$ *left-coproj A B* $\circ_c a'$
by *blast*
show $x = y$
proof −
have *right-coproj C D* $\circ_c$ ($g \circ_c b$) = (*right-coproj C D* $\circ_c g$) $\circ_c b$
using *b-def cfunc-type-def comp-associative g-def right-proj-type* **by** *auto*
also have ... = $\varphi \circ_c x$
by (*smt* φ*-def* φ*-type b-def comp-associative2 comp-type f-def*(*1*) *g-def*(*1*) *left-proj-type right-coproj-cfunc-coprod right-proj-type*)
also have ... = $\varphi \circ_c y$
by (*meson equals*)
also have ... = ($\varphi \circ_c$ *left-coproj A B*) $\circ_c a'$
using φ*-type* a'*-def comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... = (*left-coproj C D* $\circ_c f$) $\circ_c a'$
unfolding φ*-def* **using** *f-def g-def* a'*-def left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *left-coproj C D* $\circ_c$ ($f \circ_c a'$)
using *f-def* a'*-def* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
ultimately show $x = y$
by (*metis* a'*-def*(*1*) *b-def comp-type coproducts-disjoint f-def*(*1*) *g-def*(*1*))
qed
next
assume $\nexists a.\ a \in_c A \wedge y =$ *left-coproj A B* $\circ_c a$
then have $\exists\ b.\ b \in_c B \wedge y =$ *right-coproj A B* $\circ_c b$
using *y-form* **by** *blast*

then obtain b' **where** b'*-def*: $b' \in_c B$ $y = $ *right-coproj* A $B \circ_c b'$
by *blast*
then have $b = b'$
proof −
have (*right-coproj* C $D \circ_c g) \circ_c b = \varphi \circ_c x$
by (*smt* φ*-def* φ*-type* *b-def* *comp-associative2* *comp-type* *f-def*(*1*) *g-def*(*1*) *left-proj-type* *right-coproj-cfunc-coprod* *right-proj-type*)
also have ... $= \varphi \circ_c y$
by (*meson equals*)
also have ... $= (\varphi \circ_c$ *right-coproj* A $B) \circ_c b'$
using φ*-type* b'*-def* *comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... $=$ (*right-coproj* C $D \circ_c g) \circ_c b'$
unfolding φ*-def* **using** *f-def* *g-def* b'*-def* *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
ultimately show $b = b'$
by (*smt* b'*-def* *b-def* *cfunc-type-def* *coproj-g-inject* *domain-comp* *g-def* *injective-def* *right-proj-type*)
qed
then show $x = y$
by (*simp add*: b'*-def*(*2*) *b-def*)
qed
qed
qed

have *monomorphism* φ
using ‹*injective* φ› *injective-imp-monomorphism* **by** *blast*
have *epimorphism* φ
by (*simp add*: ‹*surjective* φ› *surjective-is-epimorphism*)
have *isomorphism* φ
using ‹*epimorphism* φ› ‹*monomorphism* φ› *epi-mon-is-iso* **by** *blast*
then show *?thesis*
using φ*-type* *is-isomorphic-def* **by** *blast*
qed

lemma *product-distribute-over-coproduct-right*:
$(A \coprod B) \times_c C \cong (A \times_c C) \coprod (B \times_c C)$
by (*meson coprod-pres-iso isomorphic-is-transitive product-commutes product-distribute-over-coproduct-left*)

lemma *coproduct-with-self-iso*:
$X \coprod X \cong X \times_c \Omega$
proof −
obtain ϱ **where** ϱ*-def*: $\varrho = \langle id\, X, \mathrm{t} \circ_c \beta_X \rangle \amalg \langle id\, X, \mathrm{f} \circ_c \beta_X \rangle$ **and** ϱ*-type*[*type-rule*]: $\varrho : X \coprod X \to X \times_c \Omega$
by (*typecheck-cfuncs*, *simp*)
have ϱ*-inj*: *injective* ϱ
unfolding *injective-def*
proof(*clarify*)
fix x y
assume $x \in_c$ *domain* ϱ **then have** *x-type*[*type-rule*]: $x \in_c X \coprod X$

using *ϱ-type cfunc-type-def* **by** *auto*
assume $y \in_c$ *domain* ϱ **then have** *y-type*[*type-rule*]: $y \in_c X \coprod X$
using *ϱ-type cfunc-type-def* **by** *auto*
assume *equals*: $\varrho \circ_c x = \varrho \circ_c y$
show $x = y$
proof(*cases* $\exists$ *lx*. $x =$ *left-coproj* X X $\circ_c$ *lx* $\wedge$ *lx* $\in_c X$)
assume $\exists$ *lx*. $x =$ *left-coproj* X X $\circ_c$ *lx* $\wedge$ *lx* $\in_c X$
then obtain *lx* **where** *lx-def*: $x =$ *left-coproj* X X $\circ_c$ *lx* $\wedge$ *lx* $\in_c X$
by *blast*
have *ϱx*: $\varrho \circ_c x = \langle lx, \mathrm{t}\rangle$
proof −
have $\varrho \circ_c x = (\varrho \circ_c$ *left-coproj* X $X) \circ_c$ *lx*
using *comp-associative2 lx-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle id\ X, \mathrm{t} \circ_c \beta_X\rangle \circ_c$ *lx*
unfolding *ϱ-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= \langle lx, \mathrm{t}\rangle$
by (*typecheck-cfuncs*, *metis cart-prod-extract-left lx-def*)
finally show *?thesis*.
qed
show $x = y$
proof(*cases* $\exists$ *ly*. $y =$ *left-coproj* X X $\circ_c$ *ly* $\wedge$ *ly* $\in_c X$)
assume $\exists$ *ly*. $y =$ *left-coproj* X X $\circ_c$ *ly* $\wedge$ *ly* $\in_c X$
then obtain *ly* **where** *ly-def*: $y =$ *left-coproj* X X $\circ_c$ *ly* $\wedge$ *ly* $\in_c X$
by *blast*
have $\varrho \circ_c y = \langle ly, \mathrm{t}\rangle$
proof −
have $\varrho \circ_c y = (\varrho \circ_c$ *left-coproj* X $X) \circ_c$ *ly*
using *comp-associative2 ly-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle id\ X, \mathrm{t} \circ_c \beta_X\rangle \circ_c$ *ly*
unfolding *ϱ-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= \langle ly, \mathrm{t}\rangle$
by (*typecheck-cfuncs*, *metis cart-prod-extract-left ly-def*)
finally show *?thesis*.
qed
then show $x = y$
using *ϱx cart-prod-eq2 equals lx-def ly-def true-func-type* **by** *auto*
next
assume $\nexists$ *ly*. $y =$ *left-coproj* X X $\circ_c$ *ly* $\wedge$ *ly* $\in_c X$
then obtain *ry* **where** *ry-def*: $y =$ *right-coproj* X X $\circ_c$ *ry* **and** *ry-type*[*type-rule*]: *ry* $\in_c X$
by (*meson y-type coprojs-jointly-surj*)
have *ϱy*: $\varrho \circ_c y = \langle ry, \mathrm{f}\rangle$
proof −
have $\varrho \circ_c y = (\varrho \circ_c$ *right-coproj* X $X) \circ_c$ *ry*
using *comp-associative2 ry-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle id\ X, \mathrm{f} \circ_c \beta_X\rangle \circ_c$ *ry*
unfolding *ϱ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*,

presburger)
also have ... = ⟨*ry*, f⟩
by (*typecheck-cfuncs*, *metis cart-prod-extract-left*)
finally show *?thesis*.
qed
then show *?thesis*
using *ϱx ϱy cart-prod-eq2 equals false-func-type lx-def ry-type true-false-distinct true-func-type* **by** *force*
qed
next
assume $\nexists$ *lx*. $x = \textit{left-coproj}\ X\ X \circ_c lx \wedge lx \in_c X$
then obtain *rx* **where** *rx-def*: $x = \textit{right-coproj}\ X\ X \circ_c rx \wedge rx \in_c X$
by (*typecheck-cfuncs*, *meson coprojs-jointly-surj*)
have *ϱx*: $\varrho \circ_c x = \langle rx, \mathrm{f}\rangle$
proof −
have $\varrho \circ_c x = (\varrho \circ_c \textit{right-coproj}\ X\ X) \circ_c rx$
using *comp-associative2 rx-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $\langle id\ X, \mathrm{f} \circ_c \beta_X\rangle \circ_c rx$
unfolding *ϱ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = ⟨*rx*, f⟩
by (*typecheck-cfuncs*, *metis cart-prod-extract-left rx-def*)
finally show *?thesis*.
qed
show $x = y$
proof(*cases* $\exists$ *ly*. $y = \textit{left-coproj}\ X\ X \circ_c ly \wedge ly \in_c X$)
assume $\exists$ *ly*. $y = \textit{left-coproj}\ X\ X \circ_c ly \wedge ly \in_c X$
then obtain *ly* **where** *ly-def*: $y = \textit{left-coproj}\ X\ X \circ_c ly \wedge ly \in_c X$
by *blast*
have $\varrho \circ_c y = \langle ly, \mathrm{t}\rangle$
proof −
have $\varrho \circ_c y = (\varrho \circ_c \textit{left-coproj}\ X\ X) \circ_c ly$
using *comp-associative2 ly-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $\langle id\ X, \mathrm{t} \circ_c \beta_X\rangle \circ_c ly$
unfolding *ϱ-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = ⟨*ly*, t⟩
by (*typecheck-cfuncs*, *metis cart-prod-extract-left ly-def*)
finally show *?thesis*.
qed
then show $x = y$
using *ϱx cart-prod-eq2 equals false-func-type ly-def rx-def true-false-distinct true-func-type* **by** *force*
next
assume $\nexists$ *ly*. $y = \textit{left-coproj}\ X\ X \circ_c ly \wedge ly \in_c X$
then obtain *ry* **where** *ry-def*: $y = \textit{right-coproj}\ X\ X \circ_c ry \wedge ry \in_c X$
using *coprojs-jointly-surj* **by** (*typecheck-cfuncs*, *blast*)
have *ϱy*: $\varrho \circ_c y = \langle ry, \mathrm{f}\rangle$
proof −

have $\varrho \circ_c y = (\varrho \circ_c$ *right-coproj* X $X) \circ_c ry$
using *comp-associative2 ry-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle id\ X, \mathrm{f} \circ_c \beta_X\rangle \circ_c ry$
unfolding *ϱ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= \langle ry, \mathrm{f}\rangle$
by (*typecheck-cfuncs*, *metis cart-prod-extract-left ry-def*)
finally show *?thesis*.
qed
show $x = y$
using *ϱx ϱy cart-prod-eq2 equals false-func-type rx-def ry-def* **by** *auto*
qed
qed
qed
have *surjective* ϱ
unfolding *surjective-def*
proof(*clarify*)
fix y
assume $y \in_c$ *codomain* ϱ **then have** *y-type*[*type-rule*]: $y \in_c X \times_c \Omega$
using *ϱ-type cfunc-type-def* **by** *fastforce*
then obtain x w **where** *y-def*: $y = \langle x,w\rangle \wedge x \in_c X \wedge w \in_c \Omega$
using *cart-prod-decomp* **by** *fastforce*
show $\exists x.\ x \in_c$ *domain* $\varrho \wedge \varrho \circ_c x = y$
proof(*cases* $w = \mathrm{t}$)
assume $w = \mathrm{t}$
obtain z **where** *z-def*: $z =$ *left-coproj* X $X \circ_c x$
by *simp*
have $\varrho \circ_c z = y$
proof −
have $\varrho \circ_c z = (\varrho \circ_c$ *left-coproj* X $X) \circ_c x$
using *comp-associative2 y-def z-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle id\ X, \mathrm{t} \circ_c \beta_X\rangle \circ_c x$
unfolding *ϱ-def* **using** *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= y$
using ‹$w = \mathrm{t}$› *cart-prod-extract-left y-def* **by** *auto*
finally show *?thesis*.
qed
then show *?thesis*
by (*metis ϱ-type cfunc-type-def codomain-comp domain-comp left-proj-type y-def z-def*)
next
assume $w \neq \mathrm{t}$ **then have** $w = \mathrm{f}$
by (*typecheck-cfuncs*, *meson true-false-only-truth-values y-def*)
obtain z **where** *z-def*: $z =$ *right-coproj* X $X \circ_c x$
by *simp*
have $\varrho \circ_c z = y$
proof −
have $\varrho \circ_c z = (\varrho \circ_c$ *right-coproj* X $X) \circ_c x$

using *comp-associative2 y-def z-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $\langle id\ X, \mathrm{f} \circ_c \beta_X \rangle \circ_c x$
unfolding *ϱ-def* **using** *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = y
using ‹$w = \mathrm{f}$› *cart-prod-extract-left y-def* **by** *auto*
finally show *?thesis*.
qed
then show *?thesis*
by (*metis ϱ-type cfunc-type-def codomain-comp domain-comp right-proj-type y-def z-def*)
qed
qed
then show *?thesis*
by (*metis ϱ-inj ϱ-type epi-mon-is-iso injective-imp-monomorphism is-isomorphic-def surjective-is-epimorphism*)
qed

lemma *oneUone-iso-*Ω:
$\Omega \cong \mathbf{1} \coprod \mathbf{1}$
using *case-bool-def2 case-bool-iso is-isomorphic-def* **by** *auto*

The lemma below is dual to Proposition 2.2.2 in Halvorson.

lemma *card* $\{x.\ x \in_c \Omega \coprod \Omega\} = 4$
proof −

have *f1*: (*left-coproj* Ω Ω) $\circ_c$ t ≠ (*right-coproj* Ω Ω) $\circ_c$ t
by (*typecheck-cfuncs*, *simp add*: *coproducts-disjoint*)
have *f2*: (*left-coproj* Ω Ω) $\circ_c$ t ≠ (*left-coproj* Ω Ω) $\circ_c$ f
by (*typecheck-cfuncs*, *metis cfunc-type-def left-coproj-are-monomorphisms monomorphism-def true-false-distinct*)
have *f3*: (*left-coproj* Ω Ω) $\circ_c$ t ≠ (*right-coproj* Ω Ω) $\circ_c$ f
by (*typecheck-cfuncs*, *simp add*: *coproducts-disjoint*)
have *f4*: (*right-coproj* Ω Ω) $\circ_c$ t ≠ (*left-coproj* Ω Ω) $\circ_c$ f
by (*typecheck-cfuncs*, *metis* (*no-types*) *coproducts-disjoint*)
have *f5*: (*right-coproj* Ω Ω) $\circ_c$ t ≠ (*right-coproj* Ω Ω) $\circ_c$ f
by (*typecheck-cfuncs*, *metis cfunc-type-def monomorphism-def right-coproj-are-monomorphisms true-false-distinct*)
have *f6*: (*left-coproj* Ω Ω) $\circ_c$ f ≠ (*right-coproj* Ω Ω) $\circ_c$ f
by (*typecheck-cfuncs*, *simp add*: *coproducts-disjoint*)

have $\{x.\ x \in_c \Omega \coprod \Omega\}$ = {(*left-coproj* Ω Ω) $\circ_c$ t , (*right-coproj* Ω Ω) $\circ_c$ t, (*left-coproj* Ω Ω) $\circ_c$ f, (*right-coproj* Ω Ω) $\circ_c$ f}
using *coprojs-jointly-surj true-false-only-truth-values*
by (*typecheck-cfuncs*, *auto*)
then show *card* $\{x.\ x \in_c \Omega \coprod \Omega\} = 4$
by (*simp add*: *f1 f2 f3 f4 f5 f6*)
qed

end

10 Axiom of Choice

theory *Axiom-Of-Choice*
imports *Coproduct*
begin

The two definitions below correspond to Definition 2.7.1 in Halvorson.

definition *section-of* :: *cfunc* ⇒ *cfunc* ⇒ *bool* (**infix** *sectionof 90*)
where *s sectionof f* ⟷ *s* : *codomain f* → *domain f* ∧ *f* $\circ_c$ *s* = *id* (*codomain f*)

definition *split-epimorphism* :: *cfunc* ⇒ *bool*
where *split-epimorphism f* ⟷ (∃ *s*. *s* : *codomain f* → *domain f* ∧ *f* $\circ_c$ *s* = *id* (*codomain f*))

lemma *split-epimorphism-def2*:
assumes *f-type*: *f* : *X* → *Y*
assumes *f-split-epic*: *split-epimorphism f*
shows ∃ *s*. (*f* $\circ_c$ *s* = *id Y*) ∧ (*s*: *Y* → *X*)
using *cfunc-type-def f-split-epic f-type split-epimorphism-def* **by** *auto*

lemma *sections-define-splits*:
assumes *s sectionof f*
assumes *s* : *Y* → *X*
shows *f* : *X* → *Y* ∧ *split-epimorphism*(*f*)
using *assms cfunc-type-def section-of-def split-epimorphism-def* **by** *auto*

The axiomatization below corresponds to Axiom 11 (Axiom of Choice) in Halvorson.

axiomatization
where
axiom-of-choice: *epimorphism f* ⟶ (∃ *g* . *g sectionof f*)

lemma *epis-give-monos*:
assumes *f-type*: *f* : *X* → *Y*
assumes *f-epi*: *epimorphism f*
shows ∃ *g*. *g*: *Y* → *X* ∧ *monomorphism g* ∧ *f* $\circ_c$ *g* = *id Y*
using *assms*
by (*typecheck-cfuncs-prems*, *metis axiom-of-choice cfunc-type-def comp-monic-imp-monic f-epi id-isomorphism iso-imp-epi-and-monic section-of-def*)

corollary *epis-are-split*:
assumes *f-type*: *f* : *X* → *Y*
assumes *f-epi*: *epimorphism f*
shows *split-epimorphism f*
using *epis-give-monos cfunc-type-def f-epi split-epimorphism-def* **by** *blast*

The lemma below corresponds to Proposition 2.6.8 in Halvorson.

lemma *monos-give-epis*:
  assumes *f-type*[*type-rule*]: $f : X \to Y$
  assumes *f-mono*: *monomorphism f*
  assumes *X-nonempty*: *nonempty X*
  shows $\exists g.\ g: Y \to X \wedge$ *epimorphism* $g \wedge g \circ_c f = id\ X$
proof −
  obtain *g m E* **where** *g-type*[*type-rule*]: $g : X \to E$ **and** *m-type*[*type-rule*]: $m : E \to Y$ **and**
    g-epi: *epimorphism g* **and** *m-mono*[*type-rule*]: *monomorphism m* **and** *f-eq*: $f = m \circ_c g$
    using *epi-monic-factorization2 f-type* **by** *blast*

  have *g-mono*: *monomorphism g*
  proof (*typecheck-cfuncs*, *unfold monomorphism-def3*, *clarify*)
    fix *x y A*
    assume *x-type*[*type-rule*]: $x : A \to X$ **and** *y-type*[*type-rule*]: $y : A \to X$
    assume $g \circ_c x = g \circ_c y$
    then have $(m \circ_c g) \circ_c x = (m \circ_c g) \circ_c y$
      by (*typecheck-cfuncs*, *smt comp-associative2*)
    then have $f \circ_c x = f \circ_c y$
      unfolding *f-eq* **by** *auto*
    then show $x = y$
      using *f-mono f-type monomorphism-def2 x-type y-type* **by** *blast*
  qed

  have *g-iso*: *isomorphism g*
    by (*simp add*: *epi-mon-is-iso g-epi g-mono*)
  then obtain *g-inv* **where** *g-inv-type*[*type-rule*]: $g\text{-}inv : E \to X$ **and**
      g-g-inv: $g \circ_c g\text{-}inv = id\ E$ **and** *g-inv-g*: $g\text{-}inv \circ_c g = id\ X$
    using *cfunc-type-def g-type isomorphism-def* **by** *auto*

  obtain *x* **where** *x-type*[*type-rule*]: $x \in_c X$
    using *X-nonempty nonempty-def* **by** *blast*

  show $\exists g.\ g: Y \to X \wedge$ *epimorphism* $g \wedge g \circ_c f = id_c\ X$
   proof (*intro exI*[**where** $x = (g\text{-}inv \amalg (x \circ_c \beta_{Y \setminus (E,\ m)})) \circ_c try\text{-}cast\ m$], *safe*, *typecheck-cfuncs*)
    have *func-f-elem-eq*: $\bigwedge y.\ y \in_c X \Longrightarrow (g\text{-}inv \amalg (x \circ_c \beta_{Y \setminus (E,\ m)}) \circ_c try\text{-}cast\ m) \circ_c f \circ_c y = y$
    proof −
      fix *y*
      assume *y-type*[*type-rule*]: $y \in_c X$

      have $(g\text{-}inv \amalg (x \circ_c \beta_{Y \setminus (E,\ m)}) \circ_c try\text{-}cast\ m) \circ_c f \circ_c y$
          $= g\text{-}inv \amalg (x \circ_c \beta_{Y \setminus (E,\ m)}) \circ_c (try\text{-}cast\ m \circ_c m) \circ_c g \circ_c y$
        unfolding *f-eq* **by** (*typecheck-cfuncs*, *smt comp-associative2*)
      also have ... $= (g\text{-}inv \amalg (x \circ_c \beta_{Y \setminus (E,\ m)}) \circ_c left\text{-}coproj\ E\ (Y \setminus (E,m))) \circ_c g \circ_c y$
        by (*typecheck-cfuncs*, *smt comp-associative2 m-mono try-cast-m-m*)

also have ... = (*g-inv* $\circ_c$ *g*) $\circ_c$ *y*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod*)
also have ... = *y*
by (*typecheck-cfuncs*, *simp add*: *g-inv-g id-left-unit2*)
finally show (*g-inv* II ($x \circ_c \beta_{Y \setminus (E,\ m)}$) $\circ_c$ *try-cast m*) $\circ_c$ *f* $\circ_c$ *y* = *y*.
qed
show *epimorphism* (*g-inv* II ($x \circ_c \beta_{Y \setminus (E,\ m)}$) $\circ_c$ *try-cast m*)
proof (*rule surjective-is-epimorphism*, *etcs-subst surjective-def2*, *clarify*)
fix *y*
assume *y-type*[*type-rule*]: $y \in_c X$
show $\exists$ *xa*. $xa \in_c Y$ $\wedge$ (*g-inv* II ($x \circ_c \beta_{Y \setminus (E,\ m)}$) $\circ_c$ *try-cast m*) $\circ_c$ *xa* = *y*
by (*rule exI*[**where** *x*=*f* $\circ_c$ *y*], *typecheck-cfuncs*, *smt func-f-elem-eq*)
qed
show (*g-inv* II ($x \circ_c \beta_{Y \setminus (E,\ m)}$) $\circ_c$ *try-cast m*) $\circ_c$ *f* = id_c *X*
by (*insert comp-associative2 func-f-elem-eq id-left-unit2*, *typecheck-cfuncs*, *rule one-separator*, *auto*)
qed
qed

The lemma below corresponds to Exercise 2.7.2(i) in Halvorson.

lemma *split-epis-are-regular*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
assumes *split-epimorphism f*
shows *regular-epimorphism f*
proof −
obtain *s* **where** *s-type*[*type-rule*]: $s : Y \to X$ **and** *s-splits*: $f \circ_c s$ = *id Y*
by (*meson assms*(*2*) *f-type split-epimorphism-def2*)
then have *coequalizer Y f* ($s \circ_c f$) (*id X*)
unfolding *coequalizer-def*
by (*typecheck-cfuncs*, *smt* (*verit*, *del-insts*) *comp-associative2 comp-type id-left-unit2 id-right-unit2 s-splits*)
then show *?thesis*
using *assms coequalizer-is-epimorphism epimorphisms-are-regular* **by** *blast*
qed

The lemma below corresponds to Exercise 2.7.2(ii) in Halvorson.

lemma *sections-are-regular-monos*:
assumes *s-type*: $s : Y \to X$
assumes *s sectionof f*
shows *regular-monomorphism s*
proof −
have *coequalizer Y f* ($s \circ_c f$) (*id X*)
unfolding *coequalizer-def*
by (*rule exI*[**where** *x*=*X*], *intro exI*[**where** *x*=*X*], *typecheck-cfuncs*,
smt (*z3*) *assms cfunc-type-def comp-associative2 comp-type id-left-unit id-right-unit2 section-of-def*)
then show *?thesis*
by (*metis assms*(*2*) *cfunc-type-def comp-monic-imp-monic′ id-isomorphism iso-imp-epi-and-monic mono-is-regmono section-of-def*)

qed

end

11 Empty Set and Initial Objects

theory *Initial*
  imports *Coproduct*
begin

The axiomatization below corresponds to Axiom 8 (Empty Set) in Halvorson.

axiomatization
  initial-func :: *cset* ⇒ *cfunc* (α_{-} *100*) **and**
  emptyset :: *cset* (∅)
where
  initial-func-type[*type-rule*]: *initial-func* X : ∅ → X **and**
  initial-func-unique: h : ∅ → X ⟹ h = *initial-func* X **and**
  emptyset-is-empty: $\neg(x \in_c \emptyset)$

definition *initial-object* :: *cset* ⇒ *bool* **where**
  initial-object(X) ⟷ (∀ Y. ∃! f. f : X → Y)

lemma *emptyset-is-initial*:
  initial-object(∅)
  using *initial-func-type initial-func-unique initial-object-def* **by** *blast*

lemma *initial-iso-empty*:
  assumes *initial-object*(X)
  shows $X \cong \emptyset$
  by (*metis assms cfunc-type-def comp-type emptyset-is-empty epi-mon-is-iso initial-object-def injective-def injective-imp-monomorphism is-isomorphic-def surjective-def surjective-is-epimorphism*)

The lemma below corresponds to Exercise 2.4.6 in Halvorson.

lemma *coproduct-with-empty*:
  shows $X \coprod \emptyset \cong X$
proof −
  have *comp1*: (*left-coproj* X ∅ $\circ_c$ (*id* X ⨿ α_X)) $\circ_c$ *left-coproj* X ∅ = *left-coproj* X ∅
  proof −
    have (*left-coproj* X ∅ $\circ_c$ (*id* X ⨿ α_X)) $\circ_c$ *left-coproj* X ∅ =
          left-coproj X ∅ $\circ_c$ (*id* X ⨿ α_X $\circ_c$ *left-coproj* X ∅)
      by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
    also have ... = *left-coproj* X ∅ $\circ_c$ *id*(X)
      by (*typecheck-cfuncs*, *metis left-coproj-cfunc-coprod*)
    also have ... = *left-coproj* X ∅
      by (*typecheck-cfuncs*, *metis id-right-unit2*)
    finally show *?thesis*.

qed
have *comp2*: (*left-coproj* X $\emptyset$ $\circ_c$ (*id*(X) $\amalg$ α_X)) $\circ_c$ *right-coproj* X $\emptyset$ = *right-coproj* X $\emptyset$
proof −
have ((*left-coproj* X $\emptyset$) $\circ_c$ (*id*(X) $\amalg$ α_X)) $\circ_c$ (*right-coproj* X $\emptyset$) =
(*left-coproj* X $\emptyset$) $\circ_c$ ((*id*(X) $\amalg$ α_X) $\circ_c$ (*right-coproj* X $\emptyset$))
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = (*left-coproj* X $\emptyset$) $\circ_c$ α_X
by (*typecheck-cfuncs*, *metis right-coproj-cfunc-coprod*)
also have ... = *right-coproj* X $\emptyset$
by (*typecheck-cfuncs*, *metis initial-func-unique*)
finally show *?thesis*.
qed
then have *fact1*: (*left-coproj* X $\emptyset$)$\amalg$(*right-coproj* X $\emptyset$) $\circ_c$ *left-coproj* X $\emptyset$ = *left-coproj* X $\emptyset$
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *blast*)
then have *fact2*: ((*left-coproj* X $\emptyset$)$\amalg$(*right-coproj* X $\emptyset$)) $\circ_c$ (*right-coproj* X $\emptyset$) = *right-coproj* X $\emptyset$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *blast*)
then have *concl*: (*left-coproj* X $\emptyset$) $\circ_c$ (*id*(X) $\amalg$ α_X) = ((*left-coproj* X $\emptyset$)$\amalg$(*right-coproj* X $\emptyset$))
using *cfunc-coprod-unique comp1 comp2* **by** (*typecheck-cfuncs*, *blast*)
also have ... = *id*($X \coprod \emptyset$)
using *cfunc-coprod-unique id-left-unit2* **by** (*typecheck-cfuncs*, *auto*)
then have *isomorphism*(*id*(X) $\amalg$ α_X)
unfolding *isomorphism-def*
by (*intro exI*[**where** *x*=*left-coproj* X $\emptyset$], *typecheck-cfuncs*, *simp add*: *cfunc-type-def concl left-coproj-cfunc-coprod*)
then show $X \coprod \emptyset \cong X$
using *cfunc-coprod-type id-type initial-func-type is-isomorphic-def* **by** *blast*
qed

The lemma below corresponds to Proposition 2.4.7 in Halvorson.

lemma *function-to-empty-is-iso*:
assumes f: $X \to \emptyset$
shows *isomorphism*(f)
by (*metis assms cfunc-type-def comp-type emptyset-is-empty epi-mon-is-iso injective-def injective-imp-monomorphism surjective-def surjective-is-epimorphism*)

lemma *empty-prod-X*:
$\emptyset \times_c X \cong \emptyset$
using *cfunc-type-def function-to-empty-is-iso is-isomorphic-def left-cart-proj-type* **by** *blast*

lemma *X-prod-empty*:
$X \times_c \emptyset \cong \emptyset$
using *cfunc-type-def function-to-empty-is-iso is-isomorphic-def right-cart-proj-type* **by** *blast*

The lemma below corresponds to Proposition 2.4.8 in Halvorson.

lemma *no-el-iff-iso-empty*:
is-empty $X \longleftrightarrow X \cong \emptyset$
proof *safe*
show $X \cong \emptyset \Longrightarrow$ *is-empty* X
by (*meson is-empty-def comp-type emptyset-is-empty is-isomorphic-def*)
next
assume *is-empty* X
obtain f **where** *f-type*: $f: \emptyset \to X$
using *initial-func-type* **by** *blast*

have *f-inj*: *injective*(f)
using *cfunc-type-def emptyset-is-empty f-type injective-def* **by** *auto*
then have *f-mono*: *monomorphism*(f)
using *cfunc-type-def f-type injective-imp-monomorphism* **by** *blast*
have *f-surj*: *surjective*(f)
using *is-empty-def* ‹*is-empty* X› *f-type surjective-def2* **by** *presburger*
then have *epi-f*: *epimorphism*(f)
using *surjective-is-epimorphism* **by** *blast*
then have *iso-f*: *isomorphism*(f)
using *cfunc-type-def epi-mon-is-iso f-mono f-type* **by** *blast*
then show $X \cong \emptyset$
using *f-type is-isomorphic-def isomorphic-is-symmetric* **by** *blast*
qed

lemma *initial-maps-mono*:
assumes *initial-object*(X)
assumes $f : X \to Y$
shows *monomorphism*(f)
by (*metis assms cfunc-type-def initial-iso-empty injective-def injective-imp-monomorphism no-el-iff-iso-empty is-empty-def*)

lemma *iso-empty-initial*:
assumes $X \cong \emptyset$
shows *initial-object* X
unfolding *initial-object-def*
by (*meson assms comp-type is-isomorphic-def isomorphic-is-symmetric isomorphic-is-transitive no-el-iff-iso-empty is-empty-def one-separator terminal-func-type*)

lemma *function-to-empty-set-is-iso*:
assumes $f: X \to Y$
assumes *is-empty* Y
shows *isomorphism* f
by (*metis assms cfunc-type-def comp-type epi-mon-is-iso injective-def injective-imp-monomorphism is-empty-def surjective-def surjective-is-epimorphism*)

lemma *prod-iso-to-empty-right*:
assumes *nonempty* X
assumes $X \times_c Y \cong \emptyset$
shows *is-empty* Y

by (*metis emptyset-is-empty is-empty-def cfunc-prod-type epi-is-surj is-isomorphic-def iso-imp-epi-and-monic isomorphic-is-symmetric nonempty-def surjective-def2 assms*)

lemma *prod-iso-to-empty-left*:
  assumes *nonempty* Y
  assumes $X \times_c Y \cong \emptyset$
  shows *is-empty* X
  by (*meson is-empty-def nonempty-def prod-iso-to-empty-right assms*)

lemma *empty-subset*: $(\emptyset, \alpha_X) \subseteq_c X$
  by (*metis cfunc-type-def emptyset-is-empty initial-func-type injective-def injective-imp-monomorphism subobject-of-def2*)

The lemma below corresponds to Proposition 2.2.1 in Halvorson.

lemma *one-has-two-subsets*:
  $card\ (\{(X,m).\ (X,m) \subseteq_c \mathbf{1}\}//\{((X1,m1),(X2,m2)).\ X1 \cong X2\}) = 2$
proof –
  have *one-subobject*: $(\mathbf{1},\ id\ \mathbf{1}) \subseteq_c \mathbf{1}$
    using *element-monomorphism id-type subobject-of-def2* **by** *blast*
  have *empty-subobject*: $(\emptyset, \alpha_{\mathbf{1}}) \subseteq_c \mathbf{1}$
    by (*simp add*: *empty-subset*)

  have *subobject-one-or-empty*: $\bigwedge X\ m.\ (X,m) \subseteq_c \mathbf{1} \Longrightarrow X \cong \mathbf{1} \vee X \cong \emptyset$
  proof –
    fix $X\ m$
    assume *X-m-subobject*: $(X,\ m) \subseteq_c \mathbf{1}$

    obtain χ **where** χ*-pullback*: *is-pullback* $X\ \mathbf{1}\ \mathbf{1}\ \Omega\ (\beta_X)\ \mathrm{t}\ m\ \chi$
      using *X-m-subobject characteristic-function-exists subobject-of-def2* **by** *blast*
    then have χ*-true-or-false*: $\chi = \mathrm{t} \vee \chi = \mathrm{f}$
      unfolding *is-pullback-def* **using** *true-false-only-truth-values* **by** *auto*

    have *true-iso-one*: $\chi = \mathrm{t} \Longrightarrow X \cong \mathbf{1}$
    proof –
      assume χ*-true*: $\chi = \mathrm{t}$
      then have $\exists!j.\ j \in_c X \wedge \beta_X \circ_c j = id_c\ \mathbf{1} \wedge m \circ_c j = id_c\ \mathbf{1}$
        using χ*-pullback* χ*-true is-pullback-def* **by** (*typecheck-cfuncs*, *auto*)
      then show $X \cong \mathbf{1}$
        using *single-elem-iso-one*
          by (*metis X-m-subobject subobject-of-def2 terminal-func-comp-elem terminal-func-unique*)
    qed

    have *false-iso-one*: $\chi = \mathrm{f} \Longrightarrow X \cong \emptyset$
    proof –
      assume χ*-false*: $\chi = \mathrm{f}$
      have $\nexists\ x.\ x \in_c X$
      proof *clarify*
        fix x

assume *x-in-X*: $x \in_c X$
have $\mathrm{t} \circ_c \beta_X = \mathrm{f} \circ_c m$
using *χ-false χ-pullback is-pullback-def* **by** *auto*
then have $\mathrm{t} \circ_c (\beta_X \circ_c x) = \mathrm{f} \circ_c (m \circ_c x)$
by (*smt X-m-subobject comp-associative2 false-func-type subobject-of-def2 terminal-func-type true-func-type x-in-X*)
then have $\mathrm{t} = \mathrm{f}$
by (*smt X-m-subobject cfunc-type-def comp-type false-func-type id-right-unit id-type subobject-of-def2 terminal-func-unique true-func-type x-in-X*)
then show *False*
using *true-false-distinct* **by** *auto*
qed
then show $X \cong \emptyset$
using *is-empty-def* ‹$\nexists x.\ x \in_c X$› *no-el-iff-iso-empty* **by** *blast*
qed

show $X \cong \mathbf{1} \lor X \cong \emptyset$
using *χ-true-or-false false-iso-one true-iso-one* **by** *blast*
qed

have *classes-distinct*: $\{(X, m).\ X \cong \emptyset\} \neq \{(X, m).\ X \cong \mathbf{1}\}$
by (*metis case-prod-eta curry-case-prod emptyset-is-empty id-isomorphism id-type is-isomorphic-def mem-Collect-eq*)

have $\{(X, m).\ (X, m) \subseteq_c \mathbf{1}\}\ //\ \{((X1, m1), (X2, m2)).\ X1 \cong X2\} = \{\{(X, m).\ X \cong \emptyset\}, \{(X, m).\ X \cong \mathbf{1}\}\}$
proof
show $\{(X, m).\ (X, m) \subseteq_c \mathbf{1}\}\ //\ \{((X1, m1), (X2, m2)).\ X1 \cong X2\} \subseteq \{\{(X, m).\ X \cong \emptyset\}, \{(X, m).\ X \cong \mathbf{1}\}\}$
unfolding *quotient-def* **by** (*auto*, *insert isomorphic-is-symmetric isomorphic-is-transitive subobject-one-or-empty*, *blast+*)
next
show $\{\{(X, m).\ X \cong \emptyset\}, \{(X, m).\ X \cong \mathbf{1}\}\} \subseteq \{(X, m).\ (X, m) \subseteq_c \mathbf{1}\}\ //\ \{((X1, m1), X2, m2).\ X1 \cong X2\}$
unfolding *quotient-def* **by** (*insert empty-subobject one-subobject*, *auto simp add*: *isomorphic-is-symmetric*)
qed
then show *card* $(\{(X, m).\ (X, m) \subseteq_c \mathbf{1}\}\ //\ \{((X, m1), (Y, m2)).\ X \cong Y\}) = 2$
by (*simp add*: *classes-distinct*)
qed

lemma *coprod-with-init-obj1*:
assumes *initial-object* Y
shows $X \coprod Y \cong X$
by (*meson assms coprod-pres-iso coproduct-with-empty initial-iso-empty isomorphic-is-reflexive isomorphic-is-transitive*)

lemma *coprod-with-init-obj2*:
assumes *initial-object* X
shows $X \coprod Y \cong Y$
using *assms coprod-with-init-obj1 coproduct-commutes isomorphic-is-transitive*
by *blast*

lemma *prod-with-term-obj1*:
assumes *terminal-object*(X)
shows $X \times_c Y \cong Y$
by (*meson assms isomorphic-is-reflexive isomorphic-is-transitive one-terminal-object one-x-A-iso-A prod-pres-iso terminal-objects-isomorphic*)

lemma *prod-with-term-obj2*:
assumes *terminal-object*(Y)
shows $X \times_c Y \cong X$
by (*meson assms isomorphic-is-transitive prod-with-term-obj1 product-commutes*)

end

12 Exponential Objects, Transposes and Evaluation

theory *Exponential-Objects*
imports *Initial*
begin

The axiomatization below corresponds to Axiom 9 (Exponential Objects) in Halvorson.

axiomatization
exp-set :: *cset* ⇒ *cset* ⇒ *cset* (-$^{-}$ [*100*,*100*]*100*) **and**
eval-func :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
transpose-func :: *cfunc* ⇒ *cfunc* (-$^{\sharp}$ [*100*]*100*)
where
exp-set-inj: $X^A = Y^B \Longrightarrow X = Y \wedge A = B$ **and**
eval-func-type[*type-rule*]: *eval-func* X A : $A \times_c X^A \rightarrow X$ **and**
transpose-func-type[*type-rule*]: $f : A \times_c Z \rightarrow X \Longrightarrow f^{\sharp} : Z \rightarrow X^A$ **and**
transpose-func-def: $f : A \times_c Z \rightarrow X \Longrightarrow$ (*eval-func* X A) $\circ_c$ (*id* $A \times_f f^{\sharp}$) $= f$ **and**
transpose-func-unique:
$f : A \times_c Z \rightarrow X \Longrightarrow g: Z \rightarrow X^A \Longrightarrow$ (*eval-func* X A) $\circ_c$ (*id* $A \times_f g$) $= f \Longrightarrow g = f^{\sharp}$

lemma *eval-func-surj*:
assumes *nonempty*(A)
shows *surjective*((*eval-func* X A))
unfolding *surjective-def*
proof(*clarify*)
fix x

assume *x-type*: $x \in_c$ *codomain* (*eval-func X A*)
then have *x-type2*[*type-rule*]: $x \in_c X$
  using *cfunc-type-def eval-func-type* **by** *auto*
obtain *a* **where** *a-def*[*type-rule*]: $a \in_c A$
  using *assms nonempty-def* **by** *auto*
have *needed-type*: $\langle a, (x \circ_c$ *right-cart-proj A* $\mathbf{1})^\sharp\rangle \in_c$ *domain* (*eval-func X A*)
  using *cfunc-type-def* **by** (*typecheck-cfuncs*, *auto*)
have (*eval-func X A*) $\circ_c$ $\langle a, (x \circ_c$ *right-cart-proj A* $\mathbf{1})^\sharp\rangle =$
    (*eval-func X A*) $\circ_c$ $((id(A) \times_f (x \circ_c$ *right-cart-proj A* $\mathbf{1})^\sharp) \circ_c \langle a, id(\mathbf{1})\rangle)$
  by (*typecheck-cfuncs*, *smt a-def cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2 x-type2*)
also have ... $= (($*eval-func X A*$) \circ_c (id(A) \times_f (x \circ_c$ *right-cart-proj A* $\mathbf{1})^\sharp)) \circ_c \langle a, id(\mathbf{1})\rangle$
  by (*typecheck-cfuncs*, *meson a-def comp-associative2 x-type2*)
also have ... $= (x \circ_c$ *right-cart-proj A* $\mathbf{1}) \circ_c \langle a, id(\mathbf{1})\rangle$
  by (*metis comp-type right-cart-proj-type transpose-func-def x-type2*)
also have ... $= x \circ_c ($*right-cart-proj A* $\mathbf{1} \circ_c \langle a, id(\mathbf{1})\rangle)$
  using *a-def cfunc-type-def comp-associative x-type2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= x$
  using *a-def id-right-unit2 right-cart-proj-cfunc-prod x-type2* **by** (*typecheck-cfuncs*, *auto*)
ultimately show $\exists\, y.\ y \in_c$ *domain* (*eval-func X A*) $\wedge$ *eval-func X A* $\circ_c y = x$
  using *needed-type* **by** (*typecheck-cfuncs*, *auto*)
qed

The lemma below corresponds to a note above Definition 2.5.1 in Halvorson.

lemma *exponential-object-identity*:
  (*eval-func X A*)$^\sharp = id_c(X^A)$
  by (*metis cfunc-type-def eval-func-type id-cross-prod id-right-unit id-type transpose-func-unique*)

lemma *eval-func-X-empty-injective*:
  assumes *is-empty Y*
  shows *injective* (*eval-func X Y*)
  unfolding *injective-def*
  by (*typecheck-cfuncs*,*metis assms cfunc-type-def comp-type left-cart-proj-type is-empty-def*)

12.1 Lifting Functions

The definition below corresponds to Definition 2.5.1 in Halvorson.

definition *exp-func* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ((-)$^-{}_f$ [*100*,*100*]*100*) **where**
  exp-func g A = ($g \circ_c$ *eval-func* (*domain g*) *A*)$^\sharp$

lemma *exp-func-def2*:
  assumes $g : X \to Y$
  shows *exp-func g A* = ($g \circ_c$ *eval-func X A*)$^\sharp$
  using *assms cfunc-type-def exp-func-def* **by** *auto*

lemma *exp-func-type*[*type-rule*]:
  assumes $g : X \to Y$
  shows $g^A{}_f : X^A \to Y^A$
  using *assms* **by** (*unfold exp-func-def2*, *typecheck-cfuncs*)

lemma *exp-of-id-is-id-of-exp*:
  $id(X^A) = (id(X))^A{}_f$
  by (*metis* (*no-types*) *eval-func-type exp-func-def exponential-object-identity id-domain id-left-unit2*)

The lemma below corresponds to a note below Definition 2.5.1 in Halvorson.

lemma *exponential-square-diagram*:
  assumes $g : Y \to Z$
  shows $(eval\text{-}func\ Z\ A) \circ_c (id_c(A) \times_f g^A{}_f) = g \circ_c (eval\text{-}func\ Y\ A)$
  using *assms* **by** (*typecheck-cfuncs*, *simp add*: *exp-func-def2 transpose-func-def*)

The lemma below corresponds to Proposition 2.5.2 in Halvorson.

lemma *transpose-of-comp*:
  assumes *f-type*: f: $A \times_c X \to Y$ **and** *g-type*: g: $Y \to Z$
  shows f: $A \times_c X \to Y \wedge g$: $Y \to Z \implies (g \circ_c f)^\sharp = g^A{}_f \circ_c f^\sharp$
proof *clarify*
  have *left-eq*: $(eval\text{-}func\ Z\ A) \circ_c (id(A) \times_f (g \circ_c f)^\sharp) = g \circ_c f$
    using *comp-type f-type g-type transpose-func-def* **by** *blast*
  have *right-eq*: $(eval\text{-}func\ Z\ A) \circ_c (id_c\ A \times_f (g^A{}_f \circ_c f^\sharp)) = g \circ_c f$
  proof −
    have $(eval\text{-}func\ Z\ A) \circ_c (id_c\ A \times_f (g^A{}_f \circ_c f^\sharp)) =$
        $(eval\text{-}func\ Z\ A) \circ_c ((id_c\ A \times_f (g^A{}_f)) \circ_c (id_c\ A \times_f f^\sharp))$
      by (*typecheck-cfuncs*, *smt identity-distributes-across-composition assms*)
    also have ... $= (g \circ_c eval\text{-}func\ Y\ A) \circ_c (id_c\ A \times_f f^\sharp)$
      by (*typecheck-cfuncs*, *smt comp-associative2 exp-func-def2 transpose-func-def assms*)
    also have ... $= g \circ_c f$
      by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 transpose-func-def assms*)
    finally show *?thesis*.
  qed
  show $(g \circ_c f)^\sharp = g^A{}_f \circ_c f^\sharp$
    using *assms* **by** (*typecheck-cfuncs*, *metis right-eq transpose-func-unique*)
qed

lemma *exponential-object-identity2*:
  $id(X)^A{}_f = id_c(X^A)$
  by (*metis eval-func-type exp-func-def exponential-object-identity id-domain id-left-unit2*)

The lemma below corresponds to comments below Proposition 2.5.2 and above Definition 2.5.3 in Halvorson.

lemma *eval-of-id-cross-id-sharp1*:
  $(eval\text{-}func\ (A \times_c X)\ A) \circ_c (id(A) \times_f (id(A \times_c X))^\sharp) = id(A \times_c X)$

using *id-type transpose-func-def* **by** *blast*
lemma *eval-of-id-cross-id-sharp2*:
assumes $a : Z \to A$ $x : Z \to X$
shows $((\mathit{eval\text{-}func}\ (A \times_c X)\ A) \circ_c (id(A) \times_f (id(A \times_c X))^\sharp)) \circ_c \langle a,x\rangle = \langle a,x\rangle$
by (*smt assms cfunc-cross-prod-comp-cfunc-prod eval-of-id-cross-id-sharp1 id-cross-prod id-left-unit2 id-type*)

lemma *transpose-factors*:
assumes f: $X \to Y$
assumes g: $Y \to Z$
shows $(g \circ_c f)^A{}_f = (g^A{}_f) \circ_c (f^A{}_f)$
using *assms* **by** (*typecheck-cfuncs*, *smt comp-associative2 comp-type eval-func-type exp-func-def2 transpose-of-comp*)

12.2 Inverse Transpose Function (flat)

The definition below corresponds to Definition 2.5.3 in Halvorson.

definition *inv-transpose-func* :: *cfunc* ⇒ *cfunc* ($\text{-}^\flat$ [*100*]*100*) **where**
$f^\flat = (\mathit{THE}\ g.\ \exists\ Z\ X\ A.\ \mathit{domain}\ f = Z \wedge \mathit{codomain}\ f = X^A \wedge g = (\mathit{eval\text{-}func}\ X\ A) \circ_c (\mathit{id}\ A \times_f f))$

lemma *inv-transpose-func-def2*:
assumes $f : Z \to X^A$
shows $\exists\ Z\ X\ A.\ \mathit{domain}\ f = Z \wedge \mathit{codomain}\ f = X^A \wedge f^\flat = (\mathit{eval\text{-}func}\ X\ A) \circ_c (\mathit{id}\ A \times_f f)$
unfolding *inv-transpose-func-def*
proof (*rule theI*)
show $\exists Z\ Y\ B.\ \mathit{domain}\ f = Z \wedge \mathit{codomain}\ f = Y^B \wedge \mathit{eval\text{-}func}\ X\ A \circ_c \mathit{id}_c\ A \times_f f = \mathit{eval\text{-}func}\ Y\ B \circ_c \mathit{id}_c\ B \times_f f$
using *assms cfunc-type-def* **by** *blast*
next
fix g
assume $\exists Z\ X\ A.\ \mathit{domain}\ f = Z \wedge \mathit{codomain}\ f = X^A \wedge g = \mathit{eval\text{-}func}\ X\ A \circ_c \mathit{id}_c\ A \times_f f$
then show $g = \mathit{eval\text{-}func}\ X\ A \circ_c \mathit{id}_c\ A \times_f f$
by (*metis assms cfunc-type-def exp-set-inj*)
qed

lemma *inv-transpose-func-def3*:
assumes *f-type*: $f : Z \to X^A$
shows $f^\flat = (\mathit{eval\text{-}func}\ X\ A) \circ_c (\mathit{id}\ A \times_f f)$
by (*metis cfunc-type-def exp-set-inj f-type inv-transpose-func-def2*)

lemma *flat-type*[*type-rule*]:
assumes *f-type*[*type-rule*]: $f : Z \to X^A$
shows $f^\flat : A \times_c Z \to X$
by (*etcs-subst inv-transpose-func-def3*, *typecheck-cfuncs*)

The lemma below corresponds to Proposition 2.5.4 in Halvorson.

lemma *inv-transpose-of-composition*:
assumes f: $X \to Y$ g: $Y \to Z^A$
shows $(g \circ_c f)^\flat = g^\flat \circ_c (id(A) \times_f f)$
using *assms comp-associative2 identity-distributes-across-composition*
by ((*etcs-subst inv-transpose-func-def3*)+, *typecheck-cfuncs*, *auto*)

The lemma below corresponds to Proposition 2.5.5 in Halvorson.

lemma *flat-cancels-sharp*:
$f : A \times_c Z \to X \implies (f^\sharp)^\flat = f$
using *inv-transpose-func-def3 transpose-func-def transpose-func-type* **by** *fastforce*

The lemma below corresponds to Proposition 2.5.6 in Halvorson.

lemma *sharp-cancels-flat*:
f: $Z \to X^A \implies (f^\flat)^\sharp = f$
proof –
assume *f-type*: $f : Z \to X^A$
then have *uniqueness*: $\forall$ g. $g : Z \to X^A \longrightarrow$ *eval-func* X A $\circ_c$ (id $A \times_f g$) $=$ $f^\flat \longrightarrow g = (f^\flat)^\sharp$
by (*typecheck-cfuncs*, *simp add*: *transpose-func-unique*)
have *eval-func* X A $\circ_c$ (id $A \times_f f$) $= f^\flat$
by (*metis f-type inv-transpose-func-def3*)
then show $f^{\flat\sharp} = f$
using *f-type uniqueness* **by** *auto*
qed

lemma *same-evals-equal*:
assumes $f : Z \to X^A$ g: $Z \to X^A$
shows *eval-func* X A $\circ_c$ (id $A \times_f f$) $=$ *eval-func* X A $\circ_c$ (id $A \times_f g$) $\implies f = g$
by (*metis assms inv-transpose-func-def3 sharp-cancels-flat*)

lemma *sharp-comp*:
assumes *f-type*[*type-rule*]: $f : A \times_c Z \to X$ **and** *g-type*[*type-rule*]: $g : W \to Z$
shows $f^\sharp \circ_c g = (f \circ_c (id\ A \times_f g))^\sharp$
proof (*etcs-rule same-evals-equal*[**where** X=X, **where** A=A])

have *eval-func* X A $\circ_c$ (id $A \times_f (f^\sharp \circ_c g)$) $=$ *eval-func* X A $\circ_c$ (id $A \times_f f^\sharp$) $\circ_c$ (id $A \times_f g$)
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *identity-distributes-across-composition*)
also have ... $= f \circ_c (id\ A \times_f g)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 transpose-func-def*)
also have ... $=$ *eval-func* X A $\circ_c$ (id_c $A \times_f (f \circ_c (id_c\ A \times_f g))^\sharp$)
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *transpose-func-def*)
finally show *eval-func* X A $\circ_c$ (id $A \times_f (f^\sharp \circ_c g)$) $=$ *eval-func* X A $\circ_c$ (id_c A $\times_f (f \circ_c (id_c\ A \times_f g))^\sharp$).
qed

lemma *flat-pres-epi*:
assumes *nonempty*(A)
assumes $f : Z \to X^A$

assumes *epimorphism f*
shows *epimorphism*($f^\flat$)
proof −
have *equals*: $f^\flat$ = (*eval-func X A*) $\circ_c$ (*id*(*A*) $\times_f$ *f*)
using *assms*(*2*) *inv-transpose-func-def3* **by** *auto*
have *idA-f-epi*: *epimorphism*((*id*(*A*) $\times_f$ *f*))
using *assms*(*2*) *assms*(*3*) *cfunc-cross-prod-surj epi-is-surj id-isomorphism id-type iso-imp-epi-and-monic surjective-is-epimorphism* **by** *blast*
have *eval-epi*: *epimorphism*((*eval-func X A*))
by (*simp add*: *assms*(*1*) *eval-func-surj surjective-is-epimorphism*)
have *codomain* ((*id*(*A*) $\times_f$ *f*)) = *domain* ((*eval-func X A*))
using *assms*(*2*) *cfunc-type-def* **by** (*typecheck-cfuncs*, *auto*)
then show *?thesis*
by (*simp add*: *composition-of-epi-pair-is-epi equals eval-epi idA-f-epi*)
qed

lemma *transpose-inj-is-inj*:
assumes *g*: *X* → *Y*
assumes *injective g*
shows *injective*($g^A{}_f$)
unfolding *injective-def*
proof(*clarify*)
fix *x y*
assume *x-type*[*type-rule*]: *x* $\in_c$ *domain* ($g^A{}_f$)
assume *y-type*[*type-rule*]:*y* $\in_c$ *domain* ($g^A{}_f$)
assume *eqs*: $g^A{}_f \circ_c x = g^A{}_f \circ_c y$
have *mono-g*: *monomorphism g*
by (*meson CollectI assms*(*2*) *injective-imp-monomorphism*)
have *x-type*′[*type-rule*]: *x* $\in_c$ X^A
using *assms*(*1*) *cfunc-type-def exp-func-type* **by** (*typecheck-cfuncs*, *force*)
have *y-type*′[*type-rule*]: *y* $\in_c$ X^A
using *cfunc-type-def x-type x-type*′ *y-type* **by** *presburger*
have ($g \circ_c$ *eval-func X A*)$^\sharp$ $\circ_c$ *x* = ($g \circ_c$ *eval-func X A*)$^\sharp$ $\circ_c$ *y*
unfolding *exp-func-def* **using** *assms eqs exp-func-def2* **by** *force*
then have $g \circ_c$ (*eval-func X A* $\circ_c$(*id*(*A*) $\times_f$ *x*)) = $g \circ_c$ (*eval-func X A* $\circ_c$ (*id*(*A*) $\times_f$ *y*))
by (*smt* (*z3*) *assms*(*1*) *comp-type eqs flat-cancels-sharp flat-type inv-transpose-func-def3 sharp-cancels-flat transpose-of-comp x-type*′ *y-type*′)
then have *eval-func X A* $\circ_c$(*id*(*A*) $\times_f$ *x*) = *eval-func X A* $\circ_c$ (*id*(*A*) $\times_f$ *y*)
by (*metis assms*(*1*) *mono-g flat-type inv-transpose-func-def3 monomorphism-def2 x-type*′ *y-type*′)
then show *x* = *y*
by (*meson same-evals-equal x-type*′ *y-type*′)
qed

lemma *eval-func-X-one-injective*:
injective (*eval-func X* **1**)
proof (*cases* ∃ *x*. *x* $\in_c$ *X*)
assume ∃*x*. *x* $\in_c$ *X*

then obtain x **where** *x-type*: $x \in_c X$
by *auto*
then have *eval-func* X $\mathbf{1} \circ_c id_c$ $\mathbf{1} \times_f (x \circ_c \beta_{\mathbf{1} \times_c \mathbf{1}})^\sharp = x \circ_c \beta_{\mathbf{1} \times_c \mathbf{1}}$
using *comp-type terminal-func-type transpose-func-def* **by** *blast*

show *injective* (*eval-func* X $\mathbf{1}$)
unfolding *injective-def*
proof *clarify*
fix a b
assume *a-type*: $a \in_c$ *domain* (*eval-func* X $\mathbf{1}$)
assume *b-type*: $b \in_c$ *domain* (*eval-func* X $\mathbf{1}$)
assume *evals-equal*: *eval-func* X $\mathbf{1} \circ_c a =$ *eval-func* X $\mathbf{1} \circ_c b$

have *eval-dom*: *domain*(*eval-func* X $\mathbf{1}$) $= \mathbf{1} \times_c (X^{\mathbf{1}})$
using *cfunc-type-def eval-func-type* **by** *auto*

obtain A **where** *a-def*: $A \in_c X^{\mathbf{1}} \wedge a = \langle id\ \mathbf{1}, A\rangle$
by (*typecheck-cfuncs*, *metis a-type cart-prod-decomp eval-dom terminal-func-unique*)

obtain B **where** *b-def*: $B \in_c X^{\mathbf{1}} \wedge b = \langle id\ \mathbf{1}, B\rangle$
by (*typecheck-cfuncs*, *metis b-type cart-prod-decomp eval-dom terminal-func-unique*)

have $A^\flat \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle = B^\flat \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle$
proof −
have $A^\flat \circ_c \langle id\ \mathbf{1}\ , id\ \mathbf{1}\rangle = ($*eval-func* X $\mathbf{1}) \circ_c (id\ \mathbf{1} \times_f (A^\flat)^\sharp) \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *a-def comp-associative2 inv-transpose-func-def3 sharp-cancels-flat*)
also have ... = *eval-func* X $\mathbf{1} \circ_c a$
using *a-def cfunc-cross-prod-comp-cfunc-prod id-right-unit2 sharp-cancels-flat* **by** (*typecheck-cfuncs*, *force*)
also have ... = *eval-func* X $\mathbf{1} \circ_c b$
by (*simp add*: *evals-equal*)
also have ... $= ($*eval-func* X $\mathbf{1}) \circ_c (id\ \mathbf{1} \times_f (B^\flat)^\sharp) \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle$
using *b-def cfunc-cross-prod-comp-cfunc-prod id-right-unit2 sharp-cancels-flat* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= B^\flat \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *smt* (*verit*) *b-def comp-associative2 inv-transpose-func-def3 sharp-cancels-flat*)
finally show $A^\flat \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle = B^\flat \circ_c \langle id\ \mathbf{1}, id\ \mathbf{1}\rangle$.
qed
then have $A^\flat = B^\flat$
by (*typecheck-cfuncs*, *smt swap-def a-def b-def cfunc-prod-comp comp-associative2 diagonal-def diagonal-type id-right-unit2 id-type left-cart-proj-type right-cart-proj-type swap-idempotent swap-type terminal-func-comp terminal-func-unique*)
then have $A = B$
by (*metis a-def b-def sharp-cancels-flat*)
then show $a = b$
by (*simp add*: *a-def b-def*)
qed

next
  assume $\nexists x.\ x \in_c X$
  then show *injective* (*eval-func* X **1**)
    by (*typecheck-cfuncs*, *metis cfunc-type-def comp-type injective-def*)
qed

