The Elementary Theory of the Category of Sets

James Baxter   Dustin Bryant

October 10, 2024

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
$domain :: cfunc \Rightarrow cset$ **and**
$codomain :: cfunc \Rightarrow cset$ **and**
$comp :: cfunc \Rightarrow cfunc \Rightarrow cfunc$ (**infixr** $\circ_c$ *55*) **and**
$id :: cset \Rightarrow cfunc$ (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
$(f : X \to Y) \longleftrightarrow (\mathit{domain}\ f = X \land \mathit{codomain}\ f = Y)$

lemma *comp-type*:
$f : X \to Y \Longrightarrow g : Y \to Z \Longrightarrow g \circ_c f : X \to Z$
⟨*proof*⟩

lemma *comp-associative2*:
$f : X \to Y \Longrightarrow g : Y \to Z \Longrightarrow h : Z \to W \Longrightarrow h \circ_c (g \circ_c f) = (h \circ_c g) \circ_c f$
⟨*proof*⟩

lemma *id-type*: $\mathit{id}\ X : X \to X$
⟨*proof*⟩

lemma *id-right-unit2*: $f : X \to Y \Longrightarrow f \circ_c \mathit{id}\ X = f$
⟨*proof*⟩

lemma *id-left-unit2*: $f : X \to Y \Longrightarrow \mathit{id}\ Y \circ_c f = f$
⟨*proof*⟩

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*⟩

1.1.1 typecheck_cfuncs: Tactic to Construct Type Facts

⟨*ML*⟩

1.1.2 etcs_rule: Tactic to Apply Rules with ETCS Typechecking

⟨*ML*⟩

1.1.3 etcs_subst: Tactic to Apply Substitutions with ETCS Typechecking

⟨*ML*⟩

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

⟨*ML*⟩

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

⟨*ML*⟩

1.2 Monomorphisms, Epimorphisms and Isomorphisms

1.2.1 Monomorphisms

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

lemma *monomorphism-def2*:
$monomorphism\ f \longleftrightarrow (\forall\ g\ h\ A\ X\ Y.\ g : A \to X \wedge h : A \to X \wedge f : X \to Y \longrightarrow (f \circ_c g = f \circ_c h \longrightarrow g = h))$
⟨*proof*⟩

lemma *monomorphism-def3*:
assumes $f : X \to Y$
shows $monomorphism\ f \longleftrightarrow (\forall\ g\ h\ A.\ g : A \to X \wedge h : A \to X \longrightarrow (f \circ_c g = f \circ_c h \longrightarrow g = h))$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7a in Halvorson.

lemma *comp-monic-imp-monic*:
assumes $domain\ g = codomain\ f$
shows $monomorphism\ (g \circ_c f) \Longrightarrow monomorphism\ f$
⟨*proof*⟩

lemma *comp-monic-imp-monic'*:
assumes $f : X \to Y\ g : Y \to Z$
shows $monomorphism\ (g \circ_c f) \Longrightarrow monomorphism\ f$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7c in Halvorson.

lemma *composition-of-monic-pair-is-monic*:
assumes $codomain\ f = domain\ g$
shows $monomorphism\ f \Longrightarrow monomorphism\ g \Longrightarrow monomorphism\ (g \circ_c f)$
⟨*proof*⟩

1.2.2 Epimorphisms

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

lemma *epimorphism-def2*:
epimorphism f ⟷ (∀ *g h A X Y*. $f : X \to Y \land g : Y \to A \land h : Y \to A$ ⟶ ($g \circ_c f = h \circ_c f$ ⟶ $g = h$))
⟨*proof*⟩

lemma *epimorphism-def3*:
assumes $f : X \to Y$
shows *epimorphism f* ⟷ (∀ *g h A*. $g : Y \to A \land h : Y \to A$ ⟶ ($g \circ_c f = h \circ_c f$ ⟶ $g = h$))
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7b in Halvorson.

lemma *comp-epi-imp-epi*:
assumes *domain g* = *codomain f*
shows *epimorphism* ($g \circ_c f$) ⟹ *epimorphism g*
⟨*proof*⟩

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* ⟹ *epimorphism g* ⟹ *epimorphism* ($g \circ_c f$)
⟨*proof*⟩

1.2.3 Isomorphisms

definition *isomorphism* :: *cfunc* ⇒ *bool* **where**
isomorphism f ⟷ (∃ *g*. *domain g* = *codomain f* ∧ *codomain g* = *domain f* ∧
$g \circ_c f$ = *id*(*domain f*) ∧ $f \circ_c g$ = *id*(*domain g*))

lemma *isomorphism-def2*:
isomorphism f ⟷ (∃ *g X Y*. $f : X \to Y \land g : Y \to X \land g \circ_c f = id\ X \land f \circ_c g = id\ Y$)
⟨*proof*⟩

lemma *isomorphism-def3*:
assumes $f : X \to Y$
shows *isomorphism f* ⟷ (∃ *g*. $g : Y \to X \land g \circ_c f = id\ X \land f \circ_c g = id\ Y$)
⟨*proof*⟩

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

lemma *inverse-def2*:
  assumes *isomorphism f*
  shows f^{-1} : *codomain f* $\to$ *domain f* $\wedge$ $f^{-1} \circ_c f$ = *id*(*domain f*) $\wedge$ $f \circ_c f^{-1}$ = *id*(*codomain f*)
  ⟨*proof*⟩

lemma *inverse-type*[*type-rule*]:
  assumes *isomorphism f f* : $X \to Y$
  shows f^{-1} : $Y \to X$
  ⟨*proof*⟩

lemma *inv-left*:
  assumes *isomorphism f f* : $X \to Y$
  shows $f^{-1} \circ_c f$ = *id X*
  ⟨*proof*⟩

lemma *inv-right*:
  assumes *isomorphism f f* : $X \to Y$
  shows $f \circ_c f^{-1}$ = *id Y*
  ⟨*proof*⟩

lemma *inv-iso*:
  assumes *isomorphism f*
  shows *isomorphism*(f^{-1})
  ⟨*proof*⟩

lemma *inv-idempotent*:
  assumes *isomorphism f*
  shows $(f^{-1})^{-1} = f$
  ⟨*proof*⟩

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

lemma *id-isomorphism*: *isomorphism* (*id X*)
  ⟨*proof*⟩

lemma *isomorphic-is-reflexive*: $X \cong X$
  ⟨*proof*⟩

lemma *isomorphic-is-symmetric*: $X \cong Y \longrightarrow Y \cong X$
  ⟨*proof*⟩

lemma *isomorphism-comp*:
  domain f = *codomain g* $\Longrightarrow$ *isomorphism f* $\Longrightarrow$ *isomorphism g* $\Longrightarrow$ *isomorphism* $(f \circ_c g)$
  ⟨*proof*⟩

lemma *isomorphism-comp'*:

assumes $f : Y \to Z$ $g : X \to Y$
shows *isomorphism* $f \Longrightarrow$ *isomorphism* $g \Longrightarrow$ *isomorphism* $(f \circ_c g)$
⟨*proof*⟩

lemma *isomorphic-is-transitive*: $(X \cong Y \wedge Y \cong Z) \longrightarrow X \cong Z$
⟨*proof*⟩

lemma *is-isomorphic-equiv*:
equiv UNIV $\{(X, Y). X \cong Y\}$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7e in Halvorson.

lemma *iso-imp-epi-and-monic*:
isomorphism $f \Longrightarrow$ *epimorphism* $f \wedge$ *monomorphism* f
⟨*proof*⟩

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*⟩

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* $\Rightarrow$ *cfunc* $\Rightarrow$ *cfunc* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *bool* **where**
is-cart-prod W π_0 π_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 π_0 π_1 X Y $\longleftrightarrow$
$(\forall\ f\ g\ Z.\ (f : Z \to X \wedge g : Z \to Y) \longrightarrow$
$(\exists\ h.\ h : Z \to W \wedge \pi_0 \circ_c h = f \wedge \pi_1 \circ_c h = g \wedge$
$(\forall\ h2.\ (h2 : Z \to W \wedge \pi_0 \circ_c h2 = f \wedge \pi_1 \circ_c h2 = g) \longrightarrow h2 = h)))$
⟨*proof*⟩

abbreviation *is-cart-prod-triple* :: *cset* $\times$ *cfunc* $\times$ *cfunc* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *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
⟨*proof*⟩

The lemma below corresponds to Proposition 2.1.8 in Halvorson.

lemma *cart-prods-isomorphic*:
assumes *W-cart-prod*: *is-cart-prod-triple* (W, π_0, π_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*⟩

lemma *product-commutes*:
$A \times_c B \cong B \times_c A$
⟨*proof*⟩

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)
⟨*proof*⟩

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 $a = b$

⟨*proof*⟩

lemma *cart-prod-eq2*:
assumes $a : Z \to X$ $b : Z \to Y$ $c : Z \to X$ $d : Z \to Y$
shows $\langle a, b\rangle = \langle c,d\rangle \longleftrightarrow (a = c \wedge b = d)$
⟨*proof*⟩

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*⟩

2.1 Diagonal Functions

The definition below corresponds to Definition 2.1.9 in Halvorson.

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

lemma *diagonal-type*[*type-rule*]:
$diagonal\ X : X \to X \times_c X$
⟨*proof*⟩

lemma *diag-mono*:
monomorphism(*diagonal X*)
⟨*proof*⟩

2.2 Products of Functions

The definition below corresponds to Definition 2.1.10 in Halvorson.

definition *cfunc-cross-prod* :: *cfunc* ⇒ *cfunc* ⇒ *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*)⟩

lemma *cfunc-cross-prod-def2*:
assumes $f : X \to Y$ $g : V \to W$
shows $f \times_f g = \langle f \circ_c$ *left-cart-proj X V*, $g \circ_c$ *right-cart-proj X V*⟩
⟨*proof*⟩

lemma *cfunc-cross-prod-type*[*type-rule*]:
$f : W \to Y \Longrightarrow g : X \to Z \Longrightarrow f \times_f g : W \times_c X \to Y \times_c Z$
⟨*proof*⟩

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*
⟨*proof*⟩

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* $W\ X$
⟨*proof*⟩

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* $W\ X \Longrightarrow h = f \times_f g$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

The lemma below corresponds to Exercise 2.1.12 in Halvorson.

lemma *id-cross-prod*: $id(X) \times_f id(Y) = id(X \times_c Y)$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.14 in Halvorson.

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

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*⟩

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* $(f \times_f g)$
⟨*proof*⟩

2.3 Useful Cartesian Product Permuting Functions

2.3.1 Swapping a Cartesian Product

definition *swap* :: *cset* ⇒ *cset* ⇒ *cfunc* **where**
$swap\ X\ Y = \langle right\text{-}cart\text{-}proj\ X\ Y,\ left\text{-}cart\text{-}proj\ X\ Y\rangle$

lemma *swap-type*[*type-rule*]: $swap\ X\ Y : X \times_c Y \to Y \times_c X$
⟨*proof*⟩

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*⟩

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*⟩

lemma *swap-idempotent*:
$swap\ Y\ X \circ_c swap\ X\ Y = id\ (X \times_c Y)$
⟨*proof*⟩

lemma *swap-mono*:
$monomorphism(swap\ X\ Y)$
⟨*proof*⟩

2.3.2 Permuting a Cartesian Product to Associate to the Right

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

lemma *associate-right-type*[*type-rule*]: $associate\text{-}right\ X\ Y\ Z : (X \times_c Y) \times_c Z \to X \times_c (Y \times_c Z)$
⟨*proof*⟩

lemma *associate-right-ap*:
assumes $x : A \to X\ y : A \to Y\ z : A \to Z$
shows $associate\text{-}right\ X\ Y\ Z \circ_c \langle\langle x,\ y\rangle,\ z\rangle = \langle x,\ \langle y,\ z\rangle\rangle$
⟨*proof*⟩

lemma *associate-right-crossprod-ap*:

assumes $x : A \to X$ $y : B \to Y$ $z : C \to Z$
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*⟩

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 =
⟨
⟨
left-cart-proj X $(Y \times_c Z)$,
left-cart-proj Y Z $\circ_c$ *right-cart-proj* X $(Y \times_c Z)$
⟩,
right-cart-proj Y Z $\circ_c$ *right-cart-proj* X $(Y \times_c Z)$
⟩

lemma *associate-left-type*[*type-rule*]: *associate-left* X Y Z : $X \times_c (Y \times_c Z) \to (X \times_c Y) \times_c Z$
⟨*proof*⟩

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*⟩

lemma *right-left*:
associate-right A B C $\circ_c$ *associate-left* A B C = *id* $(A \times_c (B \times_c C))$
⟨*proof*⟩

lemma *left-right*:
associate-left A B C $\circ_c$ *associate-right* A B C = *id* $((A \times_c B) \times_c C)$
⟨*proof*⟩

lemma *product-associates*:
$A \times_c (B \times_c C) \cong (A \times_c B) \times_c C$
⟨*proof*⟩

lemma *associate-left-crossprod-ap*:
assumes $x : A \to X$ $y : B \to Y$ $z : C \to Z$
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*⟩

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 $X\ Y\ Z : (X \times_c Y) \times_c Z \to X \times_c Z$
⟨*proof*⟩

lemma *distribute-right-left-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-right-left* $X\ Y\ Z \circ_c \langle\langle x, y\rangle, z\rangle = \langle x, z\rangle$
⟨*proof*⟩

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 $X\ Y\ Z : (X \times_c Y) \times_c Z \to Y \times_c Z$
⟨*proof*⟩

lemma *distribute-right-right-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-right-right* $X\ Y\ Z \circ_c \langle\langle x, y\rangle, z\rangle = \langle y, z\rangle$
⟨*proof*⟩

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 $X\ Y\ Z : (X \times_c Y) \times_c Z \to (X \times_c Z) \times_c (Y \times_c Z)$
⟨*proof*⟩

lemma *distribute-right-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-right* $X\ Y\ Z \circ_c \langle\langle x, y\rangle, z\rangle = \langle\langle x, z\rangle, \langle y, z\rangle\rangle$
⟨*proof*⟩

lemma *distribute-right-mono*:
monomorphism (*distribute-right* $X\ Y\ Z$)
⟨*proof*⟩

2.3.5 Distributing over a Cartesian Product from the Left

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

lemma *distribute-left-left-type*[*type-rule*]:
distribute-left-left $X\ Y\ Z : X \times_c (Y \times_c Z) \to X \times_c Y$
⟨*proof*⟩

lemma *distribute-left-left-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-left-left* $X\ Y\ Z \circ_c \langle x, \langle y, z\rangle\rangle = \langle x, y\rangle$
$\langle proof \rangle$

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

lemma *distribute-left-right-type*[*type-rule*]:
distribute-left-right $X\ Y\ Z : X \times_c (Y \times_c Z) \to X \times_c Z$
$\langle proof \rangle$

lemma *distribute-left-right-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-left-right* $X\ Y\ Z \circ_c \langle x, \langle y, z\rangle\rangle = \langle x, z\rangle$
$\langle proof \rangle$

definition *distribute-left* :: $cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**
distribute-left $X\ Y\ Z = \langle$*distribute-left-left* $X\ Y\ Z$, *distribute-left-right* $X\ Y\ Z\rangle$

lemma *distribute-left-type*[*type-rule*]:
distribute-left $X\ Y\ Z : X \times_c (Y \times_c Z) \to (X \times_c Y) \times_c (X \times_c Z)$
$\langle proof \rangle$

lemma *distribute-left-ap*:
assumes $x : A \to X$ $y : A \to Y$ $z : A \to Z$
shows *distribute-left* $X\ Y\ Z \circ_c \langle x, \langle y, z\rangle\rangle = \langle\langle x, y\rangle, \langle x, z\rangle\rangle$
$\langle proof \rangle$

lemma *distribute-left-mono*:
monomorphism (*distribute-left* $X\ Y\ Z$)
$\langle proof \rangle$

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* $A\ B\ C\ D : (A \times_c B) \times_c (C \times_c D) \to (A \times_c D)$
$\langle proof \rangle$

lemma *outers-apply*:
assumes $a : Z \to A$ $b : Z \to B$ $c : Z \to C$ $d : Z \to D$
shows *outers* $A\ B\ C\ D \circ_c \langle\langle a,b\rangle, \langle c,d\rangle\rangle = \langle a,d\rangle$

⟨*proof*⟩

definition *inners* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
inners A B C D = ⟨
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* : $(A \times_c B) \times_c (C \times_c D) \to (B \times_c C)$
⟨*proof*⟩

lemma *inners-apply*:
assumes $a : Z \to A$ $b : Z \to B$ $c : Z \to C$ $d : Z \to D$
shows *inners A B C D* $\circ_c$ ⟨⟨*a*,*b*⟩, ⟨*c*, *d*⟩⟩ = ⟨*b*,*c*⟩
⟨*proof*⟩

definition *lefts* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
lefts A B C D = ⟨
left-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 *lefts-type*[*type-rule*]: *lefts A B C D* : $(A \times_c B) \times_c (C \times_c D) \to (A \times_c C)$
⟨*proof*⟩

lemma *lefts-apply*:
assumes $a : Z \to A$ $b : Z \to B$ $c : Z \to C$ $d : Z \to D$
shows *lefts A B C D* $\circ_c$ ⟨⟨*a*,*b*⟩, ⟨*c*, *d*⟩⟩ = ⟨*a*,*c*⟩
⟨*proof*⟩

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* ($A \times_c B$) ($C \times_c D$)
⟩

lemma *rights-type*[*type-rule*]: *rights A B C D* : $(A \times_c B) \times_c (C \times_c D) \to (B \times_c D)$
⟨*proof*⟩

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* $\circ_c$ ⟨⟨*a*,*b*⟩, ⟨*c*, *d*⟩⟩ = ⟨*b*,*d*⟩
⟨*proof*⟩

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

lemma *one-separator-contrapos*:
  assumes $f : X \rightarrow Y$ $g : X \rightarrow Y$
  shows $f \neq g \Longrightarrow \exists\ x.\ x : \mathbf{1} \rightarrow X \wedge f \circ_c x \neq g \circ_c x$
  ⟨*proof*⟩

lemma *terminal-func-comp*:
  $x : X \rightarrow Y \Longrightarrow \beta_Y \circ_c x = \beta_X$
  ⟨*proof*⟩

lemma *terminal-func-comp-elem*:
  $x : \mathbf{1} \rightarrow X \Longrightarrow \beta_X \circ_c x = id\ \mathbf{1}$
  ⟨*proof*⟩

3.1 Set Membership and Emptiness

The abbreviation below captures Definition 2.1.16 in Halvorson.

