The Elementary Theory of the Category of Sets

James Baxter Dustin Bryant

August 17, 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

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

1 Basic types and operators for the category of sets

typedecl *cset*
typedecl *cfunc*

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

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

axiomatization
domain :: *cfunc* ⇒ *cset* **and**
codomain :: *cfunc* ⇒ *cset* **and**

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

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

definition *cfunc-type* :: *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool* (- : - → - [*50*, *50*, *50*]*50*)
where
($f : X \to Y$) ⟷ (*domain*(*f*) = *X* ∧ *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*: *id X* : $X \to X$
⟨*proof*⟩

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

lemma *id-left-unit2*: $f : X \to Y \Longrightarrow$ *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*]

$\langle ML \rangle$

1.1.1 typecheck_cfuncs: Tactic to construct type facts

$\langle ML \rangle$

1.1.2 etcs_rule: Tactic to apply rules with ETCS typechecking

$\langle ML \rangle$

1.1.3 etcs_subst: Tactic to apply substitutions with ETCS typechecking

$\langle ML \rangle$

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

$\langle ML \rangle$

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

$\langle ML \rangle$

1.2 Monomorphisms, Epimorphisms and Isomorphisms

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*:
$monomorphism\ f \longleftrightarrow (\forall\ g\ h\ A\ X\ Y.\ g : A \to X \wedge h : A \to X \wedge f : X \to Y \longrightarrow (f \circ_c g = f \circ_c h \longrightarrow g = h))$
$\langle proof \rangle$

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

definition *epimorphism* :: *cfunc* ⇒ *bool* **where**
$epimorphism\ f \longleftrightarrow (\forall\ g\ h.$

$(domain(g) = codomain(f) \wedge domain(h) = codomain(f)) \longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$

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

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

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

lemma *isomorphism-def2*:
$isomorphism(f) \longleftrightarrow (\exists\ g\ X\ Y.\ f : X \to Y \wedge g : Y \to X \wedge g \circ_c f = id\ X \wedge f \circ_c g = id\ Y)$
$\langle proof \rangle$

lemma *isomorphism-def3*:
assumes $f : X \to Y$
shows $isomorphism(f) \longleftrightarrow (\exists\ g.\ g : Y \to X \wedge g \circ_c f = id\ X \wedge f \circ_c g = id\ Y)$
$\langle proof \rangle$

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

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

lemma *inverse-type*[*type-rule*]:
assumes $isomorphism(f)\ f : X \to Y$
shows $f^{\mathbf{-1}} : Y \to X$
$\langle proof \rangle$

lemma *inv-left*:
assumes $isomorphism(f)\ f : X \to Y$
shows $f^{\mathbf{-1}} \circ_c f = id\ X$
$\langle proof \rangle$

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

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

lemma *inv-idempotent*:
assumes $isomorphism(f)$
shows $(f^{-\mathbf{1}})^{-\mathbf{1}} = f$
⟨*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 2.1.7a 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 2.1.7b in Halvorson.

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

The lemma below corresponds to Exercise 2.1.7d in Halvorson.

lemma *composition-of-epi-pair-is-epi*:
assumes *codomain f* = *domain g*
  shows *epimorphism f* $\Longrightarrow$ *epimorphism g* $\Longrightarrow$ *epimorphism* $(g \circ_c f)$
  ⟨*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
theory *Product*
  imports *Cfunc*
begin

2 Cartesian products of sets

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

axiomatization
  cart-prod :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* (**infixr** $\times_c$ *65*) **and**

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

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

abbreviation *is-cart-prod-triple* :: *cset* × *cfunc* × *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool* **where**
is-cart-prod-triple $W\pi$ X Y ≡ *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
∧ *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 $(\textit{left-cart-proj } X\ Y \circ_c a = \textit{left-cart-proj } X\ Y \circ_c b$
    $\wedge\ \textit{right-cart-proj } X\ Y \circ_c a = \textit{right-cart-proj } X\ Y \circ_c b)$
  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 function

The definition below corresponds to Definition 2.1.9 in Halvorson.

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

lemma *diagonal-type*[*type-rule*]:
  $\textit{diagonal } X : X \to X \times_c X$
  ⟨*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**
  $f \times_f g = \langle f \circ_c \textit{left-cart-proj } (\textit{domain } f)\ (\textit{domain } g),\ g \circ_c \textit{right-cart-proj } (\textit{domain } f)\ (\textit{domain } g)\rangle$

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

lemma *swap-type*[*type-rule*]: *swap* X Y : $X \times_c Y \to Y \times_c X$
⟨*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-right X Y Z =
⟨
left-cart-proj X Y $\circ_c$ *left-cart-proj* $(X \times_c Y)$ Z,
⟨
right-cart-proj X Y $\circ_c$ *left-cart-proj* $(X \times_c Y)$ Z,
right-cart-proj $(X \times_c Y)$ Z
⟩
⟩

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

lemma *associate-right-ap*:

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

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

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

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 $\circ_c \langle x, \langle y, z\rangle\rangle = \langle x, z\rangle$
⟨*proof*⟩

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

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

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

lemma *distribute-left-mono*:
monomorphism (*distribute-left* X Y Z)
⟨*proof*⟩

2.3.6 Selecting pairs from a pair of pairs

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

lemma *outers-type*[*type-rule*]: *outers* A B C D : $(A \times_c B) \times_c (C \times_c D) \to (A \times_c D)$

⟨*proof*⟩

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

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

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

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

definition *lefts* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
lefts $A\ B\ C\ D = \langle$
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* $(A \times_c B)\ (C \times_c D)$
$\rangle$

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

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

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

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

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

⟨*proof*⟩

end
theory *Terminal*
  imports *Cfunc Product*
begin

3 Terminal objects, constant functions and elements

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

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

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

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

lemma *terminal-func-comp-elem*:
  $x : one \to X \Longrightarrow \beta_X \circ_c x = id\ one$
  ⟨*proof*⟩

3.1 Set membership and emptiness

The abbreviation below captures Definition 2.1.16 in Halvorson.

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

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

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

The lemma below corresponds to Exercise 2.1.18 in Halvorson.

lemma *element-monomorphism*:
$x \in_c X \Longrightarrow$ *monomorphism* x
⟨*proof*⟩

lemma *one-unique-element*:
$\exists!\ x.\ x \in_c$ *one*
⟨*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*(*one*)
⟨*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 $X \cong$ *one*
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*:
$(\exists!\ x.\ x \in_c X) \longleftrightarrow X \cong one$
⟨*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 $\neg(\mathit{nonempty}\ X) \vee \neg(\mathit{nonempty}\ Z)$
shows $\mathit{injective}\ (f \times_f g)$
$\langle \mathit{proof} \rangle$

3.4 Surjectivity

The definition below corresponds to Definition 2.1.28 in Halvorson.

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

lemma *surjective-def2*:
assumes $f : X \to Y$
shows $\mathit{surjective}\ f \longleftrightarrow (\forall\, y.\ y \in_c Y \longrightarrow (\exists\, x.\ x \in_c X \wedge f \circ_c x = y))$
$\langle \mathit{proof} \rangle$

The lemma below corresponds to Exercise 2.1.30 in Halvorson.

lemma *surjective-is-epimorphism*:
$\mathit{surjective}\ f \Longrightarrow \mathit{epimorphism}\ f$
$\langle \mathit{proof} \rangle$

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 $\mathit{surjective}\ (f \times_f g)$
$\langle \mathit{proof} \rangle$

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

3.5 Interactions of cartesian products with terminal objects

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

lemma *one-cross-one-unique-element*:
$\exists!\ x.\ x \in_c \mathit{one} \times_c \mathit{one}$
$\langle \mathit{proof} \rangle$

The lemma below corresponds to Proposition 2.1.20 in Halvorson.

lemma *X-is-cart-prod1*:
$\mathit{is\text{-}cart\text{-}prod}\ X\ (\mathit{id}\ X)\ (\beta_X)\ X\ \mathit{one}$

$\langle proof \rangle$

lemma *X-is-cart-prod2*:
$is\text{-}cart\text{-}prod\ X\ (\beta_X)\ (id\ X)\ one\ X$
$\langle proof \rangle$

lemma *A-x-one-iso-A*:
$X \times_c one \cong X$
$\langle proof \rangle$

lemma *one-x-A-iso-A*:
$one \times_c X \cong X$
$\langle proof \rangle$

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

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

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

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

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

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

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

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-cart-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 $f : one \to X$ $g : one \to Y$
shows $\langle f,\ g\rangle = \langle id\ X,\ g \circ_c \beta_X\rangle \circ_c f$
⟨*proof*⟩

lemma *cart-prod-extract-right*:
assumes $f : one \to X$ $g : one \to Y$
shows $\langle f,\ g\rangle = \langle f \circ_c \beta_Y,\ id\ Y\rangle \circ_c g$
⟨*proof*⟩

end
theory *Equalizer*
imports *Terminal*
begin

4 Equalizers and Subobjects

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 $f : X \rightarrow Y$ $g : X \rightarrow Y$ $m : E \rightarrow X$
  assumes *equalizer* E m f g
  assumes $h : F \rightarrow X$ $f \circ_c h = g \circ_c h$
  shows $\exists!\ k.\ k : F \rightarrow E \wedge m \circ_c k = h$
  ⟨*proof*⟩

