

The Elementary Theory of the Category of Sets

James Baxter Dustin Bryant

March 17, 2025

Abstract

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

Contents

1	Basic Types and Operators for the Category of Sets	4
1.1	Tactics for Applying Typing Rules	5
1.1.1	typecheck_cfuncs: Tactic to Construct Type Facts	5
1.1.2	etcs_rule: Tactic to Apply Rules with ETCS Typechecking	5
1.1.3	etcs_subst: Tactic to Apply Substitutions with ETCS Typechecking	5
1.1.4	etcs_erule: Tactic to Apply Elimination Rules with ETCS Typechecking	6
1.2	Monomorphisms, Epimorphisms and Isomorphisms	6
1.2.1	Monomorphisms	6
1.2.2	Epimorphisms	7
1.2.3	Isomorphisms	7
2	Cartesian Products of Sets	9
2.1	Diagonal Functions	11
2.2	Products of Functions	11
2.3	Useful Cartesian Product Permuting Functions	13

2.3.1	Swapping a Cartesian Product	13
2.3.2	Permuting a Cartesian Product to Associate to the Right	13
2.3.3	Permuting a Cartesian Product to Associate to the Left	14
2.3.4	Distributing over a Cartesian Product from the Right	14
2.3.5	Distributing over a Cartesian Product from the Left .	15
2.3.6	Selecting Pairs from a Pair of Pairs	16
3	Terminal Objects and Elements	18
3.1	Set Membership and Emptiness	18
3.2	Terminal Objects (sets with one element)	19
3.3	Injectivity	20
3.4	Surjectivity	20
3.5	Interactions of Cartesian Products with Terminal Objects . .	21
3.5.1	Cartesian Products as Pullbacks	23
4	Equalizers and Subobjects	23
4.1	Equalizers	24
4.2	Subobjects	25
4.3	Inverse Image	26
4.4	Fibered Products	28
5	Truth Values and Characteristic Functions	30
5.1	Equality Predicate	32
5.2	Properties of Monomorphisms and Epimorphisms	33
5.3	Fiber Over an Element and its Connection to the Fibered Product	34
6	Set Subtraction	35
7	Graphs	37
8	Equivalence Classes and Coequalizers	38
8.1	Coequalizers	40
8.2	Regular Epimorphisms	41
8.3	Epi-monic Factorization	41
8.3.1	Image of a Function	42
8.4	<i>distribute-left</i> and <i>distribute-right</i> as Equivalence Relations .	45
9	Coproducts	46
9.1	Coproduct Function Properties	47
9.1.1	Equality Predicate with Coproduct Properties	48
9.2	Bowtie Product	49
9.3	Boolean Cases	50
9.4	Distribution of Products over Coproducts	52

9.4.1	Factor Product over Coproduct on Left	52
9.4.2	Distribute Product over Coproduct on Left	52
9.4.3	Factor Product over Coproduct on Right	53
9.4.4	Distribute Product over Coproduct on Right	54
9.5	Casting between Sets	55
9.5.1	Going from a Set or its Complement to the Superset	55
9.5.2	Going from a Set to a Subset or its Complement	55
9.6	Cases	56
9.7	Coproduct Set Properties	57
10	Axiom of Choice	58
11	Empty Set and Initial Objects	59
12	Exponential Objects, Transposes and Evaluation	62
12.1	Lifting Functions	62
12.2	Inverse Transpose Function (flat)	64
12.3	Metafunctions and their Inverses (Cnufatems)	65
12.3.1	Metafunctions	65
12.3.2	Inverse Metafunctions (Cnufatems)	66
12.3.3	Metafunction Composition	66
12.4	Partially Parameterized Functions on Pairs	68
12.5	Exponential Set Facts	69
13	Natural Number Object	70
13.1	Zero and Successor	72
13.2	Predecessor	72
13.3	Peano's Axioms and Induction	73
13.4	Function Iteration	74
13.5	Relation of Nat to Other Sets	76
14	Predicate Logic Functions	76
14.1	NOT	76
14.2	AND	77
14.3	NOR	78
14.4	OR	79
14.5	XOR	81
14.6	NAND	81
14.7	IFF	82
14.8	IMPLIES	83
14.9	Other Boolean Identities	85
15	Quantifiers	86
15.1	Universal Quantification	86
15.2	Existential Quantification	87

16 Natural Number Parity and Halving	88
16.1 Nth Even Number	88
16.2 Nth Odd Number	89
16.3 Checking if a Number is Even	89
16.4 Checking if a Number is Odd	90
16.5 Natural Number Halving	91
17 Cardinality and Finiteness	93
18 Countable Sets	96
19 Fixed Points and Cantor's Theorems	97

1 Basic Types and Operators for the Category of Sets

```
theory Cfunc
  imports Main HOL-Eisbach.Eisbach
begin
```

```
typedecl cset
typedecl cfunc
```

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

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

axiomatization

```
domain :: cfunc  $\Rightarrow$  cset and
codomain :: cfunc  $\Rightarrow$  cset and
comp :: cfunc  $\Rightarrow$  cfunc  $\Rightarrow$  cfunc (infixr  $\circ_c$  55) and
id :: cset  $\Rightarrow$  cfunc (idc)
```

where

```
domain-comp: domain g = codomain f  $\implies$  domain (g  $\circ_c$  f) = domain f and
codomain-comp: domain g = codomain f  $\implies$  codomain (g  $\circ_c$  f) = codomain g
and
comp-associative: domain h = codomain g  $\implies$  domain g = codomain f  $\implies$  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* \Rightarrow *cset* \Rightarrow *cset* \Rightarrow *bool* (- : - \rightarrow - [50, 50, 50]50)
where

(*f* : *X* \rightarrow *Y*) \longleftrightarrow (*domain* *f* = *X* \wedge *codomain* *f* = *Y*)

lemma *comp-type*:

f : *X* \rightarrow *Y* \Longrightarrow *g* : *Y* \rightarrow *Z* \Longrightarrow *g* \circ_c *f* : *X* \rightarrow *Z*
(*proof*)

lemma *comp-associative2*:

f : *X* \rightarrow *Y* \Longrightarrow *g* : *Y* \rightarrow *Z* \Longrightarrow *h* : *Z* \rightarrow *W* \Longrightarrow *h* \circ_c (*g* \circ_c *f*) = (*h* \circ_c *g*) \circ_c *f*
(*proof*)

lemma *id-type*: *id* *X* : *X* \rightarrow *X*

(*proof*)

lemma *id-right-unit2*: *f* : *X* \rightarrow *Y* \Longrightarrow *f* \circ_c *id* *X* = *f*

(*proof*)

lemma *id-left-unit2*: *f* : *X* \rightarrow *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*]

(*ML*)

1.1.1 typecheck_cfuncs: Tactic to Construct Type Facts

(*ML*)

1.1.2 etc_rule: Tactic to Apply Rules with ETCS Typechecking

(*ML*)

1.1.3 etc_subst: Tactic to Apply Substitutions with ETCS Typechecking

(*ML*)

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

⟨ML⟩

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

1.1.4 *etcs_erule*: Tactic to Apply Elimination Rules with ETCS Typechecking

⟨ML⟩

1.2 Monomorphisms, Epimorphisms and Isomorphisms

1.2.1 Monomorphisms

definition *monomorphism* :: *cfunc* ⇒ *bool* **where**

monomorphism *f* ⇔ (∀ *g h*.
(codomain *g* = domain *f* ∧ codomain *h* = domain *f*) → (*f* ∘_c *g* = *f* ∘_c *h* →
g = *h*))

lemma *monomorphism-def2*:

monomorphism *f* ⇔ (∀ *g h A X Y*. *g* : *A* → *X* ∧ *h* : *A* → *X* ∧ *f* : *X* → *Y*
→ (*f* ∘_c *g* = *f* ∘_c *h* → *g* = *h*))
⟨*proof*⟩

lemma *monomorphism-def3*:

assumes *f* : *X* → *Y*
shows *monomorphism* *f* ⇔ (∀ *g h A*. *g* : *A* → *X* ∧ *h* : *A* → *X* → (*f* ∘_c *g* =
f ∘_c *h* → *g* = *h*))
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7a in Halvorson.

lemma *comp-monic-imp-monic*:

assumes domain *g* = codomain *f*
shows *monomorphism* (*g* ∘_c *f*) ⇒ *monomorphism* *f*
⟨*proof*⟩

lemma *comp-monic-imp-monic'*:

assumes *f* : *X* → *Y* *g* : *Y* → *Z*
shows *monomorphism* (*g* ∘_c *f*) ⇒ *monomorphism* *f*
⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7c in Halvorson.

lemma *composition-of-monic-pair-is-monic*:

assumes codomain *f* = domain *g*
shows *monomorphism* *f* ⇒ *monomorphism* *g* ⇒ *monomorphism* (*g* ∘_c *f*)
⟨*proof*⟩

1.2.2 Epimorphisms

definition *epimorphism* :: *cfunc* \Rightarrow *bool* **where**

epimorphism $f \longleftrightarrow (\forall g h. \text{domain } g = \text{codomain } f \wedge \text{domain } h = \text{codomain } f \longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$

lemma *epimorphism-def2*:

epimorphism $f \longleftrightarrow (\forall g h A X Y. f : X \rightarrow Y \wedge g : Y \rightarrow A \wedge h : Y \rightarrow A \longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$
 ⟨*proof*⟩

lemma *epimorphism-def3*:

assumes $f : X \rightarrow Y$
shows *epimorphism* $f \longleftrightarrow (\forall g h A. g : Y \rightarrow A \wedge h : Y \rightarrow A \longrightarrow (g \circ_c f = h \circ_c f \longrightarrow g = h))$
 ⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7b in Halvorson.

lemma *comp-epi-imp-epi*:

assumes $\text{domain } g = \text{codomain } f$
shows *epimorphism* $(g \circ_c f) \Longrightarrow \text{epimorphism } g$
 ⟨*proof*⟩

The lemma below corresponds to Exercise 2.1.7d in Halvorson.

lemma *composition-of-epi-pair-is-epi*:

assumes $\text{codomain } f = \text{domain } g$
shows *epimorphism* $f \Longrightarrow \text{epimorphism } g \Longrightarrow \text{epimorphism } (g \circ_c f)$
 ⟨*proof*⟩

1.2.3 Isomorphisms

definition *isomorphism* :: *cfunc* \Rightarrow *bool* **where**

isomorphism $f \longleftrightarrow (\exists g. \text{domain } g = \text{codomain } f \wedge \text{codomain } g = \text{domain } f \wedge g \circ_c f = \text{id}(\text{domain } f) \wedge f \circ_c g = \text{id}(\text{domain } g))$

lemma *isomorphism-def2*:

isomorphism $f \longleftrightarrow (\exists g X Y. f : X \rightarrow Y \wedge g : Y \rightarrow X \wedge g \circ_c f = \text{id } X \wedge f \circ_c g = \text{id } Y)$
 ⟨*proof*⟩

lemma *isomorphism-def3*:

assumes $f : X \rightarrow Y$
shows *isomorphism* $f \longleftrightarrow (\exists g. g : Y \rightarrow X \wedge g \circ_c f = \text{id } X \wedge f \circ_c g = \text{id } Y)$
 ⟨*proof*⟩

definition *inverse* :: *cfunc* \Rightarrow *cfunc* $(^{-1}$ [1000] 999) **where**

inverse $f = (\text{THE } g. g : \text{codomain } f \rightarrow \text{domain } f \wedge g \circ_c f = \text{id}(\text{domain } f) \wedge f \circ_c g = \text{id}(\text{codomain } f))$

lemma *inverse-def2*:

assumes *isomorphism* f

shows $f^{-1} : \text{codomain } f \rightarrow \text{domain } f \wedge f^{-1} \circ_c f = \text{id}(\text{domain } f) \wedge f \circ_c f^{-1} = \text{id}(\text{codomain } f)$

<proof>

lemma *inverse-type*[*type-rule*]:

assumes *isomorphism* $f f : X \rightarrow Y$

shows $f^{-1} : Y \rightarrow X$

<proof>

lemma *inv-left*:

assumes *isomorphism* $f f : X \rightarrow Y$

shows $f^{-1} \circ_c f = \text{id } X$

<proof>

lemma *inv-right*:

assumes *isomorphism* $f f : X \rightarrow Y$

shows $f \circ_c f^{-1} = \text{id } Y$

<proof>

lemma *inv-iso*:

assumes *isomorphism* f

shows *isomorphism*(f^{-1})

<proof>

lemma *inv-idempotent*:

assumes *isomorphism* f

shows $(f^{-1})^{-1} = f$

<proof>

definition *is-isomorphic* :: *cset* \Rightarrow *cset* \Rightarrow *bool* (**infix** \cong 50) **where**

$X \cong Y \iff (\exists f. f : X \rightarrow Y \wedge \text{isomorphism } f)$

lemma *id-isomorphism*: *isomorphism* (*id* X)

<proof>

lemma *isomorphic-is-reflexive*: $X \cong X$

<proof>

lemma *isomorphic-is-symmetric*: $X \cong Y \implies Y \cong X$

<proof>

lemma *isomorphism-comp*:

$\text{domain } f = \text{codomain } g \implies \text{isomorphism } f \implies \text{isomorphism } g \implies \text{isomorphism } (f \circ_c g)$

<proof>

lemma *isomorphism-comp'*:

assumes $f : Y \rightarrow Z$ $g : X \rightarrow Y$
shows *isomorphism* $f \implies$ *isomorphism* $g \implies$ *isomorphism* $(f \circ_c g)$
 \langle *proof* \rangle

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

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

The lemma below corresponds to Exercise 2.1.7e in Halvorson.

lemma *iso-imp-epi-and-monic*:
isomorphism $f \implies$ *epimorphism* $f \wedge$ *monomorphism* f
 \langle *proof* \rangle

lemma *isomorphism-sandwich*:
assumes *f-type*: $f : A \rightarrow B$ **and** *g-type*: $g : B \rightarrow C$ **and** *h-type*: $h : C \rightarrow D$
assumes *f-iso*: *isomorphism* f
assumes *h-iso*: *isomorphism* h
assumes *hgf-iso*: *isomorphism* $(h \circ_c g \circ_c f)$
shows *isomorphism* g
 \langle *proof* \rangle

end

2 Cartesian Products of Sets

theory *Product*
imports *Cfunc*
begin

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

axiomatization

cart-prod :: *cset* \implies *cset* \implies *cset* (**infixr** \times_c 65) **and**
left-cart-proj :: *cset* \implies *cset* \implies *cfunc* **and**
right-cart-proj :: *cset* \implies *cset* \implies *cfunc* **and**
cfunc-prod :: *cfunc* \implies *cfunc* \implies *cfunc* (\langle -, \rangle)

where

left-cart-proj-type[*type-rule*]: *left-cart-proj* $X Y : X \times_c Y \rightarrow X$ **and**
right-cart-proj-type[*type-rule*]: *right-cart-proj* $X Y : X \times_c Y \rightarrow Y$ **and**
cfunc-prod-type[*type-rule*]: $f : Z \rightarrow X \implies g : Z \rightarrow Y \implies \langle f, g \rangle : Z \rightarrow X \times_c Y$

and

left-cart-proj-cfunc-prod: $f : Z \rightarrow X \implies g : Z \rightarrow Y \implies \text{left-cart-proj } X Y \circ_c \langle f, g \rangle = f$ **and**
right-cart-proj-cfunc-prod: $f : Z \rightarrow X \implies g : Z \rightarrow Y \implies \text{right-cart-proj } X Y \circ_c \langle f, g \rangle = g$ **and**
cfunc-prod-unique: $f : Z \rightarrow X \implies g : Z \rightarrow Y \implies h : Z \rightarrow X \times_c Y \implies$

$left\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ h = f \implies right\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ h = g \implies h = \langle f, g \rangle$

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

lemma $is\text{-}cart\text{-}prod\text{-}def2$:

assumes $\pi_0 : W \rightarrow X\ \pi_1 : W \rightarrow Y$
shows $is\text{-}cart\text{-}prod\ W\ \pi_0\ \pi_1\ X\ Y \longleftrightarrow$
 $(\forall f\ g\ Z. (f : Z \rightarrow X \wedge g : Z \rightarrow Y) \longrightarrow$
 $(\exists h. h : Z \rightarrow W \wedge \pi_0 \circ_c h = f \wedge \pi_1 \circ_c h = g \wedge$
 $(\forall h2. (h2 : Z \rightarrow W \wedge \pi_0 \circ_c h2 = f \wedge \pi_1 \circ_c h2 = g) \longrightarrow h2 = h)))$
 $\langle proof \rangle$

abbreviation $is\text{-}cart\text{-}prod\text{-}triple :: cset \times cfunc \times cfunc \Rightarrow cset \Rightarrow cset \Rightarrow bool$
where

$is\text{-}cart\text{-}prod\text{-}triple\ W\ \pi\ X\ Y \equiv is\text{-}cart\text{-}prod\ (fst\ W\ \pi)\ (fst\ (snd\ W\ \pi))\ (snd\ (snd\ W\ \pi))\ X\ Y$

lemma $canonical\text{-}cart\text{-}prod\text{-}is\text{-}cart\text{-}prod$:

$is\text{-}cart\text{-}prod\ (X \times_c Y)\ (left\text{-}cart\text{-}proj\ X\ Y)\ (right\text{-}cart\text{-}proj\ X\ Y)\ X\ Y$
 $\langle proof \rangle$

The lemma below corresponds to Proposition 2.1.8 in Halvorson.

lemma $cart\text{-}prods\text{-}isomorphic$:

assumes $W\text{-}cart\text{-}prod : is\text{-}cart\text{-}prod\text{-}triple\ (W, \pi_0, \pi_1)\ X\ Y$
assumes $W'\text{-}cart\text{-}prod : is\text{-}cart\text{-}prod\text{-}triple\ (W', \pi'_0, \pi'_1)\ X\ Y$
shows $\exists f. f : W \rightarrow W' \wedge isomorphic\ f \wedge \pi'_0 \circ_c f = \pi_0 \wedge \pi'_1 \circ_c f = \pi_1$
 $\langle proof \rangle$

lemma $product\text{-}commutes$:

$A \times_c B \cong B \times_c A$
 $\langle proof \rangle$

lemma $cart\text{-}prod\text{-}eq$:

assumes $a : Z \rightarrow X \times_c Y\ b : Z \rightarrow X \times_c Y$
shows $a = b \longleftrightarrow$
 $(left\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ a = left\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ b$
 $\wedge right\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ a = right\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ b)$
 $\langle proof \rangle$

lemma $cart\text{-}prod\text{-}eqI$:

assumes $a : Z \rightarrow X \times_c Y\ b : Z \rightarrow X \times_c Y$
assumes $(left\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ a = left\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ b$
 $\wedge right\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ a = right\text{-}cart\text{-}proj\ X\ Y\ \circ_c\ b)$
shows $a = b$

<proof>

lemma *cart-prod-eq2*:

assumes $a : Z \rightarrow X$ $b : Z \rightarrow Y$ $c : Z \rightarrow X$ $d : Z \rightarrow Y$

shows $\langle a, b \rangle = \langle c, d \rangle \iff (a = c \wedge b = d)$

<proof>

lemma *cart-prod-decomp*:

assumes $a : A \rightarrow X \times_c Y$

shows $\exists x y. a = \langle x, y \rangle \wedge x : A \rightarrow X \wedge y : A \rightarrow Y$

<proof>

2.1 Diagonal Functions

The definition below corresponds to Definition 2.1.9 in Halvorson.

definition *diagonal* :: $cset \Rightarrow cfunc$ **where**

$diagonal\ X = \langle id\ X, id\ X \rangle$

lemma *diagonal-type*[*type-rule*]:

$diagonal\ X : X \rightarrow X \times_c X$

<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* :: $cfunc \Rightarrow cfunc \Rightarrow cfunc$ (**infixr** \times_f 55) **where**

$f \times_f g = \langle f \circ_c left\text{-}cart\text{-}proj\ (domain\ f)\ (domain\ g), g \circ_c right\text{-}cart\text{-}proj\ (domain\ f)\ (domain\ g) \rangle$

lemma *cfunc-cross-prod-def2*:

assumes $f : X \rightarrow Y$ $g : V \rightarrow W$

shows $f \times_f g = \langle f \circ_c left\text{-}cart\text{-}proj\ X\ V, g \circ_c right\text{-}cart\text{-}proj\ X\ V \rangle$

<proof>

lemma *cfunc-cross-prod-type*[*type-rule*]:

$f : W \rightarrow Y \implies g : X \rightarrow Z \implies f \times_f g : W \times_c X \rightarrow Y \times_c Z$

<proof>

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

$f : W \rightarrow Y \implies g : X \rightarrow Z \implies left\text{-}cart\text{-}proj\ Y\ Z \circ_c f \times_f g = f \circ_c left\text{-}cart\text{-}proj\ W\ X$

<proof>

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

$f : W \rightarrow Y \implies g : X \rightarrow Z \implies \text{right-cart-proj } Y Z \circ_c f \times_f g = g \circ_c \text{right-cart-proj } W X$
 \(\langle \text{proof} \rangle\)

lemma *cfunc-cross-prod-unique*: $f : W \rightarrow Y \implies g : X \rightarrow Z \implies h : W \times_c X \rightarrow Y \times_c Z \implies$
 $\text{left-cart-proj } Y Z \circ_c h = f \circ_c \text{left-cart-proj } W X \implies$
 $\text{right-cart-proj } Y Z \circ_c h = g \circ_c \text{right-cart-proj } W X \implies h = f \times_f g$
 \(\langle \text{proof} \rangle\)

The lemma below corresponds to Proposition 2.1.11 in Halvorson.

lemma *identity-distributes-across-composition*:
assumes *f-type*: $f : A \rightarrow B$ **and** *g-type*: $g : B \rightarrow C$
shows $\text{id } X \times_f (g \circ_c f) = (\text{id } X \times_f g) \circ_c (\text{id } X \times_f f)$
 \(\langle \text{proof} \rangle\)

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

lemma *cfunc-prod-comp*:
assumes *f-type*: $f : X \rightarrow Y$
assumes *a-type*: $a : Y \rightarrow A$ **and** *b-type*: $b : Y \rightarrow B$
shows $\langle a, b \rangle \circ_c f = \langle a \circ_c f, b \circ_c f \rangle$
 \(\langle \text{proof} \rangle\)

The lemma below corresponds to Exercise 2.1.12 in Halvorson.

lemma *id-cross-prod*: $\text{id}(X) \times_f \text{id}(Y) = \text{id}(X \times_c Y)$
 \(\langle \text{proof} \rangle\)

The lemma below corresponds to Exercise 2.1.14 in Halvorson.

lemma *cfunc-cross-prod-comp-diagonal*:
assumes $f : X \rightarrow Y$
shows $(f \times_f f) \circ_c \text{diagonal}(X) = \text{diagonal}(Y) \circ_c f$
 \(\langle \text{proof} \rangle\)

lemma *cfunc-cross-prod-comp-cfunc-cross-prod*:
assumes $a : A \rightarrow X$ $b : B \rightarrow Y$ $x : X \rightarrow Z$ $y : Y \rightarrow W$
shows $(x \times_f y) \circ_c (a \times_f b) = (x \circ_c a) \times_f (y \circ_c b)$
 \(\langle \text{proof} \rangle\)

lemma *cfunc-cross-prod-mono*:
assumes *type-assms*: $f : X \rightarrow Y$ $g : Z \rightarrow W$
assumes *f-mono*: monomorphism f **and** *g-mono*: monomorphism g
shows monomorphism $(f \times_f g)$
 \(\langle \text{proof} \rangle\)

2.3 Useful Cartesian Product Permuting Functions

2.3.1 Swapping a Cartesian Product

definition *swap* :: *cset* \Rightarrow *cset* \Rightarrow *cfunc* **where**

swap *X Y* = \langle *right-cart-proj* *X Y*, *left-cart-proj* *X Y* \rangle

lemma *swap-type*[*type-rule*]: *swap* *X Y* : $X \times_c Y \rightarrow Y \times_c X$
\langle proof \rangle

lemma *swap-ap*:

assumes $x : A \rightarrow X$ $y : A \rightarrow Y$

shows *swap* *X Y* $\circ_c \langle x, y \rangle = \langle y, x \rangle$

\langle proof \rangle

lemma *swap-cross-prod*:

assumes $x : A \rightarrow X$ $y : B \rightarrow Y$

shows *swap* *X Y* $\circ_c (x \times_f y) = (y \times_f x) \circ_c$ *swap* *A B*

\langle proof \rangle

lemma *swap-idempotent*:

swap *Y X* \circ_c *swap* *X Y* = *id* ($X \times_c Y$)

\langle proof \rangle

lemma *swap-mono*:

monomorphism(*swap* *X Y*)

\langle proof \rangle

2.3.2 Permuting a Cartesian Product to Associate to the Right

definition *associate-right* :: *cset* \Rightarrow *cset* \Rightarrow *cset* \Rightarrow *cfunc* **where**

associate-right *X Y Z* =

\langle
 \langle
 left-cart-proj *X Y* \circ_c *left-cart-proj* ($X \times_c Y$) *Z*,
 \langle
 right-cart-proj *X Y* \circ_c *left-cart-proj* ($X \times_c Y$) *Z*,
 right-cart-proj ($X \times_c Y$) *Z*
 \rangle
 \rangle
 \rangle

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

lemma *associate-right-ap*:

assumes $x : A \rightarrow X$ $y : A \rightarrow Y$ $z : A \rightarrow Z$

shows *associate-right* *X Y Z* $\circ_c \langle \langle x, y \rangle, z \rangle = \langle x, \langle y, z \rangle \rangle$

\langle proof \rangle

lemma *associate-right-crossprod-ap*:

assumes $x : A \rightarrow X \ y : B \rightarrow Y \ z : C \rightarrow Z$
shows $\text{associate-right } X \ Y \ Z \circ_c ((x \times_f y) \times_f z) = (x \times_f (y \times_f z)) \circ_c \text{associate-right } A \ B \ C$
 <proof>

2.3.3 Permuting a Cartesian Product to Associate to the Left

definition $\text{associate-left} :: \text{cset} \Rightarrow \text{cset} \Rightarrow \text{cset} \Rightarrow \text{cfunc}$ **where**

$\text{associate-left } X \ Y \ Z =$
 <
 <
 $\text{left-cart-proj } X \ (Y \times_c Z),$
 $\text{left-cart-proj } Y \ Z \circ_c \text{right-cart-proj } X \ (Y \times_c Z)$
 >,
 $\text{right-cart-proj } Y \ Z \circ_c \text{right-cart-proj } X \ (Y \times_c Z)$
 >

lemma $\text{associate-left-type}[\text{type-rule}]$: $\text{associate-left } X \ Y \ Z : X \times_c (Y \times_c Z) \rightarrow (X \times_c Y) \times_c Z$
 <proof>

lemma associate-left-ap :

assumes $x : A \rightarrow X \ y : A \rightarrow Y \ z : A \rightarrow Z$
shows $\text{associate-left } X \ Y \ Z \circ_c \langle x, \langle y, z \rangle \rangle = \langle \langle x, y \rangle, z \rangle$
 <proof>

lemma right-left :

$\text{associate-right } A \ B \ C \circ_c \text{associate-left } A \ B \ C = \text{id } (A \times_c (B \times_c C))$
 <proof>

lemma left-right :

$\text{associate-left } A \ B \ C \circ_c \text{associate-right } A \ B \ C = \text{id } ((A \times_c B) \times_c C)$
 <proof>

lemma $\text{product-associates}$:

$A \times_c (B \times_c C) \cong (A \times_c B) \times_c C$
 <proof>

lemma $\text{associate-left-crossprod-ap}$:

assumes $x : A \rightarrow X \ y : B \rightarrow Y \ z : C \rightarrow Z$
shows $\text{associate-left } X \ Y \ Z \circ_c (x \times_f (y \times_f z)) = ((x \times_f y) \times_f z) \circ_c \text{associate-left } A \ B \ C$
 <proof>

2.3.4 Distributing over a Cartesian Product from the Right

definition $\text{distribute-right-left} :: \text{cset} \Rightarrow \text{cset} \Rightarrow \text{cset} \Rightarrow \text{cfunc}$ **where**

$\text{distribute-right-left } X \ Y \ Z =$
 < $\text{left-cart-proj } X \ Y \circ_c \text{left-cart-proj } (X \times_c Y) \ Z, \text{right-cart-proj } (X \times_c Y) \ Z$ >

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

lemma *distribute-right-left-ap*:
assumes $x : A \rightarrow X \ y : A \rightarrow Y \ z : A \rightarrow Z$
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* :: $cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**
distribute-right-right $X Y Z =$
 ⟨*right-cart-proj* $X Y \circ_c$ *left-cart-proj* $(X \times_c Y) Z, \textit{right-cart-proj} (X \times_c Y) Z$ ⟩

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

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

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

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

lemma *distribute-right-ap*:
assumes $x : A \rightarrow X \ y : A \rightarrow Y \ z : A \rightarrow Z$
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* :: $cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**
distribute-left-left $X Y Z =$
 ⟨*left-cart-proj* $X (Y \times_c Z), \textit{left-cart-proj} Y Z \circ_c \textit{right-cart-proj} X (Y \times_c Z)$ ⟩

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

lemma *distribute-left-left-ap*:

assumes $x : A \rightarrow X \ y : A \rightarrow Y \ z : A \rightarrow Z$
shows $\text{distribute-left-left } X \ Y \ Z \circ_c \langle x, \langle y, z \rangle \rangle = \langle x, y \rangle$
 $\langle \text{proof} \rangle$

definition *distribute-left-right* :: $cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**

$\text{distribute-left-right } X \ Y \ Z =$
 $\langle \text{left-cart-proj } X \ (Y \times_c Z), \text{right-cart-proj } Y \ Z \circ_c \text{right-cart-proj } X \ (Y \times_c Z) \rangle$

lemma *distribute-left-right-type*[*type-rule*]:

$\text{distribute-left-right } X \ Y \ Z : X \times_c (Y \times_c Z) \rightarrow X \times_c Z$
 $\langle \text{proof} \rangle$

lemma *distribute-left-right-ap*:

assumes $x : A \rightarrow X \ y : A \rightarrow Y \ z : A \rightarrow Z$
shows $\text{distribute-left-right } X \ Y \ Z \circ_c \langle x, \langle y, z \rangle \rangle = \langle x, z \rangle$
 $\langle \text{proof} \rangle$

definition *distribute-left* :: $cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**

$\text{distribute-left } X \ Y \ Z = \langle \text{distribute-left-left } X \ Y \ Z, \text{distribute-left-right } X \ Y \ Z \rangle$

lemma *distribute-left-type*[*type-rule*]:

$\text{distribute-left } X \ Y \ Z : X \times_c (Y \times_c Z) \rightarrow (X \times_c Y) \times_c (X \times_c Z)$
 $\langle \text{proof} \rangle$

lemma *distribute-left-ap*:

assumes $x : A \rightarrow X \ y : A \rightarrow Y \ z : A \rightarrow Z$
shows $\text{distribute-left } X \ Y \ Z \circ_c \langle x, \langle y, z \rangle \rangle = \langle \langle x, y \rangle, \langle x, z \rangle \rangle$
 $\langle \text{proof} \rangle$

lemma *distribute-left-mono*:

monomorphism ($\text{distribute-left } X \ Y \ Z$)
 $\langle \text{proof} \rangle$

2.3.6 Selecting Pairs from a Pair of Pairs

definition *outers* :: $cset \Rightarrow cset \Rightarrow cset \Rightarrow cset \Rightarrow cfunc$ **where**

$\text{outers } A \ B \ C \ D = \langle$
 $\text{left-cart-proj } A \ B \circ_c \text{left-cart-proj } (A \times_c B) \ (C \times_c D),$
 $\text{right-cart-proj } C \ D \circ_c \text{right-cart-proj } (A \times_c B) \ (C \times_c D)$
 \rangle

lemma *outers-type*[*type-rule*]: $\text{outers } A \ B \ C \ D : (A \times_c B) \times_c (C \times_c D) \rightarrow (A \times_c D)$

$\langle \text{proof} \rangle$

lemma *outers-apply*:

assumes $a : Z \rightarrow A \ b : Z \rightarrow B \ c : Z \rightarrow C \ d : Z \rightarrow D$
shows $\text{outers } A \ B \ C \ D \circ_c \langle \langle a, b \rangle, \langle c, d \rangle \rangle = \langle a, d \rangle$

<proof>

definition *inners* :: *cset* \Rightarrow *cset* \Rightarrow *cset* \Rightarrow *cset* \Rightarrow *cfunc* **where**

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

3 Terminal Objects and Elements