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

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

definition *is-empty* :: *cset* $\Rightarrow$ *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
  ⟨*proof*⟩

lemma *one-unique-element*:
  $\exists!\ x.\ x \in_c \mathbf{1}$

⟨*proof*⟩

lemma *prod-with-empty-is-empty1*:
assumes *is-empty* (A)
shows *is-empty*$(A \times_c B)$
⟨*proof*⟩

lemma *prod-with-empty-is-empty2*:
assumes *is-empty* (B)
shows *is-empty* $(A \times_c B)$
⟨*proof*⟩

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*$(\mathbf{1})$
⟨*proof*⟩

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
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.15 in Halvorson.

lemma *terminal-objects-isomorphic*:
assumes *terminal-object* X *terminal-object* Y
shows $X \cong Y$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *iso-to-term-is-term*:
assumes $X \cong Y$
assumes *terminal-object* Y
shows *terminal-object* X
⟨*proof*⟩

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*⟩

3.3 Injectivity

The definition below corresponds to Definition 2.1.24 in Halvorson.

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

lemma *injective-def2*:
assumes $f : X \to Y$
shows *injective f* $\longleftrightarrow$ ($\forall$ x y. ($x \in_c X \wedge y \in_c X \wedge f \circ_c x = f \circ_c y$) $\longrightarrow x = y$)
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.26 in Halvorson.

lemma *monomorphism-imp-injective*:
monomorphism f $\Longrightarrow$ *injective f*
⟨*proof*⟩

The lemma below corresponds to Proposition 2.1.27 in Halvorson.

lemma *injective-imp-monomorphism*:
injective f $\Longrightarrow$ *monomorphism f*
⟨*proof*⟩

lemma *cfunc-cross-prod-inj*:
assumes *type-assms*: $f : X \to Y$ $g : Z \to W$
assumes *injective f* $\wedge$ *injective g*
shows *injective* ($f \times_f g$)
⟨*proof*⟩

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*
⟨*proof*⟩

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* ($f \times_f g$)
⟨*proof*⟩

3.4 Surjectivity

The definition below corresponds to Definition 2.1.28 in Halvorson.

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

lemma *surjective-def2*:
assumes $f : X \to Y$
shows $surjective\ f \longleftrightarrow (\forall y.\ y \in_c Y \longrightarrow (\exists x.\ x \in_c X \wedge f \circ_c x = y))$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.30 in Halvorson.

lemma *surjective-is-epimorphism*:
$surjective\ f \Longrightarrow epimorphism\ f$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.10 in Halvorson.

lemma *cfunc-cross-prod-surj*:
assumes *type-assms*: $f : A \to C$ $g : B \to D$
assumes *f-surj*: *surjective f* **and** *g-surj*: *surjective g*
shows $surjective\ (f \times_f g)$
⟨*proof*⟩

lemma *cfunc-cross-prod-surj-converse*:
assumes *type-assms*: $f : A \to C$ $g : B \to D$
assumes *nonempty*: $nonempty\ C \wedge nonempty\ D$
assumes $surjective\ (f \times_f g)$
shows $surjective\ f \wedge surjective\ g$
⟨*proof*⟩

3.5 Interactions of Cartesian Products with Terminal Objects

lemma *diag-on-elements*:
assumes $x \in_c X$
shows $diagonal\ X \circ_c x = \langle x,x \rangle$
⟨*proof*⟩

lemma *one-cross-one-unique-element*:
$\exists!\ x.\ x \in_c \mathbf{1} \times_c \mathbf{1}$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.1.20 in Halvorson.

lemma *X-is-cart-prod1*:
$is\text{-}cart\text{-}prod\ X\ (id\ X)\ (\beta_X)\ X\ \mathbf{1}$
⟨*proof*⟩

lemma *X-is-cart-prod2*:
$is\text{-}cart\text{-}prod\ X\ (\beta_X)\ (id\ X)\ \mathbf{1}\ X$
⟨*proof*⟩

lemma *A-x-one-iso-A*:

$X \times_c \mathbf{1} \cong X$
⟨*proof*⟩

lemma *one-x-A-iso-A*:
$\mathbf{1} \times_c X \cong X$
⟨*proof*⟩

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

lemma *left-cart-proj-one-left-inverse*:
$\langle id\ X, \beta_X \rangle \circ_c left\text{-}cart\text{-}proj\ X\ \mathbf{1} = id\ (X \times_c \mathbf{1})$
⟨*proof*⟩

lemma *left-cart-proj-one-right-inverse*:
$left\text{-}cart\text{-}proj\ X\ \mathbf{1} \circ_c \langle id\ X, \beta_X \rangle = id\ X$
⟨*proof*⟩

lemma *right-cart-proj-one-left-inverse*:
$\langle \beta_X, id\ X \rangle \circ_c right\text{-}cart\text{-}proj\ \mathbf{1}\ X = id\ (\mathbf{1} \times_c X)$
⟨*proof*⟩

lemma *right-cart-proj-one-right-inverse*:
$right\text{-}cart\text{-}proj\ \mathbf{1}\ X \circ_c \langle \beta_X, id\ X \rangle = id\ X$
⟨*proof*⟩

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

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 $a = b \longleftrightarrow$
$(left\text{-}cart\text{-}proj\ X\ Y \circ_c a = left\text{-}cart\text{-}proj\ X\ Y \circ_c b$
$\wedge\ right\text{-}cart\text{-}proj\ X\ Y \circ_c a = right\text{-}cart\text{-}proj\ X\ Y \circ_c b)$
⟨*proof*⟩

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 $\langle x,\ y \rangle = \langle x',\ y' \rangle \longleftrightarrow x = x' \wedge y = y'$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.1.23 in Halvorson.

lemma *nonempty-right-imp-left-proj-epimorphism*:
$nonempty\ Y \Longrightarrow epimorphism\ (left\text{-}cart\text{-}proj\ X\ Y)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $bd \circ_c k = cd \circ_c h \Longrightarrow (\exists!\ j.\ j : Z \to A \wedge ab \circ_c j = k \wedge ac \circ_c j = h)$
  ⟨*proof*⟩

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*⟩

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* $E\ m\ f\ g \longleftrightarrow ((f \circ_c m = g \circ_c m)$
$\wedge\ (\forall\ h\ F.\ ((h : F \to X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k.\ (k : F \to E) \wedge m \circ_c k = h)))$
⟨*proof*⟩

lemma *equalizer-eq*:
assumes $f : X \to Y\ g : X \to Y\ m : E \to X$
assumes *equalizer* $E\ m\ f\ g$
shows $f \circ_c m = g \circ_c m$
⟨*proof*⟩

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 $\exists!\ k.\ k : F \to E \wedge m \circ_c k = h$
⟨*proof*⟩

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*⟩

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*⟩

lemma *isomorphic-to-equalizer-is-equalizer*:
assumes $\varphi: E' \to E$
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*⟩

The lemma below corresponds to Exercise 2.1.34 in Halvorson.

lemma *equalizer-is-monomorphism*:
equalizer E m f $g \Longrightarrow$ *monomorphism*(m)
⟨*proof*⟩

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*⟩

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* $f \longleftrightarrow (\exists\ h.\ (h\colon$ *domain*$(g) \to$ *domain*$(f)) \wedge f \circ_c h = g)$

lemma *factors-through-def2*:
assumes $g : X \to Z$ $f : Y \to Z$
shows g *factorsthru* $f \longleftrightarrow (\exists\ h.\ h\colon X \to Y \wedge f \circ_c h = g)$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.33 in Halvorson.

lemma *xfactorthru-equalizer-iff-fx-eq-gx*:
assumes $f\colon X \to Y$ $g\colon X \to Y$ *equalizer* E m f g $x \in_c X$
shows x *factorsthru* $m \longleftrightarrow f \circ_c x = g \circ_c x$
⟨*proof*⟩

The definition below corresponds to Definition 2.1.37 in Halvorson.

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

lemma *subobject-of-def2*:
$(B,m) \subseteq_c X = (m : B \to X \wedge$ *monomorphism* $m)$
⟨*proof*⟩

definition *relative-subset* :: *cset* × *cfunc* ⇒ *cset* ⇒ *cset* × *cfunc* ⇒ *bool* $(\text{-}\subseteq_{\text{-}}\text{-}$ $[51,50,51]50)$

where $B \subseteq_X A \longleftrightarrow$
($snd\ B : fst\ B \to X \wedge monomorphism\ (snd\ B) \wedge snd\ A : fst\ A \to X \wedge monomorphism\ (snd\ A)$
$\wedge (\exists\ k.\ k: fst\ B \to fst\ A \wedge snd\ A \circ_c k = snd\ B))$

lemma *relative-subset-def2*:
$(B,m) \subseteq_X (A,n) = (m : B \to X \wedge monomorphism\ m \wedge n : A \to X \wedge monomorphism\ n$
$\wedge (\exists\ k.\ k: B \to A \wedge n \circ_c k = m))$
$\langle proof \rangle$

lemma *subobject-is-relative-subset*: $(B,m) \subseteq_c A \longleftrightarrow (B,m) \subseteq_A (A,\ id(A))$
$\langle proof \rangle$

The definition below corresponds to Definition 2.1.39 in Halvorson.

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

lemma *relative-member-def2*:
$x \in_X (B,\ m) = (x \in_c X \wedge monomorphism\ m \wedge m : B \to X \wedge x\ factorsthru\ m)$
$\langle proof \rangle$

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 $x \in_X (A,n) \Longrightarrow x \in_X (B,m)$
$\langle proof \rangle$

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

lemma *inverse-image-is-equalizer*:
assumes $m : B \to Y\ f : X \to Y\ monomorphism\ m$
shows $\exists\, k.\ equalizer\ (f^{-1}(\!|B|\!)_m)\ k\ (f \circ_c left\text{-}cart\text{-}proj\ X\ B)\ (m \circ_c right\text{-}cart\text{-}proj\ X\ B)$
$\langle proof \rangle$

definition *inverse-image-mapping* :: $cfunc \Rightarrow cset \Rightarrow cfunc \Rightarrow cfunc$ **where**
$inverse\text{-}image\text{-}mapping\ f\ B\ m = (SOME\ k.\ \exists\ X\ Y.\ f : X \to Y \wedge m : B \to Y \wedge monomorphism\ m \wedge$

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 $m : B \to Y$ $f : X \to Y$ *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*⟩

lemma *inverse-image-mapping-type*[*type-rule*]:
assumes $m : B \to Y$ $f : X \to Y$ *monomorphism m*
shows *inverse-image-mapping f B m* : (*inverse-image f B m*) $\to X \times_c B$
⟨*proof*⟩

lemma *inverse-image-mapping-eq*:
assumes $m : B \to Y$ $f : X \to Y$ *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*
⟨*proof*⟩

lemma *inverse-image-mapping-monomorphism*:
assumes $m : B \to Y$ $f : X \to Y$ *monomorphism m*
shows *monomorphism* (*inverse-image-mapping f B m*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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* : $f^{-1}(\!|B|\!)_m \to X$
⟨*proof*⟩

lemma *inverse-image-subobject-mapping-mono*:
assumes $f : X \to Y$ $m : B \to Y$ *monomorphism m*
shows *monomorphism* ($[f^{-1}(\!|B|\!)_m]$*map*)
⟨*proof*⟩

lemma *inverse-image-subobject*:
assumes $m : B \to Y\ f : X \to Y$ *monomorphism* m
shows $(f^{-1}(\!|B|\!)_m, [f^{-1}(\!|B|\!)_m]map) \subseteq_c X$
⟨*proof*⟩

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* $f\ B\ m)\ f$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.1.41 in Halvorson.

lemma *in-inverse-image*:
assumes $f : X \to Y\ (B,m) \subseteq_c Y\ x \in_c X$
shows $(x \in_X (f^{-1}(\!|B|\!)_m,$ *left-cart-proj* $X\ B \circ_c$ *inverse-image-mapping* $f\ B\ m)) = (f \circ_c x \in_Y (B,m))$
⟨*proof*⟩

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**
$X\ {}_f\times_{cg}\ Y = (SOME\ E.\ \exists\ Z\ m.\ f : X \to Z \land g : Y \to Z \land$
equalizer $E\ m\ (f \circ_c$ *left-cart-proj* $X\ Y)\ (g \circ_c$ *right-cart-proj* $X\ Y))$

lemma *fibered-product-equalizer*:
assumes $f : X \to Z\ g : Y \to Z$
shows $\exists\ m.$ *equalizer* $(X\ {}_f\times_{cg}\ Y)\ m\ (f \circ_c$ *left-cart-proj* $X\ Y)\ (g \circ_c$ *right-cart-proj* $X\ Y)$
⟨*proof*⟩

definition *fibered-product-morphism* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cfunc* **where**
fibered-product-morphism $X\ f\ g\ Y = (SOME\ m.\ \exists\ Z.\ f : X \to Z \land g : Y \to Z \land$
equalizer $(X\ {}_f\times_{cg}\ Y)\ m\ (f \circ_c$ *left-cart-proj* $X\ Y)\ (g \circ_c$ *right-cart-proj* $X\ Y))$

lemma *fibered-product-morphism-equalizer*:
assumes $f : X \to Z\ g : Y \to Z$
shows *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* $X\ Y)$
⟨*proof*⟩

lemma *fibered-product-morphism-type*[*type-rule*]:
assumes $f : X \to Z\ g : Y \to Z$
shows *fibered-product-morphism* $X\ f\ g\ Y : X\ {}_f\times_{cg}\ Y \to X \times_c Y$
⟨*proof*⟩

lemma *fibered-product-morphism-monomorphism*:
  assumes $f : X \to Z$ $g : Y \to Z$
  shows *monomorphism* (*fibered-product-morphism* X f g Y)
  ⟨*proof*⟩

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 : $X\ {}_f\times_{cg}\ Y \to X$
  ⟨*proof*⟩

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 : $X\ {}_f\times_{cg}\ Y \to Y$
  ⟨*proof*⟩

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
  ⟨*proof*⟩

lemma *fibered-product-is-pullback*:
  assumes *f-type*[*type-rule*]: $f : X \to Z$ **and** *g-type*[*type-rule*]: $g : Y \to Z$
  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
  ⟨*proof*⟩

lemma *fibered-product-proj-eq*:
  assumes $f : X \to Z$ $g : Y \to Z$
  shows $f \circ_c$ *fibered-product-left-proj* X f g Y = $g \circ_c$ *fibered-product-right-proj* X f g Y
  ⟨*proof*⟩

lemma *fibered-product-pair-member*:
  assumes $f : X \to Z$ $g : Y \to Z$ $x \in_c X$ $y \in_c Y$
  shows $(\langle x, y\rangle \in_{X \times_c Y} (X {}_f\times_{cg} Y,$ *fibered-product-morphism* X f g Y)) = $(f \circ_c x = g \circ_c y)$
⟨*proof*⟩

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 $\forall\, x\ y.\ x \in_c X \longrightarrow y \in_c X \longrightarrow \langle x,y\rangle \in_{X\ \times_c\ X} (X\ {}_f\!\times_{cf}\ X,$ *fibered-product-morphism* $X\ f\ f\ X) \longrightarrow g \circ_c x = g \circ_c y$
⟨*proof*⟩

lemma *kernel-pair-subset*:
assumes $f\colon X \to Y$
shows $(X\ {}_f\!\times_{cf}\ X,$ *fibered-product-morphism* $X\ f\ f\ X) \subseteq_c X \times_c X$
⟨*proof*⟩

The three lemmas below correspond to Exercise 2.1.44 in Halvorson.

lemma *kern-pair-proj-iso-TFAE1*:
assumes $f\colon X \to Y$ *monomorphism* f
shows (*fibered-product-left-proj* $X\ f\ f\ X$) = (*fibered-product-right-proj* $X\ f\ f\ X$)
⟨*proof*⟩

lemma *kern-pair-proj-iso-TFAE2*:
assumes $f\colon X \to Y$ *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* $X\ f\ f\ X$)
⟨*proof*⟩

lemma *kern-pair-proj-iso-TFAE3*:
assumes $f\colon X \to Y$
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*⟩

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*⟩

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*]: t $\in_c$ Ω **and**
  false-func-type[*type-rule*]: f $\in_c$ Ω **and**
  true-false-distinct: t $\neq$ 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 **1** X Ω (β_B) t m χ

definition *characteristic-func* :: *cfunc* ⇒ *cfunc* **where**
  characteristic-func m =
    (*THE* χ. *monomorphism* m ⟶ *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*⟩

lemma *characteristic-func-type*[*type-rule*]:
  assumes $m : B \to X$ *monomorphism* m
  shows *characteristic-func* $m : X \to \Omega$
⟨*proof*⟩

lemma *characteristic-func-eq*:
  assumes $m : B \to X$ *monomorphism* m
  shows *characteristic-func* $m \circ_c m = \mathrm{t} \circ_c \beta_B$
  ⟨*proof*⟩

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 $\circ_c \beta_X$)
  ⟨*proof*⟩

lemma *characteristic-func-true-relative-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 $x \in_X (B,m)$
  ⟨*proof*⟩

lemma *characteristic-func-false-not-relative-member*:
  assumes $m : B \to X$ *monomorphism* m $x \in_c X$
  assumes *characteristic-func-true*: *characteristic-func* $m \circ_c x = \mathrm{f}$
  shows $\neg$ ($x \in_X (B,m)$)
  ⟨*proof*⟩

lemma *rel-mem-char-func-true*:
  assumes $m : B \rightarrow X$ *monomorphism* m $x \in_c X$
  assumes $x \in_X (B,m)$
  shows *characteristic-func* $m \circ_c x = \mathrm{t}$
  ⟨*proof*⟩

lemma *not-rel-mem-char-func-false*:
  assumes $m : B \rightarrow X$ *monomorphism* m $x \in_c X$
  assumes $\neg (x \in_X (B,m))$
  shows *characteristic-func* $m \circ_c x = \mathrm{f}$
  ⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.2 in Halvorson.

lemma *card* $\{x.\ x \in_c \Omega \times_c \Omega\} = 4$
⟨*proof*⟩

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 $(diagonal\ X)$ $(eq\text{-}pred\ X)$
  ⟨*proof*⟩

lemma *eq-pred-type*[*type-rule*]:
  eq-pred $X : X \times_c X \rightarrow \Omega$
  ⟨*proof*⟩

lemma *eq-pred-square*: *eq-pred* $X \circ_c$ *diagonal* $X = \mathrm{t} \circ_c \beta_X$
  ⟨*proof*⟩