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

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

lemma *equalizer-exists2*:
  assumes $f : X \rightarrow Y$ $g : X \rightarrow Y$
  shows $\exists\ E\ m.\ m : E \rightarrow X \wedge f \circ_c m = g \circ_c m \wedge (\forall\ h\ F.\ ((h : F \rightarrow X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k.\ (k : F \rightarrow E) \wedge m \circ_c k = h))$
⟨*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 $\exists\ k.\ k : E \rightarrow E' \wedge$ *isomorphism* $k \wedge m = m' \circ_c k$
⟨*proof*⟩

lemma *isomorphic-to-equalizer-is-equalizer*:
  assumes $\varphi: E' \rightarrow E$
  assumes *isomorphism* φ
  assumes *equalizer* E m f g
  assumes $f : X \rightarrow Y$
  assumes $g : X \rightarrow Y$
  assumes $m : E \rightarrow X$
  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* $\Rightarrow$ *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)
$\langle proof \rangle$

4.2 Subobjects

The definition below corresponds to Definition 2.1.32 in Halvorson.

definition *factors-through* :: *cfunc* $\Rightarrow$ *cfunc* $\Rightarrow$ *bool* (**infix** *factorsthru 90*)
where g *factorsthru* $f \longleftrightarrow (\exists\ h.\ (h\colon domain(g) \rightarrow domain(f)) \wedge f \circ_c h = g)$

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

The lemma below corresponds to Exercise 2.1.33 in Halvorson.

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

The definition below corresponds to Definition 2.1.37 in Halvorson.

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

lemma *subobject-of-def2*:
$(B,m) \subseteq_c X = (m : B \rightarrow X \wedge monomorphism\ m)$
$\langle proof \rangle$

definition *relative-subset* :: *cset* $\times$ *cfunc* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\times$ *cfunc* $\Rightarrow$ *bool* (-$\subseteq$-- [*51*,*50*,*51*]*50*)
where $B \subseteq_X A \longleftrightarrow$
$(snd\ B : fst\ B \rightarrow X \wedge monomorphism\ (snd\ B) \wedge snd\ A : fst\ A \rightarrow X \wedge monomorphism\ (snd\ A)$
$\wedge (\exists\ k.\ k\colon fst\ B \rightarrow fst\ A \wedge snd\ A \circ_c k = snd\ B))$

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

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
$x \in_X B \longleftrightarrow (x \in_c X \wedge monomorphism\ (snd\ B) \wedge snd\ B : fst\ B \rightarrow X \wedge x$ *factorsthru* $(snd\ B))$

lemma *relative-member-def2*:
$x \in_X (B,\ m) = (x \in_c X \land \mathit{monomorphism}\ m \land m : B \to X \land x\ \mathit{factorsthru}\ m)$
⟨*proof*⟩

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 $x \in_X (A,n) \Longrightarrow x \in_X (B,m)$
⟨*proof*⟩

5 Pullback

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

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*⟩

6 Inverse Image

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

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

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

definition *inverse-image-mapping* :: *cfunc* ⇒ *cset* ⇒ *cfunc* ⇒ *cfunc* **where**

inverse-image-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* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* $\Rightarrow$ *cfunc* ($[\text{-}^{-1}(\!|\text{-}|\!)_{\text{-}}]$*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*⟩

7 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. ∃ Z m. $f : X \to Z \wedge g : Y \to Z \wedge$
equalizer E m ($f \circ_c$ *left-cart-proj* X Y) ($g \circ_c$ *right-cart-proj* X Y))

lemma *fibered-product-equalizer*:
assumes $f : X \to Z$ $g : Y \to Z$
shows ∃ 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. ∃ Z. $f : X \to Z \wedge g : Y \to Z \wedge$
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 $f : X \to Z$ $g : Y \to Z$
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 $f : X \rightarrow Y$ $g : X \rightarrow E$ $x \in_c X$ $y \in_c X$
assumes $g \circ_c$ *fibered-product-left-proj* $X f f X = g \circ_c$ *fibered-product-right-proj* X $f f X$
shows $\forall x\, y.\ x \in_c X \longrightarrow y \in_c X \longrightarrow \langle x,y \rangle \in_{X \times_c X} (X\ {}_f{\times_c}_f\ X,$ *fibered-product-morphism* $X f f X) \longrightarrow g \circ_c x = g \circ_c y$
⟨*proof*⟩

lemma *kernel-pair-subset*:
assumes $f\colon X \rightarrow Y$
shows $(X\ {}_f{\times_c}_f\ X,$ *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 $f\colon X \rightarrow Y$ *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 $f\colon X \rightarrow Y$ *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 $f\colon X \rightarrow Y$
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*: $f : Y \rightarrow T$
assumes *g-type*: $g : X \rightarrow T$
shows $(X\ {}_g{\times_c}_f\ Y) \cong X \times_c Y$
⟨*proof*⟩

end
theory *Truth*
imports *Equalizer*
begin

8 Truth Values and Characteristic Functions

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: $x \in_c \Omega \Longrightarrow x = \mathrm{f} \lor x = \mathrm{t}$ **and**
characteristic-function-exists:
$m : B \to X \Longrightarrow$ *monomorphism* $m \Longrightarrow \exists!\ \chi.$ *is-pullback* B *one* X Ω (β_B) t m χ

definition *characteristic-func* :: *cfunc* ⇒ *cfunc* **where**
characteristic-func m =
(*THE* χ. *monomorphism* $m \longrightarrow$ *is-pullback* (*domain* m) *one* (*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 *one* 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 $m : B \rightarrow X$ *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*⟩

9 Equality Predicate

definition *eq-pred* :: *cset* ⇒ *cfunc* **where**
  eq-pred X = (*THE* χ. *is-pullback* X *one* $(X \times_c X)$ Ω (β_X) t (*diagonal* X) χ)

lemma *eq-pred-pullback*: *is-pullback* X *one* $(X \times_c X)$ Ω (β_X) t (*diagonal* X) (*eq-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 $x : one \rightarrow X$ $y : one \rightarrow X$
  shows $(x = y) = ($*eq-pred* $X \circ_c \langle x,\ y\rangle = \mathrm{t})$
⟨*proof*⟩

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

lemma *eq-pred-iff-eq-conv2*:

assumes $x : one \to X$ $y : one \to X$
shows $(x \neq y) = (\textit{eq-pred } X \circ_c \langle x,\ y\rangle \neq \mathrm{t})$
⟨*proof*⟩

lemma *eq-pred-of-monomorphism*:
assumes *m-type*[*type-rule*]: $m : X \to Y$ **and** *m-mono*: *monomorphism* m
shows $\textit{eq-pred } Y \circ_c (m \times_f m) = \textit{eq-pred } X$
⟨*proof*⟩

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

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

10 Properties of Monomorphisms and Epimorphisms

The lemma below corresponds to Exercise 2.2.3 in Halvorson.

lemma *regmono-is-mono*: $\textit{regular-monomorphism}(m) \Longrightarrow \textit{monomorphism}(m)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.4 in Halvorson.

lemma *mono-is-regmono*:
shows $\textit{monomorphism}(m) \Longrightarrow \textit{regular-monomorphism}(m)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.5 in Halvorson.

lemma *epi-mon-is-iso*:
assumes $\textit{epimorphism}(f)$ $\textit{monomorphism}(f)$
shows $\textit{isomorphism}(f)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.8 in Halvorson.

lemma *epi-is-surj*:
assumes p: $X \to Y$ $\textit{epimorphism}(p)$
shows $\textit{surjective}(p)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.9 in Halvorson.

lemma *pullback-of-epi-is-epi1*:
assumes f: $Y \to Z$ *epimorphism f is-pullback A Y X Z q1 f q0 g*
shows *epimorphism q0*
⟨*proof*⟩

The lemma below corresponds to Proposition 2.2.9b in Halvorson.

lemma *pullback-of-epi-is-epi2*:
assumes g: $X \to Z$ *epimorphism* g *is-pullback* A Y X Z *q1* f *q0* g
shows *epimorphism q1*
⟨*proof*⟩

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*⟩

11 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**
$f^{-1}\{y\} = (f^{-1}(\!|one|\!)_y)$

definition *fiber-morphism* :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* **where**
fiber-morphism f y = *left-cart-proj* (*domain f*) *one* $\circ_c$ *inverse-image-mapping f one 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*: $\neg$ *surjective* p
  shows $\exists\ y.\ y \in_c Y \wedge$ *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 : *one* $\to Y$
  shows $f^{-1}\{y\} \cong X\ {}_{f}{\times_c}{}_{y} one$
  ⟨*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*⟩

12 Set Subtraction

definition *set-subtraction* :: *cset* $\Rightarrow$ *cset* $\times$ *cfunc* $\Rightarrow$ *cset* (**infix** $\setminus$ *60*) **where**
  $Y \setminus X = ($*SOME* $E.\ \exists\ m'.$ *equalizer* $E\ m'$ (*characteristic-func* (*snd* X)) (f $\circ_c$ β_Y))

lemma *set-subtraction-equalizer*:
  assumes $m : X \to Y$ *monomorphism* m
  shows $\exists\ m'.$ *equalizer* $(Y \setminus (X,m))\ m'$ (*characteristic-func* m) (f $\circ_c$ β_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 $m : X \rightarrow Y$ *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 $m : X \rightarrow Y$ *monomorphism* m
shows $m^c : Y \setminus (X,m) \rightarrow Y$
⟨*proof*⟩

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

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

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

end
theory *Equivalence*
imports *Truth*
begin

13 Equivalence Classes

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

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

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

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

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

R

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

abbreviation *equiv-class'* :: *cfunc* $\Rightarrow$ *cset* $\times$ *cfunc* $\Rightarrow$ *cfunc* ([-]$_{-}$) **where**
$[x]_R \equiv$ *equiv-class* $R \circ_c x$

14 Coequalizers and Epimorphisms

14.1 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* (*quotient-set* X (R,m)) (*equiv-class* (R,m))
(*left-cart-proj* $X\ X \circ_c m$) (*right-cart-proj* $X\ X \circ_c m$)
⟨*proof*⟩

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

14.2 Regular Epimorphisms

The definition below corresponds to Definition 2.3.4 in Halvorson.