In the lemma below, the nonempty assumption is required. Consider, for example, $X = \Omega$ and $A = \emptyset$

lemma *sharp-pres-mono*:
  assumes $f : A \times_c Z \to X$
  assumes *monomorphism*(f)
  assumes *nonempty* A
  shows *monomorphism*($f^\sharp$)
  unfolding *monomorphism-def2*
proof(*clarify*)
  fix $g\ h\ U\ Y\ x$
  assume *g-type*[*type-rule*]: $g : U \to Y$
  assume *h-type*[*type-rule*]: $h : U \to Y$
  assume *f-sharp-type*[*type-rule*]: $f^\sharp : Y \to x$
  assume *equals*: $f^\sharp \circ_c g = f^\sharp \circ_c h$

  have *f-sharp-type2*: $f^\sharp : Z \to X^A$
    by (*simp add*: *assms*(*1*) *transpose-func-type*)
  have *Y-is-Z*: $Y = Z$
    using *cfunc-type-def f-sharp-type f-sharp-type2* **by** *auto*
  have *x-is-XA*: $x = X^A$
    using *cfunc-type-def f-sharp-type f-sharp-type2* **by** *auto*
  have *g-type2*: $g : U \to Z$
    using *Y-is-Z g-type* **by** *blast*
  have *h-type2*: $h : U \to Z$
    using *Y-is-Z h-type* **by** *blast*
  have *idg-type*: $(id(A) \times_f g) : A \times_c U \to A \times_c Z$
    by (*simp add*: *cfunc-cross-prod-type g-type2 id-type*)
  have *idh-type*: $(id(A) \times_f h) : A \times_c U \to A \times_c Z$
    by (*simp add*: *cfunc-cross-prod-type h-type2 id-type*)

   then have *epic*: *epimorphism*(*right-cart-proj* $A\ U$)
     using *assms*(*3*) *nonempty-left-imp-right-proj-epimorphism* **by** *blast*

   have *fIdg-is-fIdh*: $f \circ_c (id(A) \times_f g) = f \circ_c (id(A) \times_f h)$
   proof −
    have $f \circ_c (id(A) \times_f g) = (\textit{eval-func}\ X\ A \circ_c (id(A) \times_f f^\sharp)) \circ_c (id(A) \times_f g)$
      using *assms*(*1*) *transpose-func-def* **by** *auto*
    also have ... $= (\textit{eval-func}\ X\ A \circ_c (id(A) \times_f f^\sharp)) \circ_c (id(A) \times_f h)$
       by (*metis Y-is-Z equals f-sharp-type2 g-type h-type inv-transpose-func-def3 inv-transpose-of-composition*)
    also have ... $= f \circ_c (id(A) \times_f h)$
      using *assms*(*1*) *transpose-func-def* **by** *auto*
    finally show *?thesis*.

qed
then have *idg-is-idh*: $(id(A) \times_f g) = (id(A) \times_f h)$
using *assms fIdg-is-fIdh idg-type idh-type monomorphism-def3* **by** *blast*
then have $g \circ_c$ (*right-cart-proj A U*) $= h \circ_c$ (*right-cart-proj A U*)
by (*smt g-type2 h-type2 id-type right-cart-proj-cfunc-cross-prod*)
then show $g = h$
using *epic epimorphism-def2 g-type2 h-type2 right-cart-proj-type* **by** *blast*
qed

12.3 Metafunctions and their Inverses (Cnufatems)

12.3.1 Metafunctions

definition *metafunc* :: *cfunc* ⇒ *cfunc* **where**
metafunc f ≡ $(f \circ_c$ (*left-cart-proj* (*domain f*) $\mathbf{1}))^\sharp$

lemma *metafunc-def2*:
assumes $f : X \to Y$
shows *metafunc f* $= (f \circ_c$ (*left-cart-proj X* $\mathbf{1}))^\sharp$
using *assms* **unfolding** *metafunc-def cfunc-type-def* **by** *auto*

lemma *metafunc-type*[*type-rule*]:
assumes $f : X \to Y$
shows *metafunc f* $\in_c Y^X$
using *assms* **by** (*unfold metafunc-def2*, *typecheck-cfuncs*)

lemma *eval-lemma*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
assumes *x-type*[*type-rule*]: $x \in_c X$
shows *eval-func Y X* $\circ_c \langle x,$ *metafunc f*$\rangle = f \circ_c x$
proof −
have *eval-func Y X* $\circ_c \langle x,$ *metafunc f*$\rangle$ = *eval-func Y X* $\circ_c$ (*id X* $\times_f (f \circ_c$ (*left-cart-proj X* $\mathbf{1}))^\sharp) \circ_c \langle x,$ *id* $\mathbf{1}\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2 metafunc-def2*)
also have ... = (*eval-func Y X* $\circ_c$ (*id X* $\times_f (f \circ_c$ (*left-cart-proj X* $\mathbf{1}))^\sharp)) \circ_c \langle x,$ *id* $\mathbf{1}\rangle$
using *comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= (f \circ_c$ (*left-cart-proj X* $\mathbf{1})) \circ_c \langle x,$ *id* $\mathbf{1}\rangle$
by (*typecheck-cfuncs*, *metis transpose-func-def*)
also have ... $= f \circ_c x$
by (*typecheck-cfuncs*, *metis assms cfunc-type-def comp-associative left-cart-proj-cfunc-prod*)
finally show *eval-func Y X* $\circ_c \langle x,$ *metafunc f*$\rangle = f \circ_c x$.
qed

12.3.2 Inverse Metafunctions (Cnufatems)

definition *cnufatem* :: *cfunc* ⇒ *cfunc* **where**
cnufatem f = (*THE g*. ∀ *Y X*. $f : \mathbf{1} \to Y^X \longrightarrow g$ = *eval-func Y X* $\circ_c \langle$*id X*, $f \circ_c \beta_X\rangle$)

lemma *cnufatem-def2*:
  assumes $f \in_c Y^X$
  shows *cnufatem f* = *eval-func* $Y\ X \circ_c \langle id\ X, f \circ_c \beta_X \rangle$
  using *assms* **unfolding** *cnufatem-def cfunc-type-def*
  by (*smt* (*verit*, *ccfv-threshold*) *exp-set-inj theI'*)

lemma *cnufatem-type*[*type-rule*]:
  assumes $f \in_c Y^X$
  shows *cnufatem f* : $X \to Y$
  using *assms cnufatem-def2*
  by (*auto*, *typecheck-cfuncs*)

lemma *cnufatem-metafunc*:
  assumes *f-type*[*type-rule*]: $f : X \to Y$
  shows *cnufatem* (*metafunc f*) = *f*
proof(*etcs-rule one-separator*)
  fix *x*
  assume *x-type*[*type-rule*]: $x \in_c X$

  have *cnufatem* (*metafunc f*) $\circ_c x =$ *eval-func* $Y\ X \circ_c \langle id\ X$, (*metafunc f*) $\circ_c$
$\beta_X \rangle \circ_c x$
    using *cnufatem-def2 comp-associative2* **by** (*typecheck-cfuncs*, *fastforce*)
  also have ... = *eval-func* $Y\ X \circ_c \langle x$, (*metafunc f*)$\rangle$
    by (*typecheck-cfuncs*, *metis cart-prod-extract-left*)
  also have ... = $f \circ_c x$
    using *eval-lemma* **by** (*typecheck-cfuncs*, *presburger*)
  finally show *cnufatem* (*metafunc f*) $\circ_c x = f \circ_c x$.
qed

lemma *metafunc-cnufatem*:
  assumes *f-type*[*type-rule*]: $f \in_c Y^X$
  shows *metafunc* (*cnufatem f*) = *f*
proof (*etcs-rule same-evals-equal*[**where** $X = Y$, **where** $A = X$], *etcs-rule one-separator*)
  fix *x1*
  assume *x1-type*[*type-rule*]: $x1 \in_c X \times_c \mathbf{1}$
  then obtain *x* **where** *x-type*[*type-rule*]: $x \in_c X$ **and** *x-def*: $x1 = \langle x, id\ \mathbf{1} \rangle$
    by (*typecheck-cfuncs*, *metis cart-prod-decomp one-unique-element*)
  have (*eval-func* $Y\ X \circ_c id_c\ X \times_f$ *metafunc* (*cnufatem f*)) $\circ_c \langle x, id\ \mathbf{1} \rangle =$
        eval-func $Y\ X \circ_c \langle x$, *metafunc* (*cnufatem f*)$\rangle$
   by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod comp-associative2*
id-left-unit2 id-right-unit2)
  also have ... = (*cnufatem f*) $\circ_c x$
    using *eval-lemma* **by** (*typecheck-cfuncs*, *presburger*)
  also have ... = (*eval-func* $Y\ X \circ_c \langle id\ X, f \circ_c \beta_X \rangle) \circ_c x$
    using *assms cnufatem-def2* **by** *presburger*
  also have ... = *eval-func* $Y\ X \circ_c \langle id\ X, f \circ_c \beta_X \rangle \circ_c x$
    by (*typecheck-cfuncs*, *metis comp-associative2*)
  also have ... = *eval-func* $Y\ X \circ_c \langle id\ X \circ_c x\ , f \circ_c (\beta_X \circ_c x) \rangle$

by (*typecheck-cfuncs*, *metis cart-prod-extract-left id-left-unit2 id-right-unit2 terminal-func-comp-elem*)
also have ... = *eval-func* $Y\ X \circ_c \langle id\ X \circ_c x\ ,\ f \circ_c id\ \mathbf{1}\rangle$
by (*simp add*: *terminal-func-comp-elem x-type*)
also have ... = *eval-func* $Y\ X \circ_c (id_c\ X \times_f f) \circ_c \langle x,\ id\ \mathbf{1}\rangle$
using *cfunc-cross-prod-comp-cfunc-prod* **by** (*typecheck-cfuncs*, *force*)
also have ... = (*eval-func* $Y\ X \circ_c id_c\ X \times_f f) \circ_c x1$
by (*typecheck-cfuncs*, *metis comp-associative2 x-def*)
ultimately show (*eval-func* $Y\ X \circ_c id_c\ X \times_f$ *metafunc* (*cnufatem* f)) $\circ_c x1$ = (*eval-func* $Y\ X \circ_c id_c\ X \times_f f) \circ_c x1$
using *x-def* **by** *simp*
qed

12.3.3 Metafunction Composition

definition *meta-comp* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
meta-comp $X\ Y\ Z$ = (*eval-func* $Z\ Y \circ_c$ *swap* $(Z^Y)\ Y \circ_c (id(Z^Y) \times_f$ (*eval-func* $Y\ X \circ_c$ *swap* $(Y^X)\ X)) \circ_c$ (*associate-right* $(Z^Y)\ (Y^X)\ X) \circ_c$ *swap* $X\ ((Z^Y) \times_c (Y^X)))^\sharp$

lemma *meta-comp-type*[*type-rule*]:
meta-comp $X\ Y\ Z : Z^Y \times_c Y^X \rightarrow Z^X$
unfolding *meta-comp-def* **by** *typecheck-cfuncs*

definition *meta-comp2* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (**infixr** □ *55*)
where *meta-comp2* $f\ g$ = (*THE* $h.\ \exists\ W\ X\ Y.\ g : W \rightarrow Y^X \wedge h = (f^\flat \circ_c \langle g^\flat,$ *right-cart-proj* $X\ W\rangle)^\sharp)$

lemma *meta-comp2-def2*:
assumes f: $W \rightarrow Z^Y$
assumes g: $W \rightarrow Y^X$
shows $f \square g = (f^\flat \circ_c \langle g^\flat,$ *right-cart-proj* $X\ W\rangle)^\sharp$
using *assms* **unfolding** *meta-comp2-def*
by (*smt* (*z3*) *cfunc-type-def exp-set-inj the-equality*)

lemma *meta-comp2-type*[*type-rule*]:
assumes f: $W \rightarrow Z^Y$
assumes g: $W \rightarrow Y^X$
shows $f \square g : W \rightarrow Z^X$
proof −
have $(f^\flat \circ_c \langle g^\flat,$ *right-cart-proj* $X\ W\rangle)^\sharp : W \rightarrow Z^X$
using *assms* **by** *typecheck-cfuncs*
then show *?thesis*
using *assms* **by** (*simp add*: *meta-comp2-def2*)
qed

lemma *meta-comp2-elements-aux*:
assumes $f \in_c Z^Y$
assumes $g \in_c Y^X$

assumes $x \in_c X$
shows $(f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle) \circ_c \langle x, id_c\ \mathbf{1}\rangle = \textit{eval-func } Z\ Y \circ_c \langle \textit{eval-func } Y\ X \circ_c \langle x,g\rangle,f\rangle$
proof−
have $(f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle) \circ_c \langle x, id_c\ \mathbf{1}\rangle = f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle \circ_c \langle x, id_c\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... $= f^\flat \circ_c \langle g^\flat \circ_c \langle x, id_c\ \mathbf{1}\rangle, \textit{right-cart-proj } X\ \mathbf{1} \circ_c \langle x, id_c\ \mathbf{1}\rangle \rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp*)
also have ... $= f^\flat \circ_c \langle g^\flat \circ_c \langle x, id_c\ \mathbf{1}\rangle, id_c\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis one-unique-element*)
also have ... $= f^\flat \circ_c \langle(\textit{eval-func } Y\ X) \circ_c (id\ X \times_f g) \circ_c \langle x, id_c\ \mathbf{1}\rangle, id_c\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 inv-transpose-func-def3*)
also have ... $= f^\flat \circ_c \langle(\textit{eval-func } Y\ X) \circ_c \langle x, g\rangle, id_c\ \mathbf{1}\rangle$
using *assms cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2* **by** (*typecheck-cfuncs*,*force*)
also have ... $= (\textit{eval-func } Z\ Y) \circ_c (id\ Y \times_f f) \circ_c \langle(\textit{eval-func } Y\ X) \circ_c \langle x, g\rangle, id_c\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 inv-transpose-func-def3*)
also have ... $= (\textit{eval-func } Z\ Y) \circ_c \langle(\textit{eval-func } Y\ X) \circ_c \langle x, g\rangle, f\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
finally show $(f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle) \circ_c \langle x, id_c\ \mathbf{1}\rangle = \textit{eval-func } Z\ Y \circ_c \langle \textit{eval-func } Y\ X \circ_c \langle x,g\rangle,f\rangle$.
qed

lemma *meta-comp2-def3*:
assumes $f \in_c Z^Y$
assumes $g \in_c Y^X$
shows $f \square g = \textit{metafunc } ((\textit{cnufatem } f) \circ_c (\textit{cnufatem } g))$
using *assms*
proof(*unfold meta-comp2-def2 cnufatem-def2 metafunc-def meta-comp-def*)
have $f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle = ((\textit{eval-func } Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle) \circ_c \textit{eval-func } Y\ X \circ_c \langle id_c\ X, g \circ_c \beta_X\rangle) \circ_c \textit{left-cart-proj } X\ \mathbf{1}$
proof(*rule one-separator*[**where** $X = X \times_c \mathbf{1}$, **where** $Y = Z$])
show $f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle : X \times_c \mathbf{1} \to Z$
using *assms* **by** *typecheck-cfuncs*
show $((\textit{eval-func } Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle) \circ_c \textit{eval-func } Y\ X \circ_c \langle id_c\ X, g \circ_c \beta_X\rangle) \circ_c \textit{left-cart-proj } X\ \mathbf{1} : X \times_c \mathbf{1} \to Z$
using *assms* **by** *typecheck-cfuncs*
next
fix *x1*
assume *x1-type*[*type-rule*]: $x1 \in_c (X \times_c \mathbf{1})$
then obtain *x* **where** *x-type*[*type-rule*]: $x \in_c X$ **and** *x-def*: $x1 = \langle x, id_c\ \mathbf{1}\rangle$
by (*metis cart-prod-decomp id-type terminal-func-unique*)
then have $(f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj } X\ \mathbf{1}\rangle) \circ_c x1 = \textit{eval-func } Z\ Y \circ_c \langle \textit{eval-func } Y\ X \circ_c \langle x,g\rangle,f\rangle$
using *assms meta-comp2-elements-aux x-def* **by** *blast*
also have ... $= \textit{eval-func } Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle \circ_c \textit{eval-func } Y\ X \circ_c \langle id_c\ X, g$

$\circ_c$ β_X⟩ $\circ_c$ *x*
using *assms* **by** (*typecheck-cfuncs*, *metis cart-prod-extract-left*)
also have ... = (*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩ $\circ_c$ *x*
using *assms* **by** (*typecheck-cfuncs*, *meson comp-associative2*)
also have ... = ((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *x*
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj X* **1** $\circ_c$ *x1*
using *assms id-type left-cart-proj-cfunc-prod x-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj X* **1**) $\circ_c$ *x1*
using *assms* **by** (*typecheck-cfuncs*, *meson comp-associative2*)
finally show ($f^\flat$ $\circ_c$ ⟨$g^\flat$,*right-cart-proj X* **1**⟩) $\circ_c$ *x1* = (((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj X* **1**) $\circ_c$ *x1*.

qed
then show ($f^\flat$ $\circ_c$ ⟨$g^\flat$,*right-cart-proj X* **1**⟩)$^\sharp$ = (((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj* (*domain* ((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩)) **1**)$^\sharp$
using *assms cfunc-type-def cnufatem-def2 cnufatem-type domain-comp* **by** *force*
qed

lemma *meta-comp2-def4*:
assumes *f-type*[*type-rule*]: *f* $\in_c$ Z^Y **and** *g-type*[*type-rule*]: *g* $\in_c$ Y^X
shows *f* □ *g* = *meta-comp X Y Z* $\circ_c$ ⟨*f*,*g*⟩
using *assms*
proof(*unfold meta-comp2-def2 cnufatem-def2 metafunc-def meta-comp-def*)
have (((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj X* **1**) =
(*eval-func Z Y* $\circ_c$ *swap* (Z^Y) *Y* $\circ_c$ (*id*$_c$ (Z^Y) $\times_f$ (*eval-func Y X* $\circ_c$ *swap* (Y^X) *X*)) $\circ_c$ *associate-right* (Z^Y) (Y^X) *X* $\circ_c$ *swap X* (Z^Y $\times_c$ Y^X)) $\circ_c$ (*id* (*X*) $\times_f$ ⟨*f*,*g*⟩)
proof(*etcs-rule one-separator*)
fix *x1*
assume *x1-type*[*type-rule*]: *x1* $\in_c$ *X* $\times_c$ **1**
then obtain *x* **where** *x-type*[*type-rule*]: *x* $\in_c$ *X* **and** *x-def*: *x1* = ⟨*x*, *id*$_c$ **1**⟩
by (*metis cart-prod-decomp id-type terminal-func-unique*)
have (((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj X* **1**) $\circ_c$ *x1* =
((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *left-cart-proj X* **1** $\circ_c$ *x1*
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative*)
also have ... = ((*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩) $\circ_c$ *x*
using *id-type left-cart-proj-cfunc-prod x-def* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = (*eval-func Z Y* $\circ_c$ ⟨*id*$_c$ *Y*,*f* $\circ_c$ β_Y⟩) $\circ_c$ *eval-func Y X* $\circ_c$ ⟨*id*$_c$ *X*,*g* $\circ_c$ β_X⟩ $\circ_c$ *x*

by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative*)
also have ... = $\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle \circ_c \mathit{eval\text{-}func}\ Y\ X \circ_c \langle id_c\ X, g \circ_c \beta_X\rangle \circ_c x$
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative*)
also have ... = $\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle \circ_c \mathit{eval\text{-}func}\ Y\ X \circ_c \langle x, g\rangle$
by (*typecheck-cfuncs*, *metis cart-prod-extract-left*)
also have ... = $\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c \langle x, g\rangle, f\rangle$
by (*typecheck-cfuncs*, *metis cart-prod-extract-left*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y) \circ_c \langle f, \mathit{eval\text{-}func}\ Y\ X \circ_c \langle x, g\rangle\rangle$
by (*typecheck-cfuncs*, *metis comp-associative2 swap-ap*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y) \circ_c \langle id_c\ (Z^Y) \circ_c f, (\mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \langle g, x\rangle\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 id-left-unit2 swap-ap*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y) \circ_c (id_c\ (Z^Y) \times_f (\mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X)) \circ_c \langle f, \langle g, x\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X)) \circ_c \langle f, \langle g, x\rangle\rangle$
using *assms comp-associative2* **by** (*typecheck-cfuncs*, *force*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X)) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \langle\langle f, g\rangle, x\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *associate-right-ap*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X) \circ_c \langle\langle f, g\rangle, x\rangle$
using *assms comp-associative2* **by** (*typecheck-cfuncs*, *force*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X) \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X) \circ_c \langle x, \langle f, g\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *swap-ap*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X)) \circ_c \langle x, \langle f, g\rangle\rangle$
using *assms comp-associative2* **by** (*typecheck-cfuncs*, *force*)
also have ... = $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X)) \circ_c ((id_c\ X \times_f \langle f, g\rangle) \circ_c x1)$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2 id-type x-def*)
also have ... = $((\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X)) \circ_c id_c\ X \times_f \langle f, g\rangle) \circ_c x1$
by (*typecheck-cfuncs*, *meson comp-associative2*)
finally show $(((\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle) \circ_c \mathit{eval\text{-}func}\ Y\ X \circ_c \langle id_c\ X, g \circ_c \beta_X\rangle) \circ_c \mathit{left\text{-}cart\text{-}proj}\ X\ \mathbf{1}) \circ_c x1 =$
$((\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X)) \circ_c id_c\ X \times_f$

$\langle f,g\rangle) \circ_c$ *x1*.
qed
then have $(((\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle id_c\ Y, f \circ_c \beta_Y\rangle) \circ_c \mathit{eval\text{-}func}\ Y\ X \circ_c \langle id_c\ X, g \circ_c \beta_X\rangle) \circ_c$
$\mathit{left\text{-}cart\text{-}proj}\ X\ \mathbf{1})^\sharp = (\mathit{eval\text{-}func}\ Z\ Y \circ_c\ \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f (\mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X))$
$\circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X))^\sharp \circ_c \langle f,g\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *sharp-comp*)
then show $(f^\flat \circ_c \langle g^\flat, \mathit{right\text{-}cart\text{-}proj}\ X\ \mathbf{1}\rangle)^\sharp =$
$(\mathit{eval\text{-}func}\ Z\ Y \circ_c \mathit{swap}\ (Z^Y)\ Y \circ_c (id_c\ (Z^Y) \times_f \mathit{eval\text{-}func}\ Y\ X \circ_c \mathit{swap}\ (Y^X)\ X) \circ_c \mathit{associate\text{-}right}\ (Z^Y)\ (Y^X)\ X \circ_c \mathit{swap}\ X\ (Z^Y \times_c Y^X))^\sharp \circ_c \langle f,g\rangle$
using *assms cfunc-type-def cnufatem-def2 cnufatem-type domain-comp meta-comp2-def2 meta-comp2-def3 metafunc-def* **by** *force*
qed

lemma *meta-comp-on-els*:
assumes $f : W \to Z^Y$
assumes $g : W \to Y^X$
assumes $w \in_c W$
shows $(f \Box g) \circ_c w = (f \circ_c w) \Box (g \circ_c w)$
proof −
have $(f \Box g) \circ_c w = (f^\flat \circ_c \langle g^\flat, \mathit{right\text{-}cart\text{-}proj}\ X\ W\rangle)^\sharp \circ_c w$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp2-def2*)
also have ... $= (\mathit{eval\text{-}func}\ Z\ Y \circ_c (id\ Y \times_f f) \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f g), \mathit{right\text{-}cart\text{-}proj}\ X\ W\rangle)^\sharp \circ_c w$
using *assms comp-associative2 inv-transpose-func-def3* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f g), f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W\rangle)^\sharp \circ_c w$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... $= (\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f (g \circ_c w)), (f \circ_c w) \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ \mathbf{1}\rangle)^\sharp$
proof −
have $(\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f g), f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W\rangle)^{\sharp\flat} \circ_c (id\ X \times_f w) =$
$\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f (g \circ_c w)), f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W \circ_c (id\ X \times_f w)\rangle$
proof −
have $\mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f g), f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W\rangle \circ_c (id\ X \times_f w)$
$= \mathit{eval\text{-}func}\ Z\ Y \circ_c \langle(\mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f g)) \circ_c (id\ X \times_f w), (f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W) \circ_c (id\ X \times_f w)\rangle$
using *assms cfunc-prod-comp* **by** (*typecheck-cfuncs*, *force*)
also have ... $= \mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f g) \circ_c (id\ X \times_f w), f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W \circ_c (id\ X \times_f w)\rangle$
using *assms comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= \mathit{eval\text{-}func}\ Z\ Y \circ_c \langle \mathit{eval\text{-}func}\ Y\ X \circ_c (id\ X \times_f (g \circ_c w)), f \circ_c \mathit{right\text{-}cart\text{-}proj}\ X\ W \circ_c (id\ X \times_f w)\rangle$

using *assms* **by** (*typecheck-cfuncs*, *metis identity-distributes-across-composition*)
ultimately show *?thesis*
using *assms comp-associative2 flat-cancels-sharp* **by** (*typecheck-cfuncs*, *auto*)
qed
then show *?thesis*
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 inv-transpose-func-def3*

inv-transpose-of-composition right-cart-proj-cfunc-cross-prod transpose-func-unique)
qed
also have ... = $(eval\text{-}func\ Z\ Y \circ_c (id_c\ Y \times_f ((f \circ_c w) \circ_c right\text{-}cart\text{-}proj\ X\ \mathbf{1})) \circ_c \langle eval\text{-}func\ Y\ X \circ_c (id\ X \times_f (g \circ_c w)), id\ (X \times_c \mathbf{1})\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
also have ... = $(eval\text{-}func\ Z\ Y \circ_c (id_c\ Y \times_f (f \circ_c w)) \circ_c (id\ (Y) \times_f right\text{-}cart\text{-}proj\ X\ \mathbf{1}) \circ_c \langle eval\text{-}func\ Y\ X \circ_c (id\ X \times_f (g \circ_c w)), id\ (X \times_c \mathbf{1})\rangle)^\sharp$
using *assms comp-associative2 identity-distributes-across-composition* **by** (*typecheck-cfuncs*, *force*)
also have ... = $((f \circ_c w)^\flat \circ_c (id\ (Y) \times_f right\text{-}cart\text{-}proj\ X\ \mathbf{1}) \circ_c \langle eval\text{-}func\ Y\ X \circ_c (id\ X \times_f (g \circ_c w)), id\ (X \times_c \mathbf{1})\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 inv-transpose-func-def3*)
also have ... = $((f \circ_c w)^\flat \circ_c (id\ (Y) \times_f right\text{-}cart\text{-}proj\ X\ \mathbf{1}) \circ_c \langle (g \circ_c w)^\flat, id\ (X \times_c \mathbf{1})\rangle)^\sharp$
using *assms inv-transpose-func-def3* **by** (*typecheck-cfuncs*, *force*)
also have ... = $((f \circ_c w)^\flat \circ_c \langle (g \circ_c w)^\flat, right\text{-}cart\text{-}proj\ X\ \mathbf{1}\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
also have ... = $(f \circ_c w) \Box (g \circ_c w)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp2-def2*)
finally show *?thesis*.
qed

lemma *meta-comp2-def5*:
assumes $f : W \to Z^Y$
assumes $g : W \to Y^X$
shows $f \Box g = meta\text{-}comp\ X\ Y\ Z \circ_c \langle f,g\rangle$
proof(*rule one-separator*[**where** $X = W$, **where** $Y = Z^X$])
show $f \Box g : W \to Z^X$
using *assms* **by** *typecheck-cfuncs*
show $meta\text{-}comp\ X\ Y\ Z \circ_c \langle f,g\rangle : W \to Z^X$
using *assms* **by** *typecheck-cfuncs*
next
fix *w*
assume *w-type*[*type-rule*]: $w \in_c W$
have $(meta\text{-}comp\ X\ Y\ Z \circ_c \langle f,g\rangle) \circ_c w = meta\text{-}comp\ X\ Y\ Z \circ_c \langle f,g\rangle \circ_c w$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = $meta\text{-}comp\ X\ Y\ Z \circ_c \langle f \circ_c w, g \circ_c w\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp*)
also have ... = $(f \circ_c w) \Box (g \circ_c w)$

using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp2-def4*)
also have ... = $(f \square g) \circ_c w$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp-on-els*)
ultimately show $(f \square g) \circ_c w = (\textit{meta-comp}\ X\ Y\ Z \circ_c \langle f,g\rangle) \circ_c w$
by *simp*
qed

lemma *meta-left-identity*:
assumes $g \in_c X^X$
shows $g \square \textit{metafunc}\ (\textit{id}\ X) = g$
using *assms* **by** (*typecheck-cfuncs*, *metis cfunc-type-def cnufatem-metafunc cnufatem-type id-right-unit meta-comp2-def3 metafunc-cnufatem*)

lemma *meta-right-identity*:
assumes $g \in_c X^X$
shows $\textit{metafunc}(\textit{id}\ X) \square g = g$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cnufatem-metafunc cnufatem-type id-left-unit2 meta-comp2-def3 metafunc-cnufatem*)

lemma *comp-as-metacomp*:
assumes $g : X \to Y$
assumes $f : Y \to Z$
shows $f \circ_c g = \textit{cnufatem}(\textit{metafunc}\ f \square \textit{metafunc}\ g)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cnufatem-metafunc meta-comp2-def3*)

lemma *metacomp-as-comp*:
assumes $g \in_c Y^X$
assumes $f \in_c Z^Y$
shows $\textit{cnufatem}\ f \circ_c \textit{cnufatem}\ g = \textit{cnufatem}(f \square g)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-as-metacomp metafunc-cnufatem*)

lemma *meta-comp-assoc*:
assumes $e : W \to A^Z$
assumes $f : W \to Z^Y$
assumes $g : W \to Y^X$
shows $(e \square f) \square g = e \square (f \square g)$
proof −
have $(e \square f) \square g = (e^\flat \circ_c \langle f^\flat, \textit{right-cart-proj}\ Y\ W\rangle)^\sharp \square g$
using *assms* **by** (*simp add*: *meta-comp2-def2*)
also have ... = $((e^\flat \circ_c \langle f^\flat, \textit{right-cart-proj}\ Y\ W\rangle)^{\sharp\flat} \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp2-def2*)
also have ... = $((e^\flat \circ_c \langle f^\flat, \textit{right-cart-proj}\ Y\ W\rangle) \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *flat-cancels-sharp*)
also have ... = $(e^\flat \circ_c \langle f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle, \textit{right-cart-proj}\ X\ W\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2 right-cart-proj-cfunc-prod*)
also have ... = $(e^\flat \circ_c \langle (f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle)^{\sharp\flat}, \textit{right-cart-proj}\ X\ W\rangle)^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *flat-cancels-sharp*)
also have ... = $e \square (f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle)^\sharp$

using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp2-def2*)
also have ... $= e \square (f \square g)$
using *assms* **by** (*simp add*: *meta-comp2-def2*)
finally show *?thesis*.
qed

12.4 Partially Parameterized Functions on Pairs

definition *left-param* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* ($-_{[-,-]}$ [*100*,*0*]*100*) **where**
left-param k p ≡ (*THE f*. $\exists$ P Q R. $k : P \times_c Q \to R \wedge f = k \circ_c \langle p \circ_c \beta_Q, id\ Q\rangle$)

lemma *left-param-def2*:
assumes $k : P \times_c Q \to R$
shows $k_{[p,-]} \equiv k \circ_c \langle p \circ_c \beta_Q, id\ Q\rangle$
proof −
have $\exists$ P Q R. $k : P \times_c Q \to R \wedge$ *left-param k p* $= k \circ_c \langle p \circ_c \beta_Q, id\ Q\rangle$
unfolding *left-param-def* **by** (*smt* (*z3*) *cfunc-type-def the1I2 transpose-func-type assms*)
then show $k_{[p,-]} \equiv k \circ_c \langle p \circ_c \beta_Q, id\ Q\rangle$
by (*smt* (*z3*) *assms cfunc-type-def transpose-func-type*)
qed

lemma *left-param-type*[*type-rule*]:
assumes $k : P \times_c Q \to R$
assumes $p \in_c P$
shows $k_{[p,-]} : Q \to R$
using *assms* **by** (*unfold left-param-def2*, *typecheck-cfuncs*)

lemma *left-param-on-el*:
assumes $k : P \times_c Q \to R$
assumes $p \in_c P$
assumes $q \in_c Q$
shows $k_{[p,-]} \circ_c q = k \circ_c \langle p, q\rangle$
proof −
have $k_{[p,-]} \circ_c q = k \circ_c \langle p \circ_c \beta_Q, id\ Q\rangle \circ_c q$
using *assms cfunc-type-def comp-associative left-param-def2* **by** (*typecheck-cfuncs*, *force*)
also have ... $= k \circ_c \langle p, q\rangle$
using *assms*(*2*,*3*) *cart-prod-extract-right* **by** *force*
finally show *?thesis*.
qed

definition *right-param* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* ($-_{[-,-]}$ [*100*,*0*]*100*) **where**
right-param k q ≡ (*THE f*. $\exists$ P Q R. $k : P \times_c Q \to R \wedge f = k \circ_c \langle id\ P, q \circ_c \beta_P\rangle$)

lemma *right-param-def2*:

assumes $k : P \times_c Q \to R$
shows $k_{[-,q]} \equiv k \circ_c \langle id\ P,\ q \circ_c \beta_P\rangle$
proof –
have $\exists\ P\ Q\ R.\ k : P \times_c Q \to R \wedge right\text{-}param\ k\ q = k \circ_c \langle id\ P,\ q \circ_c \beta_P\rangle$
unfolding *right-param-def* **by** (*rule theI'*, *insert assms*, *auto*, *metis cfunc-type-def exp-set-inj transpose-func-type*)
then show $k_{[-,q]} \equiv k \circ_c \langle id_c\ P, q \circ_c \beta_P\rangle$
by (*smt* (*z3*) *assms cfunc-type-def exp-set-inj transpose-func-type*)
qed

lemma *right-param-type*[*type-rule*]:
assumes $k : P \times_c Q \to R$
assumes $q \in_c Q$
shows $k_{[-,q]} : P \to R$
using *assms* **by** (*unfold right-param-def2*, *typecheck-cfuncs*)

lemma *right-param-on-el*:
assumes $k : P \times_c Q \to R$
assumes $p \in_c P$
assumes $q \in_c Q$
shows $k_{[-,q]} \circ_c p = k \circ_c \langle p,\ q\rangle$
proof –
have $k_{[-,q]} \circ_c p = k \circ_c \langle id\ P,\ q \circ_c \beta_P\rangle \circ_c p$
using *assms cfunc-type-def comp-associative right-param-def2* **by** (*typecheck-cfuncs*, *force*)
also have ... $= k \circ_c \langle p,\ q\rangle$
using *assms*(*2*,*3*) *cart-prod-extract-left* **by** *force*
finally show *?thesis***.**
qed

12.5 Exponential Set Facts

The lemma below corresponds to Proposition 2.5.7 in Halvorson.

lemma *exp-one*:
$X^{\mathbf{1}} \cong X$
proof –
obtain e **where** *e-defn*: $e = eval\text{-}func\ X\ \mathbf{1}$ **and** *e-type*: $e : \mathbf{1} \times_c X^{\mathbf{1}} \to X$
using *eval-func-type* **by** *auto*
obtain i **where** *i-type*: i: $\mathbf{1} \times_c \mathbf{1} \to \mathbf{1}$
using *terminal-func-type* **by** *blast*
obtain *i-inv* **where** *i-iso*: *i-inv*: $\mathbf{1} \to\ \mathbf{1} \times_c \mathbf{1}\ \wedge$
$i \circ_c i\text{-}inv = id(\mathbf{1})\ \wedge$
$i\text{-}inv \circ_c i = id(\mathbf{1} \times_c \mathbf{1})$
by (*smt cfunc-cross-prod-comp-cfunc-prod cfunc-cross-prod-comp-diagonal cfunc-cross-prod-def cfunc-prod-type cfunc-type-def diagonal-def i-type id-cross-prod id-left-unit id-type left-cart-proj-type right-cart-proj-cfunc-prod right-cart-proj-type terminal-func-unique*)
then have *i-inv-type*: *i-inv*: $\mathbf{1} \to\ \mathbf{1} \times_c \mathbf{1}$
by *auto*

have *inj*: *injective*(*e*)
by (*simp add*: *e-defn eval-func-X-one-injective*)

have *surj*: *surjective*(*e*)
unfolding *surjective-def*
proof *clarify*
fix *y*
assume $y \in_c$ *codomain e*
then have *y-type*: $y \in_c X$
using *cfunc-type-def e-type* **by** *auto*

have *witness-type*: $(id_c\ \mathbf{1} \times_f (y \circ_c i)^\sharp) \circ_c$ *i-inv* $\in_c \mathbf{1} \times_c X^{\mathbf{1}}$
using *y-type i-type i-inv-type* **by** *typecheck-cfuncs*

have *square*: $e \circ_c (id(\mathbf{1}) \times_f (y \circ_c i)^\sharp) = y \circ_c i$
using *comp-type e-defn i-type transpose-func-def y-type* **by** *blast*
then show $\exists\, x.\ x \in_c$ *domain e* $\wedge\ e \circ_c x = y$
unfolding *cfunc-type-def* **using** *y-type i-type i-inv-type e-type*
by (*intro exI*[**where** x=$(id(\mathbf{1}) \times_f (y \circ_c i)^\sharp) \circ_c$ *i-inv*], *typecheck-cfuncs*, *metis cfunc-type-def comp-associative i-iso id-right-unit2*)
qed

have *isomorphism e*
using *epi-mon-is-iso inj injective-imp-monomorphism surj surjective-is-epimorphism* **by** *fastforce*
then show $X^{\mathbf{1}} \cong X$
using *e-type is-isomorphic-def isomorphic-is-symmetric isomorphic-is-transitive one-x-A-iso-A* **by** *blast*
qed

The lemma below corresponds to Proposition 2.5.8 in Halvorson.

lemma *exp-empty*:
$X^{\emptyset} \cong \mathbf{1}$
proof –
obtain *f* **where** *f-type*: $f = \alpha_X \circ_c$ (*left-cart-proj* $\emptyset$ $\mathbf{1}$) **and** *fsharp-type*[*type-rule*]: $f^\sharp \in_c X^{\emptyset}$
using *transpose-func-type* **by** (*typecheck-cfuncs*, *force*)
have *uniqueness*: $\forall\, z.\ z \in_c X^{\emptyset} \longrightarrow z = f^\sharp$
proof *clarify*
fix *z*
assume *z-type*[*type-rule*]: $z \in_c X^{\emptyset}$
obtain *j* **where** *j-iso*:j:$\emptyset \to \emptyset \times_c \mathbf{1} \wedge$ *isomorphism*(*j*)
using *is-isomorphic-def isomorphic-is-symmetric empty-prod-X* **by** *presburger*
obtain ψ **where** *psi-type*: $\psi : \emptyset \times_c \mathbf{1} \to \emptyset\ \wedge$
$j \circ_c \psi = id(\emptyset \times_c \mathbf{1}) \wedge \psi \circ_c j = id(\emptyset)$
using *cfunc-type-def isomorphism-def j-iso* **by** *fastforce*
then have *f-sharp* : $id(\emptyset) \times_f z = id(\emptyset) \times_f f^\sharp$
by (*typecheck-cfuncs*, *meson comp-type emptyset-is-empty one-separator*)

then show $z = f^\sharp$
using *fsharp-type same-evals-equal z-type* **by** *force*
qed
then have $\exists!\ x.\ x \in_c X^\emptyset$
by (*intro ex1I*[**where** $a = f^\sharp$], *simp-all add*: *fsharp-type*)
then show $X^\emptyset \cong \mathbf{1}$
using *single-elem-iso-one* **by** *auto*
qed

lemma *one-exp*:
$\mathbf{1}^X \cong \mathbf{1}$
proof –
have *nonempty*: *nonempty*$(\mathbf{1}^X)$
using *nonempty-def right-cart-proj-type transpose-func-type* **by** *blast*
obtain e **where** *e-defn*: e = *eval-func* $\mathbf{1}$ X **and** *e-type*: $e : X \times_c \mathbf{1}^X \to \mathbf{1}$
by (*simp add*: *eval-func-type*)
have *uniqueness*: $\forall\, y.\ (y \in_c \mathbf{1}^X \longrightarrow e \circ_c (id(X) \times_f y) : X \times_c \mathbf{1} \to \mathbf{1})$
by (*meson cfunc-cross-prod-type comp-type e-type id-type*)
have *uniquess-form*: $\forall\, y.\ (y \in_c \mathbf{1}^X \longrightarrow e \circ_c (id(X) \times_f y) = \beta_{X \times_c \mathbf{1}})$
using *terminal-func-unique uniqueness* **by** *blast*
then have *ex1*: $(\exists!\ x.\ x \in_c \mathbf{1}^X)$
by (*metis e-defn nonempty nonempty-def transpose-func-unique uniqueness*)
show $\mathbf{1}^X \cong \mathbf{1}$
using *ex1 single-elem-iso-one* **by** *auto*
qed

The lemma below corresponds to Proposition 2.5.9 in Halvorson.

lemma *power-rule*:
$(X \times_c Y)^A \cong X^A \times_c Y^A$
proof –
have *is-cart-prod* $((X \times_c Y)^A)$ ((*left-cart-proj* X $Y)^A{}_f$) (*right-cart-proj* X $Y^A{}_f$) (X^A) (Y^A)
proof (*etcs-subst is-cart-prod-def2*, *clarify*)
fix f g Z
assume *f-type*[*type-rule*]: $f : Z \to X^A$
assume *g-type*[*type-rule*]: $g : Z \to Y^A$

show $\exists\, h.\ h : Z \to (X \times_c Y)^A \wedge$
left-cart-proj X $Y^A{}_f \circ_c h = f \wedge$
right-cart-proj X $Y^A{}_f \circ_c h = g \wedge$
$(\forall\, h2.\ h2 : Z \to (X \times_c Y)^A \wedge$ *left-cart-proj* X $Y^A{}_f \circ_c h2 = f \wedge$ *right-cart-proj* X $Y^A{}_f \circ_c h2 = g \longrightarrow$
$h2 = h)$
proof (*intro exI*[**where** $x = \langle f^\flat, g^\flat \rangle^\sharp$], *safe*, *typecheck-cfuncs*)
have ((*left-cart-proj* X $Y)^A{}_f) \circ_c \langle f^\flat, g^\flat \rangle^\sharp$ = ((*left-cart-proj* X Y) $\circ_c \langle f^\flat, g^\flat \rangle)^\sharp$
by (*typecheck-cfuncs*, *metis transpose-of-comp*)
also have ... $= f^{\flat\sharp}$
by (*typecheck-cfuncs*, *simp add*: *left-cart-proj-cfunc-prod*)

also have ... = f
by (*typecheck-cfuncs*, *simp add*: *sharp-cancels-flat*)
finally show *projection-property1*: $((\mathit{left\text{-}cart\text{-}proj}\ X\ Y)^{A}{}_{f}) \circ_c \langle f^\flat\ , g^\flat\rangle^\sharp = f$.
show *projection-property2*: $((\mathit{right\text{-}cart\text{-}proj}\ X\ Y)^{A}{}_{f}) \circ_c \langle f^\flat\ , g^\flat\rangle^\sharp = g$
by (*typecheck-cfuncs*, *metis right-cart-proj-cfunc-prod sharp-cancels-flat transpose-of-comp*)
show $\bigwedge h2.\ h2 : Z \rightarrow (X \times_c\ Y)^{A} \Longrightarrow$
$f = \mathit{left\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h2 \Longrightarrow$
$g = \mathit{right\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h2 \Longrightarrow$
$h2 = \langle(\mathit{left\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h2)^\flat, (\mathit{right\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h2)^\flat\rangle^\sharp$
proof −
fix h
assume *h-type*[*type-rule*]: $h : Z \rightarrow (X \times_c\ Y)^{A}$
assume *h-property1*: $f = ((\mathit{left\text{-}cart\text{-}proj}\ X\ Y)^{A}{}_{f}) \circ_c h$
assume *h-property2*: $g = ((\mathit{right\text{-}cart\text{-}proj}\ X\ Y)^{A}{}_{f}) \circ_c h$

have $f = (\mathit{left\text{-}cart\text{-}proj}\ X\ Y)^{A}{}_{f} \circ_c h^{\flat\sharp}$
by (*metis h-property1 h-type sharp-cancels-flat*)
also have ... = $((\mathit{left\text{-}cart\text{-}proj}\ X\ Y) \circ_c h^\flat)^\sharp$
by (*typecheck-cfuncs*, *simp add*: *transpose-of-comp*)
ultimately have *computation1*: $f = ((\mathit{left\text{-}cart\text{-}proj}\ X\ Y) \circ_c h^\flat)^\sharp$
by *simp*
then have *unqiueness1*: $(\mathit{left\text{-}cart\text{-}proj}\ X\ Y) \circ_c\ h^\flat = f^\flat$
by (*typecheck-cfuncs*, *simp add*: *flat-cancels-sharp*)
have $g = ((\mathit{right\text{-}cart\text{-}proj}\ X\ Y)^{A}{}_{f}) \circ_c (h^\flat)^\sharp$
by (*metis h-property2 h-type sharp-cancels-flat*)
have ... = $((\mathit{right\text{-}cart\text{-}proj}\ X\ Y) \circ_c h^\flat)^\sharp$
by (*typecheck-cfuncs*, *metis transpose-of-comp*)
have *computation2*: $g = ((\mathit{right\text{-}cart\text{-}proj}\ X\ Y) \circ_c h^\flat)^\sharp$
by (*simp add*: ‹$g = \mathit{right\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h^{\flat\sharp}$› ‹$\mathit{right\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h^{\flat\sharp} = (\mathit{right\text{-}cart\text{-}proj}\ X\ Y \circ_c h^\flat)^\sharp$›)
then have *unqiueness2*: $(\mathit{right\text{-}cart\text{-}proj}\ X\ Y) \circ_c\ h^\flat = g^\flat$
using *h-type g-type* **by** (*typecheck-cfuncs*, *simp add*: *computation2 flat-cancels-sharp*)
then have *h-flat*: $h^\flat = \langle f^\flat,\ g^\flat\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-unique unqiueness1 unqiueness2*)
then have *h-is-sharp-prod-fflat-gflat*: $h = \langle f^\flat,\ g^\flat\rangle^\sharp$
by (*metis h-type sharp-cancels-flat*)
then show $h = \langle(\mathit{left\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h)^\flat, (\mathit{right\text{-}cart\text{-}proj}\ X\ Y^{A}{}_{f} \circ_c h)^\flat\rangle^\sharp$
using *h-property1 h-property2* **by** *force*
qed
qed
qed
then show $(X \times_c\ Y)^{A} \cong X^{A} \times_c Y^{A}$
using *canonical-cart-prod-is-cart-prod cart-prods-isomorphic fst-conv is-isomorphic-def* **by** *fastforce*
qed

lemma *exponential-coprod-distribution*:
$Z^{(X \coprod Y)} \cong (Z^X) \times_c (Z^Y)$
proof –
have *is-cart-prod* $(Z^{(X \coprod Y)})$ $((\textit{eval-func}\ Z\ (X \coprod Y) \circ_c (\textit{left-coproj}\ X\ Y) \times_f (\textit{id}(Z^{(X \coprod Y)})))^\sharp)$ $((\textit{eval-func}\ Z\ (X \coprod Y) \circ_c (\textit{right-coproj}\ X\ Y) \times_f (\textit{id}(Z^{(X \coprod Y)})))^\sharp)$ (Z^X) (Z^Y)
proof (*etcs-subst is-cart-prod-def2*, *clarify*)
fix $f\ g\ H$
assume *f-type*[*type-rule*]: $f : H \rightarrow Z^X$
assume *g-type*[*type-rule*]: $g : H \rightarrow Z^Y$
show $\exists h.\ h : H \rightarrow Z^{(X \coprod Y)} \wedge$
$(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{left-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h = f \wedge$
$(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{right-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h = g \wedge$
$(\forall h2.\ h2 : H \rightarrow Z^{(X \coprod Y)} \wedge$
$(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{left-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h2 = f \wedge$
$(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{right-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h2 = g \longrightarrow$
$h2 = h)$
proof (*intro exI*[**where** $x=(f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp$], *safe*, *typecheck-cfuncs*)
have $(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{left-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp =$
$((\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{left-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)})) \circ_c (\textit{id}\ X \times_f (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp))^\sharp$
using *sharp-comp* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= (\textit{eval-func}\ Z\ (X \coprod Y) \circ_c\ (\textit{left-coproj}\ X\ Y \times_f (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp))^\sharp$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-cross-prod comp-associative2 id-left-unit2 id-right-unit2*)
also have ... $= (\textit{eval-func}\ Z\ (X \coprod Y) \circ_c\ (\textit{id}\ (X \coprod Y) \times_f (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp) \circ_c (\textit{left-coproj}\ X\ Y \times_f \textit{id}\ H))^\sharp$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-cross-prod id-left-unit2 id-right-unit2*)
also have ... $= (f^\flat \amalg g^\flat \circ_c (\textit{dist-prod-coprod-right}\ X\ Y\ H \circ_c \textit{left-coproj}\ X\ Y \times_f \textit{id}\ H))^\sharp$
using *comp-associative2 transpose-func-def* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (f^\flat \amalg g^\flat \circ_c \textit{left-coproj}\ (X \times_c H)\ (Y \times_c H))^\sharp$
by (*simp add*: *dist-prod-coprod-right-left-coproj*)
also have ... $= f$
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod sharp-cancels-flat*)
finally show $(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{left-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp = f$.
next
have $(\textit{eval-func}\ Z\ (X \coprod Y) \circ_c \textit{right-coproj}\ X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right}\ X\ Y\ H)^\sharp =$

$((\textit{eval-func } Z\ (X \coprod Y) \circ_c \textit{right-coproj } X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)})) \circ_c (\textit{id } Y \times_f (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H)^\sharp))^\sharp$
using *sharp-comp* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= (\textit{eval-func } Z\ (X \coprod Y) \circ_c (\textit{right-coproj } X\ Y \times_f (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H)^\sharp))^\sharp$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-cross-prod comp-associative2 id-left-unit2 id-right-unit2*)
also have ... $= (\textit{eval-func } Z\ (X \coprod Y) \circ_c (\textit{id } (X \coprod Y) \times_f (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H)^\sharp) \circ_c (\textit{right-coproj } X\ Y \times_f \textit{id } H))^\sharp$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-cross-prod id-left-unit2 id-right-unit2*)
also have ... $= (f^\flat \amalg g^\flat \circ_c (\textit{dist-prod-coprod-right } X\ Y\ H \circ_c \textit{right-coproj } X\ Y \times_f \textit{id } H))^\sharp$
using *comp-associative2 transpose-func-def* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (f^\flat \amalg g^\flat \circ_c \textit{right-coproj } (X \times_c H)\ (Y \times_c H))^\sharp$
by (*simp add*: *dist-prod-coprod-right-right-coproj*)
also have ... $= g$
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod sharp-cancels-flat*)
finally show $(\textit{eval-func } Z\ (X \coprod Y) \circ_c \textit{right-coproj } X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H)^\sharp = g$**.**
next
fix h
assume *h-type*[*type-rule*]: $h : H \to Z^{(X \coprod Y)}$
assume *f-eqs*: $f = (\textit{eval-func } Z\ (X \coprod Y) \circ_c \textit{left-coproj } X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h$
assume *g-eqs*: $g = (\textit{eval-func } Z\ (X \coprod Y) \circ_c \textit{right-coproj } X\ Y \times_f \textit{id}_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h$
have $(f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H) = h^\flat$
proof(*etcs-rule one-separator*[**where** $X = (X \coprod Y) \times_c H$, **where** $Y = Z$])
show $\bigwedge xyh.\ xyh \in_c (X \coprod Y) \times_c H \Longrightarrow (f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H) \circ_c xyh = h^\flat \circ_c xyh$
proof−
fix *xyh*
assume *l-type*[*type-rule*]: $xyh \in_c (X \coprod Y) \times_c H$
then obtain *xy* **and** *z* **where** *xy-type*[*type-rule*]: $xy \in_c X \coprod Y$ **and** *z-type*[*type-rule*]: $z \in_c H$
and *xyh-def*: $xyh = \langle xy,z\rangle$
using *cart-prod-decomp* **by** *blast*
show $(f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H) \circ_c xyh = h^\flat \circ_c xyh$
proof(*cases* $\exists x.\ x \in_c X \wedge xy = \textit{left-coproj } X\ Y \circ_c x$)
assume $\exists x.\ x \in_c X \wedge xy = \textit{left-coproj } X\ Y \circ_c x$
then obtain x **where** *x-type*[*type-rule*]: $x \in_c X$ **and** *xy-def*: $xy = \textit{left-coproj } X\ Y \circ_c x$
by *blast*
have $(f^\flat \amalg g^\flat \circ_c \textit{dist-prod-coprod-right } X\ Y\ H) \circ_c xyh = (f^\flat \amalg g^\flat) \circ_c (\textit{dist-prod-coprod-right } X\ Y\ H \circ_c \langle \textit{left-coproj } X\ Y \circ_c x,z\rangle)$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 xy-def xyh-def*)
also have ... $= (f^\flat \amalg g^\flat) \circ_c ((\textit{dist-prod-coprod-right } X\ Y\ H \circ_c (\textit{left-coproj } X\ Y \times_f \textit{id } H)) \circ_c \langle x,z\rangle)$

using *dist-prod-coprod-right-ap-left dist-prod-coprod-right-left-coproj* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = $(f^\flat \amalg g^\flat) \circ_c$ (*left-coproj* $(X \times_c H)$ $(Y \times_c H)$ $\circ_c \langle x,z\rangle)$
using *dist-prod-coprod-right-left-coproj* **by** *presburger*
also have ... = $f^\flat \circ_c \langle x,z\rangle$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod*)
also have ... = ((*eval-func* Z $(X \coprod Y)$ $\circ_c$ *left-coproj* X Y $\times_f$ id_c $(Z^{(X \coprod Y)}))^\sharp \circ_c h)^\flat \circ_c \langle x,z\rangle$
using *f-eqs* **by** *fastforce*
also have ... = (((*eval-func* Z $(X \coprod Y)$ $\circ_c$ *left-coproj* X Y $\times_f$ id_c $(Z^{(X \coprod Y)}))^{\sharp\flat}) \circ_c$ (*id* $X \times_f h)) \circ_c \langle x,z\rangle$
using *inv-transpose-of-composition* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = ((*eval-func* Z $(X \coprod Y)$ $\circ_c$ *left-coproj* X Y $\times_f$ id_c $(Z^{(X \coprod Y)})) \circ_c$ (*id* $X \times_f h)) \circ_c \langle x,z\rangle$
by (*typecheck-cfuncs*, *simp add*: *flat-cancels-sharp*)
also have ... = (*eval-func* Z $(X \coprod Y)$ $\circ_c$ *left-coproj* X Y $\times_f h) \circ_c \langle x,z\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-cross-prod comp-associative2 id-left-unit2 id-right-unit2*)
also have ... = *eval-func* Z $(X \coprod Y)$ $\circ_c$ $\langle$*left-coproj* X Y $\circ_c$ x, $h \circ_c z\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod comp-associative2*)
also have ... = *eval-func* Z $(X \coprod Y)$ $\circ_c$ $((id(X \coprod Y) \times_f h) \circ_c \langle xy,z\rangle)$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 xy-def*)
also have ... = $h^\flat \circ_c xyh$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 inv-transpose-func-def3 xyh-def*)
finally show *?thesis*.
next
assume $\nexists x.\ x \in_c X \wedge xy =$ *left-coproj* X Y $\circ_c x$
then obtain y **where** *y-type*[*type-rule*]: $y \in_c Y$ **and** *xy-def*: $xy =$ *right-coproj* X Y $\circ_c y$
using *coprojs-jointly-surj* **by** (*typecheck-cfuncs*, *blast*)
have $(f^\flat \amalg g^\flat \circ_c$ *dist-prod-coprod-right* X Y $H) \circ_c xyh = (f^\flat \amalg g^\flat) \circ_c$ (*dist-prod-coprod-right* X Y H $\circ_c \langle$*right-coproj* X Y $\circ_c y,z\rangle)$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 xy-def xyh-def*)
also have ... = $(f^\flat \amalg g^\flat) \circ_c$ ((*dist-prod-coprod-right* X Y H $\circ_c$ (*right-coproj* X Y $\times_f$ *id* H)) $\circ_c \langle y,z\rangle)$
using *dist-prod-coprod-right-ap-right dist-prod-coprod-right-right-coproj* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = $(f^\flat \amalg g^\flat) \circ_c$ (*right-coproj* $(X \times_c H)$ $(Y \times_c H)$ $\circ_c \langle y,z\rangle)$
using *dist-prod-coprod-right-right-coproj* **by** *presburger*
also have ... = $g^\flat \circ_c \langle y,z\rangle$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 right-coproj-cfunc-coprod*)
also have ... = ((*eval-func* Z $(X \coprod Y)$ $\circ_c$ *right-coproj* X Y $\times_f$ id_c $(Z^{(X \coprod Y)}))^\sharp \circ_c h)^\flat \circ_c \langle y,z\rangle$
using *g-eqs* **by** *fastforce*
also have ... = (((*eval-func* Z $(X \coprod Y)$ $\circ_c$ *right-coproj* X Y $\times_f$ id_c