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

lemma *eq-pred-iff-eq-conv*:
  assumes $x : \mathbf{1} \rightarrow X$ $y : \mathbf{1} \rightarrow X$
  shows $(x \neq y) = (eq\text{-}pred\ X \circ_c \langle x,\ y\rangle = \mathrm{f})$
⟨*proof*⟩

lemma *eq-pred-iff-eq-conv2*:
  assumes $x : \mathbf{1} \rightarrow X$ $y : \mathbf{1} \rightarrow X$
  shows $(x \neq y) = (eq\text{-}pred\ X \circ_c \langle x,\ y\rangle \neq \mathrm{t})$
  ⟨*proof*⟩

lemma *eq-pred-of-monomorphism*:
  assumes *m-type*[*type-rule*]: $m : X \rightarrow Y$ **and** *m-mono*: *monomorphism* m

shows *eq-pred* $Y \circ_c (m \times_f m) = $ *eq-pred* X
$\langle proof \rangle$

lemma *eq-pred-true-extract-right*:
assumes $x \in_c X$
shows *eq-pred* $X \circ_c \langle x \circ_c \beta_X, id\ X \rangle \circ_c x = \mathrm{t}$
$\langle proof \rangle$

lemma *eq-pred-false-extract-right*:
assumes $x \in_c X$ $y \in_c X$ $x \neq y$
shows *eq-pred* $X \circ_c \langle x \circ_c \beta_X, id\ X \rangle \circ_c y = \mathrm{f}$
$\langle proof \rangle$

5.2 Properties of Monomorphisms and Epimorphisms

The lemma below corresponds to Exercise 2.2.3 in Halvorson.

lemma *regmono-is-mono*: *regular-monomorphism* $m \Longrightarrow$ *monomorphism* m
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.4 in Halvorson.

lemma *mono-is-regmono*:
shows *monomorphism* $m \Longrightarrow$ *regular-monomorphism* m
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.5 in Halvorson.

lemma *epi-mon-is-iso*:
assumes *epimorphism f monomorphism f*
shows *isomorphism f*
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.8 in Halvorson.

lemma *epi-is-surj*:
assumes p: $X \rightarrow Y$ *epimorphism* p
shows *surjective* p
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.9 in Halvorson.

lemma *pullback-of-epi-is-epi1*:
assumes f: $Y \rightarrow Z$ *epimorphism f is-pullback* A Y X Z *q1* f *q0* g
shows *epimorphism q0*
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.9b in Halvorson.

lemma *pullback-of-epi-is-epi2*:
assumes g: $X \rightarrow Z$ *epimorphism* g *is-pullback* A Y X Z *q1* f *q0* g
shows *epimorphism q1*
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.9c in Halvorson.

lemma *pullback-of-mono-is-mono1*:
assumes *g*: $X \to Z$ *monomorphism f is-pullback A Y X Z q1 f q0 g*
shows *monomorphism q0*
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.9d in Halvorson.

lemma *pullback-of-mono-is-mono2*:
assumes *g*: $X \to Z$ *monomorphism g is-pullback A Y X Z q1 f q0 g*
shows *monomorphism q1*
⟨*proof*⟩

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* ($-^{-1}\{-\}$ [*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* : $f^{-1}\{y\} \to X$
⟨*proof*⟩

lemma *fiber-subset*:
assumes $f : X \to Y$ $y \in_c Y$
shows ($f^{-1}\{y\}$, *fiber-morphism f y*) $\subseteq_c X$
⟨*proof*⟩

lemma *fiber-morphism-monomorphism*:
assumes $f : X \to Y$ $y \in_c Y$
shows *monomorphism* (*fiber-morphism f y*)
⟨*proof*⟩

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*⟩

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*: ¬ *surjective p*
shows ∃ *y*. $y \in_c Y \wedge$ *is-empty*($p^{-1}\{y\}$)
⟨*proof*⟩

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 $f^{-1}\{y\} \cong X\ {}_{f}{\times_{c}}_{y}\mathbf{1}$
⟨*proof*⟩

lemma *fib-prod-left-id-iso*:
assumes $g : Y \to X$
shows $(X\ {}_{id(X)}{\times_{c}}_{g}\ Y) \cong Y$
⟨*proof*⟩

lemma *fib-prod-right-id-iso*:
assumes $f : X \to Y$
shows $(X\ {}_{f}{\times_{c}}_{id(Y)}\ Y) \cong X$
⟨*proof*⟩

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 $\exists!\ b.\ b : X\ {}_{f}{\times_{c}}_{f}\ X \to E\ {}_{h}{\times_{c}}_{h}\ E\ \wedge$
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*⟩

6 Set Subtraction

definition *set-subtraction* :: *cset* ⇒ *cset* × *cfunc* ⇒ *cset* (**infix** \ *60*) **where**
$Y \setminus X = (SOME\ E.\ \exists\ m'.\ \ equalizer\ E\ m'\ (characteristic\text{-}func\ (snd\ X))\ (\mathrm{f} \circ_c \beta_Y))$

lemma *set-subtraction-equalizer*:
assumes $m : X \to Y$ *monomorphism m*
shows $\exists\ m'.\ \ equalizer\ (Y \setminus (X,m))\ m'\ (characteristic\text{-}func\ m)\ (\mathrm{f} \circ_c \beta_Y)$
⟨*proof*⟩

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

lemma *complement-morphism-equalizer*:
assumes $m : X \to Y$ *monomorphism m*

shows *equalizer* $(Y \setminus (X,m))$ m^c (*characteristic-func* m) $(\mathrm{f} \circ_c \beta_Y)$
⟨*proof*⟩

lemma *complement-morphism-type*[*type-rule*]:
assumes $m : X \to Y$ *monomorphism* m
shows $m^c : Y \setminus (X,m) \to Y$
⟨*proof*⟩

lemma *complement-morphism-mono*:
assumes $m : X \to Y$ *monomorphism* m
shows *monomorphism* m^c
⟨*proof*⟩

lemma *complement-morphism-eq*:
assumes $m : X \to Y$ *monomorphism* m
shows *characteristic-func* $m \circ_c m^c = (\mathrm{f} \circ_c \beta_Y) \circ_c m^c$
⟨*proof*⟩

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*⟩

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*⟩

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 $\neg\ x \in_Y (X,\ m)$
⟨*proof*⟩

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 $x \in_Y (Y \setminus (X,m),\ m^c)$
⟨*proof*⟩

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*⟩

lemma *set-subtraction-right-iso*:

assumes *m-type*[*type-rule*]: $m : A \rightarrow C$ **and** *m-mono*[*type-rule*]: *monomorphism* m
assumes *i-type*[*type-rule*]: $i : B \rightarrow A$ **and** *i-iso*: *isomorphism* i
shows $C \setminus (A,m) = C \setminus (B, m \circ_c i)$
⟨*proof*⟩

lemma *set-subtraction-left-iso*:
assumes *m-type*[*type-rule*]: $m : C \rightarrow A$ **and** *m-mono*[*type-rule*]: *monomorphism* m
assumes *i-type*[*type-rule*]: $i : A \rightarrow B$ **and** *i-iso*: *isomorphism* i
shows $A \setminus (C,m) \cong B \setminus (C, i \circ_c m)$
⟨*proof*⟩

7 Graphs

definition *functional-on* :: *cset* ⇒ *cset* ⇒ *cset* × *cfunc* ⇒ *bool* **where**
functional-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))
⟨*proof*⟩

lemma *graph-equalizer2*:
assumes $f : X \rightarrow Y$
shows ∃ m. *equalizer* (*graph* f) m ($f \circ_c$ *left-cart-proj* X Y) (*right-cart-proj* X Y)
⟨*proof*⟩

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))
⟨*proof*⟩

lemma *graph-equalizer4*:
assumes $f : X \rightarrow Y$
shows *equalizer* (*graph* f) (*graph-morph* f) ($f \circ_c$ *left-cart-proj* X Y) (*right-cart-proj* X Y)
⟨*proof*⟩

lemma *graph-subobject*:
assumes $f : X \to Y$
shows $(graph\ f,\ graph\text{-}morph\ f) \subseteq_c (X \times_c Y)$
⟨*proof*⟩

lemma *graph-morph-type*[*type-rule*]:
assumes $f : X \to Y$
shows $graph\text{-}morph(f) : graph\ f \to X \times_c Y$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.3.13 in Halvorson.

lemma *graphs-are-functional*:
assumes $f : X \to Y$
shows $functional\text{-}on\ X\ Y\ (graph\ f,\ graph\text{-}morph\ f)$
⟨*proof*⟩

lemma *functional-on-isomorphism*:
assumes $functional\text{-}on\ X\ Y\ (R,m)$
shows $isomorphism(left\text{-}cart\text{-}proj\ X\ Y \circ_c m)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.3.14 in Halvorson.

lemma *functional-relations-are-graphs*:
assumes $functional\text{-}on\ X\ Y\ (R,m)$
shows $\exists!\ f.\ f : X \to Y \wedge$
$(\exists\ i.\ i : R \to graph(f) \wedge isomorphism(i) \wedge m = graph\text{-}morph(f) \circ_c i)$
⟨*proof*⟩

end

8 Equivalence Classes and Coequalizers

theory *Equivalence*
imports *Truth*
begin

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

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

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

$(\langle x,y\rangle \in_{X \times_c X} R \wedge \langle y,z\rangle \in_{X \times_c X} R \longrightarrow \langle x,z\rangle \in_{X \times_c X} R)))$

definition *equiv-rel-on* :: *cset* ⇒ *cset* × *cfunc* ⇒ *bool* **where**
equiv-rel-on X R ⟷ (*reflexive-on* X R ∧ *symmetric-on* X R ∧ *transitive-on* X R)

definition *const-on-rel* :: *cset* ⇒ *cset* × *cfunc* ⇒ *cfunc* ⇒ *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 $\exists\, y.\ y \in_c Y \wedge\ m \circ_c y = \langle x,x\rangle$
⟨*proof*⟩

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 $\exists\, w.\ w \in_c Y \wedge\ m \circ_c w = \langle y,x\rangle$
⟨*proof*⟩

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 $\exists\, u.\ u \in_c Y \wedge\ m \circ_c u = \langle x,z\rangle$
⟨*proof*⟩

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 $(X\ {}_f{\times_c}_f\ X$, *fibered-product-morphism* $X\ f\ f\ X)$
⟨*proof*⟩

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

lemma *coequalizer-def2*:
assumes $f : Y \to X$ $g : Y \to X$ $m : X \to E$
shows *coequalizer* E m f $g \longleftrightarrow$
$(m \circ_c f = m \circ_c g)$
$\wedge\ (\forall\ h\ F.\ ((h : X \to F) \wedge (h \circ_c f = h \circ_c g)) \longrightarrow (\exists!\ k.\ (k : E \to F) \wedge k \circ_c$
$m = h))$
⟨*proof*⟩

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 $E \cong F$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.3.2 in Halvorson.

lemma *coequalizer-is-epimorphism*:
coequalizer E m f $g \Longrightarrow$ *epimorphism*(m)
⟨*proof*⟩

lemma *canonical-quotient-map-is-coequalizer*:

assumes *equiv-rel-on* X (R,m)
shows *coequalizer* $(X \mathbin{/\!\!/} (R,m))$ $(\textit{equiv-class}\ (R,m))$
$(\textit{left-cart-proj}\ X\ X \circ_c m)$ $(\textit{right-cart-proj}\ X\ X \circ_c m)$
⟨*proof*⟩

lemma *canonical-quot-map-is-epi*:
assumes *equiv-rel-on* X (R,m)
shows *epimorphism*$((\textit{equiv-class}\ (R,m)))$
⟨*proof*⟩

8.2 Regular Epimorphisms

The definition below corresponds to Definition 2.3.4 in Halvorson.

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

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*⟩

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 $(\textit{fibered-product-left-proj}\ X\ f\ f\ X)$ $(\textit{fibered-product-right-proj}\ X\ f\ f\ X)$
⟨*proof*⟩

lemma *epimorphisms-are-regular*:
assumes $f : X \to Y$ *epimorphism* f
shows *regular-epimorphism* f
⟨*proof*⟩

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 $(\textit{fibered-product-left-proj}\ X\ f\ f\ X)$ $(\textit{fibered-product-right-proj}\ X\ f\ f\ X)$
$\wedge$ *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*⟩

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* $\rightarrow$ *Y* $\longrightarrow$ *f* = *x* $\circ_c$ *g* $\longrightarrow$ *x* = *m*)
⟨*proof*⟩

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*⟩

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*) ∧
monomorphism m ∧ *f* $\circ_c$ *n* = *m* $\circ_c$ ($f{\restriction}_{(A,\ n)}$) ∧ (∀ *x*. *x* : $f(\!|A|\!)_n$ → *Y* ⟶ *f* $\circ_c$ *n* = *x* $\circ_c$ ($f{\restriction}_{(A,\ n)}$) ⟶ *x* = *m*)
⟨*proof*⟩

definition *image-subobject-mapping* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cfunc* ([-(|-|)-]*map* [*101*,*0*,*0*]*100*) **where**
[$f(\!|A|\!)_n$]*map* = (*THE m*. $f{\restriction}_{(A,\ n)}$: *A* → $f(\!|A|\!)_n$ ∧ *m* : $f(\!|A|\!)_n$ → *codomain f* ∧

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 $m \wedge f \circ_c n = m \circ_c (f\restriction_{(A,\ n)}) \wedge (\forall x.\ x : (f(\!|A|\!)_n) \to$ *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 $f\restriction_{(A,\ n)} : A \to f(\!|A|\!)_n \wedge [f(\!|A|\!)_n]map : f(\!|A|\!)_n \to Y \wedge$
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 $([f(\!|A|\!)_n]map) \wedge f \circ_c n = [f(\!|A|\!)_n]map \circ_c (f\restriction_{(A,\ n)}) \wedge (\forall x.\ x : f(\!|A|\!)_n \to Y \longrightarrow f \circ_c n = x \circ_c (f\restriction_{(A,\ n)}) \longrightarrow x = [f(\!|A|\!)_n]map)$
⟨*proof*⟩

lemma *image-rest-map-type*[*type-rule*]:
assumes $f : X \to Y$ $n : A \to X$
shows $f\restriction_{(A,\ n)} : A \to f(\!|A|\!)_n$
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *image-rest-map-epi*:
assumes $f : X \to Y$ $n : A \to X$
shows *epimorphism* $(f\restriction_{(A,\ n)})$
⟨*proof*⟩

lemma *image-subobj-map-type*[*type-rule*]:
assumes $f : X \to Y$ $n : A \to X$
shows $[f(\!|A|\!)_n]map : f(\!|A|\!)_n \to Y$
⟨*proof*⟩

lemma *image-subobj-map-mono*:
assumes $f : X \to Y$ $n : A \to X$
shows *monomorphism* $([f(\!|A|\!)_n]map)$
⟨*proof*⟩

lemma *image-subobj-comp-image-rest*:
assumes $f : X \to Y$ $n : A \to X$
shows $[f(\!|A|\!)_n]map \circ_c (f\restriction_{(A,\ n)}) = f \circ_c n$
⟨*proof*⟩

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

$\langle proof \rangle$

lemma *image-self*:
assumes $f : X \to Y$ **and** *monomorphism* f
assumes $a : A \to X$ **and** *monomorphism* a
shows $f(\!|A|\!)_a \cong A$
$\langle proof \rangle$

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* $n \Longrightarrow (f(\!|A|\!)_a, [f(\!|A|\!)_a]map) \subseteq_Y (B, n)$
$\langle proof \rangle$

lemma *images-iso*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
assumes *m-type*[*type-rule*]: $m : Z \to X$ **and** *n-type*[*type-rule*]: $n : A \to Z$
shows $(f \circ_c m)(\!|A|\!)_n \cong f(\!|A|\!)_{m \circ_c n}$
$\langle proof \rangle$

lemma *image-subset-conv*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
assumes *m-type*[*type-rule*]: $m : Z \to X$ **and** *n-type*[*type-rule*]: $n : A \to Z$
shows $\exists\, i.\ ((f \circ_c m)(\!|A|\!)_n, i) \subseteq_c B \Longrightarrow \exists j.\ (f(\!|A|\!)_{m \circ_c n}, j) \subseteq_c B$
$\langle proof \rangle$

lemma *image-rel-subset-conv*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
assumes *m-type*[*type-rule*]: $m : Z \to X$ **and** *n-type*[*type-rule*]: $n : A \to Z$
assumes *rel-sub1*: $((f \circ_c m)(\!|A|\!)_n, [(f \circ_c m)(\!|A|\!)_n]map) \subseteq_Y (B,b)$
shows $(f(\!|A|\!)_{m \circ_c n}, [f(\!|A|\!)_{m \circ_c n}]map) \subseteq_Y (B,b)$
$\langle proof \rangle$

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 $((A, a) \subseteq_X (f^{-1}(\!|B|\!)_m,[f^{-1}(\!|B|\!)_m]map)) = ((f(\!|A|\!)_a, [f(\!|A|\!)_a]map) \subseteq_Y (B,m))$
$\langle proof \rangle$

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 $(A,m) \subseteq_X (f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}, [f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}]map)$
$\langle proof \rangle$

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

lemma *left-pair-subset*:
assumes $m : Y \to X \times_c X$ *monomorphism m*
shows $(Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c (m \times_f id_c\ Z)) \subseteq_c (X \times_c Z) \times_c (X \times_c Z)$
⟨*proof*⟩

lemma *right-pair-subset*:
assumes $m : Y \to X \times_c X$ *monomorphism m*
shows $(Z \times_c Y,$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m)) \subseteq_c (Z \times_c X) \times_c (Z \times_c X)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *right-pair-transitive*:
assumes *transitive-on* $X\ (Y, m)$
shows *transitive-on* $(Z \times_c X)\ (Z \times_c Y,$ *distribute-left* $Z\ X\ X \circ_c (id_c\ Z \times_f m))$
⟨*proof*⟩

lemma *left-pair-equiv-rel*:
assumes *equiv-rel-on* $X\ (Y, m)$
shows *equiv-rel-on* $(X \times_c Z)\ (Y \times_c Z,$ *distribute-right* $X\ X\ Z \circ_c (m \times_f id\ Z))$

$\langle proof \rangle$

lemma *right-pair-equiv-rel*:
assumes *equiv-rel-on* X (Y, m)
shows *equiv-rel-on* $(Z \times_c X)$ $(Z \times_c Y,$ *distribute-left* $Z\ X\ X\ \circ_c$ $($*id* $Z \times_f m))$
$\langle proof \rangle$

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* $\Rightarrow$ *cset* $\Rightarrow$ *cset* (**infixr** $\coprod$ *65*) **and**
left-coproj :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **and**
right-coproj :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **and**
cfunc-coprod :: *cfunc* $\Rightarrow$ *cfunc* $\Rightarrow$ *cfunc* (**infixr** $\amalg$ *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*]: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f \amalg g : X \coprod Y \to Z$ **and**
left-coproj-cfunc-coprod: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f \amalg g \circ_c$ $($*left-coproj* $X\ Y) = f$ **and**
right-coproj-cfunc-coprod: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f \amalg g \circ_c$ $($*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* $X\ Y = g \Longrightarrow h = f \amalg g$

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

$(\forall\ h2.\ (h2 : W \to Z \land h2 \circ_c i_0 = f \land h2 \circ_c i_1 = g) \longrightarrow h2 = h)))$
⟨*proof*⟩