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

The lemma below corresponds to Exercise 2.3.5 in Halvorson.

lemma *reg-epi-and-mono-is-iso*:
  assumes $f : X \to Y$ *regular-epimorphism f monomorphism f*
  shows *isomorphism f*
⟨*proof*⟩

The two lemmas below correspond to Proposition 2.3.6 in Halvorson.

lemma *epimorphism-coequalizer-kernel-pair*:
  assumes $f : X \to Y$ *epimorphism f*
  shows *coequalizer Y f* (*fibered-product-left-proj X f f X*) (*fibered-product-right-proj X f f X*)
⟨*proof*⟩

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

14.3 Epi-monic Factorization

lemma *epi-monic-factorization*:
  assumes *f-type*[*type-rule*]: $f : X \to Y$
  shows ∃ *g m E*. $g : X \to E \wedge m : E \to Y$
    ∧ *coequalizer E g* (*fibered-product-left-proj X f f X*) (*fibered-product-right-proj X f f X*)
    ∧ *monomorphism m* ∧ $f = m \circ_c g$
    ∧ $(\forall x.\ x : E \to Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
⟨*proof*⟩

lemma *epi-monic-factorization2*:
  assumes *f-type*[*type-rule*]: $f : X \to Y$
  shows ∃ *g m E*. $g : X \to E \wedge m : E \to Y$
    ∧ *epimorphism g* ∧ *monomorphism m* ∧ $f = m \circ_c g$
    ∧ $(\forall x.\ x : E \to Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
  ⟨*proof*⟩

15 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*.
  $g : A \to fA$ ∧

$m : fA \rightarrow codomain\ f \wedge$
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) $\wedge$
monomorphism $m \wedge f \circ_c n = m \circ_c g \wedge (\forall x.\ x : fA \rightarrow codomain\ f \longrightarrow f \circ_c n = x \circ_c g \longrightarrow x = m))$

lemma *image-of-def2*:
assumes $f : X \rightarrow Y$ $n : A \rightarrow X$
shows $\exists\, g\ m.$
$g : A \rightarrow f(\!|A|\!)_n \wedge$
$m : f(\!|A|\!)_n \rightarrow Y \wedge$
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) $\wedge$
monomorphism $m \wedge f \circ_c n = m \circ_c g \wedge (\forall x.\ x : f(\!|A|\!)_n \rightarrow Y \longrightarrow f \circ_c n = x \circ_c g \longrightarrow x = m)$
⟨*proof*⟩

definition *image-restriction-mapping* :: *cfunc* ⇒ *cset* × *cfunc* ⇒ *cfunc* (-↾- [*101*,*0*]*100*)
where
image-restriction-mapping f An = (*SOME* $g.\ \exists\ m.\ g : fst\ An \rightarrow f(\!|fst\ An|\!)_{snd\ An} \wedge m : f(\!|fst\ An|\!)_{snd\ An} \rightarrow codomain\ f \wedge$
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* $(fst\ An)$ $(f \circ_c snd\ An)$ $(f \circ_c snd\ An)$ $(fst\ An)$) $\wedge$
monomorphism $m \wedge f \circ_c snd\ An = m \circ_c g \wedge (\forall x.\ x : f(\!|fst\ An|\!)_{snd\ An} \rightarrow codomain\ f \longrightarrow f \circ_c snd\ An = x \circ_c g \longrightarrow x = m))$

lemma *image-restriction-mapping-def2*:
assumes $f : X \rightarrow Y$ $n : A \rightarrow X$
shows $\exists\ m.\ f{\restriction}_{(A,\ n)} : A \rightarrow f(\!|A|\!)_n \wedge m : f(\!|A|\!)_n \rightarrow Y \wedge$
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 $m \wedge f \circ_c n = m \circ_c (f{\restriction}_{(A,\ n)}) \wedge (\forall x.\ x : f(\!|A|\!)_n \rightarrow Y \longrightarrow f \circ_c n = x \circ_c (f{\restriction}_{(A,\ n)}) \longrightarrow x = m)$
⟨*proof*⟩

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

lemma *image-subobject-mapping-def2*:
assumes $f : X \rightarrow Y$ $n : A \rightarrow X$
shows $f{\restriction}_{(A,\ n)} : A \rightarrow f(\!|A|\!)_n \wedge [f(\!|A|\!)_n]map : f(\!|A|\!)_n \rightarrow Y \wedge$
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 $x : f(\!|A|\!)_n \to Y \Longrightarrow f \circ_c n = x \circ_c (f{\restriction}_{(A,\ n)}) \Longrightarrow x = [f(\!|A|\!)_n]map$
⟨*proof*⟩

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

The lemma below corresponds to Proposition 2.3.8 in Halvorson.

lemma *image-smallest-subobject*:
assumes *f-type*[*type-rule*]: $f : X \to Y$ **and** *a-type*[*type-rule*]: $a : A \to X$
shows $(B, n) \subseteq_c Y \Longrightarrow f\ factorsthru\ n \Longrightarrow (f(\!|A|\!)_a, [f(\!|A|\!)_a]map) \subseteq_Y (B, n)$
⟨*proof*⟩

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 $(f \circ_c m)(\!|A|\!)_n \cong f(\!|A|\!)_{m \circ_c n}$
⟨*proof*⟩

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 $\exists\, i.\ ((f \circ_c m)(\!|A|\!)_n, i) \subseteq_c B \Longrightarrow \exists\, j.\ (f(\!|A|\!)_{m \circ_c n}, j) \subseteq_c B$
⟨*proof*⟩

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 $(f(\!|A|\!)_{m \circ_c n}, [f(\!|A|\!)_{m \circ_c n}]map) \subseteq_Y (B,b)$
⟨*proof*⟩

The lemma below corresponds to Proposition 2.3.9 in Halvorson.

lemma *subset-inv-image-iff-image-subset*:
assumes $(A,a) \subseteq_c X$ $(B,m) \subseteq_c Y$
assumes[*type-rule*]: $f : X \to Y$
shows $((A,\ a) \subseteq_X (f^{-1}(\!|B|\!)_m,[f^{-1}(\!|B|\!)_m]map)) = ((f(\!|A|\!)_a,\ [f(\!|A|\!)_a]map) \subseteq_Y (B,m))$
⟨*proof*⟩

The lemma below corresponds to Exercise 2.3.10 in Halvorson.

lemma *in-inv-image-of-image*:
assumes $(A,m) \subseteq_c X$
assumes[*type-rule*]: $f : X \to Y$
shows $(A,m) \subseteq_X (f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}, [f^{-1}(\!|f(\!|A|\!)_m|\!)_{[f(\!|A|\!)_m]map}]map)$
⟨*proof*⟩

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

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

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* $Z\ X\ X \circ_c (id\ Z \times_f m))$
⟨*proof*⟩

17 Graphs

definition *functional-on* :: *cset* ⇒ *cset* ⇒ *cset* × *cfunc* ⇒ *bool* **where**
functional-on X Y R = (R $\subseteq_c$ X $\times_c$ Y ∧
(∀ x. x $\in_c$ X ⟶ (∃! y. y $\in_c$ Y ∧
⟨x,y⟩ $\in_{X \times_c Y}$ 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* : $X \to Y$
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* : $X \to Y$
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-morph f*) $\subseteq_c$ (X $\times_c$ Y)
⟨*proof*⟩

lemma *graph-morph-type*[*type-rule*]:
assumes *f* : $X \to Y$
shows *graph-morph*(*f*) : *graph f* → 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-on* $X\ Y$ (*graph* f, *graph-morph* f)
⟨*proof*⟩

