A Formalisation of the Elementary Theory of the Category of Sets in Isabelle / HOL

Dustin Bryant[1][0000−0001−9876−3804] and James Baxter[2][0000−0001−6083−9607]

[1] Independent
brydustin@gmail.com
https://www.linkedin.com/in/dustin-bryant-79122425/
[2] University of York
james.baxter@york.ac.uk
https://www-users.york.ac.uk/~jeb531/

Abstract. The Elementary Theory of the Category of Sets (ETCS), introduced by Lawvere as a categorical alternative to ZFC, prioritizes the *form* rather than the *substance* of mathematical objects, most akin to common mathematical reasoning. It is thus currently gaining popularity as an alternate foundation for mathematics. However, there is still much to be learned about ETCS and its metatheoretic results. This paper is an aid to the logical framework by providing a comprehensive and precise axiomatization ETCS within Isabelle/HOL. After introducing the axioms, we develop its internal logic system which allows us to reason internally and develop complicated expressions such as those of arithmetic. We further expand on Lawvere's Primitive Recursion Theorem by proving the general case. The resulting development is a concrete aid in the study of the metatheory.

Keywords: Formal verification · Isabelle/HOL · ETCS · Elementary Theory of the Category of Sets · Category Theory · Axioms · Mathematical Foundations

1 Introduction

The Elementary Theory of the Category of Sets (ETCS), originally proposed by Lawvere [5], provides an axiomatization of sets and the functions between them through the lens of category theory. Unlike classical set theories such as ZFC, which characterize mathematics purely in terms of sets and set membership, ETCS treats functions as fundamental entities. Consequently, ETCS aligns closely with the intuitions of practicing mathematicians, who naturally distinguish between objects and the morphisms relating them. Leinster [6] emphasizes this intuitive appeal, noting that within standard set theories, conceptual anomalies emerge—one might awkwardly question what the "elements" of transcendental numbers such as π actually are, whereas ETCS elegantly sidesteps this difficulty by clearly delineating objects from their morphisms.

While ETCS provides an elegant and intuitive foundation, recent discussions have highlighted important foundational questions[citation needed!]. Notably, ongoing debate surrounds whether ETCS is fully equivalent or merely closely analogous to ZFC in terms of expressivity and foundational strength. Specifically, a major open question remains whether ETCS and ZFC are bi-interpretable or whether subtle distinctions emerge between the theories when carefully examined from a formal standpoint. Investigations into these potential equivalences or distinctions are critical, as they may significantly affect the acceptance and application of categorical foundations in broader mathematical practice.

This paper presents and clarifies the axioms of ETCS in a structured manner. In particular, we present an unambiguous formaluation of the axioms in Isabelle/HOL and highlight some of their major consequences.

2 The Axioms

Our presentation follows the style as presented by Halvorson in *The Logic in Philosophy of Science*.

2.1 Sets is a Category

The first axiom simply establishes that **Sets** is indeed a category, that is, it consists of two kids of things: objects, which we call sets, and arrows, which we call functions. In order to independently ground this theory within Isabelle we declare that our theory contains two new types:

```
typedecl cset
typedecl cfunc
```

Listing 1.1. The two sorts of things in ETCS

For practical purposes, we used these names to avoid ambiguity with other existing types in the Isabelle ecosystem, and we note that the c prefix here is intended to stand for *category*, and emphasises that these are category-theoretic objects.

Further we stipulate that to each function f there is a unique pair of sets, the domain and codomain, and we write $f : X \to Y$ to mean that X is the domain of f and Y is the codomain of f. For any two functions $f : X \to Y$ and $g : Y \to Z$ in **Sets**, there is a function $g \circ f : X \to Z$ in **Sets** such that $h \circ (g \circ f) = (h \circ g) \circ f$ for any functions f, g and h. We often write gf for $g \circ f$. al For each set X in **Sets**, there is a function $\mathrm{id}\, X : X \to X$ such that $\mathrm{id}\, X \circ f = f$ for any function f with codomain X and $g \circ \mathrm{id}\, X = g$ for any function g with domain X. Formally, we represent this in Isabelle as:

```
axiomatization
  domain :: "cfunc ⇒ cset" and
  codomain :: "cfunc ⇒ cset" and
  comp :: "cfunc ⇒ cfunc ⇒ cfunc" (infixr "∘" 55) and
```

```
  id :: "cset ⇒ cfunc" ("id_c")
where
  domain_comp: "domain g = codomain f ⟹ domain (g ∘ f) =
    domain f" and
  codomain_comp: "domain g = codomain f ⟹ codomain (g ∘ f) =
    codomain g" and
  comp_associative: "domain h = codomain g ⟹ domain g =
    codomain f ⟹ h ∘ (g ∘ f) = (h ∘ g) ∘ f" and
  id_domain: "domain (id X) = X" and
  id_codomain: "codomain (id X) = X" and
  id_right_unit: "f ∘ id (domain f) = f" and
  id_left_unit: "id (codomain f) ∘ f = f"
```