```

theory Terminal
  imports Cfunc Product
begin

```

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

axiomatization

```

terminal-func :: cset  $\Rightarrow$  cfunc ( $\beta$ . 100) and
one-set :: cset (1)

```

where

```

terminal-func-type[type-rule]:  $\beta_X : X \rightarrow \mathbf{1}$  and
terminal-func-unique:  $h : X \rightarrow \mathbf{1} \implies h = \beta_X$  and
one-separator:  $f : X \rightarrow Y \implies g : X \rightarrow Y \implies (\bigwedge x. x : \mathbf{1} \rightarrow X \implies f \circ_c x = g \circ_c x) \implies f = g$ 

```

lemma one-separator-contrapos:

```

assumes  $f : X \rightarrow Y$   $g : X \rightarrow Y$ 
shows  $f \neq g \implies \exists x. x : \mathbf{1} \rightarrow X \wedge f \circ_c x \neq g \circ_c x$ 
<proof>

```

lemma terminal-func-comp:

```

 $x : X \rightarrow Y \implies \beta_Y \circ_c x = \beta_X$ 
<proof>

```

lemma terminal-func-comp-elem:

```

 $x : \mathbf{1} \rightarrow X \implies \beta_X \circ_c x = id \ \mathbf{1}$ 
<proof>

```

3.1 Set Membership and Emptiness

The abbreviation below captures Definition 2.1.16 in Halvorson.

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

definition nonempty :: cset \Rightarrow bool **where**

```

nonempty X  $\equiv (\exists x. x \in_c X)$ 

```

definition is-empty :: cset \Rightarrow bool **where**

```

is-empty X  $\equiv \neg(\exists x. x \in_c X)$ 

```

The lemma below corresponds to Exercise 2.1.18 in Halvorson.

lemma element-monomorphism:

```

 $x \in_c X \implies monomorphism \ x$ 
<proof>

```

lemma one-unique-element:

```

 $\exists! x. x \in_c \ \mathbf{1}$ 

```

<proof>

lemma *prod-with-empty-is-empty1*:
 assumes *is-empty* (A)
 shows *is-empty*(A \times_c B)
 <proof>

lemma *prod-with-empty-is-empty2*:
 assumes *is-empty* (B)
 shows *is-empty* (A \times_c B)
 <proof>

3.2 Terminal Objects (sets with one element)

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

lemma *one-terminal-object*: *terminal-object*(1)
 <proof>

The lemma below is a generalisation of $?x \in_c ?X \implies \text{monomorphism } ?x$

lemma *terminal-el-monomorphism*:
 assumes $x : T \rightarrow X$
 assumes *terminal-object* T
 shows *monomorphism* x
 <proof>

The lemma below corresponds to Exercise 2.1.15 in Halvorson.

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

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

lemma *iso-to1-is-term*:
 assumes $X \cong \mathbf{1}$
 shows *terminal-object* X
 <proof>

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

The lemma below corresponds to Proposition 2.1.19 in Halvorson.

lemma *single-elem-iso-one*:
 ($\exists!$ x. $x \in_c X$) \longleftrightarrow $X \cong \mathbf{1}$
 <proof>

3.3 Injectivity

The definition below corresponds to Definition 2.1.24 in Halvorson.

definition *injective* :: *cfunc* \Rightarrow *bool* **where**

injective $f \iff (\forall x y. (x \in_c \text{domain } f \wedge y \in_c \text{domain } f \wedge f \circ_c x = f \circ_c y) \longrightarrow x = y)$

lemma *injective-def2*:

assumes $f : X \rightarrow Y$

shows *injective* $f \iff (\forall x y. (x \in_c X \wedge y \in_c X \wedge f \circ_c x = f \circ_c y) \longrightarrow x = y)$
<proof>

The lemma below corresponds to Exercise 2.1.26 in Halvorson.

lemma *monomorphism-imp-injective*:

monomorphism $f \implies \text{injective } f$

<proof>

The lemma below corresponds to Proposition 2.1.27 in Halvorson.

lemma *injective-imp-monomorphism*:

injective $f \implies \text{monomorphism } f$

<proof>

lemma *cfunc-cross-prod-inj*:

assumes *type-assms*: $f : X \rightarrow Y \ g : Z \rightarrow W$

assumes *injective* $f \wedge \text{injective } g$

shows *injective* $(f \times_f g)$

<proof>

lemma *cfunc-cross-prod-mono-converse*:

assumes *type-assms*: $f : X \rightarrow Y \ g : Z \rightarrow W$

assumes *fg-inject*: *injective* $(f \times_f g)$

assumes *nonempty*: *nonempty* $X \ \text{nonempty } Z$

shows *injective* $f \wedge \text{injective } g$

<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 $f : X \rightarrow Y \ g : Z \rightarrow W$

assumes $\neg(\text{nonempty } X) \vee \neg(\text{nonempty } Z)$

shows *injective* $(f \times_f g)$

<proof>

3.4 Surjectivity

The definition below corresponds to Definition 2.1.28 in Halvorson.

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

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

The lemma below corresponds to Exercise 2.1.30 in Halvorson.

lemma *surjective-is-epimorphism*:
surjective $f \implies \text{epimorphism } f$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Proposition 2.2.10 in Halvorson.

lemma *cfunc-cross-prod-surj*:
assumes *type-assms*: $f : A \rightarrow C \ g : B \rightarrow D$
assumes *f-surj*: *surjective* f **and** *g-surj*: *surjective* g
shows *surjective* $(f \times_f g)$
 $\langle \text{proof} \rangle$

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

3.5 Interactions of Cartesian Products with Terminal Objects

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

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

The lemma below corresponds to Proposition 2.1.20 in Halvorson.

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

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

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

$$X \times_c \mathbf{1} \cong X$$

<proof>

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

$$\mathbf{1} \times_c X \cong X$$

<proof>

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

lemma *left-cart-proj-one-left-inverse:*

$$\langle id_X, \beta_X \rangle \circ_c \text{left-cart-proj } X \mathbf{1} = id (X \times_c \mathbf{1})$$

<proof>

lemma *left-cart-proj-one-right-inverse:*

$$\text{left-cart-proj } X \mathbf{1} \circ_c \langle id_X, \beta_X \rangle = id X$$

<proof>

lemma *right-cart-proj-one-left-inverse:*

$$\langle \beta_X, id_X \rangle \circ_c \text{right-cart-proj } \mathbf{1} X = id (\mathbf{1} \times_c X)$$

<proof>

lemma *right-cart-proj-one-right-inverse:*

$$\text{right-cart-proj } \mathbf{1} X \circ_c \langle \beta_X, id_X \rangle = id X$$

<proof>

lemma *cfunc-cross-prod-right-terminal-decomp:*

assumes $f : X \rightarrow Y \quad x : \mathbf{1} \rightarrow Z$
shows $f \times_f x = \langle f, x \circ_c \beta_X \rangle \circ_c \text{left-cart-proj } X \mathbf{1}$

<proof>

The lemma below corresponds to Proposition 2.1.21 in Halvorson.

lemma *cart-prod-elem-eq:*

assumes $a \in_c X \times_c Y \quad b \in_c X \times_c Y$
shows $a = b \iff$
 $(\text{left-cart-proj } X Y \circ_c a = \text{left-cart-proj } X Y \circ_c b$
 $\wedge \text{right-cart-proj } X Y \circ_c a = \text{right-cart-proj } X Y \circ_c b)$

<proof>

The lemma below corresponds to Note 2.1.22 in Halvorson.

lemma *element-pair-eq:*

assumes $x \in_c X \quad x' \in_c X \quad y \in_c Y \quad y' \in_c Y$
shows $\langle x, y \rangle = \langle x', y' \rangle \iff x = x' \wedge y = y'$

<proof>

The lemma below corresponds to Proposition 2.1.23 in Halvorson.

lemma *nonempty-right-imp-left-proj-epimorphism:*

nonempty $Y \implies \text{epimorphism } (\text{left-cart-proj } X Y)$

<proof>

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

lemma *nonempty-left-imp-right-proj-epimorphism*:
nonempty X \implies *epimorphism (right-cart-proj X Y)*
 <proof>

lemma *cart-prod-extract-left*:
 assumes $f : \mathbf{1} \rightarrow X$ $g : \mathbf{1} \rightarrow Y$
 shows $\langle f, g \rangle = \langle id\ X, g \circ_c \beta_X \rangle \circ_c f$
 <proof>

lemma *cart-prod-extract-right*:
 assumes $f : \mathbf{1} \rightarrow X$ $g : \mathbf{1} \rightarrow Y$
 shows $\langle f, g \rangle = \langle f \circ_c \beta_Y, id\ Y \rangle \circ_c g$
 <proof>

3.5.1 Cartesian Products as Pullbacks

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

definition *is-pullback* :: *cset* \implies *cset* \implies *cset* \implies *cset* \implies *cfunc* \implies *cfunc* \implies *cfunc* \implies *cfunc* \implies *bool* **where**

is-pullback A B C D ab bd ac cd \iff
 ($ab : A \rightarrow B \wedge bd : B \rightarrow D \wedge ac : A \rightarrow C \wedge cd : C \rightarrow D \wedge bd \circ_c ab = cd \circ_c ac \wedge$
 ($\forall Z\ k\ h. (k : Z \rightarrow B \wedge h : Z \rightarrow C \wedge bd \circ_c k = cd \circ_c h) \implies$
 ($\exists! j. j : Z \rightarrow A \wedge ab \circ_c j = k \wedge ac \circ_c j = h$)))

lemma *pullback-unique*:
 assumes $ab : A \rightarrow B$ $bd : B \rightarrow D$ $ac : A \rightarrow C$ $cd : C \rightarrow D$
 assumes $k : Z \rightarrow B$ $h : Z \rightarrow C$
 assumes *is-pullback A B C D ab bd ac cd*
 shows $bd \circ_c k = cd \circ_c h \implies (\exists! j. j : Z \rightarrow A \wedge ab \circ_c j = k \wedge ac \circ_c j = h)$
 <proof>

lemma *pullback-iff-product*:
 assumes *terminal-object(T)*
 assumes *f-type[type-rule]: f : Y \rightarrow T*
 assumes *g-type[type-rule]: g : X \rightarrow T*
 shows $(is-pullback\ P\ Y\ X\ T\ (pY)\ f\ (pX)\ g) = (is-cart-prod\ P\ pX\ pY\ X\ Y)$
 <proof>

end

4 Equalizers and Subobjects

theory *Equalizer*
 imports *Terminal*
 begin

4.1 Equalizers

definition *equalizer* :: *cset* \Rightarrow *cfunc* \Rightarrow *cfunc* \Rightarrow *cfunc* \Rightarrow *bool* **where**

equalizer $E\ m\ f\ g \longleftrightarrow (\exists\ X\ Y. (f : X \rightarrow Y) \wedge (g : X \rightarrow Y) \wedge (m : E \rightarrow X) \wedge (f \circ_c m = g \circ_c m) \wedge (\forall\ h\ F. ((h : F \rightarrow X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k. (k : F \rightarrow E) \wedge m \circ_c k = h)))$

lemma *equalizer-def2*:

assumes $f : X \rightarrow Y\ g : X \rightarrow Y\ m : E \rightarrow X$
shows *equalizer* $E\ m\ f\ g \longleftrightarrow ((f \circ_c m = g \circ_c m) \wedge (\forall\ h\ F. ((h : F \rightarrow X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k. (k : F \rightarrow E) \wedge m \circ_c k = h)))$
 \langle *proof* \rangle

lemma *equalizer-eq*:

assumes $f : X \rightarrow Y\ g : X \rightarrow Y\ m : E \rightarrow X$
assumes *equalizer* $E\ m\ f\ g$
shows $f \circ_c m = g \circ_c m$
 \langle *proof* \rangle

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 $\exists!\ k. k : F \rightarrow E \wedge m \circ_c k = h$
 \langle *proof* \rangle

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

axiomatization where

equalizer-exists: $f : X \rightarrow Y \Longrightarrow g : X \rightarrow Y \Longrightarrow \exists\ E\ m. \text{equalizer } E\ m\ f\ g$

lemma *equalizer-exists2*:

assumes $f : X \rightarrow Y\ g : X \rightarrow Y$
shows $\exists\ E\ m. m : E \rightarrow X \wedge f \circ_c m = g \circ_c m \wedge (\forall\ h\ F. ((h : F \rightarrow X) \wedge (f \circ_c h = g \circ_c h)) \longrightarrow (\exists!\ k. (k : F \rightarrow E) \wedge m \circ_c k = h))$
 \langle *proof* \rangle

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 $\exists\ k. k : E \rightarrow E' \wedge \text{isomorphism } k \wedge m = m' \circ_c k$
 \langle *proof* \rangle

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 $\text{equalizer } E' (m \circ_c \varphi) f g$
 ⟨proof⟩

The lemma below corresponds to Exercise 2.1.34 in Halvorson.

lemma *equalizer-is-monomorphism*:
 $\text{equalizer } E m f g \implies \text{monomorphism}(m)$
 ⟨proof⟩

The definition below corresponds to Definition 2.1.35 in Halvorson.

definition *regular-monomorphism* :: $\text{cfunc} \Rightarrow \text{bool}$
where *regular-monomorphism* $f \iff$
 $(\exists g h. \text{domain } g = \text{codomain } f \wedge \text{domain } h = \text{codomain } f \wedge \text{equalizer}$
 $(\text{domain } f) f g h)$

The lemma below corresponds to Exercise 2.1.36 in Halvorson.

lemma *epi-regmon-is-iso*:
assumes *epimorphism* f *regular-monomorphism* f
shows *isomorphism* f
 ⟨proof⟩

4.2 Subobjects

The definition below corresponds to Definition 2.1.32 in Halvorson.

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

lemma *factors-through-def2*:
assumes $g : X \rightarrow Z f : Y \rightarrow Z$
shows $g \text{ factorsthru } f \iff (\exists h. h: X \rightarrow Y \wedge f \circ_c h = g)$
 ⟨proof⟩

The lemma below corresponds to Exercise 2.1.33 in Halvorson.

lemma *xfactorthru-equalizer-iff-fx-eq-gx*:
assumes $f: X \rightarrow Y g: X \rightarrow Y \text{equalizer } E m f g x \in_c X$
shows $x \text{ factorthru } m \iff f \circ_c x = g \circ_c x$
 ⟨proof⟩

The definition below corresponds to Definition 2.1.37 in Halvorson.

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

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

definition *relative-subset* :: $\text{cset} \times \text{cfunc} \Rightarrow \text{cset} \Rightarrow \text{cset} \times \text{cfunc} \Rightarrow \text{bool}$ ($-\subseteq_c-$
 $[51, 50, 51] 50)$

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

lemma *relative-subset-def2*:

$(B,m) \subseteq_X (A,n) = (m : B \rightarrow X \wedge monomorphism\ m \wedge n : A \rightarrow X \wedge monomor-$
 $phism\ n$
 $\wedge (\exists\ k. k : B \rightarrow A \wedge n \circ_c k = m))$
 $\langle proof \rangle$

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

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

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

lemma *relative-member-def2*:

$x \in_X (B, m) = (x \in_c X \wedge monomorphism\ m \wedge m : B \rightarrow X \wedge x\ factorsthru\ m)$
 $\langle proof \rangle$

The lemma below corresponds to Proposition 2.1.40 in Halvorson.

lemma *relative-subobject-member*:

assumes $(A,n) \subseteq_X (B,m)$ $x \in_c X$
shows $x \in_X (A,n) \implies x \in_X (B,m)$
 $\langle proof \rangle$

4.3 Inverse Image

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

definition *inverse-image* :: $cfunc \Rightarrow cset \Rightarrow cfunc \Rightarrow cset$ ($-^{-1}(|-)_ [101,0,0] 100$)

where

$inverse-image\ f\ B\ m = (SOME\ A. \exists\ X\ Y\ k. f : X \rightarrow Y \wedge m : B \rightarrow Y \wedge$
 $monomorphism\ m \wedge$
 $equalizer\ A\ k\ (f \circ_c left-cart-proj\ X\ B)\ (m \circ_c right-cart-proj\ X\ B))$

lemma *inverse-image-is-equalizer*:

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ $monomorphism\ m$
shows $\exists k. equalizer\ (f^{-1}(|B|)_m)\ k\ (f \circ_c left-cart-proj\ X\ B)\ (m \circ_c right-cart-proj\ X\ B)$
 $\langle proof \rangle$

definition *inverse-image-mapping* :: $cfunc \Rightarrow cset \Rightarrow cfunc \Rightarrow cfunc$ **where**

$inverse-image-mapping\ f\ B\ m = (SOME\ k. \exists\ X\ Y. f : X \rightarrow Y \wedge m : B \rightarrow Y \wedge$
 $monomorphism\ m \wedge$

equalizer (inverse-image f B m) k (f ∘_c left-cart-proj X B) (m ∘_c right-cart-proj X B))

lemma *inverse-image-is-equalizer2:*

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism m*

shows *equalizer (inverse-image f B m) (inverse-image-mapping f B m) (f ∘_c left-cart-proj X B) (m ∘_c right-cart-proj X B)*

<proof>

lemma *inverse-image-mapping-type[type-rule]:*

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism m*

shows *inverse-image-mapping f B m : (inverse-image f B m) \rightarrow $X \times_c B$*

<proof>

lemma *inverse-image-mapping-eq:*

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism m*

shows *f ∘_c left-cart-proj X B ∘_c inverse-image-mapping f B m*

= m ∘_c right-cart-proj X B ∘_c inverse-image-mapping f B m

<proof>

lemma *inverse-image-mapping-monomorphism:*

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *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 $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism m*

shows *monomorphism (left-cart-proj X B ∘_c inverse-image-mapping f B m)*

<proof>

definition *inverse-image-subobject-mapping :: cfunc \Rightarrow cset \Rightarrow cfunc \Rightarrow cfunc*

([⁻¹(-)]map [101,0,0]100) where

[⁻¹(B)]m]map = left-cart-proj (domain f) B ∘_c inverse-image-mapping f B m

lemma *inverse-image-subobject-mapping-def2:*

assumes $f : X \rightarrow Y$

shows *[⁻¹(B)]m]map = left-cart-proj X B ∘_c inverse-image-mapping f B m*

<proof>

lemma *inverse-image-subobject-mapping-type[type-rule]:*

assumes $f : X \rightarrow Y$ $m : B \rightarrow Y$ *monomorphism m*

shows *[⁻¹(B)]m]map : [⁻¹(B)]m \rightarrow X*

<proof>

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

assumes $f : X \rightarrow Y$ $m : B \rightarrow Y$ *monomorphism m*

shows *monomorphism ([⁻¹(B)]m]map)*

<proof>

lemma *inverse-image-subobject*:

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism* m
shows $(f^{-1}(\downarrow B))_m, [f^{-1}(\downarrow B)]_{map} \subseteq_c X$
 $\langle proof \rangle$

lemma *inverse-image-pullback*:

assumes $m : B \rightarrow Y$ $f : X \rightarrow Y$ *monomorphism* m
shows *is-pullback* $(f^{-1}(\downarrow B))_m B X Y$
 $(right\text{-}cart\text{-}proj\ X\ B\ \circ_c\ inverse\text{-}image\text{-}mapping\ f\ B\ m)\ m$
 $(left\text{-}cart\text{-}proj\ X\ B\ \circ_c\ inverse\text{-}image\text{-}mapping\ f\ B\ m)\ f$
 $\langle proof \rangle$

The lemma below corresponds to Proposition 2.1.41 in Halvorson.

lemma *in-inverse-image*:

assumes $f : X \rightarrow Y$ $(B, m) \subseteq_c Y$ $x \in_c X$
shows $(x \in_X (f^{-1}(\downarrow B))_m, left\text{-}cart\text{-}proj\ X\ B\ \circ_c\ inverse\text{-}image\text{-}mapping\ f\ B\ m) =$
 $(f \circ_c x \in_Y (B, m))$
 $\langle proof \rangle$

4.4 Fibered Products

The definition below corresponds to Definition 2.1.42 in Halvorson.

definition *fibered-product* $:: cset \Rightarrow cfunc \Rightarrow cfunc \Rightarrow cset \Rightarrow cset$ $(- \times_c -)$
 $[66, 50, 50, 65]65$ **where**
 $X \times_{f \circ c g} Y = (SOME\ E.\ \exists\ Z.\ m.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
 $equalizer\ E\ m\ (f \circ_c left\text{-}cart\text{-}proj\ X\ Y)\ (g \circ_c right\text{-}cart\text{-}proj\ X\ Y))$

lemma *fibered-product-equalizer*:

assumes $f : X \rightarrow Z$ $g : Y \rightarrow Z$
shows $\exists\ m.\ equalizer\ (X \times_{f \circ c g} Y)\ m\ (f \circ_c left\text{-}cart\text{-}proj\ X\ Y)\ (g \circ_c right\text{-}cart\text{-}proj\ X\ Y)$
 $\langle proof \rangle$

definition *fibered-product-morphism* $:: cset \Rightarrow cfunc \Rightarrow cfunc \Rightarrow cset \Rightarrow cfunc$
where

fibered-product-morphism $X\ f\ g\ Y = (SOME\ m.\ \exists\ Z.\ f : X \rightarrow Z \wedge g : Y \rightarrow Z \wedge$
 $equalizer\ (X \times_{f \circ c g} Y)\ m\ (f \circ_c left\text{-}cart\text{-}proj\ X\ Y)\ (g \circ_c right\text{-}cart\text{-}proj\ X\ Y))$

lemma *fibered-product-morphism-equalizer*:

assumes $f : X \rightarrow Z$ $g : Y \rightarrow Z$
shows $equalizer\ (X \times_{f \circ c g} Y)\ (fibered\text{-}product\text{-}morphism\ X\ f\ g\ Y)\ (f \circ_c left\text{-}cart\text{-}proj\ X\ Y)\ (g \circ_c right\text{-}cart\text{-}proj\ X\ Y)$
 $\langle proof \rangle$

lemma *fibered-product-morphism-type* $[type\text{-}rule]$:

assumes $f : X \rightarrow Z$ $g : Y \rightarrow Z$
shows $fibered\text{-}product\text{-}morphism\ X\ f\ g\ Y : X \times_{f \circ c g} Y \rightarrow X \times_c Y$
 $\langle proof \rangle$

lemma *fibred-product-morphism-monomorphism*:

assumes $f : X \rightarrow Z \ g : Y \rightarrow Z$

shows *monomorphism* (*fibred-product-morphism* $X \ f \ g \ Y$)

<proof>

definition *fibred-product-left-proj* :: $cset \Rightarrow cfunc \Rightarrow cfunc \Rightarrow cset \Rightarrow cfunc$ **where**

fibred-product-left-proj $X \ f \ g \ Y = (\text{left-cart-proj } X \ Y) \circ_c (\text{fibred-product-morphism } X \ f \ g \ Y)$

lemma *fibred-product-left-proj-type*[*type-rule*]:

assumes $f : X \rightarrow Z \ g : Y \rightarrow Z$

shows *fibred-product-left-proj* $X \ f \ g \ Y : X \ \mathit{f} \times_{cg} \ Y \rightarrow X$

<proof>

definition *fibred-product-right-proj* :: $cset \Rightarrow cfunc \Rightarrow cfunc \Rightarrow cset \Rightarrow cfunc$

where

fibred-product-right-proj $X \ f \ g \ Y = (\text{right-cart-proj } X \ Y) \circ_c (\text{fibred-product-morphism } X \ f \ g \ Y)$

lemma *fibred-product-right-proj-type*[*type-rule*]:

assumes $f : X \rightarrow Z \ g : Y \rightarrow Z$

shows *fibred-product-right-proj* $X \ f \ g \ Y : X \ \mathit{f} \times_{cg} \ Y \rightarrow Y$

<proof>

lemma *pair-factorsthru-fibred-product-morphism*:

assumes $f : X \rightarrow Z \ g : Y \rightarrow Z \ x : A \rightarrow X \ y : A \rightarrow Y$

shows $f \circ_c x = g \circ_c y \implies \langle x, y \rangle \text{ factorsthru } \text{fibred-product-morphism } X \ f \ g \ Y$

<proof>

lemma *fibred-product-is-pullback*:

assumes *f-type*[*type-rule*]: $f : X \rightarrow Z$ **and** *g-type*[*type-rule*]: $g : Y \rightarrow Z$

shows *is-pullback* $(X \ \mathit{f} \times_{cg} \ Y) \ Y \ X \ Z \ (\text{fibred-product-right-proj } X \ f \ g \ Y) \ g$
 $(\text{fibred-product-left-proj } X \ f \ g \ Y) \ f$

<proof>

lemma *fibred-product-proj-eq*:

assumes $f : X \rightarrow Z \ g : Y \rightarrow Z$

shows $f \circ_c \text{fibred-product-left-proj } X \ f \ g \ Y = g \circ_c \text{fibred-product-right-proj } X \ f \ g \ Y$

<proof>

lemma *fibred-product-pair-member*:

assumes $f : X \rightarrow Z \ g : Y \rightarrow Z \ x \in_c X \ y \in_c Y$

shows $(\langle x, y \rangle \in_X \times_c Y \ (X \ \mathit{f} \times_{cg} \ Y, \ \text{fibred-product-morphism } X \ f \ g \ Y)) = (f \circ_c x = g \circ_c y)$

<proof>

lemma *fibred-product-pair-member2*:

assumes $f : X \rightarrow Y$ $g : X \rightarrow E$ $x \in_c X$ $y \in_c X$
assumes $g \circ_c \text{fibered-product-left-proj } X f f X = g \circ_c \text{fibered-product-right-proj } X f f X$
shows $\forall x y. x \in_c X \rightarrow y \in_c X \rightarrow \langle x, y \rangle \in_{X \times_c X} (X \times_c f X, \text{fibered-product-morphism } X f f X) \rightarrow g \circ_c x = g \circ_c y$
<proof>

lemma *kernel-pair-subset:*

assumes $f : X \rightarrow Y$
shows $(X \times_c f X, \text{fibered-product-morphism } X f f X) \subseteq_c X \times_c X$
<proof>

The three lemmas below correspond to Exercise 2.1.44 in Halvorson.

lemma *kern-pair-proj-iso-TFAE1:*

assumes $f : X \rightarrow Y$ *monomorphism* f
shows $(\text{fibered-product-left-proj } X f f X) = (\text{fibered-product-right-proj } X f f X)$
<proof>

lemma *kern-pair-proj-iso-TFAE2:*

assumes $f : X \rightarrow Y$ $\text{fibered-product-left-proj } X f f X = \text{fibered-product-right-proj } X f f X$
shows *monomorphism* $f \wedge$ *isomorphism* $(\text{fibered-product-left-proj } X f f X) \wedge$ *isomorphism* $(\text{fibered-product-right-proj } X f f X)$
<proof>

lemma *kern-pair-proj-iso-TFAE3:*

assumes $f : X \rightarrow Y$
assumes *isomorphism* $(\text{fibered-product-left-proj } X f f X)$ *isomorphism* $(\text{fibered-product-right-proj } X f f X)$