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

end
theory *Coproduct*
imports *Equivalence*
begin

18 Axiom 7: Coproducts

hide-const *case-bool*

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

axiomatization
coprod :: *cset* ⇒ *cset* ⇒ *cset* (**infixr** ∐ *65*) **and**
left-coproj :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
right-coproj :: *cset* ⇒ *cset* ⇒ *cfunc* **and**
cfunc-coprod :: *cfunc* ⇒ *cfunc* ⇒ *cfunc* (**infixr** ⨿ *65*)
where
left-proj-type[*type-rule*]: *left-coproj* $X\ Y : X \to X \coprod Y$ **and**
right-proj-type[*type-rule*]: *right-coproj* $X\ Y : Y \to X \coprod Y$ **and**
cfunc-coprod-type[*type-rule*]: $f : X \to Z \Longrightarrow g : Y \to Z \Longrightarrow f \amalg g : X \coprod Y \to Z$
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* ⇒ *cfunc* ⇒ *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool* **where**
is-coprod $W\ i_0\ i_1\ X\ Y \longleftrightarrow$
$(i_0 : X \to W \land i_1 : Y \to W \land$
$(\forall\ f\ g\ Z.\ (f : X \to Z \land g : Y \to Z) \longrightarrow$
$(\exists\ h.\ h : W \to Z \land h \circ_c i_0 = f \land h \circ_c i_1 = g \land$
$(\forall\ h2.\ (h2 : W \to Z \land h2 \circ_c i_0 = f \land h2 \circ_c i_1 = g) \longrightarrow h2 = h))))$

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

18.1 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*) ⨿ (*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 $z \in_c X \coprod Y$
shows ($\exists$ *x*. ($x \in_c X \wedge z =$ (*left-coproj X Y*) $\circ_c x$))
∨ ($\exists$ *y*. ($y \in_c Y \wedge z =$ (*right-coproj X Y*) $\circ_c y$))
⟨*proof*⟩

lemma *maps-into-1u1*:

assumes *x-type*: $x \in_c (one \coprod one)$
shows $(x = \textit{left-coproj one one}) \vee (x = \textit{right-coproj one one})$
⟨*proof*⟩

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

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

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

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

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

The lemma below corresponds to Proposition 2.4.4 in Halvorson.

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

18.1.1 Equality Predicate with Coproduct Properities

lemma *eq-pred-left-coproj*:
assumes *u-type*[*type-rule*]: $u \in_c X \coprod Y$ **and** *x-type*[*type-rule*]: $x \in_c X$

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

18.2 Bowtie Product

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

lemma *cfunc-bowtie-prod-def2*:
assumes $f : X \to Y\ g : V \to W$
shows $f \bowtie_f g = (\textit{left-coproj}\ Y\ W \circ_c f) \amalg (\textit{right-coproj}\ Y\ W \circ_c g)$
⟨*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*:
$f : X \to Y \Longrightarrow g : V \to W \Longrightarrow (f \bowtie_f g) \circ_c \textit{left-coproj}\ X\ V = \textit{left-coproj}\ Y\ W \circ_c f$
⟨*proof*⟩

lemma *right-coproj-cfunc-bowtie-prod*:
$f : X \to Y \Longrightarrow g : V \to W \Longrightarrow (f \bowtie_f g) \circ_c \textit{right-coproj}\ X\ V = \textit{right-coproj}\ Y\ W \circ_c g$
⟨*proof*⟩

lemma *cfunc-bowtie-prod-unique*: $f : X \to Y \Longrightarrow g : V \to W \Longrightarrow h : X \coprod V \to Y \coprod W \Longrightarrow$
$h \circ_c \textit{left-coproj}\ X\ V = \textit{left-coproj}\ Y\ W \circ_c f \Longrightarrow$
$h \circ_c \textit{right-coproj}\ X\ V = \textit{right-coproj}\ Y\ W \circ_c g \Longrightarrow h = f \bowtie_f g$
⟨*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 *type-assms*: $f : X \to Y$ $g : V \to W$
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*
$\langle proof \rangle$

18.3 Case Bool

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

case-bool = (*THE f. f* : Ω → (*one* $\coprod$ *one*) ∧
(t II f) $\circ_c$ *f* = *id* Ω ∧ *f* $\circ_c$ (t II f) = *id* (*one* $\coprod$ *one*))

lemma *case-bool-def2*:
case-bool : Ω → (*one* $\coprod$ *one*) ∧
(t II f) $\circ_c$ *case-bool* = *id* Ω ∧ *case-bool* $\circ_c$ (t II f) = *id* (*one* $\coprod$ *one*)
⟨*proof*⟩

lemma *case-bool-type*[*type-rule*]:
case-bool : Ω → *one* $\coprod$ *one*
⟨*proof*⟩

lemma *case-bool-true-coprod-false*:
case-bool $\circ_c$ (t II f) = *id* (*one* $\coprod$ *one*)
⟨*proof*⟩

lemma *true-coprod-false-case-bool*:
(t II f) $\circ_c$ *case-bool* = *id* Ω
⟨*proof*⟩

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

lemma *case-bool-true-and-false*:
(*case-bool* $\circ_c$ t = *left-coproj one one*) ∧ (*case-bool* $\circ_c$ f = *right-coproj one one*)
⟨*proof*⟩

lemma *case-bool-true*:
case-bool $\circ_c$ t = *left-coproj one one*
⟨*proof*⟩

lemma *case-bool-false*:
case-bool $\circ_c$ f = *right-coproj one one*
⟨*proof*⟩

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

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

18.4 Distribution of Products over Coproducts

18.4.1 Distribute Product Over Coproduct Auxillary Mapping

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

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

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

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

lemma *dist-prod-coprod-mono*:
monomorphism (*dist-prod-coprod A B C*)
⟨*proof*⟩

lemma *dist-prod-coprod-epi*:
epimorphism (*dist-prod-coprod A B C*)
⟨*proof*⟩

lemma *dist-prod-coprod-iso*:
isomorphism(*dist-prod-coprod 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*⟩

18.4.2 Inverse Distribute Product Over Coproduct Auxillary Mapping

definition *dist-prod-coprod-inv* :: *cset* ⇒ *cset* ⇒ *cset* ⇒ *cfunc* **where**
dist-prod-coprod-inv A B C = (*THE f*. *f* : $A \times_c$ ($B \coprod C$) → ($A \times_c B$) $\coprod$ ($A \times_c C$)
∧ *f* $\circ_c$ *dist-prod-coprod A B C* = *id* (($A \times_c B$) $\coprod$ ($A \times_c C$))
∧ *dist-prod-coprod A B C* $\circ_c$ *f* = *id* ($A \times_c$ ($B \coprod C$)))

lemma *dist-prod-coprod-inv-def2*:
shows $dist\text{-}prod\text{-}coprod\text{-}inv\ A\ B\ C : A \times_c (B \coprod C) \to (A \times_c B) \coprod (A \times_c C)$
$\wedge\ dist\text{-}prod\text{-}coprod\text{-}inv\ A\ B\ C \circ_c dist\text{-}prod\text{-}coprod\ A\ B\ C = id\ ((A \times_c B) \coprod (A \times_c C))$
$\wedge\ dist\text{-}prod\text{-}coprod\ A\ B\ C \circ_c dist\text{-}prod\text{-}coprod\text{-}inv\ A\ B\ C = id\ (A \times_c (B \coprod C))$
⟨*proof*⟩

lemma *dist-prod-coprod-inv-type*[*type-rule*]:
$dist\text{-}prod\text{-}coprod\text{-}inv\ A\ B\ C : A \times_c (B \coprod C) \to (A \times_c B) \coprod (A \times_c C)$
⟨*proof*⟩

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

lemma *dist-prod-coprod-inv-right*:
$dist\text{-}prod\text{-}coprod\ A\ B\ C \circ_c dist\text{-}prod\text{-}coprod\text{-}inv\ A\ B\ C = id\ (A \times_c (B \coprod C))$
⟨*proof*⟩

lemma *dist-prod-coprod-inv-iso*:
$isomorphism(dist\text{-}prod\text{-}coprod\text{-}inv\ A\ B\ C)$
⟨*proof*⟩

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

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

18.4.3 Distribute Product Over Coproduct Auxillary Mapping 2

definition $dist\text{-}prod\text{-}coprod2 :: cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**
$dist\text{-}prod\text{-}coprod2\ A\ B\ C = swap\ C\ (A \coprod B) \circ_c dist\text{-}prod\text{-}coprod\ C\ A\ B \circ_c (swap\ A\ C \bowtie_f swap\ B\ C)$

lemma *dist-prod-coprod2-type*[*type-rule*]:
$dist\text{-}prod\text{-}coprod2\ A\ B\ C : (A \times_c C) \coprod (B \times_c C) \to (A \coprod B) \times_c C$
⟨*proof*⟩

lemma *dist-prod-coprod2-left-ap*:
assumes $a \in_c A\ c \in_c C$
shows $dist\text{-}prod\text{-}coprod2\ A\ B\ C \circ_c (left\text{-}coproj\ (A \times_c C)\ (B \times_c C) \circ_c \langle a,\ c\rangle) =$

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

lemma *dist-prod-coprod2-right-ap*:
assumes $b \in_c B$ $c \in_c C$
shows *dist-prod-coprod2* 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*$\rangle$