Listing 1.2. Sets is a category

We refer to $f : X \to Y$ as the *type* of f *internal* to our theory; within the context of Isabelle *externally*, of course, f is merely of type *cfunc*. In the sequel, a function's type refers to its internal type. Formally, we make this shorthand as follows:

```
definition cfunc_type :: "cfunc ⇒ cset ⇒ cset ⇒ bool" ("_ : _
    → _" [50, 50, 50]50) where
  "(f : X → Y) ⟷ (domain f = X ∧ codomain f = Y)"
```

Listing 1.3. A function's type

We formally define a function as a *monomorphism*:

```
definition monomorphism :: "cfunc ⇒ bool" where
  "monomorphism f ⟷ (∀ g h.
    (codomain g = domain f ∧ codomain h = domain f) ⟶ (f ∘ g =
     f ∘ h ⟶ g = h))"
```

Listing 1.4. Monomorphism definition

an *epimorphism* is defined as:

```
definition epimorphism :: "cfunc ⇒ bool" where
  "epimorphism f ⟷ (∀ g h.
    (domain g = codomain f ∧ domain h = codomain f) ⟶ (g ∘ f =
     h ∘ f ⟶ g = h))"
```

Listing 1.5. Epimorphism definition

and an *isomorphism* is:

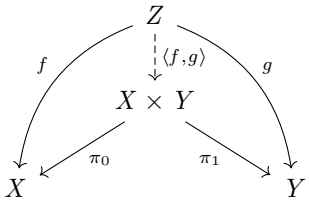
```
definition isomorphism :: "cfunc ⇒ bool" where
  "isomorphism f ⟷ (∃ g. domain g = codomain f ∧ codomain g =
    domain f ∧
    g ∘ f = id(domain f) ∧ f ∘ g = id(domain g))"
```

Listing 1.6. Isomorphism definition

2.2 Cartesian Products

For any sets X and Y in **Sets**, there is a set $X \times Y$ and functions $\pi_0 : X \times Y \to X$ and $\pi_1 : X \times Y \to Y$ in **Sets**, such that for any functions $f : Z \to X$ and $g : Z \to Y$, there is a unique function $\langle f, g \rangle : Z \to X \times Y$ such that $\pi_0 \circ \langle f, g \rangle = f$ and $\pi_1 \circ \langle f, g \rangle = g$.

As is usual in category theory, it is frequently helpful to express laws in the form of commuting diagrams. Axiom 2 can be expressed in such a diagram as shown below.



When we want to be completely unambiguous of the types for the projections, we may write $\pi_0^{X \times Y}$ or $\pi_1^{X \times Y}$. Formally, we can state this axiom as:

```
axiomatization
  cart_prod :: "cset ⇒ cset ⇒ cset" (infixr "×" 65) and
  left_cart_proj :: "cset ⇒ cset ⇒ cfunc" and
  right_cart_proj :: "cset ⇒ cset ⇒ cfunc" and
  cfunc_prod :: "cfunc ⇒ cfunc ⇒ cfunc" ("⟨_,_⟩")
where
  left_cart_proj_type[type_rule]: "left_cart_proj X Y : X × Y → X" and
  right_cart_proj_type[type_rule]: "right_cart_proj X Y : X × Y → Y" and
  cfunc_prod_type[type_rule]: "f : Z → X ⟹ g : Z → Y ⟹ ⟨f,g⟩ : Z → X × Y" and
  left_cart_proj_cfunc_prod: "f : Z → X ⟹ g : Z → Y ⟹
    left_cart_proj X Y ∘ ⟨f,g⟩ = f" and
  right_cart_proj_cfunc_prod: "f : Z → X ⟹ g : Z → Y ⟹
    right_cart_proj X Y ∘ ⟨f,g⟩ = g" and
  cfunc_prod_unique: "f : Z → X ⟹ g : Z → Y ⟹ h : Z → X × Y ⟹
    left_cart_proj X Y ∘ h = f ⟹ right_cart_proj X Y ∘ h = g
    ⟹ h = ⟨f,g⟩"
```

Listing 1.7. Cartesian Products and projections exist

Given $f : W \to Y, g : X \to Z$, it is particularly helpful to define

$$f \times g = \langle f\pi_0^{W \times X}, g\pi_1^{W \times X} \rangle$$

which is the unique function such that

$$\pi_0^{Y \times Z}(f \times g) = f\pi_0^{W \times X}, \quad \pi_1^{Y \times Z}(f \times g) = g\pi_1^{W \times X}$$

This is formally defined as:

```
definition cfunc_cross_prod :: "cfunc ⇒ cfunc ⇒ cfunc" (infixr
    "×" 55) where
  "f × g = ⟨f ∘ left_cart_proj (domain f) (domain g), g ∘
    right_cart_proj (domain f) (domain g)⟩"

lemma cfunc_cross_prod_def2:
  assumes "f : X → Y" "g : V→ W"
  shows "f × g = ⟨f ∘ left_cart_proj X V, g ∘ right_cart_proj X
    V⟩"
  using assms cfunc_cross_prod_def cfunc_type_def by auto

lemma cfunc_cross_prod_type[type_rule]:
  "f : W → Y ⟹ g : X → Z ⟹ f × g : W × X → Y × Z"
  unfolding cfunc_cross_prod_def
  using cfunc_prod_type cfunc_type_def comp_type
    left_cart_proj_type right_cart_proj_type by auto"
```