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*
⟨*proof*⟩

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 $\exists\ g.\ g : W \to W' \land$ *isomorphism* $g \land g \circ_c i_0 = i'_0 \land g \circ_c i_1 = i'_1$
⟨*proof*⟩

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*⟩

lemma *id-coprod*:
$id(A \coprod B) =$ (*left-coproj A B*) $\amalg$ (*right-coproj A B*)
⟨*proof*⟩

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*⟩

The lemma below corresponds to Proposition 2.4.2 in Halvorson.

lemma *left-coproj-are-monomorphisms*:
monomorphism(*left-coproj X Y*)
⟨*proof*⟩

lemma *right-coproj-are-monomorphisms*:
monomorphism(*right-coproj X Y*)
⟨*proof*⟩

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*⟩

lemma *maps-into-1u1*:
  assumes *x-type*: $x \in_c (\mathbf{1} \coprod \mathbf{1})$
  shows $(x = \textit{left-coproj}\ \mathbf{1}\ \mathbf{1}) \lor (x = \textit{right-coproj}\ \mathbf{1}\ \mathbf{1})$
  ⟨*proof*⟩

lemma *coprod-preserves-left-epi*:
  assumes f: $X \to Z$ g: $Y \to Z$
  assumes *surjective*(f)
  shows *surjective*(f ⨿ g)
  ⟨*proof*⟩

lemma *coprod-preserves-right-epi*:
  assumes f: $X \to Z$ g: $Y \to Z$
  assumes *surjective*(g)
  shows *surjective*(f ⨿ g)
  ⟨*proof*⟩

lemma *coprod-eq*:
  assumes $a : X \coprod Y \to Z$ $b : X \coprod Y \to Z$
  shows $a = b \longleftrightarrow$
    $(a \circ_c \textit{left-coproj}\ X\ Y = b \circ_c \textit{left-coproj}\ X\ Y$
      $\land\ a \circ_c \textit{right-coproj}\ X\ Y = b \circ_c \textit{right-coproj}\ X\ Y)$
  ⟨*proof*⟩

lemma *coprod-eqI*:
  assumes $a : X \coprod Y \to Z$ $b : X \coprod Y \to Z$
  assumes $(a \circ_c \textit{left-coproj}\ X\ Y = b \circ_c \textit{left-coproj}\ X\ Y$
      $\land\ a \circ_c \textit{right-coproj}\ X\ Y = b \circ_c \textit{right-coproj}\ X\ Y)$
  shows $a = b$
  ⟨*proof*⟩

lemma *coprod-eq2*:
  assumes $a : X \to Z$ $b : Y \to Z$ $c : X \to Z$ $d : Y \to Z$
  shows $(a \amalg b) = (c \amalg d) \longleftrightarrow (a = c \land b = d)$
  ⟨*proof*⟩

lemma *coprod-decomp*:
  assumes $a : X \coprod Y \to A$
  shows $\exists\ x\ y.\ a = (x \amalg y) \land x : X \to A \land y : Y \to A$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.4.4 in Halvorson.

lemma *truth-value-set-iso-1u1*:
  isomorphism(t⨿f)
  ⟨*proof*⟩

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* $X\ Y \circ_c x\rangle = (($*eq-pred* $X \circ_c \langle id\ X,\ x \circ_c \beta_X\rangle) \amalg (\mathrm{f} \circ_c \beta_Y)) \circ_c u$
⟨*proof*⟩

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*⟩

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)) $\circ_c f) \amalg (($*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* $Y\ W \circ_c f) \amalg ($*right-coproj* $Y\ W \circ_c g)$
⟨*proof*⟩

lemma *cfunc-bowtie-prod-type*[*type-rule*]:
$f : X \to Y \Longrightarrow g : V \to W \Longrightarrow f \bowtie_f g : X \coprod V \to Y \coprod W$
⟨*proof*⟩

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* $Y\ W \circ_c f$
⟨*proof*⟩

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* $Y\ W \circ_c g$
⟨*proof*⟩

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* $Y\ W \circ_c g \Longrightarrow h = f \bowtie_f g$
⟨*proof*⟩

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*⟩

lemma *coproduct-of-beta*:
$\beta_X \amalg \beta_Y = \beta_{X \coprod Y}$

$\langle proof \rangle$

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 $(a \amalg b) \circ_c (f \bowtie_f g) = (a \circ_c f) \amalg (b \circ_c g)$
$\langle proof \rangle$

lemma *id-bowtie-prod*: $id(X) \bowtie_f id(Y) = id(X \coprod Y)$
  $\langle proof \rangle$

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 $(x \bowtie_f y) \circ_c (f \bowtie_f g) = (x \circ_c f) \bowtie_f (y \circ_c g)$
$\langle proof \rangle$

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* $(f \bowtie_f g)$
$\langle proof \rangle$

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* $(f \bowtie_f g)$
  $\langle proof \rangle$

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* $f \wedge$ *injective* g
  $\langle proof \rangle$

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* $(f \bowtie_f g)$
  $\langle proof \rangle$

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
  $\langle proof \rangle$

9.3 Boolean Cases

definition *case-bool* :: *cfunc* **where**

$case\text{-}bool = (THE\, f.\ f : \Omega \to (\mathbf{1} \coprod \mathbf{1}) \wedge$
$(\mathrm{t} \amalg \mathrm{f}) \circ_c f = id\ \Omega \wedge f \circ_c (\mathrm{t} \amalg \mathrm{f}) = id\ (\mathbf{1} \coprod \mathbf{1}))$

lemma *case-bool-def2*:
$case\text{-}bool : \Omega \to (\mathbf{1} \coprod \mathbf{1}) \wedge$
$(\mathrm{t} \amalg \mathrm{f}) \circ_c case\text{-}bool = id\ \Omega \wedge case\text{-}bool \circ_c (\mathrm{t} \amalg \mathrm{f}) = id\ (\mathbf{1} \coprod \mathbf{1})$
⟨*proof*⟩

lemma *case-bool-type*[*type-rule*]:
$case\text{-}bool : \Omega \to \mathbf{1} \coprod \mathbf{1}$
⟨*proof*⟩

lemma *case-bool-true-coprod-false*:
$case\text{-}bool \circ_c (\mathrm{t} \amalg \mathrm{f}) = id\ (\mathbf{1} \coprod \mathbf{1})$
⟨*proof*⟩

lemma *true-coprod-false-case-bool*:
$(\mathrm{t} \amalg \mathrm{f}) \circ_c case\text{-}bool = id\ \Omega$
⟨*proof*⟩

lemma *case-bool-iso*:
isomorphism case-bool
⟨*proof*⟩

lemma *case-bool-true-and-false*:
$(case\text{-}bool \circ_c \mathrm{t} = left\text{-}coproj\ \mathbf{1}\ \mathbf{1}) \wedge (case\text{-}bool \circ_c \mathrm{f} = right\text{-}coproj\ \mathbf{1}\ \mathbf{1})$
⟨*proof*⟩

lemma *case-bool-true*:
$case\text{-}bool \circ_c \mathrm{t} = left\text{-}coproj\ \mathbf{1}\ \mathbf{1}$
⟨*proof*⟩

lemma *case-bool-false*:
$case\text{-}bool \circ_c \mathrm{f} = right\text{-}coproj\ \mathbf{1}\ \mathbf{1}$
⟨*proof*⟩

lemma *coprod-case-bool-true*:
assumes $x1 \in_c X$
assumes $x2 \in_c X$
shows $(x1 \amalg x2 \circ_c case\text{-}bool) \circ_c \mathrm{t} = x1$
⟨*proof*⟩

lemma *coprod-case-bool-false*:
assumes $x1 \in_c X$
assumes $x2 \in_c X$
shows $(x1 \amalg x2 \circ_c case\text{-}bool) \circ_c \mathrm{f} = x2$
⟨*proof*⟩

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 $A\ B\ C = (id\ A \times_f\ left\text{-}coproj\ B\ C)\ \amalg\ (id\ A \times_f\ right\text{-}coproj\ B\ C)$

lemma *factor-prod-coprod-left-type*[*type-rule*]:
factor-prod-coprod-left $A\ B\ C : (A \times_c B) \coprod (A \times_c C) \to A \times_c (B \coprod C)$
⟨*proof*⟩

lemma *factor-prod-coprod-left-ap-left*:
assumes $a \in_c A$ $b \in_c B$
shows *factor-prod-coprod-left* $A\ B\ C \circ_c left\text{-}coproj\ (A \times_c B)\ (A \times_c C) \circ_c \langle a, b\rangle = \langle a, left\text{-}coproj\ B\ C \circ_c b\rangle$
⟨*proof*⟩

lemma *factor-prod-coprod-left-ap-right*:
assumes $a \in_c A$ $c \in_c C$
shows *factor-prod-coprod-left* $A\ B\ C \circ_c right\text{-}coproj\ (A \times_c B)\ (A \times_c C) \circ_c \langle a, c\rangle = \langle a, right\text{-}coproj\ B\ C \circ_c c\rangle$
⟨*proof*⟩

lemma *factor-prod-coprod-left-mono*:
monomorphism (*factor-prod-coprod-left* $A\ B\ C$)
⟨*proof*⟩

lemma *factor-prod-coprod-left-epi*:
epimorphism (*factor-prod-coprod-left* $A\ B\ C$)
⟨*proof*⟩

lemma *dist-prod-coprod-iso*:
isomorphism(*factor-prod-coprod-left* $A\ B\ C$)
⟨*proof*⟩

The lemma below corresponds to Proposition 2.5.10 in Halvorson.

lemma *prod-distribute-coprod*:
$A \times_c (X \coprod Y) \cong (A \times_c X) \coprod (A \times_c Y)$
⟨*proof*⟩

9.4.2 Distribute Product over Coproduct on Left

definition *dist-prod-coprod-left* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *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* $A\ B\ C : A \times_c (B \coprod C) \rightarrow (A \times_c B) \coprod (A \times_c C)$
$\wedge$ *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* $A\ B\ C = id\ (A \times_c (B \coprod C))$
⟨*proof*⟩

lemma *dist-prod-coprod-left-type*[*type-rule*]:
dist-prod-coprod-left $A\ B\ C : A \times_c (B \coprod C) \rightarrow (A \times_c B) \coprod (A \times_c C)$
⟨*proof*⟩

lemma *dist-factor-prod-coprod-left*:
dist-prod-coprod-left $A\ B\ C \circ_c$ *factor-prod-coprod-left* $A\ B\ C = id\ ((A \times_c B) \coprod (A \times_c C))$
⟨*proof*⟩

lemma *factor-dist-prod-coprod-left*:
factor-prod-coprod-left $A\ B\ C \circ_c$ *dist-prod-coprod-left* $A\ B\ C = id\ (A \times_c (B \coprod C))$
⟨*proof*⟩

lemma *dist-prod-coprod-left-iso*:
isomorphism(*dist-prod-coprod-left* $A\ B\ C$)
⟨*proof*⟩

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* $(A \times_c B)\ (A \times_c C) \circ_c \langle a,b\rangle$
⟨*proof*⟩

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* $(A \times_c B)\ (A \times_c C) \circ_c \langle a,c\rangle$
⟨*proof*⟩

9.4.3 Factor Product over Coproduct on Right

definition *factor-prod-coprod-right* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *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 $A\ B\ C : (A \times_c C) \coprod (B \times_c C) \rightarrow (A \coprod B) \times_c C$
⟨*proof*⟩

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* $A\ B \circ_c a,\ c\rangle$
$\langle proof \rangle$

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* $A\ B \circ_c b,\ c\rangle$
$\langle proof \rangle$

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 $A\ B\ C : (A \coprod B) \times_c C \to (A \times_c C) \coprod (B \times_c C)$
$\langle proof \rangle$

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* $(A \times_c C)\ (B \times_c C) \circ_c \langle a, c\rangle$
$\langle proof \rangle$

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* $(A \times_c C)\ (B \times_c C) \circ_c \langle b, c\rangle$
$\langle proof \rangle$

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* $(X \times_c H)\ (Y \times_c H)$
$\langle proof \rangle$

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* $(X \times_c H)\ (Y \times_c H)$
$\langle proof \rangle$

lemma *factor-dist-prod-coprod-right*:
factor-prod-coprod-right $A\ B\ C \circ_c$ *dist-prod-coprod-right* $A\ B\ C = id\ ((A \coprod B) \times_c C)$
$\langle proof \rangle$

lemma *dist-factor-prod-coprod-right*:
dist-prod-coprod-right $A\ B\ C \circ_c$ *factor-prod-coprod-right* $A\ B\ C = id\ ((A \times_c C)$

$\coprod\ (B \times_c C))$
⟨*proof*⟩

lemma *factor-prod-coprod-right-iso*:
isomorphism(*factor-prod-coprod-right* $A\ B\ C$)
⟨*proof*⟩

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 $m = m \amalg m^c$

lemma *into-super-type*[*type-rule*]:
monomorphism $m \Longrightarrow m : X \to Y \Longrightarrow$ *into-super* $m : X \coprod (Y \setminus (X,m)) \to Y$
⟨*proof*⟩

lemma *into-super-mono*:
assumes *monomorphism* $m\ m : X \to Y$
shows *monomorphism* (*into-super* m)
⟨*proof*⟩

lemma *into-super-epi*:
assumes *monomorphism* $m\ m : X \to Y$
shows *epimorphism* (*into-super* m)
⟨*proof*⟩

lemma *into-super-iso*:
assumes *monomorphism* $m\ m : X \to Y$
shows *isomorphism* (*into-super* m)
⟨*proof*⟩

9.5.2 Going from a Set to a Subset or its Complement

definition *try-cast* :: *cfunc* ⇒ *cfunc* **where**
try-cast $m = (THE\ m'.\ m' :$ *codomain* $m \to$ *domain* $m \coprod ((\textit{codomain}\ m) \setminus ((\textit{domain}\ m),m))$
$\wedge\ m' \circ_c$ *into-super* $m = id\ (\textit{domain}\ m \coprod (\textit{codomain}\ m \setminus ((\textit{domain}\ m),m)))$
$\wedge$ *into-super* $m \circ_c m' = id\ (\textit{codomain}\ m))$

lemma *try-cast-def2*:
assumes *monomorphism* $m\ m : X \to Y$
shows *try-cast* m : *codomain* $m \to (\textit{domain}\ m) \coprod ((\textit{codomain}\ m) \setminus ((\textit{domain}\ m),m))$
$\wedge$ *try-cast* $m \circ_c$ *into-super* $m = id\ ((\textit{domain}\ m) \coprod ((\textit{codomain}\ m) \setminus ((\textit{domain}\ m),m)))$
$\wedge$ *into-super* $m \circ_c$ *try-cast* $m = id\ (\textit{codomain}\ m)$

$\langle proof \rangle$

lemma *try-cast-type*[*type-rule*]:
assumes *monomorphism m* $m : X \to Y$
shows *try-cast m* $: Y \to X \coprod (Y \setminus (X,m))$
$\langle proof \rangle$

lemma *try-cast-into-super*:
assumes *monomorphism m* $m : X \to Y$
shows *try-cast m* $\circ_c$ *into-super m* $=$ *id* $(X \coprod (Y \setminus (X,m)))$
$\langle proof \rangle$

lemma *into-super-try-cast*:
assumes *monomorphism m* $m : X \to Y$
shows *into-super m* $\circ_c$ *try-cast m* $=$ *id Y*
$\langle proof \rangle$

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* $X\ (Y \setminus (X,m)) \circ_c x$
$\langle proof \rangle$

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 $\exists\ x.\ x \in_c Y \setminus (X,m) \wedge$ *try-cast m* $\circ_c y =$ *right-coproj* $X\ (Y \setminus (X,m)) \circ_c$ x
$\langle proof \rangle$

lemma *try-cast-m-m*:
assumes *m-type*: *monomorphism m* $m : X \to Y$
shows (*try-cast m*) $\circ_c m =$ *left-coproj* $X\ (Y \setminus (X,m))$
$\langle proof \rangle$

lemma *try-cast-m-m'*:
assumes *m-type*: *monomorphism m* $m : X \to Y$
shows (*try-cast m*) $\circ_c m^c =$ *right-coproj* $X\ (Y \setminus (X,m))$
$\langle proof \rangle$

lemma *try-cast-mono*:
assumes *m-type*: *monomorphism m* $m : X \to Y$
shows *monomorphism*(*try-cast m*)
$\langle proof \rangle$

9.6 Cases

definition *cases* :: *cfunc* $\Rightarrow$ *cfunc* **where**
cases(f) $=$ ((*right-cart-proj* **1** (*domain f*)) $\bowtie_f$ (*right-cart-proj* **1** (*domain f*))) $\circ_c$

$(\textit{dist-prod-coprod-right}\ \mathbf{1}\ \mathbf{1}\ (\textit{domain}\ f)) \circ_c \langle \textit{case-bool} \circ_c f,\ \textit{id}(\textit{domain}(f))\rangle$

lemma *cases-def2*:
assumes $f : X \to \Omega$
shows $\textit{cases}(f) = ((\textit{right-cart-proj}\ \mathbf{1}\ X) \bowtie_f (\textit{right-cart-proj}\ \mathbf{1}\ X)) \circ_c (\textit{dist-prod-coprod-right}\ \mathbf{1}\ \mathbf{1}\ X) \circ_c \langle \textit{case-bool} \circ_c f,\ \textit{id}\ X\rangle$
⟨*proof*⟩

lemma *cases-type*[*type-rule*]:
assumes $f : X \to \Omega$
shows $\textit{cases}(f) : X \to X \coprod X$
⟨*proof*⟩

lemma *true-case*:
assumes *x-type*[*type-rule*]: $x \in_c X$
assumes *f-type*[*type-rule*]: $f : X \to \Omega$
assumes *true-case*: $f \circ_c x = \mathrm{t}$
shows $\textit{cases}\ f \circ_c x = \textit{left-coproj}\ X\ X \circ_c x$
⟨*proof*⟩

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 $\textit{cases}\ f \circ_c x = \textit{right-coproj}\ X\ X \circ_c x$
⟨*proof*⟩

9.7 Coproduct Set Properities

lemma *coproduct-commutes*:
$A \coprod B \cong B \coprod A$
⟨*proof*⟩

lemma *coproduct-associates*:
$A \coprod (B \coprod C) \cong (A \coprod B) \coprod C$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.5.10.

lemma *product-distribute-over-coproduct-left*:
$A \times_c (X \coprod Y) \cong (A \times_c X) \coprod (A \times_c Y)$
⟨*proof*⟩

lemma *prod-pres-iso*:
assumes $A \cong C$ $B \cong D$
shows $A \times_c B \cong C \times_c D$
⟨*proof*⟩

lemma *coprod-pres-iso*:
assumes $A \cong C$ $B \cong D$
shows $A \coprod B \cong C \coprod D$