18.4.4 Inverse Distribute Product Over Coproduct Auxillary Mapping 2

definition *dist-prod-coprod-inv2* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
dist-prod-coprod-inv2 A B C = (*swap* C A $\bowtie_f$ *swap* C B) $\circ_c$ *dist-prod-coprod-inv* C A B $\circ_c$ *swap* $(A \coprod B)$ C

lemma *dist-prod-coprod-inv2-type*[*type-rule*]:
dist-prod-coprod-inv2 A B C : $(A \coprod B) \times_c C \rightarrow (A \times_c C) \coprod (B \times_c C)$
$\langle$*proof*$\rangle$

lemma *dist-prod-coprod-inv2-left-ap*:
assumes $a \in_c A$ $c \in_c C$
shows *dist-prod-coprod-inv2* 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-inv2-right-ap*:
assumes $b \in_c B$ $c \in_c C$
shows *dist-prod-coprod-inv2* 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-inv2-left-coproj*:
dist-prod-coprod-inv2 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-inv2-right-coproj*:
dist-prod-coprod-inv2 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 *dist-prod-coprod2-inv2-id*:
dist-prod-coprod2 A B C $\circ_c$ *dist-prod-coprod-inv2* A B C = *id* $((A \coprod B) \times_c C)$
$\langle$*proof*$\rangle$

lemma *dist-prod-coprod-inv2-inv-id*:
dist-prod-coprod-inv2 A B C $\circ_c$ *dist-prod-coprod2* A B C = *id* $((A \times_c C) \coprod (B \times_c C))$

$\langle proof \rangle$

lemma *dist-prod-coprod2-iso*:
  $isomorphism(dist\text{-}prod\text{-}coprod2\ A\ B\ C)$
  $\langle proof \rangle$

18.5 Casting between sets

18.5.1 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**
  $into\text{-}super\ m = m \amalg m^c$

lemma *into-super-type*[*type-rule*]:
  $monomorphism\ m \Longrightarrow m : X \to Y \Longrightarrow into\text{-}super\ m : X \coprod (Y \setminus (X,m)) \to Y$
  $\langle proof \rangle$

lemma *into-super-mono*:
  assumes $monomorphism\ m\ m : X \to Y$
  shows $monomorphism\ (into\text{-}super\ m)$
$\langle proof \rangle$

lemma *into-super-epi*:
  assumes $monomorphism\ m\ m : X \to Y$
  shows $epimorphism\ (into\text{-}super\ m)$
$\langle proof \rangle$

lemma *into-super-iso*:
  assumes $monomorphism\ m\ m : X \to Y$
  shows $isomorphism\ (into\text{-}super\ m)$
  $\langle proof \rangle$

18.5.2 Going from a set to a subset or its complement

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

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

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

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

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

lemma *try-cast-in-X*:
  assumes *m-type*: *monomorphism m* $m : X \to Y$
  assumes *y-in-X*: $y \in_Y (X, m)$
  shows $\exists$ x. $x \in_c X \wedge$ *try-cast m* $\circ_c y$ = *left-coproj* $X\ (Y \setminus (X,m)) \circ_c x$
⟨*proof*⟩

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
⟨*proof*⟩

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

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

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

18.6 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 $A \coprod B \cong C \coprod D$
⟨*proof*⟩

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

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

lemma *oneUone-iso-*Ω:
$one \coprod one \cong \Omega$
⟨*proof*⟩

The lemma below is dual to Proposition 2.2.2 in Halvorson.

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

end
theory *Axiom-Of-Choice*
imports *Coproduct*
begin

19 Axiom of Choice

The two definitions below correspond to Definition 2.7.1 in Halvorson.

definition *section-of* :: $cfunc \Rightarrow cfunc \Rightarrow bool$ (**infix** *sectionof 90*)
where $s\ sectionof\ f \longleftrightarrow s : codomain\ f \to domain\ f \land f \circ_c s = id\ (codomain\ f)$

definition *split-epimorphism* :: $cfunc \Rightarrow bool$

where *split-epimorphism f* $\longleftrightarrow$ ($\exists$ *s*. *s* : *codomain f* $\rightarrow$ *domain f* $\wedge$ *f* $\circ_c$ *s* = *id* (*codomain f*))

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

lemma *sections-define-splits*:
assumes *s sectionof f*
assumes *s* : *Y* $\rightarrow$ *X*
shows *f* : *X* $\rightarrow$ *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 sectionof f*)

lemma *epis-give-monos*:
assumes *f-type*: *f* : *X* $\rightarrow$ *Y*
assumes *f-epi*: *epimorphism f*
shows $\exists$*g*. *g*: *Y* $\rightarrow$ *X* $\wedge$ *monomorphism g* $\wedge$ *f* $\circ_c$ *g* = *id Y*
$\langle$*proof*$\rangle$

corollary *epis-are-split*:
assumes *f-type*: *f* : *X* $\rightarrow$ *Y*
assumes *f-epi*: *epimorphism f*
shows *split-epimorphism f*
$\langle$*proof*$\rangle$

The lemma below corresponds to Proposition 2.6.8 in Halvorson.

lemma *monos-give-epis*:
assumes *f-type*: *f* : *X* $\rightarrow$ *Y*
assumes *f-mono*: *monomorphism f*
assumes *X-nonempty*: *nonempty X*
shows $\exists$*g*. *g*: *Y* $\rightarrow$ *X* $\wedge$ *epimorphism g* $\wedge$ *g* $\circ_c$ *f* = *id X*
$\langle$*proof*$\rangle$

The lemma below corresponds to Exercise 2.7.2(i) in Halvorson.

lemma *split-epis-are-regular*:
assumes *f-type*[*type-rule*]: *f* : *X* $\rightarrow$ *Y*
assumes *split-epimorphism f*
shows *regular-epimorphism f*
$\langle$*proof*$\rangle$

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
theory *Initial*
  imports *Coproduct*
begin

20 Empty Set and Initial Objects

The axiomatization below corresponds to Axiom 8 (Empty Set) in Halvorson.

axiomatization
  initial-func :: *cset* ⇒ *cfunc* ($\alpha_{\_}$ *100*) **and**
  emptyset :: *cset* (∅)
where
  initial-func-type[*type-rule*]: *initial-func* $X : \emptyset \to X$ **and**
  initial-func-unique: $h : \emptyset \to X \Longrightarrow h =$ *initial-func* X **and**
  emptyset-is-empty: $\neg(x \in_c \emptyset)$

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

lemma *emptyset-is-initial*:
  initial-object(∅)
  ⟨*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: $X \to \emptyset$
  shows *isomorphism*(f)
  ⟨*proof*⟩

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

$\langle proof \rangle$

lemma *X-prod-empty*:
$X \times_c \emptyset \cong \emptyset$
$\langle proof \rangle$

The lemma below corresponds to Proposition 2.4.8 in Halvorson.

lemma *no-el-iff-iso-empty*:
is-empty $X \longleftrightarrow X \cong \emptyset$
$\langle proof \rangle$

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

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

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

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

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

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

The lemma below corresponds to Proposition 2.2.1 in Halvorson.

lemma *one-has-two-subsets*:
card $(\{(X,m).\ (X,m) \subseteq_c one\}//\{((X1,m1),(X2,m2)).\ X1 \cong X2\}) = 2$
$\langle proof \rangle$

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 $X \times_c Y \cong X$
⟨*proof*⟩

end
theory *Exponential-Objects*
imports *Initial*
begin

21 Exponential Objects, Transposes and Evaluation

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))
⟨*proof*⟩

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

lemma *exponential-object-identity*:
$(eval\text{-}func\ X\ A)^\sharp = id_c(X^A)$
⟨*proof*⟩

lemma *eval-func-X-empty-injective*:
assumes *is-empty* Y
shows *injective* $(eval\text{-}func\ X\ Y)$
⟨*proof*⟩

21.1 Lifting Functions

The definition below corresponds to Definition 2.5.1 in Halvorson.

definition *exp-func* :: *cfunc* ⇒ *cset* ⇒ *cfunc* $((\text{-})^{\text{-}}{}_f\ [100,100]100)$ **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: $A \times_c X \to Y \wedge g$: $Y \to Z \Longrightarrow (g \circ_c f)^\sharp = g^A{}_f \circ_c f^\sharp$
⟨*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*⟩

21.2 Inverse Transpose Function (flat)

The definition below corresponds to Definition 2.5.3 in Halvorson.

definition *inv-transpose-func* :: *cfunc* ⇒ *cfunc* (-$^\flat$ [*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*:
$f : A \times_c Z \to X \implies (f^\sharp)^\flat = f$

⟨*proof*⟩

The lemma below corresponds to Proposition 2.5.6 in Halvorson.