$(Z^{(X \coprod Y)}))^{\sharp\flat}) \circ_c (id\ Y \times_f h)) \circ_c \langle y,z \rangle$
using *inv-transpose-of-composition* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = $((eval\text{-}func\ Z\ (X \coprod Y) \circ_c right\text{-}coproj\ X\ Y \times_f id_c\ (Z^{(X \coprod Y)})) \circ_c (id\ Y \times_f h)) \circ_c \langle y,z \rangle$
by (*typecheck-cfuncs*, *simp add*: *flat-cancels-sharp*)
also have ... = $(eval\text{-}func\ Z\ (X \coprod Y) \circ_c right\text{-}coproj\ X\ Y \times_f h) \circ_c \langle y,z \rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-cross-prod comp-associative2 id-left-unit2 id-right-unit2*)
also have ... = $eval\text{-}func\ Z\ (X \coprod Y) \circ_c \langle right\text{-}coproj\ X\ Y \circ_c y,\ h \circ_c z \rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod comp-associative2*)
also have ... = $eval\text{-}func\ Z\ (X \coprod Y) \circ_c ((id(X \coprod Y) \times_f h) \circ_c \langle xy,z \rangle)$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 xy-def*)
also have ... = $h^\flat \circ_c xyh$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 inv-transpose-func-def3 xyh-def*)
finally show *?thesis*.
qed
qed
qed
then show $h = (((eval\text{-}func\ Z\ (X \coprod Y) \circ_c left\text{-}coproj\ X\ Y \times_f id_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h)^\flat \amalg$
$((eval\text{-}func\ Z\ (X \coprod Y) \circ_c right\text{-}coproj\ X\ Y \times_f id_c\ (Z^{(X \coprod Y)}))^\sharp \circ_c h)^\flat \circ_c$
$dist\text{-}prod\text{-}coprod\text{-}right\ X\ Y\ H)^\sharp$
using *f-eqs g-eqs h-type sharp-cancels-flat* **by** *force*
qed
qed
then show *?thesis*
by (*metis canonical-cart-prod-is-cart-prod cart-prods-isomorphic is-isomorphic-def prod.sel*(*1*,*2*))
qed

lemma *empty-exp-nonempty*:
assumes *nonempty X*
shows $\emptyset^X \cong \emptyset$
proof−
obtain *j* **where** *j-type*[*type-rule*]: j: $\emptyset^X \to \mathbf{1} \times_c \emptyset^X$ **and** *j-def*: *isomorphism*(*j*)
using *is-isomorphic-def isomorphic-is-symmetric one-x-A-iso-A* **by** *blast*
obtain *y* **where** *y-type*[*type-rule*]: $y \in_c X$
using *assms nonempty-def* **by** *blast*
obtain *e* **where** *e-type*[*type-rule*]: e: $X \times_c \emptyset^X \to \emptyset$
using *eval-func-type* **by** *blast*
have *iso-type*[*type-rule*]: $(e \circ_c y \times_f id(\emptyset^X)) \circ_c j : \emptyset^X \to \emptyset$
by *typecheck-cfuncs*
show $\emptyset^X \cong \emptyset$
using *function-to-empty-is-iso is-isomorphic-def iso-type* **by** *blast*

qed

lemma *exp-pres-iso-left*:
  assumes $A \cong X$
  shows $A^Y \cong X^Y$
proof −
  obtain φ **where** φ-*def*: φ: $X \to A \wedge$ *isomorphism*(φ)
    using *assms is-isomorphic-def isomorphic-is-symmetric* **by** *blast*
  obtain ψ **where** ψ-*def*: ψ: $A \to X \wedge$ *isomorphism*(ψ) $\wedge$ ($\psi \circ_c \varphi$ = *id*(X))
    using φ-*def cfunc-type-def isomorphism-def* **by** *fastforce*
  have *idA*: $\varphi \circ_c \psi$ = *id*(A)
    by (*metis* φ-*def* ψ-*def cfunc-type-def comp-associative id-left-unit2 isomorphism-def*)
  have *phi-eval-type*: ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$: $X^Y \to A^Y$
    using φ-*def* **by** (*typecheck-cfuncs*, *blast*)
  have *psi-eval-type*: ($\psi \circ_c$ *eval-func* A Y)$^\sharp$: $A^Y \to X^Y$
    using ψ-*def* **by** (*typecheck-cfuncs*, *blast*)

  have *idXY*: ($\psi \circ_c$ *eval-func* A Y)$^\sharp$ $\circ_c$ ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$ = *id*(X^Y)
  proof −
    have ($\psi \circ_c$ *eval-func* A Y)$^\sharp$ $\circ_c$ ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$ = ${\psi^Y}_f \circ_c {\varphi^Y}_f$
      using φ-*def* ψ-*def exp-func-def2* **by** *auto*
    also have ... = ${(\psi \circ_c \varphi)^Y}_f$
      by (*metis* φ-*def* ψ-*def transpose-factors*)
    also have ... = ${(id\ X)^Y}_f$
      by (*simp add*: ψ-*def*)
    also have ... = *id*(X^Y)
      by (*simp add*: *exponential-object-identity2*)
    finally show ($\psi \circ_c$ *eval-func* A Y)$^\sharp$ $\circ_c$ ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$ = *id*(X^Y).
  qed
  have *idAY*: ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$ $\circ_c$ ($\psi \circ_c$ *eval-func* A Y)$^\sharp$ = *id*(A^Y)
  proof −
    have ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$ $\circ_c$ ($\psi \circ_c$ *eval-func* A Y)$^\sharp$ = ${\varphi^Y}_f \circ_c {\psi^Y}_f$
      using φ-*def* ψ-*def exp-func-def2* **by** *auto*
    also have ... = ${(\varphi \circ_c \psi)^Y}_f$
      by (*metis* φ-*def* ψ-*def transpose-factors*)
    also have ... = ${(id\ A)^Y}_f$
      by (*simp add*: *idA*)
    also have ... = *id*(A^Y)
      by (*simp add*: *exponential-object-identity2*)
    finally show ($\varphi \circ_c$ *eval-func* X Y)$^\sharp$ $\circ_c$ ($\psi \circ_c$ *eval-func* A Y)$^\sharp$ = *id*(A^Y).
  qed
  show $A^Y \cong X^Y$
   by (*metis cfunc-type-def comp-epi-imp-epi comp-monic-imp-monic epi-mon-is-iso idAY idXY id-isomorphism is-isomorphic-def iso-imp-epi-and-monic phi-eval-type psi-eval-type*)
qed

lemma *expset-power-tower*:
$(A^B)^C \cong A^{(B \times_c C)}$
proof –
obtain φ **where** φ*-def*: $\varphi = ((\textit{eval-func}\ A\ (B \times_c\ C)) \circ_c (\textit{associate-left}\ B\ C\ (A^{(B \times_c\ C)})))$ **and**
φ*-type*[*type-rule*]: φ: $B \times_c (C \times_c (A^{(B \times_c\ C)})) \to A$ **and**
φ*dbsharp-type*[*type-rule*]: $(\varphi^\sharp)^\sharp : (A^{(B \times_c\ C)}) \to ((A^B)^C)$
using *transpose-func-type* **by** (*typecheck-cfuncs*, *fastforce*)

obtain ψ **where** ψ*-def*: $\psi = (\textit{eval-func}\ A\ B) \circ_c (id(B) \times_f\ \textit{eval-func}\ (A^B)\ C) \circ_c (\textit{associate-right}\ B\ C\ ((A^B)^C))$ **and**
ψ*-type*[*type-rule*]: $\psi :\ (B \times_c\ C) \times_c ((A^B)^C) \to A$ **and**
ψ*sharp-type*[*type-rule*]: $\psi^\sharp$: $(A^B)^C \to (A^{(B \times_c\ C)})$
using *transpose-func-type* **by** (*typecheck-cfuncs*, *blast*)

have $\varphi^{\sharp\sharp} \circ_c \psi^\sharp = id((A^B)^C)$
proof(*etcs-rule same-evals-equal*[**where** $X = (A^B)$, **where** $A = C$])
show $\textit{eval-func}\ (A^B)\ C \circ_c id_c\ C \times_f \varphi^{\sharp\sharp} \circ_c \psi^\sharp =$
$\textit{eval-func}\ (A^B)\ C \circ_c id_c\ C \times_f id_c\ (A^{BC})$
proof(*etcs-rule same-evals-equal*[**where** $X = A$, **where** $A = B$])
show $\textit{eval-func}\ A\ B \circ_c id_c\ B \times_f (\textit{eval-func}\ (A^B)\ C \circ_c (id_c\ C \times_f \varphi^{\sharp\sharp} \circ_c \psi^\sharp))$
$=$
$\textit{eval-func}\ A\ B \circ_c id_c\ B \times_f \textit{eval-func}\ (A^B)\ C \circ_c id_c\ C \times_f id_c\ (A^{BC})$
proof –
have $\textit{eval-func}\ A\ B \circ_c id_c\ B \times_f (\textit{eval-func}\ (A^B)\ C \circ_c (id_c\ C \times_f \varphi^{\sharp\sharp} \circ_c \psi^\sharp)) =$
$\textit{eval-func}\ A\ B \circ_c id_c\ B \times_f (\textit{eval-func}\ (A^B)\ C \circ_c (id_c\ C \times_f \varphi^{\sharp\sharp}) \circ_c (id_c\ C \times_f \psi^\sharp))$
by (*typecheck-cfuncs*, *metis identity-distributes-across-composition*)
also have ... $= \textit{eval-func}\ A\ B \circ_c id_c\ B \times_f ((\textit{eval-func}\ (A^B)\ C \circ_c (id_c\ C \times_f \varphi^{\sharp\sharp})) \circ_c (id_c\ C \times_f \psi^\sharp))$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... $= \textit{eval-func}\ A\ B \circ_c id_c\ B \times_f (\varphi^\sharp \circ_c (id_c\ C \times_f \psi^\sharp))$
by (*typecheck-cfuncs*, *simp add*: *transpose-func-def*)
also have ... $= \textit{eval-func}\ A\ B \circ_c ((id_c\ B \times_f \varphi^\sharp) \circ_c (id_c\ B \times_f (id_c\ C \times_f \psi^\sharp)))$
using *identity-distributes-across-composition* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\textit{eval-func}\ A\ B \circ_c ((id_c\ B \times_f \varphi^\sharp))) \circ_c (id_c\ B \times_f (id_c\ C \times_f \psi^\sharp))$
using *comp-associative2* **by** (*typecheck-cfuncs*,*blast*)
also have ... $= \varphi \circ_c (id_c\ B \times_f (id_c\ C \times_f \psi^\sharp))$
by (*typecheck-cfuncs*, *simp add*: *transpose-func-def*)
also have ... $= ((\textit{eval-func}\ A\ (B \times_c\ C)) \circ_c (\textit{associate-left}\ B\ C\ (A^{(B \times_c\ C)}))) \circ_c (id_c\ B \times_f (id_c\ C \times_f \psi^\sharp))$
by (*simp add*: φ*-def*)
also have ... $= (\textit{eval-func}\ A\ (B \times_c\ C)) \circ_c (\textit{associate-left}\ B\ C\ (A^{(B \times_c\ C)})) \circ_c (id_c\ B \times_f (id_c\ C \times_f \psi^\sharp))$

using *comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*eval-func* A $(B \times_c C)$) $\circ_c$ $((id_c\ B \times_f id_c\ C) \times_f \psi^\sharp)$ $\circ_c$ *associate-left* B C $((A^B)^C)$
by (*typecheck-cfuncs*, *simp add*: *associate-left-crossprod-ap*)
also have ... = (*eval-func* A $(B \times_c C)$) $\circ_c$ $((id_c\ (B \times_c C)) \times_f \psi^\sharp)$ $\circ_c$ *associate-left* B C $((A^B)^C)$
by (*simp add*: *id-cross-prod*)
also have ... = $\psi \circ_c$ *associate-left* B C $((A^B)^C)$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 transpose-func-def*)
also have ... = ((*eval-func* A B) $\circ_c$ $(id(B) \times_f$ *eval-func* (A^B) C)) $\circ_c$ ((*associate-right* B C $((A^B)^C)$)$\circ_c$ *associate-left* B C $((A^B)^C)$)
by (*typecheck-cfuncs*, *simp add*: ψ-*def cfunc-type-def comp-associative*)
also have ... = ((*eval-func* A B) $\circ_c$ $(id(B) \times_f$ *eval-func* (A^B) C)) $\circ_c$ $id(B \times_c (C \times_c ((A^B)^C)))$
by (*simp add*: *right-left*)
also have ... = (*eval-func* A B) $\circ_c$ $(id(B) \times_f$ *eval-func* (A^B) C)
by (*typecheck-cfuncs*, *meson id-right-unit2*)
also have ... = *eval-func* A B $\circ_c$ $id_c\ B \times_f$ *eval-func* (A^B) C $\circ_c$ $id_c\ C \times_f id_c\ (A^{BC})$
by (*typecheck-cfuncs*, *simp add*: *id-cross-prod id-right-unit2*)
finally show *?thesis*.
qed
qed
qed
have $\psi^\sharp \circ_c \varphi^{\sharp\sharp} = id(A^{(B \times_c C)})$
proof(*etcs-rule same-evals-equal*[**where** $X = A$, **where** $A = (B \times_c C)$])
show *eval-func* A $(B \times_c C)$ $\circ_c$ $(id_c\ (B \times_c C) \times_f (\psi^\sharp \circ_c \varphi^{\sharp\sharp}))$ =
eval-func A $(B \times_c C)$ $\circ_c$ $id_c\ (B \times_c C) \times_f id_c\ (A^{(B \times_c C)})$
proof –
have *eval-func* A $(B \times_c C)$ $\circ_c$ $(id_c\ (B \times_c C) \times_f (\psi^\sharp \circ_c \varphi^{\sharp\sharp}))$ =
eval-func A $(B \times_c C)$ $\circ_c$ $((id_c\ (B \times_c C) \times_f (\psi^\sharp)) \circ_c (id_c\ (B \times_c C) \times_f \varphi^{\sharp\sharp}))$
by (*typecheck-cfuncs*, *simp add*: *identity-distributes-across-composition*)
also have ... = (*eval-func* A $(B \times_c C)$ $\circ_c$ $(id_c\ (B \times_c C) \times_f (\psi^\sharp)))$ $\circ_c$ $(id_c\ (B \times_c C) \times_f \varphi^{\sharp\sharp})$
using *comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... = $\psi \circ_c (id_c\ (B \times_c C) \times_f \varphi^{\sharp\sharp})$
by (*typecheck-cfuncs*, *simp add*: *transpose-func-def*)
also have ... =(*eval-func* A B) $\circ_c$ $(id(B) \times_f$ *eval-func* (A^B) C) $\circ_c$ (*associate-right* B C $((A^B)^C)$) $\circ_c$ $(id_c\ (B \times_c C) \times_f \varphi^{\sharp\sharp})$
by (*typecheck-cfuncs*, *smt* ψ-*def cfunc-type-def comp-associative domain-comp*)
also have ... =(*eval-func* A B) $\circ_c$ $(id(B) \times_f$ *eval-func* (A^B) C) $\circ_c$ (*associate-right* B C $((A^B)^C)$) $\circ_c$ $((id_c\ (B) \times_f id(\ C)) \times_f \varphi^{\sharp\sharp})$
by (*typecheck-cfuncs*, *simp add*: *id-cross-prod*)
also have ... =(*eval-func* A B) $\circ_c$ $((id(B) \times_f$ *eval-func* (A^B) C) $\circ_c$ $((id_c\ (B) \times_f (id(C) \times_f \varphi^{\sharp\sharp}))$ $\circ_c$ (*associate-right* B C $(A^{(B \times_c C)})$)))
using *associate-right-crossprod-ap* **by** (*typecheck-cfuncs*, *auto*)

also have ... $=$(*eval-func A B*) $\circ_c$ ((*id*(B)$\times_f$ *eval-func* (A^B) C) $\circ_c$ (*id*$_c$ (B) $\times_f$ (*id*(C) $\times_f$ $\varphi^{\sharp\sharp}$))) $\circ_c$ (*associate-right* B C ($A^{(B \times_c C)}$))
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... $=$(*eval-func A B*) $\circ_c$ (*id*(B)$\times_f$ ((*eval-func* (A^B) C)$\circ_c$ (*id*(C) $\times_f$ $\varphi^{\sharp\sharp}$))) $\circ_c$ (*associate-right* B C ($A^{(B \times_c C)}$))
using *identity-distributes-across-composition* **by** (*typecheck-cfuncs*, *auto*)
also have ... $=$(*eval-func A B*) $\circ_c$ (*id*(B)$\times_f$ $\varphi^{\sharp}$) $\circ_c$ (*associate-right* B C ($A^{(B \times_c C)}$))
by (*typecheck-cfuncs*, *simp add*: *transpose-func-def*)
also have ... $=$((*eval-func A B*) $\circ_c$ (*id*(B)$\times_f$ $\varphi^{\sharp}$)) $\circ_c$ (*associate-right* B C ($A^{(B \times_c C)}$))
using *comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \varphi \circ_c$ (*associate-right* B C ($A^{(B \times_c C)}$))
by (*typecheck-cfuncs*, *simp add*: *transpose-func-def*)
also have ... $=$ (*eval-func* A ($B\times_c C$)) $\circ_c$ ((*associate-left* B C ($A^{(B\times_c C)}$)) $\circ_c$ (*associate-right* B C ($A^{(B \times_c C)}$)))
by (*typecheck-cfuncs*, *simp add*: φ*-def comp-associative2*)
also have ... $=$ *eval-func* A ($B\times_c C$) $\circ_c$ *id* (($B \times_c C$) $\times_c$ ($A^{(B\times_c C)}$))
by (*typecheck-cfuncs*, *simp add*: *left-right*)
also have ... $=$ *eval-func* A ($B \times_c C$) $\circ_c$ *id*$_c$ ($B \times_c C$) $\times_f$ *id*$_c$ ($A^{(B \times_c C)}$)
by (*typecheck-cfuncs*, *simp add*: *id-cross-prod*)
finally show *?thesis***.**
qed
qed
show *?thesis*
by (*metis* ‹$\varphi^{\sharp\sharp} \circ_c \psi^{\sharp} = id_c\ (A^{BC})$› ‹$\psi^{\sharp} \circ_c \varphi^{\sharp\sharp} = id_c\ (A^{(B \times_c C)})$› φ*dbsharp-type* ψ*sharp-type cfunc-type-def is-isomorphic-def isomorphism-def*)
qed

lemma *exp-pres-iso-right*:
assumes $A \cong X$
shows $Y^A \cong Y^X$
proof −
obtain φ **where** φ*-def*: φ: $X \rightarrow A \wedge$ *isomorphism*(φ)
using *assms is-isomorphic-def isomorphic-is-symmetric* **by** *blast*
obtain ψ **where** ψ*-def*: ψ: $A \rightarrow X \wedge$ *isomorphism*(ψ) $\wedge$ ($\psi \circ_c \varphi = id(X)$)
using φ*-def cfunc-type-def isomorphism-def* **by** *fastforce*
have *idA*: $\varphi \circ_c \psi = id(A)$
by (*metis* φ*-def* ψ*-def cfunc-type-def comp-associative id-left-unit2 isomorphism-def*)
obtain f **where** f*-def*: f = (*eval-func* Y X) $\circ_c$ ($\psi \times_f id(Y^X)$) **and** f*-type*[*type-rule*]: f: $A\times_c (Y^X) \rightarrow Y$ **and** *fsharp-type*[*type-rule*]: $f^\sharp : Y^X \rightarrow Y^A$
using ψ*-def transpose-func-type* **by** (*typecheck-cfuncs*, *presburger*)
obtain g **where** g*-def*: g = (*eval-func* Y A) $\circ_c$ ($\varphi \times_f id(Y^A)$) **and** g*-type*[*type-rule*]: g: $X\times_c (Y^A) \rightarrow Y$ **and** *gsharp-type*[*type-rule*]: $g^\sharp : Y^A \rightarrow Y^X$
using φ*-def transpose-func-type* **by** (*typecheck-cfuncs*, *presburger*)

have *fsharp-gsharp-id*: $f^\sharp \circ_c g^\sharp = id(Y^A)$
proof(*etcs-rule same-evals-equal*[**where** $X = Y$, **where** $A = A$])
have *eval-func* $Y\ A \circ_c id_c\ A \times_f f^\sharp \circ_c g^\sharp =$ *eval-func* $Y\ A \circ_c (id_c\ A \times_f f^\sharp) \circ_c (id_c\ A \times_f g^\sharp)$
using *fsharp-type gsharp-type identity-distributes-across-composition* **by** *auto*
also have ... = *eval-func* $Y\ X \circ_c (\psi \times_f id(Y^X)) \circ_c (id_c\ A \times_f g^\sharp)$
using *ψ-def cfunc-type-def comp-associative f-def f-type gsharp-type transpose-func-def* **by** (*typecheck-cfuncs*, *smt*)
also have ... = *eval-func* $Y\ X \circ_c (\psi \times_f g^\sharp)$
by (*smt ψ-def cfunc-cross-prod-comp-cfunc-cross-prod gsharp-type id-left-unit2 id-right-unit2 id-type*)
also have ... = *eval-func* $Y\ X \circ_c (id\ X \times_f g^\sharp) \circ_c (\psi \times_f id(Y^A))$
by (*smt ψ-def cfunc-cross-prod-comp-cfunc-cross-prod gsharp-type id-left-unit2 id-right-unit2 id-type*)
also have ... = *eval-func* $Y\ A \circ_c (\varphi \times_f id(Y^A)) \circ_c (\psi \times_f id(Y^A))$
by (*typecheck-cfuncs*, *smt φ-def ψ-def comp-associative2 flat-cancels-sharp g-def g-type inv-transpose-func-def3*)
also have ... = *eval-func* $Y\ A \circ_c ((\varphi \circ_c \psi) \times_f (id(Y^A) \circ_c id(Y^A)))$
using *φ-def ψ-def cfunc-cross-prod-comp-cfunc-cross-prod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *eval-func* $Y\ A \circ_c id(A) \times_f id(Y^A)$
using *idA id-right-unit2* **by** (*typecheck-cfuncs*, *auto*)
finally show *eval-func* $Y\ A \circ_c id_c\ A \times_f f^\sharp \circ_c g^\sharp =$ *eval-func* $Y\ A \circ_c id_c\ A \times_f id_c\ (Y^A)$.
qed

have *gsharp-fsharp-id*: $g^\sharp \circ_c f^\sharp = id(Y^X)$
proof(*etcs-rule same-evals-equal*[**where** $X = Y$, **where** $A = X$])
have *eval-func* $Y\ X \circ_c id_c\ X \times_f g^\sharp \circ_c f^\sharp =$ *eval-func* $Y\ X \circ_c (id_c\ X \times_f g^\sharp) \circ_c (id_c\ X \times_f f^\sharp)$
using *fsharp-type gsharp-type identity-distributes-across-composition* **by** *auto*
also have ... = *eval-func* $Y\ A \circ_c (\varphi \times_f id_c\ (Y^A)) \circ_c (id_c\ X \times_f f^\sharp)$
using *φ-def cfunc-type-def comp-associative fsharp-type g-def g-type transpose-func-def* **by** (*typecheck-cfuncs*, *smt*)
also have ... = *eval-func* $Y\ A \circ_c (\varphi \times_f f^\sharp)$
by (*smt φ-def cfunc-cross-prod-comp-cfunc-cross-prod fsharp-type id-left-unit2 id-right-unit2 id-type*)
also have ... = *eval-func* $Y\ A \circ_c (id(A) \times_f f^\sharp) \circ_c (\varphi \times_f id_c\ (Y^X))$
by (*smt φ-def cfunc-cross-prod-comp-cfunc-cross-prod fsharp-type id-left-unit2 id-right-unit2 id-type*)
also have ... = *eval-func* $Y\ X \circ_c (\psi \times_f id_c\ (Y^X)) \circ_c (\varphi \times_f id_c\ (Y^X))$
by (*typecheck-cfuncs*, *smt φ-def ψ-def comp-associative2 f-def f-type flat-cancels-sharp inv-transpose-func-def3*)
also have ... = *eval-func* $Y\ X \circ_c ((\psi \circ_c \varphi) \times_f (id(Y^X) \circ_c id(Y^X)))$
using *φ-def ψ-def cfunc-cross-prod-comp-cfunc-cross-prod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *eval-func* $Y\ X \circ_c id(X) \times_f id(Y^X)$
using *ψ-def id-left-unit2* **by** (*typecheck-cfuncs*, *auto*)

finally show *eval-func* Y X $\circ_c$ id_c X $\times_f$ $g^\sharp$ $\circ_c$ $f^\sharp$ $=$ *eval-func* Y X $\circ_c$ id_c X $\times_f$ id_c (Y^X).
qed
show *?thesis*
by (*metis cfunc-type-def comp-epi-imp-epi comp-monic-imp-monic epi-mon-is-iso fsharp-gsharp-id fsharp-type gsharp-fsharp-id gsharp-type id-isomorphism is-isomorphic-def iso-imp-epi-and-monic*)
qed

lemma *exp-pres-iso*:
assumes $A \cong X$ $B \cong Y$
shows $A^B \cong X^Y$
by (*meson assms exp-pres-iso-left exp-pres-iso-right isomorphic-is-transitive*)

lemma *empty-to-nonempty*:
assumes *nonempty* X *is-empty* Y
shows $Y^X \cong \emptyset$
by (*meson assms exp-pres-iso-left isomorphic-is-transitive no-el-iff-iso-empty empty-exp-nonempty*)

lemma *exp-is-empty*:
assumes *is-empty* X
shows $Y^X \cong \mathbf{1}$
using *assms exp-pres-iso-right isomorphic-is-transitive no-el-iff-iso-empty exp-empty*
by *blast*

lemma *nonempty-to-nonempty*:
assumes *nonempty* X *nonempty* Y
shows *nonempty*(Y^X)
by (*meson assms*(*2*) *comp-type nonempty-def terminal-func-type transpose-func-type*)

lemma *empty-to-nonempty-converse*:
assumes $Y^X \cong \emptyset$
shows *is-empty* Y $\wedge$ *nonempty* X
by (*metis is-empty-def exp-is-empty assms no-el-iff-iso-empty nonempty-def nonempty-to-nonempty single-elem-iso-one*)

The definition below corresponds to Definition 2.5.11 in Halvorson.

definition *powerset* :: *cset* $\Rightarrow$ *cset* ($\mathcal{P}$- [*101*]*100*) **where**
$\mathcal{P}$ $X = \Omega^X$

lemma *sets-squared*:
$A^\Omega \cong A \times_c A$
proof $-$
obtain φ **where** φ-*def*: $\varphi = \langle$*eval-func* A Ω $\circ_c$ $\langle$t $\circ_c$ β_{A^Ω}, $id(A^\Omega)\rangle$,
eval-func A Ω $\circ_c$ $\langle$f $\circ_c$ β_{A^Ω}, $id(A^\Omega)\rangle\rangle$ **and**
φ-*type*[*type-rule*]: $\varphi : A^\Omega \to A \times_c A$
by (*typecheck-cfuncs*, *simp*)
have *injective* φ

unfolding *injective-def*
proof(*clarify*)
fix *f g*
assume $f \in_c$ *domain* φ **then have** *f-type*[*type-rule*]: $f \in_c A^{\Omega}$
using φ*-type cfunc-type-def* **by** (*typecheck-cfuncs*, *auto*)
assume $g \in_c$ *domain* φ **then have** *g-type*[*type-rule*]: $g \in_c A^{\Omega}$
using φ*-type cfunc-type-def* **by** (*typecheck-cfuncs*, *auto*)
assume *eqs*: $\varphi \circ_c f = \varphi \circ_c g$
show $f = g$
proof(*etcs-rule one-separator*)
show $\bigwedge$*id-1*. *id-1* $\in_c$ **1** $\Longrightarrow f \circ_c$ *id-1* $= g \circ_c$ *id-1*
proof(*etcs-rule same-evals-equal*[**where** $X = A$, **where** $A = \Omega$])
fix *id-1*
assume *id1-is*: *id-1* $\in_c$ **1**
then have *id1-eq*: *id-1* $=$ *id*(**1**)
using *id-type one-unique-element* **by** *auto*

obtain *a1 a2* **where** *phi-f-def*: $\varphi \circ_c f = \langle a1, a2\rangle \wedge a1 \in_c A \wedge a2 \in_c A$
using φ*-type cart-prod-decomp comp-type f-type* **by** *blast*
have *equation1*: $\langle a1, a2\rangle = \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t}, f\rangle$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f}, f\rangle\rangle$
proof −
have $\langle a1, a2\rangle = \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle\rangle \circ_c f$
using φ*-def phi-f-def* **by** *auto*
also have ... $= \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle \circ_c f$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle \circ_c f\rangle$
by (*typecheck-cfuncs*,*smt cfunc-prod-comp comp-associative2*)
also have ... $= \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}} \circ_c f, id(A^{\Omega}) \circ_c f\rangle$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}} \circ_c f, id(A^{\Omega})\circ_c f\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... $= \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t}, f\rangle$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f}, f\rangle\rangle$
by (*typecheck-cfuncs*, *metis id1-eq id1-is id-left-unit2 id-right-unit2 terminal-func-unique*)
finally show *?thesis*.
qed
have *equation2*: $\langle a1, a2\rangle = \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t}, g\rangle$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f}, g\rangle\rangle$
proof −
have $\langle a1, a2\rangle = \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle\rangle \circ_c g$
using φ*-def eqs phi-f-def* **by** *auto*
also have ... $= \langle$*eval-func* $A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle \circ_c g$,
eval-func $A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle \circ_c g\rangle$
by (*typecheck-cfuncs*,*smt cfunc-prod-comp comp-associative2*)

also have ... = ⟨*eval-func A* Ω $\circ_c$ ⟨t $\circ_c$ β_{A^Ω} $\circ_c$ *g*, *id*(A^Ω) $\circ_c$ *g*⟩,
eval-func A Ω $\circ_c$ ⟨f $\circ_c$ β_{A^Ω} $\circ_c$ *g*, *id*(A^Ω)$\circ_c$ *g* ⟩⟩
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... = ⟨*eval-func A* Ω $\circ_c$ ⟨t, *g*⟩,
eval-func A Ω $\circ_c$ ⟨f, *g*⟩⟩
by (*typecheck-cfuncs*, *metis id1-eq id1-is id-left-unit2 id-right-unit2 terminal-func-unique*)
finally show *?thesis*.
qed
have ⟨*eval-func A* Ω $\circ_c$ ⟨t, *f*⟩, *eval-func A* Ω $\circ_c$ ⟨f, *f*⟩⟩ =
⟨*eval-func A* Ω $\circ_c$ ⟨t, *g*⟩, *eval-func A* Ω $\circ_c$ ⟨f, *g*⟩⟩
using *equation1 equation2* **by** *auto*
then have *equation3*: (*eval-func A* Ω $\circ_c$ ⟨t, *f*⟩ = *eval-func A* Ω $\circ_c$ ⟨t, *g*⟩) ∧
(*eval-func A* Ω $\circ_c$ ⟨f, *f*⟩ = *eval-func A* Ω $\circ_c$ ⟨f, *g*⟩)
using *cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*)
have *eval-func A* Ω $\circ_c$ id_c Ω $\times_f$ *f* = *eval-func A* Ω $\circ_c$ id_c Ω $\times_f$ *g*
proof(*etcs-rule one-separator*)
fix *x*
assume *x-type*[*type-rule*]: *x* $\in_c$ Ω $\times_c$ **1**
then obtain *w i* **where** *x-def*: (*w* $\in_c$ Ω) ∧ (*i* $\in_c$ **1**) ∧ (*x* = ⟨*w*,*i*⟩)
using *cart-prod-decomp* **by** *blast*
then have *i-def*: *i* = *id*(**1**)
using *id1-eq id1-is one-unique-element* **by** *auto*
have *w-def*: (*w* = f) ∨ (*w* = t)
by (*simp add*: *true-false-only-truth-values x-def*)
then have *x-def2*: (*x* = ⟨f,*i*⟩) ∨ (*x* = ⟨t,*i*⟩)
using *x-def* **by** *auto*
show (*eval-func A* Ω $\circ_c$ id_c Ω $\times_f$ *f*) $\circ_c$ *x* = (*eval-func A* Ω $\circ_c$ id_c Ω $\times_f$ *g*) $\circ_c$ *x*
proof(*cases* (*x* = ⟨f,*i*⟩), *clarify*)
assume *case1*: *x* = ⟨f,*i*⟩
have (*eval-func A* Ω $\circ_c$ (id_c Ω $\times_f$ *f*)) $\circ_c$ ⟨f,*i*⟩ = *eval-func A* Ω $\circ_c$ ((id_c Ω $\times_f$ *f*) $\circ_c$ ⟨f,*i*⟩)
using *case1 comp-associative2 x-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *eval-func A* Ω $\circ_c$ ⟨id_c Ω $\circ_c$ f,*f* $\circ_c$ *i*⟩
using *cfunc-cross-prod-comp-cfunc-prod i-def id1-eq id1-is* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *eval-func A* Ω $\circ_c$ ⟨f, *f* ⟩
using *f-type false-func-type i-def id-left-unit2 id-right-unit2* **by** *auto*
also have ... = *eval-func A* Ω $\circ_c$ ⟨f, *g*⟩
using *equation3* **by** *blast*
also have ... = *eval-func A* Ω $\circ_c$ ⟨id_c Ω $\circ_c$ f,*g* $\circ_c$ *i*⟩
by (*typecheck-cfuncs*, *simp add*: *i-def id-left-unit2 id-right-unit2*)
also have ... = *eval-func A* Ω $\circ_c$ ((id_c Ω $\times_f$ *g*) $\circ_c$ ⟨f,*i*⟩)
using *cfunc-cross-prod-comp-cfunc-prod i-def id1-eq id1-is* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*eval-func A* Ω $\circ_c$ (id_c Ω $\times_f$ *g*)) $\circ_c$ ⟨f,*i*⟩
using *case1 comp-associative2 x-type* **by** (*typecheck-cfuncs*, *auto*)
finally show (*eval-func A* Ω $\circ_c$ id_c Ω $\times_f$ *f*) $\circ_c$ ⟨f,*i*⟩ = (*eval-func A* Ω

$\circ_c$ *id*$_c$ Ω $\times_f$ *g*) $\circ_c$ $\langle$f,*i*$\rangle$.
next
assume *case2*: $x \neq \langle$f,$i\rangle$
then have *x-eq*: $x = \langle$t,$i\rangle$
using *x-def2* **by** *blast*
have (*eval-func A* Ω $\circ_c$ (*id*$_c$ Ω $\times_f$ *f*)) $\circ_c$ $\langle$t,*i*$\rangle$ = *eval-func A* Ω $\circ_c$ ((*id*$_c$ Ω $\times_f$ *f*) $\circ_c$ $\langle$t,*i*$\rangle$)
using *case2 x-eq comp-associative2 x-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *eval-func A* Ω $\circ_c$ $\langle$*id*$_c$ Ω $\circ_c$ t,*f* $\circ_c$ *i*$\rangle$
using *cfunc-cross-prod-comp-cfunc-prod i-def id1-eq id1-is* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *eval-func A* Ω $\circ_c$ $\langle$t, *f* $\rangle$
using *f-type i-def id-left-unit2 id-right-unit2 true-func-type* **by** *auto*
also have ... = *eval-func A* Ω $\circ_c$ $\langle$t, *g*$\rangle$
using *equation3* **by** *blast*
also have ... = *eval-func A* Ω $\circ_c$ $\langle$*id*$_c$ Ω $\circ_c$ t,*g* $\circ_c$ *i*$\rangle$
by (*typecheck-cfuncs*, *simp add*: *i-def id-left-unit2 id-right-unit2*)
also have ... = *eval-func A* Ω $\circ_c$ ((*id*$_c$ Ω $\times_f$ *g*) $\circ_c$ $\langle$t,*i*$\rangle$)
using *cfunc-cross-prod-comp-cfunc-prod i-def id1-eq id1-is* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*eval-func A* Ω $\circ_c$ (*id*$_c$ Ω $\times_f$ *g*)) $\circ_c$ $\langle$t,*i*$\rangle$
using *comp-associative2 x-eq x-type* **by** (*typecheck-cfuncs*, *blast*)
ultimately show (*eval-func A* Ω $\circ_c$ *id*$_c$ Ω $\times_f$ *f*) $\circ_c$ *x* = (*eval-func A* Ω $\circ_c$ *id*$_c$ Ω $\times_f$ *g*) $\circ_c$ *x*
by (*simp add*: *x-eq*)
qed
qed
then show *eval-func A* Ω $\circ_c$ *id*$_c$ Ω $\times_f$ *f* $\circ_c$ *id-1* = *eval-func A* Ω $\circ_c$ *id*$_c$ Ω $\times_f$ *g* $\circ_c$ *id-1*
using *f-type g-type same-evals-equal* **by** *blast*
qed
qed
qed
then have *monomorphism*(φ)
using *injective-imp-monomorphism* **by** *auto*
have *surjective*(φ)
unfolding *surjective-def*
proof(*clarify*)
fix *y*
assume $y \in_c$ *codomain* φ **then have** *y-type*[*type-rule*]: $y \in_c A \times_c A$
using φ*-type cfunc-type-def* **by** *auto*
then obtain *a1 a2* **where** *y-def*[*type-rule*]: $y = \langle a1, a2\rangle \wedge a1 \in_c A \wedge a2 \in_c A$
using *cart-prod-decomp* **by** *blast*
then have *aua*: (*a1* II *a2*): $\mathbf{1} \coprod \mathbf{1} \rightarrow A$
by (*typecheck-cfuncs*, *simp add*: *y-def*)

obtain *f* **where** *f-def*: *f* = ((*a1* II *a2*) $\circ_c$ *case-bool* $\circ_c$ *left-cart-proj* Ω $\mathbf{1}$)$^\sharp$ **and**

f-type[*type-rule*]: $f \in_c A^{\Omega}$
by (*meson aua case-bool-type comp-type left-cart-proj-type transpose-func-type*)
have *a1-is*: $(\textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle) \circ_c f = a1$
proof−
have $(\textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle) \circ_c f = \textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle \circ_c f$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{t} \circ_c \beta_{A^{\Omega}} \circ_c f, id(A^{\Omega}) \circ_c f\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{t}, f\rangle$
by (*metis cfunc-type-def f-type id-left-unit id-right-unit id-type one-unique-element terminal-func-comp terminal-func-type true-func-type*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c \langle id(\Omega) \circ_c \mathrm{t}, f \circ_c id(\mathbf{1})\rangle$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c (id(\Omega) \times_f f) \circ_c \langle \mathrm{t}, id(\mathbf{1})\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... $= (\textit{eval-func}\ A\ \Omega \circ_c (id(\Omega) \times_f f)) \circ_c \langle \mathrm{t}, id(\mathbf{1})\rangle$
using *comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= ((a1 \amalg a2) \circ_c \textit{case-bool} \circ_c \textit{left-cart-proj}\ \Omega\ \mathbf{1}) \circ_c \langle \mathrm{t}, id(\mathbf{1})\rangle$
by (*typecheck-cfuncs*, *metis aua f-def flat-cancels-sharp inv-transpose-func-def3*)
also have ... $= (a1 \amalg a2) \circ_c \textit{case-bool} \circ_c \mathrm{t}$
by (*typecheck-cfuncs*, *smt case-bool-type aua comp-associative2 left-cart-proj-cfunc-prod*)
also have ... $= (a1 \amalg a2) \circ_c \textit{left-coproj}\ \mathbf{1}\ \mathbf{1}$
by (*simp add*: *case-bool-true*)
also have ... $= a1$
using *left-coproj-cfunc-coprod y-def* **by** *blast*
finally show *?thesis*.
qed
have *a2-is*: $(\textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle) \circ_c f = a2$
proof−
have $(\textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle) \circ_c f = \textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}}, id(A^{\Omega})\rangle \circ_c f$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{f} \circ_c \beta_{A^{\Omega}} \circ_c f, id(A^{\Omega}) \circ_c f\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c \langle \mathrm{f}, f\rangle$
by (*metis cfunc-type-def f-type id-left-unit id-right-unit id-type one-unique-element terminal-func-comp terminal-func-type false-func-type*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c \langle id(\Omega) \circ_c \mathrm{f}, f \circ_c id(\mathbf{1})\rangle$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2*)
also have ... $= \textit{eval-func}\ A\ \Omega \circ_c (id(\Omega) \times_f f) \circ_c \langle \mathrm{f}, id(\mathbf{1})\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... $= (\textit{eval-func}\ A\ \Omega \circ_c (id(\Omega) \times_f f)) \circ_c \langle \mathrm{f}, id(\mathbf{1})\rangle$
using *comp-associative2* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= ((a1 \amalg a2) \circ_c \textit{case-bool} \circ_c \textit{left-cart-proj}\ \Omega\ \mathbf{1}) \circ_c \langle \mathrm{f}, id(\mathbf{1})\rangle$
by (*typecheck-cfuncs*, *metis aua f-def flat-cancels-sharp inv-transpose-func-def3*)
also have ... $= (a1 \amalg a2) \circ_c \textit{case-bool} \circ_c \mathrm{f}$

by (*typecheck-cfuncs*, *smt aua comp-associative2 left-cart-proj-cfunc-prod*)
also have ... = (*a1* ∐ *a2*) $\circ_c$ *right-coproj* **1 1**
by (*simp add*: *case-bool-false*)
also have ... = *a2*
using *right-coproj-cfunc-coprod y-def* **by** *blast*
finally show *?thesis*.
qed
have $\varphi \circ_c f = \langle a1, a2 \rangle$
unfolding *φ-def* **by** (*typecheck-cfuncs*, *simp add*: *a1-is a2-is cfunc-prod-comp*)
then show $\exists x.\ x \in_c$ *domain* $\varphi \wedge \varphi \circ_c x = y$
using *φ-type cfunc-type-def f-type y-def* **by** *auto*
qed
then have *epimorphism*(φ)
by (*simp add*: *surjective-is-epimorphism*)
then have *isomorphism*(φ)
by (*simp add*: ‹*monomorphism* φ› *epi-mon-is-iso*)
then show *?thesis*
using *φ-type is-isomorphic-def* **by** *blast*
qed

end

13 Natural Number Object

theory *Nats*
imports *Exponential-Objects*
begin

The axiomatization below corresponds to Axiom 10 (Natural Number Object) in Halvorson.

axiomatization
natural-numbers :: *cset* ($\mathbb{N}_c$) **and**
zero :: *cfunc* **and**
successor :: *cfunc*
where
zero-type[*type-rule*]: *zero* $\in_c \mathbb{N}_c$ **and**
successor-type[*type-rule*]: *successor*: $\mathbb{N}_c \to \mathbb{N}_c$ **and**
natural-number-object-property:
$q : \mathbf{1} \to X \Longrightarrow f: X \to X \Longrightarrow$
$(\exists! u.\ u: \mathbb{N}_c \to X \wedge$
$q = u \circ_c zero \wedge$
$f \circ_c u = u \circ_c successor)$

lemma *beta-N-succ-nEqs-Id1*:
assumes *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
shows $\beta_{\mathbb{N}_c} \circ_c successor \circ_c n = id\ \mathbf{1}$
by (*typecheck-cfuncs*, *simp add*: *terminal-func-comp-elem*)

lemma *natural-number-object-property2*:

assumes $q : \mathbf{1} \to X$ f: $X \to X$
shows $\exists!u.$ u: $\mathbb{N}_c \to X \wedge u \circ_c$ *zero* $= q \wedge f \circ_c u = u \circ_c$ *successor*
using *assms natural-number-object-property*[**where** q=q, **where** f=f, **where** X=X]
by *metis*

lemma *natural-number-object-func-unique*:
assumes *u-type*: $u : \mathbb{N}_c \to X$ **and** *v-type*: $v : \mathbb{N}_c \to X$ **and** *f-type*: f: $X \to X$
assumes *zeros-eq*: $u \circ_c$ *zero* $= v \circ_c$ *zero*
assumes *u-successor-eq*: $u \circ_c$ *successor* $= f \circ_c u$
assumes *v-successor-eq*: $v \circ_c$ *successor* $= f \circ_c v$
shows $u = v$
by (*smt* (*verit*, *best*) *comp-type f-type natural-number-object-property2 u-successor-eq u-type v-successor-eq v-type zero-type zeros-eq*)

definition *is-NNO* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *bool* **where**
is-NNO Y z s ⟷(z: $\mathbf{1} \to Y \wedge s$: $Y \to Y \wedge (\forall X f q. ((q : \mathbf{1} \to X) \wedge (f$: $X \to X))$⟶
$(\exists!u.$ u: $Y \to X \wedge$
$q = u \circ_c z \wedge$
$f \circ_c u = u \circ_c s)))$

lemma *N-is-a-NNO*:
is-NNO $\mathbb{N}_c$ *zero successor*
by (*simp add*: *is-NNO-def natural-number-object-property successor-type zero-type*)

The lemma below corresponds to Exercise 2.6.5 in Halvorson.

lemma *NNOs-are-iso-N*:
assumes *is-NNO* N z s
shows $N \cong \mathbb{N}_c$
proof−
have *z-type*[*type-rule*]: $(z : \mathbf{1} \to N)$
using *assms is-NNO-def* **by** *blast*
have *s-type*[*type-rule*]: $(s : N \to N)$
using *assms is-NNO-def* **by** *blast*
then obtain u **where** *u-type*[*type-rule*]: u: $\mathbb{N}_c \to N$
and *u-triangle*: $u \circ_c$ *zero* $= z$
and *u-square*: $s \circ_c u = u \circ_c$ *successor*
using *natural-number-object-property z-type* **by** *blast*
obtain v **where** *v-type*[*type-rule*]: v: $N \to \mathbb{N}_c$
and *v-triangle*: $v \circ_c z =$ *zero*
and *v-square*: *successor* $\circ_c v = v \circ_c s$
by (*metis assms is-NNO-def successor-type zero-type*)
then have *vuzeroEqzero*: $v \circ_c (u \circ_c$ *zero*$)$ = *zero*
by (*simp add*: *u-triangle v-triangle*)
have *id-facts1*: $id(\mathbb{N}_c)$: $\mathbb{N}_c \to \mathbb{N}_c \wedge id(\mathbb{N}_c) \circ_c$ *zero* = *zero* $\wedge$
(*successor* $\circ_c id(\mathbb{N}_c) = id(\mathbb{N}_c) \circ_c$ *successor*)
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2*)
then have *vu-facts*: $v \circ_c u$: $\mathbb{N}_c \to \mathbb{N}_c \wedge (v \circ_c u) \circ_c$ *zero* = *zero* $\wedge$

successor $\circ_c$ $(v \circ_c u) = (v \circ_c u) \circ_c$ *successor*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 s-type u-square v-square vuzeroEqzero*)
then have *half-isomorphism*: $(v \circ_c u) = id(\mathbb{N}_c)$
by (*metis id-facts1 natural-number-object-property successor-type vu-facts zero-type*)
have *uvzEqz*: $u \circ_c (v \circ_c z) = z$
by (*simp add*: *u-triangle v-triangle*)
have *id-facts2*: $id(N)$: $N \to N \wedge id(N) \circ_c z = z \wedge s \circ_c id(N) = id(N) \circ_c \ s$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2*)
then have *uv-facts*: $u \circ_c v$: $N \to N \wedge$
$(u \circ_c v) \circ_c \ z = z \wedge \ s \circ_c (u \circ_c v) = \ (u \circ_c v) \circ_c s$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 successor-type u-square uvzEqz v-square*)
then have *half-isomorphism2*: $(u \circ_c v) = id(N)$
by (*smt* (*verit*, *ccfv-threshold*) *assms id-facts2 is-NNO-def*)
then show $N \cong \mathbb{N}_c$
using *cfunc-type-def half-isomorphism is-isomorphic-def isomorphism-def u-type v-type* **by** *fastforce*
qed

The lemma below is the converse to Exercise 2.6.5 in Halvorson.

lemma *Iso-to-N-is-NNO*:
assumes $N \cong \mathbb{N}_c$
shows $\exists$ *z s*. *is-NNO N z s*
proof −
obtain *i* **where** *i-type*[*type-rule*]: i: $\mathbb{N}_c \to N$ **and** *i-iso*: *isomorphism*(i)
using *assms isomorphic-is-symmetric is-isomorphic-def* **by** *blast*
obtain *z* **where** *z-type*[*type-rule*]: $z \in_c N$ **and** *z-def*: $z = i \circ_c$ *zero*
by (*typecheck-cfuncs*, *simp*)
obtain *s* **where** *s-type*[*type-rule*]: s: $N \to N$ **and** *s-def*: $s = (i \circ_c$ *successor*$) \circ_c i^{-\mathbf{1}}$
using *i-iso* **by** (*typecheck-cfuncs*, *simp*)
have *is-NNO N z s*
unfolding *is-NNO-def*
proof(*typecheck-cfuncs*)
fix *X q f*
assume *q-type*[*type-rule*]: q: $\mathbf{1} \to X$
assume *f-type*[*type-rule*]: f: $X \to X$

obtain *u* **where** *u-type*[*type-rule*]: u: $\mathbb{N}_c \to X$ **and** *u-def*: $u \circ_c$ *zero* $= \ q \wedge f \circ_c u = u \circ_c$ *successor*
using *natural-number-object-property2* **by** (*typecheck-cfuncs*, *blast*)
obtain *v* **where** *v-type*[*type-rule*]: v: $N \to X$ **and** *v-def*: $v = u \circ_c i^{-\mathbf{1}}$
using *i-iso* **by** (*typecheck-cfuncs*, *simp*)
then have *bottom-triangle*: $v \circ_c z = q$
unfolding *v-def u-def z-def* **using** *i-iso*
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative id-right-unit2 inv-left u-def*)
have *bottom-square*: $v \circ_c s = f \circ_c v$

unfolding *v-def u-def s-def* **using** *i-iso*
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *comp-associative2 id-right-unit2 inv-left u-def*)
show $\exists!u.\ u : N \to X \wedge q = u \circ_c z \wedge f \circ_c u = u \circ_c s$
proof *safe*
show $\exists u.\ u : N \to X \wedge q = u \circ_c z \wedge f \circ_c u = u \circ_c s$
by (*intro exI*[**where** *x=v*], *auto simp add*: *bottom-triangle bottom-square v-type*)
next
fix *w y*
assume *w-type*[*type-rule*]: $w : N \to X$
assume *y-type*[*type-rule*]: $y : N \to X$
assume *f-w*: $f \circ_c w = w \circ_c s$
assume *f-y*: $f \circ_c y = y \circ_c s$
assume *w-y-z*: $w \circ_c z = y \circ_c z$
assume *q-def*: $q = w \circ_c z$

have $w \circ_c i = u$
proof (*etcs-rule natural-number-object-func-unique*[**where** *f=f*])
show $(w \circ_c i) \circ_c zero = u \circ_c zero$
using *q-def u-def w-y-z z-def* **by** (*etcs-assocr*, *argo*)
show $(w \circ_c i) \circ_c successor = f \circ_c w \circ_c i$
using *i-iso* **by** (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 comp-type f-w id-right-unit2 inv-left inverse-type s-def*)
show $u \circ_c successor = f \circ_c u$
by (*simp add*: *u-def*)
qed
then have *w-eq-v*: $w = v$
unfolding *v-def* **using** *i-iso*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 id-right-unit2 inv-right*)

have $y \circ_c i = u$
proof (*etcs-rule natural-number-object-func-unique*[**where** *f=f*])
show $(y \circ_c i) \circ_c zero = u \circ_c zero$
using *q-def u-def w-y-z z-def* **by** (*etcs-assocr*, *argo*)
show $(y \circ_c i) \circ_c successor = f \circ_c y \circ_c i$
using *i-iso* **by** (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 comp-type f-y id-right-unit2 inv-left inverse-type s-def*)
show $u \circ_c successor = f \circ_c u$
by (*simp add*: *u-def*)
qed
then have *y-eq-v*: $y = v$
unfolding *v-def* **using** *i-iso*
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *comp-associative2 id-right-unit2 inv-right*)
show $w = y$

using *w-eq-v y-eq-v* **by** *auto*
qed
qed
then show *?thesis*
by *auto*
qed

13.1 Zero and Successor

lemma *zero-is-not-successor*:
assumes $n \in_c \mathbb{N}_c$
shows $zero \neq successor \circ_c n$
proof (*rule ccontr*, *clarify*)
assume *for-contradiction*: $zero = successor \circ_c n$
have $\exists! u.\ u: \mathbb{N}_c \to \Omega \wedge u \circ_c zero = \mathrm{t} \wedge (\mathrm{f} \circ_c \beta_\Omega) \circ_c u = u \circ_c successor$
by (*typecheck-cfuncs*, *rule natural-number-object-property2*)
then obtain u **where** *u-type*: $u: \mathbb{N}_c \to \Omega$ **and**
u-triangle: $u \circ_c zero = \mathrm{t}$ **and**
u-square: $(\mathrm{f} \circ_c \beta_\Omega) \circ_c u = u \circ_c successor$
by *auto*
have $\mathrm{t} = \mathrm{f}$
proof −
have $\mathrm{t} = u \circ_c zero$
by (*simp add*: *u-triangle*)
also have ... $= u \circ_c successor \circ_c n$
by (*simp add*: *for-contradiction*)
also have ... $= (\mathrm{f} \circ_c \beta_\Omega) \circ_c u \circ_c n$
using *assms u-type* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 u-square*)
also have ... $= \mathrm{f}$
using *assms u-type* **by** (*etcs-assocr*,*typecheck-cfuncs*, *simp add*: *id-right-unit2 terminal-func-comp-elem*)
finally show *?thesis*.
qed
then show *False*
using *true-false-distinct* **by** *blast*
qed

The lemma below corresponds to Proposition 2.6.6 in Halvorson.

lemma *oneUN-iso-N-isomorphism*:
isomorphism($zero \amalg successor$)
proof −
obtain *i0* **where** *i0-type*[*type-rule*]: $i0: \mathbf{1} \to (\mathbf{1} \coprod \mathbb{N}_c)$ **and** *i0-def*: $i0 = $ *left-coproj* $\mathbf{1}\ \mathbb{N}_c$
by (*typecheck-cfuncs*, *simp*)
obtain *i1* **where** *i1-type*[*type-rule*]: $i1: \mathbb{N}_c \to (\mathbf{1} \coprod \mathbb{N}_c)$ **and** *i1-def*: $i1 = $ *right-coproj* $\mathbf{1}\ \mathbb{N}_c$
by (*typecheck-cfuncs*, *simp*)
obtain g **where** *g-type*[*type-rule*]: $g: \mathbb{N}_c \to (\mathbf{1} \coprod \mathbb{N}_c)$ **and**
g-triangle: $g \circ_c zero = i0$ **and**

g-square: $g \circ_c successor = ((i1 \circ_c zero) \amalg (i1 \circ_c successor)) \circ_c g$
by (*typecheck-cfuncs*, *metis natural-number-object-property*)
then have *second-diagram3*: $g \circ_c (successor \circ_c zero) = (i1 \circ_c zero)$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *cfunc-coprod-type comp-associative2 comp-type i0-def left-coproj-cfunc-coprod*)
then have *g-s-s-Eqs-i1zUi1s-g-s*:
$(g \circ_c successor) \circ_c successor = ((i1 \circ_c zero) \amalg (i1 \circ_c successor)) \circ_c (g \circ_c successor)$
by (*typecheck-cfuncs*, *smt* (*verit*, *del-insts*) *comp-associative2 g-square*)
then have *g-s-s-zEqs-i1zUi1s-i1z*: $((g \circ_c successor) \circ_c successor) \circ_c zero =$
$((i1 \circ_c zero) \amalg (i1 \circ_c successor)) \circ_c (i1 \circ_c zero)$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *comp-associative2 g-square second-diagram3*)
then have *i1-sEqs-i1zUi1s-i1*: $i1 \circ_c successor = ((i1 \circ_c zero) \amalg (i1 \circ_c successor)) \circ_c i1$
by (*typecheck-cfuncs*, *simp add*: *i1-def right-coproj-cfunc-coprod*)
then obtain u **where** *u-type*[*type-rule*]: $(u: \mathbb{N}_c \rightarrow (\mathbf{1} \coprod \mathbb{N}_c))$ **and**
u-triangle: $u \circ_c zero = i1 \circ_c zero$ **and**
u-square: $u \circ_c successor = ((i1 \circ_c zero) \amalg (i1 \circ_c successor)) \circ_c u$
using *i1-sEqs-i1zUi1s-i1* **by** (*typecheck-cfuncs*, *blast*)
then have *u-Eqs-i1*: $u=i1$
by (*typecheck-cfuncs*, *meson cfunc-coprod-type comp-type i1-sEqs-i1zUi1s-i1 natural-number-object-func-unique successor-type zero-type*)
have *g-s-type*[*type-rule*]: $g \circ_c successor: \mathbb{N}_c \rightarrow (\mathbf{1} \coprod \mathbb{N}_c)$
by *typecheck-cfuncs*
have *g-s-triangle*: $(g \circ_c successor) \circ_c zero = i1 \circ_c zero$
using *comp-associative2 second-diagram3* **by** (*typecheck-cfuncs*, *force*)
then have *u-Eqs-g-s*: $u= g \circ_c successor$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *cfunc-coprod-type comp-type g-s-s-Eqs-i1zUi1s-g-s g-s-triangle i1-sEqs-i1zUi1s-i1 natural-number-object-func-unique u-Eqs-i1 zero-type*)
then have *g-sEqs-i1*: $g \circ_c successor = i1$
using *u-Eqs-i1* **by** *blast*
have *eq1*: $(zero \amalg successor) \circ_c g = id(\mathbb{N}_c)$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *cfunc-coprod-comp comp-associative2 g-square g-triangle i0-def i1-def i1-type id-left-unit2 id-right-unit2 left-coproj-cfunc-coprod natural-number-object-func-unique right-coproj-cfunc-coprod*)
then have *eq2*: $g \circ_c (zero \amalg successor) = id(\mathbf{1} \coprod \mathbb{N}_c)$
by (*typecheck-cfuncs*, *metis cfunc-coprod-comp g-sEqs-i1 g-triangle i0-def i1-def id-coprod*)
show *isomorphism*$(zero \amalg successor)$
using *cfunc-coprod-type eq1 eq2 g-type isomorphism-def3 successor-type zero-type*
by *blast*
qed

lemma *zUs-epic*:
epimorphism$(zero \amalg successor)$
by (*simp add*: *iso-imp-epi-and-monic oneUN-iso-N-isomorphism*)

lemma *zUs-surj*:

surjective(*zero* ⨿ *successor*)
by (*simp add*: *cfunc-type-def epi-is-surj zUs-epic*)

lemma *nonzero-is-succ-aux*:
assumes $x \in_c (\mathbf{1} \coprod \mathbb{N}_c)$
shows $(x = (\textit{left-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c \textit{id}\ \mathbf{1}) \vee$
$(\exists\, n.\ (n \in_c \mathbb{N}_c) \wedge (x = (\textit{right-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c n))$
by(*clarify*, *metis assms coprojs-jointly-surj id-type one-unique-element*)

lemma *nonzero-is-succ*:
assumes $k \in_c \mathbb{N}_c$
assumes $k \neq \textit{zero}$
shows $\exists\, n.(n \in_c \mathbb{N}_c \wedge k = \textit{successor} \circ_c n)$
proof −
have *x-exists*: $\exists\, x.\ ((x \in_c \mathbf{1} \coprod \mathbb{N}_c) \wedge (\textit{zero} \amalg \textit{successor} \circ_c x = k))$
using *assms cfunc-type-def surjective-def zUs-surj* **by** (*typecheck-cfuncs*, *auto*)
obtain *x* **where** *x-def*: $((x \in_c \mathbf{1} \coprod \mathbb{N}_c) \wedge (\textit{zero} \amalg \textit{successor} \circ_c x = k))$
using *x-exists* **by** *blast*
have *cases*: $(x = (\textit{left-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c \textit{id}\ \mathbf{1}) \vee$
$(\exists\, n.\ (n \in_c \mathbb{N}_c \wedge x = (\textit{right-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c n))$
by (*simp add*: *nonzero-is-succ-aux x-def*)
have *not-case-1*: $x \neq (\textit{left-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c \textit{id}\ \mathbf{1}$
proof(*rule ccontr*,*clarify*)
assume *bwoc*: $x = \textit{left-coproj}\ \mathbf{1}\ \mathbb{N}_c \circ_c \textit{id}_c\ \mathbf{1}$
have *contradiction*: $k = \textit{zero}$
by (*metis bwoc id-right-unit2 left-coproj-cfunc-coprod left-proj-type successor-type x-def zero-type*)
show *False*
using *contradiction assms*(*2*) **by** *force*
qed
then obtain *n* **where** *n-def*: $n \in_c \mathbb{N}_c \wedge x = (\textit{right-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c n$
using *cases* **by** *blast*
then have $k = \textit{zero} \amalg \textit{successor} \circ_c x$
using *x-def* **by** *blast*
also have ... $= \textit{zero} \amalg \textit{successor} \circ_c\ \textit{right-coproj}\ \mathbf{1}\ \mathbb{N}_c \circ_c n$
by (*simp add*: *n-def*)
also have ... $= (\textit{zero} \amalg \textit{successor} \circ_c\ \textit{right-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c n$
using *cfunc-coprod-type cfunc-type-def comp-associative n-def right-proj-type successor-type zero-type* **by** *auto*
also have ... $= \textit{successor} \circ_c n$
using *right-coproj-cfunc-coprod successor-type zero-type* **by** *auto*
finally show *?thesis*
using *n-def* **by** *auto*
qed