$\langle proof \rangle$

lemma *product-distribute-over-coproduct-right*:
$(A \coprod B) \times_c C \cong (A \times_c C) \coprod (B \times_c C)$
$\langle proof \rangle$

lemma *coproduct-with-self-iso*:
$X \coprod X \cong X \times_c \Omega$
$\langle proof \rangle$

lemma *oneUone-iso-*Ω:
$\Omega \cong \mathbf{1} \coprod \mathbf{1}$
$\langle proof \rangle$

The lemma below is dual to Proposition 2.2.2 in Halvorson.

lemma *card* $\{x.\ x \in_c \Omega \coprod \Omega\} = 4$
$\langle proof \rangle$

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* $\Rightarrow$ *cfunc* $\Rightarrow$ *bool* (**infix** *sectionof 90*)
where s *sectionof* $f \longleftrightarrow s : codomain\ f \to domain\ f \wedge f \circ_c s = id\ (codomain\ f)$

definition *split-epimorphism* :: *cfunc* $\Rightarrow$ *bool*
where *split-epimorphism* $f \longleftrightarrow (\exists\ s.\ \ s : codomain\ f \to domain\ f \wedge f \circ_c s = id\ (codomain\ f))$

lemma *split-epimorphism-def2*:
assumes *f-type*: $f : X \to Y$
assumes *f-split-epic*: *split-epimorphism* f
shows $\exists\ s.\ (f \circ_c s = id\ Y) \wedge (s\colon Y \to X)$
$\langle proof \rangle$

lemma *sections-define-splits*:
assumes s *sectionof* f
assumes $s : Y \to X$
shows $f : X \to Y \wedge$ *split-epimorphism*(f)
$\langle proof \rangle$

The axiomatization below corresponds to Axiom 11 (Axiom of Choice) in Halvorson.

axiomatization

where
axiom-of-choice: *epimorphism* $f \longrightarrow (\exists\ g\ .\ g\ \textit{sectionof}\ f)$

lemma *epis-give-monos*:
assumes *f-type*: $f : X \to Y$
assumes *f-epi*: *epimorphism f*
shows $\exists g.\ g\colon Y \to X \wedge \textit{monomorphism}\ g \wedge f \circ_c g = \textit{id}\ Y$
⟨*proof*⟩

corollary *epis-are-split*:
assumes *f-type*: $f : X \to Y$
assumes *f-epi*: *epimorphism f*
shows *split-epimorphism f*
⟨*proof*⟩

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 $\exists g.\ g\colon Y \to X \wedge \textit{epimorphism}\ g \wedge g \circ_c f = \textit{id}\ X$
⟨*proof*⟩

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*⟩

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*⟩

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* ($\emptyset$)
where
initial-func-type[*type-rule*]: *initial-func* X : $\emptyset \rightarrow X$ **and**
initial-func-unique: $h : \emptyset \rightarrow X \Longrightarrow h = $ *initial-func* X **and**
emptyset-is-empty: $\neg(x \in_c \emptyset)$

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

lemma *emptyset-is-initial*:
initial-object($\emptyset$)
⟨*proof*⟩

lemma *initial-iso-empty*:
assumes *initial-object*(X)
shows $X \cong \emptyset$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.4.6 in Halvorson.

lemma *coproduct-with-empty*:
shows $X \coprod \emptyset \cong X$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.4.7 in Halvorson.

lemma *function-to-empty-is-iso*:
assumes f: $X \rightarrow \emptyset$
shows *isomorphism*(f)
⟨*proof*⟩

lemma *empty-prod-X*:
$\emptyset \times_c X \cong \emptyset$
⟨*proof*⟩

lemma *X-prod-empty*:
$X \times_c \emptyset \cong \emptyset$
⟨*proof*⟩

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*⟩

lemma *initial-maps-mono*:
assumes *initial-object*(X)
assumes $f : X \rightarrow Y$
shows *monomorphism*(f)
⟨*proof*⟩

lemma *iso-empty-initial*:
assumes $X \cong \emptyset$
shows *initial-object* X
⟨*proof*⟩

lemma *function-to-empty-set-is-iso*:
assumes f: $X \rightarrow Y$
assumes *is-empty* Y
shows *isomorphism* f
⟨*proof*⟩

lemma *prod-iso-to-empty-right*:
assumes *nonempty* X
assumes $X \times_c Y \cong \emptyset$
shows *is-empty* Y
⟨*proof*⟩

lemma *prod-iso-to-empty-left*:
assumes *nonempty* Y
assumes $X \times_c Y \cong \emptyset$
shows *is-empty* X
⟨*proof*⟩

lemma *empty-subset*: $(\emptyset, \alpha_X) \subseteq_c X$
⟨*proof*⟩

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*⟩

lemma *coprod-with-init-obj1*:
assumes *initial-object* Y
shows $X \coprod Y \cong X$
⟨*proof*⟩

lemma *coprod-with-init-obj2*:
assumes *initial-object* X
shows $X \coprod Y \cong Y$
⟨*proof*⟩

lemma *prod-with-term-obj1*:
assumes *terminal-object*(X)
shows $X \times_c Y \cong Y$
⟨*proof*⟩

lemma *prod-with-term-obj2*:
assumes *terminal-object*(Y)
shows $X \times_c Y \cong X$

$\langle proof \rangle$

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 : $A \times_c X^A \to X$ **and**
transpose-func-type[*type-rule*]: $f : A \times_c Z \to X \Longrightarrow f^\sharp : Z \to X^A$ **and**
transpose-func-def: $f : A \times_c Z \to X \Longrightarrow$ (*eval-func* X A) $\circ_c$ (*id* A $\times_f$ $f^\sharp$) $= f$ **and**
transpose-func-unique:
$f : A \times_c Z \to X \Longrightarrow g$: $Z \to X^A \Longrightarrow$ (*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))
$\langle proof \rangle$

The lemma below corresponds to a note above Definition 2.5.1 in Halvorson.

lemma *exponential-object-identity*:
(*eval-func* X A)$^\sharp$ = $id_c(X^A)$
$\langle proof \rangle$

lemma *eval-func-X-empty-injective*:
assumes *is-empty* Y
shows *injective* (*eval-func* X Y)
$\langle proof \rangle$

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**

$exp\text{-}func\ g\ A = (g \circ_c eval\text{-}func\ (domain\ g)\ A)^\sharp$

lemma *exp-func-def2*:
assumes $g : X \to Y$
shows $exp\text{-}func\ g\ A = (g \circ_c eval\text{-}func\ X\ A)^\sharp$
⟨*proof*⟩

lemma *exp-func-type*[*type-rule*]:
assumes $g : X \to Y$
shows ${g^A}_f : X^A \to Y^A$
⟨*proof*⟩

lemma *exp-of-id-is-id-of-exp*:
$id(X^A) = {(id(X))^A}_f$
⟨*proof*⟩

The lemma below corresponds to a note below Definition 2.5.1 in Halvorson.

lemma *exponential-square-diagram*:
assumes $g : Y \to Z$
shows $(eval\text{-}func\ Z\ A) \circ_c (id_c(A) \times_f {g^A}_f) = g \circ_c (eval\text{-}func\ Y\ A)$
⟨*proof*⟩

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*⟩

lemma *exponential-object-identity2*:
${id(X)^A}_f = id_c(X^A)$
⟨*proof*⟩

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*:
$(eval\text{-}func\ (A \times_c X)\ A) \circ_c (id(A) \times_f (id(A \times_c X))^\sharp) = id(A \times_c X)$
⟨*proof*⟩
lemma *eval-of-id-cross-id-sharp2*:
assumes $a : Z \to A$ $x : Z \to X$
shows $((eval\text{-}func\ (A \times_c X)\ A) \circ_c (id(A) \times_f (id(A \times_c X))^\sharp)) \circ_c \langle a,x \rangle = \langle a,x \rangle$
⟨*proof*⟩

lemma *transpose-factors*:
assumes $f: X \to Y$
assumes $g: Y \to Z$
shows ${(g \circ_c f)^A}_f = ({g^A}_f) \circ_c ({f^A}_f)$
⟨*proof*⟩

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**
$f^\flat = (THE\ g.\ \exists\ Z\ X\ A.\ domain\ f = Z \wedge codomain\ f = X^A \wedge g = (eval\text{-}func\ X\ A) \circ_c (id\ A \times_f f))$

lemma *inv-transpose-func-def2*:
assumes $f : Z \to X^A$
shows $\exists\ Z\ X\ A.\ domain\ f = Z \wedge codomain\ f = X^A \wedge f^\flat = (eval\text{-}func\ X\ A) \circ_c (id\ A \times_f f)$
⟨*proof*⟩

lemma *inv-transpose-func-def3*:
assumes *f-type*: $f : Z \to X^A$
shows $f^\flat = (eval\text{-}func\ X\ A) \circ_c (id\ A \times_f f)$
⟨*proof*⟩

lemma *flat-type*[*type-rule*]:
assumes *f-type*[*type-rule*]: $f : Z \to X^A$
shows $f^\flat : A \times_c Z \to X$
⟨*proof*⟩

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 $(g \circ_c f)^\flat = g^\flat \circ_c (id(A) \times_f f)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.5.5 in Halvorson.

lemma *flat-cancels-sharp*:
$f : A \times_c Z \to X \Longrightarrow (f^\sharp)^\flat = f$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.5.6 in Halvorson.

lemma *sharp-cancels-flat*:
f: $Z \to X^A \Longrightarrow (f^\flat)^\sharp = f$
⟨*proof*⟩

lemma *same-evals-equal*:
assumes $f : Z \to X^A$ g: $Z \to X^A$
shows $eval\text{-}func\ X\ A \circ_c (id\ A \times_f f) = eval\text{-}func\ X\ A \circ_c (id\ A \times_f g) \Longrightarrow f = g$
⟨*proof*⟩

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*⟩

lemma *flat-pres-epi*:
  assumes *nonempty*(A)
  assumes $f : Z \to X^A$
  assumes *epimorphism* f
  shows *epimorphism*$(f^\flat)$
⟨*proof*⟩

lemma *transpose-inj-is-inj*:
  assumes g: $X \to Y$
  assumes *injective* g
  shows *injective*$({g^A}_f)$
  ⟨*proof*⟩

lemma *eval-func-X-one-injective*:
  injective (*eval-func* X **1**)
⟨*proof*⟩

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*$(f^\sharp)$
  ⟨*proof*⟩

12.3 Metafunctions and their Inverses (Cnufatems)

12.3.1 Metafunctions

definition *metafunc* :: *cfunc* ⇒ *cfunc* **where**
  metafunc $f \equiv (f \circ_c$ (*left-cart-proj* (*domain* f) **1**))$^\sharp$

lemma *metafunc-def2*:
  assumes $f : X \to Y$
  shows *metafunc* $f = (f \circ_c$ (*left-cart-proj* X **1**))$^\sharp$
  ⟨*proof*⟩

lemma *metafunc-type*[*type-rule*]:
  assumes $f : X \to Y$
  shows *metafunc* $f \in_c Y^X$
  ⟨*proof*⟩

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 $\circ_c$ ⟨x, *metafunc* f⟩ $= f \circ_c x$
⟨*proof*⟩

12.3.2 Inverse Metafunctions (Cnufatems)

definition *cnufatem* :: *cfunc* ⇒ *cfunc* **where**
cnufatem f = $(THE\ g.\ \forall\ Y\ X.\ f : \mathbf{1} \to Y^X \longrightarrow g = \textit{eval-func}\ Y\ X \circ_c \langle id\ X, f \circ_c \beta_X\rangle)$

lemma *cnufatem-def2*:
assumes $f \in_c Y^X$
shows $\textit{cnufatem}\ f = \textit{eval-func}\ Y\ X \circ_c \langle id\ X, f \circ_c \beta_X\rangle$
⟨*proof*⟩

lemma *cnufatem-type*[*type-rule*]:
assumes $f \in_c Y^X$
shows $\textit{cnufatem}\ f : X \to Y$
⟨*proof*⟩

lemma *cnufatem-metafunc*:
assumes *f-type*[*type-rule*]: $f : X \to Y$
shows $\textit{cnufatem}\ (\textit{metafunc}\ f) = f$
⟨*proof*⟩

lemma *metafunc-cnufatem*:
assumes *f-type*[*type-rule*]: $f \in_c Y^X$
shows $\textit{metafunc}\ (\textit{cnufatem}\ f) = f$
⟨*proof*⟩

12.3.3 Metafunction Composition

definition *meta-comp* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
$\textit{meta-comp}\ X\ Y\ Z = (\textit{eval-func}\ Z\ Y \circ_c \textit{swap}\ (Z^Y)\ Y \circ_c (id(Z^Y) \times_f (\textit{eval-func}\ Y\ X \circ_c \textit{swap}\ (Y^X)\ X)) \circ_c (\textit{associate-right}\ (Z^Y)\ (Y^X)\ X) \circ_c \textit{swap}\ X\ ((Z^Y) \times_c (Y^X)))^\sharp$

lemma *meta-comp-type*[*type-rule*]:
$\textit{meta-comp}\ X\ Y\ Z : Z^Y \times_c Y^X \to Z^X$
⟨*proof*⟩

definition *meta-comp2* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (**infixr** □ *55*)
where $\textit{meta-comp2}\ f\ g = (THE\ h.\ \exists\ W\ X\ Y.\ g : W \to Y^X \wedge h = (f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle)^\sharp)$

lemma *meta-comp2-def2*:
assumes f: $W \to Z^Y$
assumes g: $W \to Y^X$
shows $f \,\square\, g = (f^\flat \circ_c \langle g^\flat, \textit{right-cart-proj}\ X\ W\rangle)^\sharp$
⟨*proof*⟩

lemma *meta-comp2-type*[*type-rule*]:
assumes f: $W \to Z^Y$

assumes g: $W \to Y^X$
shows $f \Box g : W \to Z^X$
⟨*proof*⟩

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*⟩

lemma *meta-comp2-def3*:
assumes $f \in_c Z^Y$
assumes $g \in_c Y^X$
shows $f \Box g = \textit{metafunc } ((\textit{cnufatem } f) \circ_c (\textit{cnufatem } g))$
⟨*proof*⟩

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 $f \Box g = \textit{meta-comp } X\ Y\ Z \circ_c \langle f,g\rangle$
⟨*proof*⟩

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*⟩

lemma *meta-comp2-def5*:
assumes $f : W \to Z^Y$
assumes $g : W \to Y^X$
shows $f \Box g = \textit{meta-comp } X\ Y\ Z \circ_c \langle f,g\rangle$
⟨*proof*⟩

lemma *meta-left-identity*:
assumes $g \in_c X^X$
shows $g \Box \textit{metafunc } (id\ X) = g$
⟨*proof*⟩

lemma *meta-right-identity*:
assumes $g \in_c X^X$
shows $\textit{metafunc}(id\ X) \Box g = g$
⟨*proof*⟩

lemma *comp-as-metacomp*:
assumes $g : X \to Y$
assumes $f : Y \to Z$

shows $f \circ_c g = cnufatem(metafunc\ f \,\square\, metafunc\ g)$
⟨*proof*⟩

lemma *metacomp-as-comp*:
assumes $g \in_c Y^X$
assumes $f \in_c Z^Y$
shows $cnufatem\ f \circ_c cnufatem\ g = cnufatem(f \,\square\, g)$
⟨*proof*⟩

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*⟩

12.4 Partially Parameterized Functions on Pairs

definition *left-param* :: $cfunc \Rightarrow cfunc \Rightarrow cfunc$ $(\text{-}_{[\text{-},-]}\ [100,0]100)$ **where**
left-param $k\ p \equiv (THE\ f.\ \exists\ P\ Q\ R.\ k : P \times_c Q \to R \wedge f = k \circ_c \langle p \circ_c \beta_Q,\ id\ Q\rangle)$

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*⟩

lemma *left-param-type*[*type-rule*]:
assumes $k : P \times_c Q \to R$
assumes $p \in_c P$
shows $k_{[p,-]} : Q \to R$
⟨*proof*⟩

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*⟩

definition *right-param* :: $cfunc \Rightarrow cfunc \Rightarrow cfunc$ $(\text{-}_{[-,\text{-}]}\ [100,0]100)$ **where**
right-param $k\ q \equiv (THE\ f.\ \exists\ P\ Q\ R.\ k : P \times_c Q \to R \wedge f = k \circ_c \langle id\ P,\ q \circ_c \beta_P\rangle)$

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*⟩

lemma *right-param-type*[*type-rule*]:
assumes $k : P \times_c Q \rightarrow R$
assumes $q \in_c Q$
shows $k_{[-,q]} : P \rightarrow R$
⟨*proof*⟩

lemma *right-param-on-el*:
assumes $k : P \times_c Q \rightarrow R$
assumes $p \in_c P$
assumes $q \in_c Q$
shows $k_{[-,q]} \circ_c p = k \circ_c \langle p, q \rangle$
⟨*proof*⟩

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*⟩

The lemma below corresponds to Proposition 2.5.8 in Halvorson.

lemma *exp-empty*:
$X^{\emptyset} \cong \mathbf{1}$
⟨*proof*⟩

lemma *one-exp*:
$\mathbf{1}^X \cong \mathbf{1}$
⟨*proof*⟩

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*⟩

lemma *exponential-coprod-distribution*:
$Z^{(X \coprod Y)} \cong (Z^X) \times_c (Z^Y)$
⟨*proof*⟩

lemma *empty-exp-nonempty*:
assumes *nonempty* X
shows $\emptyset^X \cong \emptyset$
⟨*proof*⟩

lemma *exp-pres-iso-left*:
assumes $A \cong X$
shows $A^Y \cong X^Y$
⟨*proof*⟩

lemma *expset-power-tower*:
$(A^B)^C \cong A^{(B \times_c C)}$
⟨*proof*⟩

lemma *exp-pres-iso-right*:
assumes $A \cong X$
shows $Y^A \cong Y^X$
⟨*proof*⟩

lemma *exp-pres-iso*:
assumes $A \cong X$ $B \cong Y$
shows $A^B \cong X^Y$
⟨*proof*⟩

lemma *empty-to-nonempty*:
assumes *nonempty* X *is-empty* Y
shows $Y^X \cong \emptyset$
⟨*proof*⟩

lemma *exp-is-empty*:
assumes *is-empty* X
shows $Y^X \cong \mathbf{1}$
⟨*proof*⟩

lemma *nonempty-to-nonempty*:
assumes *nonempty* X *nonempty* Y
shows *nonempty*(Y^X)
⟨*proof*⟩

lemma *empty-to-nonempty-converse*:
assumes $Y^X \cong \emptyset$
shows *is-empty* $Y \wedge$ *nonempty* X
⟨*proof*⟩

The definition below corresponds to Definition 2.5.11 in Halvorson.