Listing 1.8. Product of Functions

Often when working in ETCS one finds half the work is simply getting the types right when defining new functions; in particular, as the operations we work with become more complicated we sometimes need to shuffle arguments so they live in the correct domain where we may apply our function of interest. For example, if we have an element $\langle a, \langle b, c\rangle\rangle$ but need it to look like $\langle\langle a, b\rangle, c\rangle$, then what we need is to precompose it with $\langle\langle\pi_0, \pi_0 \circ \pi_1\rangle, \pi_1 \circ \pi_1\rangle$. Of course, having to fit this into the definition of a function of interest every time it is needed is quite cumbersome so instead we defined the reuseable associate_left that does precisely this:

```
definition associate_left :: "cset ⇒ cset ⇒ cset ⇒ cfunc"
    where
  "associate_left X Y Z =
    ⟨
      ⟨
        left_cart_proj X (Y × Z),
        left_cart_proj Y Z ∘ right_cart_proj X (Y × Z)
      ⟩,
      right_cart_proj Y Z ∘ right_cart_proj X (Y × Z)
    ⟩"
```

Listing 1.9. Associate left helper function

In a similar fashion we defined associate_right.

2.3 Terminal Objects

The axioms of ETCS frequently make use of concepts from category theory. One such concept is the notion of a terminal object, which is an object that has a unique morphism from each object to itself. Axiom 3 asserts the existence of

such an object in ETCS, a set which we refer to as **1**. It also asserts that **1** is a separator, which means functions are judged to be the same when their compositions with functions having a domain of **1** are all the same.

Sets contains a set **1** such that, for any set X, there is a unique function $\beta_X : X \to \mathbf{1}$. Furthermore, for any functions $f, g : X \to Y$, if $f \circ x = g \circ x$ for all functions $x : \mathbf{1} \to X$, then $f = g$.

```
axiomatization
  terminal_func :: "cset ⇒ cfunc" ("β_" 100) and
  one_set :: "cset" ("1")
where
  terminal_func_type[type_rule]: "β_X : X → 1" and
  terminal_func_unique: "h :  X → 1 ⟹ h = β_X" and
  one_separator: "f : X → Y ⟹ g : X → Y ⟹ (⋀ x. x : 1 → X
    ⟹ f ∘ x = g ∘ x) ⟹ f = g"
```

Listing 1.10. Terminal Objects Exist

The set **1** may be seen intuitively as a single element set. Since a function from a single element set into any other set X identifies exactly one element of X, we take them to represent elements of X and write $x \in X$ for $x : \mathbf{1} \to X$.

```
abbreviation member :: "cfunc ⇒ cset ⇒ bool" (infix "∈" 50)
    where
  "x ∈ X ≡ (x : 1 → X)"
```

Listing 1.11. Element-of notation

It then follows from the uniqueness of $\beta_{\mathbf{1}}$, that **1** does indeed have a single element, which must be equal to both $\beta_{\mathbf{1}}$ and $\mathrm{id}_{\mathbf{1}}$.

```
lemma element_of_1:
  "x ∈ 1  ⟹ x = id 1"
  by (metis id_type terminal_func_unique)
```

Listing 1.12. The element of 1

Thus, while ETCS does not start with an axiomatization of set membership, an intuitive notion of set membership can be defined within it. Application of a function to an element of its domain can then be simply defined by composing the function with the element.

The functions β_X are useful for defining constant functions, since, for any function $y : \mathbf{1} \to Y$, $y \circ \beta_X : X \to Y$ represents the function that yields y when applied to any $x : \mathbf{1} \to X$. The separator property in Axiom 3 is also occasionally useful in proofs since it allows us to apply a form of function extensionality.

2.4 Equalizers

Axiom 4 asserts the existence of equalizers in ETCS, that is, a set E and a function $m : E \to X$ that makes functions $f, g : X \to Y$ equal when composed

with them. That is, for any functions $f, g : X \to Y$ in **Sets**, there is a set E and function $m : E \to X$ in **Sets** such that $fm = gm$ and, for any other set F and function $h : F \to X$, there is a unique function $k : F \to E$ such that $mk = h$. We formally define equalizers and declare their existence as follows:

```
definition equalizer :: "cset ⇒ cfunc ⇒ cfunc ⇒ cfunc ⇒ bool"
     where
  "equalizer E m f g ⟷ (∃ X Y. (f : X → Y) ∧ (g : X → Y) ∧ (m
    : E → X)
    ∧ (f ∘ m = g ∘ m)
    ∧ (∀ h F. ((h : F → X) ∧ (f ∘ h = g ∘ h)) ⟶ (∃! k. (k : F
    → E) ∧ m ∘ k = h)))"
axiomatization where
  equalizer_exists: "f : X → Y ⟹ g : X → Y ⟹ ∃ E m. equalizer
     E m f g"
```

Listing 1.13. Equalizers Exist

It is also the case that there is a mapping between different equalizers of the same two functions. From this fact it can be shown that any equalizer is a monomorphism, and hence defines a subobject. Axiom 4 is thus useful in that it allows us to define subsets of the form $\{x \in X \mid f \circ x = g \circ x\}$ for any functions $f, g : X \to Y$.