13.2 Predecessor

definition *predecessor′* :: *cfunc* **where**
$\textit{predecessor}' = (\textit{THE}\ f.\ f : \mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$

$\wedge$ $f \circ_c$ (*zero* $\amalg$ *successor*) = *id* ($\mathbf{1} \coprod \mathbb{N}_c$) $\wedge$ (*zero* $\amalg$ *successor*) $\circ_c f$ = *id* $\mathbb{N}_c$)

lemma *predecessor′-def2*:
predecessor′ : $\mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$ $\wedge$ *predecessor′* $\circ_c$ (*zero* $\amalg$ *successor*) = *id* ($\mathbf{1} \coprod$ $\mathbb{N}_c$)
$\wedge$ (*zero* $\amalg$ *successor*) $\circ_c$ *predecessor′* = *id* $\mathbb{N}_c$
unfolding *predecessor′-def*
proof (*rule theI′*, *safe*)
show $\exists x.\ x : \mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$ $\wedge$
$x \circ_c$ *zero* $\amalg$ *successor* = id_c ($\mathbf{1} \coprod \mathbb{N}_c$) $\wedge$ *zero* $\amalg$ *successor* $\circ_c x$ = id_c $\mathbb{N}_c$
using *oneUN-iso-N-isomorphism* **by** (*typecheck-cfuncs*, *unfold isomorphism-def cfunc-type-def*, *auto*)
next
fix *x y*
assume *x-type*[*type-rule*]: $x : \mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$ **and** *y-type*[*type-rule*]: $y : \mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$
assume *x-left-inv*: *zero* $\amalg$ *successor* $\circ_c x$ = id_c $\mathbb{N}_c$
assume $x \circ_c$ *zero* $\amalg$ *successor* = id_c ($\mathbf{1} \coprod \mathbb{N}_c$) $y \circ_c$ *zero* $\amalg$ *successor* = id_c ($\mathbf{1} \coprod \mathbb{N}_c$)
then have $x \circ_c$ *zero* $\amalg$ *successor* = $y \circ_c$ *zero* $\amalg$ *successor*
by *auto*
then have $x \circ_c$ *zero* $\amalg$ *successor* $\circ_c x$ = $y \circ_c$ *zero* $\amalg$ *successor* $\circ_c x$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
then show $x = y$
using *id-right-unit2 x-left-inv x-type y-type* **by** *auto*
qed

lemma *predecessor′-type*[*type-rule*]:
predecessor′ : $\mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$
by (*simp add*: *predecessor′-def2*)

lemma *predecessor′-left-inv*:
(*zero* $\amalg$ *successor*) $\circ_c$ *predecessor′* = *id* $\mathbb{N}_c$
by (*simp add*: *predecessor′-def2*)

lemma *predecessor′-right-inv*:
predecessor′ $\circ_c$ (*zero* $\amalg$ *successor*) = *id* ($\mathbf{1} \coprod \mathbb{N}_c$)
by (*simp add*: *predecessor′-def2*)

lemma *predecessor′-successor*:
predecessor′ $\circ_c$ *successor* = *right-coproj* $\mathbf{1}$ $\mathbb{N}_c$
proof −
have *predecessor′* $\circ_c$ *successor* = *predecessor′* $\circ_c$ (*zero* $\amalg$ *successor*) $\circ_c$ *right-coproj* $\mathbf{1}$ $\mathbb{N}_c$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*predecessor′* $\circ_c$ (*zero* $\amalg$ *successor*)) $\circ_c$ *right-coproj* $\mathbf{1}$ $\mathbb{N}_c$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *right-coproj* $\mathbf{1}$ $\mathbb{N}_c$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 predecessor′-def2*)

finally show *?thesis*.
qed

lemma *predecessor′-zero*:
predecessor′ $\circ_c$ *zero* = *left-coproj* **1** $\mathbb{N}_c$
proof −
have *predecessor′* $\circ_c$ *zero* = *predecessor′* $\circ_c$ (*zero* ⨿ *successor*) $\circ_c$ *left-coproj* **1** $\mathbb{N}_c$
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*predecessor′* $\circ_c$ (*zero* ⨿ *successor*)) $\circ_c$ *left-coproj* **1** $\mathbb{N}_c$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
also have ... = *left-coproj* **1** $\mathbb{N}_c$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 predecessor′-def2*)
finally show *?thesis*.
qed

definition *predecessor* :: *cfunc*
where *predecessor* = (*zero* ⨿ *id* $\mathbb{N}_c$) $\circ_c$ *predecessor′*

lemma *predecessor-type*[*type-rule*]:
predecessor : $\mathbb{N}_c \to \mathbb{N}_c$
unfolding *predecessor-def* **by** *typecheck-cfuncs*

lemma *predecessor-zero*:
predecessor $\circ_c$ *zero* = *zero*
unfolding *predecessor-def*
using *left-coproj-cfunc-coprod predecessor′-zero* **by** (*etcs-assocr*, *typecheck-cfuncs*, *presburger*)

lemma *predecessor-successor*:
predecessor $\circ_c$ *successor* = *id* $\mathbb{N}_c$
unfolding *predecessor-def*
by (*etcs-assocr*, *typecheck-cfuncs*, *metis* (*full-types*) *predecessor′-successor right-coproj-cfunc-coprod*)

13.3 Peano's Axioms and Induction

The lemma below corresponds to Proposition 2.6.7 in Halvorson.

lemma *Peano′s-Axioms*:
injective successor ∧ ¬ *surjective successor*
proof −
have *i1-mono*: *monomorphism*(*right-coproj* **1** $\mathbb{N}_c$)
by (*simp add*: *right-coproj-are-monomorphisms*)
have *zUs-iso*: *isomorphism*(*zero* ⨿ *successor*)
using *oneUN-iso-N-isomorphism* **by** *blast*
have *zUsi1EqsS*: (*zero* ⨿ *successor*) $\circ_c$ (*right-coproj* **1** $\mathbb{N}_c$) = *successor*
using *right-coproj-cfunc-coprod successor-type zero-type* **by** *auto*
then have *succ-mono*: *monomorphism*(*successor*)
by (*metis cfunc-coprod-type cfunc-type-def composition-of-monic-pair-is-monic i1-mono iso-imp-epi-and-monic oneUN-iso-N-isomorphism right-proj-type succes-*

sor-type zero-type)
obtain *u* **where** *u-type*: *u*: $\mathbb{N}_c \rightarrow \Omega$ **and** *u-def*: $u \circ_c$ *zero* = t $\wedge$ (f$\circ_c\beta_\Omega$) $\circ_c$ *u* = *u* $\circ_c$ *successor*
by (*typecheck-cfuncs*, *metis natural-number-object-property*)
have *s-not-surj*: $\neg$ *surjective successor*
proof (*rule ccontr*, *clarify*)
assume *BWOC* : *surjective successor*
obtain *n* **where** *n-type*: $n : \mathbf{1} \rightarrow \mathbb{N}_c$ **and** *snEqz*: *successor* $\circ_c$ *n* = *zero*
using *BWOC cfunc-type-def successor-type surjective-def zero-type* **by** *auto*
then show *False*
by (*metis zero-is-not-successor*)
qed
then show *injective successor* $\wedge$ $\neg$ *surjective successor*
using *monomorphism-imp-injective succ-mono* **by** *blast*
qed

lemma *succ-inject*:
assumes $n \in_c \mathbb{N}_c$ $m \in_c \mathbb{N}_c$
shows *successor* $\circ_c$ *n* = *successor* $\circ_c$ *m* $\Longrightarrow$ *n* = *m*
by (*metis Peano's-Axioms assms cfunc-type-def injective-def successor-type*)

theorem *nat-induction*:
assumes *p-type*[*type-rule*]: $p : \mathbb{N}_c \rightarrow \Omega$ **and** *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
assumes *base-case*: $p \circ_c$ *zero* = t
assumes *induction-case*: $\bigwedge n.\ n \in_c \mathbb{N}_c \Longrightarrow p \circ_c n =$ t $\Longrightarrow p \circ_c$ *successor* $\circ_c n$ = t
shows $p \circ_c n$ = t
proof −
obtain p' P **where**
p'*-type*[*type-rule*]: $p' : P \rightarrow \mathbb{N}_c$ **and**
p'*-equalizer*: $p \circ_c p' =$ (t $\circ_c \beta_{\mathbb{N}_c}$) $\circ_c p'$ **and**
p'*-uni-prop*: $\forall$ *h F*. ($h : F \rightarrow \mathbb{N}_c \wedge p \circ_c h =$ (t $\circ_c \beta_{\mathbb{N}_c}$) $\circ_c h$) $\longrightarrow$ ($\exists!$ *k*. $k : F \rightarrow P \wedge p' \circ_c k = h$)
using *equalizer-exists2* **by** (*typecheck-cfuncs*, *blast*)

from *base-case* **have** $p \circ_c$ *zero* = (t $\circ_c \beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
by (*etcs-assocr*, *etcs-subst terminal-func-comp-elem id-right-unit2*, −)
then obtain z' **where**
z'*-type*[*type-rule*]: $z' \in_c P$ **and**
z'*-def*: *zero* = $p' \circ_c z'$
using p'*-uni-prop* **by** (*typecheck-cfuncs*, *metis*)

have $p \circ_c$ *successor* $\circ_c p'$ = (t $\circ_c \beta_{\mathbb{N}_c}$) $\circ_c$ *successor* $\circ_c p'$
proof (*etcs-rule one-separator*)
fix *m*
assume *m-type*[*type-rule*]: $m \in_c P$

have $p \circ_c p' \circ_c m$ = t $\circ_c \beta_{\mathbb{N}_c} \circ_c p' \circ_c m$
by (*etcs-assocl*, *simp add*: p'*-equalizer*)

then have $p \circ_c p' \circ_c m = \mathrm{t}$
by ($-$, *etcs-subst-asm terminal-func-comp-elem id-right-unit2*, *simp*)
then have $p \circ_c$ *successor* $\circ_c p' \circ_c m = \mathrm{t}$
using *induction-case* **by** (*typecheck-cfuncs*, *simp*)
then show ($p \circ_c$ *successor* $\circ_c p'$) $\circ_c m$ = (($\mathrm{t} \circ_c \beta_{\mathbb{N}_c}$) $\circ_c$ *successor* $\circ_c p'$) $\circ_c m$
by (*etcs-assocr*, *etcs-subst terminal-func-comp-elem id-right-unit2*, $-$)
qed
then obtain s' **where**
s'-*type*[*type-rule*]: $s' : P \to P$ **and**
s'-*def*: $p' \circ_c s'$ = *successor* $\circ_c p'$
using p'-*uni-prop* **by** (*typecheck-cfuncs*, *metis*)

obtain u **where**
u-type[*type-rule*]: $u : \mathbb{N}_c \to P$ **and**
u-zero: $u \circ_c$ *zero* = z' **and**
u-succ: $u \circ_c$ *successor* = $s' \circ_c u$
using *natural-number-object-property2* **by** (*typecheck-cfuncs*, *metis* s'-*type*)

have p'-*u-is-id*: $p' \circ_c u$ = *id* $\mathbb{N}_c$
proof (*etcs-rule natural-number-object-func-unique*[**where** f=*successor*])
show ($p' \circ_c u$) $\circ_c$ *zero* = id_c $\mathbb{N}_c$ $\circ_c$ *zero*
by (*etcs-subst id-left-unit2*, *etcs-assocr*, *simp add*: *u-zero sym*[*OF* z'-*def*])
show ($p' \circ_c u$) $\circ_c$ *successor* = *successor* $\circ_c p' \circ_c u$
by (*etcs-assocr*, *subst u-succ*, *etcs-assocl*, *simp add*: s'-*def*)
show id_c $\mathbb{N}_c$ $\circ_c$ *successor* = *successor* $\circ_c$ id_c $\mathbb{N}_c$
by (*etcs-subst id-right-unit2 id-left-unit2*, *simp*)
qed

have $p \circ_c p' \circ_c u \circ_c n$ = ($\mathrm{t} \circ_c \beta_{\mathbb{N}_c}$) $\circ_c p' \circ_c u \circ_c n$
by (*typecheck-cfuncs*, *smt comp-associative2* p'-*equalizer*)
then show $p \circ_c n = \mathrm{t}$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 id-left-unit2 id-right-unit2* p'-*type* p'-*u-is-id terminal-func-comp-elem terminal-func-type u-type*)
qed

13.4 Function Iteration

definition *ITER-curried* :: *cset* $\Rightarrow$ *cfunc* **where**
ITER-curried U = (*THE* u . $u : \mathbb{N}_c \to (U^U)^{U^U}$ $\wedge$ $u \circ_c$ *zero* = (*metafunc* (*id* U) $\circ_c$ (*right-cart-proj* (U^U) **1**))$^\sharp$ $\wedge$
((*meta-comp* U U U) $\circ_c$ (*id* (U^U) $\times_f$ *eval-func* (U^U) (U^U)) $\circ_c$ (*associate-right* (U^U) (U^U) ((U^U)$^{U^U}$)) $\circ_c$ (*diagonal*(U^U)$\times_f$ *id* ((U^U)$^{U^U}$)))$^\sharp$ $\circ_c$ u = u $\circ_c$ *successor*)

lemma *ITER-curried-def2*:
ITER-curried $U : \mathbb{N}_c \to (U^U)^{U^U}$ $\wedge$ *ITER-curried* $U \circ_c$ *zero* = (*metafunc* (*id* U) $\circ_c$ (*right-cart-proj* (U^U) **1**))$^\sharp$ $\wedge$
((*meta-comp* U U U) $\circ_c$ (*id* (U^U) $\times_f$ *eval-func* (U^U) (U^U)) $\circ_c$ (*associate-right*

(U^U) (U^U) $((U^U)^{U^U})$) $\circ_c$ (*diagonal*$(U^U)\times_f$ *id* $((U^U)^{U^U})$))$^\sharp$ $\circ_c$ *ITER-curried* U = *ITER-curried* U $\circ_c$ *successor*
unfolding *ITER-curried-def*
by(*rule theI′*, *etcs-rule natural-number-object-property2*)

lemma *ITER-curried-type*[*type-rule*]:
ITER-curried U : $\mathbb{N}_c \rightarrow (U^U)^{U^U}$
by (*simp add*: *ITER-curried-def2*)

lemma *ITER-curried-zero*:
ITER-curried U $\circ_c$ *zero* = (*metafunc* (*id* U) $\circ_c$ (*right-cart-proj* (U^U) **1**))$^\sharp$
by (*simp add*: *ITER-curried-def2*)

lemma *ITER-curried-successor*:
ITER-curried U $\circ_c$ *successor* = (*meta-comp* U U U $\circ_c$ (*id* (U^U) $\times_f$ *eval-func* (U^U) (U^U)) $\circ_c$ (*associate-right* (U^U) (U^U) $((U^U)^{U^U})$) $\circ_c$ (*diagonal*$(U^U)\times_f$ *id* $((U^U)^{U^U})$))$^\sharp$ $\circ_c$ *ITER-curried* U
using *ITER-curried-def2* **by** *simp*

definition *ITER* :: *cset* ⇒ *cfunc* **where**
ITER U = (*ITER-curried* U)$^\flat$

lemma *ITER-type*[*type-rule*]:
ITER U : $((U^U) \times_c \mathbb{N}_c) \rightarrow (U^U)$
unfolding *ITER-def* **by** *typecheck-cfuncs*

lemma *ITER-zero*:
assumes *f-type*[*type-rule*]: $f : Z \rightarrow (U^U)$
shows *ITER* U $\circ_c$ ⟨f, *zero* $\circ_c$ β_Z⟩ = *metafunc* (*id* U) $\circ_c$ β_Z
proof(*etcs-rule one-separator*)
fix z
assume *z-type*[*type-rule*]: $z \in_c Z$
have (*ITER* U $\circ_c$ ⟨f,*zero* $\circ_c$ β_Z⟩) $\circ_c$ z = *ITER* U $\circ_c$ ⟨f,*zero* $\circ_c$ β_Z⟩ $\circ_c$ z
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *ITER* U $\circ_c$ ⟨f $\circ_c$ z,*zero*⟩
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2 id-right-unit2 terminal-func-comp-elem*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ (*id*$_c$ (U^U) $\times_f$ *ITER-curried* U) $\circ_c$ ⟨f $\circ_c$ z,*zero*⟩
using *assms ITER-def comp-associative2 inv-transpose-func-def3* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ ⟨f $\circ_c$ z,*ITER-curried* U $\circ_c$ *zero*⟩
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ ⟨f $\circ_c$ z,(*metafunc* (*id* U) $\circ_c$ (*right-cart-proj* (U^U) **1**))$^\sharp$⟩
using *assms* **by** (*simp add*: *ITER-curried-def2*)

also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ $\langle f \circ_c z$,((*left-cart-proj* (U) $\mathbf{1})^\sharp$ $\circ_c$ (*right-cart-proj* (U^U) $\mathbf{1}))^\sharp\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2 metafunc-def2*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ (id_c (U^U) $\times_f$ ((*left-cart-proj* (U) $\mathbf{1})^\sharp$ $\circ_c$ (*right-cart-proj* (U^U) $\mathbf{1}))^\sharp$) $\circ_c$ $\langle f \circ_c z, id_c\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
also have ... = (*left-cart-proj* (U) $\mathbf{1})^\sharp$ $\circ_c$ (*right-cart-proj* (U^U) $\mathbf{1}$) $\circ_c$ $\langle f \circ_c z, id_c\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*,*simp add*: *cfunc-type-def comp-associative transpose-func-def*)
also have ... = (*left-cart-proj* (U) $\mathbf{1})^\sharp$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *id-right-unit2 right-cart-proj-cfunc-prod*)
also have ... = (*metafunc* (id_c U))
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2 metafunc-def2*)
also have ... = (*metafunc* (id_c U) $\circ_c$ β_Z) $\circ_c$ z
using *assms* **by** (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative id-right-unit2 terminal-func-comp-elem*)
finally show (*ITER U* $\circ_c$ $\langle f$,*zero* $\circ_c$ $\beta_Z\rangle$) $\circ_c$ z = (*metafunc* (id_c U) $\circ_c$ β_Z) $\circ_c$ z**.**
qed

lemma *ITER-zero'*:
assumes $f \in_c (U^U)$
shows *ITER U* $\circ_c$ $\langle f$, *zero*$\rangle$ = *metafunc* (*id U*)
by (*typecheck-cfuncs*, *metis ITER-zero assms id-right-unit2 id-type one-unique-element terminal-func-type*)

lemma *ITER-succ*:
assumes *f-type*[*type-rule*]: $f : Z \to (U^U)$ **and** *n-type*[*type-rule*]: $n : Z \to \mathbb{N}_c$
shows *ITER U* $\circ_c$ $\langle f$, *successor* $\circ_c$ $n\rangle$ = f $\square$ (*ITER U* $\circ_c$ $\langle f, n\rangle$)
proof(*etcs-rule one-separator*)
fix *z*
assume *z-type*[*type-rule*]: $z \in_c Z$
have (*ITER U* $\circ_c$ $\langle f$,*successor* $\circ_c$ $n\rangle$) $\circ_c$ z = *ITER U* $\circ_c$ $\langle f$,*successor* $\circ_c$ $n\rangle$ $\circ_c$ z
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *ITER U* $\circ_c$ $\langle f \circ_c z$, *successor* $\circ_c$ (n $\circ_c$ z)$\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp comp-associative2*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ (id_c (U^U) $\times_f$ *ITER-curried U*) $\circ_c$ $\langle f$ $\circ_c$ z, *successor* $\circ_c$ (n $\circ_c$ z)$\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *ITER-def comp-associative2 inv-transpose-func-def3*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ $\langle f \circ_c z$, *ITER-curried U* $\circ_c$ (*successor* $\circ_c$ (n $\circ_c$ z))$\rangle$
using *assms cfunc-cross-prod-comp-cfunc-prod id-left-unit2* **by** (*typecheck-cfuncs*, *force*)
also have ... = (*eval-func* (U^U) (U^U)) $\circ_c$ $\langle f \circ_c z$, (*ITER-curried U* $\circ_c$ *successor*) $\circ_c$ (n $\circ_c$ z)$\rangle$
using *assms* **by**(*typecheck-cfuncs*, *metis comp-associative2*)

also have ... = $(eval\text{-}func\ (U^U)\ (U^U)) \circ_c \langle f \circ_c z, ((meta\text{-}comp\ U\ U\ U \circ_c (id\ (U^U) \times_f eval\text{-}func\ (U^U)\ (U^U)) \circ_c (associate\text{-}right\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (diagonal(U^U) \times_f id\ ((U^U)^{U^U})))^\sharp \circ_c ITER\text{-}curried\ U) \circ_c (n \circ_c z)\rangle$
using *assms ITER-curried-successor* **by** *presburger*
also have ... = $(eval\text{-}func\ (U^U)\ (U^U)) \circ_c (id\ (U^U) \times_f ((meta\text{-}comp\ U\ U\ U \circ_c (id\ (U^U) \times_f eval\text{-}func\ (U^U)\ (U^U)) \circ_c (associate\text{-}right\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (diagonal(U^U) \times_f id\ ((U^U)^{U^U})))^\sharp \circ_c ITER\text{-}curried\ U) \circ_c (n \circ_c z)) \circ_c \langle f \circ_c z, id\ \mathbf{1}\rangle$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
also have ... = $(eval\text{-}func\ (U^U)\ (U^U)) \circ_c (id\ (U^U) \times_f ((meta\text{-}comp\ U\ U\ U \circ_c (id\ (U^U) \times_f eval\text{-}func\ (U^U)\ (U^U)) \circ_c (associate\text{-}right\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (diagonal(U^U) \times_f id\ ((U^U)^{U^U})))^\sharp)) \circ_c \langle f \circ_c z, ITER\text{-}curried\ U \circ_c (n \circ_c z)\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod comp-associative2 id-right-unit2*)
also have ... = $(meta\text{-}comp\ U\ U\ U \circ_c (id\ (U^U) \times_f eval\text{-}func\ (U^U)\ (U^U)) \circ_c (associate\text{-}right\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (diagonal(U^U) \times_f id\ ((U^U)^{U^U}))) \circ_c \langle f \circ_c z, ITER\text{-}curried\ U \circ_c (n \circ_c z)\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative transpose-func-def*)
also have ... = $(meta\text{-}comp\ U\ U\ U \circ_c (id\ (U^U) \times_f eval\text{-}func\ (U^U)\ (U^U)) \circ_c (associate\text{-}right\ (U^U)\ (U^U)\ ((U^U)^{U^U}))) \circ_c \langle\langle f \circ_c z, f \circ_c z\rangle, ITER\text{-}curried\ U \circ_c (n \circ_c z)\rangle$
using *assms* **by** (*etcs-assocr*, *typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod diag-on-elements id-left-unit2*)
also have ... = $meta\text{-}comp\ U\ U\ U \circ_c (id\ (U^U) \times_f eval\text{-}func\ (U^U)\ (U^U)) \circ_c \langle f \circ_c z, \langle f \circ_c z, ITER\text{-}curried\ U \circ_c (n \circ_c z)\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *associate-right-ap comp-associative2*)
also have ... = $meta\text{-}comp\ U\ U\ U \circ_c \langle f \circ_c z, eval\text{-}func\ (U^U)\ (U^U) \circ_c \langle f \circ_c z, ITER\text{-}curried\ U \circ_c (n \circ_c z)\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = $meta\text{-}comp\ U\ U\ U \circ_c \langle f \circ_c z, eval\text{-}func\ (U^U)\ (U^U) \circ_c (id(U^U) \times_f ITER\text{-}curried\ U) \circ_c \langle f \circ_c z, n \circ_c z\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = $meta\text{-}comp\ U\ U\ U \circ_c \langle f \circ_c z, ITER\ U \circ_c \langle f \circ_c z, n \circ_c z\rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *ITER-def comp-associative2 inv-transpose-func-def3*)
also have ... = $meta\text{-}comp\ U\ U\ U \circ_c \langle f, ITER\ U \circ_c \langle f, n\rangle\rangle \circ_c z$
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2*)
also have ... = $(meta\text{-}comp\ U\ U\ U \circ_c \langle f, ITER\ U \circ_c \langle f, n\rangle\rangle) \circ_c z$
using *assms* **by** (*typecheck-cfuncs*, *meson comp-associative2*)
also have ... = $(f \square (ITER\ U \circ_c \langle f,n\rangle)) \circ_c z$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *meta-comp2-def5*)
finally show $(ITER\ U \circ_c \langle f, successor \circ_c n\rangle) \circ_c z = (f \square ITER\ U \circ_c \langle f,n\rangle) \circ_c$

z.
qed

lemma *ITER-one*:
assumes $f \in_c (U^U)$
shows $ITER\ U \circ_c \langle f,\ successor \circ_c zero\rangle = f \ \Box\ (metafunc\ (id\ U))$
using *ITER-succ ITER-zero′ assms zero-type* **by** *presburger*

definition *iter-comp* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (-$^{\circ}$-[55,0]55) **where**
iter-comp g n ≡ *cnufatem* (*ITER* (*domain g*) $\circ_c$ ⟨*metafunc g*,*n*⟩)

lemma *iter-comp-def2*:
$g^{\circ n}$ ≡ *cnufatem*(*ITER* (*domain g*) $\circ_c$ ⟨*metafunc g*,*n*⟩)
by (*simp add*: *iter-comp-def*)

lemma *iter-comp-type*[*type-rule*]:
assumes $g : X \to X$
assumes $n \in_c \mathbb{N}_c$
shows $g^{\circ n}: X \to X$
unfolding *iter-comp-def2*
by (*smt* (*verit*, *ccfv-SIG*) *ITER-type assms cfunc-type-def cnufatem-type comp-type metafunc-type right-param-on-el right-param-type*)

lemma *iter-comp-def3*:
assumes $g : X \to X$
assumes $n \in_c \mathbb{N}_c$
shows $g^{\circ n}$ = *cnufatem* (*ITER X* $\circ_c$ ⟨*metafunc g*,*n*⟩)
using *assms cfunc-type-def iter-comp-def2* **by** *auto*

lemma *zero-iters*:
assumes *g-type*[*type-rule*]: $g : X \to X$
shows $g^{\circ zero} = id_c\ X$
proof(*etcs-rule one-separator*)
fix *x*
assume *x-type*[*type-rule*]: $x \in_c X$
have $(g^{\circ zero}) \circ_c x$ = (*cnufatem* (*ITER X* $\circ_c$ ⟨*metafunc g*,*zero*⟩)) $\circ_c$ *x*
using *assms iter-comp-def3* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *cnufatem* (*metafunc* (*id X*)) $\circ_c$ *x*
by (*simp add*: *ITER-zero′ assms metafunc-type*)
also have ... = $id_c\ X \circ_c x$
by (*metis cnufatem-metafunc id-type*)
also have ... = *x*
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)
ultimately show $(g^{\circ zero}) \circ_c x = id_c\ X \circ_c x$
by *simp*
qed

lemma *succ-iters*:
assumes $g : X \to X$

assumes $n \in_c \mathbb{N}_c$
shows $g^{\circ(successor \circ_c n)} = g \circ_c (g^{\circ n})$
proof −
have $g^{\circ successor \circ_c n} = cnufatem(ITER\ X \circ_c \langle metafunc\ g, successor \circ_c n \rangle)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *iter-comp-def3*)
also have ... $= cnufatem(metafunc\ g \square ITER\ X \circ_c \langle metafunc\ g, n \rangle)$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *ITER-succ*)
also have ... $= cnufatem(metafunc\ g \square metafunc\ (g^{\circ n}))$
using *assms* **by** (*typecheck-cfuncs*, *metis iter-comp-def3 metafunc-cnufatem*)
also have ... $= g \circ_c (g^{\circ n})$
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-as-metacomp*)
finally show *?thesis*.
qed

corollary *one-iter*:
assumes $g : X \to X$
shows $g^{\circ(successor \circ_c zero)} = g$
using *assms id-right-unit2 succ-iters zero-iters zero-type* **by** *force*

lemma *eval-lemma-for-ITER*:
assumes $f : X \to X$
assumes $x \in_c X$
assumes $m \in_c \mathbb{N}_c$
shows $(f^{\circ m}) \circ_c x = eval\text{-}func\ X\ X \circ_c \langle x, ITER\ X \circ_c \langle metafunc\ f, m \rangle\rangle$
using *assms* **by** (*typecheck-cfuncs*, *metis eval-lemma iter-comp-def3 metafunc-cnufatem*)

lemma *n-accessible-by-succ-iter-aux*:
$eval\text{-}func\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \langle zero \circ_c \beta_{\mathbb{N}_c}, ITER\ \mathbb{N}_c \circ_c \langle (metafunc\ successor) \circ_c \beta_{\mathbb{N}_c}, id\ \mathbb{N}_c \rangle\rangle = id\ \mathbb{N}_c$
proof(*rule natural-number-object-func-unique*[**where** X=$\mathbb{N}_c$, **where** f = *successor*])
show $eval\text{-}func\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \langle zero \circ_c \beta_{\mathbb{N}_c}, ITER\ \mathbb{N}_c \circ_c \langle metafunc\ successor \circ_c \beta_{\mathbb{N}_c}, id_c\ \mathbb{N}_c \rangle\rangle : \mathbb{N}_c \to \mathbb{N}_c$
by *typecheck-cfuncs*
show $id_c\ \mathbb{N}_c : \mathbb{N}_c \to \mathbb{N}_c$
by *typecheck-cfuncs*
show $successor : \mathbb{N}_c \to \mathbb{N}_c$
by *typecheck-cfuncs*
next
have $(eval\text{-}func\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \langle zero \circ_c \beta_{\mathbb{N}_c}, ITER\ \mathbb{N}_c \circ_c \langle metafunc\ successor \circ_c \beta_{\mathbb{N}_c}, id_c\ \mathbb{N}_c \rangle\rangle) \circ_c zero =$
$eval\text{-}func\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \langle zero \circ_c \beta_{\mathbb{N}_c} \circ_c zero, ITER\ \mathbb{N}_c \circ_c \langle metafunc\ successor \circ_c \beta_{\mathbb{N}_c} \circ_c zero, id_c\ \mathbb{N}_c \circ_c zero \rangle\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2*)
also have ... $= eval\text{-}func\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \langle zero, ITER\ \mathbb{N}_c \circ_c \langle metafunc\ successor, zero \rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2 terminal-func-comp-elem*)
also have ... $= eval\text{-}func\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \langle zero, metafunc\ (id\ \mathbb{N}_c) \rangle$
by (*typecheck-cfuncs*, *simp add*: *ITER-zero′*)
also have ... $= id_c\ \mathbb{N}_c \circ_c zero$

using *eval-lemma* **by** (*typecheck-cfuncs*, *blast*)
finally show (*eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩) $\circ_c$ *zero* = *id*$_c$ $\mathbb{N}_c$ $\circ_c$ *zero*.
show (*eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩) $\circ_c$ *successor* =
successor $\circ_c$ *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩
proof(*etcs-rule one-separator*)
fix *m*
assume *m-type*[*type-rule*]: *m* $\in_c$ $\mathbb{N}_c$
have (*successor* $\circ_c$ *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩) $\circ_c$ *m* =
successor $\circ_c$ *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *m*,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *m*,*id*$_c$ $\mathbb{N}_c$ $\circ_c$ *m*⟩⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2*)
also have ... = *successor* $\circ_c$ *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* ,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* ,*m*⟩⟩
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2 terminal-func-comp-elem*)
also have ... = *successor* $\circ_c$ ($successor^{\circ m}$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *eval-lemma-for-ITER*)
also have ... = ($successor^{\circ successor \circ_c m}$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 succ-iters*)
also have ... = *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* ,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* ,*successor* $\circ_c$ *m*⟩⟩
by (*typecheck-cfuncs*, *simp add*: *eval-lemma-for-ITER*)
also have ... = *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ (*successor* $\circ_c$ *m*),*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$$\circ_c$ (*successor* $\circ_c$ *m*),*id*$_c$ $\mathbb{N}_c$ $\circ_c$ (*successor* $\circ_c$ *m*)⟩⟩
by (*typecheck-cfuncs*,*simp add*: *id-left-unit2 id-right-unit2 terminal-func-comp-elem*)
also have ... = ((*eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩) $\circ_c$ *successor*) $\circ_c$ *m*
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2*)
ultimately show ((*eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩) $\circ_c$ *successor*) $\circ_c$ *m* =
(*successor* $\circ_c$ *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$,*ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$,*id*$_c$ $\mathbb{N}_c$⟩⟩) $\circ_c$ *m*
by *simp*
qed
show *id*$_c$ $\mathbb{N}_c$ $\circ_c$ *successor* = *successor* $\circ_c$ *id*$_c$ $\mathbb{N}_c$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2*)
qed

lemma *n-accessible-by-succ-iter*:
assumes *n* $\in_c$ $\mathbb{N}_c$
shows ($successor^{\circ n}$) $\circ_c$ *zero* = *n*
proof −
have *n* = *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$, *ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$, *id* $\mathbb{N}_c$⟩⟩ $\circ_c$ *n*
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *comp-associative2 id-left-unit2 n-accessible-by-succ-iter-aux*)

also have ... = *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *n* , *ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor* $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *n*, *id* $\mathbb{N}_c$ $\circ_c$ *n*⟩⟩
using *assms* **by** (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2*)
also have ... = *eval-func* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ ⟨*zero*, *ITER* $\mathbb{N}_c$ $\circ_c$ ⟨*metafunc successor*, *n*⟩⟩
using *assms* **by** (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2 terminal-func-comp-elem*)
also have ... = (*successor*$^{\circ n}$) $\circ_c$ *zero*
using *assms* **by** (*typecheck-cfuncs*, *metis eval-lemma iter-comp-def3 metafunc-cnufatem*)
ultimately show *?thesis*
by *simp*
qed

13.5 Relation of Nat to Other Sets

lemma *oneUN-iso-N*:
$\mathbf{1} \coprod \mathbb{N}_c \cong \mathbb{N}_c$
using *cfunc-coprod-type is-isomorphic-def oneUN-iso-N-isomorphism successor-type zero-type* **by** *blast*

lemma *NUone-iso-N*:
$\mathbb{N}_c \coprod \mathbf{1} \cong \mathbb{N}_c$
using *coproduct-commutes isomorphic-is-transitive oneUN-iso-N* **by** *blast*

end

14 Predicate Logic Functions

theory *Pred-Logic*
imports *Coproduct*
begin

14.1 NOT

definition *NOT* :: *cfunc* **where**
NOT = (*THE* χ. *is-pullback* **1** **1** Ω Ω ($\beta_{\mathbf{1}}$) t f χ)

lemma *NOT-is-pullback*:
is-pullback **1** **1** Ω Ω ($\beta_{\mathbf{1}}$) t f *NOT*
unfolding *NOT-def*
using *characteristic-function-exists false-func-type element-monomorphism*
by (*subst the1I2*, *auto*)

lemma *NOT-type*[*type-rule*]:
NOT : Ω → Ω
using *NOT-is-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *NOT-false-is-true*:
NOT $\circ_c$ f = t

using *NOT-is-pullback* **unfolding** *is-pullback-def*
by (*metis cfunc-type-def id-right-unit id-type one-unique-element*)

lemma *NOT-true-is-false*:
$NOT \circ_c \mathrm{t} = \mathrm{f}$
proof(*rule ccontr*)
assume $NOT \circ_c \mathrm{t} \neq \mathrm{f}$
then have $NOT \circ_c \mathrm{t} = \mathrm{t}$
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have $\mathrm{t} \circ_c id_c\ \mathbf{1} = NOT \circ_c \mathrm{t}$
using *id-right-unit2 true-func-type* **by** *auto*
then obtain j **where** *j-type*: $j \in_c \mathbf{1}$ **and** *j-id*: $\beta_{\mathbf{1}} \circ_c j = id_c\ \mathbf{1}$ **and** *f-j-eq-t*: $\mathrm{f} \circ_c j = \mathrm{t}$
using *NOT-is-pullback* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *blast*)
then have $j = id_c\ \mathbf{1}$
using *id-type one-unique-element* **by** *blast*
then have $\mathrm{f} = \mathrm{t}$
using *f-j-eq-t false-func-type id-right-unit2* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *NOT-is-true-implies-false*:
assumes $p \in_c \Omega$
shows $NOT \circ_c p = \mathrm{t} \Longrightarrow p = \mathrm{f}$
using *NOT-true-is-false assms true-false-only-truth-values* **by** *fastforce*

lemma *NOT-is-false-implies-true*:
assumes $p \in_c \Omega$
shows $NOT \circ_c p = \mathrm{f} \Longrightarrow p = \mathrm{t}$
using *NOT-false-is-true assms true-false-only-truth-values* **by** *fastforce*

lemma *double-negation*:
$NOT \circ_c NOT = id\ \Omega$
by (*typecheck-cfuncs*, *smt* (*verit*, *del-insts*)
NOT-false-is-true NOT-true-is-false cfunc-type-def comp-associative id-left-unit2 one-separator
true-false-only-truth-values)

14.2 AND

definition *AND* :: *cfunc* **where**
$AND = (THE\ \chi.\ is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \langle \mathrm{t},\mathrm{t}\rangle\ \chi)$

lemma *AND-is-pullback*:
$is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \langle \mathrm{t},\mathrm{t}\rangle\ AND$
unfolding *AND-def*
using *element-monomorphism characteristic-function-exists*
by (*typecheck-cfuncs*, *subst the1I2*, *auto*)

lemma *AND-type*[*type-rule*]:
  $AND : \Omega \times_c \Omega \to \Omega$
  using *AND-is-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *AND-true-true-is-true*:
  $AND \circ_c \langle \mathrm{t},\mathrm{t} \rangle = \mathrm{t}$
  using *AND-is-pullback* **unfolding** *is-pullback-def*
  by (*metis cfunc-type-def id-right-unit id-type one-unique-element*)

lemma *AND-false-left-is-false*:
  assumes $p \in_c \Omega$
  shows $AND \circ_c \langle \mathrm{f},p \rangle = \mathrm{f}$
proof (*rule ccontr*)
  assume $AND \circ_c \langle \mathrm{f},p \rangle \neq \mathrm{f}$
  then have $AND \circ_c \langle \mathrm{f},p \rangle = \mathrm{t}$
    using *assms true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
  then have $\mathrm{t} \circ_c id\ \mathbf{1} = AND \circ_c \langle \mathrm{f},p \rangle$
    using *assms* **by** (*typecheck-cfuncs*, *simp add*: *id-right-unit2*)
  then obtain j **where** *j-type*: $j \in_c \mathbf{1}$ **and** *j-id*: $\beta_{\mathbf{1}} \circ_c j = id_c\ \mathbf{1}$ **and** *tt-j-eq-fp*: $\langle \mathrm{t},\mathrm{t} \rangle \circ_c j = \langle \mathrm{f},p \rangle$
    using *AND-is-pullback assms* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *blast*)
  then have $j = id_c\ \mathbf{1}$
    using *id-type one-unique-element* **by** *auto*
  then have $\langle \mathrm{t},\mathrm{t} \rangle = \langle \mathrm{f},p \rangle$
    by (*typecheck-cfuncs*, *metis tt-j-eq-fp id-right-unit2*)
  then have $\mathrm{t} = \mathrm{f}$
    using *assms cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*)
  then show *False*
    using *true-false-distinct* **by** *auto*
qed

lemma *AND-false-right-is-false*:
  assumes $p \in_c \Omega$
  shows $AND \circ_c \langle p,\mathrm{f} \rangle = \mathrm{f}$
proof(*rule ccontr*)
  assume $AND \circ_c \langle p,\mathrm{f} \rangle \neq \mathrm{f}$
  then have $AND \circ_c \langle p,\mathrm{f} \rangle = \mathrm{t}$
    using *assms true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
  then have $\mathrm{t} \circ_c id\ \mathbf{1} = AND \circ_c \langle p,\mathrm{f} \rangle$
    using *assms* **by** (*typecheck-cfuncs*, *simp add*: *id-right-unit2*)
  then obtain j **where** *j-type*: $j \in_c \mathbf{1}$ **and** *j-id*: $\beta_{\mathbf{1}} \circ_c j = id_c\ \mathbf{1}$ **and** *tt-j-eq-fp*: $\langle \mathrm{t},\mathrm{t} \rangle \circ_c j = \langle p,\mathrm{f} \rangle$
    using *AND-is-pullback assms* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *blast*)
  then have $j = id_c\ \mathbf{1}$
    using *id-type one-unique-element* **by** *auto*
  then have $\langle \mathrm{t},\mathrm{t} \rangle = \langle p,\mathrm{f} \rangle$

by (*typecheck-cfuncs*, *metis tt-j-eq-fp id-right-unit2*)
then have t = f
using *assms cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto*)
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *AND-commutative*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $AND \circ_c \langle p,q \rangle = AND \circ_c \langle q,p \rangle$
by (*metis AND-false-left-is-false AND-false-right-is-false assms true-false-only-truth-values*)

lemma *AND-idempotent*:
assumes $p \in_c \Omega$
shows $AND \circ_c \langle p,p \rangle = p$
using *AND-false-right-is-false AND-true-true-is-true assms true-false-only-truth-values*
by *blast*

lemma *AND-associative*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $r \in_c \Omega$
shows $AND \circ_c \langle AND \circ_c \langle p,q \rangle, r \rangle = AND \circ_c \langle p, AND \circ_c \langle q,r \rangle \rangle$
by (*metis AND-commutative AND-false-left-is-false AND-true-true-is-true assms true-false-only-truth-values*)

lemma *AND-complementary*:
assumes $p \in_c \Omega$
shows $AND \circ_c \langle p, NOT \circ_c p \rangle =$ f
by (*metis AND-false-left-is-false AND-false-right-is-false NOT-false-is-true NOT-true-is-false assms true-false-only-truth-values true-func-type*)

14.3 NOR

definition *NOR* :: *cfunc* **where**
$NOR = (THE\ \chi.\ is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \langle \mathrm{f}, \mathrm{f} \rangle\ \chi)$

lemma *NOR-is-pullback*:
$is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \langle \mathrm{f}, \mathrm{f} \rangle\ NOR$
unfolding *NOR-def*
using *characteristic-function-exists element-monomorphism*
by (*typecheck-cfuncs*, *simp add*: *the1I2*)

lemma *NOR-type*[*type-rule*]:
$NOR : \Omega \times_c \Omega \to \Omega$
using *NOR-is-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *NOR-false-false-is-true*:

NOR $\circ_c$ ⟨f,f⟩ = t
using *NOR-is-pullback* **unfolding** *is-pullback-def*
by (*auto*, *metis cfunc-type-def id-right-unit id-type one-unique-element*)

lemma *NOR-left-true-is-false*:
assumes $p \in_c \Omega$
shows *NOR* $\circ_c$ ⟨t,p⟩ = f
proof (*rule ccontr*)
assume *NOR* $\circ_c$ ⟨t,p⟩ ≠ f
then have *NOR* $\circ_c$ ⟨t,p⟩ = t
using *assms true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have *NOR* $\circ_c$ ⟨t,p⟩ = t $\circ_c$ *id* **1**
using *id-right-unit2 true-func-type* **by** *auto*
then obtain j **where** *j-type*: $j \in_c$ **1** **and** *j-id*: $\beta_{\mathbf{1}}$ $\circ_c$ j = *id* **1** **and** *ff-j-eq-tp*: ⟨f,f⟩ $\circ_c$ j = ⟨t,p⟩
using *NOR-is-pullback assms* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *metis*)
then have j = *id* **1**
using *id-type one-unique-element* **by** *blast*
then have ⟨f,f⟩ = ⟨t,p⟩
using *cfunc-prod-comp false-func-type ff-j-eq-tp id-right-unit2 j-type* **by** *auto*
then have f = t
using *assms cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *NOR-right-true-is-false*:
assumes $p \in_c \Omega$
shows *NOR* $\circ_c$ ⟨p,t⟩ = f
proof (*rule ccontr*)
assume *NOR* $\circ_c$ ⟨p,t⟩ ≠ f
then have *NOR* $\circ_c$ ⟨p,t⟩ = t
using *assms true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have *NOR* $\circ_c$ ⟨p,t⟩ = t $\circ_c$ *id* **1**
using *id-right-unit2 true-func-type* **by** *auto*
then obtain j **where** *j-type*: $j \in_c$ **1** **and** *j-id*: $\beta_{\mathbf{1}}$ $\circ_c$ j = *id* **1** **and** *ff-j-eq-tp*: ⟨f,f⟩ $\circ_c$ j = ⟨p,t⟩
using *NOR-is-pullback assms* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *metis*)
then have j = *id* **1**
using *id-type one-unique-element* **by** *blast*
then have ⟨f,f⟩ = ⟨p,t⟩
using *cfunc-prod-comp false-func-type ff-j-eq-tp id-right-unit2 j-type* **by** *auto*
then have f = t
using *assms cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *NOR-true-implies-both-false*:
assumes *X-nonempty*: *nonempty* X **and** *Y-nonempty*: *nonempty* Y
assumes *P-Q-types*[*type-rule*]: $P : X \to \Omega$ $Q : Y \to \Omega$
assumes *NOR-true*: $NOR \circ_c (P \times_f Q) = \mathrm{t} \circ_c \beta_{X \times_c Y}$
shows $P = \mathrm{f} \circ_c \beta_X \land Q = \mathrm{f} \circ_c \beta_Y$
proof −
obtain z **where** *z-type*[*type-rule*]: $z : X \times_c Y \to \mathbf{1}$ **and** $P \times_f Q = \langle \mathrm{f},\mathrm{f} \rangle \circ_c z$
using *NOR-is-pullback NOR-true* **unfolding** *is-pullback-def*
by (*metis P-Q-types cfunc-cross-prod-type terminal-func-type*)
then have $P \times_f Q = \langle \mathrm{f},\mathrm{f} \rangle \circ_c \beta_{X \times_c Y}$
using *terminal-func-unique* **by** *auto*
then have $P \times_f Q = \langle \mathrm{f} \circ_c \beta_{X \times_c Y}, \mathrm{f} \circ_c \beta_{X \times_c Y} \rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp*)
then have $P \times_f Q = \langle \mathrm{f} \circ_c \beta_X \circ_c$ *left-cart-proj* X Y, $\mathrm{f} \circ_c \beta_Y \circ_c$ *right-cart-proj* X $Y\rangle$
by (*typecheck-cfuncs-prems*, *metis left-cart-proj-type right-cart-proj-type terminal-func-comp*)
then have $\langle P \circ_c$ *left-cart-proj* X Y, $Q \circ_c$ *right-cart-proj* X $Y\rangle$
$= \langle \mathrm{f} \circ_c \beta_X \circ_c$ *left-cart-proj* X Y, $\mathrm{f} \circ_c \beta_Y \circ_c$ *right-cart-proj* X $Y\rangle$
by (*typecheck-cfuncs*, *unfold cfunc-cross-prod-def2*, *auto*)
then have $P \circ_c$ *left-cart-proj* X $Y = (\mathrm{f} \circ_c \beta_X) \circ_c$ *left-cart-proj* X Y
$\land$ $Q \circ_c$ *right-cart-proj* X $Y = (\mathrm{f} \circ_c \beta_Y) \circ_c$ *right-cart-proj* X Y
using *cart-prod-eq2* **by** (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
then have *eqs*: $P = \mathrm{f} \circ_c \beta_X \land Q = \mathrm{f} \circ_c \beta_Y$
using *assms epimorphism-def3 nonempty-left-imp-right-proj-epimorphism nonempty-right-imp-left-proj-epim*
by (*typecheck-cfuncs-prems*, *blast*)
then have $P \neq \mathrm{t} \circ_c \beta_X \land Q \neq \mathrm{t} \circ_c \beta_Y$
proof *safe*
show $\mathrm{f} \circ_c \beta_X = \mathrm{t} \circ_c \beta_X \Longrightarrow$ *False*
by (*typecheck-cfuncs-prems*, *smt X-nonempty comp-associative2 nonempty-def one-separator-contrapos terminal-func-comp terminal-func-unique true-false-distinct*)
show $\mathrm{f} \circ_c \beta_Y = \mathrm{t} \circ_c \beta_Y \Longrightarrow$ *False*
by (*typecheck-cfuncs-prems*, *smt Y-nonempty comp-associative2 nonempty-def one-separator-contrapos terminal-func-comp terminal-func-unique true-false-distinct*)
qed
then show *?thesis*
using *eqs* **by** *linarith*
qed

lemma *NOR-true-implies-neither-true*:
assumes *X-nonempty*: *nonempty* X **and** *Y-nonempty*: *nonempty* Y
assumes *P-Q-types*[*type-rule*]: $P : X \to \Omega$ $Q : Y \to \Omega$
assumes *NOR-true*: $NOR \circ_c (P \times_f Q) = \mathrm{t} \circ_c \beta_{X \times_c Y}$
shows $\neg (P = \mathrm{t} \circ_c \beta_X \lor Q = \mathrm{t} \circ_c \beta_Y)$
by (*smt* (*verit*, *ccfv-SIG*) *NOR-true NOT-false-is-true NOT-true-is-false NOT-type X-nonempty Y-nonempty assms*(*3*,*4*) *comp-associative2 comp-type nonempty-def terminal-func-type true-false-distinct true-false-only-truth-values NOR-true-implies-both-false*)

14.4 OR

definition *OR* :: *cfunc* **where**
OR = (*THE* χ. *is-pullback* ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$) $\mathbf{1}$ ($\Omega\times_c\Omega$) Ω ($\beta_{\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})}$) t (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) χ)

lemma *pre-OR-type*[*type-rule*]:
⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩) : $\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \rightarrow \Omega \times_c \Omega$
by *typecheck-cfuncs*

lemma *set-three*:
{x. $x \in_c$ ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$)} = {
(*left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) ,
(*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$),
right-coproj $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$(*right-coproj* $\mathbf{1}$ $\mathbf{1}$)}
by(*typecheck-cfuncs*, *safe*, *typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 coprojs-jointly-surj one-unique-element*)

lemma *set-three-card*:
card {x. $x \in_c$ ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$)} = *3*
proof −
have *f1*: *left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\neq$ *right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$
by (*typecheck-cfuncs*, *metis cfunc-type-def coproducts-disjoint id-right-unit id-type*)
have *f2*: *left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\neq$ *right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$ *right-coproj* $\mathbf{1}$ $\mathbf{1}$
by (*typecheck-cfuncs*, *metis cfunc-type-def coproducts-disjoint id-right-unit id-type*)
have *f3*: *right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$ $\neq$ *right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$ *right-coproj* $\mathbf{1}$ $\mathbf{1}$
by (*typecheck-cfuncs*, *metis cfunc-type-def coproducts-disjoint monomorphism-def one-unique-element right-coproj-are-monomorphisms*)
show *?thesis*
by (*simp add*: *f1 f2 f3 set-three*)
qed

lemma *pre-OR-injective*:
injective(⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩))
unfolding *injective-def*
proof(*clarify*)
fix *x y*
assume $x \in_c$ *domain* (⟨t,t⟩ II ⟨t,f⟩ II ⟨f,t⟩)
then have *x-type*: $x \in_c$ ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$)
using *cfunc-type-def pre-OR-type* **by** *force*
then have *x-form*: ($\exists$ w. ($w \in_c \mathbf{1} \wedge x$ = (*left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ w))
$\vee$ ($\exists$ w. ($w \in_c$ ($\mathbf{1}\coprod\mathbf{1}$) $\wedge$ x = (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ w))
using *coprojs-jointly-surj* **by** *auto*

assume $y \in_c$ *domain* (⟨t,t⟩ II ⟨t,f⟩ II ⟨f,t⟩)
then have *y-type*: $y \in_c$ ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$)
using *cfunc-type-def pre-OR-type* **by** *force*
then have *y-form*: ($\exists$ w. ($w \in_c \mathbf{1} \wedge y$ = (*left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ w))
$\vee$ ($\exists$ w. ($w \in_c$ ($\mathbf{1}\coprod\mathbf{1}$) $\wedge$ y = (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ w))

using *coprojs-jointly-surj* **by** *auto*

assume *mx-eqs-my*: $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ x = $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ y

have *f1*: $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ *left-coproj* **1** (**1** ∐ **1**) = $\langle$t,t$\rangle$
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
have *f2*: $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**) = $\langle$t,f$\rangle$
proof−
have $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**) =
($\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ *right-coproj* **1** (**1**∐**1**))$\circ_c$ *left-coproj* **1 1**
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ *left-coproj* **1 1**
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *smt*)
also have ... = $\langle$t,f$\rangle$
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
finally show *?thesis*.
qed
have *f3*: $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *right-coproj* **1 1**) = $\langle$f,t$\rangle$
proof−
have $\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *right-coproj* **1 1**) =
($\langle$t,t$\rangle$ ⨿ $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ *right-coproj* **1** (**1**∐**1**))$\circ_c$ *right-coproj* **1 1**
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = $\langle$t,f$\rangle$ ⨿ $\langle$f,t$\rangle$ $\circ_c$ *right-coproj* **1 1**
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *smt*)
also have ... = $\langle$f,t$\rangle$
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
finally show *?thesis*.
qed
show $x = y$
proof(*cases* x = *left-coproj* **1** (**1** ∐ **1**))
assume *case1*: x = *left-coproj* **1** (**1** ∐ **1**)
then show $x = y$
by (*typecheck-cfuncs*, *smt* (*z3*) *mx-eqs-my element-pair-eq f1 f2 f3 false-func-type maps-into-1u1 terminal-func-unique true-false-distinct true-func-type x-form y-form*)
next
assume *not-case1*: $x \neq$ *left-coproj* **1** (**1** ∐ **1**)
then have *case2-or-3*: x = (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**)∨
x = *right-coproj* **1** (**1**∐**1**) $\circ_c$(*right-coproj* **1 1**)
by (*metis id-right-unit2 id-type left-proj-type maps-into-1u1 terminal-func-unique x-form*)
show $x = y$
proof(*cases* x = (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**))
assume *case2*: x = *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1 1**
then show $x = y$
by (*typecheck-cfuncs*, *smt* (*z3*) *cart-prod-eq2 case2 f1 f2 f3 false-func-type id-right-unit2 left-proj-type maps-into-1u1 mx-eqs-my terminal-func-comp terminal-func-comp-elem terminal-func-unique true-false-distinct true-func-type y-form*)

next

assume *not-case2*: $x \neq$ *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1 1**
then have *case3*: $x =$ *right-coproj* **1** (**1**∐**1**) $\circ_c$(*right-coproj* **1 1**)
using *case2-or-3* **by** *blast*
then show $x = y$
by (*smt* (*verit*, *best*) *f1 f2 f3 NOR-false-false-is-true NOR-is-pullback case3 cfunc-prod-comp comp-associative2 element-pair-eq false-func-type is-pullback-def left-proj-type maps-into-1u1 mx-eqs-my pre-OR-type terminal-func-unique true-false-distinct true-func-type y-form*)
qed
qed
qed

lemma *OR-is-pullback*:
is-pullback (**1**∐(**1**∐**1**)) **1** ($\Omega \times_c \Omega$) Ω ($\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))}$) t (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) *OR*
unfolding *OR-def*
using *element-monomorphism characteristic-function-exists*
by (*typecheck-cfuncs*, *simp add*: *the1I2 injective-imp-monomorphism pre-OR-injective*)

lemma *OR-type*[*type-rule*]:
OR : $\Omega \times_c \Omega \to \Omega$
unfolding *OR-def*
by (*metis OR-def OR-is-pullback is-pullback-def*)

lemma *OR-true-left-is-true*:
assumes $p \in_c \Omega$
shows *OR* $\circ_c$ ⟨t,p⟩ = t
proof −
have ∃ j. $j \in_c$ **1**∐(**1**∐**1**) ∧ (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ j = ⟨t,p⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *assms comp-associative2 comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type true-false-only-truth-values*)
then show *?thesis*
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *NOT-false-is-true NOT-is-pullback OR-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *OR-true-right-is-true*:
assumes $p \in_c \Omega$
shows *OR* $\circ_c$ ⟨p,t⟩ = t
proof −
have ∃ j. $j \in_c$ **1**∐(**1**∐**1**) ∧ (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ j = ⟨p,t⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *assms comp-associative2 comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type true-false-only-truth-values*)
then show *?thesis*
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *NOT-false-is-true NOT-is-pullback OR-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *OR-false-false-is-false*:
  OR $\circ_c$ ⟨f,f⟩ = f
proof(*rule ccontr*)
  assume *OR* $\circ_c$ ⟨f,f⟩ ≠ f
  then have *OR* $\circ_c$ ⟨f,f⟩ = t
    using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
  then obtain *j* **where** *j-type*[*type-rule*]: *j* $\in_c$ **1**∐(**1**∐**1**) **and** *j-def*: (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ *j* = ⟨f,f⟩
    using *OR-is-pullback* **unfolding** *is-pullback-def*
    by (*typecheck-cfuncs*, *metis id-right-unit2 id-type*)
  have *trichotomy*: (⟨t, t⟩ = ⟨f,f⟩) ∨ ((⟨t, f⟩ = ⟨f,f⟩) ∨ (⟨f, t⟩ = ⟨f,f⟩))
  proof(*cases j* = *left-coproj* **1** (**1** ∐ **1**))
    assume *case1*: *j* = *left-coproj* **1** (**1** ∐ **1**)
    then show *?thesis*
     using *case1 cfunc-coprod-type j-def left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *force*)
  next
    assume *not-case1*: *j* ≠ *left-coproj* **1** (**1** ∐ **1**)
    then have *case2-or-3*: *j* = *right-coproj* **1** (**1**∐**1**) $\circ_c$ *left-coproj* **1 1** ∨
                       j = *right-coproj* **1** (**1**∐**1**) $\circ_c$ *right-coproj* **1 1**
      using *not-case1 set-three* **by** (*typecheck-cfuncs*, *auto*)
    show *?thesis*
    proof(*cases j* = (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**))
      assume *case2*: *j* = *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1 1**
      have ⟨t, f⟩ = ⟨f,f⟩
      proof −
       have (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ *j* = ((⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ *right-coproj* **1** (**1** ∐ **1**)) $\circ_c$ *left-coproj* **1 1**
          by (*typecheck-cfuncs*, *simp add*: *case2 comp-associative2*)
        also have ... = (⟨t, f⟩ II⟨f, t⟩) $\circ_c$ *left-coproj* **1 1**
          using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
        also have ... = ⟨t, f⟩
          by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
        finally show *?thesis*
          using *j-def* **by** *simp*
      qed
      then show *?thesis*
        by *blast*
    next
      assume *not-case2*: *j* ≠ *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1 1**
      then have *case3*: *j* = *right-coproj* **1** (**1**∐**1**) $\circ_c$ *right-coproj* **1 1**
        using *case2-or-3* **by** *blast*
      have ⟨f, t⟩ = ⟨f,f⟩
      proof −
       have (⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ *j* = ((⟨t, t⟩II (⟨t, f⟩ II⟨f, t⟩)) $\circ_c$ *right-coproj* **1** (**1** ∐ **1**)) $\circ_c$ *right-coproj* **1 1**
          by (*typecheck-cfuncs*, *simp add*: *case3 comp-associative2*)
        also have ... = (⟨t, f⟩ II⟨f, t⟩) $\circ_c$ *right-coproj* **1 1**

using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = $\langle$f, t$\rangle$
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
finally show *?thesis*
using *j-def* **by** *simp*
qed
then show *?thesis*
by *blast*
qed
qed
then have t = f
using *trichotomy cart-prod-eq2* **by** (*typecheck-cfuncs*, *force*)
then show *False*
using *true-false-distinct* **by** *smt*
qed

lemma *OR-true-implies-one-is-true*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $OR \circ_c \langle p,q\rangle = \mathrm{t}$
shows $(p = \mathrm{t}) \vee (q = \mathrm{t})$
by (*metis OR-false-false-is-false assms true-false-only-truth-values*)

lemma *NOT-NOR-is-OR*:
$OR = NOT \circ_c NOR$
proof(*etcs-rule one-separator*)
fix x
assume *x-type*[*type-rule*]: $x \in_c \Omega \times_c \Omega$
then obtain p q **where** *p-type*[*type-rule*]: $p \in_c \Omega$ **and** *q-type*[*type-rule*]: $q \in_c \Omega$
and *x-def*: $x = \langle p,q\rangle$
by (*meson cart-prod-decomp*)
show $OR \circ_c x = (NOT \circ_c NOR) \circ_c x$
proof(*cases* $p = \mathrm{t}$)
show $p = \mathrm{t} \Longrightarrow OR \circ_c x = (NOT \circ_c NOR) \circ_c x$
by (*typecheck-cfuncs*, *metis NOR-left-true-is-false NOT-false-is-true OR-true-left-is-true comp-associative2 q-type x-def*)
next
assume $p \neq \mathrm{t}$
then have $p = \mathrm{f}$
using *p-type true-false-only-truth-values* **by** *blast*
show $OR \circ_c x = (NOT \circ_c NOR) \circ_c x$
proof(*cases* $q = \mathrm{t}$)
show $q = \mathrm{t} \Longrightarrow OR \circ_c x = (NOT \circ_c NOR) \circ_c x$
by (*typecheck-cfuncs*, *metis NOR-right-true-is-false NOT-false-is-true OR-true-right-is-true*

cfunc-type-def comp-associative p-type x-def)
next
assume $q \neq \mathrm{t}$
then show *?thesis*

by (*typecheck-cfuncs*,*metis NOR-false-false-is-true NOT-is-true-implies-false OR-false-false-is-false*
‹$p = \mathrm{f}$› *comp-associative2 q-type true-false-only-truth-values x-def*)
qed
qed
qed

lemma *OR-commutative*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $OR \circ_c \langle p,q\rangle = OR \circ_c \langle q,p\rangle$
by (*metis OR-true-left-is-true OR-true-right-is-true assms true-false-only-truth-values*)

lemma *OR-idempotent*:
assumes $p \in_c \Omega$
shows $OR \circ_c \langle p,p\rangle = p$
using *OR-false-false-is-false OR-true-left-is-true assms true-false-only-truth-values*
by *blast*

lemma *OR-associative*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $r \in_c \Omega$
shows $OR \circ_c \langle OR \circ_c \langle p,q\rangle, r\rangle = OR \circ_c \langle p, OR \circ_c \langle q,r\rangle\rangle$
by (*metis OR-commutative OR-false-false-is-false OR-true-right-is-true assms true-false-only-truth-values*)

lemma *OR-complementary*:
assumes $p \in_c \Omega$
shows $OR \circ_c \langle p, NOT \circ_c p\rangle = \mathrm{t}$
by (*metis NOT-false-is-true NOT-true-is-false OR-true-left-is-true OR-true-right-is-true assms false-func-type true-false-only-truth-values*)