definition *powerset* :: *cset* ⇒ *cset* ($\mathcal{P}$- [*101*]*100*) **where**
$\mathcal{P}\ X = \Omega^X$

lemma *sets-squared*:
$A^\Omega \cong A \times_c A$
⟨*proof*⟩

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*]: $successor \colon \mathbb{N}_c \to \mathbb{N}_c$ **and**
natural-number-object-property:
$q : \mathbf{1} \to X \Longrightarrow f \colon X \to X \Longrightarrow$
$(\exists!u.\ u \colon \mathbb{N}_c \to X \wedge$
$q = u \circ_c zero \wedge$
$f \circ_c u = u \circ_c successor)$

lemma *beta-N-succ-nEqs-Id1*:
assumes *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
shows $\beta_{\mathbb{N}_c} \circ_c successor \circ_c n = id\ \mathbf{1}$
⟨*proof*⟩

lemma *natural-number-object-property2*:
assumes $q : \mathbf{1} \to X\ f \colon X \to X$
shows $\exists!u.\ u \colon \mathbb{N}_c \to X \wedge u \circ_c zero = q \wedge f \circ_c u = u \circ_c successor$
⟨*proof*⟩

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*: $f \colon X \to X$
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 $u = v$
⟨*proof*⟩

definition *is-NNO* :: *cset* ⇒ *cfunc* ⇒ *cfunc* ⇒ *bool* **where**
$is\text{-}NNO\ Y\ z\ s \longleftrightarrow (z \colon \mathbf{1} \to Y \wedge s \colon Y \to Y \wedge (\forall\ X\ f\ q.\ ((q : \mathbf{1} \to X) \wedge (f \colon X \to X)) \longrightarrow$
$(\exists!u.\ u \colon Y \to X \wedge$
$q = u \circ_c z \wedge$
$f \circ_c u = u \circ_c s)))$

lemma *N-is-a-NNO*:
is-NNO $\mathbb{N}_c$ *zero successor*
⟨*proof*⟩

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*⟩

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 $\exists\ z\ s.\ \textit{is-NNO}\ N\ z\ s$
⟨*proof*⟩

13.1 Zero and Successor

lemma *zero-is-not-successor*:
assumes $n \in_c \mathbb{N}_c$
shows $\textit{zero} \neq \textit{successor} \circ_c n$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.6.6 in Halvorson.

lemma *oneUN-iso-N-isomorphism*:
$\textit{isomorphism}(\textit{zero} \amalg \textit{successor})$
⟨*proof*⟩

lemma *zUs-epic*:
$\textit{epimorphism}(\textit{zero} \amalg \textit{successor})$
⟨*proof*⟩

lemma *zUs-surj*:
$\textit{surjective}(\textit{zero} \amalg \textit{successor})$
⟨*proof*⟩

lemma *nonzero-is-succ-aux*:
assumes $x \in_c (\mathbf{1} \coprod \mathbb{N}_c)$
shows $(x = (\textit{left-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c \textit{id}\ \mathbf{1}) \vee$
$(\exists\, n.\ (n \in_c \mathbb{N}_c) \wedge (x = (\textit{right-coproj}\ \mathbf{1}\ \mathbb{N}_c) \circ_c n))$
⟨*proof*⟩

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*⟩

13.2 Predecessor

definition *predecessor'* :: *cfunc* **where**
$\textit{predecessor}' = (\textit{THE}\, f.\ f : \mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$
$\wedge f \circ_c (\textit{zero} \amalg \textit{successor}) = \textit{id}\ (\mathbf{1} \coprod \mathbb{N}_c) \wedge\ (\textit{zero} \amalg \textit{successor}) \circ_c f = \textit{id}\ \mathbb{N}_c)$

lemma *predecessor'-def2*:
$\textit{predecessor}' : \mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c \wedge \textit{predecessor}' \circ_c (\textit{zero} \amalg \textit{successor}) = \textit{id}\ (\mathbf{1} \coprod \mathbb{N}_c)$

$\wedge$ (*zero* $\amalg$ *successor*) $\circ_c$ *predecessor'* $=$ *id* $\mathbb{N}_c$
⟨*proof*⟩

lemma *predecessor'-type*[*type-rule*]:
predecessor' : $\mathbb{N}_c \to \mathbf{1} \coprod \mathbb{N}_c$
⟨*proof*⟩

lemma *predecessor'-left-inv*:
(*zero* $\amalg$ *successor*) $\circ_c$ *predecessor'* $=$ *id* $\mathbb{N}_c$
⟨*proof*⟩

lemma *predecessor'-right-inv*:
predecessor' $\circ_c$ (*zero* $\amalg$ *successor*) $=$ *id* $(\mathbf{1} \coprod \mathbb{N}_c)$
⟨*proof*⟩

lemma *predecessor'-successor*:
predecessor' $\circ_c$ *successor* $=$ *right-coproj* $\mathbf{1}$ $\mathbb{N}_c$
⟨*proof*⟩

lemma *predecessor'-zero*:
predecessor' $\circ_c$ *zero* $=$ *left-coproj* $\mathbf{1}$ $\mathbb{N}_c$
⟨*proof*⟩

definition *predecessor* :: *cfunc*
where *predecessor* $=$ (*zero* $\amalg$ *id* $\mathbb{N}_c$) $\circ_c$ *predecessor'*

lemma *predecessor-type*[*type-rule*]:
predecessor : $\mathbb{N}_c \to \mathbb{N}_c$
⟨*proof*⟩

lemma *predecessor-zero*:
predecessor $\circ_c$ *zero* $=$ *zero*
⟨*proof*⟩

lemma *predecessor-successor*:
predecessor $\circ_c$ *successor* $=$ *id* $\mathbb{N}_c$
⟨*proof*⟩

13.3 Peano's Axioms and Induction

The lemma below corresponds to Proposition 2.6.7 in Halvorson.

lemma *Peano's-Axioms*:
injective successor $\wedge \neg$ *surjective successor*
⟨*proof*⟩

lemma *succ-inject*:
assumes $n \in_c \mathbb{N}_c$ $m \in_c \mathbb{N}_c$
shows *successor* $\circ_c n =$ *successor* $\circ_c m \Longrightarrow n = m$
⟨*proof*⟩

theorem *nat-induction*:
assumes *p-type*[*type-rule*]: $p : \mathbb{N}_c \to \Omega$ **and** *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
assumes *base-case*: $p \circ_c \mathit{zero} = \mathrm{t}$
assumes *induction-case*: $\bigwedge n.\ n \in_c \mathbb{N}_c \Longrightarrow p \circ_c n = \mathrm{t} \Longrightarrow p \circ_c \mathit{successor} \circ_c n = \mathrm{t}$
shows $p \circ_c n = \mathrm{t}$
⟨*proof*⟩

13.4 Function Iteration

definition *ITER-curried* :: *cset* ⇒ *cfunc* **where**
$\mathit{ITER\text{-}curried}\ U = (\mathit{THE}\ u\ .\ u : \mathbb{N}_c \to (U^U)^{U^U} \wedge u \circ_c \mathit{zero} = (\mathit{metafunc}\ (\mathit{id}\ U) \circ_c (\mathit{right\text{-}cart\text{-}proj}\ (U^U)\ \mathbf{1}))^\sharp \wedge$
$((\mathit{meta\text{-}comp}\ U\ U\ U) \circ_c (\mathit{id}\ (U^U) \times_f \mathit{eval\text{-}func}\ (U^U)\ (U^U)) \circ_c (\mathit{associate\text{-}right}\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (\mathit{diagonal}(U^U) \times_f \mathit{id}\ ((U^U)^{U^U})))^\sharp \circ_c u = u \circ_c \mathit{successor})$

lemma *ITER-curried-def2*:
$\mathit{ITER\text{-}curried}\ U : \mathbb{N}_c \to (U^U)^{U^U} \wedge \mathit{ITER\text{-}curried}\ U \circ_c \mathit{zero} = (\mathit{metafunc}\ (\mathit{id}\ U) \circ_c (\mathit{right\text{-}cart\text{-}proj}\ (U^U)\ \mathbf{1}))^\sharp \wedge$
$((\mathit{meta\text{-}comp}\ U\ U\ U) \circ_c (\mathit{id}\ (U^U) \times_f \mathit{eval\text{-}func}\ (U^U)\ (U^U)) \circ_c (\mathit{associate\text{-}right}\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (\mathit{diagonal}(U^U) \times_f \mathit{id}\ ((U^U)^{U^U})))^\sharp \circ_c \mathit{ITER\text{-}curried}\ U = \mathit{ITER\text{-}curried}\ U \circ_c \mathit{successor}$
⟨*proof*⟩

lemma *ITER-curried-type*[*type-rule*]:
$\mathit{ITER\text{-}curried}\ U : \mathbb{N}_c \to (U^U)^{U^U}$
⟨*proof*⟩

lemma *ITER-curried-zero*:
$\mathit{ITER\text{-}curried}\ U \circ_c \mathit{zero} = (\mathit{metafunc}\ (\mathit{id}\ U) \circ_c (\mathit{right\text{-}cart\text{-}proj}\ (U^U)\ \mathbf{1}))^\sharp$
⟨*proof*⟩

lemma *ITER-curried-successor*:
$\mathit{ITER\text{-}curried}\ U \circ_c \mathit{successor} = (\mathit{meta\text{-}comp}\ U\ U\ U \circ_c (\mathit{id}\ (U^U) \times_f \mathit{eval\text{-}func}\ (U^U)\ (U^U)) \circ_c (\mathit{associate\text{-}right}\ (U^U)\ (U^U)\ ((U^U)^{U^U})) \circ_c (\mathit{diagonal}(U^U) \times_f \mathit{id}\ ((U^U)^{U^U})))^\sharp \circ_c \mathit{ITER\text{-}curried}\ U$
⟨*proof*⟩

definition *ITER* :: *cset* ⇒ *cfunc* **where**
$\mathit{ITER}\ U = (\mathit{ITER\text{-}curried}\ U)^\flat$

lemma *ITER-type*[*type-rule*]:
$\mathit{ITER}\ U : ((U^U) \times_c \mathbb{N}_c) \to (U^U)$
⟨*proof*⟩

lemma *ITER-zero*:
assumes *f-type*[*type-rule*]: $f : Z \to (U^U)$
shows $ITER\ U \circ_c \langle f,\ zero \circ_c \beta_Z\rangle = metafunc\ (id\ U) \circ_c \beta_Z$
⟨*proof*⟩

lemma *ITER-zero'*:
assumes $f \in_c (U^U)$
shows $ITER\ U \circ_c \langle f,\ zero\rangle = metafunc\ (id\ U)$
⟨*proof*⟩

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*⟩

lemma *ITER-one*:
assumes $f \in_c (U^U)$
shows $ITER\ U \circ_c \langle f,\ successor \circ_c zero\rangle = f \,\square\, (metafunc\ (id\ U))$
⟨*proof*⟩

definition *iter-comp* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (-$^{\circ}$-[*55*,*0*]*55*) **where**
iter-comp g n $\equiv$ $cnufatem\ (ITER\ (domain\ g) \circ_c \langle metafunc\ g, n\rangle)$

lemma *iter-comp-def2*:
$g^{\circ n} \equiv cnufatem(ITER\ (domain\ g) \circ_c \langle metafunc\ g, n\rangle)$
⟨*proof*⟩

lemma *iter-comp-type*[*type-rule*]:
assumes $g : X \to X$
assumes $n \in_c \mathbb{N}_c$
shows $g^{\circ n}$: $X \to X$
⟨*proof*⟩

lemma *iter-comp-def3*:
assumes $g : X \to X$
assumes $n \in_c \mathbb{N}_c$
shows $g^{\circ n} = cnufatem\ (ITER\ X \circ_c \langle metafunc\ g, n\rangle)$
⟨*proof*⟩

lemma *zero-iters*:
assumes *g-type*[*type-rule*]: $g : X \to X$
shows $g^{\circ zero} = id_c\ X$
⟨*proof*⟩

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*⟩

corollary *one-iter*:
  assumes $g : X \to X$
  shows $g^{\circ(successor \circ_c zero)} = g$
  ⟨*proof*⟩

lemma *eval-lemma-for-ITER*:
  assumes $f : X \to X$
  assumes $x \in_c X$
  assumes $m \in_c \mathbb{N}_c$
  shows $(f^{\circ m}) \circ_c x = eval\text{-}func\ X\ X \circ_c \langle x\ ,ITER\ X \circ_c \langle metafunc\ f\ ,m\rangle\rangle$
  ⟨*proof*⟩

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*⟩

lemma *n-accessible-by-succ-iter*:
  assumes $n \in_c \mathbb{N}_c$
  shows $(successor^{\circ n}) \circ_c zero = n$
⟨*proof*⟩

13.5 Relation of Nat to Other Sets

lemma *oneUN-iso-N*:
  $\mathbf{1} \coprod \mathbb{N}_c \cong \mathbb{N}_c$
  ⟨*proof*⟩

lemma *NUone-iso-N*:
  $\mathbb{N}_c \coprod \mathbf{1} \cong \mathbb{N}_c$
  ⟨*proof*⟩

end

14 Predicate Logic Functions

theory *Pred-Logic*
  imports *Coproduct*
begin

14.1 NOT

definition *NOT* :: *cfunc* **where**
  $NOT = (THE\ \chi.\ is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ \Omega\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \mathrm{f}\ \chi)$

lemma *NOT-is-pullback*:
  $is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ \Omega\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \mathrm{f}\ NOT$

$\langle proof \rangle$

lemma *NOT-type*[*type-rule*]:
$NOT : \Omega \rightarrow \Omega$
$\langle proof \rangle$

lemma *NOT-false-is-true*:
$NOT \circ_c \mathrm{f} = \mathrm{t}$
$\langle proof \rangle$

lemma *NOT-true-is-false*:
$NOT \circ_c \mathrm{t} = \mathrm{f}$
$\langle proof \rangle$

lemma *NOT-is-true-implies-false*:
assumes $p \in_c \Omega$
shows $NOT \circ_c p = \mathrm{t} \Longrightarrow p = \mathrm{f}$
$\langle proof \rangle$

lemma *NOT-is-false-implies-true*:
assumes $p \in_c \Omega$
shows $NOT \circ_c p = \mathrm{f} \Longrightarrow p = \mathrm{t}$
$\langle proof \rangle$

lemma *double-negation*:
$NOT \circ_c NOT = \ id\ \Omega$
$\langle proof \rangle$

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*:
$is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \langle \mathrm{t},\mathrm{t} \rangle\ AND$
$\langle proof \rangle$

lemma *AND-type*[*type-rule*]:
$AND : \Omega \times_c \Omega \rightarrow \Omega$
$\langle proof \rangle$

lemma *AND-true-true-is-true*:
$AND \circ_c \langle \mathrm{t},\mathrm{t} \rangle = \mathrm{t}$
$\langle proof \rangle$

lemma *AND-false-left-is-false*:
assumes $p \in_c \Omega$
shows $AND \circ_c \langle \mathrm{f},p \rangle = \mathrm{f}$
$\langle proof \rangle$

lemma *AND-false-right-is-false*:
  assumes $p \in_c \Omega$
  shows $AND \circ_c \langle p, \mathrm{f} \rangle = \mathrm{f}$
$\langle proof \rangle$

lemma *AND-commutative*:
  assumes $p \in_c \Omega$
  assumes $q \in_c \Omega$
  shows $AND \circ_c \langle p, q \rangle = AND \circ_c \langle q, p \rangle$
  $\langle proof \rangle$

lemma *AND-idempotent*:
  assumes $p \in_c \Omega$
  shows $AND \circ_c \langle p, p \rangle = p$
  $\langle proof \rangle$

lemma *AND-associative*:
  assumes $p \in_c \Omega$
  assumes $q \in_c \Omega$
  assumes $r \in_c \Omega$
  shows $AND \circ_c \langle AND \circ_c \langle p, q \rangle, r \rangle = AND \circ_c \langle p, AND \circ_c \langle q, r \rangle \rangle$
  $\langle proof \rangle$

lemma *AND-complementary*:
  assumes $p \in_c \Omega$
  shows $AND \circ_c \langle p, NOT \circ_c p \rangle = \mathrm{f}$
  $\langle proof \rangle$

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*:
  $is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ \langle \mathrm{f}, \mathrm{f} \rangle\ NOR$
  $\langle proof \rangle$

lemma *NOR-type*[*type-rule*]:
  $NOR : \Omega \times_c \Omega \to \Omega$
  $\langle proof \rangle$

lemma *NOR-false-false-is-true*:
  $NOR \circ_c \langle \mathrm{f}, \mathrm{f} \rangle = \mathrm{t}$
  $\langle proof \rangle$

lemma *NOR-left-true-is-false*:
  assumes $p \in_c \Omega$
  shows $NOR \circ_c \langle \mathrm{t}, p \rangle = \mathrm{f}$

⟨*proof*⟩

lemma *NOR-right-true-is-false*:
  assumes $p \in_c \Omega$
  shows $NOR \circ_c \langle p,\mathrm{t}\rangle = \mathrm{f}$
⟨*proof*⟩

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 $P = \mathrm{f} \circ_c \beta_X \wedge Q = \mathrm{f} \circ_c \beta_Y$
⟨*proof*⟩

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 $\neg (P = \mathrm{t} \circ_c \beta_X \vee Q = \mathrm{t} \circ_c \beta_Y)$
  ⟨*proof*⟩

14.4 OR

definition *OR* :: *cfunc* **where**
  $OR = (THE\ \chi.\ is\text{-}pullback\ (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))\ \mathbf{1}\ (\Omega\times_c\Omega)\ \Omega\ (\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))})\ \mathrm{t}\ (\langle \mathrm{t},\mathrm{t}\rangle\amalg$ $(\langle \mathrm{t},\mathrm{f}\rangle\ \amalg\langle \mathrm{f},\mathrm{t}\rangle))\ \chi)$

lemma *pre-OR-type*[*type-rule*]:
  $\langle \mathrm{t},\mathrm{t}\rangle\amalg\ (\langle \mathrm{t},\mathrm{f}\rangle\ \amalg\langle \mathrm{f},\mathrm{t}\rangle) : \mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \to \Omega \times_c \Omega$
  ⟨*proof*⟩

lemma *set-three*:
  $\{x.\ x \in_c (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))\} = \{$
 $(left\text{-}coproj\ \mathbf{1}\ (\mathbf{1}\coprod\mathbf{1}))$,
 $(right\text{-}coproj\ \mathbf{1}\ (\mathbf{1}\coprod\mathbf{1}) \circ_c left\text{-}coproj\ \mathbf{1}\ \mathbf{1})$,
  $right\text{-}coproj\ \mathbf{1}\ (\mathbf{1}\coprod\mathbf{1}) \circ_c (right\text{-}coproj\ \mathbf{1}\ \mathbf{1})\}$
  ⟨*proof*⟩

lemma *set-three-card*:
 $card\ \{x.\ x \in_c (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))\} = 3$
⟨*proof*⟩

lemma *pre-OR-injective*:
  $injective(\langle \mathrm{t},\mathrm{t}\rangle\amalg\ (\langle \mathrm{t},\mathrm{f}\rangle\ \amalg\langle \mathrm{f},\mathrm{t}\rangle))$
  ⟨*proof*⟩

lemma *OR-is-pullback*:
  $is\text{-}pullback\ (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))\ \mathbf{1}\ (\Omega\times_c\Omega)\ \Omega\ (\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))})\ \mathrm{t}\ (\langle \mathrm{t},\mathrm{t}\rangle\amalg\ (\langle \mathrm{t},\mathrm{f}\rangle\ \amalg\langle \mathrm{f},\mathrm{t}\rangle))$ OR