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

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

lemma *sharp-comp*:
assumes $f : A \times_c Z \to X$ $g : W \to Z$
shows $f^\sharp \circ_c g = (f \circ_c (\textit{id}\ A \times_f g))^\sharp$
⟨*proof*⟩

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

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

lemma *eval-func-X-one-injective*:
injective (*eval-func X one*)
⟨*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 $\textit{monomorphism}(f)$
assumes *nonempty A*
shows $\textit{monomorphism}(f^\sharp)$
⟨*proof*⟩

22 Metafunctions and their Inverses (Cnufatems)

22.1 Metafunctions

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

lemma *metafunc-def2*:
assumes $f : X \rightarrow Y$
shows *metafunc* $f = (f \circ_c (\textit{left-cart-proj } X \textit{ one}))^\sharp$
$\langle proof \rangle$

lemma *metafunc-type*[*type-rule*]:
assumes $f : X \rightarrow Y$
shows *metafunc* $f \in_c Y^X$
$\langle proof \rangle$

lemma *eval-lemma*:
assumes $f : X \rightarrow Y$
assumes $x \in_c X$
shows *eval-func* $Y\ X \circ_c \langle x, \textit{metafunc } f\rangle = f \circ_c x$
$\langle proof \rangle$

22.2 Inverse Metafunctions (Cnufatems)

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

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

lemma *cnufatem-type*[*type-rule*]:
assumes $f \in_c Y^X$
shows *cnufatem* $f : X \rightarrow Y$
$\langle proof \rangle$

lemma *cnufatem-metafunc*:
assumes $f : X \rightarrow Y$
shows *cnufatem* (*metafunc* f) $= f$
$\langle proof \rangle$

lemma *metafunc-cnufatem*:
assumes $f \in_c Y^X$
shows *metafunc* (*cnufatem* f) $= f$
$\langle proof \rangle$

22.3 Metafunction Composition

definition *meta-comp* :: *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cset* $\Rightarrow$ *cfunc* **where**
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*]:
  meta-comp $X\ Y\ Z : Z^Y \times_c Y^X \to Z^X$
  ⟨*proof*⟩

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

lemma *meta-comp2-def2*:
  assumes f: $W \to Z^Y$
  assumes g: $W \to Y^X$
  shows $f \square g\ = (f^\flat\ \circ_c \langle g^\flat,$ *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 $f \square g : W \to Z^X$
⟨*proof*⟩

lemma *meta-comp2-elements-aux*:
  assumes $f \in_c Z^Y$
  assumes $g \in_c Y^X$
  assumes $x \in_c X$
  shows $(f^\flat \circ_c \langle g^\flat,$*right-cart-proj* X *one*$\rangle)\ \circ_c \langle x,\ id_c$ *one*$\rangle$ = *eval-func* $Z\ Y \circ_c$ ⟨*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 $f \square g$ = *metafunc* ((*cnufatem* f) $\circ_c$ (*cnufatem* g))
  ⟨*proof*⟩

lemma *meta-comp2-def4*:
  assumes $f \in_c Z^Y$
  assumes $g \in_c Y^X$
  shows $f \square g\ $ = *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 $(f \square g) \circ_c w = (f \circ_c w) \square (g \circ_c w)$
⟨*proof*⟩

lemma *meta-comp2-def5*:

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

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

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

lemma *comp-as-metacomp*:
assumes $g : X \to Y$
assumes $f : Y \to Z$
shows $f \circ_c g = \mathit{cnufatem}(\mathit{metafunc}\ f \Box \mathit{metafunc}\ g)$
⟨*proof*⟩

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

lemma *meta-comp-assoc*:
assumes $e : W \to A^Z$
assumes $f : W \to Z^Y$
assumes $g : W \to Y^X$
shows $(e \Box f) \Box \;\; g \; = e \Box (f \Box g)$
⟨*proof*⟩

23 Partially Parameterized Functions on Pairs

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

lemma *left-param-def2*:
assumes $k : P \times_c Q \to R$
shows $k_{[p,-]} \equiv k \circ_c \langle p \circ_c \beta_Q,\ \mathit{id}\ Q\rangle$
⟨*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* ⇒ *cfunc* ⇒ *cfunc* (-$_{[-,\text{-}]}$ [*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 $k : P \times_c Q \to R$
assumes $q \in_c Q$
shows $k_{[-,q]} : P \to R$
⟨*proof*⟩

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

24 Exponential Set Facts

The lemma below corresponds to Proposition 2.5.7 in Halvorson.

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

The lemma below corresponds to Proposition 2.5.8 in Halvorson.

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

lemma *one-exp*:
$one^{X} \cong one$
⟨*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 $Y^X \cong$ *one*
⟨*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
theory *Nats*
imports *Exponential-Objects*
begin

25 Natural Number Object

The axiomatization below corresponds to Axiom 10 (Natural Number Object) in Halvorson.

axiomatization
natural-numbers :: *cset* ($\mathbb{N}_c$) **and**
zero :: *cfunc* **and**
successor :: *cfunc*
where
zero-type[*type-rule*]: *zero* $\in_c \mathbb{N}_c$ **and**
successor-type[*type-rule*]: *successor*: $\mathbb{N}_c \to \mathbb{N}_c$ **and**
natural-number-object-property:
q : *one* $\to X \Longrightarrow f$: $X \to X \Longrightarrow$
($\exists!u.\ u$: $\mathbb{N}_c \to X \wedge$
$q = u \circ_c$ *zero* $\wedge$
$f \circ_c u = u \circ_c$ *successor*)

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

lemma *natural-number-object-property2*:
assumes q : *one* $\to X\ f$: $X \to X$
shows $\exists!u.\ u$: $\mathbb{N}_c \to X \wedge u \circ_c$ *zero* $= q \wedge f \circ_c u = u \circ_c$ *successor*
⟨*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: $X \to X$
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-NNO $Y\ z\ s \longleftrightarrow (z: one \to Y \land s: Y \to Y \land (\forall\ X\ f\ q.\ ((q : one \to X) \land (f: X \to X)) \longrightarrow$
$(\exists !u.\ u: Y \to X \land$
$q = u \circ_c z \land$
$f \circ_c u = u \circ_c s)))$

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

26 Zero and Successor

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

The lemma below corresponds to Proposition 2.6.6 in Halvorson.

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

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

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

lemma *nonzero-is-succ-aux*:
assumes $x \in_c (one \coprod \mathbb{N}_c)$

shows $(x = (\textit{left-coproj one } \mathbb{N}_c) \circ_c \textit{id one}) \vee$
$(\exists\, n.\ (n \in_c \mathbb{N}_c) \wedge (x = (\textit{right-coproj one } \mathbb{N}_c) \circ_c n))$
$\langle \textit{proof} \rangle$

lemma *nonzero-is-succ*:
assumes $k \in_c \mathbb{N}_c$
assumes $k \neq \textit{zero}$
shows $\exists\, n.(n \in_c \mathbb{N}_c \wedge k = \textit{successor} \circ_c n)$
$\langle \textit{proof} \rangle$

27 Predecessor

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

lemma *predecessor-def2*:
$\textit{predecessor} : \mathbb{N}_c \to \textit{one} \coprod \mathbb{N}_c \wedge \textit{predecessor} \circ_c (\textit{zero} \amalg \textit{successor}) = \textit{id } (\textit{one} \coprod$
$\mathbb{N}_c)$
$\wedge (\textit{zero} \amalg \textit{successor}) \circ_c \textit{predecessor} = \textit{id } \mathbb{N}_c$
$\langle \textit{proof} \rangle$

lemma *predecessor-type*[*type-rule*]:
$\textit{predecessor} : \mathbb{N}_c \to \textit{one} \coprod \mathbb{N}_c$
$\langle \textit{proof} \rangle$

lemma *predecessor-left-inv*:
$(\textit{zero} \amalg \textit{successor}) \circ_c \textit{predecessor} = \textit{id } \mathbb{N}_c$
$\langle \textit{proof} \rangle$

lemma *predecessor-right-inv*:
$\textit{predecessor} \circ_c (\textit{zero} \amalg \textit{successor}) = \textit{id } (\textit{one} \coprod \mathbb{N}_c)$
$\langle \textit{proof} \rangle$

lemma *predecessor-successor*:
$\textit{predecessor} \circ_c \textit{successor} = \textit{right-coproj one } \mathbb{N}_c$
$\langle \textit{proof} \rangle$

lemma *predecessor-zero*:
$\textit{predecessor} \circ_c \textit{zero} = \textit{left-coproj one } \mathbb{N}_c$
$\langle \textit{proof} \rangle$

28 Peano's Axioms and Induction

The lemma below corresponds to Proposition 2.6.7 in Halvorson.

lemma *Peano's-Axioms*:
$\textit{injective}(\textit{successor}) \wedge \neg\textit{surjective}(\textit{successor})$

⟨*proof*⟩

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

29 Function Iteration

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

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

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

lemma *ITER-zero*:
assumes $f : Z \rightarrow (U^U)$
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 $f : Z \rightarrow (U^U)$
assumes $n : Z \rightarrow \mathbb{N}_c$
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**
$iter\text{-}comp\ g\ n\ \equiv\ cnufatem\ (ITER\ (domain\ g) \circ_c \langle metafunc\ g{,}n\rangle)$

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 $g : X \rightarrow X$
assumes $n \in_c \mathbb{N}_c$
shows $g^{\circ n}$: $X \rightarrow X$
⟨*proof*⟩

lemma *iter-comp-def3*:
assumes $g : X \rightarrow X$
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 $g : X \to X$
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*⟩