14.5 XOR

definition *XOR* :: *cfunc* **where**
$XOR = (THE\ \chi.\ is\text{-}pullback\ (\mathbf{1}\coprod\mathbf{1})\ \mathbf{1}\ (\Omega\times_c\Omega)\ \Omega\ (\beta_{(\mathbf{1}\coprod\mathbf{1})})\ \mathrm{t}\ (\langle \mathrm{t}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle)\ \chi)$

lemma *pre-XOR-type*[*type-rule*]:
$\langle \mathrm{t}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle : \mathbf{1}\coprod\mathbf{1} \to \Omega \times_c \Omega$
by *typecheck-cfuncs*

lemma *pre-XOR-injective*:
injective$(\langle \mathrm{t}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle)$
unfolding *injective-def*
proof(*clarify*)
fix $x\ y$
assume $x \in_c$ *domain* $(\langle \mathrm{t},\mathrm{f}\rangle \amalg \langle \mathrm{f},\mathrm{t}\rangle)$
then have *x-type*: $x \in_c \mathbf{1}\coprod\mathbf{1}$

using *cfunc-type-def pre-XOR-type* **by** *force*
then have *x-form*: $(\exists\ w.\ w \in_c \mathbf{1} \wedge x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w)$
$\vee\ (\exists\ w.\ w \in_c \mathbf{1} \wedge x = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w)$
using *coprojs-jointly-surj* **by** *auto*

assume $y \in_c \textit{domain}\ (\langle t,f\rangle \amalg \langle f,t\rangle)$
then have *y-type*: $y \in_c \mathbf{1} \coprod \mathbf{1}$
using *cfunc-type-def pre-XOR-type* **by** *force*
then have *y-form*: $(\exists\ w.\ w \in_c \mathbf{1} \wedge y = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w)$
$\vee\ (\exists\ w.\ w \in_c \mathbf{1} \wedge y = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w)$
using *coprojs-jointly-surj* **by** *auto*

assume *eqs*: $\langle t,f\rangle \amalg \langle f,t\rangle \circ_c x = \langle t,f\rangle \amalg \langle f,t\rangle \circ_c y$

show $x = y$
proof(*cases* $\exists\ w.\ w \in_c \mathbf{1} \wedge x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$)
assume *a1*: $\exists\ w.\ w \in_c \mathbf{1} \wedge x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$
then obtain w **where** *x-def*: $w \in_c \mathbf{1} \wedge x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$
by *blast*
then have *w-is*: $w = \textit{id}(\mathbf{1})$
by (*typecheck-cfuncs*, *metis terminal-func-unique x-def*)
have $\exists\ v.\ v \in_c \mathbf{1} \wedge y = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
proof(*rule ccontr*)
assume *a2*: $\nexists v.\ v \in_c \mathbf{1} \wedge y = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
then obtain v **where** *y-def*: $v \in_c \mathbf{1} \wedge y = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
using *y-form* **by** (*typecheck-cfuncs*, *blast*)
then have *v-is*: $v = \textit{id}(\mathbf{1})$
by (*typecheck-cfuncs*, *metis terminal-func-unique y-def*)
then have $\langle t,f\rangle \amalg \langle f,t\rangle \circ_c \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} = \langle t,f\rangle \amalg \langle f,t\rangle \circ_c \textit{right-coproj}\ \mathbf{1}\ \mathbf{1}$
using *w-is eqs id-right-unit2 x-def y-def* **by** (*typecheck-cfuncs*, *force*)
then have $\langle t,f\rangle = \langle f,t\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-unique coprod-eq2 pre-XOR-type right-coproj-cfunc-coprod*)
then have $t = f \wedge f = t$
using *cart-prod-eq2 false-func-type true-func-type* **by** *blast*
then show *False*
using *true-false-distinct* **by** *blast*
qed
then obtain v **where** *y-def*: $v \in_c \mathbf{1} \wedge y = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
by *blast*
then have $v = \textit{id}(\mathbf{1})$
by (*typecheck-cfuncs*, *metis terminal-func-unique*)
then show *?thesis*
by (*simp add*: *w-is x-def y-def*)
next
assume $\nexists w.\ w \in_c \mathbf{1} \wedge x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$
then obtain w **where** *x-def*: $w \in_c \mathbf{1} \wedge x = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$
using *x-form* **by** *force*
then have *w-is*: $w = \textit{id}\ \mathbf{1}$

by (*typecheck-cfuncs*, *metis terminal-func-unique x-def*)
have ∃ *v*. *v* $\in_c$ **1** ∧ *y* = *right-coproj* **1 1** $\circ_c$ *v*
proof(*rule ccontr*)
assume *a2*: ∄ *v*. *v* $\in_c$ **1** ∧ *y* = *right-coproj* **1 1** $\circ_c$ *v*
then obtain *v* **where** *y-def*: *v* $\in_c$ **1** ∧ *y* = *left-coproj* **1 1** $\circ_c$ *v*
using *y-form* **by** (*typecheck-cfuncs*, *blast*)
then have *v* = *id* **1**
by (*typecheck-cfuncs*, *metis terminal-func-unique y-def*)
then have ⟨t,f⟩ ∐ ⟨f,t⟩ $\circ_c$ *left-coproj* **1 1** = ⟨t,f⟩ ∐ ⟨f,t⟩ $\circ_c$ *right-coproj* **1 1**
using *w-is eqs id-right-unit2 x-def y-def* **by** (*typecheck-cfuncs*, *force*)
then have ⟨t,f⟩ = ⟨f,t⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-unique coprod-eq2 pre-XOR-type right-coproj-cfunc-coprod*)
then have t = f ∧ f = t
using *cart-prod-eq2 false-func-type true-func-type* **by** *blast*
then show *False*
using *true-false-distinct* **by** *blast*
qed
then obtain *v* **where** *y-def*: *v* $\in_c$ **1** ∧ *y* = *right-coproj* **1 1** $\circ_c$ *v*
by *blast*
then have *v* = *id* **1**
by (*typecheck-cfuncs*, *metis terminal-func-unique*)
then show *?thesis*
by (*simp add*: *w-is x-def y-def*)
qed
qed

lemma *XOR-is-pullback*:
is-pullback (**1**∐**1**) **1** (Ω $\times_c$ Ω) Ω ($\beta_{(\mathbf{1}\coprod\mathbf{1})}$) t (⟨t, f⟩ ∐ ⟨f, t⟩) *XOR*
unfolding *XOR-def*
using *element-monomorphism characteristic-function-exists*
by (*typecheck-cfuncs*, *simp add*: *the1I2 injective-imp-monomorphism pre-XOR-injective*)

lemma *XOR-type*[*type-rule*]:
XOR : Ω $\times_c$ Ω → Ω
unfolding *XOR-def*
by (*metis XOR-def XOR-is-pullback is-pullback-def*)

lemma *XOR-only-true-left-is-true*:
XOR $\circ_c$ ⟨t,f⟩ = t
proof −
have ∃ *j*. *j* $\in_c$ **1**∐**1** ∧ (⟨t, f⟩ ∐⟨f, t⟩) $\circ_c$ *j* = ⟨t,f⟩
by (*typecheck-cfuncs*, *meson left-coproj-cfunc-coprod left-proj-type*)
then show *?thesis*
by (*smt* (*verit*, *best*) *XOR-is-pullback comp-associative2 id-right-unit2 is-pullback-def terminal-func-comp-elem*)
qed

lemma *XOR-only-true-right-is-true*:

XOR $\circ_c$ ⟨f,t⟩ = t
proof −
have ∃ *j*. *j* $\in_c$ $\mathbf{1} \coprod \mathbf{1}$ ∧ (⟨t, f⟩ ⨿⟨f, t⟩) $\circ_c$ *j* = ⟨f,t⟩
by (*typecheck-cfuncs*, *meson right-coproj-cfunc-coprod right-proj-type*)
then show *?thesis*
by (*smt* (*verit*, *best*) *XOR-is-pullback comp-associative2 id-right-unit2 is-pullback-def terminal-func-comp-elem*)
qed

lemma *XOR-false-false-is-false*:
XOR $\circ_c$ ⟨f,f⟩ = f
proof(*rule ccontr*)
assume *XOR* $\circ_c$ ⟨f,f⟩ ≠ f
then have *XOR* $\circ_c$ ⟨f,f⟩ = t
by (*metis NOR-is-pullback XOR-type comp-type is-pullback-def true-false-only-truth-values*)
then obtain *j* **where** *j-def*: *j* $\in_c$ $\mathbf{1} \coprod \mathbf{1}$ ∧ (⟨t, f⟩ ⨿⟨f, t⟩) $\circ_c$ *j* = ⟨f,f⟩
by (*typecheck-cfuncs*, *auto*, *smt* (*verit*, *ccfv-threshold*) *XOR-is-pullback id-right-unit2 id-type is-pullback-def*)
show *False*
proof(*cases j* = *left-coproj* **1 1**)
assume *j* = *left-coproj* **1 1**
then have (⟨t, f⟩ ⨿⟨f, t⟩) $\circ_c$ *j* = ⟨t, f⟩
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have ⟨t, f⟩ = ⟨f,f⟩
using *j-def* **by** *auto*
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
next
assume *j* ≠ *left-coproj* **1 1**
then have *j* = *right-coproj* **1 1**
by (*meson j-def maps-into-1u1*)
then have (⟨t, f⟩ ⨿⟨f, t⟩) $\circ_c$ *j* = ⟨f, t⟩
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have ⟨f, t⟩ = ⟨f,f⟩
using *j-def* **by** *auto*
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed
qed

lemma *XOR-true-true-is-false*:
XOR $\circ_c$ ⟨t,t⟩ = f
proof(*rule ccontr*)
assume *XOR* $\circ_c$ ⟨t,t⟩ ≠ f
then have *XOR* $\circ_c$ ⟨t,t⟩ = t

by (*metis XOR-type comp-type diag-on-elements diagonal-type true-false-only-truth-values true-func-type*)
then obtain *j* **where** *j-def*: $j \in_c \mathbf{1} \coprod \mathbf{1} \wedge (\langle t, f\rangle \amalg \langle f, t\rangle) \circ_c j = \langle t,t\rangle$
by (*typecheck-cfuncs*, *auto*, *smt* (*verit*, *ccfv-threshold*) *XOR-is-pullback id-right-unit2 id-type is-pullback-def*)
show *False*
proof(*cases* j = *left-coproj* **1 1**)
assume j = *left-coproj* **1 1**
then have $(\langle t, f\rangle \amalg \langle f, t\rangle) \circ_c j = \langle t, f\rangle$
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have $\langle t, f\rangle = \langle t,t\rangle$
using *j-def* **by** *auto*
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
next
assume $j \neq$ *left-coproj* **1 1**
then have j = *right-coproj* **1 1**
by (*meson j-def maps-into-1u1*)
then have $(\langle t, f\rangle \amalg \langle f, t\rangle) \circ_c j = \langle f, t\rangle$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have $\langle f, t\rangle = \langle t,t\rangle$
using *j-def* **by** *auto*
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed
qed

14.6 NAND

definition *NAND* :: *cfunc* **where**
NAND = (*THE* χ. *is-pullback* $(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))$ **1** $(\Omega \times_c \Omega)$ Ω $(\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))})$ t $(\langle f, f\rangle \amalg (\langle t, f\rangle \amalg \langle f, t\rangle))$ χ)

lemma *pre-NAND-type*[*type-rule*]:
$\langle f, f\rangle \amalg (\langle t, f\rangle \amalg \langle f, t\rangle) : \mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \rightarrow \Omega \times_c \Omega$
by *typecheck-cfuncs*

lemma *pre-NAND-injective*:
injective$(\langle f, f\rangle \amalg (\langle t, f\rangle \amalg \langle f, t\rangle))$
unfolding *injective-def*
proof(*clarify*)
fix *x y*
assume *x-type*: $x \in_c$ *domain* $(\langle f, f\rangle \amalg \langle t,f\rangle \amalg \langle f,t\rangle)$
then have *x-type'*: $x \in_c \mathbf{1} \coprod (\mathbf{1}\coprod\mathbf{1})$
using *cfunc-type-def pre-NAND-type* **by** *force*

then have *x-form*: ($\exists$ *w*. $w \in_c$ **1** $\wedge$ x = *left-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *w*)
$\vee$ ($\exists$ *w*. $w \in_c$ **1**$\coprod$**1** $\wedge$ x = *right-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *w*)
using *coprojs-jointly-surj* **by** *auto*

assume *y-type*: $y \in_c$ *domain* (⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩)
then have *y-type'*: $y \in_c$ **1**$\coprod$ (**1**$\coprod$**1**)
using *cfunc-type-def pre-NAND-type* **by** *force*
then have *y-form*: ($\exists$ *w*. $w \in_c$ **1** $\wedge$ y = *left-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *w*)
$\vee$ ($\exists$ *w*. $w \in_c$ **1**$\coprod$**1** $\wedge$ y = *right-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *w*)
using *coprojs-jointly-surj* **by** *auto*

assume *mx-eqs-my*: ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *x* = ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *y*

have *f1*: ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *left-coproj* **1** (**1** $\coprod$ **1**) = ⟨f, f⟩
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
have *f2*: ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ (*right-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *left-coproj* **1 1**) = ⟨t,f⟩
proof−
have ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *left-coproj* **1 1** =
(⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1** (**1**$\coprod$**1**)) $\circ_c$ *left-coproj* **1 1**
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *left-coproj* **1 1**
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *smt*)
also have ... = ⟨t,f⟩
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
finally show *?thesis*.
qed
have *f3*: ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ (*right-coproj* **1** (**1**$\coprod$**1**)$\circ_c$ *right-coproj* **1 1**) =
⟨f,t⟩
proof−
have ⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ (*right-coproj* **1** (**1**$\coprod$**1**)$\circ_c$ *right-coproj* **1 1**) =
(⟨f, f⟩ ⨿ ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1** (**1**$\coprod$**1**))$\circ_c$ *right-coproj* **1 1**
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ⟨t,f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1 1**
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *smt*)
also have ... = ⟨f,t⟩
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
finally show *?thesis*.
qed
show $x = y$
proof(*cases* x = *left-coproj* **1** (**1** $\coprod$ **1**))
assume *case1*: x = *left-coproj* **1** (**1** $\coprod$ **1**)
then show $x = y$
by (*typecheck-cfuncs*, *smt* (*z3*) *mx-eqs-my element-pair-eq f1 f2 f3 false-func-type maps-into-1u1 terminal-func-unique true-false-distinct true-func-type x-form y-form*)
next
assume *not-case1*: $x \neq$ *left-coproj* **1** (**1** $\coprod$ **1**)
then have *case2-or-3*: x = *right-coproj* **1** (**1**$\coprod$**1**)$\circ_c$ *left-coproj* **1 1** $\vee$
x = *right-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *right-coproj* **1 1**
by (*metis id-right-unit2 id-type left-proj-type maps-into-1u1 terminal-func-unique*

x-form)
  show $x = y$
  proof(*cases* $x =$ *right-coproj* **1** ($\mathbf{1}\coprod\mathbf{1}$)$\circ_c$ *left-coproj* **1 1**)
    assume *case2*: $x =$ *right-coproj* **1** ($\mathbf{1} \coprod \mathbf{1}$) $\circ_c$ *left-coproj* **1 1**
    then show $x = y$
     by (*smt* (*z3*) *NOT-false-is-true NOT-is-pullback NOT-true-is-false NOT-type x-type x-type′ cart-prod-eq2 case2 cfunc-type-def characteristic-func-eq characteristic-func-is-pullback characteristic-function-exists comp-associative diag-on-elements diagonal-type element-monomorphism f1 f2 f3 false-func-type left-proj-type maps-into-1u1 mx-eqs-my terminal-func-unique true-false-distinct true-func-type x-type y-form*)
  next
    assume *not-case2*: $x \neq$ *right-coproj* **1** ($\mathbf{1} \coprod \mathbf{1}$) $\circ_c$ *left-coproj* **1 1**
    then have *case3*: $x =$ *right-coproj* **1** ($\mathbf{1}\coprod\mathbf{1}$) $\circ_c$ *right-coproj* **1 1**
      using *case2-or-3* **by** *blast*
    then show $x = y$
     by (*smt* (*z3*) *NOT-false-is-true NOT-is-pullback NOT-true-is-false NOT-type x-type x-type′ cart-prod-eq2 case3 cfunc-type-def characteristic-func-eq characteristic-func-is-pullback characteristic-function-exists comp-associative diag-on-elements diagonal-type element-monomorphism f1 f2 f3 false-func-type left-proj-type maps-into-1u1 mx-eqs-my terminal-func-unique true-false-distinct true-func-type x-type y-form*)
  qed
 qed
qed

lemma *NAND-is-pullback*:
 is-pullback ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$) **1** ($\Omega\times_c\Omega$) Ω ($\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))}$) t ($\langle$f, f$\rangle$II ($\langle$t, f$\rangle$ II$\langle$f, t$\rangle$)) *NAND*
 unfolding *NAND-def*
 using *element-monomorphism characteristic-function-exists*
 by (*typecheck-cfuncs*, *simp add*: *the1I2 injective-imp-monomorphism pre-NAND-injective*)

lemma *NAND-type*[*type-rule*]:
 NAND : $\Omega \times_c \Omega \to \Omega$
 unfolding *NAND-def*
 by (*metis NAND-def NAND-is-pullback is-pullback-def*)

lemma *NAND-left-false-is-true*:
 assumes $p \in_c \Omega$
 shows *NAND* $\circ_c$ $\langle$f,$p\rangle$ = t
proof −
 have $\exists$ *j*. $j \in_c \mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \wedge$ ($\langle$f, f$\rangle$ II ($\langle$t, f$\rangle$ II$\langle$f, t$\rangle$)) $\circ_c$ j = $\langle$f,$p\rangle$
  by (*typecheck-cfuncs*, *smt* (*z3*) *assms comp-associative2 comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type true-false-only-truth-values*)
 then show *?thesis*
  by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *NAND-is-pullback comp-associative2 id-right-unit2 is-pullback-def terminal-func-comp-elem*)
qed

lemma *NAND-right-false-is-true*:

assumes $p \in_c \Omega$
shows $NAND \circ_c \langle p,\mathrm{f}\rangle = \mathrm{t}$
proof −
have $\exists\ j.\ j \in_c \mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \wedge (\langle \mathrm{f},\ \mathrm{f}\rangle \amalg (\langle \mathrm{t},\ \mathrm{f}\rangle \amalg \langle \mathrm{f},\ \mathrm{t}\rangle)) \circ_c j = \langle p,\mathrm{f}\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *assms comp-associative2 comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type true-false-only-truth-values*)
then show *?thesis*
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *NAND-is-pullback NOT-false-is-true NOT-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *NAND-true-true-is-false*:
$NAND \circ_c \langle \mathrm{t},\mathrm{t}\rangle = \mathrm{f}$
proof(*rule ccontr*)
assume $NAND \circ_c \langle \mathrm{t},\mathrm{t}\rangle \neq \mathrm{f}$
then have $NAND \circ_c \langle \mathrm{t},\mathrm{t}\rangle = \mathrm{t}$
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then obtain *j* **where** *j-type*[*type-rule*]: $j \in_c \mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$ **and** *j-def*: $(\langle \mathrm{f},\ \mathrm{f}\rangle \amalg (\langle \mathrm{t},\ \mathrm{f}\rangle \amalg \langle \mathrm{f},\ \mathrm{t}\rangle)) \circ_c j = \langle \mathrm{t},\mathrm{t}\rangle$
using *NAND-is-pullback* **unfolding** *is-pullback-def*
by (*typecheck-cfuncs*, *smt* (*z3*) *NAND-is-pullback id-right-unit2 id-type*)
then have *trichotomy*: $(\langle \mathrm{f},\mathrm{f}\rangle = \langle \mathrm{t},\mathrm{t}\rangle) \vee (\langle \mathrm{t},\ \mathrm{f}\rangle = \langle \mathrm{t},\mathrm{t}\rangle) \vee (\langle \mathrm{f},\ \mathrm{t}\rangle = \langle \mathrm{t},\mathrm{t}\rangle)$
proof(*cases* $j =$ *left-coproj* $\mathbf{1}$ $(\mathbf{1} \coprod \mathbf{1})$)
assume *case1*: $j =$ *left-coproj* $\mathbf{1}$ $(\mathbf{1} \coprod \mathbf{1})$
then show *?thesis*
by (*metis cfunc-coprod-type cfunc-prod-type false-func-type j-def left-coproj-cfunc-coprod true-func-type*)
next
assume *not-case1*: $j \neq$ *left-coproj* $\mathbf{1}$ $(\mathbf{1} \coprod \mathbf{1})$
then have *case2-or-3*: $j =$ *right-coproj* $\mathbf{1}$ $(\mathbf{1}\coprod\mathbf{1}) \circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$ $\vee$
$j =$ *right-coproj* $\mathbf{1}$ $(\mathbf{1}\coprod\mathbf{1}) \circ_c$ *right-coproj* $\mathbf{1}$ $\mathbf{1}$
using *not-case1 set-three* **by** (*typecheck-cfuncs*, *auto*)
show *?thesis*
proof(*cases* $j =$ *right-coproj* $\mathbf{1}$ $(\mathbf{1}\coprod\mathbf{1}) \circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$)
assume *case2*: $j =$ *right-coproj* $\mathbf{1}$ $(\mathbf{1} \coprod \mathbf{1}) \circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$
have $\langle \mathrm{t},\ \mathrm{f}\rangle = \langle \mathrm{t},\mathrm{t}\rangle$
proof −
have $(\langle \mathrm{f},\ \mathrm{f}\rangle \amalg (\langle \mathrm{t},\ \mathrm{f}\rangle \amalg \langle \mathrm{f},\ \mathrm{t}\rangle)) \circ_c j = ((\langle \mathrm{f},\ \mathrm{f}\rangle \amalg (\langle \mathrm{t},\ \mathrm{f}\rangle \amalg \langle \mathrm{f},\ \mathrm{t}\rangle)) \circ_c$ *right-coproj* $\mathbf{1}$ $(\mathbf{1} \coprod \mathbf{1})) \circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$
by (*typecheck-cfuncs*, *simp add*: *case2 comp-associative2*)
also have ... $= (\langle \mathrm{t},\ \mathrm{f}\rangle \amalg \langle \mathrm{f},\ \mathrm{t}\rangle) \circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= \langle \mathrm{t},\ \mathrm{f}\rangle$
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
finally show *?thesis*
using *j-def* **by** *simp*
qed
then show *?thesis*
by *blast*

next
assume *not-case2*: $j \neq$ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *left-coproj* **1 1**
then have *case3*: $j =$ *right-coproj* **1** (**1**$\coprod$**1**) $\circ_c$ *right-coproj* **1 1**
using *case2-or-3* **by** *blast*
have ⟨f, t⟩ = ⟨t,t⟩
proof −
have (⟨f, f⟩⨿ (⟨t, f⟩ ⨿⟨f, t⟩)) $\circ_c$ j = ((⟨f, f⟩⨿ (⟨t, f⟩ ⨿⟨f, t⟩)) $\circ_c$ *right-coproj* **1** (**1** $\coprod$ **1**)) $\circ_c$ *right-coproj* **1 1**
by (*typecheck-cfuncs*, *simp add*: *case3 comp-associative2*)
also have ... = (⟨t, f⟩ ⨿⟨f, t⟩) $\circ_c$ *right-coproj* **1 1**
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = ⟨f, t⟩
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
finally show *?thesis*
using *j-def* **by** *simp*
qed
then show *?thesis*
by *blast*
qed
qed
then have t = f
using *trichotomy cart-prod-eq2* **by** (*typecheck-cfuncs*, *force*)
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *NAND-true-implies-one-is-false*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes *NAND* $\circ_c$ ⟨p,q⟩ = t
shows p = f ∨ q = f
by (*metis* (*no-types*) *NAND-true-true-is-false assms true-false-only-truth-values*)

lemma *NOT-AND-is-NAND*:
NAND = *NOT* $\circ_c$ *AND*
proof(*etcs-rule one-separator*)
fix x
assume *x-type*: $x \in_c \Omega \times_c \Omega$
then obtain p q **where** *x-def*: $p \in_c \Omega \wedge q \in_c \Omega \wedge x = \langle p,q \rangle$
by (*meson cart-prod-decomp*)
show *NAND* $\circ_c$ x = (*NOT* $\circ_c$ *AND*) $\circ_c$ x
by (*typecheck-cfuncs*, *metis AND-false-left-is-false AND-false-right-is-false AND-true-true-is-true NAND-left-false-is-true NAND-right-false-is-true NAND-true-implies-one-is-false NOT-false-is-true NOT-true-is-false comp-associative2 true-false-only-truth-values x-def x-type*)
qed

lemma *NAND-not-idempotent*:
assumes $p \in_c \Omega$
shows *NAND* $\circ_c$ ⟨p,p⟩ = *NOT* $\circ_c$ p

using *NAND-right-false-is-true NAND-true-true-is-false NOT-false-is-true NOT-true-is-false assms true-false-only-truth-values* **by** *fastforce*

14.7 IFF

definition *IFF* :: *cfunc* **where**
$IFF = (THE\ \chi.\ is\text{-}pullback\ (\mathbf{1}\coprod\mathbf{1})\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1}\coprod\mathbf{1})})\ t\ (\langle t, t\rangle\ \amalg\langle f, f\rangle)\ \chi)$

lemma *pre-IFF-type*[*type-rule*]:
$\langle t, t\rangle\ \amalg\ \langle f, f\rangle : \mathbf{1}\coprod\mathbf{1} \to \Omega \times_c \Omega$
by *typecheck-cfuncs*

lemma *pre-IFF-injective*:
injective($\langle t, t\rangle\ \amalg\langle f, f\rangle$)
unfolding *injective-def*
proof(*clarify*)
fix $x\ y$
assume $x \in_c$ *domain* ($\langle t, t\rangle\ \amalg\langle f, f\rangle$)
then have *x-type*: $x \in_c (\mathbf{1}\coprod\mathbf{1})$
using *cfunc-type-def pre-IFF-type* **by** *force*
then have *x-form*: $(\exists\ w.\ (w \in_c \mathbf{1} \wedge x = (left\text{-}coproj\ \mathbf{1}\ \mathbf{1}) \circ_c w))$
$\vee\ (\exists\ w.\ (w \in_c \mathbf{1} \wedge x = (right\text{-}coproj\ \mathbf{1}\ \mathbf{1}) \circ_c w))$
using *coprojs-jointly-surj* **by** *auto*

assume $y \in_c$ *domain* ($\langle t, t\rangle\ \amalg\langle f, f\rangle$)
then have *y-type*: $y \in_c (\mathbf{1}\coprod\mathbf{1})$
using *cfunc-type-def pre-IFF-type* **by** *force*
then have *y-form*: $(\exists\ w.\ (w \in_c \mathbf{1} \wedge y = (left\text{-}coproj\ \mathbf{1}\ \mathbf{1}) \circ_c w))$
$\vee\ (\exists\ w.\ (w \in_c \mathbf{1} \wedge y = (right\text{-}coproj\ \mathbf{1}\ \mathbf{1}) \circ_c w))$
using *coprojs-jointly-surj* **by** *auto*

assume *eqs*: $\langle t, t\rangle\ \amalg\langle f, f\rangle \circ_c x = \langle t, t\rangle\ \amalg\langle f, f\rangle \circ_c y$

show $x = y$
proof(*cases* $\exists\ w.\ w \in_c \mathbf{1} \wedge x = left\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c w$)
assume *a1*: $\exists\ w.\ w \in_c \mathbf{1} \wedge x = left\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c w$
then obtain w **where** *x-def*: $w \in_c \mathbf{1} \wedge x = left\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c w$
by *blast*
then have $w = id\ \mathbf{1}$
by (*typecheck-cfuncs*, *metis terminal-func-unique x-def*)
have $\exists\ v.\ v \in_c \mathbf{1} \wedge y = left\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c v$
proof(*rule ccontr*)
assume *a2*: $\nexists v.\ v \in_c \mathbf{1} \wedge y = left\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c v$
then obtain v **where** *y-def*: $v \in_c \mathbf{1} \wedge y = right\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c v$
using *y-form* **by** (*typecheck-cfuncs*, *blast*)
then have $v = id\ \mathbf{1}$
by (*typecheck-cfuncs*, *metis terminal-func-unique y-def*)
then have $\langle t, t\rangle\ \amalg\langle f, f\rangle \circ_c left\text{-}coproj\ \mathbf{1}\ \mathbf{1} = \langle t, t\rangle\ \amalg\langle f, f\rangle \circ_c right\text{-}coproj\ \mathbf{1}\ \mathbf{1}$
using ‹$v = id_c\ \mathbf{1}$› ‹$w = id_c\ \mathbf{1}$› *eqs id-right-unit2 x-def y-def* **by** (*typecheck-cfuncs*,

force)
then have ⟨t, t⟩ = ⟨f,f⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-unique coprod-eq2 pre-IFF-type right-coproj-cfunc-coprod*)
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* by *blast*
then show *False*
using *true-false-distinct* by *blast*
qed
then obtain *v* where *y-def*: $v \in_c \mathbf{1} \land y = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
by *blast*
then have $v = \textit{id}\ \mathbf{1}$
by (*typecheck-cfuncs*, *metis terminal-func-unique*)
then show *?thesis*
by (*simp add*: ‹$w = id_c$ **1**› *x-def y-def*)
next
assume $\nexists w.\ w \in_c \mathbf{1} \land x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$
then obtain *w* where *x-def*: $w \in_c \mathbf{1} \land x = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c w$
using *x-form* by *force*
then have $w = \textit{id}\ \mathbf{1}$
by (*typecheck-cfuncs*, *metis terminal-func-unique x-def*)
have $\exists\ v.\ v \in_c \mathbf{1} \land y = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
proof(*rule ccontr*)
assume *a2*: $\nexists v.\ v \in_c \mathbf{1} \land y = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
then obtain *v* where *y-def*: $v \in_c \mathbf{1} \land y = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1} \circ_c v$
using *y-form* by (*typecheck-cfuncs*, *blast*)
then have $v = \textit{id}\ \mathbf{1}$
by (*typecheck-cfuncs*, *metis terminal-func-unique y-def*)
then have ⟨t, t⟩ II⟨f, f⟩ $\circ_c$ *left-coproj* **1 1** = ⟨t, t⟩ II⟨f, f⟩ $\circ_c$ *right-coproj* **1 1**
using ‹$v = id_c$ **1**› ‹$w = id_c$ **1**› *eqs id-right-unit2 x-def y-def* by (*typecheck-cfuncs*, *force*)
then have ⟨t, t⟩ = ⟨f, f⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-coprod-unique coprod-eq2 pre-IFF-type right-coproj-cfunc-coprod*)
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* by *blast*
then show *False*
using *true-false-distinct* by *blast*
qed
then obtain *v* where *y-def*: $v \in_c \mathbf{1} \land y = (\textit{right-coproj}\ \mathbf{1}\ \mathbf{1}) \circ_c v$
by *blast*
then have $v = \textit{id}\ \mathbf{1}$
by (*typecheck-cfuncs*, *metis terminal-func-unique*)
then show *?thesis*
by (*simp add*: ‹$w = id_c$ **1**› *x-def y-def*)
qed
qed

lemma *IFF-is-pullback*:

is-pullback $(\mathbf{1}\coprod\mathbf{1})$ $\mathbf{1}$ $(\Omega\times_c\Omega)$ Ω $(\beta_{(\mathbf{1}\coprod\mathbf{1})})$ t (⟨t, t⟩ ⨿⟨f, f⟩) *IFF*
unfolding *IFF-def*
using *element-monomorphism characteristic-function-exists*
by (*typecheck-cfuncs*, *simp add*: *the1I2 injective-imp-monomorphism pre-IFF-injective*)

lemma *IFF-type*[*type-rule*]:
IFF : $\Omega \times_c \Omega \to \Omega$
unfolding *IFF-def*
by (*metis IFF-def IFF-is-pullback is-pullback-def*)

lemma *IFF-true-true-is-true*:
IFF $\circ_c$ ⟨t,t⟩ = t
proof −
have ∃ *j*. *j* $\in_c$ $(\mathbf{1}\coprod\mathbf{1})$ ∧ (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ *j* = ⟨t,t⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type true-false-only-truth-values*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *AND-is-pullback AND-true-true-is-true IFF-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *IFF-false-false-is-true*:
IFF $\circ_c$ ⟨f,f⟩ = t
proof −
have ∃ *j*. *j* $\in_c$ $(\mathbf{1}\coprod\mathbf{1})$ ∧ (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ *j* = ⟨f,f⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type true-false-only-truth-values*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *AND-is-pullback AND-true-true-is-true IFF-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *IFF-true-false-is-false*:
IFF $\circ_c$ ⟨t,f⟩ = f
proof(*rule ccontr*)
assume *IFF* $\circ_c$ ⟨t,f⟩ ≠ f
then have *IFF* $\circ_c$ ⟨t,f⟩ = t
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then obtain *j* **where** *j-type*[*type-rule*]: *j* $\in_c$ $\mathbf{1}\coprod\mathbf{1}$ ∧ (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ *j* = ⟨t,f⟩
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *IFF-is-pullback characteristic-function-exists element-monomorphism is-pullback-def*)
show *False*
proof(*cases j* = *left-coproj* **1 1**)
assume *j* = *left-coproj* **1 1**
then have (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ *j* = ⟨t, t⟩
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have ⟨t, f⟩ = ⟨t,t⟩
using *j-type* **by** *argo*
then have t = f

using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
next
assume $j \neq$ *left-coproj* **1 1**
then have $j =$ *right-coproj* **1 1**
using *j-type maps-into-1u1* **by** *auto*
then have (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ j = ⟨f, f⟩
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have ⟨f, t⟩ = ⟨f, f⟩
using *XOR-false-false-is-false XOR-only-true-left-is-true j-type* **by** *argo*
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed
qed

lemma *IFF-false-true-is-false*:
IFF $\circ_c$ ⟨f,t⟩ = f
proof(*rule ccontr*)
assume *IFF* $\circ_c$ ⟨f,t⟩ ≠ f
then have *IFF* $\circ_c$ ⟨f,t⟩ = t
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then obtain j **where** *j-type*[*type-rule*]: $j \in_c$ **1**∐**1** **and** *j-def*: (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ j = ⟨f,t⟩
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *IFF-is-pullback id-right-unit2 is-pullback-def one-unique-element terminal-func-comp terminal-func-comp-elem terminal-func-unique*)
show *False*
proof(*cases* $j =$ *left-coproj* **1 1**)
assume $j =$ *left-coproj* **1 1**
then have (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ j = ⟨t, t⟩
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have ⟨f,t⟩ = ⟨t,t⟩
using *j-def* **by** *auto*
then have t = f
using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
next
assume $j \neq$ *left-coproj* **1 1**
then have $j =$ *right-coproj* **1 1**
using *j-type maps-into-1u1* **by** *blast*
then have (⟨t, t⟩ ⨿⟨f, f⟩) $\circ_c$ j = ⟨f, f⟩
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
then have ⟨f,t⟩ = ⟨f, f⟩
using *XOR-false-false-is-false XOR-only-true-left-is-true j-def* **by** *fastforce*
then have t = f

using *cart-prod-eq2 false-func-type true-func-type* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed
qed

lemma *NOT-IFF-is-XOR*:
$NOT \circ_c IFF = XOR$
proof(*etcs-rule one-separator*)
fix x
assume *x-type*: $x \in_c \Omega \times_c \Omega$
then obtain u w **where** *x-def*: $u \in_c \Omega \wedge w \in_c \Omega \wedge x = \langle u,w \rangle$
using *cart-prod-decomp* **by** *blast*
show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
proof(*cases* $u = \mathrm{t}$)
show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
proof(*cases* $w = \mathrm{t}$)
show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
by (*metis IFF-false-false-is-true IFF-false-true-is-false IFF-true-false-is-false IFF-true-true-is-true IFF-type NOT-false-is-true NOT-true-is-false NOT-type XOR-false-false-is-false XOR-only-true-left-is-true XOR-only-true-right-is-true XOR-true-true-is-false cfunc-type-def comp-associative true-false-only-truth-values x-def x-type*)
next
assume $w \neq \mathrm{t}$
then have $w = \mathrm{f}$
by (*metis true-false-only-truth-values x-def*)
then show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
by (*metis IFF-false-false-is-true IFF-true-false-is-false IFF-type NOT-false-is-true NOT-true-is-false NOT-type XOR-false-false-is-false XOR-only-true-left-is-true comp-associative2 true-false-only-truth-values x-def x-type*)
qed
next
assume $u \neq \mathrm{t}$
then have $u = \mathrm{f}$
by (*metis true-false-only-truth-values x-def*)
show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
proof(*cases* $w = \mathrm{t}$)
show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
by (*metis IFF-false-false-is-true IFF-false-true-is-false IFF-type NOT-false-is-true NOT-true-is-false NOT-type XOR-false-false-is-false XOR-only-true-right-is-true* ‹$u = \mathrm{f}$› *comp-associative2 true-false-only-truth-values x-def x-type*)
next
assume $w \neq \mathrm{t}$
then have $w = \mathrm{f}$
by (*metis true-false-only-truth-values x-def*)
then show $(NOT \circ_c IFF) \circ_c x = XOR \circ_c x$
by (*metis IFF-false-false-is-true IFF-type NOT-true-is-false NOT-type XOR-false-false-is-false* ‹$u = \mathrm{f}$› *cfunc-type-def comp-associative x-def x-type*)
qed

qed
qed

14.8 IMPLIES

definition *IMPLIES* :: *cfunc* **where**
IMPLIES = (*THE* χ. *is-pullback* ($\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1})$) $\mathbf{1}$ ($\Omega\times_c\Omega$) Ω ($\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))}$) t (⟨t, t⟩II (⟨f, f⟩ II⟨f, t⟩)) χ)

lemma *pre-IMPLIES-type*[*type-rule*]:
⟨t, t⟩ II (⟨f, f⟩ II ⟨f, t⟩) : $\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \to \Omega\times_c\Omega$
by *typecheck-cfuncs*

lemma *pre-IMPLIES-injective*:
injective(⟨t, t⟩ II (⟨f, f⟩ II⟨f, t⟩))
unfolding *injective-def*
proof(*clarify*)
fix *x y*
assume *a1*: $x \in_c$ *domain* (⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩)
then have *x-type*[*type-rule*]: $x \in_c (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))$
using *cfunc-type-def pre-IMPLIES-type* **by** *force*
then have *x-form*: ($\exists$ *w*. ($w \in_c \mathbf{1} \wedge x$ = (*left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ *w*))
$\vee$ ($\exists$ *w*. ($w \in_c (\mathbf{1}\coprod\mathbf{1}) \wedge x$ = (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ *w*))
using *coprojs-jointly-surj* **by** *auto*

assume $y \in_c$ *domain* (⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩)
then have *y-type*: $y \in_c (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))$
using *cfunc-type-def pre-IMPLIES-type* **by** *force*
then have *y-form*: ($\exists$ *w*. ($w \in_c \mathbf{1} \wedge y$ = (*left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ *w*))
$\vee$ ($\exists$ *w*. ($w \in_c (\mathbf{1}\coprod\mathbf{1}) \wedge y$ = (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)) $\circ_c$ *w*))
using *coprojs-jointly-surj* **by** *auto*

assume *mx-eqs-my*: ⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ *x* = ⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ *y*

have *f1*: ⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ *left-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$) = ⟨t,t⟩
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
have *f2*: ⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)$\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$) = ⟨f, f⟩
proof−
have ⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)$\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$) =
(⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ *right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$))$\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ *left-coproj* $\mathbf{1}$ $\mathbf{1}$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *smt*)
also have ... = ⟨f, f⟩
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
finally show *?thesis*.
qed
have *f3*: ⟨t,t⟩ II ⟨f, f⟩ II ⟨f,t⟩ $\circ_c$ (*right-coproj* $\mathbf{1}$ ($\mathbf{1}\coprod\mathbf{1}$)$\circ_c$ *right-coproj* $\mathbf{1}$ $\mathbf{1}$) =

⟨f,t⟩
proof−
have ⟨t,t⟩ ⨿ ⟨f, f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1** (**1**∐**1**)$\circ_c$ *right-coproj* **1** **1** =
(⟨t,t⟩ ⨿ ⟨f, f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1** (**1**∐**1**))$\circ_c$ *right-coproj* **1** **1**
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ⟨f, f⟩ ⨿ ⟨f,t⟩ $\circ_c$ *right-coproj* **1** **1**
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *smt*)
also have ... = ⟨f,t⟩
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
finally show *?thesis*.
qed
show $x = y$
proof(*cases* x = *left-coproj* **1** (**1** ∐ **1**))
assume *case1*: x = *left-coproj* **1** (**1** ∐ **1**)
then show $x = y$
by (*typecheck-cfuncs*, *smt* (*z3*) *mx-eqs-my element-pair-eq f1 f2 f3 false-func-type maps-into-1u1 terminal-func-unique true-false-distinct true-func-type x-form y-form*)
next
assume *not-case1*: $x \neq$ *left-coproj* **1** (**1** ∐ **1**)
then have *case2-or-3*: x = (*right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1** **1**)∨
x = *right-coproj* **1** (**1**∐**1**) $\circ_c$(*right-coproj* **1** **1**)
by (*metis id-right-unit2 id-type left-proj-type maps-into-1u1 terminal-func-unique x-form*)
show $x = y$
proof(*cases* x = *right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1** **1**)
assume *case2*: x = *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1** **1**
then show $x = y$
by (*typecheck-cfuncs*, *smt* (*z3*) *a1 NOT-false-is-true NOT-is-pullback cart-prod-eq2 cfunc-prod-comp cfunc-type-def characteristic-func-eq characteristic-func-is-pullback characteristic-function-exists comp-associative element-monomorphism f1 f2 f3 false-func-type left-proj-type maps-into-1u1 mx-eqs-my terminal-func-unique true-false-distinct true-func-type y-form*)
next
assume *not-case2*: $x \neq$ *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1** **1**
then have *case3*: x = *right-coproj* **1** (**1**∐**1**) $\circ_c$(*right-coproj* **1** **1**)
using *case2-or-3* **by** *blast*
then show $x = y$
by (*smt* (*z3*) *NOT-false-is-true NOT-is-pullback a1 cart-prod-eq2 cfunc-type-def characteristic-func-eq characteristic-func-is-pullback characteristic-function-exists comp-associative diag-on-elements diagonal-type element-monomorphism f1 f2 f3 false-func-type left-proj-type maps-into-1u1 mx-eqs-my terminal-func-unique true-false-distinct true-func-type x-type y-form*)
qed
qed
qed

lemma *IMPLIES-is-pullback*:
is-pullback (**1**∐(**1**∐**1**)) **1** (Ω$\times_c$Ω) Ω ($\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))}$) t (⟨t, t⟩⨿ (⟨f, f⟩ ⨿⟨f, t⟩))
IMPLIES

unfolding *IMPLIES-def*
using *element-monomorphism characteristic-function-exists*
by (*typecheck-cfuncs*, *simp add*: *the1I2 injective-imp-monomorphism pre-IMPLIES-injective*)

lemma *IMPLIES-type*[*type-rule*]:
IMPLIES : $\Omega \times_c \Omega \rightarrow \Omega$
unfolding *IMPLIES-def*
by (*metis IMPLIES-def IMPLIES-is-pullback is-pullback-def*)

lemma *IMPLIES-true-true-is-true*:
IMPLIES $\circ_c$ ⟨t,t⟩ = t
proof −
have $\exists$ *j*. *j* $\in_c$ **1** ∐ (**1**∐**1**) ∧ (⟨t, t⟩⨿ (⟨f, f⟩ ⨿⟨f, t⟩)) $\circ_c$ *j* = ⟨t,t⟩
by (*typecheck-cfuncs*, *meson left-coproj-cfunc-coprod left-proj-type*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *IMPLIES-is-pullback NOT-false-is-true NOT-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *IMPLIES-false-true-is-true*:
IMPLIES $\circ_c$ ⟨f,t⟩ = t
proof −
have $\exists$ *j*. *j* $\in_c$ **1**∐(**1**∐**1**) ∧ (⟨t, t⟩⨿ (⟨f, f⟩ ⨿⟨f, t⟩)) $\circ_c$ *j* = ⟨f,t⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 comp-type right-coproj-cfunc-coprod right-proj-type*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *IMPLIES-is-pullback NOT-false-is-true NOT-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *IMPLIES-false-false-is-true*:
IMPLIES $\circ_c$ ⟨f,f⟩ = t
proof −
have $\exists$ *j*. *j* $\in_c$ **1**∐(**1**∐**1**) ∧ (⟨t, t⟩⨿ (⟨f, f⟩ ⨿⟨f, t⟩)) $\circ_c$ *j* = ⟨f,f⟩
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *cfunc-type-def comp-associative comp-type left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *IMPLIES-is-pullback NOT-false-is-true NOT-is-pullback comp-associative2 is-pullback-def terminal-func-comp*)
qed

lemma *IMPLIES-true-false-is-false*:
IMPLIES $\circ_c$ ⟨t,f⟩ = f
proof(*rule ccontr*)
assume *IMPLIES* $\circ_c$ ⟨t,f⟩ ≠ f
then have *IMPLIES* $\circ_c$ ⟨t,f⟩ = t
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then obtain *j* **where** *j-def*: *j* $\in_c$ **1**∐(**1**∐**1**) ∧ (⟨t, t⟩⨿ (⟨f, f⟩ ⨿⟨f, t⟩)) $\circ_c$ *j* = ⟨t,f⟩

by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-threshold*) *IMPLIES-is-pullback id-right-unit2 is-pullback-def one-unique-element terminal-func-comp terminal-func-comp-elem terminal-func-unique*)
show *False*
proof(*cases* $j = $ *left-coproj* **1** (**1**∐**1**))
assume *case1*: $j = $ *left-coproj* **1** (**1**∐**1**)
show *False*
proof −
have (⟨t, t⟩II (⟨f, f⟩ II⟨f, t⟩)) $\circ_c$ j = ⟨t, t⟩
by (*typecheck-cfuncs*, *simp add*: *case1 left-coproj-cfunc-coprod*)
then have ⟨t, t⟩ = ⟨t,f⟩
using *j-def* **by** *presburger*
then have t = f
using *IFF-true-false-is-false IFF-true-true-is-true* **by** *auto*
then show *False*
using *true-false-distinct* **by** *blast*
qed
next
assume $j \neq$ *left-coproj* **1** (**1** ∐ **1**)
then have *case2-or-3*: $j = $ *right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1** ∨
$j = $ *right-coproj* **1** (**1**∐**1**) $\circ_c$ *right-coproj* **1 1**
by (*metis coprojs-jointly-surj id-right-unit2 id-type j-def left-proj-type maps-into-1u1 one-unique-element*)
show *False*
proof(*cases* $j = $ *right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**)
assume *case2*: $j = $ *right-coproj* **1** (**1**∐**1**)$\circ_c$ *left-coproj* **1 1**
show *False*
proof −
have (⟨t, t⟩II (⟨f, f⟩ II⟨f, t⟩)) $\circ_c$ j = ⟨f, f⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *case2 comp-associative2 left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type*)
then have ⟨t, t⟩ = ⟨f,f⟩
using *XOR-false-false-is-false XOR-only-true-left-is-true j-def* **by** *auto*
then have t = f
by (*metis XOR-only-true-left-is-true XOR-true-true-is-false* ‹⟨t,t⟩ II ⟨f,f⟩ II ⟨f,t⟩ $\circ_c$ j = ⟨f,f⟩› *j-def*)
then show *False*
using *true-false-distinct* **by** *blast*
qed
next
assume $j \neq$ *right-coproj* **1** (**1** ∐ **1**) $\circ_c$ *left-coproj* **1 1**
then have *case3*: $j = $ *right-coproj* **1** (**1**∐**1**) $\circ_c$ *right-coproj* **1 1**
using *case2-or-3* **by** *blast*
show *False*
proof −
have (⟨t, t⟩II (⟨f, f⟩ II⟨f, t⟩)) $\circ_c$ j = ⟨f, t⟩
by (*typecheck-cfuncs*, *smt* (*z3*) *case3 comp-associative2 left-coproj-cfunc-coprod left-proj-type right-coproj-cfunc-coprod right-proj-type*)
then have ⟨t, t⟩ = ⟨f, t⟩

by (*metis cart-prod-eq2 false-func-type j-def true-func-type*)
then have t = f
using *XOR-only-true-right-is-true XOR-true-true-is-false* **by** *auto*
then show *False*
using *true-false-distinct* **by** *blast*
qed
qed
qed
qed

lemma *IMPLIES-false-is-true-false*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes *IMPLIES* $\circ_c$ $\langle p,q \rangle$ = f
shows p = t $\wedge$ q = f
by (*metis IMPLIES-false-false-is-true IMPLIES-false-true-is-true IMPLIES-true-true-is-true assms true-false-only-truth-values*)