2.5 Truth Object

Axiom 5 builds an internal Boolean type which allows us to, together with the previous axiom, evaluate predicates. Later, we will show that we can also build a more extensive internal logic system so that predicates are true *internally* if and only if they are true *externally* within our metalogic. We declare there is a set Ω with the following features:

1. Ω has exactly two elements, $\mathtt{t} : 1 \to \Omega$ and $\mathtt{f} : 1 \to \Omega$.
2. For any set X, and subobject $m : B \to X$, there is a unique function $\chi_B : X \to \Omega$ such that the following diagram is a pullback:

$$\begin{array}{ccc} B & \longrightarrow & 1 \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle \mathtt{t}} \\ X & \xrightarrow{\chi_B} & \Omega \end{array}$$

Informally, $B = \{x \in X : \chi_B(x) = \mathtt{t}\}$. Now we state this axiom formally:

```
axiomatization
  true_func :: "cfunc" ("t") and
  false_func  :: "cfunc" ("f") and
  truth_value_set :: "cset" ("Ω")
where
  true_func_type[type_rule]: "t ∈ Ω" and
```

```
  false_func_type[type_rule]: "f ∈ Ω" and
  true_false_distinct: "t ≠ f" and
  true_false_only_truth_values: "x ∈ Ω ⟹ x = t ∨ x = f"
    and
  characteristic_function_exists:
    "m : B → X ⟹ monomorphism m ⟹ ∃! χ. is_pullback B 1 X
     Ω (β_B) t m χ"
```

Listing 1.14. Boolean Set exists

We define *the* characteristic function of the *monomorphism* $m : B \to X$ as the χ_B which makes a diagram like the one above a pullback. Notice the following particularly useful "converse" to this axiom. Given a map $\chi : X \to \Omega$, we can always construct a subobject (B_χ, m) of X by taking it as the equalizer of the maps $\chi, \mathtt{t} \circ \beta_X : X \to \Omega$ from the last axiom. Moreover, if one applies the Truth axiom it is immediate that $\chi_{B_\chi} = \chi$.

2.6 Equivalence Classes Exist

An equivalence class on X is a certain sort of relation (i.e. subset of $X \times X$) which we will posit to exist. Recall, an equivalence relation is a relation which is reflexive, symmetric, and transitive. Given an equivalence relation R on X, and some $x \in X$, we let $[x]_R := \{y \in X : \langle x, y\rangle \in R\}$; we say that $[x]$ is *the equivalence class* of x. Formally, we represent these definitions as follows:

```
definition reflexive_on :: "cset ⇒ cset × cfunc ⇒ bool" where
  "reflexive_on X R = (R ⊆ X×X ∧
    (∀x. x ∈ X ⟶ (⟨x,x⟩ ∈_X×X R)))"

definition symmetric_on :: "cset ⇒ cset × cfunc ⇒ bool" where
  "symmetric_on X R = (R  ⊆ X × X ∧
    (∀x y. x ∈ X ∧  y ∈ X ⟶
      (⟨x,y⟩ ∈_X×X R ⟶ ⟨y,x⟩ ∈_X×X R)))"

definition transitive_on :: "cset ⇒ cset × cfunc ⇒ bool" where
   (∀x y z. x ∈ X ∧  y ∈ X ∧ z ∈ X  ⟶
      (⟨x,y⟩ ∈ X×X R ∧ ⟨y,z⟩ ∈ X×X R ⟶ ⟨x,z⟩ ∈_X×X R)))"

definition equiv_rel_on :: "cset ⇒ cset × cfunc ⇒ bool" where
  "equiv_rel_on X R  ⟷ (reflexive_on X R ∧ symmetric_on X R
    ∧ transitive_on X R)"

definition const_on_rel :: "cset ⇒ cset × cfunc ⇒ cfunc ⇒ bool
    " where
  "const_on_rel X R f = (∀x y. x ∈ X ⟶ y ∈ X ⟶ ⟨x, y⟩ ∈_X×X
    R ⟶ f ∘ x = f ∘ y)"
```

Listing 1.15. Equivalence Relation Definition

Our axiom says there is a set X/R, which is the collection of all such equivalence classes $[x]_R$ for every $x \in X$, and that there always exists a canonical mapping $q_R : X \to X/R$ such that $x \mapsto [x]$. Informally, $X/R = \{[x] : x \in X\}$. Below we state the formal version of this axiom, please note that (`equiv_class` R) corresponds to our informal notation q_R.