$\langle proof \rangle$

lemma *OR-type*[*type-rule*]:
$OR : \Omega \times_c \Omega \to \Omega$
$\langle proof \rangle$

lemma *OR-true-left-is-true*:
assumes $p \in_c \Omega$
shows $OR \circ_c \langle \mathrm{t}, p \rangle = \mathrm{t}$
$\langle proof \rangle$

lemma *OR-true-right-is-true*:
assumes $p \in_c \Omega$
shows $OR \circ_c \langle p, \mathrm{t} \rangle = \mathrm{t}$
$\langle proof \rangle$

lemma *OR-false-false-is-false*:
$OR \circ_c \langle \mathrm{f}, \mathrm{f} \rangle = \mathrm{f}$
$\langle proof \rangle$

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 $(p = \mathrm{t}) \vee (q = \mathrm{t})$
$\langle proof \rangle$

lemma *NOT-NOR-is-OR*:
$OR = NOT \circ_c NOR$
$\langle proof \rangle$

lemma *OR-commutative*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $OR \circ_c \langle p, q \rangle = OR \circ_c \langle q, p \rangle$
$\langle proof \rangle$

lemma *OR-idempotent*:
assumes $p \in_c \Omega$
shows $OR \circ_c \langle p, p \rangle = p$
$\langle proof \rangle$

lemma *OR-associative*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $r \in_c \Omega$
shows $OR \circ_c \langle OR \circ_c \langle p, q \rangle, r \rangle = OR \circ_c \langle p, OR \circ_c \langle q, r \rangle \rangle$
$\langle proof \rangle$

lemma *OR-complementary*:
assumes $p \in_c \Omega$
shows $OR \circ_c \langle p,\ NOT \circ_c p\rangle = \mathrm{t}$
$\langle proof \rangle$

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*]:
$\langle \mathrm{t},\ \mathrm{f}\rangle\ \amalg\ \langle \mathrm{f},\ \mathrm{t}\rangle : \mathbf{1}\coprod\mathbf{1} \to \Omega \times_c \Omega$
$\langle proof \rangle$

lemma *pre-XOR-injective*:
$injective(\langle \mathrm{t},\ \mathrm{f}\rangle\ \amalg\langle \mathrm{f},\ \mathrm{t}\rangle)$
$\langle proof \rangle$

lemma *XOR-is-pullback*:
$is\text{-}pullback\ (\mathbf{1}\coprod\mathbf{1})\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1}\coprod\mathbf{1})})\ \mathrm{t}\ (\langle \mathrm{t},\ \mathrm{f}\rangle\ \amalg\ \langle \mathrm{f},\ \mathrm{t}\rangle)\ XOR$
$\langle proof \rangle$

lemma *XOR-type*[*type-rule*]:
$XOR : \Omega \times_c \Omega \to \Omega$
$\langle proof \rangle$

lemma *XOR-only-true-left-is-true*:
$XOR \circ_c \langle \mathrm{t},\mathrm{f}\rangle = \mathrm{t}$
$\langle proof \rangle$

lemma *XOR-only-true-right-is-true*:
$XOR \circ_c \langle \mathrm{f},\mathrm{t}\rangle = \mathrm{t}$
$\langle proof \rangle$

lemma *XOR-false-false-is-false*:
$XOR \circ_c \langle \mathrm{f},\mathrm{f}\rangle = \mathrm{f}$
$\langle proof \rangle$

lemma *XOR-true-true-is-false*:
$XOR \circ_c \langle \mathrm{t},\mathrm{t}\rangle = \mathrm{f}$
$\langle proof \rangle$

14.6 NAND

definition *NAND* :: *cfunc* **where**
$NAND = (THE\ \chi.\ is\text{-}pullback\ (\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}))})\ \mathrm{t}\ (\langle \mathrm{f},\ \mathrm{f}\rangle\amalg\ (\langle \mathrm{t},\ \mathrm{f}\rangle\ \amalg\langle \mathrm{f},\ \mathrm{t}\rangle))\ \chi)$

lemma *pre-NAND-type*[*type-rule*]:

$\langle$f, f$\rangle$ II ($\langle$t, f$\rangle$ II $\langle$f, t$\rangle$) : $\mathbf{1}\coprod(\mathbf{1}\coprod\mathbf{1}) \rightarrow \Omega \times_c \Omega$
⟨*proof*⟩

lemma *pre-NAND-injective*:
injective($\langle$f, f$\rangle$ II ($\langle$t, f$\rangle$ II $\langle$f, t$\rangle$))
⟨*proof*⟩

lemma *NAND-is-pullback*:
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 ($\langle$f, f$\rangle$II ($\langle$t, f$\rangle$ II$\langle$f, t$\rangle$))
NAND
⟨*proof*⟩

lemma *NAND-type*[*type-rule*]:
NAND : $\Omega \times_c \Omega \rightarrow \Omega$
⟨*proof*⟩

lemma *NAND-left-false-is-true*:
assumes $p \in_c \Omega$
shows *NAND* $\circ_c$ $\langle$f,$p\rangle$ = t
⟨*proof*⟩

lemma *NAND-right-false-is-true*:
assumes $p \in_c \Omega$
shows *NAND* $\circ_c$ $\langle p$,f$\rangle$ = t
⟨*proof*⟩

lemma *NAND-true-true-is-false*:
NAND $\circ_c$ $\langle$t,t$\rangle$ = f
⟨*proof*⟩

lemma *NAND-true-implies-one-is-false*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes *NAND* $\circ_c$ $\langle p,q\rangle$ = t
shows p = f $\vee$ q = f
⟨*proof*⟩

lemma *NOT-AND-is-NAND*:
NAND = *NOT* $\circ_c$ *AND*
⟨*proof*⟩

lemma *NAND-not-idempotent*:
assumes $p \in_c \Omega$
shows *NAND* $\circ_c$ $\langle p,p\rangle$ = *NOT* $\circ_c$ p
⟨*proof*⟩

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*]:
$\langle t, t\rangle\ \amalg\ \langle f, f\rangle : \mathbf{1} \coprod \mathbf{1} \to \Omega \times_c \Omega$
⟨*proof*⟩

lemma *pre-IFF-injective*:
$injective(\langle t, t\rangle\ \amalg \langle f, f\rangle)$
⟨*proof*⟩

lemma *IFF-is-pullback*:
$is\text{-}pullback\ (\mathbf{1} \coprod \mathbf{1})\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1} \coprod \mathbf{1})})\ t\ (\langle t, t\rangle\ \amalg \langle f, f\rangle)\ IFF$
⟨*proof*⟩

lemma *IFF-type*[*type-rule*]:
$IFF : \Omega \times_c \Omega \to \Omega$
⟨*proof*⟩

lemma *IFF-true-true-is-true*:
$IFF \circ_c \langle t,t\rangle = t$
⟨*proof*⟩

lemma *IFF-false-false-is-true*:
$IFF \circ_c \langle f,f\rangle = t$
⟨*proof*⟩

lemma *IFF-true-false-is-false*:
$IFF \circ_c \langle t,f\rangle = f$
⟨*proof*⟩

lemma *IFF-false-true-is-false*:
$IFF \circ_c \langle f,t\rangle = f$
⟨*proof*⟩

lemma *NOT-IFF-is-XOR*:
$NOT \circ_c IFF = XOR$
⟨*proof*⟩

14.8 IMPLIES

definition *IMPLIES* :: *cfunc* **where**
$IMPLIES = (THE\ \chi.\ is\text{-}pullback\ (\mathbf{1} \coprod (\mathbf{1} \coprod \mathbf{1}))\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1} \coprod (\mathbf{1} \coprod \mathbf{1}))})\ t\ (\langle t, t\rangle \amalg\ (\langle f, f\rangle\ \amalg \langle f, t\rangle))\ \chi)$

lemma *pre-IMPLIES-type*[*type-rule*]:
$\langle t, t\rangle\ \amalg\ (\langle f, f\rangle\ \amalg\ \langle f, t\rangle) : \mathbf{1} \coprod (\mathbf{1} \coprod \mathbf{1}) \to \Omega \times_c \Omega$
⟨*proof*⟩

lemma *pre-IMPLIES-injective*:

$injective(\langle \mathrm{t}, \mathrm{t}\rangle \amalg (\langle \mathrm{f}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle))$
⟨*proof*⟩

lemma *IMPLIES-is-pullback*:
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 $(\langle \mathrm{t}, \mathrm{t}\rangle \amalg (\langle \mathrm{f}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle))$
IMPLIES
⟨*proof*⟩

lemma *IMPLIES-type*[*type-rule*]:
$IMPLIES : \Omega \times_c \Omega \to \Omega$
⟨*proof*⟩

lemma *IMPLIES-true-true-is-true*:
$IMPLIES \circ_c \langle \mathrm{t},\mathrm{t}\rangle = \mathrm{t}$
⟨*proof*⟩

lemma *IMPLIES-false-true-is-true*:
$IMPLIES \circ_c \langle \mathrm{f},\mathrm{t}\rangle = \mathrm{t}$
⟨*proof*⟩

lemma *IMPLIES-false-false-is-true*:
$IMPLIES \circ_c \langle \mathrm{f},\mathrm{f}\rangle = \mathrm{t}$
⟨*proof*⟩

lemma *IMPLIES-true-false-is-false*:
$IMPLIES \circ_c \langle \mathrm{t},\mathrm{f}\rangle = \mathrm{f}$
⟨*proof*⟩

lemma *IMPLIES-false-is-true-false*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $IMPLIES \circ_c \langle p,q\rangle = \mathrm{f}$
shows $p = \mathrm{t} \wedge q = \mathrm{f}$
⟨*proof*⟩

ETCS analog to $(A \iff B) = (A \implies B) \wedge (B \implies A)$

lemma *iff-is-and-implies-implies-swap*:
$IFF = AND \circ_c \langle IMPLIES, IMPLIES \circ_c swap\ \Omega\ \Omega\rangle$
⟨*proof*⟩

lemma *IMPLIES-is-OR-NOT-id*:
$IMPLIES = OR \circ_c (NOT \times_f id(\Omega))$
⟨*proof*⟩

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 $P = \mathrm{t} \circ_c \beta_X \Longrightarrow Q = \mathrm{t} \circ_c \beta_Y$

$\langle proof \rangle$

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 $(P = \mathrm{t} \circ_c \beta_X) \Longrightarrow ((Q = \mathrm{t} \circ_c \beta_Y) \Longrightarrow R) \Longrightarrow R$
$\langle proof \rangle$

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 $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
$\langle proof \rangle$

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 $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
$\langle proof \rangle$

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 $(P = \mathrm{t} \Longrightarrow Q = \mathrm{t}) \Longrightarrow IMPLIES \circ_c \langle P,\ Q \rangle = \mathrm{t}$
$\langle proof \rangle$

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 $AND \circ_c \langle p,\ OR \circ_c \langle q,r \rangle\rangle = OR \circ_c \langle AND \circ_c \langle p,q \rangle,\ AND \circ_c \langle p,r \rangle\rangle$
$\langle proof \rangle$

lemma *OR-AND-distributive*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes $r \in_c \Omega$
shows $OR \circ_c \langle p,\ AND \circ_c \langle q,r \rangle\rangle = AND \circ_c \langle OR \circ_c \langle p,q \rangle,\ OR \circ_c \langle p,r \rangle\rangle$
$\langle proof \rangle$

lemma *OR-AND-absorption*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $OR \circ_c \langle p,\ AND \circ_c \langle p,q \rangle\rangle = p$
$\langle proof \rangle$

lemma *AND-OR-absorption*:

assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $AND \circ_c \langle p,\ OR \circ_c \langle p,q\rangle\rangle = p$
$\langle proof \rangle$

lemma *deMorgan-Law1*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $NOT \circ_c OR \circ_c \langle p,q\rangle = AND \circ_c \langle NOT \circ_c p,\ NOT \circ_c q\rangle$
$\langle proof \rangle$

lemma *deMorgan-Law2*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $NOT \circ_c AND \circ_c \langle p,q\rangle = OR \circ_c \langle NOT \circ_c p,\ NOT \circ_c q\rangle$
$\langle proof \rangle$

end

15 Quantifiers

theory *Quant-Logic*
imports *Pred-Logic Exponential-Objects*
begin

15.1 Universal Quantification

definition *FORALL* :: *cset* ⇒ *cfunc* **where**
$FORALL\ X = (THE\ \chi.\ is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega^X)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ ((\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp)\ \chi)$

lemma *FORALL-is-pullback*:
$is\text{-}pullback\ \mathbf{1}\ \mathbf{1}\ (\Omega^X)\ \Omega\ (\beta_{\mathbf{1}})\ \mathrm{t}\ ((\mathrm{t} \circ_c \beta_{X \times_c \mathbf{1}})^\sharp)\ (FORALL\ X)$
$\langle proof \rangle$

lemma *FORALL-type*[*type-rule*]:
$FORALL\ X : \Omega^X \to \Omega$
$\langle proof \rangle$

lemma *all-true-implies-FORALL-true*:
assumes *p-type*[*type-rule*]: $p : X \to \Omega$ **and** *all-p-true*: $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c x = \mathrm{t}$
shows $FORALL\ X \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ \mathbf{1})^\sharp = \mathrm{t}$
$\langle proof \rangle$

lemma *all-true-implies-FORALL-true2*:
assumes *p-type*[*type-rule*]: $p : X \times_c Y \to \Omega$ **and** *all-p-true*: $\bigwedge xy.\ xy \in_c X \times_c Y \Longrightarrow p \circ_c xy = \mathrm{t}$
shows $FORALL\ X \circ_c p^\sharp = \mathrm{t} \circ_c \beta_Y$
$\langle proof \rangle$

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*⟩

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*⟩

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*⟩

lemma *FORALL-true-implies-all-true3*:
  assumes *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \to \Omega$ **and** *FORALL-p-true*: $FORALL\ X \circ_c p^\sharp = \mathrm{t}$
  shows $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c \langle x,\ id\ \mathbf{1}\rangle = \mathrm{t}$
  ⟨*proof*⟩

lemma *FORALL-elim*:
  assumes *FORALL-p-true*: $FORALL\ X \circ_c p^\sharp = \mathrm{t}$ **and** *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \to \Omega$
  assumes *x-type*[*type-rule*]: $x \in_c X$
  shows $(p \circ_c \langle x,\ id\ \mathbf{1}\rangle = \mathrm{t} \Longrightarrow P) \Longrightarrow P$
  ⟨*proof*⟩

lemma *FORALL-elim'*:
  assumes *FORALL-p-true*: $FORALL\ X \circ_c p^\sharp = \mathrm{t}$ **and** *p-type*[*type-rule*]: $p : X \times_c \mathbf{1} \to \Omega$
  shows $((\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c \langle x,\ id\ \mathbf{1}\rangle = \mathrm{t}) \Longrightarrow P) \Longrightarrow P$
  ⟨*proof*⟩

15.2 Existential Quantification

definition *EXISTS* :: *cset* ⇒ *cfunc* **where**
  $EXISTS\ X = NOT \circ_c FORALL\ X \circ_c NOT^X{}_f$

lemma *EXISTS-type*[*type-rule*]:
  $EXISTS\ X : \Omega^X \to \Omega$
  ⟨*proof*⟩

lemma *EXISTS-true-implies-exists-true*:
  assumes *p-type*: $p : X \to \Omega$ **and** *EXISTS-p-true*: $EXISTS\ X \circ_c (p \circ_c left\text{-}cart\text{-}proj$

X $\mathbf{1})^\sharp = \mathrm{t}$
shows $\exists\ x.\ x \in_c X \wedge p \circ_c x = \mathrm{t}$
$\langle proof \rangle$

lemma *EXISTS-elim*:
assumes *EXISTS-p-true*: *EXISTS* $X \circ_c (p \circ_c$ *left-cart-proj* X $\mathbf{1})^\sharp = \mathrm{t}$ **and** *p-type*: $p : X \to \Omega$
shows $(\bigwedge\ x.\ x \in_c X \Longrightarrow p \circ_c x = \mathrm{t} \Longrightarrow Q) \Longrightarrow Q$
$\langle proof \rangle$

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* $X \circ_c (p \circ_c$ *left-cart-proj* X $\mathbf{1})^\sharp = \mathrm{t}$
$\langle proof \rangle$

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-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-even: $\mathbb{N}_c \to \mathbb{N}_c \wedge$ *nth-even* $\circ_c$ *zero* = *zero* $\wedge$ (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-even* = *nth-even* $\circ_c$ *successor*
$\langle proof \rangle$

lemma *nth-even-type*[*type-rule*]:
nth-even: $\mathbb{N}_c \to \mathbb{N}_c$
$\langle proof \rangle$

lemma *nth-even-zero*:
nth-even $\circ_c$ *zero* = *zero*
$\langle proof \rangle$

lemma *nth-even-successor*:
nth-even $\circ_c$ *successor* = (*successor* $\circ_c$ *successor*) $\circ_c$ *nth-even*
$\langle proof \rangle$

lemma *nth-even-successor2*:
nth-even $\circ_c$ *successor* = *successor* $\circ_c$ *successor* $\circ_c$ *nth-even*

$\langle proof \rangle$

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*:
$nth\text{-}odd: \mathbb{N}_c \to \mathbb{N}_c \wedge nth\text{-}odd \circ_c zero = successor \circ_c zero \wedge (successor \circ_c successor) \circ_c nth\text{-}odd = nth\text{-}odd \circ_c successor$
$\langle proof \rangle$

lemma *nth-odd-type*[*type-rule*]:
$nth\text{-}odd: \mathbb{N}_c \to \mathbb{N}_c$
$\langle proof \rangle$

lemma *nth-odd-zero*:
$nth\text{-}odd \circ_c zero = successor \circ_c zero$
$\langle proof \rangle$

lemma *nth-odd-successor*:
$nth\text{-}odd \circ_c successor = (successor \circ_c successor) \circ_c nth\text{-}odd$
$\langle proof \rangle$

lemma *nth-odd-successor2*:
$nth\text{-}odd \circ_c successor = successor \circ_c successor \circ_c nth\text{-}odd$
$\langle proof \rangle$

lemma *nth-odd-is-succ-nth-even*:
$nth\text{-}odd = successor \circ_c nth\text{-}even$
$\langle proof \rangle$

lemma *succ-nth-odd-is-nth-even-succ*:
$successor \circ_c nth\text{-}odd = nth\text{-}even \circ_c successor$
$\langle proof \rangle$