ETCS analog to $(A \iff B) = (A \implies B) \wedge (B \implies A)$

lemma *iff-is-and-implies-implies-swap*:
IFF = *AND* $\circ_c$ $\langle$*IMPLIES*, *IMPLIES* $\circ_c$ *swap* Ω $\Omega\rangle$
proof(*etcs-rule one-separator*)
fix x
assume *x-type*: $x \in_c \Omega \times_c \Omega$
then obtain p q **where** *x-def*: $p \in_c \Omega \wedge q \in_c \Omega \wedge x = \langle p,q \rangle$
by (*meson cart-prod-decomp*)
show *IFF* $\circ_c$ x = (*AND* $\circ_c$ $\langle$*IMPLIES*,*IMPLIES* $\circ_c$ *swap* Ω $\Omega\rangle$) $\circ_c$ x
proof(*cases* p = t)
assume p = t
show *?thesis*
proof(*cases* q = t)
assume q = t
show *?thesis*
proof −
have (*AND* $\circ_c$ $\langle$*IMPLIES*,*IMPLIES* $\circ_c$ *swap* Ω $\Omega\rangle$) $\circ_c$ x =
AND $\circ_c$ $\langle$*IMPLIES*,*IMPLIES* $\circ_c$ *swap* Ω $\Omega\rangle$ $\circ_c$ x
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... = *AND* $\circ_c$ $\langle$*IMPLIES* $\circ_c$ x,*IMPLIES* $\circ_c$ *swap* Ω Ω $\circ_c$ $x\rangle$
using *cfunc-prod-comp comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... = *AND* $\circ_c$ $\langle$*IMPLIES* $\circ_c$ $\langle$t,t$\rangle$, *IMPLIES* $\circ_c$ $\langle$t,t$\rangle\rangle$
using ‹p = t› ‹q = t› *swap-ap x-def* **by** (*typecheck-cfuncs*, *presburger*)
also have ... = *AND* $\circ_c$ $\langle$t, t$\rangle$
using *IMPLIES-true-true-is-true* **by** *presburger*
also have ... = t
by (*simp add*: *AND-true-true-is-true*)
also have ... = *IFF* $\circ_c$ x
by (*simp add*: *IFF-true-true-is-true* ‹p = t› ‹q = t› *x-def*)
finally show *?thesis*

by *simp*
qed
next
assume $q \neq$ t
then have $q =$ f
by (*meson true-false-only-truth-values x-def*)
show *?thesis*
proof −
have $(AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle) \circ_c x =$
$AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle \circ_c x$
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= AND \circ_c \langle IMPLIES \circ_c x, IMPLIES \circ_c swap\ \Omega\ \Omega \circ_c x\rangle$
using *cfunc-prod-comp comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= AND \circ_c \langle IMPLIES \circ_c \langle$t,f$\rangle$, $IMPLIES \circ_c \langle$f,t$\rangle\rangle$
using ‹$p =$ t› ‹$q =$ f› *swap-ap x-def* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= AND \circ_c \langle$f, t$\rangle$
using *IMPLIES-false-true-is-true IMPLIES-true-false-is-false* **by** *presburger*
also have ... = f
by (*simp add*: *AND-false-left-is-false true-func-type*)
also have ... $= IFF \circ_c x$
by (*simp add*: *IFF-true-false-is-false* ‹$p =$ t› ‹$q =$ f› *x-def*)
finally show *?thesis*
by *simp*
qed
qed
next
assume $p \neq$ t
then have $p =$ f
using *true-false-only-truth-values x-def* **by** *blast*
show *?thesis*
proof(*cases* $q =$ t)
assume $q =$ t
show *?thesis*
proof −
have $(AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle) \circ_c x =$
$AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle \circ_c x$
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= AND \circ_c \langle IMPLIES \circ_c x, IMPLIES \circ_c swap\ \Omega\ \Omega \circ_c x\rangle$
using *cfunc-prod-comp comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= AND \circ_c \langle IMPLIES \circ_c \langle$f,t$\rangle$, $IMPLIES \circ_c \langle$t,f$\rangle\rangle$
using ‹$p =$ f› ‹$q =$ t› *swap-ap x-def* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= AND \circ_c \langle$t, f$\rangle$
by (*simp add*: *IMPLIES-false-true-is-true IMPLIES-true-false-is-false*)
also have ... = f
by (*simp add*: *AND-false-right-is-false true-func-type*)
also have ... $= IFF \circ_c x$
by (*simp add*: *IFF-false-true-is-false* ‹$p =$ f› ‹$q =$ t› *x-def*)

finally show *?thesis*
by *simp*
qed
next
assume $q \neq$ t
then have $q =$ f
by (*meson true-false-only-truth-values x-def*)
show *?thesis*
proof −
have $(AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle) \circ_c x =$
$AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle \circ_c x$
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= AND \circ_c \langle IMPLIES \circ_c x, IMPLIES \circ_c swap\ \Omega\ \Omega \circ_c x\rangle$
using *cfunc-prod-comp comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= AND \circ_c \langle IMPLIES \circ_c \langle$f,f$\rangle$, $IMPLIES \circ_c \langle$f,f$\rangle\rangle$
using ‹$p =$ f› ‹$q =$ f› *swap-ap x-def* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= AND \circ_c \langle$t, t$\rangle$
by (*simp add*: *IMPLIES-false-false-is-true*)
also have ... = t
by (*simp add*: *AND-true-true-is-true*)
also have ... $= IFF \circ_c x$
by (*simp add*: *IFF-false-false-is-true* ‹$p =$ f› ‹$q =$ f› *x-def*)
finally show *?thesis*
by *simp*
qed
qed
qed
qed

lemma *IMPLIES-is-OR-NOT-id*:
$IMPLIES = OR \circ_c (NOT \times_f id(\Omega))$
proof(*etcs-rule one-separator*)
fix x
assume *x-type*: $x \in_c \Omega \times_c \Omega$
then obtain u v **where** *x-form*: $u \in_c \Omega \wedge v \in_c \Omega \wedge x = \langle u, v\rangle$
using *cart-prod-decomp* **by** *blast*
show $IMPLIES \circ_c x = (OR \circ_c NOT \times_f id_c\ \Omega) \circ_c x$
proof(*cases* $u =$ t)
assume $u =$ t
show *?thesis*
proof(*cases* $v =$ t)
assume $v =$ t
have $(OR \circ_c NOT \times_f id_c\ \Omega) \circ_c x = OR \circ_c (NOT \times_f id_c\ \Omega) \circ_c x$
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= OR \circ_c \langle NOT \circ_c$ t, $id_c\ \Omega \circ_c$ t$\rangle$
by (*typecheck-cfuncs*, *simp add*: ‹$u =$ t› ‹$v =$ t› *cfunc-cross-prod-comp-cfunc-prod x-form*)
also have ... $= OR \circ_c \langle$f, t$\rangle$

by (*typecheck-cfuncs*, *simp add*: *NOT-true-is-false id-left-unit2*)
also have ... = t
by (*simp add*: *OR-true-right-is-true false-func-type*)
also have ... = *IMPLIES* $\circ_c$ *x*
by (*simp add*: *IMPLIES-true-true-is-true* ‹*u* = t› ‹*v* = t› *x-form*)
finally show *?thesis*
by *simp*
next
assume $v \neq$ t
then have *v* = f
by (*metis true-false-only-truth-values x-form*)
have (*OR* $\circ_c$ *NOT* $\times_f$ id_c Ω) $\circ_c$ *x* = *OR* $\circ_c$ (*NOT* $\times_f$ id_c Ω) $\circ_c$ *x*
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... = *OR* $\circ_c$ ⟨*NOT* $\circ_c$ t, id_c Ω $\circ_c$ f⟩
by (*typecheck-cfuncs*, *simp add*: ‹*u* = t› ‹*v* = f› *cfunc-cross-prod-comp-cfunc-prod x-form*)
also have ... = *OR* $\circ_c$ ⟨f, f⟩
by (*typecheck-cfuncs*, *simp add*: *NOT-true-is-false id-left-unit2*)
also have ... = f
by (*simp add*: *OR-false-false-is-false false-func-type*)
also have ... = *IMPLIES* $\circ_c$ *x*
by (*simp add*: *IMPLIES-true-false-is-false* ‹*u* = t› ‹*v* = f› *x-form*)
finally show *?thesis*
by *simp*
qed
next
assume $u \neq$ t
then have *u* = f
by (*metis true-false-only-truth-values x-form*)
show *?thesis*
proof(*cases v* = t)
assume *v* = t
have (*OR* $\circ_c$ *NOT* $\times_f$ id_c Ω) $\circ_c$ *x* = *OR* $\circ_c$ (*NOT* $\times_f$ id_c Ω) $\circ_c$ *x*
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... = *OR* $\circ_c$ ⟨*NOT* $\circ_c$ f, id_c Ω $\circ_c$ t⟩
by (*typecheck-cfuncs*, *simp add*: ‹*u* = f› ‹*v* = t› *cfunc-cross-prod-comp-cfunc-prod x-form*)
also have ... = *OR* $\circ_c$ ⟨t, t⟩
using *NOT-false-is-true id-left-unit2 true-func-type* **by** *smt*
also have ... = t
by (*simp add*: *OR-true-right-is-true true-func-type*)
also have ... = *IMPLIES* $\circ_c$ *x*
by (*simp add*: *IMPLIES-false-true-is-true* ‹*u* = f› ‹*v* = t› *x-form*)
finally show *?thesis*
by *simp*
next
assume $v \neq$ t
then have *v* = f
by (*metis true-false-only-truth-values x-form*)

have $(OR \circ_c NOT \times_f id_c\ \Omega) \circ_c x = OR \circ_c (NOT \times_f id_c\ \Omega) \circ_c x$
using *comp-associative2 x-type* **by** (*typecheck-cfuncs*, *force*)
also have ... $= OR \circ_c \langle NOT \circ_c$ f, $id_c\ \Omega \circ_c$ f$\rangle$
by (*typecheck-cfuncs*, *simp add*: ‹u = f› ‹v = f› *cfunc-cross-prod-comp-cfunc-prod x-form*)
also have ... $= OR \circ_c \langle$t, f$\rangle$
using *NOT-false-is-true false-func-type id-left-unit2* **by** *presburger*
also have ... = t
by (*simp add*: *OR-true-left-is-true false-func-type*)
also have ... $= IMPLIES \circ_c x$
by (*simp add*: *IMPLIES-false-false-is-true* ‹u = f› ‹v = f› *x-form*)
finally show *?thesis*
by *simp*
qed
qed
qed

lemma *IMPLIES-implies-implies*:
assumes *P-type*[*type-rule*]: $P : X \to \Omega$ **and** *Q-type*[*type-rule*]: $Q : Y \to \Omega$
assumes *X-nonempty*: $\exists\, x.\ x \in_c X$
assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = \mathrm{t} \circ_c \beta_{X \times_c Y}$
shows $P = \mathrm{t} \circ_c \beta_X \Longrightarrow Q = \mathrm{t} \circ_c \beta_Y$
proof −
obtain z **where** *z-type*[*type-rule*]: $z : X \times_c Y \to \mathbf{1} \coprod \mathbf{1} \coprod \mathbf{1}$
and *z-eq*: $P \times_f Q = (\langle$t,t$\rangle$ II $\langle$f,f$\rangle$ II $\langle$f,t$\rangle) \circ_c z$
using *IMPLIES-is-pullback* **unfolding** *is-pullback-def*
by (*auto*, *typecheck-cfuncs*, *metis IMPLIES-true terminal-func-type*)
assume *P-true*: $P = \mathrm{t} \circ_c \beta_X$

have *left-cart-proj* $\Omega\ \Omega \circ_c (P \times_f Q) =$ *left-cart-proj* $\Omega\ \Omega \circ_c (\langle$t,t$\rangle$ II $\langle$f,f$\rangle$ II $\langle$f,t$\rangle)$ $\circ_c z$
using *z-eq* **by** *simp*
then have $P \circ_c$ *left-cart-proj* $X\ Y =$ (*left-cart-proj* $\Omega\ \Omega \circ_c (\langle$t,t$\rangle$ II $\langle$f,f$\rangle$ II $\langle$f,t$\rangle))$ $\circ_c z$
using *Q-type comp-associative2 left-cart-proj-cfunc-cross-prod* **by** (*typecheck-cfuncs*, *force*)
then have $P \circ_c$ *left-cart-proj* $X\ Y$
= ((*left-cart-proj* $\Omega\ \Omega \circ_c \langle$t,t$\rangle$) II (*left-cart-proj* $\Omega\ \Omega \circ_c \langle$f,f$\rangle$) II (*left-cart-proj* $\Omega\ \Omega \circ_c \langle$f,t$\rangle$)) $\circ_c z$
by (*typecheck-cfuncs-prems*, *simp add*: *cfunc-coprod-comp*)
then have $P \circ_c$ *left-cart-proj* $X\ Y =$ (t II f II f) $\circ_c z$
by (*typecheck-cfuncs-prems*, *smt left-cart-proj-cfunc-prod*)

show $Q = \mathrm{t} \circ_c \beta_Y$
proof (*etcs-rule one-separator*)
fix y
assume *y-in-Y*[*type-rule*]: $y \in_c Y$
obtain x **where** *x-in-X*[*type-rule*]: $x \in_c X$
using *X-nonempty* **by** *blast*

have $z \circ_c \langle x,y\rangle = $ *left-coproj* **1** (**1** $\coprod$ **1**)
$\vee$ $z \circ_c \langle x,y\rangle = $ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *left-coproj* **1 1**
$\vee$ $z \circ_c \langle x,y\rangle = $ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *right-coproj* **1 1**
by (*typecheck-cfuncs*, *smt comp-associative2 coprojs-jointly-surj one-unique-element*)
then show $Q \circ_c y = (\mathrm{t} \circ_c \beta_Y) \circ_c y$
proof *safe*
assume $z \circ_c \langle x,y\rangle = $ *left-coproj* **1** (**1** $\coprod$ **1**)
then have $(P \times_f Q) \circ_c \langle x,y\rangle = (\langle \mathrm{t},\mathrm{t}\rangle \amalg \langle \mathrm{f},\mathrm{f}\rangle \amalg \langle \mathrm{f},\mathrm{t}\rangle) \circ_c$ *left-coproj* **1** (**1** $\coprod$ **1**)
by (*typecheck-cfuncs*, *smt comp-associative2 z-eq z-type*)
then have $(P \times_f Q) \circ_c \langle x,y\rangle = \langle \mathrm{t},\mathrm{t}\rangle$
by (*typecheck-cfuncs-prems*, *smt left-coproj-cfunc-coprod*)
then have $Q \circ_c y = \mathrm{t}$
by (*typecheck-cfuncs-prems*, *smt* (*verit*, *best*) *cfunc-cross-prod-comp-cfunc-prod comp-associative2 comp-type id-right-unit2 right-cart-proj-cfunc-prod*)
then show $Q \circ_c y = (\mathrm{t} \circ_c \beta_Y) \circ_c y$
by (*smt* (*verit*, *best*) *comp-associative2 id-right-unit2 terminal-func-comp-elem terminal-func-type true-func-type y-in-Y*)
next
assume $z \circ_c \langle x,y\rangle = $ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *left-coproj* **1 1**
then have $(P \times_f Q) \circ_c \langle x,y\rangle = (\langle \mathrm{t},\mathrm{t}\rangle \amalg \langle \mathrm{f},\mathrm{f}\rangle \amalg \langle \mathrm{f},\mathrm{t}\rangle) \circ_c$ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *left-coproj* **1 1**
by (*typecheck-cfuncs*, *smt comp-associative2 z-eq z-type*)
then have $(P \times_f Q) \circ_c \langle x,y\rangle = (\langle \mathrm{f},\mathrm{f}\rangle \amalg \langle \mathrm{f},\mathrm{t}\rangle) \circ_c$ *left-coproj* **1 1**
by (*typecheck-cfuncs-prems*, *smt right-coproj-cfunc-coprod comp-associative2*)
then have $(P \times_f Q) \circ_c \langle x,y\rangle = \langle \mathrm{f},\mathrm{f}\rangle$
by (*typecheck-cfuncs-prems*, *smt left-coproj-cfunc-coprod*)
then have $P \circ_c x = \mathrm{f}$
by (*typecheck-cfuncs-prems*, *smt* (*verit*, *best*) *cfunc-cross-prod-comp-cfunc-prod comp-associative2 comp-type id-right-unit2 left-cart-proj-cfunc-prod*)
also have $P \circ_c x = \mathrm{t}$
using *P-true* **by** (*typecheck-cfuncs-prems*, *smt* (*z3*) *comp-associative2 id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type x-in-X*)
ultimately have *False*
using *true-false-distinct* **by** *simp*
then show $Q \circ_c y = (\mathrm{t} \circ_c \beta_Y) \circ_c y$
by *simp*
next
assume $z \circ_c \langle x,y\rangle = $ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *right-coproj* **1 1**
then have $(P \times_f Q) \circ_c \langle x,y\rangle = (\langle \mathrm{t},\mathrm{t}\rangle \amalg \langle \mathrm{f},\mathrm{f}\rangle \amalg \langle \mathrm{f},\mathrm{t}\rangle) \circ_c$ *right-coproj* **1** (**1** $\coprod$ **1**) $\circ_c$ *right-coproj* **1 1**
by (*typecheck-cfuncs*, *smt comp-associative2 z-eq z-type*)
then have $(P \times_f Q) \circ_c \langle x,y\rangle = (\langle \mathrm{f},\mathrm{f}\rangle \amalg \langle \mathrm{f},\mathrm{t}\rangle) \circ_c$ *right-coproj* **1 1**
by (*typecheck-cfuncs-prems*, *smt right-coproj-cfunc-coprod comp-associative2*)
then have $(P \times_f Q) \circ_c \langle x,y\rangle = \langle \mathrm{f},\mathrm{t}\rangle$
by (*typecheck-cfuncs-prems*, *smt right-coproj-cfunc-coprod*)
then have $Q \circ_c y = \mathrm{t}$
by (*typecheck-cfuncs-prems*, *smt* (*verit*, *best*) *cfunc-cross-prod-comp-cfunc-prod*

comp-associative2 comp-type id-right-unit2 right-cart-proj-cfunc-prod)
then show $Q \circ_c y = (\mathrm{t} \circ_c \beta_Y) \circ_c y$
by (*typecheck-cfuncs*, *smt* (*z3*) *comp-associative2 id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type*)
qed
qed
qed

lemma *IMPLIES-elim*:
assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = \mathrm{t} \circ_c \beta_{X \times_c Y}$
assumes *P-type*[*type-rule*]: $P : X \to \Omega$ **and** *Q-type*[*type-rule*]: $Q : Y \to \Omega$
assumes *X-nonempty*: $\exists x.\ x \in_c X$
shows $(P = \mathrm{t} \circ_c \beta_X) \Longrightarrow ((Q = \mathrm{t} \circ_c \beta_Y) \Longrightarrow R) \Longrightarrow R$
using *IMPLIES-implies-implies assms* **by** *blast*

lemma *IMPLIES-elim''*:
assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = \mathrm{t}$
assumes *P-type*[*type-rule*]: $P : \mathbf{1} \to \Omega$ **and** *Q-type*[*type-rule*]: $Q : \mathbf{1} \to \Omega$
shows $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
proof −
have *one-nonempty*: $\exists x.\ x \in_c \mathbf{1}$
using *one-unique-element* **by** *blast*
have $(IMPLIES \circ_c (P \times_f Q) = \mathrm{t} \circ_c \beta_{\mathbf{1} \times_c \mathbf{1}})$
by (*typecheck-cfuncs*, *metis IMPLIES-true id-right-unit2 id-type one-unique-element terminal-func-comp terminal-func-type*)
then have $(P = \mathrm{t} \circ_c \beta_{\mathbf{1}}) \Longrightarrow ((Q = \mathrm{t} \circ_c \beta_{\mathbf{1}}) \Longrightarrow R) \Longrightarrow R$
using *one-nonempty* **by** (−, *etcs-erule IMPLIES-elim*, *auto*)
then show $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
by (*typecheck-cfuncs*, *metis id-right-unit2 id-type one-unique-element terminal-func-type*)
qed

lemma *IMPLIES-elim'*:
assumes *IMPLIES-true*: $IMPLIES \circ_c \langle P, Q\rangle = \mathrm{t}$
assumes *P-type*[*type-rule*]: $P : \mathbf{1} \to \Omega$ **and** *Q-type*[*type-rule*]: $Q : \mathbf{1} \to \Omega$
shows $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
using *IMPLIES-true IMPLIES-true-false-is-false Q-type true-false-only-truth-values*
by *force*

lemma *implies-implies-IMPLIES*:
assumes *P-type*[*type-rule*]: $P : \mathbf{1} \to \Omega$ **and** *Q-type*[*type-rule*]: $Q : \mathbf{1} \to \Omega$
shows $(P = \mathrm{t} \Longrightarrow Q = \mathrm{t}) \Longrightarrow IMPLIES \circ_c \langle P, Q\rangle = \mathrm{t}$
by (*typecheck-cfuncs*, *metis IMPLIES-false-is-true-false true-false-only-truth-values*)

14.9 Other Boolean Identities

lemma *AND-OR-distributive*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$

assumes $r \in_c \Omega$
shows $AND \circ_c \langle p,\ OR \circ_c \langle q,r\rangle\rangle = OR \circ_c \langle AND \circ_c \langle p,q\rangle,\ AND \circ_c \langle p,r\rangle\rangle$
by (*metis AND-commutative AND-false-right-is-false AND-true-true-is-true OR-false-false-is-false OR-true-left-is-true OR-true-right-is-true assms true-false-only-truth-values*)

lemma *OR-AND-distributive*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $r \in_c \Omega$
shows $OR \circ_c \langle p,\ AND \circ_c \langle q,r\rangle\rangle = AND \circ_c \langle OR \circ_c \langle p,q\rangle,\ OR \circ_c \langle p,r\rangle\rangle$
by (*smt* (*z3*) *AND-commutative AND-false-right-is-false AND-true-true-is-true OR-commutative OR-false-false-is-false OR-true-right-is-true assms true-false-only-truth-values*)

lemma *OR-AND-absorption*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $OR \circ_c \langle p,\ AND \circ_c \langle p,q\rangle\rangle = p$
by (*metis AND-commutative AND-complementary AND-idempotent NOT-true-is-false OR-false-false-is-false OR-true-left-is-true assms true-false-only-truth-values*)

lemma *AND-OR-absorption*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $AND \circ_c \langle p,\ OR \circ_c \langle p,q\rangle\rangle = p$
by (*metis AND-commutative AND-complementary AND-idempotent NOT-true-is-false OR-AND-absorption OR-commutative assms true-false-only-truth-values*)

lemma *deMorgan-Law1*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $NOT \circ_c OR \circ_c \langle p,q\rangle = AND \circ_c \langle NOT \circ_c p,\ NOT \circ_c q\rangle$
by (*metis AND-OR-absorption AND-complementary AND-true-true-is-true NOT-false-is-true NOT-true-is-false OR-AND-absorption OR-commutative OR-idempotent assms false-func-type true-false-only-truth-values*)

lemma *deMorgan-Law2*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $NOT \circ_c AND \circ_c \langle p,q\rangle = OR \circ_c \langle NOT \circ_c p,\ NOT \circ_c q\rangle$
by (*metis AND-complementary AND-idempotent NOT-false-is-true NOT-true-is-false OR-complementary OR-false-false-is-false OR-idempotent assms true-false-only-truth-values true-func-type*)

end

15 Quantifiers

theory *Quant-Logic*
imports *Pred-Logic Exponential-Objects*

begin

15.1 Universal Quantification

definition *FORALL* :: *cset* ⇒ *cfunc* **where**
FORALL X = (*THE* χ. *is-pullback* **1** **1** (Ω^X) Ω ($\beta_{\mathbf{1}}$) t ((t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$)$^\sharp$) χ)

lemma *FORALL-is-pullback*:
is-pullback **1** **1** (Ω^X) Ω ($\beta_{\mathbf{1}}$) t ((t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$)$^\sharp$) (*FORALL X*)
unfolding *FORALL-def*
using *characteristic-function-exists element-monomorphism*
by (*typecheck-cfuncs*, *simp add*: *the1I2*)

lemma *FORALL-type*[*type-rule*]:
FORALL X : $\Omega^X \to \Omega$
using *FORALL-is-pullback* **unfolding** *is-pullback-def* **by** *auto*

lemma *all-true-implies-FORALL-true*:
assumes *p-type*[*type-rule*]: $p : X \to \Omega$ **and** *all-p-true*: $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c x$
= t
shows *FORALL X* $\circ_c$ ($p \circ_c$ *left-cart-proj X* **1**)$^\sharp$ = t
proof −
have $p \circ_c$ *left-cart-proj X* **1** = t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$
proof (*etcs-rule one-separator*)
fix *x*
assume *x-type*: $x \in_c X \times_c \mathbf{1}$

have ($p \circ_c$ *left-cart-proj X* **1**) $\circ_c x = p \circ_c$ (*left-cart-proj X* **1** $\circ_c x$)
using *x-type p-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... = t
using *x-type all-p-true* **by** (*typecheck-cfuncs*, *auto*)
also have ... = t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$ $\circ_c x$
using *x-type* **by** (*typecheck-cfuncs*, *metis id-right-unit2 id-type one-unique-element*)
also have ... = (t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$) $\circ_c x$
using *x-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
finally show ($p \circ_c$ *left-cart-proj X* **1**) $\circ_c x$ = (t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$) $\circ_c x$**.**
qed
then have ($p \circ_c$ *left-cart-proj X* **1**)$^\sharp$ = (t $\circ_c$ $\beta_{X \times_c \mathbf{1}}$)$^\sharp$
by *simp*
then have *FORALL X* $\circ_c$ ($p \circ_c$ *left-cart-proj X* **1**)$^\sharp$ = t $\circ_c$ $\beta_{\mathbf{1}}$
using *FORALL-is-pullback* **unfolding** *is-pullback-def* **by** *auto*
then show *FORALL X* $\circ_c$ ($p \circ_c$ *left-cart-proj X* **1**)$^\sharp$ = t
using *NOT-false-is-true NOT-is-pullback is-pullback-def* **by** *auto*
qed

lemma *all-true-implies-FORALL-true2*:
assumes *p-type*[*type-rule*]: $p : X \times_c Y \to \Omega$ **and** *all-p-true*: $\bigwedge xy.\ xy \in_c X \times_c$
$Y \Longrightarrow p \circ_c xy$ = t
shows *FORALL X* $\circ_c$ $p^\sharp$ = t $\circ_c$ β_Y

proof −
  have $p = \mathrm{t} \circ_c \beta_{X \times_c Y}$
  proof (*etcs-rule one-separator*)
    fix *xy*
    assume *xy-type*[*type-rule*]: $xy \in_c X \times_c Y$
    then have $p \circ_c xy = \mathrm{t}$
      using *all-p-true* **by** *blast*
    then have $p \circ_c xy = \mathrm{t} \circ_c (\beta_{X \times_c Y} \circ_c xy)$
      by (*typecheck-cfuncs*, *metis id-right-unit2 id-type one-unique-element*)
    then show $p \circ_c xy = (\mathrm{t} \circ_c \beta_{X \times_c Y}) \circ_c xy$
      by (*typecheck-cfuncs*, *smt comp-associative2*)
  qed
  then have $p^\sharp = (\mathrm{t} \circ_c \beta_{X \times_c Y})^\sharp$
    by *blast*
  then have $p^\sharp = (\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}} \circ_c (id\ X \times_f \beta_Y))^\sharp$
    by (*typecheck-cfuncs*, *metis terminal-func-unique*)
  then have $p^\sharp = ((\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}}) \circ_c (id\ X \times_f \beta_Y))^\sharp$
    by (*typecheck-cfuncs*, *smt comp-associative2*)
  then have $p^\sharp = (\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp \circ_c \beta_Y$
    by (*typecheck-cfuncs*, *simp add*: *sharp-comp*)
  then have $FORALL\ X \circ_c p^\sharp = (FORALL\ X \circ_c (\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp) \circ_c \beta_Y$
    by (*typecheck-cfuncs*, *smt comp-associative2*)
  then have $FORALL\ X \circ_c p^\sharp = (\mathrm{t} \circ_c \beta_{\mathbf{1}}) \circ_c \beta_Y$
    using *FORALL-is-pullback* **unfolding** *is-pullback-def* **by** *auto*
  then show $FORALL\ X \circ_c p^\sharp = \mathrm{t} \circ_c \beta_Y$
    by (*metis id-right-unit2 id-type terminal-func-unique true-func-type*)
qed

lemma *all-true-implies-FORALL-true3*:
  assumes *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \to \Omega$ **and** *all-p-true*: $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c \langle x,\ id\ \mathbf{1}\rangle = \mathrm{t}$
  shows $FORALL\ X \circ_c p^\sharp = \mathrm{t}$
proof −
  have $FORALL\ X \circ_c p^\sharp = \mathrm{t} \circ_c \beta_{\mathbf{1}}$
    by (*etcs-rule all-true-implies-FORALL-true2*, *metis all-p-true cart-prod-decomp id-type one-unique-element*)
  then show *?thesis*
    by (*metis id-right-unit2 id-type terminal-func-unique true-func-type*)
qed

lemma *FORALL-true-implies-all-true*:
  assumes *p-type*: $p : X \to \Omega$ **and** *FORALL-p-true*: $FORALL\ X \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ \mathbf{1})^\sharp = \mathrm{t}$
  shows $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c x = \mathrm{t}$
proof (*rule ccontr*)
  fix *x*
  assume *x-type*: $x \in_c X$
  assume $p \circ_c x \neq \mathrm{t}$
  then have $p \circ_c x = \mathrm{f}$

using *comp-type p-type true-false-only-truth-values x-type* **by** *blast*
then have $p \circ_c$ *left-cart-proj* X **1** $\circ_c \langle x, id\ \mathbf{1}\rangle = \mathrm{f}$
using *id-type left-cart-proj-cfunc-prod x-type* **by** *auto*
then have *p-left-proj-false*: $p \circ_c$ *left-cart-proj* X **1** $\circ_c \langle x, id\ \mathbf{1}\rangle = \mathrm{f} \circ_c \beta_{X \times_c \mathbf{1}} \circ_c \langle x, id\ \mathbf{1}\rangle$
using *x-type* **by** (*typecheck-cfuncs*, *metis id-right-unit2 one-unique-element*)

have $\mathrm{t} \circ_c id\ \mathbf{1} = FORALL\ X \circ_c (p \circ_c$ *left-cart-proj* $X\ \mathbf{1})^\sharp$
using *FORALL-p-true id-right-unit2 true-func-type* **by** *auto*
then obtain j **where**
j-type: $j \in_c \mathbf{1}$ **and**
j-id: $\beta_{\mathbf{1}} \circ_c j = id\ \mathbf{1}$ **and**
t-j-eq-p-left-proj: $(\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp \circ_c j = (p \circ_c$ *left-cart-proj* $X\ \mathbf{1})^\sharp$
using *FORALL-is-pullback p-type* **unfolding** *is-pullback-def* **by** (*typecheck-cfuncs*, *blast*)
then have $j = id\ \mathbf{1}$
using *id-type one-unique-element* **by** *blast*
then have $(\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp = (p \circ_c$ *left-cart-proj* $X\ \mathbf{1})^\sharp$
using *id-right-unit2 t-j-eq-p-left-proj p-type* **by** (*typecheck-cfuncs*, *auto*)
then have $\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}} = p \circ_c$ *left-cart-proj* X **1**
using *p-type* **by** (*typecheck-cfuncs*, *metis flat-cancels-sharp*)
then have *p-left-proj-true*: $\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}} \circ_c \langle x, id\ \mathbf{1}\rangle = p \circ_c$ *left-cart-proj* X **1** $\circ_c \langle x, id\ \mathbf{1}\rangle$
using *p-type x-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)

have $\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}} \circ_c \langle x, id\ \mathbf{1}\rangle = \mathrm{f} \circ_c \beta_{X \times_c \mathbf{1}} \circ_c \langle x, id\ \mathbf{1}\rangle$
using *p-left-proj-false p-left-proj-true* **by** *auto*
then have $\mathrm{t} \circ_c id\ \mathbf{1} = \mathrm{f} \circ_c id\ \mathbf{1}$
by (*metis id-type right-cart-proj-cfunc-prod right-cart-proj-type terminal-func-unique x-type*)
then have $\mathrm{t} = \mathrm{f}$
using *true-func-type false-func-type id-right-unit2* **by** *auto*
then show *False*
using *true-false-distinct* **by** *auto*
qed

lemma *FORALL-true-implies-all-true2*:
assumes *p-type*[*type-rule*]: $p : X \times_c Y \to \Omega$ **and** *FORALL-p-true*: $FORALL\ X \circ_c p^\sharp = \mathrm{t} \circ_c \beta_Y$
shows $\bigwedge x\ y.\ x \in_c X \Longrightarrow y \in_c Y \Longrightarrow p \circ_c \langle x, y\rangle = \mathrm{t}$
proof −
have $p^\sharp = (\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp \circ_c \beta_Y$
using *FORALL-is-pullback FORALL-p-true* **unfolding** *is-pullback-def*
by (*typecheck-cfuncs*, *metis terminal-func-unique*)
then have $p^\sharp = ((\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}}) \circ_c (id\ X \times_f \beta_Y))^\sharp$
by (*typecheck-cfuncs*, *simp add*: *sharp-comp*)
then have $p^\sharp = (\mathrm{t} \circ_c \beta_{X \times_c Y})^\sharp$
by (*typecheck-cfuncs-prems*, *smt* (*z3*) *comp-associative2 terminal-func-comp*)
then have $p = \mathrm{t} \circ_c \beta_{X \times_c Y}$

by (*typecheck-cfuncs*, *metis flat-cancels-sharp*)
then have $\bigwedge$ *x y*. $x \in_c X \Longrightarrow y \in_c Y \Longrightarrow p \circ_c \langle x,\ y\rangle = (\mathrm{t} \circ_c \beta_{X \times_c Y}) \circ_c \langle x,\ y\rangle$
by *auto*
then show $\bigwedge$ *x y*. $x \in_c X \Longrightarrow y \in_c Y \Longrightarrow p \circ_c \langle x,\ y\rangle = \mathrm{t}$
proof −
fix *x y*
assume *xy-types*[*type-rule*]: $x \in_c X$ $y \in_c Y$
assume $\bigwedge x\ y.\ x \in_c X \Longrightarrow y \in_c Y \Longrightarrow p \circ_c \langle x,y\rangle = (\mathrm{t} \circ_c \beta_{X \times_c Y}) \circ_c \langle x,y\rangle$
then have $p \circ_c \langle x,y\rangle = (\mathrm{t} \circ_c \beta_{X \times_c Y}) \circ_c \langle x,y\rangle$
using *xy-types* **by** *auto*
then have $p \circ_c \langle x,y\rangle = \mathrm{t} \circ_c (\beta_{X \times_c Y} \circ_c \langle x,y\rangle)$
by (*typecheck-cfuncs*, *smt comp-associative2*)
then show $p \circ_c \langle x,\ y\rangle = \mathrm{t}$
by (*typecheck-cfuncs-prems*, *metis id-right-unit2 id-type one-unique-element*)
qed
qed

lemma *FORALL-true-implies-all-true3*:
assumes *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \rightarrow \Omega$ **and** *FORALL-p-true*: $\mathit{FORALL}\ X \circ_c p^\sharp = \mathrm{t}$
shows $\bigwedge$ *x*. $x \in_c X \Longrightarrow p \circ_c \langle x,\ \mathit{id}\ \mathbf{1}\rangle = \mathrm{t}$
using *FORALL-p-true FORALL-true-implies-all-true2 id-right-unit2 terminal-func-unique*
by (*typecheck-cfuncs*, *auto*)

lemma *FORALL-elim*:
assumes *FORALL-p-true*: $\mathit{FORALL}\ X \circ_c p^\sharp = \mathrm{t}$ **and** *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \rightarrow \Omega$
assumes *x-type*[*type-rule*]: $x \in_c X$
shows $(p \circ_c \langle x,\ \mathit{id}\ \mathbf{1}\rangle = \mathrm{t} \Longrightarrow P) \Longrightarrow P$
using *FORALL-p-true FORALL-true-implies-all-true3 p-type x-type* **by** *blast*

lemma *FORALL-elim'*:
assumes *FORALL-p-true*: $\mathit{FORALL}\ X \circ_c p^\sharp = \mathrm{t}$ **and** *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \rightarrow \Omega$
shows $((\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c \langle x,\ \mathit{id}\ \mathbf{1}\rangle = \mathrm{t}) \Longrightarrow P) \Longrightarrow P$
using *FORALL-p-true FORALL-true-implies-all-true3 p-type* **by** *auto*

15.2 Existential Quantification

definition *EXISTS* :: *cset* ⇒ *cfunc* **where**
$\mathit{EXISTS}\ X = \mathit{NOT} \circ_c \mathit{FORALL}\ X \circ_c \mathit{NOT}^X{}_f$

lemma *EXISTS-type*[*type-rule*]:
$\mathit{EXISTS}\ X : \Omega^X \rightarrow \Omega$
unfolding *EXISTS-def* **by** *typecheck-cfuncs*

lemma *EXISTS-true-implies-exists-true*:
assumes *p-type*: $p : X \rightarrow \Omega$ **and** *EXISTS-p-true*: $\mathit{EXISTS}\ X \circ_c (p \circ_c$ *left-cart-proj*

X $\mathbf{1})^\sharp = \mathrm{t}$
shows $\exists\ x.\ x \in_c X \wedge p \circ_c x = \mathrm{t}$
proof −
have $NOT \circ_c FORALL\ X \circ_c NOT^X{}_f \circ_c (p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp = \mathrm{t}$
using *p-type EXISTS-p-true cfunc-type-def comp-associative comp-type*
unfolding *EXISTS-def*
by (*typecheck-cfuncs*, *auto*)
then have $NOT \circ_c FORALL\ X \circ_c (NOT \circ_c p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp = \mathrm{t}$
using *p-type transpose-of-comp* **by** (*typecheck-cfuncs*, *auto*)
then have $FORALL\ X \circ_c (NOT \circ_c p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp \neq \mathrm{t}$
using *NOT-true-is-false true-false-distinct* **by** *auto*
then have $FORALL\ X \circ_c ((NOT \circ_c p) \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp \neq \mathrm{t}$
using *p-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have $\neg\ (\forall\ x.\ x \in_c X \longrightarrow (NOT \circ_c p) \circ_c x = \mathrm{t})$
using *NOT-type all-true-implies-FORALL-true comp-type p-type* **by** *blast*
then have $\neg\ (\forall\ x.\ x \in_c X \longrightarrow NOT \circ_c (p \circ_c x) = \mathrm{t})$
using *p-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
then have $\neg\ (\forall\ x.\ x \in_c X \longrightarrow p \circ_c x \neq \mathrm{t})$
using *NOT-false-is-true comp-type p-type true-false-only-truth-values* **by** *fast-force*
then show $\exists x.\ x \in_c X \wedge p \circ_c x = \mathrm{t}$
by *blast*
qed

lemma *EXISTS-elim*:
assumes *EXISTS-p-true*: $EXISTS\ X \circ_c (p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp = \mathrm{t}$ **and** *p-type*: $p : X \to \Omega$
shows $(\bigwedge\ x.\ x \in_c X \Longrightarrow p \circ_c x = \mathrm{t} \Longrightarrow Q) \Longrightarrow Q$
using *EXISTS-p-true EXISTS-true-implies-exists-true p-type* **by** *auto*

lemma *exists-true-implies-EXISTS-true*:
assumes *p-type*: $p : X \to \Omega$ **and** *exists-p-true*: $\exists\ x.\ x \in_c X \wedge p \circ_c x = \mathrm{t}$
shows $EXISTS\ X \circ_c (p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp = \mathrm{t}$
proof −
have $\neg\ (\forall\ x.\ x \in_c X \longrightarrow p \circ_c x \neq \mathrm{t})$
using *exists-p-true* **by** *blast*
then have $\neg\ (\forall\ x.\ x \in_c X \longrightarrow NOT \circ_c (p \circ_c x) = \mathrm{t})$
using *NOT-true-is-false true-false-distinct* **by** *auto*
then have $\neg\ (\forall\ x.\ x \in_c X \longrightarrow (NOT \circ_c p) \circ_c x = \mathrm{t})$
using *p-type* **by** (*typecheck-cfuncs*, *metis NOT-true-is-false cfunc-type-def comp-associative exists-p-true true-false-distinct*)
then have $FORALL\ X \circ_c ((NOT \circ_c p) \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp \neq \mathrm{t}$
using *FORALL-true-implies-all-true NOT-type comp-type p-type* **by** *blast*
then have $FORALL\ X \circ_c (NOT \circ_c p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp \neq \mathrm{t}$
using *NOT-type cfunc-type-def comp-associative left-cart-proj-type p-type* **by** *auto*
then have $NOT \circ_c FORALL\ X \circ_c (NOT \circ_c p \circ_c \textit{left-cart-proj}\ X\ \mathbf{1})^\sharp = \mathrm{t}$
using *assms NOT-is-false-implies-true true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)

then have $NOT \circ_c FORALL\ X \circ_c NOT^X{}_f \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ \mathbf{1})^\sharp = \mathrm{t}$
using *assms transpose-of-comp* **by** (*typecheck-cfuncs*, *auto*)
then have $(NOT \circ_c FORALL\ X \circ_c NOT^X{}_f) \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ \mathbf{1})^\sharp = \mathrm{t}$
using *assms cfunc-type-def comp-associative* **by** (*typecheck-cfuncs*,*auto*)
then show $EXISTS\ X \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ \mathbf{1})^\sharp = \mathrm{t}$
by (*simp add*: *EXISTS-def*)
qed

end

16 Natural Number Parity and Halving

theory *Nat-Parity*
imports *Nats Quant-Logic*
begin

16.1 Nth Even Number

definition *nth-even* :: *cfunc* **where**
$nth\text{-}even = (THE\ u.\ u: \mathbb{N}_c \to \mathbb{N}_c \wedge$
$u \circ_c zero = zero \wedge$
$(successor \circ_c successor) \circ_c u = u \circ_c successor)$

lemma *nth-even-def2*:
$nth\text{-}even: \mathbb{N}_c \to \mathbb{N}_c \wedge nth\text{-}even \circ_c zero = zero \wedge (successor \circ_c successor) \circ_c nth\text{-}even = nth\text{-}even \circ_c successor$
unfolding *nth-even-def* **by** (*rule theI'*, *etcs-rule natural-number-object-property2*)

lemma *nth-even-type*[*type-rule*]:
$nth\text{-}even: \mathbb{N}_c \to \mathbb{N}_c$
by (*simp add*: *nth-even-def2*)

lemma *nth-even-zero*:
$nth\text{-}even \circ_c zero = zero$
by (*simp add*: *nth-even-def2*)

lemma *nth-even-successor*:
$nth\text{-}even \circ_c successor = (successor \circ_c successor) \circ_c nth\text{-}even$
by (*simp add*: *nth-even-def2*)

lemma *nth-even-successor2*:
$nth\text{-}even \circ_c successor = successor \circ_c successor \circ_c nth\text{-}even$
using *comp-associative2 nth-even-def2* **by** (*typecheck-cfuncs*, *auto*)

16.2 Nth Odd Number

definition *nth-odd* :: *cfunc* **where**
$nth\text{-}odd = (THE\ u.\ u: \mathbb{N}_c \to \mathbb{N}_c \wedge$
$u \circ_c zero = successor \circ_c zero \wedge$

(*successor* $\circ_c$ *successor*) $\circ_c$ *u* = *u* $\circ_c$ *successor*)

lemma *nth-odd-def2*:
nth-odd: $\mathbb{N}_c \to \mathbb{N}_c$ ∧ *nth-odd* $\circ_c$ *zero* = *successor* $\circ_c$ *zero* ∧ (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-odd* = *nth-odd* $\circ_c$ *successor*
unfolding *nth-odd-def* **by** (*rule theI′*, *etcs-rule natural-number-object-property2*)

lemma *nth-odd-type*[*type-rule*]:
nth-odd: $\mathbb{N}_c \to \mathbb{N}_c$
by (*simp add*: *nth-odd-def2*)

lemma *nth-odd-zero*:
nth-odd $\circ_c$ *zero* = *successor* $\circ_c$ *zero*
by (*simp add*: *nth-odd-def2*)

lemma *nth-odd-successor*:
nth-odd $\circ_c$ *successor* = (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-odd*
by (*simp add*: *nth-odd-def2*)

lemma *nth-odd-successor2*:
nth-odd $\circ_c$ *successor* = *successor* $\circ_c$ *successor* $\circ_c$ *nth-odd*
using *comp-associative2 nth-odd-def2* **by** (*typecheck-cfuncs*, *auto*)

lemma *nth-odd-is-succ-nth-even*:
nth-odd = *successor* $\circ_c$ *nth-even*
proof (*etcs-rule natural-number-object-func-unique*[**where** X=$\mathbb{N}_c$, **where** *f*=*successor* $\circ_c$ *successor*])
show *nth-odd* $\circ_c$ *zero* = (*successor* $\circ_c$ *nth-even*) $\circ_c$ *zero*
proof −
have *nth-odd* $\circ_c$ *zero* = *successor* $\circ_c$ *zero*
by (*simp add*: *nth-odd-zero*)
also have ... = (*successor* $\circ_c$ *nth-even*) $\circ_c$ *zero*
using *comp-associative2 nth-even-def2 successor-type zero-type* **by** *fastforce*
finally show *?thesis*.
qed

show *nth-odd* $\circ_c$ *successor* = (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-odd*
by (*simp add*: *nth-odd-successor*)

show (*successor* $\circ_c$ *nth-even*) $\circ_c$ *successor* = (*successor* $\circ_c$ *successor*) $\circ_c$ *successor* $\circ_c$ *nth-even*
proof −
have (*successor* $\circ_c$ *nth-even*) $\circ_c$ *successor* = *successor* $\circ_c$ *nth-even* $\circ_c$ *successor*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *successor* $\circ_c$ *successor* $\circ_c$ *successor* $\circ_c$ *nth-even*
by (*simp add*: *nth-even-successor2*)
also have ... = (*successor* $\circ_c$ *successor*) $\circ_c$ *successor* $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
finally show *?thesis*.

qed
qed

lemma *succ-nth-odd-is-nth-even-succ*:
successor $\circ_c$ *nth-odd* = *nth-even* $\circ_c$ *successor*
proof (*etcs-rule natural-number-object-func-unique*[**where** *f*=*successor* $\circ_c$ *successor*])
show (*successor* $\circ_c$ *nth-odd*) $\circ_c$ *zero* = (*nth-even* $\circ_c$ *successor*) $\circ_c$ *zero*
by (*simp add*: *nth-even-successor2 nth-odd-is-succ-nth-even*)
show (*successor* $\circ_c$ *nth-odd*) $\circ_c$ *successor* = (*successor* $\circ_c$ *successor*) $\circ_c$ *successor* $\circ_c$ *nth-odd*
by (*metis cfunc-type-def codomain-comp comp-associative nth-odd-def2 successor-type*)
then show (*nth-even* $\circ_c$ *successor*) $\circ_c$ *successor* = (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-even* $\circ_c$ *successor*
using *nth-even-successor2 nth-odd-is-succ-nth-even* **by** *auto*
qed

16.3 Checking if a Number is Even

definition *is-even* :: *cfunc* **where**
is-even = (*THE u. u*: $\mathbb{N}_c \to \Omega \wedge u \circ_c$ *zero* = t $\wedge$ *NOT* $\circ_c$ *u* = *u* $\circ_c$ *successor*)

lemma *is-even-def2*:
is-even : $\mathbb{N}_c \to \Omega \wedge$ *is-even* $\circ_c$ *zero* = t $\wedge$ *NOT* $\circ_c$ *is-even* = *is-even* $\circ_c$ *successor*
unfolding *is-even-def* **by** (*rule theI'*, *etcs-rule natural-number-object-property2*)

lemma *is-even-type*[*type-rule*]:
is-even : $\mathbb{N}_c \to \Omega$
by (*simp add*: *is-even-def2*)

lemma *is-even-zero*:
is-even $\circ_c$ *zero* = t
by (*simp add*: *is-even-def2*)

lemma *is-even-successor*:
is-even $\circ_c$ *successor* = *NOT* $\circ_c$ *is-even*
by (*simp add*: *is-even-def2*)

16.4 Checking if a Number is Odd

definition *is-odd* :: *cfunc* **where**
is-odd = (*THE u. u*: $\mathbb{N}_c \to \Omega \wedge u \circ_c$ *zero* = f $\wedge$ *NOT* $\circ_c$ *u* = *u* $\circ_c$ *successor*)

lemma *is-odd-def2*:
is-odd : $\mathbb{N}_c \to \Omega \wedge$ *is-odd* $\circ_c$ *zero* = f $\wedge$ *NOT* $\circ_c$ *is-odd* = *is-odd* $\circ_c$ *successor*
unfolding *is-odd-def* **by** (*rule theI'*, *etcs-rule natural-number-object-property2*)

lemma *is-odd-type*[*type-rule*]:
is-odd : $\mathbb{N}_c \to \Omega$

by (*simp add*: *is-odd-def2*)

lemma *is-odd-zero*:
is-odd $\circ_c$ *zero* = f
by (*simp add*: *is-odd-def2*)

lemma *is-odd-successor*:
is-odd $\circ_c$ *successor* = *NOT* $\circ_c$ *is-odd*
by (*simp add*: *is-odd-def2*)

lemma *is-even-not-is-odd*:
is-even = *NOT* $\circ_c$ *is-odd*
proof (*typecheck-cfuncs*, *rule natural-number-object-func-unique*[**where** *f*=*NOT*, **where** *X*=Ω], *clarify*)
show *is-even* $\circ_c$ *zero* = (*NOT* $\circ_c$ *is-odd*) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *metis NOT-false-is-true cfunc-type-def comp-associative is-even-def2 is-odd-def2*)

show *is-even* $\circ_c$ *successor* = *NOT* $\circ_c$ *is-even*
by (*simp add*: *is-even-successor*)

show (*NOT* $\circ_c$ *is-odd*) $\circ_c$ *successor* = *NOT* $\circ_c$ *NOT* $\circ_c$ *is-odd*
by (*typecheck-cfuncs*, *simp add*: *cfunc-type-def comp-associative is-odd-def2*)
qed

lemma *is-odd-not-is-even*:
is-odd = *NOT* $\circ_c$ *is-even*
proof (*typecheck-cfuncs*, *rule natural-number-object-func-unique*[**where** *f*=*NOT*, **where** *X*=Ω], *clarify*)
show *is-odd* $\circ_c$ *zero* = (*NOT* $\circ_c$ *is-even*) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *metis NOT-true-is-false cfunc-type-def comp-associative is-even-def2 is-odd-def2*)

show *is-odd* $\circ_c$ *successor* = *NOT* $\circ_c$ *is-odd*
by (*simp add*: *is-odd-successor*)

show (*NOT* $\circ_c$ *is-even*) $\circ_c$ *successor* = *NOT* $\circ_c$ *NOT* $\circ_c$ *is-even*
by (*typecheck-cfuncs*, *simp add*: *cfunc-type-def comp-associative is-even-def2*)
qed

lemma *not-even-and-odd*:
assumes $m \in_c \mathbb{N}_c$
shows ¬(*is-even* $\circ_c$ *m* = t ∧ *is-odd* $\circ_c$ *m* = t)
using *assms NOT-true-is-false NOT-type comp-associative2 is-even-not-is-odd true-false-distinct* **by** (*typecheck-cfuncs*, *fastforce*)

lemma *even-or-odd*:
assumes $n \in_c \mathbb{N}_c$
shows *is-even* $\circ_c$ *n* = t ∨ *is-odd* $\circ_c$ *n* = t

by (*typecheck-cfuncs*, *metis NOT-false-is-true NOT-type comp-associative2 is-even-not-is-odd true-false-only-truth-values assms*)

lemma *is-even-nth-even-true*:
is-even $\circ_c$ *nth-even* = t $\circ_c$ $\beta_{\mathbb{N}_c}$
proof (*etcs-rule natural-number-object-func-unique*[**where** *f*=*id* Ω, **where** *X*=Ω])
show (*is-even* $\circ_c$ *nth-even*) $\circ_c$ *zero* = (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
proof −
have (*is-even* $\circ_c$ *nth-even*) $\circ_c$ *zero* = *is-even* $\circ_c$ *nth-even* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = t
by (*simp add*: *is-even-zero nth-even-zero*)
also have ... = (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *metis comp-associative2 id-right-unit2 terminal-func-comp-elem*)
finally show *?thesis*.
qed

show (*is-even* $\circ_c$ *nth-even*) $\circ_c$ *successor* = id_c Ω $\circ_c$ *is-even* $\circ_c$ *nth-even*
proof −
have (*is-even* $\circ_c$ *nth-even*) $\circ_c$ *successor* = *is-even* $\circ_c$ *nth-even* $\circ_c$ *successor*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *is-even* $\circ_c$ *successor* $\circ_c$ *successor* $\circ_c$ *nth-even*
by (*simp add*: *nth-even-successor2*)
also have ... = ((*is-even* $\circ_c$ *successor*) $\circ_c$ *successor*) $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *smt comp-associative2*)
also have ... = *is-even* $\circ_c$ *nth-even*
using *is-even-def2 is-even-not-is-odd is-odd-def2 is-odd-not-is-even* **by** (*typecheck-cfuncs*, *auto*)
also have ... = *id* Ω $\circ_c$ *is-even* $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)
finally show *?thesis*.
qed

show (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *successor* = id_c Ω $\circ_c$ t $\circ_c$ $\beta_{\mathbb{N}_c}$
by (*typecheck-cfuncs*, *smt comp-associative2 id-left-unit2 terminal-func-comp*)
qed

lemma *is-odd-nth-odd-true*:
is-odd $\circ_c$ *nth-odd* = t $\circ_c$ $\beta_{\mathbb{N}_c}$
proof (*etcs-rule natural-number-object-func-unique*[**where** *f*=*id* Ω, **where** *X*=Ω])
show (*is-odd* $\circ_c$ *nth-odd*) $\circ_c$ *zero* = (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
proof −
have (*is-odd* $\circ_c$ *nth-odd*) $\circ_c$ *zero* = *is-odd* $\circ_c$ *nth-odd* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = t
using *comp-associative2 is-even-not-is-odd is-even-zero is-odd-def2 nth-odd-def2 successor-type zero-type* **by** *auto*
also have ... = (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *metis comp-associative2 is-even-nth-even-true is-even-type*

is-even-zero nth-even-def2)
  finally show *?thesis*.
 qed

 show (*is-odd* $\circ_c$ *nth-odd*) $\circ_c$ *successor* = id_c Ω $\circ_c$ *is-odd* $\circ_c$ *nth-odd*
 proof −
  have (*is-odd* $\circ_c$ *nth-odd*) $\circ_c$ *successor* = *is-odd* $\circ_c$ *nth-odd* $\circ_c$ *successor*
   by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
  also have ... = *is-odd* $\circ_c$ *successor* $\circ_c$ *successor* $\circ_c$ *nth-odd*
   by (*simp add*: *nth-odd-successor2*)
  also have ... = ((*is-odd* $\circ_c$ *successor*) $\circ_c$ *successor*) $\circ_c$ *nth-odd*
   by (*typecheck-cfuncs*, *smt comp-associative2*)
  also have ... = *is-odd* $\circ_c$ *nth-odd*
  using *is-even-def2 is-even-not-is-odd is-odd-def2 is-odd-not-is-even* **by** (*typecheck-cfuncs*, *auto*)
  also have ... = *id* Ω $\circ_c$ *is-odd* $\circ_c$ *nth-odd*
   by (*typecheck-cfuncs*, *simp add*: *id-left-unit2*)
  finally show *?thesis*.
 qed
 show (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *successor* = id_c Ω $\circ_c$ t $\circ_c$ $\beta_{\mathbb{N}_c}$
  by (*typecheck-cfuncs*, *smt comp-associative2 id-left-unit2 terminal-func-comp*)
qed

lemma *is-odd-nth-even-false*:
 is-odd $\circ_c$ *nth-even* = f $\circ_c$ $\beta_{\mathbb{N}_c}$
 by (*smt NOT-true-is-false NOT-type comp-associative2 is-even-def2 is-even-nth-even-true is-odd-not-is-even nth-even-def2 terminal-func-type true-func-type*)

lemma *is-even-nth-odd-false*:
 is-even $\circ_c$ *nth-odd* = f $\circ_c$ $\beta_{\mathbb{N}_c}$
 by (*smt NOT-true-is-false NOT-type comp-associative2 is-odd-def2 is-odd-nth-odd-true is-even-not-is-odd nth-odd-def2 terminal-func-type true-func-type*)

lemma *EXISTS-zero-nth-even*:
 (*EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ *nth-even* $\times_f$ id_c $\mathbb{N}_c$)$^\sharp$) $\circ_c$ *zero* = t
proof −
 have (*EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ *nth-even* $\times_f$ id_c $\mathbb{N}_c$)$^\sharp$) $\circ_c$ *zero*
   = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ *nth-even* $\times_f$ id_c $\mathbb{N}_c$)$^\sharp$ $\circ_c$ *zero*
  by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
 also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ (*nth-even* $\times_f$ id_c $\mathbb{N}_c$) $\circ_c$ (id_c $\mathbb{N}_c$ $\times_f$ *zero*))$^\sharp$
  by (*typecheck-cfuncs*, *simp add*: *comp-associative2 sharp-comp*)
 also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ (*nth-even* $\times_f$ *zero*))$^\sharp$
  by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-cross-prod id-left-unit2 id-right-unit2*)
 also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ ⟨*nth-even* $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**, *zero* $\circ_c$ $\beta_{\mathbb{N}_c \times_c \mathbf{1}}$⟩)$^\sharp$
  by (*typecheck-cfuncs*, *metis cfunc-cross-prod-def cfunc-type-def right-cart-proj-type terminal-func-unique*)

also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even* $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**, (*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**$\rangle$)$^\sharp$
by (*typecheck-cfuncs*, *smt comp-associative2 terminal-func-comp*)
also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ ((*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even*, *zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$) $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**)$^\sharp$
by (*typecheck-cfuncs*, *smt cfunc-prod-comp comp-associative2*)
also have ... = t
proof (*rule exists-true-implies-EXISTS-true*)
show *eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even*,*zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$: $\mathbb{N}_c \rightarrow \Omega$
by *typecheck-cfuncs*
show $\exists$ *x*. *x* $\in_c$ $\mathbb{N}_c$ $\wedge$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even*,*zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$) $\circ_c$ *x* = t
proof (*typecheck-cfuncs*, *intro exI*[**where** *x*=*zero*], *clarify*)
have (*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even*,*zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$) $\circ_c$ *zero*
= *eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even*,*zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$ $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even* $\circ_c$ *zero*, *zero*$\rangle$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-prod-comp comp-associative2 id-right-unit2 terminal-func-comp-elem*)
also have ... = t
using *eq-pred-iff-eq nth-even-zero* **by** (*typecheck-cfuncs*, *blast*)
finally show (*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-even*,*zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$) $\circ_c$ *zero* = t**.**
qed
qed
finally show *?thesis***.**
qed

lemma *not-EXISTS-zero-nth-odd*:
(*EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ *nth-odd* $\times_f$ *id*$_c$ $\mathbb{N}_c$)$^\sharp$) $\circ_c$ *zero* = f
proof –
have (*EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ *nth-odd* $\times_f$ *id*$_c$ $\mathbb{N}_c$)$^\sharp$) $\circ_c$ *zero* = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ *nth-odd* $\times_f$ *id*$_c$ $\mathbb{N}_c$)$^\sharp$ $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ (*nth-odd* $\times_f$ *id*$_c$ $\mathbb{N}_c$) $\circ_c$ (*id*$_c$ $\mathbb{N}_c$ $\times_f$ *zero*))$^\sharp$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 sharp-comp*)
also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ (*nth-odd* $\times_f$ *zero*))$^\sharp$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-cross-prod id-left-unit2 id-right-unit2*)
also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-odd* $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**, *zero* $\circ_c$ $\beta_{\mathbb{N}_c \times_c \mathbf{1}}$$\rangle$)$^\sharp$
by (*typecheck-cfuncs*, *metis cfunc-cross-prod-def cfunc-type-def right-cart-proj-type terminal-func-unique*)
also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ (*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-odd* $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**, (*zero* $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**$\rangle$)$^\sharp$
by (*typecheck-cfuncs*, *smt comp-associative2 terminal-func-comp*)
also have ... = *EXISTS* $\mathbb{N}_c$ $\circ_c$ ((*eq-pred* $\mathbb{N}_c$ $\circ_c$ $\langle$*nth-odd*, *zero* $\circ_c$ $\beta_{\mathbb{N}_c}\rangle$) $\circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**)$^\sharp$
by (*typecheck-cfuncs*, *smt cfunc-prod-comp comp-associative2*)
also have ... = f

proof –
have $\nexists$ *x*. $x \in_c \mathbb{N}_c \wedge$ (*eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd*, *zero* $\circ_c \beta_{\mathbb{N}_c}\rangle) \circ_c x = \mathrm{t}$
proof *clarify*
fix *x*
assume *x-type*[*type-rule*]: $x \in_c \mathbb{N}_c$
assume (*eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd*,*zero* $\circ_c \beta_{\mathbb{N}_c}\rangle) \circ_c x = \mathrm{t}$
then have *eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd*, *zero* $\circ_c \beta_{\mathbb{N}_c}\rangle \circ_c x = \mathrm{t}$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
then have *eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd* $\circ_c$ *x*, *zero* $\circ_c \beta_{\mathbb{N}_c} \circ_c x\rangle = \mathrm{t}$
by (*typecheck-cfuncs-prems*, *auto simp add*: *cfunc-prod-comp comp-associative2*)
then have *eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd* $\circ_c$ *x*, *zero*$\rangle = \mathrm{t}$
by (*typecheck-cfuncs-prems*, *metis cfunc-type-def id-right-unit id-type one-unique-element*)
then have *nth-odd* $\circ_c$ *x* = *zero*
using *eq-pred-iff-eq* **by** (*typecheck-cfuncs-prems*, *blast*)
then show *False*
by (*typecheck-cfuncs-prems*, *smt comp-associative2 comp-type nth-even-def2 nth-odd-is-succ-nth-even successor-type zero-is-not-successor*)
qed
then have *EXISTS* $\mathbb{N}_c \circ_c$ ((*eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd*,*zero* $\circ_c \beta_{\mathbb{N}_c}\rangle) \circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**$)^\sharp \neq \mathrm{t}$
using *EXISTS-true-implies-exists-true* **by** (*typecheck-cfuncs*, *blast*)
then show *EXISTS* $\mathbb{N}_c \circ_c$ ((*eq-pred* $\mathbb{N}_c \circ_c \langle$*nth-odd*,*zero* $\circ_c \beta_{\mathbb{N}_c}\rangle) \circ_c$ *left-cart-proj* $\mathbb{N}_c$ **1**$)^\sharp = \mathrm{f}$
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
qed
finally show *?thesis*.
qed

16.5 Natural Number Halving

definition *halve-with-parity* :: *cfunc* **where**
halve-with-parity = (*THE u*. *u*: $\mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c \wedge$
$u \circ_c$ *zero* = *left-coproj* $\mathbb{N}_c\ \mathbb{N}_c \circ_c$ *zero* $\wedge$
(*right-coproj* $\mathbb{N}_c\ \mathbb{N}_c \amalg$ (*left-coproj* $\mathbb{N}_c\ \mathbb{N}_c \circ_c$ *successor*)) $\circ_c u = u \circ_c$ *successor*)

lemma *halve-with-parity-def2*:
halve-with-parity : $\mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c \wedge$
halve-with-parity $\circ_c$ *zero* = *left-coproj* $\mathbb{N}_c\ \mathbb{N}_c \circ_c$ *zero* $\wedge$
(*right-coproj* $\mathbb{N}_c\ \mathbb{N}_c \amalg$ (*left-coproj* $\mathbb{N}_c\ \mathbb{N}_c \circ_c$ *successor*)) $\circ_c$ *halve-with-parity* = *halve-with-parity* $\circ_c$ *successor*
unfolding *halve-with-parity-def* **by** (*rule theI'*, *etcs-rule natural-number-object-property2*)

lemma *halve-with-parity-type*[*type-rule*]:
halve-with-parity : $\mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c$
by (*simp add*: *halve-with-parity-def2*)

lemma *halve-with-parity-zero*:
halve-with-parity $\circ_c$ *zero* = *left-coproj* $\mathbb{N}_c\ \mathbb{N}_c \circ_c$ *zero*
by (*simp add*: *halve-with-parity-def2*)

lemma *halve-with-parity-successor*:
(*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity* = *halve-with-parity* $\circ_c$ *successor*
by (*simp add*: *halve-with-parity-def2*)

lemma *halve-with-parity-nth-even*:
halve-with-parity $\circ_c$ *nth-even* = *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$
proof (*etcs-rule natural-number-object-func-unique*[**where** *X*=$\mathbb{N}_c$ $\coprod$ $\mathbb{N}_c$, **where** *f*=(*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\amalg$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)])
show (*halve-with-parity* $\circ_c$ *nth-even*) $\circ_c$ *zero* = *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
proof −
have (*halve-with-parity* $\circ_c$ *nth-even*) $\circ_c$ *zero* = *halve-with-parity* $\circ_c$ *nth-even* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *halve-with-parity* $\circ_c$ *zero*
by (*simp add*: *nth-even-zero*)
also have ... = *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
by (*simp add*: *halve-with-parity-zero*)
finally show *?thesis***.**
qed

show (*halve-with-parity* $\circ_c$ *nth-even*) $\circ_c$ *successor* =
((*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\amalg$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity* $\circ_c$ *nth-even*
proof −
have (*halve-with-parity* $\circ_c$ *nth-even*) $\circ_c$ *successor* = *halve-with-parity* $\circ_c$ *nth-even* $\circ_c$ *successor*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *halve-with-parity* $\circ_c$ (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-even*
by (*simp add*: *nth-even-successor*)
also have ... = ((*halve-with-parity* $\circ_c$ *successor*) $\circ_c$ *successor*) $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = (((*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*) $\circ_c$ *successor*) $\circ_c$ *nth-even*
by (*simp add*: *halve-with-parity-def2*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*))
$\circ_c$ (*halve-with-parity* $\circ_c$ *successor*) $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*))
$\circ_c$ ((*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*) $\circ_c$ *nth-even*
by (*simp add*: *halve-with-parity-def2*)
also have ... = ((*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*))
$\circ_c$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)))
$\circ_c$ *halve-with-parity* $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ((*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\amalg$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*))