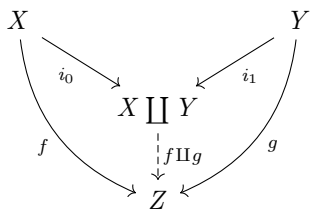
```
axiomatization
  quotient_set :: "cset ⇒ (cset × cfunc) ⇒ cset"  and
  equiv_class :: "(cset × cfunc) ⇒ cfunc" and
  quotient_func :: "cfunc ⇒ (cset × cfunc) ⇒ cfunc"
where
  equiv_class_type[type_rule]: "equiv_rel_on X R ⟹ equiv_class
     R : X → quotient_set X R" and
  equiv_class_eq: "equiv_rel_on X R ⟹ (⟨x, y⟩ ∈ X×X ⟹
    (⟨x, y⟩ ∈_X×X R) ⟷ equiv_class R ∘ x = equiv_class R ∘ y)"
     and
  quotient_func_type[type_rule]: "equiv_rel_on X R ⟹ f : X → Y
     ⟹ (const_on_rel X R f) ⟹
      quotient_func f R : quotient_set X R → Y" and
  quotient_func_eq: "equiv_rel_on X R ⟹ f : X → Y ⟹ (
    const_on_rel X R f) ⟹
     quotient_func f R ∘ equiv_class R = f" and
  quotient_func_unique: "equiv_rel_on X R ⟹ f : X → Y ⟹ (
    const_on_rel X R f) ⟹
    h : quotient_set X R → Y ⟹ h ∘ equiv_class R = f ⟹ h =
    quotient_func f R"

abbreviation equiv_class' :: "cfunc ⇒ cset × cfunc ⇒ cfunc" ("
    [_]_") where
  "[x]_R ≡ equiv_class R ∘ x"
```

Listing 1.16. Equivalence Classes and Quotient Mappings Exist

2.7 Coproducts

The dual of Cartesian products is coproducts, which informally may be viewed as representing the disjoint union of two sets. Axiom 7 asserts the existence of coproducts in ETCS and provides coprojection functions i_0 and i_1 to map into each side of a coproduct. For any sets X and Y in **Sets**, there is a set $X \coprod Y$ and functions $i_0 : X \to X \coprod Y$ and $i_1 : Y \to X \coprod Y$ in **Sets**, such that for any functions $f : X \to Z$ and $g : Y \to Z$, there is a unique function $f \amalg g : X \coprod Y \to Z$ such that $(f \amalg g) \circ i_0 = f$ and $(f \amalg g) \circ i_1 = g$.

We can summarize this axiom with the following diagram:



Formally, we state this axiom as:

```
axiomatization
  coprod :: "cset ⇒ cset ⇒ cset" (infixr "∐" 65) and
  left_coproj :: "cset ⇒ cset ⇒ cfunc" and
  right_coproj :: "cset ⇒ cset ⇒ cfunc" and
  cfunc_coprod :: "cfunc ⇒ cfunc ⇒ cfunc" (infixr "∐" 65)
where
  left_proj_type[type_rule]: "left_coproj X Y : X → X∐Y" and
  right_proj_type[type_rule]: "right_coproj X Y : Y → X∐Y" and
  cfunc_coprod_type[type_rule]: "f : X → Z ⟹ g : Y → Z ⟹ f∐g
     :  X∐Y → Z" and
  left_coproj_cfunc_coprod: "f : X → Z ⟹ g : Y → Z ⟹ f∐g ∘ (
    left_coproj X Y)  = f" and
  right_coproj_cfunc_coprod: "f : X → Z ⟹ g : Y → Z ⟹ f∐g ∘
    (right_coproj X Y)  = g" and
  cfunc_coprod_unique: "f : X → Z ⟹ g : Y → Z ⟹ h : X ∐ Y →
    Z ⟹
    h ∘ left_coproj X Y = f ⟹ h ∘ right_coproj X Y = g ⟹ h =
    f ∐ g"
```

Listing 1.17. Coproducts and coprojections Exist

Dual to the product of functions, the bowtie product of functions is another very helpful object which we formally define here:

```
definition cfunc_bowtie_prod :: "cfunc ⇒ cfunc ⇒ cfunc" (
    infixr "⋈" 55) where
  "f ⋈ g = ((left_coproj (codomain f) (codomain g)) ∘ f) ∐ ((
    right_coproj (codomain f) (codomain g)) ∘ g)"

lemma cfunc_bowtie_prod_def2:
  assumes "f : X → Y" "g : V → W"
  shows "f ⋈ g = (left_coproj Y W ∘ f) ∐ (right_coproj Y W ∘ g)
    "
  using assms cfunc_bowtie_prod_def cfunc_type_def by auto

lemma cfunc_bowtie_prod_type[type_rule]:
  "f : X → Y ⟹ g : V → W ⟹ f ⋈ g : X ∐ V → Y ∐ W"
  unfolding cfunc_bowtie_prod_def
```

```
12  using cfunc_coprod_type cfunc_type_def comp_type
      left_proj_type right_proj_type by auto
```

Listing 1.18. Bowtie Product of Functions

Complements and Partial Functions Suppose we have a function $f_D : D \to B$ with $d : D \hookrightarrow A$ so that we can think of f_D as the restriction of the partial function $f : A \rightharpoonup B$ to its domain where it is defined. We would like to construct a function which is similar to f but which is total since ETCS only handles total functions; we do this by defining $\hat{f} : A \to B \coprod 1$ in such a way that $\hat{f}$ behaves like f_D on relative elements of D and sends everything else to 1, where f is undefined. Since $d^{\complement} : D^{\complement} \hookrightarrow A$ and $i_A : D \coprod D^{\complement} \to A$ with $i_A \circ i_0 = d$ and $i_A \circ i_1 = d^{\complement}$ we may define an auxilary function $h : D \coprod D^{\complement} \to B \coprod \mathbf{1}$ so that it captures the desired semantics, namely $h \circ i_0 = i_0 \circ f_D$ and $h \circ i_1 = i_1 \circ \beta_{D^{\complement}}$. To this end, we define

$$h := (i_0 \circ f_D) \amalg (i_1 \circ \beta_{D^{\complement}}) : D \coprod D^{\complement} \to B \coprod \mathbf{1}\,.$$