30 Relation of Nat to Other Sets

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

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

end
theory *Pred-Logic*
imports *Coproduct*
begin

31 Predicate logic functions

31.1 NOT

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

lemma *NOT-is-pullback*:
$is\text{-}pullback\ one\ one\ \Omega\ \Omega\ (\beta_{one})\ \mathrm{t}\ \mathrm{f}\ NOT$
⟨*proof*⟩

lemma *NOT-type*[*type-rule*]:
$NOT : \Omega \rightarrow \Omega$
⟨*proof*⟩

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

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

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

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

lemma *double-negation*:
$NOT \circ_c NOT = id\ \Omega$
⟨*proof*⟩

31.2 AND

definition *AND* :: *cfunc* **where**
$AND = (THE\ \chi.\ is\text{-}pullback\ one\ one\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{one})\ \mathrm{t}\ \langle \mathrm{t},\mathrm{t}\rangle\ \chi)$

lemma *AND-is-pullback*:
$is\text{-}pullback\ one\ one\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{one})\ \mathrm{t}\ \langle \mathrm{t},\mathrm{t}\rangle\ AND$
⟨*proof*⟩

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

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

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

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

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

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

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

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

31.3 NOR

definition *NOR* :: *cfunc* **where**
$NOR = (THE\ \chi.\ is\text{-}pullback\ \ one\ one\ (\Omega{\times_c}\Omega)\ \Omega\ (\beta_{one})\ \mathrm{t}\ \langle \mathrm{f},\ \mathrm{f}\rangle\ \chi)$

lemma *NOR-is-pullback*:
$is\text{-}pullback\ \ one\ one\ (\Omega{\times_c}\Omega)\ \Omega\ (\beta_{one})\ \mathrm{t}\ \langle \mathrm{f},\ \mathrm{f}\rangle\ NOR$
⟨*proof*⟩

lemma *NOR-type*[*type-rule*]:
$NOR : \Omega \times_c \Omega \to \Omega$
⟨*proof*⟩

lemma *NOR-false-false-is-true*:
$NOR \circ_c \langle f,f\rangle = t$
⟨*proof*⟩

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

lemma *NOR-right-true-is-false*:
assumes $p \in_c \Omega$
shows $NOR \circ_c \langle p,t\rangle = f$
⟨*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*: $NOR \circ_c (P \times_f Q) = t \circ_c \beta_{X \times_c Y}$
shows $(P = f \circ_c \beta_X) \wedge (Q = f \circ_c \beta_Y)$
⟨*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*: $NOR \circ_c (P \times_f Q) = t \circ_c \beta_{X \times_c Y}$
shows $\neg ((P = t \circ_c \beta_X) \vee (Q = t \circ_c \beta_Y))$
⟨*proof*⟩

31.4 OR

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

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

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

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

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

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

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 $OR \circ_c \langle OR \circ_c \langle p,q\rangle,\ r\rangle = OR \circ_c \langle p,\ OR \circ_c \langle q,r\rangle\rangle$
⟨*proof*⟩

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

31.5 XOR

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

lemma *pre-XOR-type*[*type-rule*]:
$\langle \mathrm{t},\ \mathrm{f}\rangle \amalg \langle \mathrm{f},\ \mathrm{t}\rangle : one \coprod one \to \Omega \times_c \Omega$
⟨*proof*⟩

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

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

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

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

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

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

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

$\langle proof \rangle$

31.6 NAND

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

lemma *pre-NAND-type*[*type-rule*]:
$\langle f,\ f\rangle \amalg (\langle t,\ f\rangle \amalg \langle f,\ t\rangle) : one \coprod (one \coprod one) \to \Omega \times_c \Omega$
$\langle proof \rangle$

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

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

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

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

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

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

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 $(p = f) \vee (q = f)$
$\langle proof \rangle$

lemma *NOT-AND-is-NAND*:
$NAND = NOT \circ_c AND$
$\langle proof \rangle$

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

31.7 IFF

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

lemma *pre-IFF-type*[*type-rule*]:
$\langle \mathrm{t}, \mathrm{t}\rangle \amalg \langle \mathrm{f}, \mathrm{f}\rangle : one \coprod one \rightarrow \Omega \times_c \Omega$
$\langle proof \rangle$

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

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

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

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

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

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

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

lemma *NOT-IFF-is-XOR*:
$NOT \circ_c IFF = XOR$
$\langle proof \rangle$

31.8 IMPLIES

definition *IMPLIES* :: *cfunc* **where**
IMPLIES = (*THE* χ. *is-pullback* ($one \coprod (one \coprod one)$) *one* ($\Omega \times_c \Omega$) Ω ($\beta_{(one \coprod (one \coprod one))}$) t ($\langle \mathrm{t}, \mathrm{t}\rangle \amalg (\langle \mathrm{f}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle)$) χ)

lemma *pre-IMPLIES-type*[*type-rule*]:
$\langle \mathrm{t}, \mathrm{t}\rangle \amalg (\langle \mathrm{f}, \mathrm{f}\rangle \amalg \langle \mathrm{f}, \mathrm{t}\rangle) : one \coprod (one \coprod one) \rightarrow \Omega \times_c \Omega$
⟨*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 ($one \coprod (one \coprod one)$) *one* ($\Omega \times_c \Omega$) Ω ($\beta_{(one \coprod (one \coprod one))}$) 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 : $\Omega \times_c \Omega \rightarrow \Omega$
⟨*proof*⟩

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

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

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

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

lemma *IMPLIES-false-is-true-false*:
assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes *IMPLIES* $\circ_c$ $\langle p,q\rangle$ = 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* $\circ_c$ ⟨*IMPLIES*, *IMPLIES* $\circ_c$ *swap* Ω Ω⟩
⟨*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*]: $P : X \rightarrow \Omega$ **and** *Q-type*[*type-rule*]: $Q : Y \rightarrow \Omega$
  assumes *X-nonempty*: $\exists x.\ x \in_c X$
  assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = \mathrm{t} \circ_c \beta_{X \times_c Y}$
  shows $(P = \mathrm{t} \circ_c \beta_X) \Longrightarrow (Q = \mathrm{t} \circ_c \beta_Y)$
⟨*proof*⟩

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*]: $P : X \rightarrow \Omega$ **and** *Q-type*[*type-rule*]: $Q : Y \rightarrow \Omega$
  assumes *X-nonempty*: $\exists x.\ x \in_c X$
  shows $(P = \mathrm{t} \circ_c \beta_X) \Longrightarrow ((Q = \mathrm{t} \circ_c \beta_Y) \Longrightarrow R) \Longrightarrow R$
  ⟨*proof*⟩

lemma *IMPLIES-elim''*:
  assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = \mathrm{t}$
  assumes *P-type*[*type-rule*]: $P : one \rightarrow \Omega$ **and** *Q-type*[*type-rule*]: $Q : one \rightarrow \Omega$
  shows $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

lemma *IMPLIES-elim'*:
  assumes *IMPLIES-true*: $IMPLIES \circ_c \langle P,\ Q\rangle = \mathrm{t}$
  assumes *P-type*[*type-rule*]: $P : one \rightarrow \Omega$ **and** *Q-type*[*type-rule*]: $Q : one \rightarrow \Omega$
  shows $(P = \mathrm{t}) \Longrightarrow ((Q = \mathrm{t}) \Longrightarrow R) \Longrightarrow R$
  ⟨*proof*⟩

lemma *implies-implies-IMPLIES*:
  assumes *P-type*[*type-rule*]: $P : one \rightarrow \Omega$ **and** *Q-type*[*type-rule*]: $Q : one \rightarrow \Omega$
  shows $(P = \mathrm{t} \Longrightarrow Q = \mathrm{t}) \Longrightarrow IMPLIES \circ_c \langle P,\ Q\rangle = \mathrm{t}$
  ⟨*proof*⟩

31.9 Other Boolean Identities

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

lemma *OR-AND-distributive*:
  assumes $p \in_c \Omega$
  assumes $q \in_c \Omega$
  assumes $r \in_c \Omega$

shows $OR \circ_c \langle p, AND \circ_c \langle q,r\rangle\rangle = AND \circ_c \langle OR \circ_c \langle p,q\rangle, OR \circ_c \langle p,r\rangle\rangle$
⟨*proof*⟩

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

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

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

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

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

32 Universal Quantification

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

lemma *FORALL-is-pullback*:
$is\text{-}pullback\ one\ one\ (\Omega^X)\ \Omega\ (\beta_{one})\ \mathrm{t}\ ((\mathrm{t} \circ_c \beta_{X \times_c one})^\sharp)\ (FORALL\ X)$
⟨*proof*⟩

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

lemma *all-true-implies-FORALL-true*:

assumes *p-type*: $p : X \rightarrow \Omega$ **and** *all-p-true*: $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c x = \mathrm{t}$
shows *FORALL* $X \circ_c (p \circ_c \textit{left-cart-proj } X\ \textit{one})^\sharp = \mathrm{t}$
⟨*proof*⟩

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

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

lemma *FORALL-true-implies-all-true*:
assumes *p-type*: $p : X \rightarrow \Omega$ **and** *FORALL-p-true*: *FORALL* $X \circ_c (p \circ_c \textit{left-cart-proj } X\ \textit{one})^\sharp = \mathrm{t}$
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*]: $p : X \times_c Y \rightarrow \Omega$ **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*]: $p : X \times_c \textit{one} \rightarrow \Omega$ **and** *FORALL-p-true*: *FORALL* $X \circ_c p^\sharp = \mathrm{t}$
shows $\bigwedge x.\ x \in_c X \Longrightarrow p \circ_c \langle x,\ \textit{id one}\rangle = \mathrm{t}$
⟨*proof*⟩