$\circ_c$ *halve-with-parity* $\circ_c$ *nth-even*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2 left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
finally show *?thesis*.
qed

show *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor* =
(*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) II (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
qed

lemma *halve-with-parity-nth-odd*:
halve-with-parity $\circ_c$ *nth-odd* = *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$
proof (*etcs-rule natural-number-object-func-unique*[**where** $X=\mathbb{N}_c \coprod \mathbb{N}_c$, **where** *f*=(*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) II (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)])
show (*halve-with-parity* $\circ_c$ *nth-odd*) $\circ_c$ *zero* = *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
proof −
have (*halve-with-parity* $\circ_c$ *nth-odd*) $\circ_c$ *zero* = *halve-with-parity* $\circ_c$ *nth-odd* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *halve-with-parity* $\circ_c$ *successor* $\circ_c$ *zero*
by (*simp add*: *nth-odd-def2*)
also have ... = (*halve-with-parity* $\circ_c$ *successor*) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *halve-with-parity*) $\circ_c$ *zero*
by (*simp add*: *halve-with-parity-def2*)
also have ... = *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *halve-with-parity* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
by (*simp add*: *halve-with-parity-def2*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
finally show *?thesis*.
qed

show (*halve-with-parity* $\circ_c$ *nth-odd*) $\circ_c$ *successor* =
(*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) II (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *halve-with-parity* $\circ_c$ *nth-odd*
proof −
have (*halve-with-parity* $\circ_c$ *nth-odd*) $\circ_c$ *successor* = *halve-with-parity* $\circ_c$ *nth-odd* $\circ_c$ *successor*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)

also have ... = *halve-with-parity* $\circ_c$ (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-odd*
by (*simp add*: *nth-odd-successor*)
also have ... = ((*halve-with-parity* $\circ_c$ *successor*) $\circ_c$ *successor*) $\circ_c$ *nth-odd*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ((*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *halve-with-parity*)
$\circ_c$ *successor*) $\circ_c$ *nth-odd*
by (*simp add*: *halve-with-parity-successor*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)
$\circ_c$ (*halve-with-parity* $\circ_c$ *successor*)) $\circ_c$ *nth-odd*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)
$\circ_c$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *halve-with-parity*)) $\circ_c$ *nth-odd*
by (*simp add*: *halve-with-parity-successor*)
also have ... = (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)
$\circ_c$ *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity* $\circ_c$ *nth-odd*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = ((*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) II (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity* $\circ_c$ *nth-odd*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2 left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
finally show *?thesis*.
qed

show *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor* =
(*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) II (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$
by (*typecheck-cfuncs*, *simp add*: *right-coproj-cfunc-coprod*)
qed

lemma *nth-even-nth-odd-halve-with-parity*:
(*nth-even* II *nth-odd*) $\circ_c$ *halve-with-parity* = *id* $\mathbb{N}_c$
proof (*etcs-rule natural-number-object-func-unique*[**where** X=$\mathbb{N}_c$, **where** f=*successor*])
show (*nth-even* II *nth-odd* $\circ_c$ *halve-with-parity*) $\circ_c$ *zero* = id_c $\mathbb{N}_c$ $\circ_c$ *zero*
proof −
have (*nth-even* II *nth-odd* $\circ_c$ *halve-with-parity*) $\circ_c$ *zero* = *nth-even* II *nth-odd* $\circ_c$ *halve-with-parity* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *nth-even* II *nth-odd* $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
by (*simp add*: *halve-with-parity-zero*)
also have ... = (*nth-even* II *nth-odd* $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *nth-even* $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
also have ... = id_c $\mathbb{N}_c$ $\circ_c$ *zero*
using *id-left-unit2 nth-even-def2 zero-type* **by** *auto*
finally show *?thesis*.

qed

show (*nth-even* $\amalg$ *nth-odd* $\circ_c$ *halve-with-parity*) $\circ_c$ *successor* =
successor $\circ_c$ *nth-even* $\amalg$ *nth-odd* $\circ_c$ *halve-with-parity*
proof −
have (*nth-even* $\amalg$ *nth-odd* $\circ_c$ *halve-with-parity*) $\circ_c$ *successor* = *nth-even* $\amalg$ *nth-odd* $\circ_c$ *halve-with-parity* $\circ_c$ *successor*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *nth-even* $\amalg$ *nth-odd* $\circ_c$ *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*) $\circ_c$ *halve-with-parity*
by (*simp add*: *halve-with-parity-successor*)
also have ... = (*nth-even* $\amalg$ *nth-odd* $\circ_c$ *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\amalg$ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = *nth-odd* $\amalg$ (*nth-even* $\circ_c$ *successor*) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2 left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
also have ... = (*successor* $\circ_c$ *nth-even*) $\amalg$ ((*successor* $\circ_c$ *successor*) $\circ_c$ *nth-even*) $\circ_c$ *halve-with-parity*
by (*simp add*: *nth-even-successor nth-odd-is-succ-nth-even*)
also have ... = (*successor* $\circ_c$ *nth-even*) $\amalg$ (*successor* $\circ_c$ *successor* $\circ_c$ *nth-even*) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2*)
also have ... = (*successor* $\circ_c$ *nth-even*) $\amalg$ (*successor* $\circ_c$ *nth-odd*) $\circ_c$ *halve-with-parity*
by (*simp add*: *nth-odd-is-succ-nth-even*)
also have ... = *successor* $\circ_c$ *nth-even* $\amalg$ *nth-odd* $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *cfunc-coprod-comp comp-associative2*)
finally show *?thesis*.
qed
show id_c $\mathbb{N}_c$ $\circ_c$ *successor* = *successor* $\circ_c$ id_c $\mathbb{N}_c$
using *id-left-unit2 id-right-unit2 successor-type* **by** *auto*
qed

lemma *halve-with-parity-nth-even-nth-odd*:
halve-with-parity $\circ_c$ (*nth-even* $\amalg$ *nth-odd*) = *id* ($\mathbb{N}_c$ $\coprod$ $\mathbb{N}_c$)
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp halve-with-parity-nth-even halve-with-parity-nth-odd id-coprod*)

lemma *even-odd-iso*:
isomorphism (*nth-even* $\amalg$ *nth-odd*)
unfolding *isomorphism-def*
proof (*intro exI*[**where** *x=halve-with-parity*], *safe*)
show *domain halve-with-parity* = *codomain* (*nth-even* $\amalg$ *nth-odd*)
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto*)
show *codomain halve-with-parity* = *domain* (*nth-even* $\amalg$ *nth-odd*)
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto*)
show *halve-with-parity* $\circ_c$ *nth-even* $\amalg$ *nth-odd* = id_c (*domain* (*nth-even* $\amalg$ *nth-odd*))
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto simp add*: *halve-with-parity-nth-even-nth-odd*)
show *nth-even* $\amalg$ *nth-odd* $\circ_c$ *halve-with-parity* = id_c (*domain halve-with-parity*)

by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto simp add*: *nth-even-nth-odd-halve-with-parity*)
qed

lemma *halve-with-parity-iso*:
isomorphism halve-with-parity
unfolding *isomorphism-def*
proof (*intro exI*[**where** x=*nth-even* ⨿ *nth-odd*], *safe*)
show *domain* (*nth-even* ⨿ *nth-odd*) = *codomain halve-with-parity*
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto*)
show *codomain* (*nth-even* ⨿ *nth-odd*) = *domain halve-with-parity*
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto*)
show *nth-even* ⨿ *nth-odd* $\circ_c$ *halve-with-parity* = id_c (*domain halve-with-parity*)
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto simp add*: *nth-even-nth-odd-halve-with-parity*)
show *halve-with-parity* $\circ_c$ *nth-even* ⨿ *nth-odd* = id_c (*domain* (*nth-even* ⨿ *nth-odd*))
by (*typecheck-cfuncs*, *unfold cfunc-type-def*, *auto simp add*: *halve-with-parity-nth-even-nth-odd*)
qed

definition *halve* :: *cfunc* **where**
halve = (*id* $\mathbb{N}_c$ ⨿ *id* $\mathbb{N}_c$) $\circ_c$ *halve-with-parity*

lemma *halve-type*[*type-rule*]:
halve : $\mathbb{N}_c \to \mathbb{N}_c$
unfolding *halve-def* **by** *typecheck-cfuncs*

lemma *halve-nth-even*:
halve $\circ_c$ *nth-even* = *id* $\mathbb{N}_c$
unfolding *halve-def* **by** (*typecheck-cfuncs*, *smt comp-associative2 halve-with-parity-nth-even left-coproj-cfunc-coprod*)

lemma *halve-nth-odd*:
halve $\circ_c$ *nth-odd* = *id* $\mathbb{N}_c$
unfolding *halve-def* **by** (*typecheck-cfuncs*, *smt comp-associative2 halve-with-parity-nth-odd right-coproj-cfunc-coprod*)

lemma *is-even-def3*:
is-even = ((t $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (f $\circ_c$ $\beta_{\mathbb{N}_c}$)) $\circ_c$ *halve-with-parity*
proof (*etcs-rule natural-number-object-func-unique*[**where** X=Ω, **where** f=*NOT*])
show *is-even* $\circ_c$ *zero* = ((t $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *zero*
proof −
have ((t $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *zero*
= (t $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative halve-with-parity-zero*)
also have ... = (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod*)
also have ... = t
using *comp-associative2 is-even-def2 is-even-nth-even-true nth-even-def2* **by** (*typecheck-cfuncs*, *force*)
also have ... = *is-even* $\circ_c$ *zero*
by (*simp add*: *is-even-zero*)

finally show *?thesis*
by *simp*
qed

show *is-even* $\circ_c$ *successor* = *NOT* $\circ_c$ *is-even*
by (*simp add*: *is-even-successor*)

show ((t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *successor* =
NOT $\circ_c$ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*
proof −
have ((t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *successor*
= (t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ II (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 halve-with-parity-successor*)
also have ... =
(((t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$)
II
((t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*))
$\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2*)
also have ... = ((f $\circ_c$ $\beta_{\mathbb{N}_c}$) II (t $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
also have ... = ((*NOT* $\circ_c$ t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (*NOT* $\circ_c$ f $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *NOT-false-is-true NOT-true-is-false comp-associative2*)
also have ... = *NOT* $\circ_c$ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) II (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2 terminal-func-unique*)
finally show *?thesis*.
qed
qed

lemma *is-odd-def3*:
is-odd = ((f $\circ_c$ $\beta_{\mathbb{N}_c}$) II (t $\circ_c$ $\beta_{\mathbb{N}_c}$)) $\circ_c$ *halve-with-parity*
proof (*etcs-rule natural-number-object-func-unique*[**where** X=Ω, **where** f=*NOT*])
show *is-odd* $\circ_c$ *zero* = ((f $\circ_c$ $\beta_{\mathbb{N}_c}$) II (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *zero*
proof −
have ((f $\circ_c$ $\beta_{\mathbb{N}_c}$) II (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *zero*
= (f $\circ_c$ $\beta_{\mathbb{N}_c}$) II (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *zero*
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative halve-with-parity-zero*)
also have ... = (f $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *zero*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod*)
also have ... = f
using *comp-associative2 is-odd-nth-even-false is-odd-type is-odd-zero nth-even-def2*
by (*typecheck-cfuncs*, *force*)
also have ... = *is-odd* $\circ_c$ *zero*
by (*simp add*: *is-odd-def2*)
finally show *?thesis*
by *simp*

qed

show *is-odd* $\circ_c$ *successor* = *NOT* $\circ_c$ *is-odd*
by (*simp add*: *is-odd-successor*)

show ((f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *successor* =
NOT $\circ_c$ (f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*
proof −
have ((f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*) $\circ_c$ *successor*
= (f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ (*right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ ⨿ (*left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 halve-with-parity-successor*)
also have ... =
(((f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$)
⨿
((f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *successor*))
$\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2*)
also have ... = ((t $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (f $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 left-coproj-cfunc-coprod right-coproj-cfunc-coprod*)
also have ... = ((*NOT* $\circ_c$ f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (*NOT* $\circ_c$ t $\circ_c$ $\beta_{\mathbb{N}_c}$ $\circ_c$ *successor*)) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *simp add*: *NOT-false-is-true NOT-true-is-false comp-associative2*)
also have ... = *NOT* $\circ_c$ (f $\circ_c$ $\beta_{\mathbb{N}_c}$) ⨿ (t $\circ_c$ $\beta_{\mathbb{N}_c}$) $\circ_c$ *halve-with-parity*
by (*typecheck-cfuncs*, *smt cfunc-coprod-comp comp-associative2 terminal-func-unique*)
finally show *?thesis*.
qed
qed

lemma *nth-even-or-nth-odd*:
assumes $n \in_c \mathbb{N}_c$
shows ($\exists$ *m*. $m \in_c \mathbb{N}_c$ ∧ *nth-even* $\circ_c$ *m* = *n*) ∨ ($\exists$ *m*. $m \in_c \mathbb{N}_c$ ∧ *nth-odd* $\circ_c$ *m* = *n*)
proof −
have ($\exists$ *m*. $m \in_c \mathbb{N}_c$ ∧ *halve-with-parity* $\circ_c$ *n* = *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *m*)
∨ ($\exists$ *m*. $m \in_c \mathbb{N}_c$ ∧ *halve-with-parity* $\circ_c$ *n* = *right-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *m*)
by (*rule coprojs-jointly-surj*, *insert assms*, *typecheck-cfuncs*)
then show *?thesis*
proof
assume $\exists$ *m*. $m \in_c \mathbb{N}_c$ ∧ *halve-with-parity* $\circ_c$ *n* = *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *m*
then obtain *m* **where** *m-type*: $m \in_c \mathbb{N}_c$ **and** *m-def*: *halve-with-parity* $\circ_c$ *n* = *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$ $\circ_c$ *m*
by *auto*
then have ((*nth-even* ⨿ *nth-odd*) $\circ_c$ *halve-with-parity*) $\circ_c$ *n* = ((*nth-even* ⨿ *nth-odd*) $\circ_c$ *left-coproj* $\mathbb{N}_c$ $\mathbb{N}_c$) $\circ_c$ *m*
by (*typecheck-cfuncs*, *smt assms comp-associative2*)
then have *n* = *nth-even* $\circ_c$ *m*
using *assms* **by** (*typecheck-cfuncs-prems*, *smt comp-associative2 halve-with-parity-nth-even*

id-left-unit2 nth-even-nth-odd-halve-with-parity)
  then have $\exists\, m.\ m \in_c \mathbb{N}_c \wedge \textit{nth-even} \circ_c m = n$
    using *m-type* **by** *auto*
  then show *?thesis*
    by *simp*
 next
  assume $\exists\, m.\ m \in_c \mathbb{N}_c \wedge \textit{halve-with-parity} \circ_c n = \textit{right-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c m$
  then obtain *m* **where** *m-type*: $m \in_c \mathbb{N}_c$ **and** *m-def*: $\textit{halve-with-parity} \circ_c n = \textit{right-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c m$
    by *auto*
  then have $((\textit{nth-even} \amalg \textit{nth-odd}) \circ_c \textit{halve-with-parity}) \circ_c n = ((\textit{nth-even} \amalg \textit{nth-odd}) \circ_c \textit{right-coproj}\ \mathbb{N}_c\ \mathbb{N}_c) \circ_c m$
    by (*typecheck-cfuncs*, *smt assms comp-associative2*)
  then have $n = \textit{nth-odd} \circ_c m$
  using *assms* **by** (*typecheck-cfuncs-prems*, *smt comp-associative2 halve-with-parity-nth-odd id-left-unit2 nth-even-nth-odd-halve-with-parity*)
  then show *?thesis*
    using *m-type* **by** *auto*
 qed
qed

lemma *is-even-exists-nth-even*:
 assumes $\textit{is-even} \circ_c n = \mathrm{t}$ **and** *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
 shows $\exists\, m.\ m \in_c \mathbb{N}_c \wedge n = \textit{nth-even} \circ_c m$
proof (*rule ccontr*)
 assume $\nexists\, m.\ m \in_c \mathbb{N}_c \wedge n = \textit{nth-even} \circ_c m$
 then obtain *m* **where** *m-type*[*type-rule*]: $m \in_c \mathbb{N}_c$ **and** *n-def*: $n = \textit{nth-odd} \circ_c m$
  using *n-type nth-even-or-nth-odd* **by** *blast*
 then have $\textit{is-even} \circ_c \textit{nth-odd} \circ_c m = \mathrm{t}$
  using *assms*(*1*) **by** *blast*
 then have $\textit{is-odd} \circ_c \textit{nth-odd} \circ_c m = \mathrm{f}$
  using *NOT-true-is-false NOT-type comp-associative2 is-even-def2 is-odd-not-is-even n-def n-type* **by** *fastforce*
 then have $\mathrm{t} \circ_c \beta_{\mathbb{N}_c} \circ_c m = \mathrm{f}$
  by (*typecheck-cfuncs-prems*, *smt comp-associative2 is-odd-nth-odd-true terminal-func-type true-func-type*)
 then have $\mathrm{t} = \mathrm{f}$
  by (*typecheck-cfuncs-prems*, *metis id-right-unit2 id-type one-unique-element*)
 then show *False*
  using *true-false-distinct* **by** *auto*
qed

lemma *is-odd-exists-nth-odd*:
 assumes $\textit{is-odd} \circ_c n = \mathrm{t}$ **and** *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
 shows $\exists\, m.\ m \in_c \mathbb{N}_c \wedge n = \textit{nth-odd} \circ_c m$
proof (*rule ccontr*)
 assume $\nexists\, m.\ m \in_c \mathbb{N}_c \wedge n = \textit{nth-odd} \circ_c m$
 then obtain *m* **where** *m-type*[*type-rule*]: $m \in_c \mathbb{N}_c$ **and** *n-def*: $n = \textit{nth-even} \circ_c$

m
  using *n-type nth-even-or-nth-odd* **by** *blast*
 then have *is-odd* $\circ_c$ *nth-even* $\circ_c$ $m = \mathrm{t}$
  using *assms(1)* **by** *blast*
 then have *is-even* $\circ_c$ *nth-even* $\circ_c$ $m = \mathrm{f}$
  using *NOT-true-is-false NOT-type comp-associative2 is-even-not-is-odd is-odd-def2 n-def n-type* **by** *fastforce*
 then have $\mathrm{t} \circ_c \beta_{\mathbb{N}_c} \circ_c m = \mathrm{f}$
  by (*typecheck-cfuncs-prems*, *smt comp-associative2 is-even-nth-even-true terminal-func-type true-func-type*)
 then have $\mathrm{t} = \mathrm{f}$
  by (*typecheck-cfuncs-prems*, *metis id-right-unit2 id-type one-unique-element*)
 then show *False*
  using *true-false-distinct* **by** *auto*
qed

end

17 Cardinality and Finiteness

theory *Cardinality*
 imports *Exponential-Objects*
begin

The definitions below correspond to Definition 2.6.1 in Halvorson.

definition *is-finite* :: *cset* ⇒ *bool* **where**
 is-finite X ⟷ (∀ *m*. (*m* : *X* → *X* ∧ *monomorphism m*) ⟶ *isomorphism m*)

definition *is-infinite* :: *cset* ⇒ *bool* **where**
 is-infinite X ⟷ (∃ *m*. *m* : *X* → *X* ∧ *monomorphism m* ∧ ¬*surjective m*)

lemma *either-finite-or-infinite*:
 is-finite X ∨ *is-infinite X*
 using *epi-mon-is-iso is-finite-def is-infinite-def surjective-is-epimorphism* **by** *blast*

The definition below corresponds to Definition 2.6.2 in Halvorson.

definition *is-smaller-than* :: *cset* ⇒ *cset* ⇒ *bool* (**infix** $\leq_c$ *50*) **where**
 $X \leq_c Y$ ⟷ (∃ *m*. *m* : *X* → *Y* ∧ *monomorphism m*)

The purpose of the following lemma is simply to unify the two notations used in the book.

lemma *subobject-iff-smaller-than*:
 $(X \leq_c Y) = (\exists m.\ (X,m) \subseteq_c Y)$
 using *is-smaller-than-def subobject-of-def2* **by** *auto*

lemma *set-card-transitive*:
 assumes $A \leq_c B$
 assumes $B \leq_c C$
 shows $A \leq_c C$

by (*typecheck-cfuncs*, *metis* (*full-types*) *assms cfunc-type-def comp-type composition-of-monic-pair-is-monic is-smaller-than-def*)

lemma *all-emptysets-are-finite*:
assumes *is-empty X*
shows *is-finite X*
by (*metis assms epi-mon-is-iso epimorphism-def3 is-finite-def is-empty-def one-separator*)

lemma *emptyset-is-smallest-set*:
$\emptyset \leq_c X$
using *empty-subset is-smaller-than-def subobject-of-def2* **by** *auto*

lemma *truth-set-is-finite*:
is-finite Ω
unfolding *is-finite-def*
proof(*clarify*)
fix *m*
assume *m-type*[*type-rule*]: $m : \Omega \to \Omega$
assume *m-mono*: *monomorphism m*
have *surjective m*
unfolding *surjective-def*
proof(*clarify*)
fix *y*
assume $y \in_c$ *codomain m*
then have $y \in_c \Omega$
using *cfunc-type-def m-type* **by** *force*
then show $\exists x.\ x \in_c$ *domain m* $\wedge\ m \circ_c x = y$
by (*smt* (*verit*, *del-insts*) *cfunc-type-def codomain-comp domain-comp injective-def m-mono m-type monomorphism-imp-injective true-false-only-truth-values*)
qed
then show *isomorphism m*
by (*simp add*: *epi-mon-is-iso m-mono surjective-is-epimorphism*)
qed

lemma *smaller-than-finite-is-finite*:
assumes $X \leq_c Y$ *is-finite Y*
shows *is-finite X*
unfolding *is-finite-def*
proof(*clarify*)
fix *x*
assume *x-type*: $x : X \to X$
assume *x-mono*: *monomorphism x*

obtain *m* **where** *m-def*: m: $X \to Y \wedge$ *monomorphism m*
using *assms*(*1*) *is-smaller-than-def* **by** *blast*
obtain φ **where** φ*-def*: $\varphi =$ *into-super m* $\circ_c$ $(x \bowtie_f id(Y \setminus (X,m))) \circ_c$ *try-cast m*
by *auto*

have *φ-type*: $\varphi : Y \to Y$
unfolding *φ-def*
using *x-type m-def* **by** (*typecheck-cfuncs*, *blast*)

have *injective*($x \bowtie_f id(Y \setminus (X,m))$)
using *cfunc-bowtieprod-inj id-isomorphism id-type iso-imp-epi-and-monic monomorphism-imp-injective x-mono x-type* **by** *blast*
then have *mono1*: *monomorphism*($x \bowtie_f id(Y \setminus (X,m))$)
using *injective-imp-monomorphism* **by** *auto*
have *mono2*: *monomorphism*(*try-cast m*)
using *m-def try-cast-mono* **by** *blast*
have *mono3*: *monomorphism*($(x \bowtie_f id(Y \setminus (X,m))) \circ_c$ *try-cast m*)
using *cfunc-type-def composition-of-monic-pair-is-monic m-def mono1 mono2 x-type* **by** (*typecheck-cfuncs*, *auto*)
then have *φ-mono*: *monomorphism* φ
unfolding *φ-def*
using *cfunc-type-def composition-of-monic-pair-is-monic*
into-super-mono m-def mono3 x-type **by** (*typecheck-cfuncs*,*auto*)
then have *isomorphism* φ
using *φ-def φ-type assms*(*2*) *is-finite-def* **by** *blast*
have *iso-x-bowtie-id*: *isomorphism*($x \bowtie_f id(Y \setminus (X,m))$)
by (*typecheck-cfuncs*, *smt* ‹*isomorphism* φ› *φ-def comp-associative2 id-left-unit2 into-super-iso into-super-try-cast into-super-type isomorphism-sandwich m-def try-cast-type x-type*)
have *left-coproj* X $(Y \setminus (X,m)) \circ_c x = (x \bowtie_f id(Y \setminus (X,m))) \circ_c$ *left-coproj* X $(Y \setminus (X,m))$
using *x-type*
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-bowtie-prod*)
have *epimorphism*($x \bowtie_f id(Y \setminus (X,m))$)
using *iso-imp-epi-and-monic iso-x-bowtie-id* **by** *blast*
then have *surjective*($x \bowtie_f id(Y \setminus (X,m))$)
using *epi-is-surj x-type* **by** (*typecheck-cfuncs*, *blast*)
then have *epimorphism x*
using *x-type cfunc-bowtieprod-surj-converse id-type surjective-is-epimorphism* **by** *blast*
then show *isomorphism x*
by (*simp add*: *epi-mon-is-iso x-mono*)
qed

lemma *larger-than-infinite-is-infinite*:
assumes $X \leq_c Y$ *is-infinite X*
shows *is-infinite Y*
using *assms either-finite-or-infinite epi-is-surj is-finite-def is-infinite-def*
iso-imp-epi-and-monic smaller-than-finite-is-finite **by** *blast*

lemma *iso-pres-finite*:
assumes $X \cong Y$
assumes *is-finite X*
shows *is-finite Y*

using *assms is-isomorphic-def is-smaller-than-def iso-imp-epi-and-monic isomorphic-is-symmetric smaller-than-finite-is-finite* **by** *blast*

lemma *not-finite-and-infinite*:
$\neg$(*is-finite X* $\wedge$ *is-infinite X*)
using *epi-is-surj is-finite-def is-infinite-def iso-imp-epi-and-monic* **by** *blast*

lemma *iso-pres-infinite*:
assumes $X \cong Y$
assumes *is-infinite X*
shows *is-infinite Y*
using *assms either-finite-or-infinite not-finite-and-infinite iso-pres-finite isomorphic-is-symmetric* **by** *blast*

lemma *size-2-sets*:
$(X \cong \Omega) = (\exists\ x1.\ \exists\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2 \wedge (\forall x.\ x \in_c X \longrightarrow x = x1 \vee x = x2))$
proof
assume $X \cong \Omega$
then obtain φ **where** *φ-type*[*type-rule*]: $\varphi : X \to \Omega$ **and** *φ-iso*: *isomorphism* φ
using *is-isomorphic-def* **by** *blast*
obtain *x1 x2* **where** *x1-type*[*type-rule*]: $x1 \in_c X$ **and** *x1-def*: $\varphi \circ_c x1 = \mathrm{t}$ **and**
x2-type[*type-rule*]: $x2 \in_c X$ **and** *x2-def*: $\varphi \circ_c x2 = \mathrm{f}$ **and**
distinct: $x1 \neq x2$
by (*typecheck-cfuncs*, *smt* (*z3*) *φ-iso cfunc-type-def comp-associative comp-type id-left-unit2 isomorphism-def true-false-distinct*)
then show $\exists x1\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2 \wedge (\forall x.\ x \in_c X \longrightarrow x = x1 \vee x = x2)$
by (*smt* (*verit*, *best*) *φ-iso φ-type cfunc-type-def comp-associative2 comp-type id-left-unit2 isomorphism-def true-false-only-truth-values*)
next
assume *exactly-two*: $\exists x1\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2 \wedge (\forall x.\ x \in_c X \longrightarrow x = x1 \vee x = x2)$
then obtain *x1 x2* **where** *x1-type*[*type-rule*]: $x1 \in_c X$ **and** *x2-type*[*type-rule*]: $x2 \in_c X$ **and** *distinct*: $x1 \neq x2$
by *force*
have *iso-type*: $((x1 \amalg x2) \circ_c \textit{case-bool}) : \Omega \to X$
by *typecheck-cfuncs*
have *surj*: *surjective* $((x1 \amalg x2) \circ_c \textit{case-bool})$
by (*typecheck-cfuncs*, *smt* (*verit*, *best*) *exactly-two cfunc-type-def coprod-case-bool-false coprod-case-bool-true distinct false-func-type surjective-def true-func-type*)
have *inj*: *injective* $((x1 \amalg x2) \circ_c \textit{case-bool})$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *distinct case-bool-true-and-false comp-associative2*
coprod-case-bool-false injective-def2 left-coproj-cfunc-coprod true-false-only-truth-values)
then have *isomorphism* $((x1 \amalg x2) \circ_c \textit{case-bool})$
by (*meson epi-mon-is-iso injective-imp-monomorphism singletonI surj surjective-is-epimorphism*)
then show $X \cong \Omega$

using *is-isomorphic-def iso-type isomorphic-is-symmetric* **by** *blast*
qed

lemma *size-2plus-sets*:
$(\Omega \leq_c X) = (\exists\ x1.\ \exists\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2)$
proof *standard*
show $\Omega \leq_c X \Longrightarrow \exists x1\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2$
by (*meson comp-type false-func-type is-smaller-than-def monomorphism-def3 true-false-distinct true-func-type*)
next
assume $\exists x1\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2$
then obtain *x1 x2* **where** *x1-type*[*type-rule*]: $x1 \in_c X$ **and**
x2-type[*type-rule*]: $x2 \in_c X$ **and**
distinct: $x1 \neq x2$
by *blast*
have *mono-type*: $((x1\ \amalg\ x2) \circ_c\ case\text{-}bool) : \Omega \to X$
by *typecheck-cfuncs*
have *inj*: *injective* $((x1\ \amalg\ x2) \circ_c\ case\text{-}bool)$
by (*typecheck-cfuncs*, *smt* (*verit*, *ccfv-SIG*) *distinct case-bool-true-and-false comp-associative2*
coprod-case-bool-false injective-def2 left-coproj-cfunc-coprod true-false-only-truth-values)

then show $\Omega \leq_c X$
using *injective-imp-monomorphism is-smaller-than-def mono-type* **by** *blast*
qed

lemma *not-init-not-term*:
$(\neg(initial\text{-}object\ X) \wedge \neg(terminal\text{-}object\ X)) = (\exists\ x1.\ \exists\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2)$
by (*metis is-empty-def initial-iso-empty iso-empty-initial iso-to1-is-term no-el-iff-iso-empty single-elem-iso-one terminal-object-def*)

lemma *sets-size-3-plus*:
$(\neg(initial\text{-}object\ X) \wedge \neg(terminal\text{-}object\ X) \wedge \neg(X \cong \Omega)) = (\exists\ x1.\ \exists\ x2.\ \exists\ x3.\ x1 \in_c X \wedge x2 \in_c X \wedge x3 \in_c X \wedge x1 \neq x2 \wedge x2 \neq x3 \wedge x1 \neq x3)$
by (*metis not-init-not-term size-2-sets*)

The next two lemmas below correspond to Proposition 2.6.3 in Halvorson.

lemma *smaller-than-coproduct1*:
$X \leq_c X \coprod Y$
using *is-smaller-than-def left-coproj-are-monomorphisms left-proj-type* **by** *blast*

lemma *smaller-than-coproduct2*:
$X \leq_c Y \coprod X$
using *is-smaller-than-def right-coproj-are-monomorphisms right-proj-type* **by** *blast*

The next two lemmas below correspond to Proposition 2.6.4 in Halvorson.

lemma *smaller-than-product1*:
assumes *nonempty Y*
shows $X \leq_c X \times_c Y$
unfolding *is-smaller-than-def*
proof −
obtain *y* **where** *y-type*: $y \in_c Y$
using *assms nonempty-def* **by** *blast*
have *map-type*: $\langle id(X), y \circ_c \beta_X \rangle : X \to X \times_c Y$
using *y-type cfunc-prod-type cfunc-type-def codomain-comp domain-comp id-type terminal-func-type* **by** *auto*
have *mono*: $monomorphism(\langle id\ X,\ y \circ_c \beta_X \rangle)$
using *map-type*
proof (*unfold monomorphism-def3*, *clarify*)
fix *g h A*
assume *g-h-types*: $g : A \to X\ h : A \to X$

assume $\langle id_c\ X, y \circ_c \beta_X \rangle \circ_c g = \langle id_c\ X, y \circ_c \beta_X \rangle \circ_c h$
then have $\langle id_c\ X \circ_c g,\ y \circ_c \beta_X \circ_c g \rangle = \langle id_c\ X \circ_c h,\ y \circ_c \beta_X \circ_c h \rangle$
using *y-type g-h-types* **by** (*typecheck-cfuncs*, *smt cfunc-prod-comp comp-associative2 comp-type*)
then have $\langle g,\ y \circ_c \beta_A \rangle = \langle h,\ y \circ_c \beta_A \rangle$
using *y-type g-h-types id-left-unit2 terminal-func-comp* **by** (*typecheck-cfuncs*, *auto*)
then show $g = h$
using *g-h-types y-type*
by (*metis* (*full-types*) *comp-type left-cart-proj-cfunc-prod terminal-func-type*)
qed
show $\exists m.\ m : X \to X \times_c Y \wedge monomorphism\ m$
using *mono map-type* **by** *auto*
qed

lemma *smaller-than-product2*:
assumes *nonempty Y*
shows $X \leq_c Y \times_c X$
unfolding *is-smaller-than-def*
proof −
have $X \leq_c X \times_c Y$
by (*simp add*: *assms smaller-than-product1*)
then obtain *m* **where** *m-def*: $m : X \to X \times_c Y \wedge monomorphism\ m$
using *is-smaller-than-def* **by** *blast*
obtain *i* **where** $i : (X \times_c Y) \to (Y \times_c X) \wedge isomorphism\ i$
using *is-isomorphic-def product-commutes* **by** *blast*
then have $i \circ_c m : X \to (Y \times_c X) \wedge monomorphism(i \circ_c m)$
using *cfunc-type-def comp-type composition-of-monic-pair-is-monic iso-imp-epi-and-monic m-def* **by** *auto*
then show $\exists m.\ m : X \to Y \times_c X \wedge monomorphism\ m$
by *blast*
qed

lemma *coprod-leq-product*:
  assumes *X-not-init*: $\neg(initial\text{-}object(X))$
  assumes *Y-not-init*: $\neg(initial\text{-}object(Y))$
  assumes *X-not-term*: $\neg(terminal\text{-}object(X))$
  assumes *Y-not-term*: $\neg(terminal\text{-}object(Y))$
  shows $X \coprod Y \leq_c X \times_c Y$
proof −
  obtain *x1 x2* **where** *x1x2-def*[*type-rule*]: $(x1 \in_c X)$ $(x2 \in_c X)$ $(x1 \neq x2)$
   using *is-empty-def X-not-init X-not-term iso-empty-initial iso-to1-is-term no-el-iff-iso-empty single-elem-iso-one* **by** *blast*
  obtain *y1 y2* **where** *y1y2-def*[*type-rule*]: $(y1 \in_c Y)$ $(y2 \in_c Y)$ $(y1 \neq y2)$
   using *is-empty-def Y-not-init Y-not-term iso-empty-initial iso-to1-is-term no-el-iff-iso-empty single-elem-iso-one* **by** *blast*
  then have *y1-mono*[*type-rule*]: *monomorphism*(*y1*)
    using *element-monomorphism* **by** *blast*
  obtain *m* **where** *m-def*: $m = \langle id(X), y1 \circ_c \beta_X\rangle \amalg ((\langle x2, y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)}, y1^c\rangle) \circ_c$ *try-cast y1*)
    by *simp*
  have *type1*: $\langle id(X), y1 \circ_c \beta_X\rangle : X \to (X \times_c Y)$
    by (*meson cfunc-prod-type comp-type id-type terminal-func-type y1y2-def*)
  have *trycast-y1-type*: *try-cast y1* $: Y \to \mathbf{1} \coprod (Y \setminus (\mathbf{1},y1))$
    by (*meson element-monomorphism try-cast-type y1y2-def*)
  have *y1'-type*[*type-rule*]: $y1^c : Y \setminus (\mathbf{1},y1) \to Y$
   using *complement-morphism-type one-terminal-object terminal-el-monomorphism y1y2-def* **by** *blast*
  have *type4*: $\langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)}, y1^c\rangle : Y \setminus (\mathbf{1},y1) \to (X \times_c Y)$
    using *cfunc-prod-type comp-type terminal-func-type x1x2-def y1'-type* **by** *blast*
  have *type5*: $\langle x2, y2\rangle \in_c (X \times_c Y)$
    by (*simp add*: *cfunc-prod-type x1x2-def y1y2-def*)
  then have *type6*: $\langle x2, y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)}, y1^c\rangle :(\mathbf{1} \coprod (Y \setminus (\mathbf{1},y1))) \to (X \times_c Y)$
    using *cfunc-coprod-type type4* **by** *blast*
  then have *type7*: $((\langle x2, y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)}, y1^c\rangle) \circ_c$ *try-cast y1*) $: Y \to (X \times_c Y)$
    using *comp-type trycast-y1-type* **by** *blast*
  then have *m-type*: $m : X \coprod Y \to (X \times_c Y)$
    by (*simp add*: *cfunc-coprod-type m-def type1*)

  have *relative*: $\bigwedge y.\ y \in_c Y \Longrightarrow (y \in_Y (\mathbf{1}, y1)) = (y = y1)$
  proof(*safe*)
    fix *y*
    assume *y-type*: $y \in_c Y$
    show $y \in_Y (\mathbf{1}, y1) \Longrightarrow y = y1$
     by (*metis cfunc-type-def factors-through-def id-right-unit2 id-type one-unique-element relative-member-def2*)
  next
    show $y1 \in_c Y \Longrightarrow y1 \in_Y (\mathbf{1}, y1)$
     by (*metis cfunc-type-def factors-through-def id-right-unit2 id-type relative-member-def2 y1-mono*)

qed

have *injective*(m)
unfolding *injective-def*
proof(*clarify*)
fix a b
assume $a \in_c$ *domain* m $b \in_c$ *domain* m
then have *a-type*[*type-rule*]: $a \in_c X \coprod Y$ **and** *b-type*[*type-rule*]: $b \in_c X \coprod Y$
using *m-type* **unfolding** *cfunc-type-def* **by** *auto*
assume *eqs*: $m \circ_c a = m \circ_c b$

have *m-leftproj-l-equals*: $\bigwedge l.\ l \in_c X \Longrightarrow m \circ_c$ *left-coproj* X $Y \circ_c l = \langle l,\ y1 \rangle$
proof−
fix l
assume *l-type*: $l \in_c X$
have $m \circ_c$ *left-coproj* X $Y \circ_c l = (\langle id(X),\ y1 \circ_c \beta_X \rangle \amalg ((\langle x2,\ y2 \rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1}, y1)},\ y1^c \rangle) \circ_c$ *try-cast* $y1)) \circ_c$ *left-coproj* X $Y \circ_c l$
by (*simp add*: *m-def*)
also have ... $= ((\langle id(X),\ y1 \circ_c \beta_X \rangle \amalg ((\langle x2,\ y2 \rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1}, y1)},\ y1^c \rangle) \circ_c$ *try-cast* $y1) \circ_c$ *left-coproj* X $Y) \circ_c l$
using *comp-associative2 l-type* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle id(X),\ y1 \circ_c \beta_X \rangle \circ_c l$
by (*typecheck-cfuncs*, *simp add*: *left-coproj-cfunc-coprod*)
also have ... $= \langle id(X) \circ_c l\ ,\ (y1 \circ_c \beta_X) \circ_c l \rangle$
using *l-type cfunc-prod-comp* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= \langle l\ ,\ y1 \circ_c \beta_X \circ_c l \rangle$
using *l-type comp-associative2 id-left-unit2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= \langle l\ ,\ y1 \rangle$
using *l-type* **by** (*typecheck-cfuncs*,*metis id-right-unit2 id-type one-unique-element*)
finally show $m \circ_c$ *left-coproj* X $Y \circ_c l = \langle l, y1 \rangle$**.**
qed

have *m-rightproj-y1-equals*: $m \circ_c$ *right-coproj* X $Y \circ_c y1 = \langle x2,\ y2 \rangle$
proof −
have $m \circ_c$ *right-coproj* X $Y \circ_c y1 = (m \circ_c$ *right-coproj* X $Y) \circ_c y1$
using *comp-associative2 m-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= ((\langle x2,\ y2 \rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1}, y1)},\ y1^c \rangle) \circ_c$ *try-cast* $y1) \circ_c y1$
using *m-def right-coproj-cfunc-coprod type1* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\langle x2,\ y2 \rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1}, y1)},\ y1^c \rangle) \circ_c$ *try-cast* $y1 \circ_c y1$
using *comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\langle x2,\ y2 \rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1}, y1)},\ y1^c \rangle) \circ_c$ *left-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1}, y1))$
using *try-cast-m-m y1-mono y1y2-def*(*1*) **by** *auto*
also have ... $= \langle x2,\ y2 \rangle$
using *left-coproj-cfunc-coprod type4 type5* **by** *blast*
finally show *?thesis***.**
qed

have *m-rightproj-not-y1-equals*: $\bigwedge r.\ r\ \in_c\ Y \wedge r \neq y1 \Longrightarrow$
$\exists k.\ k \in_c\ Y \setminus (\mathbf{1},y1) \wedge$ *try-cast y1* $\circ_c$ r = *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1},y1)) \circ_c$ $k \wedge$
$m \circ_c$ *right-coproj* $X\ Y \circ_c r = \langle x1,\ y1^c \circ_c k\rangle$
proof *clarify*
fix r
assume *r-type*: $r \in_c\ Y$
assume *r-not-y1*: $r \neq y1$
then obtain k **where** *k-def*: $k \in_c\ Y \setminus (\mathbf{1},y1) \wedge$ *try-cast y1* $\circ_c r$ = *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1},y1)) \circ_c k$
using *r-type relative try-cast-not-in-X y1-mono y1y2-def(1)* **by** *blast*
have *m-rightproj-l-equals*: $m \circ_c$ *right-coproj* $X\ Y \circ_c r = \langle x1,\ y1^c \circ_c k\rangle$

proof –
have $m \circ_c$ *right-coproj* $X\ Y \circ_c r = (m \circ_c$ *right-coproj* $X\ Y) \circ_c r$
using *r-type comp-associative2 m-type* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= ((\langle x2,\ y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)},\ y1^c\rangle) \circ_c$ *try-cast y1*$) \circ_c r$
using *m-def right-coproj-cfunc-coprod type1* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\langle x2,\ y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)},\ y1^c\rangle) \circ_c$ (*try-cast y1* $\circ_c r)$
using *r-type comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= (\langle x2,\ y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)},\ y1^c\rangle) \circ_c$ (*right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1},y1)) \circ_c k)$
using *k-def* **by** *auto*
also have ... $= ((\langle x2,\ y2\rangle \amalg \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)},\ y1^c\rangle) \circ_c$ *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1},y1))) \circ_c k$
using *comp-associative2 k-def* **by** (*typecheck-cfuncs*, *blast*)
also have ... $= \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)},\ y1^c\rangle \circ_c k$
using *right-coproj-cfunc-coprod type4 type5* **by** *auto*
also have ... $= \langle x1 \circ_c \beta_{Y \setminus (\mathbf{1},y1)} \circ_c k,\ y1^c \circ_c k\ \rangle$
using *cfunc-prod-comp comp-associative2 k-def* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= \langle x1,\ y1^c \circ_c k\rangle$
by (*metis id-right-unit2 id-type k-def one-unique-element terminal-func-comp terminal-func-type x1x2-def(1)*)
finally show *?thesis*.
qed
then show $\exists k.\ k \in_c\ Y \setminus (\mathbf{1},\ y1) \wedge$
try-cast y1 $\circ_c r$ = *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1},\ y1)) \circ_c k \wedge$
$m \circ_c$ *right-coproj* $X\ Y \circ_c r = \langle x1, y1^c \circ_c k\rangle$
using *k-def* **by** *blast*
qed

show $a = b$
proof(*cases* $\exists x.\ a$ = *left-coproj* $X\ Y \circ_c x\ \wedge x \in_c X$)
assume $\exists x.\ a$ = *left-coproj* $X\ Y \circ_c x\ \wedge x \in_c X$

then obtain *x* **where** *x-def*: $a = left\text{-}coproj\ X\ Y \circ_c x \ \wedge x \in_c X$
  by *auto*
then have *m-proj-a*: $m \circ_c left\text{-}coproj\ X\ Y \circ_c x = \langle x, y1 \rangle$
  using *m-leftproj-l-equals* **by** (*simp add*: *x-def*)
show $a = b$
proof(*cases* $\exists c.\ b = left\text{-}coproj\ X\ Y \circ_c c \ \wedge c \in_c X$)
  assume $\exists c.\ b = left\text{-}coproj\ X\ Y \circ_c c \wedge c \in_c X$
  then obtain *c* **where** *c-def*: $b = left\text{-}coproj\ X\ Y \circ_c c \ \wedge c \in_c X$
    by *auto*
  then have $m \circ_c left\text{-}coproj\ X\ Y \circ_c c = \langle c, y1 \rangle$
    by (*simp add*: *m-leftproj-l-equals*)
  then show *?thesis*
    using *c-def element-pair-eq eqs m-proj-a x-def y1y2-def*(*1*) **by** *auto*
next
  assume $\nexists c.\ b = left\text{-}coproj\ X\ Y \circ_c c \wedge c \in_c X$
  then obtain *c* **where** *c-def*: $b = right\text{-}coproj\ X\ Y \circ_c c \ \wedge c \in_c Y$
    using *b-type coprojs-jointly-surj* **by** *blast*
  show $a = b$
  proof(*cases* $c = y1$)
    assume $c = y1$
    have *m-rightproj-l-equals*: $m \circ_c right\text{-}coproj\ X\ Y \circ_c c = \langle x2, y2 \rangle$
      by (*simp add*: ‹$c = y1$› *m-rightproj-y1-equals*)
    then show *?thesis*
      using ‹$c = y1$› *c-def cart-prod-eq2 eqs m-proj-a x1x2-def*(*2*) *x-def y1y2-def*(*2*) *y1y2-def*(*3*) **by** *auto*
  next
    assume $c \neq y1$
    then obtain *k* **where** *k-def*: $m \circ_c right\text{-}coproj\ X\ Y \circ_c c = \langle x1, y1^c \circ_c k \rangle$
      using *c-def m-rightproj-not-y1-equals* **by** *blast*
    then have $\langle x, y1 \rangle = \langle x1, y1^c \circ_c k \rangle$
      using *c-def eqs m-proj-a x-def* **by** *auto*
    then have $(x = x1) \wedge (y1 = y1^c \circ_c k)$
      by (*smt* ‹$c \neq y1$› *c-def cfunc-type-def comp-associative comp-type element-pair-eq k-def m-rightproj-not-y1-equals monomorphism-def3 try-cast-m-m′ try-cast-mono trycast-y1-type x1x2-def*(*1*) *x-def y1′-type y1-mono y1y2-def*(*1*))
    then have *False*
      by (*smt* ‹$c \neq y1$› *c-def comp-type complement-disjoint element-pair-eq id-right-unit2 id-type k-def m-rightproj-not-y1-equals x-def y1′-type y1-mono y1y2-def*(*1*))
    then show *?thesis* **by** *auto*
  qed
qed
next
  assume $\nexists x.\ a = left\text{-}coproj\ X\ Y \circ_c x \wedge x \in_c X$
  then obtain *y* **where** *y-def*: $a = right\text{-}coproj\ X\ Y \circ_c y \wedge y \in_c Y$
    using *a-type coprojs-jointly-surj* **by** *blast*
  show $a = b$
  proof(*cases* $y = y1$)
    assume $y = y1$
    then have *m-rightproj-y-equals*: $m \circ_c right\text{-}coproj\ X\ Y \circ_c y = \langle x2, y2 \rangle$

using *m-rightproj-y1-equals* **by** *blast*
then have $m \circ_c a = \langle x2, y2 \rangle$
using *y-def* **by** *blast*
show $a = b$
proof(*cases* $\exists c.\ b =$ *left-coproj* $X\ Y \circ_c c \wedge c \in_c X$)
assume $\exists c.\ b =$ *left-coproj* $X\ Y \circ_c c \wedge c \in_c X$
then obtain c **where** *c-def*: $b =$ *left-coproj* $X\ Y \circ_c c \wedge c \in_c X$
by *blast*
then show $a = b$
using *cart-prod-eq2 eqs m-leftproj-l-equals m-rightproj-y-equals x1x2-def*(*2*) *y1y2-def y-def* **by** *auto*
next
assume $\nexists c.\ b =$ *left-coproj* $X\ Y \circ_c c \wedge c \in_c X$
then obtain c **where** *c-def*: $b =$ *right-coproj* $X\ Y \circ_c c \wedge c \in_c Y$
using *b-type coprojs-jointly-surj* **by** *blast*
show $a = b$
proof(*cases* $c = y$)
assume $c = y$
show $a = b$
by (*simp add*: ‹$c = y$› *c-def y-def*)
next
assume $c \neq y$
then have $c \neq y1$
by (*simp add*: ‹$y = y1$›)
then obtain k **where** *k-def*: $k \in_c Y \setminus (\mathbf{1}, y1) \wedge$ *try-cast* $y1 \circ_c c =$ *right-coproj* $\mathbf{1}\ (Y \setminus (\mathbf{1}, y1)) \circ_c k \wedge$
$m \circ_c$ *right-coproj* $X\ Y \circ_c c = \langle x1, y1^c \circ_c k \rangle$
using *c-def m-rightproj-not-y1-equals* **by** *blast*
then have $\langle x2, y2 \rangle = \langle x1, y1^c \circ_c k \rangle$
using ‹$m \circ_c a = \langle x2, y2 \rangle$› *c-def eqs* **by** *auto*
then have *False*
using *comp-type element-pair-eq k-def x1x2-def y1'-type y1y2-def*(*2*) **by** *auto*
then show *?thesis*
by *simp*
qed
qed
next
assume $y \neq y1$
then obtain k **where** *k-def*: $k \in_c Y \setminus (\mathbf{1}, y1) \wedge$ *try-cast* $y1 \circ_c y =$ *right-coproj* $\mathbf{1}\ (Y \setminus (\mathbf{1}, y1)) \circ_c k \wedge$
$m \circ_c$ *right-coproj* $X\ Y \circ_c y = \langle x1, y1^c \circ_c k \rangle$
using *m-rightproj-not-y1-equals y-def* **by** *blast*
then have $m \circ_c a = \langle x1, y1^c \circ_c k \rangle$
using *y-def* **by** *blast*
show $a = b$
proof(*cases* $\exists c.\ b =$ *right-coproj* $X\ Y \circ_c c \wedge c \in_c Y$)
assume $\exists c.\ b =$ *right-coproj* $X\ Y \circ_c c \wedge c \in_c Y$
then obtain c **where** *c-def*: $b =$ *right-coproj* $X\ Y \circ_c c \wedge c \in_c Y$

by *blast*
show $a = b$
proof(*cases* $c = y1$)
assume $c = y1$
show $a = b$
proof –
obtain *cc* :: *cfunc* **where**
f1: $cc \in_c Y \setminus (\mathbf{1}, y1) \wedge$ *try-cast* $y1 \circ_c y =$ *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1}, y1)) \circ_c cc \wedge m \circ_c$ *right-coproj* $X\ Y \circ_c y = \langle x1, y1^c \circ_c cc\rangle$
using ‹$\bigwedge$*thesis*. ($\bigwedge k.\ k \in_c Y \setminus (\mathbf{1}, y1) \wedge$ *try-cast* $y1 \circ_c y =$ *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1}, y1)) \circ_c k \wedge m \circ_c$ *right-coproj* $X\ Y \circ_c y = \langle x1, y1^c \circ_c k\rangle$ $\Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*› **by** *blast*
have $\langle x2, y2\rangle = m \circ_c a$
using ‹$c = y1$› *c-def eqs m-rightproj-y1-equals* **by** *presburger*
then show *?thesis*
using *f1 cart-prod-eq2 comp-type x1x2-def y1'-type y1y2-def*(*2*) *y-def* **by** *force*
qed
next
assume $c \neq y1$
then obtain k' **where** k'*-def*: $k' \in_c Y \setminus (\mathbf{1}, y1) \wedge$ *try-cast* $y1 \circ_c c =$ *right-coproj* $\mathbf{1}$ $(Y \setminus (\mathbf{1}, y1)) \circ_c k' \wedge$
$m \circ_c$ *right-coproj* $X\ Y \circ_c c = \langle x1, y1^c \circ_c k'\rangle$
using *c-def m-rightproj-not-y1-equals* **by** *blast*
then have $\langle x1, y1^c \circ_c k'\rangle = \langle x1, y1^c \circ_c k\rangle$
using *c-def eqs k-def y-def* **by** *auto*
then have $(x1 = x1) \wedge (y1^c \circ_c k' = y1^c \circ_c k)$
using *element-pair-eq* k'*-def k-def* **by** (*typecheck-cfuncs*, *blast*)
then have $k' = k$
by (*metis cfunc-type-def complement-morphism-mono* k'*-def k-def monomorphism-def y1'-type y1-mono*)
then have $c = y$
by (*metis c-def cfunc-type-def* k'*-def k-def monomorphism-def try-cast-mono trycast-y1-type y1-mono y-def*)
then show $a = b$
by (*simp add*: *c-def y-def*)
qed
next
assume $\nexists c.\ b =$ *right-coproj* $X\ Y \circ_c c \wedge c \in_c Y$
then obtain c **where** *c-def*: $b =$ *left-coproj* $X\ Y \circ_c c \wedge c \in_c X$
using *b-type coprojs-jointly-surj* **by** *blast*
then have $m \circ_c$ *left-coproj* $X\ Y \circ_c c = \langle c, y1\rangle$
by (*simp add*: *m-leftproj-l-equals*)
then have $\langle c, y1\rangle = \langle x1, y1^c \circ_c k\rangle$
using ‹$m \circ_c a = \langle x1, y1^c \circ_c k\rangle$› ‹$m \circ_c$ *left-coproj* $X\ Y \circ_c c = \langle c, y1\rangle$› *c-def eqs* **by** *auto*
then have $(c = x1) \wedge (y1 = y1^c \circ_c k)$
using *c-def cart-prod-eq2 comp-type k-def x1x2-def*(*1*) *y1'-type y1y2-def*(*1*) **by** *auto*

then have *False*
by (*metis cfunc-type-def complement-disjoint id-right-unit id-type k-def y1-mono y1y2-def*(*1*))
then show *?thesis*
by *simp*
qed
qed
qed
qed
then have *monomorphism m*
using *injective-imp-monomorphism* **by** *auto*
then show *?thesis*
using *is-smaller-than-def m-type* **by** *blast*
qed

lemma *prod-leq-exp*:
assumes ¬ *terminal-object Y*
shows $X \times_c Y \leq_c Y^X$
proof(*cases initial-object Y*)
show *initial-object Y* $\Longrightarrow X \times_c Y \leq_c Y^X$
by (*metis X-prod-empty initial-iso-empty initial-maps-mono initial-object-def is-smaller-than-def iso-empty-initial isomorphic-is-reflexive isomorphic-is-transitive prod-pres-iso*)
next
assume ¬ *initial-object Y*
then obtain *y1 y2* **where** *y1-type*[*type-rule*]: $y1 \in_c Y$ **and** *y2-type*[*type-rule*]: $y2 \in_c Y$ **and** *y1-not-y2*: $y1 \neq y2$
using *assms not-init-not-term* **by** *blast*
show $X \times_c Y \leq_c Y^X$
proof(*cases* $X \cong \Omega$)
assume $X \cong \Omega$
have $\Omega \leq_c Y$
using ‹¬ *initial-object Y*› *assms not-init-not-term size-2plus-sets* **by** *blast*
then obtain *m* **where** *m-type*[*type-rule*]: $m : \Omega \to Y$ **and** *m-mono*: *monomorphism m*
using *is-smaller-than-def* **by** *blast*
then have *m-id-type*[*type-rule*]: $m \times_f id(Y) : \Omega \times_c Y \to Y \times_c Y$
by *typecheck-cfuncs*
have *m-id-mono*: *monomorphism* ($m \times_f id(Y)$)
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-mono id-isomorphism iso-imp-epi-and-monic m-mono*)
obtain *n* **where** *n-type*[*type-rule*]: $n : Y \times_c Y \to Y^\Omega$ **and** *n-mono*: *monomorphism n*
using *is-isomorphic-def iso-imp-epi-and-monic isomorphic-is-symmetric sets-squared* **by** *blast*
obtain *r* **where** *r-type*[*type-rule*]: $r : Y^\Omega \to Y^X$ **and** *r-mono*: *monomorphism r*
by (*meson* ‹$X \cong \Omega$› *exp-pres-iso-right is-isomorphic-def iso-imp-epi-and-monic isomorphic-is-symmetric*)

obtain q **where** *q-type*[*type-rule*]: $q : X \times_c Y \rightarrow \Omega \times_c Y$ **and** *q-mono*: *monomorphism q*
by (*meson* ‹$X \cong \Omega$› *id-isomorphism id-type is-isomorphic-def iso-imp-epi-and-monic prod-pres-iso*)
have *rnmq-type*[*type-rule*]: $r \circ_c n \circ_c (m \times_f id(Y)) \circ_c q : X \times_c Y \rightarrow Y^X$
by *typecheck-cfuncs*
have *monomorphism*$(r \circ_c n \circ_c (m \times_f id(Y)) \circ_c q)$
by (*typecheck-cfuncs*, *simp add*: *cfunc-type-def composition-of-monic-pair-is-monic m-id-mono n-mono q-mono r-mono*)
then show *?thesis*
by (*meson is-smaller-than-def rnmq-type*)
next
assume $\neg X \cong \Omega$
show $X \times_c Y \leq_c Y^X$
proof(*cases initial-object X*)
show *initial-object* $X \Longrightarrow X \times_c Y \leq_c Y^X$
by (*metis is-empty-def initial-iso-empty initial-maps-mono initial-object-def*

is-smaller-than-def isomorphic-is-transitive no-el-iff-iso-empty
not-init-not-term prod-with-empty-is-empty2 product-commutes terminal-object-def)
next
assume $\neg$ *initial-object X*
show $X \times_c Y \leq_c Y^X$
proof(*cases terminal-object X*)
assume *terminal-object X*
then have $X \cong \mathbf{1}$
by (*simp add*: *one-terminal-object terminal-objects-isomorphic*)
have $X \times_c Y \cong Y$
by (*simp add*: ‹*terminal-object X*› *prod-with-term-obj1*)
then have $X \times_c Y \cong Y^X$
by (*meson* ‹$X \cong \mathbf{1}$› *exp-pres-iso-right exp-set-inj isomorphic-is-symmetric isomorphic-is-transitive exp-one*)
then show *?thesis*
using *is-isomorphic-def is-smaller-than-def iso-imp-epi-and-monic* **by** *blast*
next
assume $\neg$ *terminal-object X*

obtain *into* **where** *into-def*: *into* = (*left-cart-proj* Y $\mathbf{1}$ $\amalg$ ((*y2* $\amalg$ *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y $\circ_c$ (*id* Y $\times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y $\mathbf{1}$ $\mathbf{1}$ $\circ_c$ (*id* Y $\times_f$ *case-bool*) $\circ_c$ (*id* Y $\times_f$ *eq-pred* X)
by *simp*
then have *into-type*[*type-rule*]: *into* : $Y \times_c (X \times_c X) \rightarrow Y$
by (*simp*, *typecheck-cfuncs*)

obtain Θ **where** Θ-*def*: $\Theta = (into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X))^\sharp \circ_c swap\ X\ Y$