shows $\text{fibered-product-left-proj } X f f X = \text{fibered-product-right-proj } X f f X$
<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 $(X \times_c g Y) \cong X \times_c Y$
<proof>

end

5 Truth Values and Characteristic Functions

theory *Truth*

imports *Equalizer*

begin

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

axiomatization*true-func* :: *cfunc* (t) **and***false-func* :: *cfunc* (f) **and***truth-value-set* :: *cset* (Ω)**where***true-func-type*[*type-rule*]: $t \in_c \Omega$ **and***false-func-type*[*type-rule*]: $f \in_c \Omega$ **and***true-false-distinct*: $t \neq f$ **and***true-false-only-truth-values*: $x \in_c \Omega \implies x = f \vee x = t$ **and***characteristic-function-exists*: $m : B \rightarrow X \implies \text{monomorphism } m \implies \exists! \chi. \text{is-pullback } B \mathbf{1} X \Omega (\beta_B) t m \chi$ **definition** *characteristic-func* :: *cfunc* \Rightarrow *cfunc* **where***characteristic-func* $m =$ $(\text{THE } \chi. \text{monomorphism } m \longrightarrow \text{is-pullback } (\text{domain } m) \mathbf{1} (\text{codomain } m) \Omega (\beta_{\text{domain } m}) t m \chi)$ **lemma** *characteristic-func-is-pullback*:**assumes** $m : B \rightarrow X$ *monomorphism* m **shows** *is-pullback* $B \mathbf{1} X \Omega (\beta_B) t m (\text{characteristic-func } m)$ $\langle \text{proof} \rangle$ **lemma** *characteristic-func-type*[*type-rule*]:**assumes** $m : B \rightarrow X$ *monomorphism* m **shows** *characteristic-func* $m : X \rightarrow \Omega$ $\langle \text{proof} \rangle$ **lemma** *characteristic-func-eq*:**assumes** $m : B \rightarrow X$ *monomorphism* m **shows** *characteristic-func* $m \circ_c m = t \circ_c \beta_B$ $\langle \text{proof} \rangle$ **lemma** *monomorphism-equalizes-char-func*:**assumes** *m-type*[*type-rule*]: $m : B \rightarrow X$ **and** *m-mono*[*type-rule*]: *monomorphism* m **shows** *equalizer* $B m (\text{characteristic-func } m) (t \circ_c \beta_X)$ $\langle \text{proof} \rangle$ **lemma** *characteristic-func-true-relative-member*:**assumes** $m : B \rightarrow X$ *monomorphism* m $x \in_c X$ **assumes** *characteristic-func-true*: *characteristic-func* $m \circ_c x = t$ **shows** $x \in_X (B, m)$ $\langle \text{proof} \rangle$ **lemma** *characteristic-func-false-not-relative-member*:**assumes** $m : B \rightarrow X$ *monomorphism* m $x \in_c X$ **assumes** *characteristic-func-true*: *characteristic-func* $m \circ_c x = f$ **shows** $\neg (x \in_X (B, m))$ $\langle \text{proof} \rangle$

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* $m \circ_c x = t$
 $\langle proof \rangle$

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* $m \circ_c x = f$
 $\langle proof \rangle$

The lemma below corresponds to Proposition 2.2.2 in Halvorson.

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

5.1 Equality Predicate

definition *eq-pred* :: *cset* \Rightarrow *cfunc* **where**
 $eq_pred\ X = (THE\ \chi. is_pullback\ X\ \mathbf{1}\ (X \times_c X)\ \Omega\ (\beta_X)\ t\ (diagonal\ X)\ \chi)$

lemma *eq-pred-pullback*: *is-pullback* $X\ \mathbf{1}\ (X \times_c X)\ \Omega\ (\beta_X)\ t\ (diagonal\ X)\ (eq_pred\ X)$
 $\langle proof \rangle$

lemma *eq-pred-type*[*type-rule*]:
 $eq_pred\ X : X \times_c X \rightarrow \Omega$
 $\langle proof \rangle$

lemma *eq-pred-square*: $eq_pred\ X \circ_c diagonal\ X = t \circ_c \beta_X$
 $\langle proof \rangle$

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

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

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

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

shows $eq\text{-pred } Y \circ_c (m \times_f m) = eq\text{-pred } X$
<proof>

lemma *eq-pred-true-extract-right*:

assumes $x \in_c X$
shows $eq\text{-pred } X \circ_c \langle x \circ_c \beta_X, id \ X \rangle \circ_c x = t$
<proof>

lemma *eq-pred-false-extract-right*:

assumes $x \in_c X \ y \in_c X \ x \neq y$
shows $eq\text{-pred } X \circ_c \langle x \circ_c \beta_X, id \ X \rangle \circ_c y = f$
<proof>

5.2 Properties of Monomorphisms and Epimorphisms

The lemma below corresponds to Exercise 2.2.3 in Halvorson.

lemma *regmono-is-mono*: *regular-monomorphism* $m \implies$ *monomorphism* m
<proof>

The lemma below corresponds to Proposition 2.2.4 in Halvorson.

lemma *mono-is-regmono*:
shows *monomorphism* $m \implies$ *regular-monomorphism* m
<proof>

The lemma below corresponds to Proposition 2.2.5 in Halvorson.

lemma *epi-mon-is-iso*:
assumes *epimorphism* f *monomorphism* f
shows *isomorphism* f
<proof>

The lemma below corresponds to Proposition 2.2.8 in Halvorson.

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

The lemma below corresponds to Proposition 2.2.9 in Halvorson.

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

The lemma below corresponds to Proposition 2.2.9b in Halvorson.

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

The lemma below corresponds to Proposition 2.2.9c in Halvorson.

lemma *pullback-of-mono-is-mono1*:

assumes $g: X \rightarrow Z$ *monomorphism* f *is-pullback* $A \ Y \ X \ Z \ q1 \ f \ q0 \ g$

shows *monomorphism* $q0$

<proof>

The lemma below corresponds to Proposition 2.2.9d in Halvorson.

lemma *pullback-of-mono-is-mono2*:

assumes $g: X \rightarrow Z$ *monomorphism* g *is-pullback* $A \ Y \ X \ Z \ q1 \ f \ q0 \ g$

shows *monomorphism* $q1$

<proof>

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

The definition below corresponds to Definition 2.2.6 in Halvorson.

definition *fiber* :: $cfunc \Rightarrow cfunc \Rightarrow cset \ (-^{-1}\{-\} \ [100,100]100)$ **where**

$$f^{-1}\{y\} = (f^{-1}(\mathbf{1})y)$$

definition *fiber-morphism* :: $cfunc \Rightarrow cfunc \Rightarrow cfunc$ **where**

$$fiber-morphism \ f \ y = left-cart-proj \ (domain \ f) \ \mathbf{1} \ \circ_c \ inverse-image-mapping \ f \ \mathbf{1} \ y$$

lemma *fiber-morphism-type*[*type-rule*]:

assumes $f: X \rightarrow Y \ y \in_c \ Y$

shows *fiber-morphism* $f \ y: f^{-1}\{y\} \rightarrow X$

<proof>

lemma *fiber-subset*:

assumes $f: X \rightarrow Y \ y \in_c \ Y$

shows $(f^{-1}\{y\}, fiber-morphism \ f \ y) \subseteq_c \ X$

<proof>

lemma *fiber-morphism-monomorphism*:

assumes $f: X \rightarrow Y \ y \in_c \ Y$

shows *monomorphism* $(fiber-morphism \ f \ y)$

<proof>

lemma *fiber-morphism-eq*:

assumes $f: X \rightarrow Y \ y \in_c \ Y$

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*]: $p: X \rightarrow Y$ **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 $f\text{-type}[type\text{-rule}]$: $f : X \rightarrow Y$
assumes $y\text{-type}[type\text{-rule}]$: $y : \mathbf{1} \rightarrow Y$
shows $f^{-1}\{y\} \cong X_{f \times_c y} \mathbf{1}$
 $\langle proof \rangle$

lemma *fib-prod-left-id-iso*:
assumes $g : Y \rightarrow X$
shows $(X_{id(X) \times_c g} Y) \cong Y$
 $\langle proof \rangle$

lemma *fib-prod-right-id-iso*:
assumes $f : X \rightarrow Y$
shows $(X_{f \times_c id(Y)} Y) \cong X$
 $\langle proof \rangle$

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

lemma *kernel-pair-connection*:
assumes $f\text{-type}[type\text{-rule}]$: $f : X \rightarrow Y$ **and** $g\text{-type}[type\text{-rule}]$: $g : X \rightarrow E$
assumes $g\text{-epi}$: epimorphism g
assumes $h\text{-}g\text{-eq}\text{-}f$: $h \circ_c g = f$
assumes $g\text{-eq}$: $g \circ_c \text{fibered-product-left-proj } X f f X = g \circ_c \text{fibered-product-right-proj } X f f X$
assumes $h\text{-type}[type\text{-rule}]$: $h : E \rightarrow Y$
shows $\exists! b. b : X_{f \times_c f} X \rightarrow E_{h \times_c h} E \wedge$
 $\text{fibered-product-left-proj } E h h E \circ_c b = g \circ_c \text{fibered-product-left-proj } X f f X \wedge$
 $\text{fibered-product-right-proj } E h h E \circ_c b = g \circ_c \text{fibered-product-right-proj } X f f X$
 \wedge
 $\text{epimorphism } b$
 $\langle proof \rangle$

6 Set Subtraction

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

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

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

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

shows *equalizer* $(Y \setminus (X, m)) m^c$ (*characteristic-func* m) $(f \circ_c \beta_Y)$
<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* $m \circ_c m^c = (f \circ_c \beta_Y) \circ_c m^c$
<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* $m \circ_c x = f$
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* $m \circ_c x = f$
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 $m : X \rightarrow Y$ *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 $m\text{-type}[type\text{-rule}]$: $m : A \rightarrow C$ **and** $m\text{-mono}[type\text{-rule}]$: *monomorphism*
 m
assumes $i\text{-type}[type\text{-rule}]$: $i : B \rightarrow A$ **and** $i\text{-iso}$: *isomorphism* i
shows $C \setminus (A, m) = C \setminus (B, m \circ_c i)$
 $\langle proof \rangle$

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

7 Graphs

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

The definition below corresponds to Definition 2.3.12 in Halvorson.

definition *graph* :: $cfunc \Rightarrow cset$ **where**
 $graph f = (SOME E. \exists m. equalizer E m (f \circ_c left\text{-cart}\text{-proj } (domain f) (codomain f)) (right\text{-cart}\text{-proj } (domain f) (codomain f)))$

lemma *graph-equalizer*:
 $\exists m. equalizer (graph f) m (f \circ_c left\text{-cart}\text{-proj } (domain f) (codomain f)) (right\text{-cart}\text{-proj } (domain f) (codomain f))$
 $\langle proof \rangle$

lemma *graph-equalizer2*:
assumes $f : X \rightarrow Y$
shows $\exists m. equalizer (graph f) m (f \circ_c left\text{-cart}\text{-proj } X Y) (right\text{-cart}\text{-proj } X Y)$
 $\langle proof \rangle$

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

lemma *graph-equalizer3*:
 $equalizer (graph f) (graph\text{-morph } f) (f \circ_c left\text{-cart}\text{-proj } (domain f) (codomain f)) (right\text{-cart}\text{-proj } (domain f) (codomain f))$
 $\langle proof \rangle$

lemma *graph-equalizer4*:
assumes $f : X \rightarrow Y$
shows $equalizer (graph f) (graph\text{-morph } f) (f \circ_c left\text{-cart}\text{-proj } X Y) (right\text{-cart}\text{-proj } X Y)$
 $\langle proof \rangle$

lemma *graph-subobject*:
assumes $f : X \rightarrow Y$
shows $(\text{graph } f, \text{graph-morph } f) \subseteq_c (X \times_c Y)$
 $\langle \text{proof} \rangle$

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

The lemma below corresponds to Exercise 2.3.13 in Halvorson.

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

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

The lemma below corresponds to Proposition 2.3.14 in Halvorson.

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

end

8 Equivalence Classes and Coequalizers

theory *Equivalence*
imports *Truth*
begin

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

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

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

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

definition *equiv-rel-on* :: *cset* \Rightarrow *cset* \times *cfunc* \Rightarrow *bool* **where**

equiv-rel-on *X R* \iff (*reflexive-on* *X R* \wedge *symmetric-on* *X R* \wedge *transitive-on* *X R*)

definition *const-on-rel* :: *cset* \Rightarrow *cset* \times *cfunc* \Rightarrow *cfunc* \Rightarrow *bool* **where**

const-on-rel *X R f* = ($\forall x y. x \in_c X \longrightarrow y \in_c X \longrightarrow \langle x, y \rangle \in_{X \times_c X} R \longrightarrow f \circ_c x = f \circ_c y$)

lemma *reflexive-def2*:

assumes *reflexive-Y*: *reflexive-on* *X* (*Y*, *m*)

assumes *x-type*: $x \in_c X$

shows $\exists y. y \in_c Y \wedge m \circ_c y = \langle x, x \rangle$

<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* \rightarrow *Y*

shows *equiv-rel-on* *X* (*X* $f \times_c f$ *X*, *fibred-product-morphism* *X f f* *X*)

<proof>

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

axiomatization

quotient-set :: *cset* \Rightarrow (*cset* \times *cfunc*) \Rightarrow *cset* (**infix** // 50) **and**

equiv-class :: *cset* \times *cfunc* \Rightarrow *cfunc* **and**

quotient-func :: *cfunc* \Rightarrow *cset* \times *cfunc* \Rightarrow *cfunc*

where

equiv-class-type[*type-rule*]: *equiv-rel-on* *X R* \implies *equiv-class* *R* : *X* \rightarrow *quotient-set* *X R* **and**

equiv-class-eq: $\text{equiv-rel-on } X R \implies \langle x, y \rangle \in_c X \times_c X \implies$
 $\langle x, y \rangle \in_{X \times_c X} R \iff \text{equiv-class } R \circ_c x = \text{equiv-class } R \circ_c y$ **and**
quotient-func-type[*type-rule*]:
 $\text{equiv-rel-on } X R \implies f : X \rightarrow Y \implies (\text{const-on-rel } X R f) \implies$
 $\text{quotient-func } f R : \text{quotient-set } X R \rightarrow Y$ **and**
quotient-func-eq: $\text{equiv-rel-on } X R \implies f : X \rightarrow Y \implies (\text{const-on-rel } X R f) \implies$
 $\text{quotient-func } f R \circ_c \text{equiv-class } R = f$ **and**
quotient-func-unique: $\text{equiv-rel-on } X R \implies f : X \rightarrow Y \implies (\text{const-on-rel } X R f)$
 \implies
 $h : \text{quotient-set } X R \rightarrow Y \implies h \circ_c \text{equiv-class } R = f \implies h = \text{quotient-func } f R$

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

8.1 Coequalizers

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

definition *coequalizer* :: $cset \Rightarrow cfunc \Rightarrow cfunc \Rightarrow cfunc \Rightarrow bool$ **where**
 $\text{coequalizer } E m f g \iff (\exists X Y. (f : Y \rightarrow X) \wedge (g : Y \rightarrow X) \wedge (m : X \rightarrow E)$
 $\wedge (m \circ_c f = m \circ_c g)$
 $\wedge (\forall h F. ((h : X \rightarrow F) \wedge (h \circ_c f = h \circ_c g)) \longrightarrow (\exists! k. (k : E \rightarrow F) \wedge k \circ_c m = h)))$

lemma *coequalizer-def2*:

assumes $f : Y \rightarrow X$ $g : Y \rightarrow X$ $m : X \rightarrow E$
shows $\text{coequalizer } E m f g \iff$
 $(m \circ_c f = m \circ_c g)$
 $\wedge (\forall h F. ((h : X \rightarrow F) \wedge (h \circ_c f = h \circ_c g)) \longrightarrow (\exists! k. (k : E \rightarrow F) \wedge k \circ_c m = h))$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Exercise 2.3.1 in Halvorson.

lemma *coequalizer-unique*:

assumes $\text{coequalizer } E m f g$ $\text{coequalizer } F n f g$
shows $E \cong F$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Exercise 2.3.2 in Halvorson.

lemma *coequalizer-is-epimorphism*:

$\text{coequalizer } E m f g \implies \text{epimorphism}(m)$
 $\langle \text{proof} \rangle$

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

assumes *equiv-rel-on* X (R, m)
shows *coequalizer* $(X // (R, m))$ (*equiv-class* (R, m))
 $(\text{left-cart-proj } X \ X \ \circ_c \ m)$ $(\text{right-cart-proj } X \ X \ \circ_c \ m)$
 $\langle \text{proof} \rangle$

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

8.2 Regular Epimorphisms

The definition below corresponds to Definition 2.3.4 in Halvorson.

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

The lemma below corresponds to Exercise 2.3.5 in Halvorson.

lemma *reg-epi-and-mono-is-iso*:
assumes $f : X \rightarrow Y$ *regular-epimorphism* f *monomorphism* f
shows *isomorphism* f
 $\langle \text{proof} \rangle$

The two lemmas below correspond to Proposition 2.3.6 in Halvorson.

lemma *epimorphism-coequalizer-kernel-pair*:
assumes $f : X \rightarrow Y$ *epimorphism* f
shows *coequalizer* $Y \ f$ (*fibred-product-left-proj* $X \ f \ f \ X$) (*fibred-product-right-proj* $X \ f \ f \ X$)
 $\langle \text{proof} \rangle$

lemma *epimorphisms-are-regular*:
assumes $f : X \rightarrow Y$ *epimorphism* f
shows *regular-epimorphism* f
 $\langle \text{proof} \rangle$

8.3 Epi-monic Factorization

lemma *epi-monic-factorization*:
assumes *f-type* $[\text{type-rule}]$: $f : X \rightarrow Y$
shows $\exists \ g \ m \ E. \ g : X \rightarrow E \wedge m : E \rightarrow Y$
 $\wedge \ \text{coequalizer } E \ g$ (*fibred-product-left-proj* $X \ f \ f \ X$) (*fibred-product-right-proj* $X \ f \ f \ X$)
 $\wedge \ \text{monomorphism } m \wedge f = m \circ_c g$
 $\wedge (\forall x. \ x : E \rightarrow Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
 $\langle \text{proof} \rangle$

lemma *epi-monic-factorization2*:
assumes *f-type* $[\text{type-rule}]$: $f : X \rightarrow Y$
shows $\exists \ g \ m \ E. \ g : X \rightarrow E \wedge m : E \rightarrow Y$
 $\wedge \ \text{epimorphism } g \wedge \ \text{monomorphism } m \wedge f = m \circ_c g$

$\wedge (\forall x. x : E \rightarrow Y \longrightarrow f = x \circ_c g \longrightarrow x = m)$
 $\langle \text{proof} \rangle$

8.3.1 Image of a Function

The definition below corresponds to Definition 2.3.7 in Halvorson.

definition *image-of* :: $cfunc \Rightarrow cset \Rightarrow cfunc \Rightarrow cset$ $([-]- [101,0,0]100)$ **where**
image-of $f A n = (SOME fA. \exists g m.$
 $g : A \rightarrow fA \wedge$
 $m : fA \rightarrow \text{codomain } f \wedge$
 $\text{coequalizer } fA g$ (*fibred-product-left-proj* $A (f \circ_c n) (f \circ_c n) A$) (*fibred-product-right-proj*
 $A (f \circ_c n) (f \circ_c n) A$) \wedge
 $\text{monomorphism } m \wedge f \circ_c n = m \circ_c g \wedge (\forall x. x : fA \rightarrow \text{codomain } f \longrightarrow f \circ_c n$
 $= x \circ_c g \longrightarrow x = m))$

lemma *image-of-def2*:

assumes $f : X \rightarrow Y n : A \rightarrow X$

shows $\exists g m.$

$g : A \rightarrow f(A)_n \wedge$

$m : f(A)_n \rightarrow Y \wedge$

$\text{coequalizer } (f(A)_n) g$ (*fibred-product-left-proj* $A (f \circ_c n) (f \circ_c n) A$) (*fibred-product-right-proj*
 $A (f \circ_c n) (f \circ_c n) A$) \wedge

$\text{monomorphism } m \wedge f \circ_c n = m \circ_c g \wedge (\forall x. x : f(A)_n \rightarrow Y \longrightarrow f \circ_c n = x$
 $\circ_c g \longrightarrow x = m)$

$\langle \text{proof} \rangle$

definition *image-restriction-mapping* :: $cfunc \Rightarrow cset \times cfunc \Rightarrow cfunc$ $([-] [101,0]100)$
where

image-restriction-mapping $f An = (SOME g. \exists m. g : \text{fst } An \rightarrow f(\text{fst } An)_{\text{snd } An}$
 $\wedge m : f(\text{fst } An)_{\text{snd } An} \rightarrow \text{codomain } f \wedge$

$\text{coequalizer } (f(\text{fst } An)_{\text{snd } An}) g$ (*fibred-product-left-proj* $(\text{fst } An) (f \circ_c \text{snd } An)$
 $(f \circ_c \text{snd } An) (\text{fst } An)$) (*fibred-product-right-proj* $(\text{fst } An) (f \circ_c \text{snd } An)$ $(f \circ_c \text{snd } An)$
 $(\text{fst } An)$) \wedge

$\text{monomorphism } m \wedge f \circ_c \text{snd } An = m \circ_c g \wedge (\forall x. x : f(\text{fst } An)_{\text{snd } An} \rightarrow$
 $\text{codomain } f \longrightarrow f \circ_c \text{snd } An = x \circ_c g \longrightarrow x = m))$

lemma *image-restriction-mapping-def2*:

assumes $f : X \rightarrow Y n : A \rightarrow X$

shows $\exists m. f \upharpoonright_{(A, n)} : A \rightarrow f(A)_n \wedge m : f(A)_n \rightarrow Y \wedge$

$\text{coequalizer } (f(A)_n) (f \upharpoonright_{(A, n)})$ (*fibred-product-left-proj* $A (f \circ_c n) (f \circ_c n) A$)
fibred-product-right-proj $A (f \circ_c n) (f \circ_c n) A$) \wedge

$\text{monomorphism } m \wedge f \circ_c n = m \circ_c (f \upharpoonright_{(A, n)}) \wedge (\forall x. x : f(A)_n \rightarrow Y \longrightarrow f \circ_c$
 $n = x \circ_c (f \upharpoonright_{(A, n)}) \longrightarrow x = m)$

$\langle \text{proof} \rangle$

definition *image-subobject-mapping* :: $cfunc \Rightarrow cset \Rightarrow cfunc \Rightarrow cfunc$ $([-(-)] \text{map}$
 $[101,0,0]100)$ **where**

$[f(A)_n] \text{map} = (THE m. f \upharpoonright_{(A, n)} : A \rightarrow f(A)_n \wedge m : f(A)_n \rightarrow \text{codomain } f \wedge$

$coequalizer (f \downarrow A)_n (f \downarrow_{(A, n)}) (fibered-product-left-proj A (f \circ_c n) (f \circ_c n) A)$
 $(fibered-product-right-proj A (f \circ_c n) (f \circ_c n) A) \wedge$
 $monomorphism m \wedge f \circ_c n = m \circ_c (f \downarrow_{(A, n)}) \wedge (\forall x. x : (f \downarrow A)_n \rightarrow codomain$
 $f \rightarrow f \circ_c n = x \circ_c (f \downarrow_{(A, n)}) \rightarrow x = m)$

lemma *image-subobject-mapping-def2*:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows $f \downarrow_{(A, n)} : A \rightarrow f \downarrow A)_n \wedge [f \downarrow A)_n]map : f \downarrow A)_n \rightarrow Y \wedge$

$coequalizer (f \downarrow A)_n (f \downarrow_{(A, n)}) (fibered-product-left-proj A (f \circ_c n) (f \circ_c n) A)$
 $(fibered-product-right-proj A (f \circ_c n) (f \circ_c n) A) \wedge$

$monomorphism ([f \downarrow A)_n]map) \wedge f \circ_c n = [f \downarrow A)_n]map \circ_c (f \downarrow_{(A, n)}) \wedge (\forall x. x :$
 $f \downarrow A)_n \rightarrow Y \rightarrow f \circ_c n = x \circ_c (f \downarrow_{(A, n)}) \rightarrow x = [f \downarrow A)_n]map)$

<proof>

lemma *image-rest-map-type*[type-rule]:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows $f \downarrow_{(A, n)} : A \rightarrow f \downarrow A)_n$