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

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

33 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*: $EXISTS\ X \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ one)^\sharp = \mathrm{t}$
shows $\exists\ x.\ x \in_c X \wedge p \circ_c x = \mathrm{t}$
⟨*proof*⟩

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

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 $EXISTS\ X \circ_c (p \circ_c left\text{-}cart\text{-}proj\ X\ one)^\sharp = \mathrm{t}$
⟨*proof*⟩

end
theory *Nat-Parity*
imports *Nats Quant-Logic*
begin

34 Nth Even Number

definition *nth-even* :: *cfunc* **where**
$nth\text{-}even = (THE\ u.\ u\colon \mathbb{N}_c \to \mathbb{N}_c \wedge$
$u \circ_c zero = zero \wedge$
$(successor \circ_c successor) \circ_c u = u \circ_c successor)$

lemma *nth-even-def2*:
$nth\text{-}even\colon \mathbb{N}_c \to \mathbb{N}_c \wedge nth\text{-}even \circ_c zero = zero \wedge (successor \circ_c successor) \circ_c nth\text{-}even = nth\text{-}even \circ_c successor$
⟨*proof*⟩

lemma *nth-even-type*[*type-rule*]:
$nth\text{-}even\colon \mathbb{N}_c \to \mathbb{N}_c$
⟨*proof*⟩

lemma *nth-even-zero*:
$nth\text{-}even \circ_c zero = zero$

$\langle proof \rangle$

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

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

35 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*:
$successor \circ_c nth\text{-}odd = nth\text{-}even \circ_c successor$
$\langle proof \rangle$

36 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*:
$is\text{-}even : \mathbb{N}_c \to \Omega \wedge is\text{-}even \circ_c zero = \mathrm{t} \wedge NOT \circ_c is\text{-}even = is\text{-}even \circ_c successor$
⟨*proof*⟩

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

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*⟩

37 Checking if a Number is Odd

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

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 $\neg(\textit{is-even} \circ_c m = \mathrm{t} \wedge \textit{is-odd} \circ_c m = \mathrm{t})$
  ⟨*proof*⟩

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

38 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*]:
halve-with-parity : $\mathbb{N}_c \to \mathbb{N}_c \coprod \mathbb{N}_c$
⟨*proof*⟩

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

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

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

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

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

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

lemma *even-odd-iso*:
isomorphism (*nth-even* II *nth-odd*)
⟨*proof*⟩

lemma *halve-with-parity-iso*:
isomorphism halve-with-parity
⟨*proof*⟩

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

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

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

lemma *halve-nth-odd*:

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

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

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

lemma *nth-even-or-nth-odd*:
assumes $n \in_c \mathbb{N}_c$
shows $(\exists\ m.\ m \in_c \mathbb{N}_c \wedge$ *nth-even* $\circ_c m = n) \vee (\exists\ m.\ m \in_c \mathbb{N}_c \wedge$ *nth-odd* $\circ_c m$ $= n)$
⟨*proof*⟩

lemma *is-even-exists-nth-even*:
assumes *is-even* $\circ_c n = \mathrm{t}$ **and** *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
shows $\exists m.\ m \in_c \mathbb{N}_c \wedge n =$ *nth-even* $\circ_c m$
⟨*proof*⟩

lemma *is-odd-exists-nth-odd*:
assumes *is-odd* $\circ_c n = \mathrm{t}$ **and** *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$
shows $\exists m.\ m \in_c \mathbb{N}_c \wedge n =$ *nth-odd* $\circ_c m$
⟨*proof*⟩

end
theory *Cardinality*
imports *Exponential-Objects*
begin

39 Cardinality and Finiteness

The definitions below correspond to Definition 2.6.1 in Halvorson.

definition *is-finite* :: *cset* $\Rightarrow$ *bool* **where**
is-finite$(X) \longleftrightarrow (\forall m.\ (m : X \to X \wedge$ *monomorphism*$(m)) \longrightarrow$ *isomorphism*$(m))$

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

lemma *either-finite-or-infinite*:
is-finite$(X) \vee$ *is-infinite*(X)
⟨*proof*⟩

The definition below corresponds to Definition 2.6.2 in Halvorson.

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

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

lemma *subobject-iff-smaller-than*:
$(X \leq_c Y) = (\exists\, m.\ (X,m) \subseteq_c Y)$
⟨*proof*⟩

lemma *set-card-transitive*:
assumes $A \leq_c B$
assumes $B \leq_c C$
shows $A \leq_c C$
⟨*proof*⟩

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)

$\langle proof \rangle$

lemma *size-2-sets*:
$(X \cong \Omega) = (\exists\ x1.\ (\exists\ x2.\ ((x1 \in_c X) \wedge (x2 \in_c X) \wedge (x1 \neq x2) \wedge (\forall x.\ x \in_c X \longrightarrow (x=x1) \vee (x=x2))\ )))$
$\langle proof \rangle$

lemma *size-2plus-sets*:
$(\Omega \leq_c X) = (\exists\ x1.\ (\exists\ x2.\ ((x1 \in_c X) \wedge (x2 \in_c X) \wedge (x1 \neq x2))))$
$\langle proof \rangle$

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

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

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

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

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

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

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

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

lemma *coprod-leq-product*:
assumes *X-not-init*: $\neg(\mathit{initial\text{-}object}(X))$
assumes *Y-not-init*: $\neg(\mathit{initial\text{-}object}(Y))$
assumes *X-not-term*: $\neg(\mathit{terminal\text{-}object}(X))$
assumes *Y-not-term*: $\neg(\mathit{terminal\text{-}object}(Y))$

shows $(X \coprod Y) \leq_c (X \times_c Y)$
⟨*proof*⟩

lemma *prod-leq-exp*:
assumes $\neg$(*terminal-object* Y)
shows $(X \times_c Y) \leq_c (Y^X)$
⟨*proof*⟩

lemma *Y-nonempty-then-X-le-XtoY*:
assumes *nonempty* Y
shows $X \leq_c X^Y$
⟨*proof*⟩

lemma *non-init-non-ter-sets*:
assumes $\neg$(*terminal-object* X)
assumes $\neg$(*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 *nonempty*(X)
shows $X^A \leq_c Y^B$
⟨*proof*⟩

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

lemma *emptyset-is-not-epi-countable*:
$\neg$ (*epi-countable* $\emptyset$)
$\langle proof \rangle$

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

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

lemma *epi-countable-is-countable*:
assumes *epi-countable* X
shows *countable* X
$\langle proof \rangle$

lemma *emptyset-is-countable*:
countable $\emptyset$
$\langle proof \rangle$

lemma *natural-numbers-are-countably-infinite*:
(*countable* $\mathbb{N}_c$) $\wedge$ (*is-infinite* $\mathbb{N}_c$)
$\langle proof \rangle$

lemma *iso-to-N-is-countably-infinite*:
assumes $X \cong \mathbb{N}_c$
shows (*countable* X) $\wedge$ (*is-infinite* X)
$\langle proof \rangle$

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

lemma *iso-pres-countable*:
assumes $X \cong Y$ *countable* Y
shows *countable* X
$\langle proof \rangle$

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

The lemma below corresponds to Exercise 2.6.11 in Halvorson.

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

end
theory *Fixed-Points*
  imports *Axiom-Of-Choice Pred-Logic Cardinality*
begin

The definitions below correspond to Definition 2.6.12 in Halvorson.

definition *fixed-point* :: *cfunc* ⇒ *cfunc* ⇒ *bool* **where**
  $\textit{fixed-point } a\ g \longleftrightarrow (\exists\ A.\ g : A \to A \land a \in_c A \land g \circ_c a = a)$
definition *has-fixed-point* :: *cfunc* ⇒ *bool* **where**
  $\textit{has-fixed-point } g \longleftrightarrow (\exists\ a.\ \textit{fixed-point } a\ g)$
definition *fixed-point-property* :: *cset* ⇒ *bool* **where**
  $\textit{fixed-point-property } A \longleftrightarrow (\forall\ g.\ g : A \to A \longrightarrow \textit{has-fixed-point } g)$

lemma *fixed-point-def2*:
  assumes $g : A \to A\ a \in_c A$
  shows $\textit{fixed-point } a\ g = (g \circ_c a = a)$
  ⟨*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*:
  $\nexists\ s.\ s : X \to \mathcal{P}\ X \land \textit{surjective}(s)$
⟨*proof*⟩

The theorem below corresponds to Exercise 2.6.15 in Halvorson.

theorem *Cantors-Positive-Theorem*:
  $\exists\, m.\ m : X \to \Omega^X \land \textit{injective } m$
⟨*proof*⟩

The corollary below corresponds to Corollary 2.6.16 in Halvorson.

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

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

corollary *Generalized-Cantors-Negative-Theorem*:
  assumes ¬(*initial-object X*)
  assumes ¬(*terminal-object Y*)

shows $\nexists\ s.\ s : X \to Y^X \wedge \mathit{surjective}(s)$
$\langle \mathit{proof} \rangle$

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.