As i_A is an isomorphism, we can define $i_A^{-1} : A \to D \coprod D^{\complement}$, so we can transport f_D to the total operator:

$$\hat{f} = h \circ i_A^{-1} : A \to B \coprod \mathbf{1}$$

Suppose $a \in A$ factors through D then $a = d \circ x = \iota_A \circ i_0 \circ x$ for a unique $x \in D$ and

$$\begin{aligned}
\widehat{f} \circ a &= h \circ \iota_A^{-1} \circ a \\
&= h \circ \iota_A^{-1} \circ \iota_A \circ i_0 \circ x \\
&= h \circ i_0 \circ x \\
&= (i_0 \circ f_D) \circ x \\
&= i_0\big(f_D \circ x\big),
\end{aligned}$$

otherwise $a \in A$ factors through $D^{\complement}$ and $a = d^{\complement} \circ y = \iota_A \circ i_1 \circ y$ for a unique $y \in D^{\complement}$ and

$$\begin{aligned}
\widehat{f} \circ a &= h \circ \iota_A^{-1} \circ a \\
&= h \circ \iota_A^{-1} \circ \iota_A \circ i_1 \circ y \\
&= h \circ i_1 \circ y \\
&= (i_1 \circ \beta_{D^{\complement}}) \circ y \\
&= i_1\big(\beta_{D^{\complement}} \circ y\big),
\end{aligned}$$

2.8 Initial Objects

Dual to the terminal objects are the so-called initial objects. In **Sets**, we declare that there is at least one initial object, $\emptyset$, with the following properties:

1. For any set X, there is a unique function

$$\alpha_X : \emptyset \longrightarrow X$$

2. $\emptyset$ is empty – there does not exist a function x such that $x : 1 \to \emptyset$.

It is worth noting that while in ZF there is exactly one set which is empty, ETCS merely declares that there is *at least* one such set. Moreover, because ETCS lacks an axiom of extensionality we cannot show $\emptyset$ is the unique empty set in **Sets**.

We now formally state the axiom for the existence of $\emptyset$:

```
axiomatization
  initial_func :: "cset ⇒ cfunc" ("α_" 100) and
  emptyset :: "cset" ("∅")
where
  initial_func_type[type_rule]: "initial_func X :  ∅ → X" and
  initial_func_unique: "h : ∅ → X ⟹ h = initial_func X" and
  emptyset_is_empty: "¬(x ∈ ∅)"
```

Listing 1.19. An empty set exists

Formally, we define initial sets, show that $\emptyset$ is initial, and prove that initial sets are isomorphic to $\emptyset$:

```
definition initial_object :: "cset ⇒ bool" where
  "initial_object(X) ⟷ (∀ Y. ∃! f. f : X → Y)"

lemma emptyset_is_initial:
  "initial_object(∅)"
  using initial_func_type initial_func_unique initial_object_def
    by blast

lemma initial_iso_empty:
  assumes "initial_object(X)"
  shows "X ≅ ∅"
  by (metis assms cfunc_type_def comp_type emptyset_is_empty
    epi_mon_is_iso initial_object_def injective_def
    injective_imp_monomorphism is_isomorphic_def surjective_def
    surjective_is_epimorphism)
```

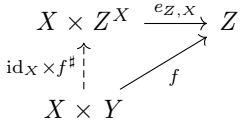
Listing 1.20. Characterizing Initial sets

2.9 Exponential Objects

In ETCS, we view functions as mappings between sets and as representing their structure. However, it is also often useful to consider a set whose elements just are functions between a pair of sets. Axiom 9 provides for the existence of exponential objects of the form Y^X, which can informally be thought of as the set of functions

from X to Y. It also supplies a transpose operator $^\sharp$ to convert functions to members of the exponential set and an evaluation function $e_{Y,X}$ to evaluate the transposed functions as the original.

For any sets A and X in **Sets**, there is a set X^A and a function $e_{X,A} : A \times X^A \to X$ in **Sets** such that, for any set Z and function $f : A \times Z \to X$, there is a *unique* function $f^\sharp : Z \to X^A$ such that $e_{X,A} \circ (\mathrm{id}\, A \times f^\sharp) = f$. We summarize this axiom by the following commuting diagram:



Formally, we state this axiom as:

```
axiomatization
  exp_set :: "cset ⇒ cset ⇒ cset" ("_^_" [100,100]100) and
  eval_func  :: "cset ⇒ cset ⇒ cfunc" and
  transpose_func :: "cfunc ⇒ cfunc" ("_^♯" [100]100)
where
  exp_set_inj: "X^A = Y^B ⟹ X = Y ∧ A = B" and
  eval_func_type[type_rule]: "eval_func X A : A× X^A → X" and
  transpose_func_type[type_rule]: "f : A × Z → X ⟹ f^♯ : Z → X^
    A" and
  transpose_func_def: "f : A × Z → X ⟹ (eval_func X A) ∘ (id A
     × f^♯) = f" and
  transpose_func_unique:
    "f : A×Z → X ⟹ g: Z → X^A ⟹ (eval_func X A) ∘ (id A × g)
    = f ⟹ g = f^♯"
```