by *auto*

have Θ-*type*[*type-rule*]: $\Theta : X \times_c Y \to Y^X$
unfolding Θ-*def* **by** *typecheck-cfuncs*

have *f0*: $\bigwedge x.\ \bigwedge y.\ \bigwedge z.\ x \in_c X \wedge y \in_c Y \wedge z \in_c X \Longrightarrow (\Theta \circ_c \langle x,\ y\rangle)^\flat \circ_c \langle id\ X,\ \beta_X\rangle \circ_c z = into \circ_c \ \langle y,\ \langle x,\ z\rangle\rangle$
proof(*clarify*)
fix *x y z*
assume *x-type*[*type-rule*]: $x \in_c X$
assume *y-type*[*type-rule*]: $y \in_c Y$
assume *z-type*[*type-rule*]: $z \in_c X$
show $(\Theta \circ_c \langle x,y\rangle)^\flat \circ_c \langle id_c\ X,\beta_X\rangle \circ_c z = into \circ_c \langle y,\langle x,z\rangle\rangle$
proof −
have $(\Theta \circ_c \langle x,y\rangle)^\flat \circ_c \langle id_c\ X,\beta_X\rangle \circ_c z = (\Theta \circ_c \langle x,y\rangle)^\flat \circ_c \langle id_c\ X \circ_c z,\beta_X \circ_c z\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-prod-comp*)
also have ... $= (\Theta \circ_c \langle x,y\rangle)^\flat \circ_c \langle z,id\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *metis id-left-unit2 one-unique-element*)
also have ... $= (\Theta^\flat \circ_c (id(X) \times_f \langle x,y\rangle)) \circ_c \langle z,id\ \mathbf{1}\rangle$
using *inv-transpose-of-composition* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= \Theta^\flat \circ_c (id(X) \times_f \langle x,y\rangle) \circ_c \langle z,id\ \mathbf{1}\rangle$
using *comp-associative2* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= \Theta^\flat \circ_c \langle id(X) \circ_c \ z,\ \langle x,y\rangle \circ_c \ id\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... $= \Theta^\flat \circ_c \langle z,\langle x,y\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *id-left-unit2 id-right-unit2*)
also have ... $= ((into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X))^\sharp \circ_c swap\ X\ Y)^\flat \circ_c \langle z,\langle x,y\rangle\rangle$
by (*simp add*: Θ-*def*)
also have ... $= ((into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X))^{\sharp\flat} \circ_c (id\ X \times_f swap\ X\ Y)) \circ_c \langle z,\langle x,y\rangle\rangle$
using *inv-transpose-of-composition* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= (into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X)) \circ_c (id\ X \times_f swap\ X\ Y) \circ_c \langle z,\langle x,y\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *comp-associative2 inv-transpose-func-def3 transpose-func-def*)
also have ... $= (into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X)) \circ_c \langle id\ X \circ_c z,\ swap\ X\ Y \circ_c \langle x,y\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... $= (into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X)) \circ_c \langle z,\ \langle y,x\rangle\rangle$
using *id-left-unit2 swap-ap* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= into \circ_c associate\text{-}right\ Y\ X\ X \circ_c swap\ X\ (Y \times_c X) \circ_c \langle z,\ \langle y,x\rangle\rangle$
by (*typecheck-cfuncs*, *metis cfunc-type-def comp-associative*)
also have ... $= into \circ_c associate\text{-}right\ Y\ X\ X \circ_c \ \langle\langle y,x\rangle,\ z\rangle$
using *swap-ap* **by** (*typecheck-cfuncs*, *presburger*)
also have ... $= into \circ_c \ \langle y,\ \langle x,\ z\rangle\rangle$

using *associate-right-ap* **by** (*typecheck-cfuncs*, *presburger*)
finally show *?thesis*.
qed
qed

have *f1*: $\bigwedge x\ y.\ x \in_c X \Longrightarrow y \in_c Y \Longrightarrow (\Theta \circ_c \langle x, y\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c x = y$
proof −
fix *x y*
assume *x-type*[*type-rule*]: $x \in_c X$
assume *y-type*[*type-rule*]: $y \in_c Y$
have $(\Theta \circ_c \langle x, y\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c x = into \circ_c \langle y, \langle x, x\rangle\rangle$
by (*simp add*: *f0 x-type y-type*)
also have ... $= (left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c dist\text{-}prod\text{-}coprod\text{-}left\ Y\ \mathbf{1}\ \mathbf{1} \circ_c (id\ Y \times_f case\text{-}bool) \circ_c (id\ Y \times_f eq\text{-}pred\ X) \circ_c \langle y, \langle x, x\rangle\rangle$
using *cfunc-type-def comp-associative comp-type into-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... $= (left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c dist\text{-}prod\text{-}coprod\text{-}left\ Y\ \mathbf{1}\ \mathbf{1} \circ_c (id\ Y \times_f case\text{-}bool) \circ_c \langle id\ Y \circ_c y, eq\text{-}pred\ X \circ_c \langle x, x\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... $= (left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c dist\text{-}prod\text{-}coprod\text{-}left\ Y\ \mathbf{1}\ \mathbf{1} \circ_c (id\ Y \times_f case\text{-}bool) \circ_c \langle y, \mathrm{t}\rangle$
by (*typecheck-cfuncs*, *metis eq-pred-iff-eq id-left-unit2*)
also have ... $= (left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c dist\text{-}prod\text{-}coprod\text{-}left\ Y\ \mathbf{1}\ \mathbf{1} \circ_c \langle y, left\text{-}coproj\ \mathbf{1}\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *simp add*: *case-bool-true cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... $= (left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c dist\text{-}prod\text{-}coprod\text{-}left\ Y\ \mathbf{1}\ \mathbf{1} \circ_c \langle y, left\text{-}coproj\ \mathbf{1}\ \mathbf{1} \circ_c id\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *metis id-right-unit2*)
also have ... $= (left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c left\text{-}coproj\ (Y \times_c \mathbf{1})\ (Y \times_c \mathbf{1}) \circ_c \langle y, id\ \mathbf{1}\rangle$
using *dist-prod-coprod-left-ap-left* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= ((left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \amalg ((y2 \amalg y1) \circ_c case\text{-}bool \circ_c eq\text{-}pred\ Y \circ_c (id\ Y \times_f y1)))$
$\circ_c left\text{-}coproj\ (Y \times_c \mathbf{1})\ (Y \times_c \mathbf{1})) \circ_c \langle y, id\ \mathbf{1}\rangle$
by (*typecheck-cfuncs*, *meson comp-associative2*)
also have ... $= left\text{-}cart\text{-}proj\ Y\ \mathbf{1} \circ_c \langle y, id\ \mathbf{1}\rangle$
using *left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)

also have ... = y
by (*typecheck-cfuncs*, *simp add*: *left-cart-proj-cfunc-prod*)
finally show $(\Theta \circ_c \langle x, y\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c x = y$.
qed

have *f2*: $\bigwedge x\ y\ z.\ x \in_c X \Longrightarrow y \in_c Y \Longrightarrow z \in_c X \Longrightarrow z \neq x \Longrightarrow y \neq y1$
$\Longrightarrow (\Theta \circ_c \langle x, y\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c z = y1$
proof −
fix $x\ y\ z$
assume *x-type*[*type-rule*]: $x \in_c X$
assume *y-type*[*type-rule*]: $y \in_c Y$
assume *z-type*[*type-rule*]: $z \in_c X$
assume $z \neq x$
assume $y \neq y1$
have $(\Theta \circ_c \langle x, y\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c z = into \circ_c \langle y, \langle x, z\rangle\rangle$
by (*simp add*: *f0 x-type y-type z-type*)
also have ... = (*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y
$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y **1** **1** $\circ_c$ (*id* $Y \times_f$ *case-bool*) $\circ_c$
(*id* $Y \times_f$ *eq-pred* X) $\circ_c$ $\langle y, \langle x, z\rangle\rangle$
using *cfunc-type-def comp-associative comp-type into-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y
$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y **1** **1** $\circ_c$ (*id* $Y \times_f$ *case-bool*) $\circ_c$
$\langle id\ Y \circ_c y$, *eq-pred* $X \circ_c \langle x, z\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = (*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y
$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y **1** **1** $\circ_c$ (*id* $Y \times_f$ *case-bool*) $\circ_c$
$\langle y$, f$\rangle$
by (*typecheck-cfuncs*, *metis* ‹$z \neq x$› *eq-pred-iff-eq-conv id-left-unit2*)
also have ... = (*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y
$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y **1** **1** $\circ_c$ $\langle y$, *right-coproj* **1** **1**$\rangle$
by (*typecheck-cfuncs*, *simp add*: *case-bool-false cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = (*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y
$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y **1** **1** $\circ_c$ $\langle y$, *right-coproj* **1** **1**
$\circ_c$ *id* **1**$\rangle$
by (*typecheck-cfuncs*, *simp add*: *id-right-unit2*)
also have ... = (*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y
$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *right-coproj* ($Y \times_c$ **1**) ($Y \times_c$ **1**) $\circ_c$ $\langle y$,*id* **1**$\rangle$
using *dist-prod-coprod-left-ap-right* **by** (*typecheck-cfuncs*, *auto*)
also have ... = ((*left-cart-proj* Y **1** II ((*y2* II *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred*
$Y \circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *right-coproj* ($Y \times_c$ **1**) ($Y \times_c$ **1**)) $\circ_c$ $\langle y$,*id* **1**$\rangle$

by (*typecheck-cfuncs*, *meson comp-associative2*)
also have ... = $((y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c (id\ Y \times_f y1)) \circ_c$ $\langle y, id\ \mathbf{1}\rangle$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = $(y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c (id\ Y \times_f y1) \circ_c$ $\langle y, id\ \mathbf{1}\rangle$
using *comp-associative2* **by** (*typecheck-cfuncs*, *force*)
also have ... = $(y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c \langle y, y1\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
also have ... = $(y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ f
by (*typecheck-cfuncs*, *metis* ‹$y \neq y1$› *eq-pred-iff-eq-conv*)
also have ... = *y1*
using *case-bool-false right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
finally show $(\Theta \circ_c \langle x, y\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c z = y1$.
qed

have *f3*: $\bigwedge x\ z.\ x \in_c X \Longrightarrow z \in_c X \Longrightarrow z \neq x \Longrightarrow (\Theta \circ_c \langle x, y1\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c z = y2$
proof −
fix *x y z*
assume *x-type*[*type-rule*]: $x \in_c X$
assume *z-type*[*type-rule*]: $z \in_c X$
assume $z \neq x$
have $(\Theta \circ_c \langle x, y1\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c z = into \circ_c \langle y1, \langle x, z\rangle\rangle$
by (*simp add*: *f0 x-type y1-type z-type*)
also have ... = (*left-cart-proj* Y $\mathbf{1}$ $\amalg$ $((y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c (id\ Y \times_f y1)))$
$\circ_c$ *dist-prod-coprod-left* Y $\mathbf{1}$ $\mathbf{1}$ $\circ_c (id\ Y \times_f$ *case-bool*$) \circ_c$ $(id\ Y \times_f$ *eq-pred* $X) \circ_c \langle y1, \langle x, z\rangle\rangle$
using *cfunc-type-def comp-associative comp-type into-def* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*left-cart-proj* Y $\mathbf{1}$ $\amalg$ $((y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c (id\ Y \times_f y1)))$
$\circ_c$ *dist-prod-coprod-left* Y $\mathbf{1}$ $\mathbf{1}$ $\circ_c (id\ Y \times_f$ *case-bool*$) \circ_c$ $\langle id\ Y \circ_c y1,$ *eq-pred* $X \circ_c \langle x, z\rangle\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod*)
also have ... = (*left-cart-proj* Y $\mathbf{1}$ $\amalg$ $((y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c (id\ Y \times_f y1)))$
$\circ_c$ *dist-prod-coprod-left* Y $\mathbf{1}$ $\mathbf{1}$ $\circ_c (id\ Y \times_f$ *case-bool*$) \circ_c$ $\langle y1, \mathrm{f}\rangle$
by (*typecheck-cfuncs*, *metis* ‹$z \neq x$› *eq-pred-iff-eq-conv id-left-unit2*)
also have ... = (*left-cart-proj* Y $\mathbf{1}$ $\amalg$ $((y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* $Y \circ_c (id\ Y \times_f y1)))$
$\circ_c$ *dist-prod-coprod-left* Y $\mathbf{1}$ $\mathbf{1}$ $\circ_c \langle y1,$ *right-coproj* $\mathbf{1}$ $\mathbf{1}\rangle$
by (*typecheck-cfuncs*, *simp add*: *case-bool-false cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
also have ... = (*left-cart-proj* Y $\mathbf{1}$ $\amalg$ $((y2 \amalg y1) \circ_c$ *case-bool* $\circ_c$ *eq-pred* Y

$\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *dist-prod-coprod-left* Y **1 1** $\circ_c$ $\langle$*y1*, *right-coproj* **1 1** $\circ_c$ *id* **1**$\rangle$
by (*typecheck-cfuncs*, *simp add*: *id-right-unit2*)
also have ... = (*left-cart-proj* Y **1** ∐ ((*y2* ∐ *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y $\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *right-coproj* ($Y \times_c$ **1**) ($Y \times_c$ **1**) $\circ_c$ $\langle$*y1*,*id* **1**$\rangle$
using *dist-prod-coprod-left-ap-right* **by** (*typecheck-cfuncs*, *auto*)
also have ... = ((*left-cart-proj* Y **1** ∐ ((*y2* ∐ *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y $\circ_c$ (*id* $Y \times_f$ *y1*)))
$\circ_c$ *right-coproj* ($Y \times_c$ **1**) ($Y \times_c$ **1**)) $\circ_c$ $\langle$*y1*,*id* **1**$\rangle$
by (*typecheck-cfuncs*, *meson comp-associative2*)
also have ... = ((*y2* ∐ *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y $\circ_c$ (*id* $Y \times_f$ *y1*)) $\circ_c$ $\langle$*y1*,*id* **1**$\rangle$
using *right-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *auto*)
also have ... = (*y2* ∐ *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y $\circ_c$ (*id* $Y \times_f$ *y1*) $\circ_c$ $\langle$*y1*,*id* **1**$\rangle$
using *comp-associative2* **by** (*typecheck-cfuncs*, *force*)
also have ... = (*y2* ∐ *y1*) $\circ_c$ *case-bool* $\circ_c$ *eq-pred* Y $\circ_c$ $\langle$*y1*,*y1*$\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-right-unit2*)
also have ... = (*y2* ∐ *y1*) $\circ_c$ *case-bool* $\circ_c$ t
by (*typecheck-cfuncs*, *metis eq-pred-iff-eq*)
also have ... = *y2*
using *case-bool-true left-coproj-cfunc-coprod* **by** (*typecheck-cfuncs*, *presburger*)
finally show $(\Theta \circ_c \langle x, y1\rangle)^\flat \circ_c \langle id\ X, \beta_X\rangle \circ_c z = y2$**.**
qed

have Θ-*injective*: *injective*(Θ)
unfolding *injective-def*
proof(*clarify*)
fix *xy st*
assume *xy-type*[*type-rule*]: *xy* $\in_c$ *domain* Θ
assume *st-type*[*type-rule*]: *st* $\in_c$ *domain* Θ
assume *equals*: Θ $\circ_c$ *xy* = Θ $\circ_c$ *st*
obtain *x y* **where** *x-type*[*type-rule*]: $x \in_c X$ **and** *y-type*[*type-rule*]: $y \in_c Y$ **and** *xy-def*: *xy* = $\langle x,y\rangle$
by (*metis* Θ-*type cart-prod-decomp cfunc-type-def xy-type*)
obtain *s t* **where** *s-type*[*type-rule*]: $s \in_c X$ **and** *t-type*[*type-rule*]: $t \in_c Y$ **and** *st-def*: *st* = $\langle s,t\rangle$
by (*metis* Θ-*type cart-prod-decomp cfunc-type-def st-type*)
have *equals2*: Θ $\circ_c$ $\langle x,y\rangle$ = Θ $\circ_c$ $\langle s,t\rangle$
using *equals st-def xy-def* **by** *auto*
have $\langle x,y\rangle = \langle s,t\rangle$
proof(*cases* $y = y1$)
assume $y = y1$
show $\langle x,y\rangle = \langle s,t\rangle$
proof(*cases* $t = y1$)

```
          show t = y1 ⟹ ⟨x,y⟩ = ⟨s,t⟩
           by (typecheck-cfuncs, metis ‹y = y1› equals f1 f3 st-def xy-def y1-not-y2)
        next
          assume t ≠ y1
          show ⟨x,y⟩ = ⟨s,t⟩
          proof(cases s = x)
            show s = x ⟹ ⟨x,y⟩ = ⟨s,t⟩
              by (typecheck-cfuncs, metis equals2 f1)
          next
            assume s ≠ x
              obtain z where z-type[type-rule]: z ∈c X and z-not-x: z ≠ x and
z-not-s: z ≠ s
                by (metis ‹¬ X ≅ Ω› ‹¬ initial-object X› ‹¬ terminal-object X›
sets-size-3-plus)
            have t-sz: (Θ ∘c ⟨s, t⟩)♭ ∘c ⟨id X, β_X⟩ ∘c z = y1
              by (simp add: ‹t ≠ y1› f2 s-type t-type z-not-s z-type)
            have y-xz: (Θ ∘c ⟨x, y⟩)♭ ∘c ⟨id X, β_X⟩ ∘c z = y2
              by (simp add: ‹y = y1› f3 x-type z-not-x z-type)
            then have y1 = y2
              using equals2 t-sz by auto
            then have False
              using y1-not-y2 by auto
            then show ⟨x,y⟩ = ⟨s,t⟩
              by simp
          qed
        qed
      next
        assume y ≠ y1
        show ⟨x,y⟩ = ⟨s,t⟩
        proof(cases y = y2)
          assume y = y2
          show ⟨x,y⟩ = ⟨s,t⟩
          proof(cases t = y2, clarify)
            show t = y2 ⟹ ⟨x,y⟩ = ⟨s,y2⟩
                by (typecheck-cfuncs, metis ‹y = y2› ‹y ≠ y1› equals f1 f2 st-def
xy-def)
          next
            assume t ≠ y2
            show ⟨x,y⟩ = ⟨s,t⟩
            proof(cases x = s, clarify)
              show x = s ⟹ ⟨s,y⟩ = ⟨s,t⟩
                by (metis equals2 f1 s-type t-type y-type)
            next
              assume x ≠ s
              show ⟨x,y⟩ = ⟨s,t⟩
              proof(cases t = y1,clarify)
                show t = y1 ⟹ ⟨x,y⟩ = ⟨s,y1⟩
                 by (metis ‹¬ X ≅ Ω› ‹¬ initial-object X› ‹¬ terminal-object X› ‹y
= y2› ‹y ≠ y1› equals f2 f3 s-type sets-size-3-plus st-def x-type xy-def y2-type)
```

next
assume *t* ≠ *y1*
show ⟨*x*,*y*⟩ = ⟨*s*,*t*⟩
by (*typecheck-cfuncs*, *metis* ‹*t* ≠ *y1*› ‹*y* ≠ *y1*› *equals f1 f2 st-def xy-def*)
qed
qed
qed
next
assume *y* ≠ *y2*
show ⟨*x*,*y*⟩ = ⟨*s*,*t*⟩
proof(*cases s* = *x*, *clarify*)
show *s* = *x* ⟹ ⟨*x*,*y*⟩ = ⟨*x*,*t*⟩
by (*metis equals2 f1 t-type x-type y-type*)
show *s* ≠ *x* ⟹ ⟨*x*,*y*⟩ = ⟨*s*,*t*⟩
by (*metis* ‹*y* ≠ *y1*› ‹*y* ≠ *y2*› *equals f1 f2 f3 s-type st-def t-type x-type xy-def y-type*)
qed
qed
qed
then show *xy* = *st*
by (*typecheck-cfuncs*, *simp add*: *st-def xy-def*)
qed
then show *?thesis*
using Θ-*type injective-imp-monomorphism is-smaller-than-def* **by** *blast*
qed
qed
qed
qed

lemma *Y-nonempty-then-X-le-XtoY*:
assumes *nonempty Y*
shows $X \leq_c X^Y$
proof −
obtain *f* **where** *f-def*: *f* = (*right-cart-proj Y X*)$^\sharp$
by *blast*
then have *f-type*: $f : X \to X^Y$
by (*simp add*: *right-cart-proj-type transpose-func-type*)
have *mono-f*: *injective*(*f*)
unfolding *injective-def*
proof(*clarify*)
fix *x y*
assume *x-type*: $x \in_c$ *domain f*
assume *y-type*: $y \in_c$ *domain f*
assume *equals*: $f \circ_c x = f \circ_c y$
have *x-type2* : $x \in_c X$
using *cfunc-type-def f-type x-type* **by** *auto*
have *y-type2* : $y \in_c X$
using *cfunc-type-def f-type y-type* **by** *auto*

have $x \circ_c$ (*right-cart-proj* Y **1**) = (*right-cart-proj* Y X) $\circ_c$ ($id(Y) \times_f x$)
using *right-cart-proj-cfunc-cross-prod x-type2* **by** (*typecheck-cfuncs*, *auto*)
also have ... = ((*eval-func* X Y) $\circ_c$ ($id(Y) \times_f f$)) $\circ_c$ ($id(Y) \times_f x$)
by (*typecheck-cfuncs*, *simp add*: *f-def transpose-func-def*)
also have ... = (*eval-func* X Y) $\circ_c$ (($id(Y) \times_f f$) $\circ_c$ ($id(Y) \times_f x$))
using *comp-associative2 f-type x-type2* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*eval-func* X Y) $\circ_c$ ($id(Y) \times_f (f \circ_c x)$)
using *f-type identity-distributes-across-composition x-type2* **by** *auto*
also have ... = (*eval-func* X Y) $\circ_c$ ($id(Y) \times_f (f \circ_c y)$)
by (*simp add*: *equals*)
also have ... = (*eval-func* X Y) $\circ_c$ (($id(Y) \times_f f$) $\circ_c$ ($id(Y) \times_f y$))
using *f-type identity-distributes-across-composition y-type2* **by** *auto*
also have ... = ((*eval-func* X Y) $\circ_c$ ($id(Y) \times_f f$)) $\circ_c$ ($id(Y) \times_f y$)
using *comp-associative2 f-type y-type2* **by** (*typecheck-cfuncs*, *fastforce*)
also have ... = (*right-cart-proj* Y X) $\circ_c$ ($id(Y) \times_f y$)
by (*typecheck-cfuncs*, *simp add*: *f-def transpose-func-def*)
also have ... = $y \circ_c$ (*right-cart-proj* Y **1**)
using *right-cart-proj-cfunc-cross-prod y-type2* **by** (*typecheck-cfuncs*, *auto*)
ultimately show $x = y$
using *assms epimorphism-def3 nonempty-left-imp-right-proj-epimorphism right-cart-proj-type x-type2 y-type2* **by** *fastforce*
qed
then show $X \leq_c X^Y$
using *f-type injective-imp-monomorphism is-smaller-than-def* **by** *blast*
qed

lemma *non-init-non-ter-sets*:
assumes ¬(*terminal-object* X)
assumes ¬(*initial-object* X)
shows $\Omega \leq_c X$
proof −
obtain *x1* **and** *x2* **where** *x1-type*[*type-rule*]: $x1 \in_c X$ **and**
x2-type[*type-rule*]: $x2 \in_c X$ **and**
distinct: $x1 \neq x2$
using *is-empty-def assms iso-empty-initial iso-to1-is-term no-el-iff-iso-empty single-elem-iso-one* **by** *blast*
then have *map-type*: ($x1$ ⨿ $x2$) $\circ_c$ *case-bool* : $\Omega \to X$
by *typecheck-cfuncs*
have *injective*: *injective*(($x1$ ⨿ $x2$) $\circ_c$ *case-bool*)
unfolding *injective-def*
proof(*clarify*)
fix $\omega 1$ $\omega 2$
assume $\omega 1 \in_c$ *domain* ($x1$ ⨿ $x2$ $\circ_c$ *case-bool*)
then have *ω1-type*[*type-rule*]: $\omega 1 \in_c \Omega$
using *cfunc-type-def map-type* **by** *auto*
assume $\omega 2 \in_c$ *domain* ($x1$ ⨿ $x2$ $\circ_c$ *case-bool*)
then have *ω2-type*[*type-rule*]: $\omega 2 \in_c \Omega$
using *cfunc-type-def map-type* **by** *auto*

assume *equals*: $(x1 \amalg x2 \circ_c \textit{case-bool}) \circ_c \omega 1 = (x1 \amalg x2 \circ_c \textit{case-bool}) \circ_c \omega 2$
show $\omega 1 = \omega 2$
proof(*cases* $\omega 1 = \mathrm{t}$, *clarify*)
assume $\omega 1 = \mathrm{t}$
show $\mathrm{t} = \omega 2$
proof(*rule ccontr*)
assume $\mathrm{t} \neq \omega 2$
then have $\mathrm{f} = \omega 2$
using ‹$\mathrm{t} \neq \omega 2$› *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have *RHS*: $(x1 \amalg x2 \circ_c \textit{case-bool}) \circ_c \omega 2 = x2$
by (*meson coprod-case-bool-false x1-type x2-type*)
have $(x1 \amalg x2 \circ_c \textit{case-bool}) \circ_c \omega 1 = x1$
using ‹$\omega 1 = \mathrm{t}$› *coprod-case-bool-true x1-type x2-type* **by** *blast*
then show *False*
using *RHS distinct equals* **by** *force*
qed
next
assume $\omega 1 \neq \mathrm{t}$
then have $\omega 1 = \mathrm{f}$
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
have $\omega 2 = \mathrm{f}$
proof(*rule ccontr*)
assume $\omega 2 \neq \mathrm{f}$
then have $\omega 2 = \mathrm{t}$
using *true-false-only-truth-values* **by** (*typecheck-cfuncs*, *blast*)
then have *RHS*: $(x1 \amalg x2 \circ_c \textit{case-bool}) \circ_c \omega 2 = x2$
using ‹$\omega 1 = \mathrm{f}$› *coprod-case-bool-false equals x1-type x2-type* **by** *auto*
have $(x1 \amalg x2 \circ_c \textit{case-bool}) \circ_c \omega 1 = x1$
using ‹$\omega 2 = \mathrm{t}$› *coprod-case-bool-true equals x1-type x2-type* **by** *presburger*
then show *False*
using *RHS distinct equals* **by** *auto*
qed
show $\omega 1 = \omega 2$
by (*simp add*: ‹$\omega 1 = \mathrm{f}$› ‹$\omega 2 = \mathrm{f}$›)
qed
qed
then have *monomorphism*$((x1 \amalg x2) \circ_c \textit{case-bool})$
using *injective-imp-monomorphism* **by** *auto*
then show $\Omega \leq_c X$
using *is-smaller-than-def map-type* **by** *blast*
qed

lemma *exp-preserves-card1*:
assumes $A \leq_c B$
assumes *nonempty* X
shows $X^A \leq_c X^B$
unfolding *is-smaller-than-def*
proof –
obtain x **where** *x-type*[*type-rule*]: $x \in_c X$

using *assms*(*2*) **unfolding** *nonempty-def* **by** *auto*
obtain *m* **where** *m-def*[*type-rule*]: $m : A \rightarrow B$ *monomorphism m*
using *assms*(*1*) **unfolding** *is-smaller-than-def* **by** *auto*
show $\exists\, m.\ m : X^A \rightarrow X^B \wedge$ *monomorphism m*
proof (*intro exI*[**where** x=(((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\, m))}$))
$\circ_c$ *dist-prod-coprod-left* (X^A) *A* $(B \setminus (A,\, m))$
$\circ_c$ *swap* ($A \coprod (B \setminus (A,\, m))$) (X^A) $\circ_c$ (*try-cast m* $\times_f$ *id* (X^A))))$^\sharp$], *safe*)

show ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\, m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* $(B \setminus (A,\, m))$ $\circ_c$ *swap* ($A \coprod (B \setminus (A,\, m))$) (X^A) $\circ_c$
try-cast m $\times_f$ id_c (X^A))$^\sharp$: $X^A \rightarrow X^B$
by *typecheck-cfuncs*
then show *monomorphism*
(((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\, m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* $(B \setminus (A,\, m))$ $\circ_c$
swap ($A \coprod (B \setminus (A,\, m))$) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A))$^\sharp$)
proof (*unfold monomorphism-def3*, *clarify*)
fix *g h Z*
assume *g-type*[*type-rule*]: $g : Z \rightarrow X^A$
assume *h-type*[*type-rule*]: $h : Z \rightarrow X^A$
assume *eq*: ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\, m))}$)
$\circ_c$
dist-prod-coprod-left (X^A) *A* $(B \setminus (A,\, m))$ $\circ_c$
swap ($A \coprod (B \setminus (A,\, m))$) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A))$^\sharp$ $\circ_c$ *g*
=
((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\, m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* $(B \setminus (A,\, m))$ $\circ_c$
swap ($A \coprod (B \setminus (A,\, m))$) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A))$^\sharp$ $\circ_c$ *h*

show $g = h$
proof (*typecheck-cfuncs*, *rule same-evals-equal*[**where** Z=Z, **where** A=A,
where X=X], *clarify*)
show *eval-func X A* $\circ_c$ id_c $A \times_f g$ = *eval-func X A* $\circ_c$ id_c $A \times_f h$
proof (*typecheck-cfuncs*, *rule one-separator*[**where** X=$A \times_c Z$, **where**
Y=X], *clarify*)
fix *az*
assume *az-type*[*type-rule*]: $az \in_c A \times_c Z$

obtain *a z* **where** *az-types*[*type-rule*]: $a \in_c A$ $z \in_c Z$ **and** *az-def*: az =
$\langle a,z \rangle$
using *cart-prod-decomp az-type* **by** *blast*

have (*eval-func X B*) $\circ_c$ (*id B* $\times_f$ (((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\amalg$
($x \circ_c \beta_{X^A \times_c (B \setminus (A,\, m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* $(B \setminus (A,\, m))$ $\circ_c$

swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c $(X^A))^\sharp \circ_c g)) =$
(*eval-func* X B) $\circ_c$ (*id* $B \times_f$ (((*eval-func* X $A \circ_c$ *swap* (X^A) A) $\amalg$ ($x \circ_c$
$\beta_{X^A \times_c (B \setminus (A, m))}) \circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c $(X^A))^\sharp \circ_c h))$
using *eq* **by** *simp*
then have (*eval-func* X B)$\circ_c$ (*id* $B \times_f$ (((*eval-func* X $A \circ_c$ *swap* (X^A) A)
$\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A, m))}) \circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c $(X^A))^\sharp)) \circ_c$ (*id* B
$\times_f$ $g) =$
(*eval-func* X B)$\circ_c$ (*id* $B \times_f$ (((*eval-func* X $A \circ_c$ *swap* (X^A) A) $\amalg$ ($x \circ_c$
$\beta_{X^A \times_c (B \setminus (A, m))}) \circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c $(X^A))^\sharp)) \circ_c$ (*id* B
$\times_f$ h)
using *identity-distributes-across-composition* **by** (*typecheck-cfuncs*, *auto*)
then have ((*eval-func* X B)$\circ_c$ (*id* $B \times_f$ (((*eval-func* X $A \circ_c$ *swap* (X^A)
A) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A, m))}) \circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c $(X^A))^\sharp))) \circ_c$ (*id*
$B \times_f$ $g) =$
((*eval-func* X B)$\circ_c$ (*id* $B \times_f$ (((*eval-func* X $A \circ_c$ *swap* (X^A) A) $\amalg$ ($x \circ_c$
$\beta_{X^A \times_c (B \setminus (A, m))}) \circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c $(X^A))^\sharp))) \circ_c$ (*id*
$B \times_f$ h)
by (*typecheck-cfuncs*, *smt eq inv-transpose-func-def3 inv-transpose-of-composition*)
then have ((*eval-func* X $A \circ_c$ *swap* (X^A) A) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A, m))})$
$\circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c (X^A)) $\circ_c$ (*id* B
$\times_f$ g)
= ((*eval-func* X $A \circ_c$ *swap* (X^A) A) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A, m))}) \circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c (X^A)) $\circ_c$ (*id* B
$\times_f$ h)
using *transpose-func-def* **by** (*typecheck-cfuncs*,*auto*)
then have (((*eval-func* X $A \circ_c$ *swap* (X^A) A) $\amalg$ ($x \circ_c \beta_{X^A \times_c (B \setminus (A, m))})$
$\circ_c$
dist-prod-coprod-left (X^A) A ($B \setminus (A, m)$) $\circ_c$
swap ($A \coprod (B \setminus (A, m))$) ($X^A$) $\circ_c$ *try-cast* $m \times_f$ id_c (X^A)) $\circ_c$ (*id* B
$\times_f$ g)) $\circ_c \langle m \circ_c a, z \rangle$

= (((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ (*id B* $\times_f$ *h*)) $\circ_c$ ⟨*m* $\circ_c$ *a*, *z*⟩
by *auto*
then have ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ (*id B* $\times_f$ *g*) $\circ_c$ ⟨*m* $\circ_c$ *a*, *z*⟩
= ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ (*id B* $\times_f$ *h*) $\circ_c$ ⟨*m* $\circ_c$ *a*, *z*⟩
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
then have ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ ⟨*m* $\circ_c$ *a*, *g* $\circ_c$ *z*⟩
= ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ *try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ ⟨*m* $\circ_c$ *a*, *h* $\circ_c$ *z*⟩
by (*typecheck-cfuncs*, *smt cfunc-cross-prod-comp-cfunc-prod id-left-unit2 id-type*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ (*try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ ⟨*m* $\circ_c$ *a*, *g* $\circ_c$ *z*⟩
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ (*try-cast m* $\times_f$ id_c (X^A)) $\circ_c$ ⟨*m* $\circ_c$ *a*, *h* $\circ_c$ *z*⟩
by (*typecheck-cfuncs-prems*, *smt comp-associative2*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ ⟨*try-cast m* $\circ_c$ *m* $\circ_c$ *a*, *g* $\circ_c$ *z*⟩
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) ⨿ (*x* $\circ_c$ $\beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* (*B* \ (*A*, *m*)) $\circ_c$
swap (*A* ∐ (*B* \ (*A*, *m*))) (X^A) $\circ_c$ ⟨*try-cast m* $\circ_c$ *m* $\circ_c$ *a*, *h* $\circ_c$ *z*⟩

using *cfunc-cross-prod-comp-cfunc-prod id-left-unit2* **by** (*typecheck-cfuncs-prems*, *smt*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* ($B \setminus (A,\ m)$) $\circ_c$
swap ($A \coprod (B \setminus (A,\ m))$) ($X^A$) $\circ_c$ $\langle$(*try-cast m* $\circ_c$ *m*) $\circ_c$ *a*, *g* $\circ_c$ *z*$\rangle$
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* ($B \setminus (A,\ m)$) $\circ_c$
swap ($A \coprod (B \setminus (A,\ m))$) ($X^A$) $\circ_c$ $\langle$(*try-cast m* $\circ_c$ *m*) $\circ_c$ *a*, *h* $\circ_c$ *z*$\rangle$
by (*typecheck-cfuncs*, *auto simp add*: *comp-associative2*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* ($B \setminus (A,\ m)$) $\circ_c$
swap ($A \coprod (B \setminus (A,\ m))$) ($X^A$) $\circ_c$ $\langle$*left-coproj A* ($B \setminus (A,m)$) $\circ_c$ *a*, *g* $\circ_c$ *z*$\rangle$
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* ($B \setminus (A,\ m)$) $\circ_c$
swap ($A \coprod (B \setminus (A,\ m))$) ($X^A$) $\circ_c$ $\langle$*left-coproj A* ($B \setminus (A,m)$) $\circ_c$ *a*, *h* $\circ_c$ *z*$\rangle$
using *m-def*(*2*) *try-cast-m-m* **by** (*typecheck-cfuncs*, *auto*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* ($B \setminus (A,\ m)$) $\circ_c$ $\langle$*g* $\circ_c$ *z*, *left-coproj A* ($B \setminus (A,m)$) $\circ_c$ *a*$\rangle$
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
dist-prod-coprod-left (X^A) *A* ($B \setminus (A,\ m)$) $\circ_c$ $\langle$*h* $\circ_c$ *z*, *left-coproj A* ($B \setminus (A,m)$) $\circ_c$ *a*$\rangle$
using *swap-ap* **by** (*typecheck-cfuncs*, *auto*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
left-coproj ($X^A \times_c A$) ($X^A \times_c (B \setminus (A,m))$) $\circ_c$ $\langle$*g* $\circ_c$ *z*, *a*$\rangle$
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
left-coproj ($X^A \times_c A$) ($X^A \times_c (B \setminus (A,m))$) $\circ_c$ $\langle$*h* $\circ_c$ *z*,*a*$\rangle$
using *dist-prod-coprod-left-ap-left* **by** (*typecheck-cfuncs*, *auto*)
then have ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
left-coproj ($X^A \times_c A$) ($X^A \times_c (B \setminus (A,m))$)) $\circ_c$ $\langle$*g* $\circ_c$ *z*, *a*$\rangle$
= ((*eval-func X A* $\circ_c$ *swap* (X^A) *A*) II ($x \circ_c \beta_{X^A \times_c (B \setminus (A,\ m))}$) $\circ_c$
left-coproj ($X^A \times_c A$) ($X^A \times_c (B \setminus (A,m))$)) $\circ_c$ $\langle$*h* $\circ_c$ *z*,*a*$\rangle$
by (*typecheck-cfuncs-prems*, *auto simp add*: *comp-associative2*)
then have (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\circ_c$ $\langle$*g* $\circ_c$ *z*, *a*$\rangle$
= (*eval-func X A* $\circ_c$ *swap* (X^A) *A*) $\circ_c$ $\langle$*h* $\circ_c$ *z*,*a*$\rangle$
by (*typecheck-cfuncs-prems*, *auto simp add*: *left-coproj-cfunc-coprod*)

then have *eval-func X A* $\circ_c$ *swap* (X^A) *A* $\circ_c$ $\langle g \circ_c z,\ a\rangle$
= *eval-func X A* $\circ_c$ *swap* (X^A) *A* $\circ_c$ $\langle h \circ_c z{,}a\rangle$
by (*typecheck-cfuncs-prems*, *auto simp add*: *comp-associative2*)
then have *eval-func X A* $\circ_c$ $\langle a,\ g \circ_c z\rangle$ = *eval-func X A* $\circ_c$ $\langle a,\ h \circ_c z\rangle$
by (*typecheck-cfuncs-prems*, *auto simp add*: *swap-ap*)
then have *eval-func X A* $\circ_c$ (*id A* $\times_f$ *g*) $\circ_c$ $\langle a,\ z\rangle$ = *eval-func X A* $\circ_c$ (*id A* $\times_f$ *h*) $\circ_c$ $\langle a,\ z\rangle$
by (*typecheck-cfuncs*, *simp add*: *cfunc-cross-prod-comp-cfunc-prod id-left-unit2*)
then show (*eval-func X A* $\circ_c$ *id*$_c$ *A* $\times_f$ *g*) $\circ_c$ *az* = (*eval-func X A* $\circ_c$ *id*$_c$ *A* $\times_f$ *h*) $\circ_c$ *az*
unfolding *az-def* **by** (*typecheck-cfuncs-prems*, *auto simp add*: *comp-associative2*)
qed
qed
qed
qed
qed

lemma *exp-preserves-card2*:
assumes $A \leq_c B$
shows $A^X \leq_c B^X$
unfolding *is-smaller-than-def*
proof −
obtain *m* **where** *m-def*[*type-rule*]: $m : A \to B$ *monomorphism m*
using *assms* **unfolding** *is-smaller-than-def* **by** *auto*
show $\exists\, m.\ m : A^X \to B^X \land$ *monomorphism m*
proof (*intro exI*[**where** *x*=(*m* $\circ_c$ *eval-func A X*)$^\sharp$], *safe*)
show (*m* $\circ_c$ *eval-func A X*)$^\sharp$: $A^X \to B^X$
by *typecheck-cfuncs*
then show *monomorphism*((*m* $\circ_c$ *eval-func A X*)$^\sharp$)
proof (*unfold monomorphism-def3*, *clarify*)
fix *g h Z*
assume *g-type*[*type-rule*]: $g : Z \to A^X$
assume *h-type*[*type-rule*]: $h : Z \to A^X$

assume *eq*: (*m* $\circ_c$ *eval-func A X*)$^\sharp$ $\circ_c$ *g* = (*m* $\circ_c$ *eval-func A X*)$^\sharp$ $\circ_c$ *h*
show *g* = *h*
proof (*typecheck-cfuncs*, *rule same-evals-equal*[**where** *Z*=*Z*, **where** *A*=*X*, **where** *X*=*A*], *clarify*)
have ((*eval-func B X*) $\circ_c$ (*id X* $\times_f$ (*m* $\circ_c$ *eval-func A X*)$^\sharp$)) $\circ_c$ (*id X* $\times_f$ *g*) =
((*eval-func B X*) $\circ_c$ (*id X* $\times_f$ (*m* $\circ_c$ *eval-func A X*)$^\sharp$)) $\circ_c$ (*id X* $\times_f$ *h*)
by (*typecheck-cfuncs*, *smt comp-associative2 eq inv-transpose-func-def3 inv-transpose-of-composition*)
then have (*m* $\circ_c$ *eval-func A X*) $\circ_c$ (*id X* $\times_f$ *g*) = (*m* $\circ_c$ *eval-func A X*) $\circ_c$ (*id X* $\times_f$ *h*)
by (*smt comp-type eval-func-type m-def*(*1*) *transpose-func-def*)
then have *m* $\circ_c$ (*eval-func A X* $\circ_c$ (*id X* $\times_f$ *g*)) = *m* $\circ_c$ (*eval-func A X* $\circ_c$ (*id X* $\times_f$ *h*))

```
          by (typecheck-cfuncs, smt comp-associative2)
        then have eval-func A X ∘c (id X ×f g) = eval-func A X ∘c (id X ×f h)
          using m-def monomorphism-def3 by (typecheck-cfuncs, blast)
        then show (eval-func A X ∘c (id X ×f g)) = (eval-func A X ∘c (id X ×f h))
          by (typecheck-cfuncs, smt comp-associative2)
      qed
    qed
  qed
qed

lemma exp-preserves-card3:
  assumes A ≤c B
  assumes X ≤c Y
  assumes nonempty(X)
  shows X^A ≤c Y^B
proof –
  have leq1: X^A ≤c X^B
    by (simp add: assms(1,3) exp-preserves-card1)
  have leq2: X^B ≤c Y^B
    by (simp add: assms(2) exp-preserves-card2)
  show X^A ≤c Y^B
    using leq1 leq2 set-card-transitive by blast
qed

end
```

18 Countable Sets

```
theory Countable
  imports Nats Axiom-Of-Choice Nat-Parity Cardinality
begin
```

The definition below corresponds to Definition 2.6.9 in Halvorson.

```
definition epi-countable :: cset ⇒ bool where
  epi-countable X ⟷ (∃ f. f : ℕc → X ∧ epimorphism f)

lemma emptyset-is-not-epi-countable:
  ¬ epi-countable ∅
  using comp-type emptyset-is-empty epi-countable-def zero-type by blast
```

The fact that the empty set is not countable according to the definition from Halvorson (*epi-countable ?X* $= (\exists f.\ f : \mathbb{N}_c \to ?X \wedge$ *epimorphism* $f)$) motivated the following definition.

```
definition countable :: cset ⇒ bool where
  countable X ⟷ (∃ f. f : X → ℕc ∧ monomorphism f)
```

lemma *epi-countable-is-countable*:
assumes *epi-countable X*
shows *countable X*
using *assms countable-def epi-countable-def epis-give-monos* **by** *blast*

lemma *emptyset-is-countable*:
countable $\emptyset$
using *countable-def empty-subset subobject-of-def2* **by** *blast*

lemma *natural-numbers-are-countably-infinite*:
countable $\mathbb{N}_c \wedge$ *is-infinite* $\mathbb{N}_c$
by (*meson CollectI Peano's-Axioms countable-def injective-imp-monomorphism is-infinite-def successor-type*)

lemma *iso-to-N-is-countably-infinite*:
assumes $X \cong \mathbb{N}_c$
shows *countable X* $\wedge$ *is-infinite X*
by (*meson assms countable-def is-isomorphic-def is-smaller-than-def iso-imp-epi-and-monic isomorphic-is-symmetric larger-than-infinite-is-infinite natural-numbers-are-countably-infinite*)

lemma *smaller-than-countable-is-countable*:
assumes $X \leq_c Y$ *countable Y*
shows *countable X*
by (*smt assms cfunc-type-def comp-type composition-of-monic-pair-is-monic countable-def is-smaller-than-def*)

lemma *iso-pres-countable*:
assumes $X \cong Y$ *countable Y*
shows *countable X*
using *assms is-isomorphic-def is-smaller-than-def iso-imp-epi-and-monic smaller-than-countable-is-countable* **by** *blast*

lemma *NuN-is-countable*:
countable$(\mathbb{N}_c \coprod \mathbb{N}_c)$
using *countable-def epis-give-monos halve-with-parity-iso halve-with-parity-type iso-imp-epi-and-monic* **by** *smt*

The lemma below corresponds to Exercise 2.6.11 in Halvorson.

lemma *coproduct-of-countables-is-countable*:
assumes *countable X countable Y*
shows *countable*$(X \coprod Y)$
unfolding *countable-def*
proof−
obtain *x* **where** *x-def*: $x : X \to \mathbb{N}_c \wedge$ *monomorphism x*
using *assms*(*1*) *countable-def* **by** *blast*
obtain *y* **where** *y-def*: $y : Y \to \mathbb{N}_c \wedge$ *monomorphism y*
using *assms*(*2*) *countable-def* **by** *blast*
obtain *n* **where** *n-def*: $n : \mathbb{N}_c \coprod \mathbb{N}_c \to \mathbb{N}_c \wedge$ *monomorphism n*
using *NuN-is-countable countable-def* **by** *blast*

have *xy-type*: $x \bowtie_f y : X \coprod Y \to \mathbb{N}_c \coprod \mathbb{N}_c$
using *x-def y-def* **by** (*typecheck-cfuncs*, *auto*)
then have *nxy-type*: $n \circ_c (x \bowtie_f y) : X \coprod Y \to \mathbb{N}_c$
using *comp-type n-def* **by** *blast*
have *injective*$(x \bowtie_f y)$
using *cfunc-bowtieprod-inj monomorphism-imp-injective x-def y-def* **by** *blast*
then have *monomorphism*$(x \bowtie_f y)$
using *injective-imp-monomorphism* **by** *auto*
then have *monomorphism*$(n \circ_c (x \bowtie_f y))$
using *cfunc-type-def composition-of-monic-pair-is-monic n-def xy-type* **by** *auto*
then show $\exists f.\ f : X \coprod Y \to \mathbb{N}_c \wedge$ *monomorphism* f
using *nxy-type* **by** *blast*
qed

end

19 Fixed Points and Cantor's Theorems

theory *Fixed-Points*
imports *Axiom-Of-Choice Pred-Logic Cardinality*
begin

The definitions below correspond to Definition 2.6.12 in Halvorson.

definition *fixed-point* :: *cfunc* ⇒ *cfunc* ⇒ *bool* **where**
fixed-point a g $\longleftrightarrow (\exists\ A.\ g : A \to A \wedge a \in_c A \wedge g \circ_c a = a)$
definition *has-fixed-point* :: *cfunc* ⇒ *bool* **where**
has-fixed-point g $\longleftrightarrow (\exists\ a.$ *fixed-point a g*$)$
definition *fixed-point-property* :: *cset* ⇒ *bool* **where**
fixed-point-property A $\longleftrightarrow (\forall\ g.\ g : A \to A \longrightarrow$ *has-fixed-point g*$)$

lemma *fixed-point-def2*:
assumes $g : A \to A$ $a \in_c A$
shows *fixed-point a g* $= (g \circ_c a = a)$
unfolding *fixed-point-def* **using** *assms* **by** *blast*

The lemma below corresponds to Theorem 2.6.13 in Halvorson.

lemma *Lawveres-fixed-point-theorem*:
assumes *p-type*[*type-rule*]: $p : X \to A^X$
assumes *p-surj*: *surjective p*
shows *fixed-point-property A*
unfolding *fixed-point-property-def has-fixed-point-def*
proof(*clarify*)
fix *g*
assume *g-type*[*type-rule*]: $g : A \to A$
obtain φ **where** φ*-def*: $\varphi = p^\flat$
by *auto*
then have φ*-type*[*type-rule*]: $\varphi : X \times_c X \to A$
by (*simp add*: *flat-type p-type*)
obtain *f* **where** *f-def*: $f = g \circ_c \varphi \circ_c$ *diagonal*(X)

by *auto*
then have *f-type*[*type-rule*]:$f : X \to A$
using φ*-type comp-type diagonal-type f-def g-type* **by** *blast*
obtain *x-f* **where** *x-f*: *metafunc f* $=$ *p* $\circ_c$ *x-f* **and** *x-f-type*[*type-rule*]: *x-f* $\in_c X$
using *assms* **by** (*typecheck-cfuncs*, *metis p-surj surjective-def2*)
have $\varphi_{[-,x\text{-}f]} = f$
proof(*etcs-rule one-separator*)
fix x
assume *x-type*[*type-rule*]: $x \in_c X$
have $\varphi_{[-,x\text{-}f]} \circ_c x = \varphi \circ_c \langle x,\ x\text{-}f\rangle$
by (*typecheck-cfuncs*, *meson right-param-on-el x-f*)
also have ... $= ((\textit{eval-func } A\ X) \circ_c (\textit{id } X \times_f p)) \circ_c \langle x,\ x\text{-}f\rangle$
using *assms* φ*-def inv-transpose-func-def3* **by** *auto*
also have ... $= (\textit{eval-func } A\ X) \circ_c (\textit{id } X \times_f p) \circ_c \langle x,\ x\text{-}f\rangle$
by (*typecheck-cfuncs*, *metis comp-associative2*)
also have ... $= (\textit{eval-func } A\ X) \circ_c \langle \textit{id } X \circ_c x,\ p \circ_c x\text{-}f\rangle$
using *cfunc-cross-prod-comp-cfunc-prod x-f* **by** (*typecheck-cfuncs*, *force*)
also have ... $= (\textit{eval-func } A\ X) \circ_c \langle x,\ \textit{metafunc } f\rangle$
using *id-left-unit2 x-f* **by** (*typecheck-cfuncs*, *auto*)
also have ... $= f \circ_c x$
by (*simp add*: *eval-lemma f-type x-type*)
finally show $\varphi_{[-,x\text{-}f]} \circ_c x = f \circ_c x$.
qed
then have $\varphi_{[-,x\text{-}f]} \circ_c x\text{-}f = g \circ_c \varphi \circ_c \textit{diagonal}(X) \circ_c x\text{-}f$
by (*typecheck-cfuncs*, *smt* (*z3*) *cfunc-type-def comp-associative domain-comp f-def x-f*)
then have $\varphi \circ_c \langle x\text{-}f,\ x\text{-}f\rangle = g \circ_c \varphi \circ_c \langle x\text{-}f,\ x\text{-}f\rangle$
using *diag-on-elements right-param-on-el x-f* **by** (*typecheck-cfuncs*, *auto*)
then have *fixed-point* $(\varphi \circ_c \langle x\text{-}f,\ x\text{-}f\rangle)\ g$
using *fixed-point-def2* **by** (*typecheck-cfuncs*, *auto*)
then show $\exists\, a.$ *fixed-point a g*
using *fixed-point-def* **by** *auto*
qed

The theorem below corresponds to Theorem 2.6.14 in Halvorson.

theorem *Cantors-Negative-Theorem*:
$\nexists\ s.\ s : X \to \mathcal{P}\ X \wedge \textit{surjective } s$
proof(*rule ccontr*, *clarify*)
fix s
assume *s-type*: $s : X \to \mathcal{P}\ X$
assume *s-surj*: *surjective s*
then have *Omega-has-ffp*: *fixed-point-property* Ω
using *Lawveres-fixed-point-theorem powerset-def s-type* **by** *auto*
have *Omega-doesnt-have-ffp*: $\neg$(*fixed-point-property* Ω)
unfolding *fixed-point-property-def has-fixed-point-def fixed-point-def*
proof
assume *BWOC*: $\forall\, g.\ g : \Omega \to \Omega \longrightarrow (\exists\, a\ A.\ g : A \to A \wedge a \in_c A \wedge g \circ_c a = a)$
have $NOT : \Omega \to \Omega \wedge (\forall\, a.\ \forall\, A.\ a \in_c A \longrightarrow NOT : A \to A \longrightarrow NOT \circ_c a$

```
≠ a ∨ ¬ a ∈c Ω)
    by (typecheck-cfuncs, metis AND-complementary AND-idempotent OR-complementary
OR-idempotent true-false-distinct)
    then have ∃ g. g : Ω → Ω ∧ (∀ a. ∀ A. a ∈c A ⟶ g : A → A ⟶ g ∘c a ≠ a)
      by (metis cfunc-type-def)
    then show False
      using BWOC by presburger
  qed
  show False
    using Omega-doesnt-have-ffp Omega-has-ffp by auto
qed
```

The theorem below corresponds to Exercise 2.6.15 in Halvorson.

```
theorem Cantors-Positive-Theorem:
  ∃ m. m : X → Ω^X ∧ injective m
proof −
  have eq-pred-sharp-type[type-rule]: eq-pred X♯ : X →  Ω^X
    by typecheck-cfuncs
  have injective(eq-pred X♯)
    unfolding injective-def
  proof (clarify)
    fix x y
    assume x ∈c domain (eq-pred X♯) then have x-type[type-rule]: x ∈c X
      using cfunc-type-def eq-pred-sharp-type by auto
    assume y ∈c domain (eq-pred X♯) then have y-type[type-rule]:y ∈c X
      using cfunc-type-def eq-pred-sharp-type by auto
    assume eq: eq-pred X♯ ∘c x = eq-pred X♯ ∘c y
    have eq-pred X ∘c ⟨x, x⟩ = eq-pred X ∘c ⟨x, y⟩
    proof −
      have eq-pred X ∘c ⟨x, x⟩ = ((eval-func Ω X) ∘c (id X ×f (eq-pred X♯)) ) ∘c
⟨x, x⟩
        using transpose-func-def by (typecheck-cfuncs, presburger)
      also have ... = (eval-func Ω X) ∘c (id X ×f (eq-pred X♯)) ∘c ⟨x, x⟩
        by (typecheck-cfuncs, simp add: comp-associative2)
      also have ... = (eval-func Ω X) ∘c ⟨id X ∘c x, (eq-pred X♯) ∘c x⟩
        using cfunc-cross-prod-comp-cfunc-prod by (typecheck-cfuncs, force)
      also have ... = (eval-func Ω X) ∘c ⟨id X ∘c x, (eq-pred X♯) ∘c y⟩
        by (simp add: eq)
      also have ... = (eval-func Ω X) ∘c (id X ×f (eq-pred X♯)) ∘c ⟨x, y⟩
        by (typecheck-cfuncs, simp add: cfunc-cross-prod-comp-cfunc-prod)
      also have ... = ((eval-func Ω X) ∘c (id X ×f (eq-pred X♯)) ) ∘c ⟨x, y⟩
        using comp-associative2 by (typecheck-cfuncs, blast)
      also have ... = eq-pred X ∘c ⟨x, y⟩
        using transpose-func-def by (typecheck-cfuncs, presburger)
      finally show ?thesis.
    qed
    then show x = y
      by (metis eq-pred-iff-eq x-type y-type)
  qed
```

then show $\exists\, m.\ m : X \to \Omega^X \wedge$ *injective m*
  using *eq-pred-sharp-type injective-imp-monomorphism* **by** *blast*
qed

The corollary below corresponds to Corollary 2.6.16 in Halvorson.

corollary
  $X \leq_c \mathcal{P}\ X \wedge \neg\ (X \cong \mathcal{P}\ X)$
  using *Cantors-Negative-Theorem Cantors-Positive-Theorem*
  unfolding *is-smaller-than-def is-isomorphic-def powerset-def*
  by (*metis epi-is-surj injective-imp-monomorphism iso-imp-epi-and-monic*)

corollary *Generalized-Cantors-Positive-Theorem*:
  assumes ¬ *terminal-object Y*
  assumes ¬ *initial-object Y*
  shows $X\ \leq_c\ Y^X$
proof −
  have $\Omega \leq_c Y$
    by (*simp add*: *assms non-init-non-ter-sets*)
  then have *fact*: $\Omega^X \leq_c\ \ Y^X$
    by (*simp add*: *exp-preserves-card2*)
  have $X \leq_c \Omega^X$
    by (*meson Cantors-Positive-Theorem CollectI injective-imp-monomorphism is-smaller-than-def*)
  then show *?thesis*
    using *fact set-card-transitive* **by** *blast*
qed

corollary *Generalized-Cantors-Negative-Theorem*:
  assumes ¬ *initial-object X*
  assumes ¬ *terminal-object Y*
  shows $\nexists\ s.\ s : X \to Y^X \wedge$ *surjective s*
proof(*rule ccontr*, *clarify*)
  fix *s*
  assume *s-type*: $s : X \to Y^X$
  assume *s-surj*: *surjective s*
  obtain *m* **where** *m-type*: $m :\ Y^X \to X$ **and** *m-mono*: *monomorphism*(*m*)
    by (*meson epis-give-monos s-surj s-type surjective-is-epimorphism*)
  have *nonempty X*
    using *is-empty-def assms*(*1*) *iso-empty-initial no-el-iff-iso-empty nonempty-def*
by *blast*

  then have *nonempty*: *nonempty* (Ω^X)
    using *nonempty-def nonempty-to-nonempty true-func-type* **by** *blast*
  show *False*
  proof(*cases initial-object Y*)
    assume *initial-object Y*
    then have $Y^X \cong \emptyset$
    by (*simp add*: ‹*nonempty X*› *empty-to-nonempty initial-iso-empty no-el-iff-iso-empty*)

then show *False*
by (*meson is-empty-def assms*(*1*) *comp-type iso-empty-initial no-el-iff-iso-empty s-type*)
next
assume ¬ *initial-object* Y
then have $\Omega \leq_c Y$
by (*simp add*: *assms*(*2*) *non-init-non-ter-sets*)
then obtain n **where** *n-type*: $n : \Omega^X \to Y^X$ **and** *n-mono*: *monomorphism*(n)
by (*meson exp-preserves-card2 is-smaller-than-def*)
then have *mn-type*: $m \circ_c n : \Omega^X \to X$
by (*meson comp-type m-type*)
have *mn-mono*: *monomorphism*($m \circ_c n$)
using *cfunc-type-def composition-of-monic-pair-is-monic m-mono m-type n-mono n-type* **by** *presburger*
then have $\exists\, g.\ g\colon X \to \Omega^X \wedge$ *epimorphism*(g) $\wedge\ g \circ_c (m \circ_c n) = id\ (\Omega^X)$
by (*simp add*: *mn-type monos-give-epis nonempty*)
then show *False*
by (*metis Cantors-Negative-Theorem epi-is-surj powerset-def*)
qed
qed

end
theory *ETCS*
imports *Axiom-Of-Choice Nats Quant-Logic Countable Fixed-Points*
begin
end

References

[1] H. Halvorson. *The Logic in Philosophy of Science*. Cambridge University Press, 2019.