<proof>

lemma *image-rest-map-coequalizer*:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows $coequalizer (f \downarrow A)_n (f \downarrow_{(A, n)}) (fibered-product-left-proj A (f \circ_c n) (f \circ_c$
 $n) A) (fibered-product-right-proj A (f \circ_c n) (f \circ_c n) A)$

<proof>

lemma *image-rest-map-epi*:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows *epimorphism* $(f \downarrow_{(A, n)})$

<proof>

lemma *image-subobj-map-type*[type-rule]:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows $[f \downarrow A)_n]map : f \downarrow A)_n \rightarrow Y$

<proof>

lemma *image-subobj-map-mono*:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows *monomorphism* $([f \downarrow A)_n]map)$

<proof>

lemma *image-subobj-comp-image-rest*:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows $[f \downarrow A)_n]map \circ_c (f \downarrow_{(A, n)}) = f \circ_c n$

<proof>

lemma *image-subobj-map-unique*:

assumes $f : X \rightarrow Y \ n : A \rightarrow X$

shows $x : f \downarrow A)_n \rightarrow Y \implies f \circ_c n = x \circ_c (f \downarrow_{(A, n)}) \implies x = [f \downarrow A)_n]map$

<proof>

lemma *image-self*:

assumes $f : X \rightarrow Y$ **and** *monomorphism* f

assumes $a : A \rightarrow X$ **and** *monomorphism* a

shows $f \langle A \rangle_a \cong A$

<proof>

The lemma below corresponds to Proposition 2.3.8 in Halvorson.

lemma *image-smallest-subobject*:

assumes f -*type*[*type-rule*]: $f : X \rightarrow Y$ **and** a -*type*[*type-rule*]: $a : A \rightarrow X$

shows $(B, n) \subseteq_c Y \implies f$ *factorsthru* $n \implies (f \langle A \rangle_a, [f \langle A \rangle_a]map) \subseteq_Y (B, n)$

<proof>

lemma *images-iso*:

assumes f -*type*[*type-rule*]: $f : X \rightarrow Y$

assumes m -*type*[*type-rule*]: $m : Z \rightarrow X$ **and** n -*type*[*type-rule*]: $n : A \rightarrow Z$

shows $(f \circ_c m) \langle A \rangle_n \cong f \langle A \rangle_{m \circ_c n}$

<proof>

lemma *image-subset-conv*:

assumes f -*type*[*type-rule*]: $f : X \rightarrow Y$

assumes m -*type*[*type-rule*]: $m : Z \rightarrow X$ **and** n -*type*[*type-rule*]: $n : A \rightarrow Z$

shows $\exists i. ((f \circ_c m) \langle A \rangle_n, i) \subseteq_c B \implies \exists j. (f \langle A \rangle_{m \circ_c n}, j) \subseteq_c B$

<proof>

lemma *image-rel-subset-conv*:

assumes f -*type*[*type-rule*]: $f : X \rightarrow Y$

assumes m -*type*[*type-rule*]: $m : Z \rightarrow X$ **and** n -*type*[*type-rule*]: $n : A \rightarrow Z$

assumes *rel-sub1*: $((f \circ_c m) \langle A \rangle_n, [(f \circ_c m) \langle A \rangle_n]map) \subseteq_Y (B, b)$

shows $(f \langle A \rangle_{m \circ_c n}, [f \langle A \rangle_{m \circ_c n}]map) \subseteq_Y (B, b)$

<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*]: $f : X \rightarrow Y$

shows $((A, a) \subseteq_X (f^{-1} \langle B \rangle_m, [f^{-1} \langle B \rangle_m]map)) = ((f \langle A \rangle_a, [f \langle A \rangle_a]map) \subseteq_Y (B, m))$

<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*]: $f : X \rightarrow Y$

shows $(A, m) \subseteq_X (f^{-1} \langle f \langle A \rangle_m \rangle, [f^{-1} \langle f \langle A \rangle_m \rangle]map, [f^{-1} \langle f \langle A \rangle_m \rangle]map)$

<proof>

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

lemma *left-pair-subset*:

assumes $m : Y \rightarrow X \times_c X$ monomorphism m

shows $(Y \times_c Z, \text{distribute-right } X \ X \ Z \ \circ_c \ (m \times_f \text{id}_c \ Z)) \subseteq_c (X \times_c Z) \times_c (X \times_c Z)$

<proof>

lemma *right-pair-subset*:

assumes $m : Y \rightarrow X \times_c X$ monomorphism m

shows $(Z \times_c Y, \text{distribute-left } Z \ X \ X \ \circ_c \ (\text{id}_c \ Z \times_f \ m)) \subseteq_c (Z \times_c X) \times_c (Z \times_c X)$

<proof>

lemma *left-pair-reflexive*:

assumes reflexive-on X (Y, m)

shows reflexive-on $(X \times_c Z)$ $(Y \times_c Z, \text{distribute-right } X \ X \ Z \ \circ_c \ (m \times_f \text{id}_c \ Z))$

<proof>

lemma *right-pair-reflexive*:

assumes reflexive-on X (Y, m)

shows reflexive-on $(Z \times_c X)$ $(Z \times_c Y, \text{distribute-left } Z \ X \ X \ \circ_c \ (\text{id}_c \ Z \times_f \ m))$

<proof>

lemma *left-pair-symmetric*:

assumes symmetric-on X (Y, m)

shows symmetric-on $(X \times_c Z)$ $(Y \times_c Z, \text{distribute-right } X \ X \ Z \ \circ_c \ (m \times_f \text{id}_c \ Z))$

<proof>

lemma *right-pair-symmetric*:

assumes symmetric-on X (Y, m)

shows symmetric-on $(Z \times_c X)$ $(Z \times_c Y, \text{distribute-left } Z \ X \ X \ \circ_c \ (\text{id}_c \ Z \times_f \ m))$

<proof>

lemma *left-pair-transitive*:

assumes transitive-on X (Y, m)

shows transitive-on $(X \times_c Z)$ $(Y \times_c Z, \text{distribute-right } X \ X \ Z \ \circ_c \ (m \times_f \text{id}_c \ Z))$

<proof>

lemma *right-pair-transitive*:

assumes transitive-on X (Y, m)

shows transitive-on $(Z \times_c X)$ $(Z \times_c Y, \text{distribute-left } Z \ X \ X \ \circ_c \ (\text{id}_c \ Z \times_f \ m))$

<proof>

lemma *left-pair-equiv-rel*:

assumes equiv-rel-on X (Y, m)

shows equiv-rel-on $(X \times_c Z)$ $(Y \times_c Z, \text{distribute-right } X \ X \ Z \ \circ_c \ (m \times_f \text{id}_c \ Z))$

<proof>

lemma *right-pair-equiv-rel:*

assumes *equiv-rel-on* $X (Y, m)$

shows *equiv-rel-on* $(Z \times_c X) (Z \times_c Y, \text{distribute-left } Z X X \circ_c (\text{id } Z \times_f m))$

<proof>

end

9 Coproducts

theory *Coproduct*

imports *Equivalence*

begin

hide-const *case-bool*

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

axiomatization

coprod :: *cset* \Rightarrow *cset* \Rightarrow *cset* (**infixr** \coprod 65) **and**

left-coproj :: *cset* \Rightarrow *cset* \Rightarrow *cfunc* **and**

right-coproj :: *cset* \Rightarrow *cset* \Rightarrow *cfunc* **and**

cfunc-coprod :: *cfunc* \Rightarrow *cfunc* \Rightarrow *cfunc* (**infixr** \amalg 65)

where

left-proj-type[*type-rule*]: *left-coproj* $X Y : X \rightarrow X \coprod Y$ **and**

right-proj-type[*type-rule*]: *right-coproj* $X Y : Y \rightarrow X \coprod Y$ **and**

cfunc-coprod-type[*type-rule*]: $f : X \rightarrow Z \Longrightarrow g : Y \rightarrow Z \Longrightarrow f \amalg g : X \coprod Y \rightarrow Z$

and

left-coproj-cfunc-coprod: $f : X \rightarrow Z \Longrightarrow g : Y \rightarrow Z \Longrightarrow f \amalg g \circ_c (\text{left-coproj } X Y) = f$ **and**

right-coproj-cfunc-coprod: $f : X \rightarrow Z \Longrightarrow g : Y \rightarrow Z \Longrightarrow f \amalg g \circ_c (\text{right-coproj } X Y) = g$ **and**

cfunc-coprod-unique: $f : X \rightarrow Z \Longrightarrow g : Y \rightarrow Z \Longrightarrow h : X \coprod Y \rightarrow Z \Longrightarrow$

$h \circ_c \text{left-coproj } X Y = f \Longrightarrow h \circ_c \text{right-coproj } X Y = g \Longrightarrow h = f \amalg g$

definition *is-coprod* :: *cset* \Rightarrow *cfunc* \Rightarrow *cfunc* \Rightarrow *cset* \Rightarrow *cset* \Rightarrow *bool* **where**

is-coprod $W i_0 i_1 X Y \longleftrightarrow$

$(i_0 : X \rightarrow W \wedge i_1 : Y \rightarrow W \wedge$

$(\forall f g Z. (f : X \rightarrow Z \wedge g : Y \rightarrow Z) \longrightarrow$

$(\exists h. h : W \rightarrow Z \wedge h \circ_c i_0 = f \wedge h \circ_c i_1 = g \wedge$

$(\forall h2. (h2 : W \rightarrow Z \wedge h2 \circ_c i_0 = f \wedge h2 \circ_c i_1 = g) \longrightarrow h2 = h))))$

lemma *is-coprod-def2*:

assumes $i_0 : X \rightarrow W i_1 : Y \rightarrow W$

shows *is-coprod* $W i_0 i_1 X Y \longleftrightarrow$

$(\forall f g Z. (f : X \rightarrow Z \wedge g : Y \rightarrow Z) \longrightarrow$

$(\exists h. h : W \rightarrow Z \wedge h \circ_c i_0 = f \wedge h \circ_c i_1 = g \wedge$

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

abbreviation *is-coprod-triple* :: *cset* × *cfunc* × *cfunc* ⇒ *cset* ⇒ *cset* ⇒ *bool*
where

is-coprod-triple *Wi X Y* ≡ *is-coprod* (*fst Wi*) (*fst (snd Wi)*) (*snd (snd Wi)*) *X Y*

lemma *canonical-coprod-is-coprod*:

is-coprod (*X* \coprod *Y*) (*left-coproj X Y*) (*right-coproj X Y*) *X Y*
 ⟨proof⟩

The lemma below is dual to Proposition 2.1.8 in Halvorson.

lemma *coprods-isomorphic*:

assumes *W-coprod*: *is-coprod-triple* (*W*, *i*₀, *i*₁) *X Y*
assumes *W'-coprod*: *is-coprod-triple* (*W'*, *i'*₀, *i'*₁) *X Y*
shows $\exists g. g : W \rightarrow W' \wedge \text{isomorphism } g \wedge g \circ_c i_0 = i'_0 \wedge g \circ_c i_1 = i'_1$
 ⟨proof⟩

9.1 Coproduct Function Properties

lemma *cfunc-coprod-comp*:

assumes *a* : *Y* → *Z* *b* : *X* → *Y* *c* : *W* → *Y*
shows (*a* ∘_c *b*) \amalg (*a* ∘_c *c*) = *a* ∘_c (*b* \amalg *c*)
 ⟨proof⟩

lemma *id-coprod*:

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

The lemma below corresponds to Proposition 2.4.1 in Halvorson.

lemma *coproducts-disjoint*:

$x \in_c X \implies y \in_c Y \implies (\text{left-coproj } X Y) \circ_c x \neq (\text{right-coproj } X Y) \circ_c y$
 ⟨proof⟩

The lemma below corresponds to Proposition 2.4.2 in Halvorson.

lemma *left-coproj-are-monomorphisms*:

monomorphism(*left-coproj X Y*)
 ⟨proof⟩

lemma *right-coproj-are-monomorphisms*:

monomorphism(*right-coproj X Y*)
 ⟨proof⟩

The lemma below corresponds to Exercise 2.4.3 in Halvorson.

lemma *coprojs-jointly-surj*:

assumes *z-type*[*type-rule*]: *z* ∈_c *X* \amalg *Y*
shows ($\exists x. (x \in_c X \wedge z = (\text{left-coproj } X Y) \circ_c x)$)
 \vee ($\exists y. (y \in_c Y \wedge z = (\text{right-coproj } X Y) \circ_c y)$)
 ⟨proof⟩

lemma *maps-into-1u1*:

assumes *x-type*: $x \in_c (\mathbf{1} \amalg \mathbf{1})$

shows $(x = \text{left-coproj } \mathbf{1} \ \mathbf{1}) \vee (x = \text{right-coproj } \mathbf{1} \ \mathbf{1})$

<proof>

lemma *coprod-preserves-left-epi*:

assumes $f: X \rightarrow Z$ $g: Y \rightarrow Z$

assumes *surjective*(f)

shows *surjective*($f \amalg g$)

<proof>

lemma *coprod-preserves-right-epi*:

assumes $f: X \rightarrow Z$ $g: Y \rightarrow Z$

assumes *surjective*(g)

shows *surjective*($f \amalg g$)

<proof>

lemma *coprod-eq*:

assumes $a: X \amalg Y \rightarrow Z$ $b: X \amalg Y \rightarrow Z$

shows $a = b \iff$

$(a \circ_c \text{left-coproj } X \ Y = b \circ_c \text{left-coproj } X \ Y$

$\wedge a \circ_c \text{right-coproj } X \ Y = b \circ_c \text{right-coproj } X \ Y)$

<proof>

lemma *coprod-eqI*:

assumes $a: X \amalg Y \rightarrow Z$ $b: X \amalg Y \rightarrow Z$

assumes $(a \circ_c \text{left-coproj } X \ Y = b \circ_c \text{left-coproj } X \ Y$

$\wedge a \circ_c \text{right-coproj } X \ Y = b \circ_c \text{right-coproj } X \ Y)$

shows $a = b$

<proof>

lemma *coprod-eq2*:

assumes $a: X \rightarrow Z$ $b: Y \rightarrow Z$ $c: X \rightarrow Z$ $d: Y \rightarrow Z$

shows $(a \amalg b) = (c \amalg d) \iff (a = c \wedge b = d)$

<proof>

lemma *coprod-decomp*:

assumes $a: X \amalg Y \rightarrow A$

shows $\exists x \ y. a = (x \amalg y) \wedge x: X \rightarrow A \wedge y: Y \rightarrow A$

<proof>

The lemma below corresponds to Proposition 2.4.4 in Halvorson.

lemma *truth-value-set-iso-1u1*:

isomorphism($\text{tIII}f$)

<proof>

9.1.1 Equality Predicate with Coproduct Properties

lemma *eq-pred-left-coproj*:

assumes $u\text{-type}[type\text{-rule}]$: $u \in_c X \amalg Y$ **and** $x\text{-type}[type\text{-rule}]$: $x \in_c X$
shows $eq\text{-pred} (X \amalg Y) \circ_c \langle u, left\text{-coproj} X Y \circ_c x \rangle = ((eq\text{-pred} X \circ_c \langle id X, x \circ_c \beta_X \rangle) \amalg (f \circ_c \beta_Y)) \circ_c u$
 $\langle proof \rangle$

lemma $eq\text{-pred-right-coproj}$:

assumes $u\text{-type}[type\text{-rule}]$: $u \in_c X \amalg Y$ **and** $y\text{-type}[type\text{-rule}]$: $y \in_c Y$
shows $eq\text{-pred} (X \amalg Y) \circ_c \langle u, right\text{-coproj} X Y \circ_c y \rangle = ((f \circ_c \beta_X) \amalg (eq\text{-pred} Y \circ_c \langle id Y, y \circ_c \beta_Y \rangle)) \circ_c u$
 $\langle proof \rangle$

9.2 Bowtie Product

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

lemma $cfunc\text{-bowtie-prod-def2}$:

assumes $f : X \rightarrow Y$ $g : V \rightarrow W$
shows $f \bowtie_f g = (left\text{-coproj} Y W \circ_c f) \amalg (right\text{-coproj} Y W \circ_c g)$
 $\langle proof \rangle$

lemma $cfunc\text{-bowtie-prod-type}[type\text{-rule}]$:

$f : X \rightarrow Y \Longrightarrow g : V \rightarrow W \Longrightarrow f \bowtie_f g : X \amalg V \rightarrow Y \amalg W$
 $\langle proof \rangle$

lemma $left\text{-coproj-cfunc-bowtie-prod}$:

$f : X \rightarrow Y \Longrightarrow g : V \rightarrow W \Longrightarrow (f \bowtie_f g) \circ_c left\text{-coproj} X V = left\text{-coproj} Y W \circ_c f$
 $\langle proof \rangle$

lemma $right\text{-coproj-cfunc-bowtie-prod}$:

$f : X \rightarrow Y \Longrightarrow g : V \rightarrow W \Longrightarrow (f \bowtie_f g) \circ_c right\text{-coproj} X V = right\text{-coproj} Y W \circ_c g$
 $\langle proof \rangle$

lemma $cfunc\text{-bowtie-prod-unique}$: $f : X \rightarrow Y \Longrightarrow g : V \rightarrow W \Longrightarrow h : X \amalg V \rightarrow Y \amalg W \Longrightarrow$

$h \circ_c left\text{-coproj} X V = left\text{-coproj} Y W \circ_c f \Longrightarrow$
 $h \circ_c right\text{-coproj} X V = right\text{-coproj} Y W \circ_c g \Longrightarrow h = f \bowtie_f g$
 $\langle proof \rangle$

The lemma below is dual to Proposition 2.1.11 in Halvorson.

lemma $identity\text{-distributes-across-composition-dual}$:

assumes $f\text{-type}$: $f : A \rightarrow B$ **and** $g\text{-type}$: $g : B \rightarrow C$
shows $(g \circ_c f) \bowtie_f id X = (g \bowtie_f id X) \circ_c (f \bowtie_f id X)$
 $\langle proof \rangle$

lemma $coproduct-of-beta$:

$\beta_X \amalg \beta_Y = \beta_{X \amalg Y}$

<proof>

lemma *cfunc-bowtieprod-comp-cfunc-coprod*:

assumes *a-type*: $a : Y \rightarrow Z$ **and** *b-type*: $b : W \rightarrow Z$

assumes *f-type*: $f : X \rightarrow Y$ **and** *g-type*: $g : V \rightarrow W$

shows $(a \amalg b) \circ_c (f \bowtie_f g) = (a \circ_c f) \amalg (b \circ_c g)$

<proof>

lemma *id-bowtie-prod*: $id(X) \bowtie_f id(Y) = id(X \amalg Y)$

<proof>

lemma *cfunc-bowtie-prod-comp-cfunc-bowtie-prod*:

assumes $f : X \rightarrow Y$ $g : V \rightarrow W$ $x : Y \rightarrow S$ $y : W \rightarrow T$

shows $(x \bowtie_f y) \circ_c (f \bowtie_f g) = (x \circ_c f) \bowtie_f (y \circ_c g)$

<proof>

lemma *cfunc-bowtieprod-epi*:

assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$ **and** *g-type*[*type-rule*]: $g : V \rightarrow W$

assumes *f-epi*: *epimorphism* f **and** *g-epi*: *epimorphism* g

shows *epimorphism* $(f \bowtie_f g)$

<proof>

lemma *cfunc-bowtieprod-inj*:

assumes *type-assms*: $f : X \rightarrow Y$ $g : V \rightarrow W$

assumes *f-epi*: *injective* f **and** *g-epi*: *injective* g

shows *injective* $(f \bowtie_f g)$

<proof>

lemma *cfunc-bowtieprod-inj-converse*:

assumes *type-assms*: $f : X \rightarrow Y$ $g : Z \rightarrow W$

assumes *inj-f-bowtie-g*: *injective* $(f \bowtie_f g)$

shows *injective* $f \wedge$ *injective* g

<proof>

lemma *cfunc-bowtieprod-iso*:

assumes *type-assms*: $f : X \rightarrow Y$ $g : V \rightarrow W$

assumes *f-iso*: *isomorphism* f **and** *g-iso*: *isomorphism* g

shows *isomorphism* $(f \bowtie_f g)$

<proof>

lemma *cfunc-bowtieprod-surj-converse*:

assumes *type-assms*: $f : X \rightarrow Y$ $g : Z \rightarrow W$

assumes *inj-f-bowtie-g*: *surjective* $(f \bowtie_f g)$

shows *surjective* $f \wedge$ *surjective* g

<proof>

9.3 Boolean Cases

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

$$\begin{aligned} \text{case-bool} &= (\text{THE } f. f : \Omega \rightarrow (\mathbf{1} \amalg \mathbf{1}) \wedge \\ &(\text{t } \amalg \text{f}) \circ_c f = \text{id } \Omega \wedge f \circ_c (\text{t } \amalg \text{f}) = \text{id } (\mathbf{1} \amalg \mathbf{1})) \end{aligned}$$

lemma *case-bool-def2*:

$$\begin{aligned} \text{case-bool} &: \Omega \rightarrow (\mathbf{1} \amalg \mathbf{1}) \wedge \\ &(\text{t } \amalg \text{f}) \circ_c \text{case-bool} = \text{id } \Omega \wedge \text{case-bool} \circ_c (\text{t } \amalg \text{f}) = \text{id } (\mathbf{1} \amalg \mathbf{1}) \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *case-bool-type[type-rule]*:

$$\begin{aligned} \text{case-bool} &: \Omega \rightarrow \mathbf{1} \amalg \mathbf{1} \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *case-bool-true-coproduct-false*:

$$\begin{aligned} \text{case-bool} \circ_c (\text{t } \amalg \text{f}) &= \text{id } (\mathbf{1} \amalg \mathbf{1}) \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *true-coproduct-false-case-bool*:

$$\begin{aligned} (\text{t } \amalg \text{f}) \circ_c \text{case-bool} &= \text{id } \Omega \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *case-bool-iso*:

$$\begin{aligned} &\text{isomorphism case-bool} \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *case-bool-true-and-false*:

$$\begin{aligned} (\text{case-bool} \circ_c \text{t} = \text{left-coproj } \mathbf{1} \ \mathbf{1}) \wedge (\text{case-bool} \circ_c \text{f} = \text{right-coproj } \mathbf{1} \ \mathbf{1}) \\ \langle \text{proof} \rangle \end{aligned}$$

lemma *case-bool-true*:

$$\begin{aligned} \text{case-bool} \circ_c \text{t} &= \text{left-coproj } \mathbf{1} \ \mathbf{1} \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *case-bool-false*:

$$\begin{aligned} \text{case-bool} \circ_c \text{f} &= \text{right-coproj } \mathbf{1} \ \mathbf{1} \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *coproduct-case-bool-true*:

$$\begin{aligned} &\text{assumes } x1 \in_c X \\ &\text{assumes } x2 \in_c X \\ &\text{shows } (x1 \amalg x2 \circ_c \text{case-bool}) \circ_c \text{t} = x1 \\ &\langle \text{proof} \rangle \end{aligned}$$

lemma *coproduct-case-bool-false*:

$$\begin{aligned} &\text{assumes } x1 \in_c X \\ &\text{assumes } x2 \in_c X \\ &\text{shows } (x1 \amalg x2 \circ_c \text{case-bool}) \circ_c \text{f} = x2 \\ &\langle \text{proof} \rangle \end{aligned}$$

9.4 Distribution of Products over Coproducts

9.4.1 Factor Product over Coproduct on Left

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

lemma *factor-prod-coprod-left-type*[*type-rule*]:
factor-prod-coprod-left *A B C* : (*A* \times_c *B*) \amalg (*A* \times_c *C*) \rightarrow *A* \times_c (*B* \amalg *C*)
 <proof>

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

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

lemma *factor-prod-coprod-left-mono*:
monomorphism (factor-prod-coprod-left *A B C)*
 <proof>

lemma *factor-prod-coprod-left-epi*:
epimorphism (factor-prod-coprod-left *A B C)*
 <proof>

lemma *dist-prod-coprod-iso*:
isomorphism(factor-prod-coprod-left *A B C)*
 <proof>

The lemma below corresponds to Proposition 2.5.10 in Halvorson.

lemma *prod-distribute-coprod*:
A \times_c (*X* \amalg *Y*) \cong (*A* \times_c *X*) \amalg (*A* \times_c *Y*)
 <proof>

9.4.2 Distribute Product over Coproduct on Left

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

lemma *dist-prod-coprod-left-def2*:

shows $dist\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C : A \times_c (B \amalg C) \rightarrow (A \times_c B) \amalg (A \times_c C)$
 $\wedge dist\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C \circ_c factor\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C = id\ ((A \times_c B) \amalg (A \times_c C))$
 $\wedge factor\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C \circ_c dist\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C = id\ (A \times_c (B \amalg C))$
 $\langle proof \rangle$

lemma $dist\text{-}prod\text{-}coprod\text{-}left\text{-}type[type\text{-}rule]$:
 $dist\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C : A \times_c (B \amalg C) \rightarrow (A \times_c B) \amalg (A \times_c C)$
 $\langle proof \rangle$

lemma $dist\text{-}factor\text{-}prod\text{-}coprod\text{-}left$:
 $dist\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C \circ_c factor\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C = id\ ((A \times_c B) \amalg (A \times_c C))$
 $\langle proof \rangle$

lemma $factor\text{-}dist\text{-}prod\text{-}coprod\text{-}left$:
 $factor\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C \circ_c dist\text{-}prod\text{-}coprod\text{-}left\ A\ B\ C = id\ (A \times_c (B \amalg C))$
 $\langle proof \rangle$

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

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

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

9.4.3 Factor Product over Coproduct on Right

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

lemma $factor\text{-}prod\text{-}coprod\text{-}right\text{-}type[type\text{-}rule]$:
 $factor\text{-}prod\text{-}coprod\text{-}right\ A\ B\ C : (A \times_c C) \amalg (B \times_c C) \rightarrow (A \amalg B) \times_c C$
 $\langle proof \rangle$

lemma $factor\text{-}prod\text{-}coprod\text{-}right\text{-}ap\text{-}left$:
assumes $a \in_c A\ c \in_c C$

shows $\text{factor-prod-coproduct-right } A B C \circ_c (\text{left-coproj } (A \times_c C) (B \times_c C) \circ_c \langle a, c \rangle) = \langle \text{left-coproj } A B \circ_c a, c \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{factor-prod-coproduct-right-ap-right}$:

assumes $b \in_c B \ c \in_c C$

shows $\text{factor-prod-coproduct-right } A B C \circ_c \text{right-coproj } (A \times_c C) (B \times_c C) \circ_c \langle b, c \rangle = \langle \text{right-coproj } A B \circ_c b, c \rangle$
 $\langle \text{proof} \rangle$

9.4.4 Distribute Product over Coproduct on Right

definition $\text{dist-prod-coproduct-right} :: \text{cset} \Rightarrow \text{cset} \Rightarrow \text{cset} \Rightarrow \text{cfunc}$ **where**

$\text{dist-prod-coproduct-right } A B C = (\text{swap } C A \bowtie_f \text{swap } C B) \circ_c \text{dist-prod-coproduct-left } C A B \circ_c \text{swap } (A \coprod B) C$

lemma $\text{dist-prod-coproduct-right-type}$ [type-rule]:

$\text{dist-prod-coproduct-right } A B C : (A \coprod B) \times_c C \rightarrow (A \times_c C) \coprod (B \times_c C)$
 $\langle \text{proof} \rangle$

lemma $\text{dist-prod-coproduct-right-ap-left}$:

assumes $a \in_c A \ c \in_c C$

shows $\text{dist-prod-coproduct-right } A B C \circ_c \langle \text{left-coproj } A B \circ_c a, c \rangle = \text{left-coproj } (A \times_c C) (B \times_c C) \circ_c \langle a, c \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{dist-prod-coproduct-right-ap-right}$:

assumes $b \in_c B \ c \in_c C$

shows $\text{dist-prod-coproduct-right } A B C \circ_c \langle \text{right-coproj } A B \circ_c b, c \rangle = \text{right-coproj } (A \times_c C) (B \times_c C) \circ_c \langle b, c \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{dist-prod-coproduct-right-left-coproj}$:

$\text{dist-prod-coproduct-right } X Y H \circ_c (\text{left-coproj } X Y \times_f \text{id } H) = \text{left-coproj } (X \times_c H) (Y \times_c H)$
 $\langle \text{proof} \rangle$

lemma $\text{dist-prod-coproduct-right-right-coproj}$:

$\text{dist-prod-coproduct-right } X Y H \circ_c (\text{right-coproj } X Y \times_f \text{id } H) = \text{right-coproj } (X \times_c H) (Y \times_c H)$
 $\langle \text{proof} \rangle$

lemma $\text{factor-dist-prod-coproduct-right}$:

$\text{factor-prod-coproduct-right } A B C \circ_c \text{dist-prod-coproduct-right } A B C = \text{id } ((A \coprod B) \times_c C)$
 $\langle \text{proof} \rangle$

lemma $\text{dist-factor-prod-coproduct-right}$:

$\text{dist-prod-coproduct-right } A B C \circ_c \text{factor-prod-coproduct-right } A B C = \text{id } (A \times_c C)$

$\coprod (B \times_c C)$
 ⟨proof⟩

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

9.5 Casting between Sets

9.5.1 Going from a Set or its Complement to the Superset

This subsection corresponds to Proposition 2.4.5 in Halvorson.

definition *into-super* :: *cfunc* \Rightarrow *cfunc* **where**
into-super m = m \coprod m^c

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

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

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

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

9.5.2 Going from a Set to a Subset or its Complement

definition *try-cast* :: *cfunc* \Rightarrow *cfunc* **where**
try-cast m = (THE m'. m' : *codomain* m \rightarrow *domain* m \coprod ((*codomain* m) \setminus ((*domain* m),m))
 \wedge m' \circ_c *into-super* m = id (*domain* m \coprod (*codomain* m \setminus ((*domain* m),m)))
 \wedge *into-super* m \circ_c m' = id (*codomain* m))

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

$\langle proof \rangle$

lemma *try-cast-type*[*type-rule*]:

assumes *monomorphism* $m : X \rightarrow Y$

shows *try-cast* $m : Y \rightarrow X \coprod (Y \setminus (X, m))$

$\langle proof \rangle$

lemma *try-cast-into-super*:

assumes *monomorphism* $m : X \rightarrow Y$

shows *try-cast* $m \circ_c \text{into-super } m = \text{id } (X \coprod (Y \setminus (X, m)))$

$\langle proof \rangle$

lemma *into-super-try-cast*:

assumes *monomorphism* $m : X \rightarrow Y$

shows *into-super* $m \circ_c \text{try-cast } m = \text{id } Y$

$\langle proof \rangle$

lemma *try-cast-in-X*:

assumes *m-type: monomorphism* $m : X \rightarrow Y$

assumes *y-in-X*: $y \in_Y (X, m)$

shows $\exists x. x \in_c X \wedge \text{try-cast } m \circ_c y = \text{left-coproj } X (Y \setminus (X, m)) \circ_c x$

$\langle proof \rangle$

lemma *try-cast-not-in-X*:

assumes *m-type: monomorphism* $m : X \rightarrow Y$

assumes *y-in-X*: $\neg y \in_Y (X, m)$ **and** *y-type*: $y \in_c Y$

shows $\exists x. x \in_c Y \setminus (X, m) \wedge \text{try-cast } m \circ_c y = \text{right-coproj } X (Y \setminus (X, m)) \circ_c x$

$\langle proof \rangle$

lemma *try-cast-m-m*:

assumes *m-type: monomorphism* $m : X \rightarrow Y$

shows $(\text{try-cast } m) \circ_c m = \text{left-coproj } X (Y \setminus (X, m))$

$\langle proof \rangle$

lemma *try-cast-m-m'*:

assumes *m-type: monomorphism* $m : X \rightarrow Y$

shows $(\text{try-cast } m) \circ_c m^c = \text{right-coproj } X (Y \setminus (X, m))$

$\langle proof \rangle$

lemma *try-cast-mono*:

assumes *m-type: monomorphism* $m : X \rightarrow Y$

shows *monomorphism*(*try-cast* m)

$\langle proof \rangle$

9.6 Cases

definition *cases* :: *cfunc* \Rightarrow *cfunc* **where**

$\text{cases}(f) = ((\text{right-cart-proj } \mathbf{1} (\text{domain } f)) \bowtie_f (\text{right-cart-proj } \mathbf{1} (\text{domain } f))) \circ_c$

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

lemma *cases-def2*:

assumes $f : X \rightarrow \Omega$

shows $\text{cases}(f) = ((\text{right-cart-proj } \mathbf{1} \ X) \bowtie_f (\text{right-cart-proj } \mathbf{1} \ X)) \circ_c (\text{dist-prod-coproduct-right } \mathbf{1} \ \mathbf{1} \ X) \circ_c \langle \text{case-bool } \circ_c f, \text{id } X \rangle$
 $\langle \text{proof} \rangle$

lemma *cases-type*[*type-rule*]:

assumes $f : X \rightarrow \Omega$

shows $\text{cases}(f) : X \rightarrow X \coprod X$

$\langle \text{proof} \rangle$

lemma *true-case*:

assumes $x\text{-type}[type\text{-rule}] : x \in_c X$

assumes $f\text{-type}[type\text{-rule}] : f : X \rightarrow \Omega$

assumes *true-case*: $f \circ_c x = \text{t}$

shows $\text{cases } f \circ_c x = \text{left-coproj } X \ X \circ_c x$

$\langle \text{proof} \rangle$

lemma *false-case*:

assumes $x\text{-type}[type\text{-rule}] : x \in_c X$

assumes $f\text{-type}[type\text{-rule}] : f : X \rightarrow \Omega$

assumes *false-case*: $f \circ_c x = \text{f}$

shows $\text{cases } f \circ_c x = \text{right-coproj } X \ X \circ_c x$

$\langle \text{proof} \rangle$

9.7 Coproduct Set Properties

lemma *coproduct-commutes*:

$A \coprod B \cong B \coprod A$

$\langle \text{proof} \rangle$

lemma *coproduct-associates*:

$A \coprod (B \coprod C) \cong (A \coprod B) \coprod C$

$\langle \text{proof} \rangle$

The lemma below corresponds to Proposition 2.5.10.

lemma *product-distribute-over-coproduct-left*:

$A \times_c (X \coprod Y) \cong (A \times_c X) \coprod (A \times_c Y)$

$\langle \text{proof} \rangle$

lemma *prod-pres-iso*:

assumes $A \cong C \ B \cong D$

shows $A \times_c B \cong C \times_c D$

$\langle \text{proof} \rangle$

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:*

$$\Omega \cong \mathbf{1} \coprod \mathbf{1}$$

<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

10 Axiom of Choice

theory *Axiom-Of-Choice*

imports *Coproduct*

begin

The two definitions below correspond to Definition 2.7.1 in Halvorson.

definition *section-of* :: *cfunc* \Rightarrow *cfunc* \Rightarrow *bool* (**infix** *sectionof* 90)

where *s sectionof f* $\longleftrightarrow s : \text{codomain } f \rightarrow \text{domain } f \wedge f \circ_c s = \text{id } (\text{codomain } f)$

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

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

lemma *split-epimorphism-def2:*

assumes *f-type:* $f : X \rightarrow Y$

assumes *f-split-epic:* *split-epimorphism f*

shows $\exists s. (f \circ_c s = \text{id } Y) \wedge (s : Y \rightarrow X)$

<proof>

lemma *sections-define-splits:*

assumes *s sectionof f*

assumes $s : Y \rightarrow X$

shows $f : X \rightarrow Y \wedge \text{split-epimorphism}(f)$

<proof>

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

axiomatization

where

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

lemma *epis-give-monos*:

assumes *f-type*: $f : X \rightarrow Y$

assumes *f-epi*: *epimorphism* f

shows $\exists g . g : Y \rightarrow X \wedge \text{monomorphism } g \wedge f \circ_c g = \text{id } Y$

<proof>

corollary *epis-are-split*:

assumes *f-type*: $f : X \rightarrow Y$

assumes *f-epi*: *epimorphism* f

shows *split-epimorphism* f

<proof>

The lemma below corresponds to Proposition 2.6.8 in Halvorson.

lemma *monos-give-epis*:

assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$

assumes *f-mono*: *monomorphism* f

assumes *X-nonempty*: *nonempty* X

shows $\exists g . g : Y \rightarrow X \wedge \text{epimorphism } g \wedge g \circ_c f = \text{id } X$

<proof>

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

lemma *split-epis-are-regular*:

assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$

assumes *split-epimorphism* f

shows *regular-epimorphism* f

<proof>

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

lemma *sections-are-regular-monos*:

assumes *s-type*: $s : Y \rightarrow X$

assumes *s sectionof* f

shows *regular-monomorphism* s

<proof>

end

11 Empty Set and Initial Objects

theory *Initial*

imports *Coproduct*

begin

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

axiomatization

initial-func :: *cset* \Rightarrow *cfunc* (α - 100) **and**
emptyset :: *cset* (\emptyset)

where

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

definition *initial-object* :: *cset* \Rightarrow *bool* **where**

initial-object(X) $\longleftrightarrow (\forall Y. \exists! f. f : X \rightarrow Y)$

lemma *emptyset-is-initial*:

initial-object(\emptyset)
<proof>

lemma *initial-iso-empty*:

assumes *initial-object*(X)
shows $X \cong \emptyset$
<proof>

The lemma below corresponds to Exercise 2.4.6 in Halvorson.

lemma *coproduct-with-empty*:

shows $X \coprod \emptyset \cong X$
<proof>

The lemma below corresponds to Proposition 2.4.7 in Halvorson.

lemma *function-to-empty-is-iso*:

assumes $f : X \rightarrow \emptyset$
shows *isomorphism*(f)
<proof>

lemma *empty-prod-X*:

$\emptyset \times_c X \cong \emptyset$
<proof>

lemma *X-prod-empty*:

$X \times_c \emptyset \cong \emptyset$
<proof>

The lemma below corresponds to Proposition 2.4.8 in Halvorson.

lemma *no-el-iff-iso-empty*:

is-empty $X \longleftrightarrow X \cong \emptyset$
<proof>

lemma *initial-maps-mono*:

assumes *initial-object*(X)
assumes $f : X \rightarrow Y$
shows *monomorphism*(f)
<proof>

lemma *iso-empty-initial*:

assumes $X \cong \emptyset$

shows *initial-object* X

<proof>

lemma *function-to-empty-set-is-iso*:

assumes $f: X \rightarrow Y$

assumes *is-empty* Y

shows *isomorphism* f

<proof>

lemma *prod-iso-to-empty-right*:

assumes *nonempty* X

assumes $X \times_c Y \cong \emptyset$

shows *is-empty* Y

<proof>

lemma *prod-iso-to-empty-left*:

assumes *nonempty* Y

assumes $X \times_c Y \cong \emptyset$

shows *is-empty* X

<proof>

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

<proof>

The lemma below corresponds to Proposition 2.2.1 in Halvorson.

lemma *one-has-two-subsets*:

card ($\{(X,m). (X,m) \subseteq_c \mathbf{1}\} // \{((X1,m1),(X2,m2)). X1 \cong X2\}$) = 2

<proof>

lemma *coprod-with-init-obj1*:

assumes *initial-object* Y

shows $X \coprod Y \cong X$

<proof>

lemma *coprod-with-init-obj2*:

assumes *initial-object* X

shows $X \coprod Y \cong Y$

<proof>

lemma *prod-with-term-obj1*:

assumes *terminal-object*(X)

shows $X \times_c Y \cong Y$

<proof>

lemma *prod-with-term-obj2*:

assumes *terminal-object*(Y)

shows $X \times_c Y \cong X$

<proof>

end

12 Exponential Objects, Transposes and Evaluation

theory *Exponential-Objects*

imports *Initial*

begin

The axiomatization below corresponds to Axiom 9 (Exponential Objects) in Halvorson.

axiomatization

exp-set :: $cset \Rightarrow cset \Rightarrow cset$ ($-$ [100,100]100) **and**

eval-func :: $cset \Rightarrow cset \Rightarrow cfunc$ **and**

transpose-func :: $cfunc \Rightarrow cfunc$ ($\#$ [100]100)

where

exp-set-inj: $X^A = Y^B \Longrightarrow X = Y \wedge A = B$ **and**

eval-func-type[*type-rule*]: $eval-func\ X\ A : A \times_c X^A \rightarrow X$ **and**

transpose-func-type[*type-rule*]: $f : A \times_c Z \rightarrow X \Longrightarrow f^\# : Z \rightarrow X^A$ **and**

transpose-func-def: $f : A \times_c Z \rightarrow X \Longrightarrow (eval-func\ X\ A) \circ_c (id\ A \times_f f^\#) = f$

and

transpose-func-unique:

$f : A \times_c Z \rightarrow X \Longrightarrow g : Z \rightarrow X^A \Longrightarrow (eval-func\ X\ A) \circ_c (id\ A \times_f g) = f \Longrightarrow g = f^\#$

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*:

$(eval-func\ X\ A)^\# = id_c(X^A)$

<proof>

lemma *eval-func-X-empty-injective*:

assumes *is-empty* Y

shows *injective* (*eval-func* $X\ Y$)

<proof>

12.1 Lifting Functions

The definition below corresponds to Definition 2.5.1 in Halvorson.

definition *exp-func* :: $cfunc \Rightarrow cset \Rightarrow cfunc$ ($(-)_f$ [100,100]100) **where**

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

lemma *exp-func-def2*:

assumes $g : X \rightarrow Y$

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

<proof>

lemma *exp-func-type[type-rule]*:

assumes $g : X \rightarrow Y$

shows $g^A_f : X^A \rightarrow Y^A$

<proof>

lemma *exp-of-id-is-id-of-exp*:

$\text{id}(X^A) = (\text{id}(X))^A_f$

<proof>

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

lemma *exponential-square-diagram*:

assumes $g : Y \rightarrow Z$

shows $(\text{eval-func } Z \ A) \circ_c (\text{id}_c(A) \times_f g^A_f) = g \circ_c (\text{eval-func } Y \ A)$

<proof>

The lemma below corresponds to Proposition 2.5.2 in Halvorson.

lemma *transpose-of-comp*:

assumes *f-type*: $f : A \times_c X \rightarrow Y$ **and** *g-type*: $g : Y \rightarrow Z$

shows $f : A \times_c X \rightarrow Y \wedge g : Y \rightarrow Z \implies (g \circ_c f)^\sharp = g^A_f \circ_c f^\sharp$

<proof>

lemma *exponential-object-identity2*:

$\text{id}(X)^A_f = \text{id}_c(X^A)$

<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*:

$(\text{eval-func } (A \times_c X) \ A) \circ_c (\text{id}(A) \times_f (\text{id}(A \times_c X))^\sharp) = \text{id}(A \times_c X)$

<proof>

lemma *eval-of-id-cross-id-sharp2*:

assumes $a : Z \rightarrow A \ x : Z \rightarrow X$

shows $((\text{eval-func } (A \times_c X) \ A) \circ_c (\text{id}(A) \times_f (\text{id}(A \times_c X))^\sharp)) \circ_c \langle a, x \rangle = \langle a, x \rangle$

<proof>

lemma *transpose-factors*:

assumes $f : X \rightarrow Y$

assumes $g : Y \rightarrow Z$

shows $(g \circ_c f)^A_f = (g^A_f) \circ_c (f^A_f)$

<proof>

12.2 Inverse Transpose Function (flat)

The definition below corresponds to Definition 2.5.3 in Halvorson.

definition *inv-transpose-func* :: *cfunc* \Rightarrow *cfunc* (^b [100]100) **where**
 $f^b = (\text{THE } g. \exists Z X A. \text{domain } f = Z \wedge \text{codomain } f = X^A \wedge g = (\text{eval-func } X A) \circ_c (\text{id } A \times_f f))$

lemma *inv-transpose-func-def2*:

assumes $f : Z \rightarrow X^A$
shows $\exists Z X A. \text{domain } f = Z \wedge \text{codomain } f = X^A \wedge f^b = (\text{eval-func } X A) \circ_c (\text{id } A \times_f f)$
 $\langle \text{proof} \rangle$

lemma *inv-transpose-func-def3*:

assumes *f-type*: $f : Z \rightarrow X^A$
shows $f^b = (\text{eval-func } X A) \circ_c (\text{id } A \times_f f)$
 $\langle \text{proof} \rangle$

lemma *flat-type[type-rule]*:

assumes *f-type[type-rule]*: $f : Z \rightarrow X^A$
shows $f^b : A \times_c Z \rightarrow X$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Proposition 2.5.4 in Halvorson.

lemma *inv-transpose-of-composition*:

assumes $f : X \rightarrow Y$ $g : Y \rightarrow Z^A$
shows $(g \circ_c f)^b = g^b \circ_c (\text{id}(A) \times_f f)$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Proposition 2.5.5 in Halvorson.

lemma *flat-cancels-sharp*:

$f : A \times_c Z \rightarrow X \implies (f^\sharp)^b = f$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Proposition 2.5.6 in Halvorson.

lemma *sharp-cancels-flat*:

$f : Z \rightarrow X^A \implies (f^b)^\sharp = f$
 $\langle \text{proof} \rangle$

lemma *same-evals-equal*:

assumes $f : Z \rightarrow X^A$ $g : Z \rightarrow X^A$
shows $\text{eval-func } X A \circ_c (\text{id } A \times_f f) = \text{eval-func } X A \circ_c (\text{id } A \times_f g) \implies f = g$
 $\langle \text{proof} \rangle$

lemma *sharp-comp*:

assumes *f-type[type-rule]*: $f : A \times_c Z \rightarrow X$ **and** *g-type[type-rule]*: $g : W \rightarrow Z$
shows $f^\sharp \circ_c g = (f \circ_c (\text{id } A \times_f g))^\sharp$
 $\langle \text{proof} \rangle$

lemma *flat-pres-epi*:
assumes *nonempty*(A)
assumes $f : Z \rightarrow X^A$
assumes *epimorphism* f
shows *epimorphism*(f^b)
 \langle *proof* \rangle

lemma *transpose-inj-is-inj*:
assumes $g : X \rightarrow Y$
assumes *injective* g
shows *injective*(g^A_f)
 \langle *proof* \rangle

lemma *eval-func-X-one-injective*:
injective (*eval-func* X $\mathbf{1}$)
 \langle *proof* \rangle

In the lemma below, the nonempty assumption is required. Consider, for example, $X = \Omega$ and $A = \emptyset$

lemma *sharp-pres-mono*:
assumes $f : A \times_c Z \rightarrow X$
assumes *monomorphism*(f)
assumes *nonempty* A
shows *monomorphism*($f^\#$)
 \langle *proof* \rangle

12.3 Metafunctions and their Inverses (Cnufatems)

12.3.1 Metafunctions

definition *metafunc* :: *cfunc* \Rightarrow *cfunc* **where**
metafunc $f \equiv (f \circ_c (\text{left-cart-proj } (\text{domain } f) \mathbf{1}))^\#$

lemma *metafunc-def2*:
assumes $f : X \rightarrow Y$
shows *metafunc* $f = (f \circ_c (\text{left-cart-proj } X \mathbf{1}))^\#$
 \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-type[type-rule]*: $f : X \rightarrow Y$
assumes *x-type[type-rule]*: $x \in_c X$
shows *eval-func* $Y X \circ_c \langle x, \text{metafunc } f \rangle = f \circ_c x$
 \langle *proof* \rangle

12.3.2 Inverse Metafunctions (Cnufatems)

definition *cnufatem* :: *cfunc* \Rightarrow *cfunc* **where**

cnufatem $f = (\text{THE } g. \forall Y X. f : \mathbf{1} \rightarrow Y^X \longrightarrow g = \text{eval-func } Y X \circ_c \langle \text{id } X, f \circ_c \beta_X \rangle)$

lemma *cnufatem-def2*:

assumes $f \in_c Y^X$

shows $\text{cnufatem } f = \text{eval-func } Y X \circ_c \langle \text{id } X, f \circ_c \beta_X \rangle$

$\langle \text{proof} \rangle$

lemma *cnufatem-type*[*type-rule*]:

assumes $f \in_c Y^X$

shows $\text{cnufatem } f : X \rightarrow Y$

$\langle \text{proof} \rangle$

lemma *cnufatem-metafunc*:

assumes *f-type*[*type-rule*]: $f : X \rightarrow Y$

shows $\text{cnufatem } (\text{metafunc } f) = f$

$\langle \text{proof} \rangle$

lemma *metafunc-cnufatem*:

assumes *f-type*[*type-rule*]: $f \in_c Y^X$

shows $\text{metafunc } (\text{cnufatem } f) = f$

$\langle \text{proof} \rangle$

12.3.3 Metafunction Composition

definition *meta-comp* :: *cset* \Rightarrow *cset* \Rightarrow *cset* \Rightarrow *cfunc* **where**

meta-comp $X Y Z = (\text{eval-func } Z Y \circ_c \text{swap } (Z^Y) Y \circ_c (\text{id}(Z^Y) \times_f (\text{eval-func } Y X \circ_c \text{swap } (Y^X) X)) \circ_c (\text{associate-right } (Z^Y) (Y^X) X) \circ_c \text{swap } X ((Z^Y) \times_c (Y^X)))^\#$

lemma *meta-comp-type*[*type-rule*]:

meta-comp $X Y Z : Z^Y \times_c Y^X \rightarrow Z^X$

$\langle \text{proof} \rangle$

definition *meta-comp2* :: *cfunc* \Rightarrow *cfunc* \Rightarrow *cfunc* (**infixr** \square 55)

where *meta-comp2* $f g = (\text{THE } h. \exists W X Y. g : W \rightarrow Y^X \wedge h = (f^b \circ_c \langle g^b, \text{right-cart-proj } X W \rangle)^\#)$

lemma *meta-comp2-def2*:

assumes $f : W \rightarrow Z^Y$

assumes $g : W \rightarrow Y^X$

shows $f \square g = (f^b \circ_c \langle g^b, \text{right-cart-proj } X W \rangle)^\#$

$\langle \text{proof} \rangle$

lemma *meta-comp2-type*[*type-rule*]:

assumes $f : W \rightarrow Z^Y$

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

lemma *meta-comp2-elements-aux*:

assumes $f \in_c Z^Y$
assumes $g \in_c Y^X$
assumes $x \in_c X$
shows $(f^\flat \circ_c \langle g^\flat, \text{right-cart-proj } X \mathbf{1} \rangle) \circ_c \langle x, \text{id}_c \mathbf{1} \rangle = \text{eval-func } Z \ Y \circ_c \langle \text{eval-func } Y \ X \circ_c \langle x, g \rangle, f \rangle$
 ⟨*proof*⟩

lemma *meta-comp2-def3*:

assumes $f \in_c Z^Y$
assumes $g \in_c Y^X$
shows $f \square g = \text{metafunc } ((\text{cnufatem } f) \circ_c (\text{cnufatem } g))$
 ⟨*proof*⟩

lemma *meta-comp2-def4*:

assumes $f\text{-type}[type\text{-rule}]: f \in_c Z^Y$ **and** $g\text{-type}[type\text{-rule}]: g \in_c Y^X$
shows $f \square g = \text{meta-comp } X \ Y \ Z \circ_c \langle f, g \rangle$