Listing 1.21. Characterizing Initial sets

We call $f^\sharp$ the *transpose* of f or sometimes more casually the *sharp* of f or f *sharp*. We take special note of the rule `exp_set_inj` which declares two exponential sets are equal if and only if their parts are equal. We cannot prove this as a fact without an axiom of extensionality [can we prove this?!] so we require it as part of the axiom. However, we need `exp_set_inj` to prove that $f^\flat$, the *inverse transpose* of f or f flat, is unique. Certainly, if we have that $f^\sharp$ is unique and we want $f = f^{\sharp\flat} = f^{\flat\sharp}$ then it had better be the case that $f^\flat$ is uniquely defined for f.

The following is the formal defition for $f^\flat$ and a proof that this definition is well-defined:

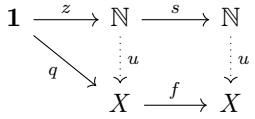
```
definition inv_transpose_func :: "cfunc ⇒ cfunc" ("_^♭"
    [100]100) where
  "f^♭ = (THE g. ∃ Z X A. domain f = Z ∧ codomain f = X^A ∧ g =
    (eval_func X A) ∘ (id A × f))"

lemma inv_transpose_func_def2:
```

```
  assumes "f : Z → X^A"
  shows "∃ Z X A. domain f = Z ∧ codomain f = X^A ∧ f^♭ = (
    eval_func X A) ∘ (id A × f)"
  unfolding inv_transpose_func_def
proof (rule theI)
  show "∃Z Y B. domain f = Z ∧ codomain f = Y^B ∧ eval_func X A
    ∘ id A × f = eval_func Y B ∘ id B × f"
    using assms cfunc_type_def by blast
next
  fix g
  assume "∃Z X A. domain f = Z ∧ codomain f = X^A ∧ g =
    eval_func X A ∘ id A × f"
  then show "g = eval_func X A ∘ id A × f"
    by (metis assms cfunc_type_def exp_set_inj)
qed

lemma inv_transpose_func_def3:
  assumes f_type: "f : Z → X^A"
  shows "f^♭ = (eval_func X A) ∘ (id A × f)"
  by (metis cfunc_type_def exp_set_inj f_type
    inv_transpose_func_def2)
```

Listing 1.22. Inverse Transpose Definition

2.10 Natural Number Object

Axiom 10 states there is an object $\mathbb{N}$, and functions $z : \mathbf{1} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ such that for any other set X with functions $q : \mathbf{1} \to X$ and $f : X \to X$, there is a unique function $u : \mathbb{N} \to X$ such that $uz = q$ and $us = fu$. We can represent this axiom by the following commuting diagram:



This object $\mathbb{N}$ is called a natural number object, we can identify its elements as the usual ordinals starting with z representing zero and obtaining the other elements through the successor operator, s. Formally, we state this axiom as follows:

```
axiomatization
  natural_numbers :: "cset" ("ℕ") and
  zero :: "cfunc" and
  successor :: "cfunc"
where
  zero_type[type_rule]: "zero ∈ ℕ" and
  successor_type[type_rule]: "successor: ℕ → ℕ" and
  natural_number_object_property:
```

```
  "q : 1 → X ⟹ f: X → X ⟹
   (∃!u. u: ℕ → X ∧
   q = u ∘ zero ∧
   f ∘ u = u ∘ successor)"
```

Listing 1.23. A Natural Number Object Exists

One major theorem about the natural numbers is that induction is possible. Formally, we state this theorem as:

```
theorem nat_induction:
  assumes predicate_type[type_rule]: "p : ℕ → Ω" and n_type[
    type_rule]: "n ∈ ℕ"
  assumes base_case: "p ∘ zero = t"
  assumes induction_case: "⋀n. n ∈ ℕ ⟹ p ∘ n = t ⟹ p ∘
    successor ∘ n = t"
  shows "p ∘ n = t"
<proof>
```

Listing 1.24. Mathematical Induction

2.11 Axiom of Choice

Our final axiom, Axiom 12, declares that every epimorphism has a left inverse. Precisely, suppose that $f : X \to Y$; we say that f is a split epimorphism just in case there is a function $s : Y \to X$ such that $fs = \mathrm{id}_Y$. In this case, we say that s is a *section* of f. The axiom of choice is: Every epimorphism in **Sets** has a section. Formally, we define split epimorphisms and declare the axiom of choice as follows:

```
definition section_of :: "cfunc ⇒ cfunc ⇒ bool" (infix "
    sectionof" 90)
  where "s sectionof f ⟷ s : codomain f → domain f ∧ f ∘ s =
    id (codomain f)"

definition split_epimorphism :: "cfunc ⇒ bool"
  where "split_epimorphism f ⟷ (∃ s.  s : codomain f → domain
    f ∧ f ∘ s = id (codomain f))"

axiomatization
  where
  axiom_of_choice: "epimorphism f ⟶ (∃ g . g sectionof f)"
```

Listing 1.25. Sections, Split Epimorphisms, and the Axiom of Choice

3 Typecheck Cfuncs

One of the major challenges in the mechanisation of ETCS within Isabelle/HOL was the repeated need to check the type of functions while applying Axiom 1.