16.3 Checking if a Number is Even

definition *is-even* :: *cfunc* **where**
$is\text{-}even = (THE\ u.\ u: \mathbb{N}_c \to \Omega \wedge u \circ_c zero = \mathrm{t} \wedge NOT \circ_c u = u \circ_c successor)$

lemma *is-even-def2*:
$is\text{-}even : \mathbb{N}_c \to \Omega \wedge is\text{-}even \circ_c zero = \mathrm{t} \wedge NOT \circ_c is\text{-}even = is\text{-}even \circ_c successor$
$\langle proof \rangle$

lemma *is-even-type*[*type-rule*]:
$is\text{-}even : \mathbb{N}_c \to \Omega$
$\langle proof \rangle$

lemma *is-even-zero*:
$is\text{-}even \circ_c zero = \mathrm{t}$
⟨*proof*⟩

lemma *is-even-successor*:
$is\text{-}even \circ_c successor = NOT \circ_c is\text{-}even$
⟨*proof*⟩

16.4 Checking if a Number is Odd

definition *is-odd* :: *cfunc* **where**
$is\text{-}odd = (THE\ u.\ u\colon \mathbb{N}_c \to \Omega \wedge u \circ_c zero = \mathrm{f} \wedge NOT \circ_c u = u \circ_c successor)$

lemma *is-odd-def2*:
$is\text{-}odd : \mathbb{N}_c \to \Omega \wedge is\text{-}odd \circ_c zero = \mathrm{f} \wedge NOT \circ_c is\text{-}odd = is\text{-}odd \circ_c successor$
⟨*proof*⟩

lemma *is-odd-type*[*type-rule*]:
$is\text{-}odd : \mathbb{N}_c \to \Omega$
⟨*proof*⟩

lemma *is-odd-zero*:
$is\text{-}odd \circ_c zero = \mathrm{f}$
⟨*proof*⟩

lemma *is-odd-successor*:
$is\text{-}odd \circ_c successor = NOT \circ_c is\text{-}odd$
⟨*proof*⟩

lemma *is-even-not-is-odd*:
$is\text{-}even = NOT \circ_c is\text{-}odd$
⟨*proof*⟩

lemma *is-odd-not-is-even*:
$is\text{-}odd = NOT \circ_c is\text{-}even$
⟨*proof*⟩

lemma *not-even-and-odd*:
assumes $m \in_c \mathbb{N}_c$
shows $\neg(is\text{-}even \circ_c m = \mathrm{t} \wedge is\text{-}odd \circ_c m = \mathrm{t})$
⟨*proof*⟩

lemma *even-or-odd*:
assumes $n \in_c \mathbb{N}_c$
shows $is\text{-}even \circ_c n = \mathrm{t} \vee is\text{-}odd \circ_c n = \mathrm{t}$
⟨*proof*⟩

lemma *is-even-nth-even-true*:

$\textit{is-even} \circ_c \textit{nth-even} = \mathrm{t} \circ_c \beta_{\mathbb{N}_c}$
⟨*proof*⟩

lemma *is-odd-nth-odd-true*:
$\textit{is-odd} \circ_c \textit{nth-odd} = \mathrm{t} \circ_c \beta_{\mathbb{N}_c}$
⟨*proof*⟩

lemma *is-odd-nth-even-false*:
$\textit{is-odd} \circ_c \textit{nth-even} = \mathrm{f} \circ_c \beta_{\mathbb{N}_c}$
⟨*proof*⟩

lemma *is-even-nth-odd-false*:
$\textit{is-even} \circ_c \textit{nth-odd} = \mathrm{f} \circ_c \beta_{\mathbb{N}_c}$
⟨*proof*⟩

lemma *EXISTS-zero-nth-even*:
$(\textit{EXISTS}\ \mathbb{N}_c \circ_c (\textit{eq-pred}\ \mathbb{N}_c \circ_c \textit{nth-even} \times_f \textit{id}_c\ \mathbb{N}_c)^\sharp) \circ_c \textit{zero} = \mathrm{t}$
⟨*proof*⟩

lemma *not-EXISTS-zero-nth-odd*:
$(\textit{EXISTS}\ \mathbb{N}_c \circ_c (\textit{eq-pred}\ \mathbb{N}_c \circ_c \textit{nth-odd} \times_f \textit{id}_c\ \mathbb{N}_c)^\sharp) \circ_c \textit{zero} = \mathrm{f}$
⟨*proof*⟩

16.5 Natural Number Halving

definition *halve-with-parity* :: *cfunc* **where**
$\textit{halve-with-parity} = (\textit{THE}\ u.\ u\colon \mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c \wedge$
$u \circ_c \textit{zero} = \textit{left-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \textit{zero} \wedge$
$(\textit{right-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \amalg (\textit{left-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \textit{successor})) \circ_c u = u \circ_c \textit{successor})$

lemma *halve-with-parity-def2*:
$\textit{halve-with-parity} : \mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c \wedge$
$\textit{halve-with-parity} \circ_c \textit{zero} = \textit{left-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \textit{zero} \wedge$
$(\textit{right-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \amalg (\textit{left-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \textit{successor})) \circ_c \textit{halve-with-parity} = \textit{halve-with-parity} \circ_c \textit{successor}$
⟨*proof*⟩

lemma *halve-with-parity-type*[*type-rule*]:
$\textit{halve-with-parity} : \mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c$
⟨*proof*⟩

lemma *halve-with-parity-zero*:
$\textit{halve-with-parity} \circ_c \textit{zero} = \textit{left-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \textit{zero}$
⟨*proof*⟩

lemma *halve-with-parity-successor*:
$(\textit{right-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \amalg (\textit{left-coproj}\ \mathbb{N}_c\ \mathbb{N}_c \circ_c \textit{successor})) \circ_c \textit{halve-with-parity} = \textit{halve-with-parity} \circ_c \textit{successor}$
⟨*proof*⟩

lemma *halve-with-parity-nth-even*:
  $\mathit{halve\text{-}with\text{-}parity} \circ_c \mathit{nth\text{-}even} = \mathit{left\text{-}coproj}\ \mathbb{N}_c\ \mathbb{N}_c$
⟨*proof*⟩

lemma *halve-with-parity-nth-odd*:
  $\mathit{halve\text{-}with\text{-}parity} \circ_c \mathit{nth\text{-}odd} = \mathit{right\text{-}coproj}\ \mathbb{N}_c\ \mathbb{N}_c$
⟨*proof*⟩

lemma *nth-even-nth-odd-halve-with-parity*:
  $(\mathit{nth\text{-}even} \amalg \mathit{nth\text{-}odd}) \circ_c \mathit{halve\text{-}with\text{-}parity} = \mathit{id}\ \mathbb{N}_c$
⟨*proof*⟩

lemma *halve-with-parity-nth-even-nth-odd*:
  $\mathit{halve\text{-}with\text{-}parity} \circ_c (\mathit{nth\text{-}even} \amalg \mathit{nth\text{-}odd}) = \mathit{id}\ (\mathbb{N}_c \coprod \mathbb{N}_c)$
  ⟨*proof*⟩

lemma *even-odd-iso*:
  $\mathit{isomorphism}\ (\mathit{nth\text{-}even} \amalg \mathit{nth\text{-}odd})$
  ⟨*proof*⟩

lemma *halve-with-parity-iso*:
  $\mathit{isomorphism}\ \mathit{halve\text{-}with\text{-}parity}$
   ⟨*proof*⟩

definition *halve* :: *cfunc* **where**
  $\mathit{halve} = (\mathit{id}\ \mathbb{N}_c \amalg \mathit{id}\ \mathbb{N}_c) \circ_c \mathit{halve\text{-}with\text{-}parity}$

lemma *halve-type*[*type-rule*]:
  $\mathit{halve} : \mathbb{N}_c \to \mathbb{N}_c$
  ⟨*proof*⟩

lemma *halve-nth-even*:
  $\mathit{halve} \circ_c \mathit{nth\text{-}even} = \mathit{id}\ \mathbb{N}_c$
  ⟨*proof*⟩

lemma *halve-nth-odd*:
  $\mathit{halve} \circ_c \mathit{nth\text{-}odd} = \mathit{id}\ \mathbb{N}_c$
  ⟨*proof*⟩

lemma *is-even-def3*:
  $\mathit{is\text{-}even} = ((\mathrm{t} \circ_c \beta_{\mathbb{N}_c}) \amalg (\mathrm{f} \circ_c \beta_{\mathbb{N}_c})) \circ_c \mathit{halve\text{-}with\text{-}parity}$
⟨*proof*⟩

lemma *is-odd-def3*:
  $\mathit{is\text{-}odd} = ((\mathrm{f} \circ_c \beta_{\mathbb{N}_c}) \amalg (\mathrm{t} \circ_c \beta_{\mathbb{N}_c})) \circ_c \mathit{halve\text{-}with\text{-}parity}$
⟨*proof*⟩

lemma *nth-even-or-nth-odd*:

assumes $n \in_c \mathbb{N}_c$
shows $(\exists\ m.\ m \in_c \mathbb{N}_c \wedge \textit{nth-even} \circ_c m = n) \vee (\exists\ m.\ m \in_c \mathbb{N}_c \wedge \textit{nth-odd} \circ_c m = n)$
$\langle \textit{proof} \rangle$

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 $\exists m.\ m \in_c \mathbb{N}_c \wedge n = \textit{nth-even} \circ_c m$
$\langle \textit{proof} \rangle$

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 $\exists m.\ m \in_c \mathbb{N}_c \wedge n = \textit{nth-odd} \circ_c m$
$\langle \textit{proof} \rangle$

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**
$\textit{is-finite}\ X \longleftrightarrow (\forall m.\ (m : X \to X \wedge \textit{monomorphism}\ m) \longrightarrow \textit{isomorphism}\ m)$

definition *is-infinite* :: *cset* ⇒ *bool* **where**
$\textit{is-infinite}\ X \longleftrightarrow (\exists\ m.\ m : X \to X \wedge \textit{monomorphism}\ m \wedge \neg \textit{surjective}\ m)$

lemma *either-finite-or-infinite*:
$\textit{is-finite}\ X \vee \textit{is-infinite}\ X$
$\langle \textit{proof} \rangle$

The definition below corresponds to Definition 2.6.2 in Halvorson.

definition *is-smaller-than* :: *cset* ⇒ *cset* ⇒ *bool* (**infix** $\leq_c$ *50*) **where**
$X \leq_c Y \longleftrightarrow (\exists\ m.\ m : X \to Y \wedge \textit{monomorphism}\ m)$

The purpose of the following lemma is simply to unify the two notations used in the book.

lemma *subobject-iff-smaller-than*:
$(X \leq_c Y) = (\exists m.\ (X,m) \subseteq_c Y)$
$\langle \textit{proof} \rangle$

lemma *set-card-transitive*:
assumes $A \leq_c B$
assumes $B \leq_c C$
shows $A \leq_c C$
$\langle \textit{proof} \rangle$

lemma *all-emptysets-are-finite*:
assumes *is-empty* X
shows *is-finite* X
⟨*proof*⟩

lemma *emptyset-is-smallest-set*:
$\emptyset \leq_c X$
⟨*proof*⟩

lemma *truth-set-is-finite*:
is-finite Ω
⟨*proof*⟩

lemma *smaller-than-finite-is-finite*:
assumes $X \leq_c Y$ *is-finite* Y
shows *is-finite* X
⟨*proof*⟩

lemma *larger-than-infinite-is-infinite*:
assumes $X \leq_c Y$ *is-infinite* X
shows *is-infinite* Y
⟨*proof*⟩

lemma *iso-pres-finite*:
assumes $X \cong Y$
assumes *is-finite* X
shows *is-finite* Y
⟨*proof*⟩

lemma *not-finite-and-infinite*:
$\neg($*is-finite* $X \wedge$ *is-infinite* $X)$
⟨*proof*⟩

lemma *iso-pres-infinite*:
assumes $X \cong Y$
assumes *is-infinite* X
shows *is-infinite* Y
⟨*proof*⟩

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*⟩

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*⟩

lemma *not-init-not-term*:
$(\neg(\textit{initial-object } X) \wedge \neg(\textit{terminal-object } X)) = (\exists\ x1.\ \exists\ x2.\ x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2)$
$\langle \textit{proof} \rangle$

lemma *sets-size-3-plus*:
$(\neg(\textit{initial-object } X) \wedge \neg(\textit{terminal-object } X) \wedge \neg(X \cong \Omega)) = (\exists\ x1.\ \exists\ x2.\ \exists\ x3.\ x1 \in_c X \wedge x2 \in_c X \wedge x3 \in_c X \wedge x1 \neq x2 \wedge x2 \neq x3 \wedge x1 \neq x3)$
$\langle \textit{proof} \rangle$

The next two lemmas below correspond to Proposition 2.6.3 in Halvorson.

lemma *smaller-than-coproduct1*:
$X \leq_c X \coprod Y$
$\langle \textit{proof} \rangle$

lemma *smaller-than-coproduct2*:
$X \leq_c Y \coprod X$
$\langle \textit{proof} \rangle$

The next two lemmas below correspond to Proposition 2.6.4 in Halvorson.

lemma *smaller-than-product1*:
assumes *nonempty* Y
shows $X \leq_c X \times_c Y$
$\langle \textit{proof} \rangle$

lemma *smaller-than-product2*:
assumes *nonempty* Y
shows $X \leq_c Y \times_c X$
$\langle \textit{proof} \rangle$

lemma *coprod-leq-product*:
assumes *X-not-init*: $\neg(\textit{initial-object}(X))$
assumes *Y-not-init*: $\neg(\textit{initial-object}(Y))$
assumes *X-not-term*: $\neg(\textit{terminal-object}(X))$
assumes *Y-not-term*: $\neg(\textit{terminal-object}(Y))$
shows $X \coprod Y \leq_c X \times_c Y$
$\langle \textit{proof} \rangle$

lemma *prod-leq-exp*:
assumes $\neg$ *terminal-object* Y
shows $X \times_c Y \leq_c Y^X$
$\langle \textit{proof} \rangle$

lemma *Y-nonempty-then-X-le-XtoY*:
assumes *nonempty* Y
shows $X \leq_c X^Y$
$\langle \textit{proof} \rangle$

lemma *non-init-non-ter-sets*:
assumes $\neg(\textit{terminal-object}\ X)$
assumes $\neg(\textit{initial-object}\ X)$
shows $\Omega \leq_c X$
⟨*proof*⟩

lemma *exp-preserves-card1*:
assumes $A \leq_c B$
assumes *nonempty* X
shows $X^A \leq_c X^B$
⟨*proof*⟩

lemma *exp-preserves-card2*:
assumes $A \leq_c B$
shows $A^X \leq_c B^X$
⟨*proof*⟩

lemma *exp-preserves-card3*:
assumes $A \leq_c B$
assumes $X \leq_c Y$
assumes $\textit{nonempty}(X)$
shows $X^A \leq_c Y^B$
⟨*proof*⟩

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**
$\textit{epi-countable}\ X \longleftrightarrow (\exists\ f.\ f : \mathbb{N}_c \to X \land \textit{epimorphism}\ f)$

lemma *emptyset-is-not-epi-countable*:
$\neg\ \textit{epi-countable}\ \emptyset$
⟨*proof*⟩

The fact that the empty set is not countable according to the definition from Halvorson ($\textit{epi-countable}\ ?X = (\exists f.\ f : \mathbb{N}_c \to ?X \land \textit{epimorphism}\ f)$) motivated the following definition.

definition *countable* :: *cset* ⇒ *bool* **where**
$\textit{countable}\ X \longleftrightarrow (\exists\ f.\ f : X \to \mathbb{N}_c \land \textit{monomorphism}\ f)$

lemma *epi-countable-is-countable*:

assumes *epi-countable X*
shows *countable X*
⟨*proof*⟩

lemma *emptyset-is-countable*:
countable $\emptyset$
⟨*proof*⟩

lemma *natural-numbers-are-countably-infinite*:
countable $\mathbb{N}_c \wedge$ *is-infinite* $\mathbb{N}_c$
⟨*proof*⟩

lemma *iso-to-N-is-countably-infinite*:
assumes $X \cong \mathbb{N}_c$
shows *countable X* $\wedge$ *is-infinite X*
⟨*proof*⟩

lemma *smaller-than-countable-is-countable*:
assumes $X \leq_c Y$ *countable Y*
shows *countable X*
⟨*proof*⟩

lemma *iso-pres-countable*:
assumes $X \cong Y$ *countable Y*
shows *countable X*
⟨*proof*⟩

lemma *NuN-is-countable*:
countable$(\mathbb{N}_c \coprod \mathbb{N}_c)$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.6.11 in Halvorson.

lemma *coproduct-of-countables-is-countable*:
assumes *countable X countable Y*
shows *countable*$(X \coprod Y)$
⟨*proof*⟩

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* $\Rightarrow$ *cfunc* $\Rightarrow$ *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* $\Rightarrow$ *bool* **where**

has-fixed-point g $\longleftrightarrow$ ($\exists$ *a*. *fixed-point a g*)
definition *fixed-point-property* :: *cset* $\Rightarrow$ *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* = ($g \circ_c a = a$)
⟨*proof*⟩

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*
⟨*proof*⟩

The theorem below corresponds to Theorem 2.6.14 in Halvorson.

theorem *Cantors-Negative-Theorem*:
$\nexists$ *s*. $s : X \to \mathcal{P}\ X$ $\wedge$ *surjective s*
⟨*proof*⟩

The theorem below corresponds to Exercise 2.6.15 in Halvorson.

theorem *Cantors-Positive-Theorem*:
$\exists$ *m*. $m : X \to \Omega^X$ $\wedge$ *injective m*
⟨*proof*⟩

The corollary below corresponds to Corollary 2.6.16 in Halvorson.

corollary
$X \leq_c \mathcal{P}\ X \wedge \neg\ (X \cong \mathcal{P}\ X)$
⟨*proof*⟩

corollary *Generalized-Cantors-Positive-Theorem*:
assumes $\neg$ *terminal-object Y*
assumes $\neg$ *initial-object Y*
shows $X \leq_c Y^X$
⟨*proof*⟩

corollary *Generalized-Cantors-Negative-Theorem*:
assumes $\neg$ *initial-object X*
assumes $\neg$ *terminal-object Y*
shows $\nexists$ *s*. $s : X \to Y^X$ $\wedge$ *surjective s*
⟨*proof*⟩

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.