 ⟨*proof*⟩

lemma *meta-comp-on-els*:

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

lemma *meta-left-identity*:

assumes $g \in_c X^X$
shows $g \square \text{metafunc } (\text{id } X) = g$
 ⟨*proof*⟩

lemma *meta-right-identity*:

assumes $g \in_c X^X$
shows $\text{metafunc}(\text{id } X) \square g = g$
 ⟨*proof*⟩

lemma *comp-as-metacomp*:

assumes $g : X \rightarrow Y$
assumes $f : Y \rightarrow Z$

shows $f \circ_c g = \text{cnufatem}(\text{metafunc } f \square \text{metafunc } g)$
 $\langle \text{proof} \rangle$

lemma *metacomp-as-comp*:

assumes $g \in_c Y^X$

assumes $f \in_c Z^Y$

shows $\text{cnufatem } f \circ_c \text{cnufatem } g = \text{cnufatem}(f \square g)$

$\langle \text{proof} \rangle$

lemma *meta-comp-assoc*:

assumes $e : W \rightarrow A^Z$

assumes $f : W \rightarrow Z^Y$

assumes $g : W \rightarrow Y^X$

shows $(e \square f) \square g = e \square (f \square g)$

$\langle \text{proof} \rangle$

12.4 Partially Parameterized Functions on Pairs

definition *left-param* :: *cfunc* \Rightarrow *cfunc* \Rightarrow *cfunc* $(\text{-}_{[-,-]} [100,0]100)$ **where**

left-param k $p \equiv (\text{THE } f. \exists P Q R. k : P \times_c Q \rightarrow R \wedge f = k \circ_c \langle p \circ_c \beta_Q, \text{id } Q \rangle)$

lemma *left-param-def2*:

assumes $k : P \times_c Q \rightarrow R$

shows $k_{[p,-]} \equiv k \circ_c \langle p \circ_c \beta_Q, \text{id } Q \rangle$

$\langle \text{proof} \rangle$

lemma *left-param-type*[*type-rule*]:

assumes $k : P \times_c Q \rightarrow R$

assumes $p \in_c P$

shows $k_{[p,-]} : Q \rightarrow R$

$\langle \text{proof} \rangle$

lemma *left-param-on-el*:

assumes $k : P \times_c Q \rightarrow R$

assumes $p \in_c P$

assumes $q \in_c Q$

shows $k_{[p,-]} \circ_c q = k \circ_c \langle p, q \rangle$

$\langle \text{proof} \rangle$

definition *right-param* :: *cfunc* \Rightarrow *cfunc* \Rightarrow *cfunc* $(\text{-}_{[-,-]} [100,0]100)$ **where**

right-param k $q \equiv (\text{THE } f. \exists P Q R. k : P \times_c Q \rightarrow R \wedge f = k \circ_c \langle \text{id } P, q \circ_c \beta_P \rangle)$

lemma *right-param-def2*:

assumes $k : P \times_c Q \rightarrow R$

shows $k_{[-,q]} \equiv k \circ_c \langle \text{id } P, q \circ_c \beta_P \rangle$

$\langle \text{proof} \rangle$

lemma *right-param-type*[*type-rule*]:

assumes $k : P \times_c Q \rightarrow R$

assumes $q \in_c Q$

shows $k_{[-,q]} : P \rightarrow R$

<proof>

lemma *right-param-on-el*:

assumes $k : P \times_c Q \rightarrow R$

assumes $p \in_c P$

assumes $q \in_c Q$

shows $k_{[-,q]} \circ_c p = k \circ_c \langle p, q \rangle$

<proof>

12.5 Exponential Set Facts

The lemma below corresponds to Proposition 2.5.7 in Halvorson.

lemma *exp-one*:

$X^{\mathbf{1}} \cong X$

<proof>

The lemma below corresponds to Proposition 2.5.8 in Halvorson.

lemma *exp-empty*:

$X^\emptyset \cong \mathbf{1}$

<proof>

lemma *one-exp*:

$\mathbf{1}^X \cong \mathbf{1}$

<proof>

The lemma below corresponds to Proposition 2.5.9 in Halvorson.

lemma *power-rule*:

$(X \times_c Y)^A \cong X^A \times_c Y^A$

<proof>

lemma *exponential-coprod-distribution*:

$Z(X \amalg Y) \cong (Z^X) \times_c (Z^Y)$

<proof>

lemma *empty-exp-nonempty*:

assumes *nonempty* X

shows $\emptyset^X \cong \emptyset$

<proof>

lemma *exp-pres-iso-left*:

assumes $A \cong X$

shows $A^Y \cong X^Y$

<proof>

lemma *expset-power-tower*:

$$(A^B)^C \cong A^{(B \times_c C)}$$

<proof>

lemma *exp-pres-iso-right*:

assumes $A \cong X$

shows $Y^A \cong Y^X$

<proof>

lemma *exp-pres-iso*:

assumes $A \cong X$ $B \cong Y$

shows $A^B \cong X^Y$

<proof>

lemma *empty-to-nonempty*:

assumes *nonempty* X *is-empty* Y

shows $Y^X \cong \emptyset$

<proof>

lemma *exp-is-empty*:

assumes *is-empty* X

shows $Y^X \cong \mathbf{1}$

<proof>

lemma *nonempty-to-nonempty*:

assumes *nonempty* X *nonempty* Y

shows *nonempty*(Y^X)

<proof>

lemma *empty-to-nonempty-converse*:

assumes $Y^X \cong \emptyset$

shows *is-empty* $Y \wedge$ *nonempty* X

<proof>

The definition below corresponds to Definition 2.5.11 in Halvorson.

definition *powerset* :: *cset* \Rightarrow *cset* (\mathcal{P} -[101]100) **where**

$$\mathcal{P} X = \Omega^X$$

lemma *sets-squared*:

$$A^\Omega \cong A \times_c A$$

<proof>

end

13 Natural Number Object

theory *Nats*

imports *Exponential-Objects*

begin

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

axiomatization

natural-numbers :: *cset* (\mathbb{N}_c) **and**

zero :: *cfunc* **and**

successor :: *cfunc*

where

zero-type[*type-rule*]: $zero \in_c \mathbb{N}_c$ **and**

successor-type[*type-rule*]: $successor: \mathbb{N}_c \rightarrow \mathbb{N}_c$ **and**

natural-number-object-property:

$q : \mathbf{1} \rightarrow X \implies f: X \rightarrow X \implies$

$(\exists! u. u: \mathbb{N}_c \rightarrow X \wedge$

$q = u \circ_c zero \wedge$

$f \circ_c u = u \circ_c successor)$

lemma *beta-N-succ-nEqs-Id1*:

assumes *n-type*[*type-rule*]: $n \in_c \mathbb{N}_c$

shows $\beta_{\mathbb{N}_c} \circ_c successor \circ_c n = id \mathbf{1}$

<proof>

lemma *natural-number-object-property2*:

assumes $q : \mathbf{1} \rightarrow X$ $f: X \rightarrow X$

shows $\exists! u. u: \mathbb{N}_c \rightarrow X \wedge u \circ_c zero = q \wedge f \circ_c u = u \circ_c successor$

<proof>

lemma *natural-number-object-func-unique*:

assumes *u-type*: $u : \mathbb{N}_c \rightarrow X$ **and** *v-type*: $v : \mathbb{N}_c \rightarrow X$ **and** *f-type*: $f: X \rightarrow X$

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* \implies *cfunc* \implies *cfunc* \implies *bool* **where**

is-NNO $Y z s \iff (z: \mathbf{1} \rightarrow Y \wedge s: Y \rightarrow Y \wedge (\forall X f q. ((q : \mathbf{1} \rightarrow X) \wedge (f: X \rightarrow X)) \implies$

$(\exists! u. u: Y \rightarrow X \wedge$

$q = u \circ_c z \wedge$

$f \circ_c u = u \circ_c s)))$

lemma *N-is-a-NNO*:

is-NNO \mathbb{N}_c *zero* *successor*

<proof>

The lemma below corresponds to Exercise 2.6.5 in Halvorson.

lemma *NNOs-are-iso-N*:

assumes *is-NNO* $N z s$

shows $N \cong \mathbf{N}_c$
 ⟨*proof*⟩

The lemma below is the converse to Exercise 2.6.5 in Halvorson.

lemma *Iso-to-N-is-NNO*:
assumes $N \cong \mathbf{N}_c$
shows $\exists z s. \text{is-NNO } N z s$
 ⟨*proof*⟩

13.1 Zero and Successor

lemma *zero-is-not-successor*:
assumes $n \in_c \mathbf{N}_c$
shows $\text{zero} \neq \text{successor} \circ_c n$
 ⟨*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 $x \in_c (\mathbf{1} \amalg \mathbf{N}_c)$
shows $(x = (\text{left-coproj } \mathbf{1} \mathbf{N}_c) \circ_c \text{id } \mathbf{1}) \vee$
 $(\exists n. (n \in_c \mathbf{N}_c) \wedge (x = (\text{right-coproj } \mathbf{1} \mathbf{N}_c) \circ_c n))$
 ⟨*proof*⟩

lemma *nonzero-is-succ*:
assumes $k \in_c \mathbf{N}_c$
assumes $k \neq \text{zero}$
shows $\exists n. (n \in_c \mathbf{N}_c \wedge k = \text{successor} \circ_c n)$
 ⟨*proof*⟩

13.2 Predecessor

definition *predecessor'* :: *cfunc* **where**
 $\text{predecessor}' = (\text{THE } f. f : \mathbf{N}_c \rightarrow \mathbf{1} \amalg \mathbf{N}_c$
 $\wedge f \circ_c (\text{zero} \amalg \text{successor}) = \text{id } (\mathbf{1} \amalg \mathbf{N}_c) \wedge (\text{zero} \amalg \text{successor}) \circ_c f = \text{id } \mathbf{N}_c)$

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

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

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

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

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

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

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

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

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

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

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

13.3 Peano's Axioms and Induction

The lemma below corresponds to Proposition 2.6.7 in Halvorson.

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

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

theorem *nat-induction*:

assumes *p-type*[*type-rule*]: $p : \mathbf{N}_c \rightarrow \Omega$ **and** *n-type*[*type-rule*]: $n \in_c \mathbf{N}_c$
assumes *base-case*: $p \circ_c \text{zero} = t$
assumes *induction-case*: $\bigwedge n. n \in_c \mathbf{N}_c \implies p \circ_c n = t \implies p \circ_c \text{successor} \circ_c n = t$
shows $p \circ_c n = t$
 $\langle \text{proof} \rangle$

13.4 Function Iteration

definition *ITER-curried* :: $cset \Rightarrow cfunc$ **where**

ITER-curried $U = (\text{THE } u . u : \mathbf{N}_c \rightarrow (U^U)U^U \wedge u \circ_c \text{zero} = (\text{metafunc } (\text{id } U) \circ_c (\text{right-cart-proj } (U^U) \mathbf{1}))^\# \wedge$
 $((\text{meta-comp } U \ U \ U) \circ_c (\text{id } (U^U) \times_f \text{eval-func } (U^U) (U^U)) \circ_c (\text{associate-right } (U^U) (U^U) ((U^U)U^U))) \circ_c (\text{diagonal}(U^U) \times_f \text{id } ((U^U)U^U)))^\# \circ_c u = u \circ_c \text{successor})$

lemma *ITER-curried-def2*:

ITER-curried $U : \mathbf{N}_c \rightarrow (U^U)U^U \wedge \text{ITER-curried } U \circ_c \text{zero} = (\text{metafunc } (\text{id } U) \circ_c (\text{right-cart-proj } (U^U) \mathbf{1}))^\# \wedge$
 $((\text{meta-comp } U \ U \ U) \circ_c (\text{id } (U^U) \times_f \text{eval-func } (U^U) (U^U)) \circ_c (\text{associate-right } (U^U) (U^U) ((U^U)U^U))) \circ_c (\text{diagonal}(U^U) \times_f \text{id } ((U^U)U^U)))^\# \circ_c \text{ITER-curried } U = \text{ITER-curried } U \circ_c \text{successor}$
 $\langle \text{proof} \rangle$

lemma *ITER-curried-type*[*type-rule*]:

ITER-curried $U : \mathbf{N}_c \rightarrow (U^U)U^U$
 $\langle \text{proof} \rangle$

lemma *ITER-curried-zero*:

ITER-curried $U \circ_c \text{zero} = (\text{metafunc } (\text{id } U) \circ_c (\text{right-cart-proj } (U^U) \mathbf{1}))^\#$
 $\langle \text{proof} \rangle$

lemma *ITER-curried-successor*:

ITER-curried $U \circ_c \text{successor} = (\text{meta-comp } U \ U \ U \circ_c (\text{id } (U^U) \times_f \text{eval-func } (U^U) (U^U)) \circ_c (\text{associate-right } (U^U) (U^U) ((U^U)U^U))) \circ_c (\text{diagonal}(U^U) \times_f \text{id } ((U^U)U^U)))^\# \circ_c \text{ITER-curried } U$
 $\langle \text{proof} \rangle$

definition *ITER* :: $cset \Rightarrow cfunc$ **where**

ITER $U = (\text{ITER-curried } U)^\flat$

lemma *ITER-type*[*type-rule*]:

ITER $U : ((U^U) \times_c \mathbf{N}_c) \rightarrow (U^U)$
 $\langle \text{proof} \rangle$

lemma *ITER-zero*:

assumes *f-type*[*type-rule*]: $f : Z \rightarrow (U^U)$

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-type*[*type-rule*]: $f : Z \rightarrow (U^U)$ **and** *n-type*[*type-rule*]: $n : Z \rightarrow \mathbb{N}_c$

shows $ITER\ U \circ_c \langle f, successor \circ_c n \rangle = f \sqcap (ITER\ U \circ_c \langle f, n \rangle)$

<proof>

lemma *ITER-one*:

assumes $f \in_c (U^U)$

shows $ITER\ U \circ_c \langle f, successor \circ_c zero \rangle = f \sqcap (metafunc\ (id\ U))$

<proof>

definition *iter-comp* :: *cfunc* \Rightarrow *cfunc* \Rightarrow *cfunc* ($-\circ^{[55,0]55}$) **where**

iter-comp $g\ n \equiv cnufatem\ (ITER\ (domain\ g) \circ_c \langle metafunc\ g, n \rangle)$

lemma *iter-comp-def2*:

$g^{\circ n} \equiv cnufatem\ (ITER\ (domain\ g) \circ_c \langle metafunc\ g, n \rangle)$

<proof>

lemma *iter-comp-type*[*type-rule*]:

assumes $g : X \rightarrow X$

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-type*[*type-rule*]: $g : X \rightarrow X$

shows $g^{\circ zero} = id_c\ X$

<proof>

lemma *succ-iters*:

assumes $g : X \rightarrow X$

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 $g : X \rightarrow X$

shows $g^{\circ(\text{successor} \circ_c \text{zero})} = g$

<proof>

lemma *eval-lemma-for-ITER:*

assumes $f : X \rightarrow X$

assumes $x \in_c X$

assumes $m \in_c \mathbf{N}_c$

shows $(f^{\circ m}) \circ_c x = \text{eval-func } X \ X \ \circ_c \langle x, \text{ITER } X \ \circ_c \langle \text{metafunc } f, m \rangle \rangle$

<proof>

lemma *n-accessible-by-succ-iter-aux:*

$\text{eval-func } \mathbf{N}_c \ \mathbf{N}_c \ \circ_c \langle \text{zero} \circ_c \beta_{\mathbf{N}_c}, \text{ITER } \mathbf{N}_c \ \circ_c \langle (\text{metafunc } \text{successor}) \circ_c \beta_{\mathbf{N}_c}, \text{id } \mathbf{N}_c \rangle \rangle = \text{id } \mathbf{N}_c$

<proof>

lemma *n-accessible-by-succ-iter:*

assumes $n \in_c \mathbf{N}_c$

shows $(\text{successor}^{\circ n}) \circ_c \text{zero} = n$

<proof>

13.5 Relation of Nat to Other Sets

lemma *oneUN-iso-N:*

$\mathbf{1} \coprod \mathbf{N}_c \cong \mathbf{N}_c$

<proof>

lemma *NUone-iso-N:*

$\mathbf{N}_c \coprod \mathbf{1} \cong \mathbf{N}_c$

<proof>

end

14 Predicate Logic Functions

theory *Pred-Logic*

imports *Coproduct*

begin

14.1 NOT

definition *NOT* :: *cfunc* **where**

$\text{NOT} = (\text{THE } \chi. \text{is-pullback } \mathbf{1} \ \mathbf{1} \ \Omega \ (\beta_{\mathbf{1}}) \ \text{t f } \chi)$

lemma *NOT-is-pullback:*

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

<proof>

lemma *NOT-type[type-rule]:*

$NOT : \Omega \rightarrow \Omega$

<proof>

lemma *NOT-false-is-true:*

$NOT \circ_c f = t$

<proof>

lemma *NOT-true-is-false:*

$NOT \circ_c t = f$

<proof>

lemma *NOT-is-true-implies-false:*

assumes $p \in_c \Omega$

shows $NOT \circ_c p = t \implies p = f$

<proof>

lemma *NOT-is-false-implies-true:*

assumes $p \in_c \Omega$

shows $NOT \circ_c p = f \implies p = t$

<proof>

lemma *double-negation:*

$NOT \circ_c NOT = id \ \Omega$

<proof>

14.2 AND

definition *AND :: cfunc where*

$AND = (THE \ \chi. \ is_pullback \ \mathbf{1} \ \mathbf{1} \ (\Omega \times_c \ \Omega) \ \Omega \ (\beta_{\mathbf{1}}) \ t \ \langle t, t \rangle \ \chi)$

lemma *AND-is-pullback:*

$is_pullback \ \mathbf{1} \ \mathbf{1} \ (\Omega \times_c \ \Omega) \ \Omega \ (\beta_{\mathbf{1}}) \ t \ \langle t, t \rangle \ AND$

<proof>

lemma *AND-type[type-rule]:*

$AND : \Omega \times_c \ \Omega \rightarrow \Omega$

<proof>

lemma *AND-true-true-is-true:*

$AND \circ_c \langle t, t \rangle = t$

<proof>

lemma *AND-false-left-is-false:*

assumes $p \in_c \Omega$

shows $AND \circ_c \langle f, p \rangle = f$

<proof>

lemma *AND-false-right-is-false*:

assumes $p \in_c \Omega$

shows $AND \circ_c \langle p, f \rangle = f$

<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 $AND \circ_c \langle p, NOT \circ_c p \rangle = f$

<proof>

14.3 NOR

definition *NOR* :: *cfunc where*

$NOR = (THE \chi. is_pullback \mathbf{1} \mathbf{1} (\Omega \times_c \Omega) \Omega (\beta_{\mathbf{1}}) t \langle f, f \rangle \chi)$

lemma *NOR-is-pullback*:

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

<proof>

lemma *NOR-type[type-rule]*:

$NOR : \Omega \times_c \Omega \rightarrow \Omega$

<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 $NOR \circ_c \langle t, p \rangle = f$

$\langle proof \rangle$

lemma *NOR-right-true-is-false*:

assumes $p \in_c \Omega$

shows $NOR \circ_c \langle p, t \rangle = f$

$\langle proof \rangle$

lemma *NOR-true-implies-both-false*:

assumes *X-nonempty*: $nonempty\ X$ **and** *Y-nonempty*: $nonempty\ Y$

assumes *P-Q-types*[*type-rule*]: $P : X \rightarrow \Omega\ Q : Y \rightarrow \Omega$

assumes *NOR-true*: $NOR \circ_c (P \times_f Q) = t \circ_c \beta_X \times_c Y$

shows $P = f \circ_c \beta_X \wedge Q = f \circ_c \beta_Y$

$\langle proof \rangle$

lemma *NOR-true-implies-neither-true*:

assumes *X-nonempty*: $nonempty\ X$ **and** *Y-nonempty*: $nonempty\ Y$

assumes *P-Q-types*[*type-rule*]: $P : X \rightarrow \Omega\ Q : Y \rightarrow \Omega$

assumes *NOR-true*: $NOR \circ_c (P \times_f Q) = t \circ_c \beta_X \times_c Y$

shows $\neg (P = t \circ_c \beta_X \vee Q = t \circ_c \beta_Y)$

$\langle proof \rangle$

14.4 OR

definition *OR* :: *cfunc* **where**

$OR = (THE\ \chi. is_pullback\ (\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))})\ t\ ((t, t) \amalg ((t, f) \amalg (f, t)))\ \chi)$

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

$\langle t, t \rangle \amalg ((t, f) \amalg (f, t)) : \mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}) \rightarrow \Omega \times_c \Omega$

$\langle proof \rangle$

lemma *set-three*:

$\{x. x \in_c (\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))\} = \{$

$(left_coproj\ \mathbf{1}\ (\mathbf{1} \amalg \mathbf{1})),$

$(right_coproj\ \mathbf{1}\ (\mathbf{1} \amalg \mathbf{1}) \circ_c left_coproj\ \mathbf{1}\ \mathbf{1}),$

$right_coproj\ \mathbf{1}\ (\mathbf{1} \amalg \mathbf{1}) \circ_c (right_coproj\ \mathbf{1}\ \mathbf{1})\}$

$\langle proof \rangle$

lemma *set-three-card*:

$card\ \{x. x \in_c (\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))\} = 3$

$\langle proof \rangle$

lemma *pre-OR-injective*:

$injective\ ((t, t) \amalg ((t, f) \amalg (f, t)))$

$\langle proof \rangle$

lemma *OR-is-pullback*:

$is_pullback\ (\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))\ \mathbf{1}\ (\Omega \times_c \Omega)\ \Omega\ (\beta_{(\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))})\ t\ ((t, t) \amalg ((t, f) \amalg (f, t)))$

OR

$\langle proof \rangle$

lemma *OR-type[type-rule]*:

$OR : \Omega \times_c \Omega \rightarrow \Omega$

$\langle proof \rangle$

lemma *OR-true-left-is-true*:

assumes $p \in_c \Omega$

shows $OR \circ_c \langle t, p \rangle = t$

$\langle proof \rangle$

lemma *OR-true-right-is-true*:

assumes $p \in_c \Omega$

shows $OR \circ_c \langle p, t \rangle = t$

$\langle proof \rangle$

lemma *OR-false-false-is-false*:

$OR \circ_c \langle f, f \rangle = f$

$\langle proof \rangle$

lemma *OR-true-implies-one-is-true*:

assumes $p \in_c \Omega$

assumes $q \in_c \Omega$

assumes $OR \circ_c \langle p, q \rangle = t$

shows $(p = t) \vee (q = t)$

$\langle proof \rangle$

lemma *NOT-NOR-is-OR*:

$OR = NOT \circ_c NOR$

$\langle proof \rangle$

lemma *OR-commutative*:

assumes $p \in_c \Omega$

assumes $q \in_c \Omega$

shows $OR \circ_c \langle p, q \rangle = OR \circ_c \langle q, p \rangle$

$\langle proof \rangle$

lemma *OR-idempotent*:

assumes $p \in_c \Omega$

shows $OR \circ_c \langle p, p \rangle = p$

$\langle proof \rangle$

lemma *OR-associative*:

assumes $p \in_c \Omega$

assumes $q \in_c \Omega$

assumes $r \in_c \Omega$

shows $OR \circ_c \langle OR \circ_c \langle p, q \rangle, r \rangle = OR \circ_c \langle p, OR \circ_c \langle q, r \rangle \rangle$

$\langle proof \rangle$

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

14.5 XOR

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

lemma *pre-XOR-type*[*type-rule*]:
 $\langle t, f \rangle \sqcup \langle f, t \rangle : \mathbf{1} \sqcup \mathbf{1} \rightarrow \Omega \times_c \Omega$
 $\langle proof \rangle$

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

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

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

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

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

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

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

14.6 NAND

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

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

$\langle f, f \rangle \amalg (\langle t, f \rangle \amalg \langle f, t \rangle) : \mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}) \rightarrow \Omega \times_c \Omega$
 $\langle proof \rangle$

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-pullback ($\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1})$) $\mathbf{1}$ ($\Omega \times_c \Omega$) Ω ($\beta_{(\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))}$) 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 : $\Omega \times_c \Omega \rightarrow \Omega$
 $\langle proof \rangle$

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* $\circ_c \langle p, p \rangle = \text{NOT} \circ_c p$
 $\langle proof \rangle$

14.7 IFF

definition *IFF* :: *cfunc* **where**

$IFF = (THE \chi. is-pullback (1 \amalg 1) 1 (\Omega \times_c \Omega) \Omega (\beta_{(1 \amalg 1)}) t ((t, t) \amalg (f, f)) \chi)$

lemma *pre-IFF-type*[type-rule]:
 $\langle t, t \rangle \amalg \langle f, f \rangle : 1 \amalg 1 \rightarrow \Omega \times_c \Omega$
 <proof>

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

lemma *IFF-is-pullback*:
 is-pullback $(1 \amalg 1) 1 (\Omega \times_c \Omega) \Omega (\beta_{(1 \amalg 1)}) t ((t, t) \amalg (f, f)) IFF$
 <proof>

lemma *IFF-type*[type-rule]:
 $IFF : \Omega \times_c \Omega \rightarrow \Omega$
 <proof>

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

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

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

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

lemma *NOT-IFF-is-XOR*:
 $NOT \circ_c IFF = XOR$
 <proof>

14.8 IMPLIES

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

lemma *pre-IMPLIES-type*[type-rule]:
 $\langle t, t \rangle \amalg ((f, f) \amalg (f, t)) : 1 \amalg (1 \amalg 1) \rightarrow \Omega \times_c \Omega$
 <proof>

lemma *pre-IMPLIES-injective*:

injective($\langle t, t \rangle \amalg (\langle f, f \rangle \amalg \langle f, t \rangle)$)
 $\langle proof \rangle$

lemma *IMPLIES-is-pullback*:

is-pullback ($\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1})$) $\mathbf{1}$ ($\Omega \times_c \Omega$) Ω ($\beta(\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}))$) t ($\langle t, t \rangle \amalg (\langle f, f \rangle \amalg \langle f, t \rangle)$)
IMPLIES
 $\langle proof \rangle$

lemma *IMPLIES-type[type-rule]*:

IMPLIES : $\Omega \times_c \Omega \rightarrow \Omega$
 $\langle proof \rangle$

lemma *IMPLIES-true-true-is-true*:

IMPLIES $\circ_c \langle t, t \rangle = t$
 $\langle proof \rangle$

lemma *IMPLIES-false-true-is-true*:

IMPLIES $\circ_c \langle f, t \rangle = t$
 $\langle proof \rangle$

lemma *IMPLIES-false-false-is-true*:

IMPLIES $\circ_c \langle f, f \rangle = t$
 $\langle proof \rangle$

lemma *IMPLIES-true-false-is-false*:

IMPLIES $\circ_c \langle t, f \rangle = f$
 $\langle proof \rangle$

lemma *IMPLIES-false-is-true-false*:

assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
assumes *IMPLIES* $\circ_c \langle p, q \rangle = f$
shows $p = t \wedge q = f$
 $\langle proof \rangle$

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

lemma *iff-is-and-implies-implies-swap*:

IFF = *AND* $\circ_c \langle \text{IMPLIES}, \text{IMPLIES} \circ_c \text{swap} \Omega \Omega \rangle$
 $\langle proof \rangle$

lemma *IMPLIES-is-OR-NOT-id*:

IMPLIES = *OR* $\circ_c (\text{NOT} \times_f \text{id}(\Omega))$
 $\langle proof \rangle$

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* $\circ_c (P \times_f Q) = t \circ_c \beta_X \times_c Y$
shows $P = t \circ_c \beta_X \implies Q = t \circ_c \beta_Y$

<proof>

lemma *IMPLIES-elim*:

assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = t \circ_c \beta_{X \times_c Y}$
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 $(P = t \circ_c \beta_X) \implies ((Q = t \circ_c \beta_Y) \implies R) \implies R$
<proof>

lemma *IMPLIES-elim''*:

assumes *IMPLIES-true*: $IMPLIES \circ_c (P \times_f Q) = t$
assumes *P-type[type-rule]*: $P : \mathbf{1} \rightarrow \Omega$ **and** *Q-type[type-rule]*: $Q : \mathbf{1} \rightarrow \Omega$
shows $(P = t) \implies ((Q = t) \implies R) \implies R$
<proof>

lemma *IMPLIES-elim'*:

assumes *IMPLIES-true*: $IMPLIES \circ_c \langle P, Q \rangle = t$
assumes *P-type[type-rule]*: $P : \mathbf{1} \rightarrow \Omega$ **and** *Q-type[type-rule]*: $Q : \mathbf{1} \rightarrow \Omega$
shows $(P = t) \implies ((Q = t) \implies R) \implies R$
<proof>

lemma *implies-implies-IMPLIES*:

assumes *P-type[type-rule]*: $P : \mathbf{1} \rightarrow \Omega$ **and** *Q-type[type-rule]*: $Q : \mathbf{1} \rightarrow \Omega$
shows $(P = t \implies Q = t) \implies IMPLIES \circ_c \langle P, Q \rangle = t$
<proof>

14.9 Other Boolean Identities

lemma *AND-OR-distributive*:

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

lemma *deMorgan-Law1*:

assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $NOT \circ_c OR \circ_c \langle p, q \rangle = AND \circ_c \langle NOT \circ_c p, NOT \circ_c q \rangle$
 $\langle proof \rangle$

lemma *deMorgan-Law2*:

assumes $p \in_c \Omega$
assumes $q \in_c \Omega$
shows $NOT \circ_c AND \circ_c \langle p, q \rangle = OR \circ_c \langle NOT \circ_c p, NOT \circ_c q \rangle$
 $\langle proof \rangle$

end

15 Quantifiers

theory *Quant-Logic*

imports *Pred-Logic Exponential-Objects*

begin

15.1 Universal Quantification

definition *FORALL* :: *cset* \Rightarrow *cfunc* **where**

$FORALL X = (THE \chi. is-pullback \mathbf{1} \mathbf{1} (\Omega^X) \Omega (\beta_{\mathbf{1}}) t ((t \circ_c \beta_{X \times_c \mathbf{1}})^\#) \chi)$

lemma *FORALL-is-pullback*:

$is-pullback \mathbf{1} \mathbf{1} (\Omega^X) \Omega (\beta_{\mathbf{1}}) t ((t \circ_c \beta_{X \times_c \mathbf{1}})^\#) (FORALL X)$
 $\langle proof \rangle$

lemma *FORALL-type[type-rule]*:

$FORALL X : \Omega^X \rightarrow \Omega$
 $\langle proof \rangle$

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

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

lemma *all-true-implies-FORALL-true2*:

assumes $p\text{-type}[type\text{-rule}] : p : X \times_c Y \rightarrow \Omega$ **and** $all\text{-}p\text{-true} : \bigwedge xy. xy \in_c X \times_c Y \implies p \circ_c xy = t$
shows $FORALL X \circ_c p^\# = t \circ_c \beta_Y$
 $\langle proof \rangle$

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

lemma *FORALL-true-implies-all-true*:
assumes *p-type*: $p : X \rightarrow \Omega$ **and** *FORALL-p-true*: $FORALL X \circ_c (p \circ_c left\text{-}cart\text{-}proj X \mathbf{1})^\sharp = t$
shows $\bigwedge x. x \in_c X \implies p \circ_c x = t$
<proof>

lemma *FORALL-true-implies-all-true2*:
assumes *p-type*[*type-rule*]: $p : X \times_c Y \rightarrow \Omega$ **and** *FORALL-p-true*: $FORALL X \circ_c p^\sharp = t \circ_c \beta Y$
shows $\bigwedge x y. x \in_c X \implies y \in_c Y \implies p \circ_c \langle x, y \rangle = t$
<proof>

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

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

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

15.2 Existential Quantification

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

lemma *EXISTS-type*[*type-rule*]:
 $EXISTS X : \Omega^X \rightarrow \Omega$
<proof>

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

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

lemma *EXISTS-elim*:

assumes *EXISTS-p-true*: $EXISTS X \circ_c (p \circ_c \text{left-cart-proj } X \mathbf{1})^\# = t$ **and** *p-type*:
 $p : X \rightarrow \Omega$
shows $(\bigwedge x. x \in_c X \implies p \circ_c x = t \implies Q) \implies Q$
 $\langle proof \rangle$

lemma *exists-true-implies-EXISTS-true*:

assumes *p-type*: $p : X \rightarrow \Omega$ **and** *exists-p-true*: $\exists x. x \in_c X \wedge p \circ_c x = t$
shows $EXISTS X \circ_c (p \circ_c \text{left-cart-proj } X \mathbf{1})^\# = t$
 $\langle proof \rangle$

end

16 Natural Number Parity and Halving

theory *Nat-Parity*

imports *Nats Quant-Logic*

begin

16.1 Nth Even Number

definition *nth-even* :: *cfunc* **where**

$nth\text{-even} = (THE\ u. u : \mathbf{N}_c \rightarrow \mathbf{N}_c \wedge$
 $u \circ_c \text{zero} = \text{zero} \wedge$
 $(\text{successor} \circ_c \text{successor}) \circ_c u = u \circ_c \text{successor})$

lemma *nth-even-def2*:

$nth\text{-even} : \mathbf{N}_c \rightarrow \mathbf{N}_c \wedge nth\text{-even} \circ_c \text{zero} = \text{zero} \wedge (\text{successor} \circ_c \text{successor}) \circ_c$
 $nth\text{-even} = nth\text{-even} \circ_c \text{successor}$
 $\langle proof \rangle$

lemma *nth-even-type*[*type-rule*]:

$nth\text{-even} : \mathbf{N}_c \rightarrow \mathbf{N}_c$
 $\langle proof \rangle$

lemma *nth-even-zero*:

$nth\text{-even} \circ_c \text{zero} = \text{zero}$
 $\langle proof \rangle$

lemma *nth-even-successor*:

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

lemma *nth-even-successor2*:

$nth\text{-even} \circ_c \text{successor} = \text{successor} \circ_c \text{successor} \circ_c nth\text{-even}$

<proof>

16.2 Nth Odd Number

definition *nth-odd* :: *cfunc* **where**

nth-odd = (*THE* *u*. *u*: $\mathbb{N}_c \rightarrow \mathbb{N}_c \wedge$
 $u \circ_c \text{zero} = \text{successor} \circ_c \text{zero} \wedge$
 $(\text{successor} \circ_c \text{successor}) \circ_c u = u \circ_c \text{successor}$)

lemma *nth-odd-def2*:

nth-odd: $\mathbb{N}_c \rightarrow \mathbb{N}_c \wedge \text{nth-odd} \circ_c \text{zero} = \text{successor} \circ_c \text{zero} \wedge (\text{successor} \circ_c \text{successor}) \circ_c \text{nth-odd} = \text{nth-odd} \circ_c \text{successor}$
<proof>

lemma *nth-odd-type*[*type-rule*]:

nth-odd: $\mathbb{N}_c \rightarrow \mathbb{N}_c$
<proof>

lemma *nth-odd-zero*:

nth-odd $\circ_c \text{zero} = \text{successor} \circ_c \text{zero}$
<proof>

lemma *nth-odd-successor*:

nth-odd $\circ_c \text{successor} = (\text{successor} \circ_c \text{successor}) \circ_c \text{nth-odd}$
<proof>

lemma *nth-odd-successor2*:

nth-odd $\circ_c \text{successor} = \text{successor} \circ_c \text{successor} \circ_c \text{nth-odd}$
<proof>

lemma *nth-odd-is-succ-nth-even*:

nth-odd = *successor* \circ_c *nth-even*
<proof>

lemma *succ-nth-odd-is-nth-even-succ*:

successor \circ_c *nth-odd* = *nth-even* \circ_c *successor*
<proof>

16.3 Checking if a Number is Even

definition *is-even* :: *cfunc* **where**

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

lemma *is-even-def2*:

is-even : $\mathbb{N}_c \rightarrow \Omega \wedge \text{is-even} \circ_c \text{zero} = \text{t} \wedge \text{NOT} \circ_c \text{is-even} = \text{is-even} \circ_c \text{successor}$
<proof>

lemma *is-even-type*[*type-rule*]:

is-even : $\mathbb{N}_c \rightarrow \Omega$
<proof>

lemma *is-even-zero*:

$is\text{-}even \circ_c zero = t$
 $\langle proof \rangle$

lemma *is-even-successor*:

$is\text{-}even \circ_c successor = NOT \circ_c is\text{-}even$
 $\langle proof \rangle$

16.4 Checking if a Number is Odd

definition *is-odd* :: *cfunc* **where**

$is\text{-}odd = (THE\ u.\ u : \mathbb{N}_c \rightarrow \Omega \wedge u \circ_c zero = f \wedge NOT \circ_c u = u \circ_c successor)$

lemma *is-odd-def2*:

$is\text{-}odd : \mathbb{N}_c \rightarrow \Omega \wedge is\text{-}odd \circ_c zero = f \wedge NOT \circ_c is\text{-}odd = is\text{-}odd \circ_c successor$
 $\langle proof \rangle$

lemma *is-odd-type*[*type-rule*]:

$is\text{-}odd : \mathbb{N}_c \rightarrow \Omega$
 $\langle proof \rangle$

lemma *is-odd-zero*:

$is\text{-}odd \circ_c zero = f$
 $\langle proof \rangle$

lemma *is-odd-successor*:

$is\text{-}odd \circ_c successor = NOT \circ_c is\text{-}odd$
 $\langle proof \rangle$

lemma *is-even-not-is-odd*:

$is\text{-}even = NOT \circ_c is\text{-}odd$
 $\langle proof \rangle$

lemma *is-odd-not-is-even*:

$is\text{-}odd = NOT \circ_c is\text{-}even$
 $\langle proof \rangle$

lemma *not-even-and-odd*:

assumes $m \in_c \mathbb{N}_c$
shows $\neg(is\text{-}even \circ_c m = t \wedge is\text{-}odd \circ_c m = t)$
 $\langle proof \rangle$

lemma *even-or-odd*:

assumes $n \in_c \mathbb{N}_c$
shows $is\text{-}even \circ_c n = t \vee is\text{-}odd \circ_c n = t$
 $\langle proof \rangle$

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

$is\text{-}even \circ_c nth\text{-}even = t \circ_c \beta_{\mathbf{N}_c}$
 ⟨proof⟩

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

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

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

lemma *EXISTS-zero-nth-even*:
 $(EXISTS \mathbf{N}_c \circ_c (eq\text{-}pred \mathbf{N}_c \circ_c nth\text{-}even \times_f id_c \mathbf{N}_c)^\#) \circ_c zero = t$
 ⟨proof⟩

lemma *not-EXISTS-zero-nth-odd*:
 $(EXISTS \mathbf{N}_c \circ_c (eq\text{-}pred \mathbf{N}_c \circ_c nth\text{-}odd \times_f id_c \mathbf{N}_c)^\#) \circ_c zero = f$
 ⟨proof⟩

16.5 Natural Number Halving

definition *halve-with-parity* :: *cfunc* **where**
 $halve\text{-}with\text{-}parity = (THE \ u. \ u: \mathbf{N}_c \rightarrow \mathbf{N}_c \coprod \mathbf{N}_c \wedge$
 $u \circ_c zero = left\text{-}coproj \ \mathbf{N}_c \ \mathbf{N}_c \circ_c zero \wedge$
 $(right\text{-}coproj \ \mathbf{N}_c \ \mathbf{N}_c \ \Pi \ (left\text{-}coproj \ \mathbf{N}_c \ \mathbf{N}_c \circ_c successor)) \circ_c u = u \circ_c successor)$

lemma *halve-with-parity-def2*:
 $halve\text{-}with\text{-}parity : \mathbf{N}_c \rightarrow \mathbf{N}_c \coprod \mathbf{N}_c \wedge$
 $halve\text{-}with\text{-}parity \circ_c zero = left\text{-}coproj \ \mathbf{N}_c \ \mathbf{N}_c \circ_c zero \wedge$
 $(right\text{-}coproj \ \mathbf{N}_c \ \mathbf{N}_c \ \Pi \ (left\text{-}coproj \ \mathbf{N}_c \ \mathbf{N}_c \circ_c successor)) \circ_c halve\text{-}with\text{-}parity =$
 $halve\text{-}with\text{-}parity \circ_c successor$
 ⟨proof⟩

lemma *halve-with-parity-type[type-rule]*:
 $halve\text{-}with\text{-}parity : \mathbf{N}_c \rightarrow \mathbf{N}_c \coprod \mathbf{N}_c$
 ⟨proof⟩

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

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

lemma *halve-with-parity-nth-even*:

$halve-with-parity \circ_c nth-even = left-coproj \mathbf{N}_c \mathbf{N}_c$
<proof>

lemma *halve-with-parity-nth-odd*:

$halve-with-parity \circ_c nth-odd = right-coproj \mathbf{N}_c \mathbf{N}_c$
<proof>

lemma *nth-even-nth-odd-halve-with-parity*:

$(nth-even \amalg nth-odd) \circ_c halve-with-parity = id \mathbf{N}_c$
<proof>

lemma *halve-with-parity-nth-even-nth-odd*:

$halve-with-parity \circ_c (nth-even \amalg nth-odd) = id (\mathbf{N}_c \amalg \mathbf{N}_c)$
<proof>

lemma *even-odd-iso*:

isomorphism $(nth-even \amalg nth-odd)$
<proof>

lemma *halve-with-parity-iso*:

isomorphism $halve-with-parity$
<proof>

definition *halve* :: *cfunc* **where**

$halve = (id \mathbf{N}_c \amalg id \mathbf{N}_c) \circ_c halve-with-parity$

lemma *halve-type[type-rule]*:

$halve : \mathbf{N}_c \rightarrow \mathbf{N}_c$
<proof>

lemma *halve-nth-even*:

$halve \circ_c nth-even = id \mathbf{N}_c$
<proof>

lemma *halve-nth-odd*:

$halve \circ_c nth-odd = id \mathbf{N}_c$
<proof>

lemma *is-even-def3*:

$is-even = ((t \circ_c \beta_{\mathbf{N}_c}) \amalg (f \circ_c \beta_{\mathbf{N}_c})) \circ_c halve-with-parity$
<proof>

lemma *is-odd-def3*:

$is-odd = ((f \circ_c \beta_{\mathbf{N}_c}) \amalg (t \circ_c \beta_{\mathbf{N}_c})) \circ_c halve-with-parity$
<proof>

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

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

lemma *is-even-exists-nth-even*:
assumes $is\text{-even } \circ_c n = t$ **and** $n\text{-type}[type\text{-rule}] : n \in_c \mathbf{N}_c$
shows $\exists m. m \in_c \mathbf{N}_c \wedge n = nth\text{-even } \circ_c m$
 $\langle proof \rangle$

lemma *is-odd-exists-nth-odd*:
assumes $is\text{-odd } \circ_c n = t$ **and** $n\text{-type}[type\text{-rule}] : n \in_c \mathbf{N}_c$
shows $\exists m. m \in_c \mathbf{N}_c \wedge n = nth\text{-odd } \circ_c m$
 $\langle proof \rangle$

end

17 Cardinality and Finiteness

theory *Cardinality*
imports *Exponential-Objects*
begin

The definitions below correspond to Definition 2.6.1 in Halvorson.

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

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

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

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**
 $X \leq_c Y \longleftrightarrow (\exists m. m : X \rightarrow Y \wedge monomorphism\ m)$

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

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

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

lemma *all-emptysets-are-finite:*

assumes *is-empty* X

shows *is-finite* X

<proof>

lemma *emptyset-is-smallest-set:*

$\emptyset \leq_c X$

<proof>

lemma *truth-set-is-finite:*

is-finite Ω

<proof>

lemma *smaller-than-finite-is-finite:*

assumes $X \leq_c Y$ *is-finite* Y

shows *is-finite* X

<proof>

lemma *larger-than-infinite-is-infinite:*

assumes $X \leq_c Y$ *is-infinite* X

shows *is-infinite* Y

<proof>

lemma *iso-pres-finite:*

assumes $X \cong Y$

assumes *is-finite* X

shows *is-finite* Y

<proof>

lemma *not-finite-and-infinite:*

$\neg(\text{is-finite } X \wedge \text{is-infinite } X)$

<proof>

lemma *iso-pres-infinite:*

assumes $X \cong Y$

assumes *is-infinite* X

shows *is-infinite* Y

<proof>

lemma *size-2-sets:*

$(X \cong \Omega) = (\exists x1. \exists x2. x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2 \wedge (\forall x. x \in_c X \longrightarrow x = x1 \vee x = x2))$

<proof>

lemma *size-2plus-sets:*

$(\Omega \leq_c X) = (\exists x1. \exists x2. x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2)$

<proof>

lemma *not-init-not-term*:

$(\neg(\text{initial-object } X) \wedge \neg(\text{terminal-object } X)) = (\exists x1. \exists x2. x1 \in_c X \wedge x2 \in_c X \wedge x1 \neq x2)$
 ⟨proof⟩

lemma *sets-size-3-plus*:

$(\neg(\text{initial-object } X) \wedge \neg(\text{terminal-object } X) \wedge \neg(X \cong \Omega)) = (\exists x1. \exists x2. \exists x3. x1 \in_c X \wedge x2 \in_c X \wedge x3 \in_c X \wedge x1 \neq x2 \wedge x2 \neq x3 \wedge x1 \neq x3)$
 ⟨proof⟩

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

lemma *smaller-than-coproduct1*:

$X \leq_c X \coprod Y$
 ⟨proof⟩

lemma *smaller-than-coproduct2*:

$X \leq_c Y \coprod X$
 ⟨proof⟩

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

lemma *smaller-than-product1*:

assumes *nonempty* Y
shows $X \leq_c X \times_c Y$
 ⟨proof⟩

lemma *smaller-than-product2*:

assumes *nonempty* Y
shows $X \leq_c Y \times_c X$
 ⟨proof⟩

lemma *coprod-leq-product*:

assumes *X-not-init*: $\neg(\text{initial-object}(X))$
assumes *Y-not-init*: $\neg(\text{initial-object}(Y))$
assumes *X-not-term*: $\neg(\text{terminal-object}(X))$
assumes *Y-not-term*: $\neg(\text{terminal-object}(Y))$
shows $X \coprod Y \leq_c X \times_c Y$
 ⟨proof⟩

lemma *prod-leq-exp*:

assumes $\neg \text{terminal-object } Y$
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 $\neg(\text{terminal-object } X)$
assumes $\neg(\text{initial-object } X)$
shows $\Omega \leq_c X$
 $\langle \text{proof} \rangle$

lemma *exp-preserves-card1*:
assumes $A \leq_c B$
assumes *nonempty* X
shows $X^A \leq_c X^B$
 $\langle \text{proof} \rangle$

lemma *exp-preserves-card2*:
assumes $A \leq_c B$
shows $A^X \leq_c B^X$
 $\langle \text{proof} \rangle$

lemma *exp-preserves-card3*:
assumes $A \leq_c B$
assumes $X \leq_c Y$
assumes *nonempty* X
shows $X^A \leq_c Y^B$
 $\langle \text{proof} \rangle$

end

18 Countable Sets

theory *Countable*
imports *Nats Axiom-Of-Choice Nat-Parity Cardinality*
begin

The definition below corresponds to Definition 2.6.9 in Halvorson.

definition *epi-countable* :: *cset* \Rightarrow *bool* **where**
epi-countable $X \iff (\exists f. f : \mathbb{N}_c \rightarrow X \wedge \text{epimorphism } f)$

lemma *emptyset-is-not-epi-countable*:
 $\neg \text{epi-countable } \emptyset$
 $\langle \text{proof} \rangle$

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

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

lemma *epi-countable-is-countable*:

assumes *epi-countable* X
shows *countable* X
<proof>

lemma *emptyset-is-countable*:
countable \emptyset
<proof>

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

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

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

lemma *iso-pres-countable*:
assumes $X \cong Y$ *countable* Y
shows *countable* X
<proof>

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

The lemma below corresponds to Exercise 2.6.11 in Halvorson.

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

end

19 Fixed Points and Cantor's Theorems

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

The definitions below correspond to Definition 2.6.12 in Halvorson.

definition *fixed-point* :: *cfunc* \Rightarrow *cfunc* \Rightarrow *bool* **where**
fixed-point a $g \iff (\exists A. g : A \rightarrow A \wedge a \in_c A \wedge g \circ_c a = a)$

definition *has-fixed-point* :: *cfunc* \Rightarrow *bool* **where**

has-fixed-point $g \iff (\exists a. \text{fixed-point } a \ g)$
definition *fixed-point-property* :: *cset* \Rightarrow *bool* **where**
fixed-point-property $A \iff (\forall g. g : A \rightarrow A \longrightarrow \text{has-fixed-point } g)$

lemma *fixed-point-def2*:
assumes $g : A \rightarrow A \ a \in_c A$
shows $\text{fixed-point } a \ g = (g \circ_c a = a)$
 $\langle \text{proof} \rangle$

The lemma below corresponds to Theorem 2.6.13 in Halvorson.

lemma *Lawveres-fixed-point-theorem*:
assumes $p\text{-type}[type\text{-rule}] : p : X \rightarrow A^X$
assumes $p\text{-surj} : \text{surjective } p$
shows *fixed-point-property* A
 $\langle \text{proof} \rangle$

The theorem below corresponds to Theorem 2.6.14 in Halvorson.

theorem *Cantors-Negative-Theorem*:
 $\nexists s. s : X \rightarrow \mathcal{P} X \wedge \text{surjective } s$
 $\langle \text{proof} \rangle$

The theorem below corresponds to Exercise 2.6.15 in Halvorson.

theorem *Cantors-Positive-Theorem*:
 $\exists m. m : X \rightarrow \Omega^X \wedge \text{injective } m$
 $\langle \text{proof} \rangle$

The corollary below corresponds to Corollary 2.6.16 in Halvorson.

corollary
 $X \leq_c \mathcal{P} X \wedge \neg (X \cong \mathcal{P} X)$
 $\langle \text{proof} \rangle$

corollary *Generalized-Cantors-Positive-Theorem*:
assumes $\neg \text{terminal-object } Y$
assumes $\neg \text{initial-object } Y$
shows $X \leq_c Y^X$
 $\langle \text{proof} \rangle$

corollary *Generalized-Cantors-Negative-Theorem*:
assumes $\neg \text{initial-object } X$
assumes $\neg \text{terminal-object } Y$
shows $\nexists s. s : X \rightarrow Y^X \wedge \text{surjective } s$
 $\langle \text{proof} \rangle$

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.