This resulted in proofs which were incredibly long even for a proof assistant. To this end, we developed the tactic `typecheck_cfuncs` using Metalanguage (ML) within the Isabelle environment. When the user employs the proof tactic `typecheck_cfuncs`, Isabelle checks all the subexpressions that have the type `cfunc` against a list of typing rules, starting with the smallest expressions. The premises of matching typing rules are discharged using the premises of the current goal and any typing facts already proved to provide a series of lemmas giving the types of each ETCS expression. This systematic generation of typing lemmas allows for easy verification that the domain and codomain of all functions involved are compatible when applying axioms and lemmas. In a similar fashion, we developed ML to allow us to simplify the application of lemmas, with a specialized tactic for associativity of functions, so that proofs are more natural and shorter.

4 Internal Logic System

We desire to develop a robust theory of arithmetic within ETCS, and perhaps the most simple predicate about natural numbers is comparison, normally defined as:

$$n \leq m \equiv \exists\, k \in \mathbb{N}.\ n + k = m$$

But for this definition to make sense, we need $\leq$ to be able to compare elements from $\mathbb{N}$. The most obvious choice is to say $\leq : \mathbb{N} \times \mathbb{N} \to \Omega$; so that $\leq$ is an object-level truth claim and not a metalevel truth claim; that is, we should be able to derive facts of this sort inside ETCS. For example, we want to be able to prove that $\leq \langle 0, n \rangle = \mathsf{t}$ if t is supposed to in any meaningful sense correspond to truth. This keeps the theory light in the sense that we don't have to add further axioms about the ordering, rather $\leq$ *is* whatever the right-hand side is and it equals t for some pairs of elements and f for others. However, since this definition depends on $\exists$, evidently $\exists$ is some function of type $X \to \Omega$, where X is to be determined later.

Motivated by this example, it is clear that we need a rich, *internal* logical system beyond simply t and f; there should also be functions which represent negation, implication, equivalence, and qualification in the same way that t and f represent truth and falsity. The most simple logical function seems to be negation or NOT. We need to uniquely define a function $\mathsf{t} \mapsto \mathsf{f}$ and $\mathsf{f} \mapsto \mathsf{t}$. We obtain a $\text{NOT} : \Omega \to \Omega$ function by taking the characteristic function of $\mathsf{f} : \mathbf{1} \to \Omega$, such that the following diagram is a pullback by the Truth Object Axiom:

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{\beta_1} & \mathbf{1} \\ {\scriptstyle \mathsf{f}}\downarrow & & \downarrow{\scriptstyle \mathsf{t}} \\ \Omega & \xrightarrow[\text{NOT}]{} & \Omega \end{array}$$

It follows from this that $\text{NOT}(\mathsf{f}) = \mathsf{t}$. We also have that $\text{NOT}(\mathsf{t}) = \mathsf{f}$, since if it were equal to t then from the pullback property we have $\mathsf{t} = \mathsf{f}$ which is a

contradiction. Therefore, NOT(t) is *not* t and hence equal to f. Using the axiom about pullbacks is natural in the sense that given any predicate $p : X \to \Omega$, the pullback property forces

Suppose we have a predicate $p : X \to \Omega$ such that for some element $x \in X$, the property represented by p holds, then we should write $p(x) = \mathtt{t}$ to represent this in ETCS. Ordinarily, if p is true for *all* $x \in X$ then we write $\forall\, x \in X.p(x)$. We now wish to translate this quantified predicate so that it is stated inside ETCS. Because every part of the statement must be translated into a composition of functions, then $\forall$ must be a function as well, and what's more, no individual $x \in X$ can be referenced since we're having to compose actual functions. That is to say, if our ETCS version of $\forall\, x \in X.p(x)$ has "x" as part of its formula then that would be some *actual* x rather than a bounded x. Put still another way, since $\forall$ is going to be *some* sort of legitimate function with a domain and codomain, it cannot have be composed with any one element $x \in X$. Evidently, we cannot have bounded variables in our logical sentences *inside* ETCS. In short, we cannot have sentences with quantified variables, the typing just won't allow it, but this should not come as a surprise, normally $\forall$ isn't thought of as a function. But since our quantifier *is* a function $\forall : A \to B$, we should specify its type more precisely. We want $\forall$ to compose with predicates $p : X \to \Omega$, and return t just in case $p(x) = \mathtt{t}$ for all $x \in X$. Identifying Ω^X with the space of predicates on X, we need $\forall$ to return true exactly for those predicates that are true for every $x \in X$. But since there is *really* only one such function $p : X \to \Omega$, we must identify it with the singleton subobject of Ω^X which represents the constant true functions. Since $X \cong X \times \mathbf{1}$, we can use $(\mathtt{t} \circ \pi_1^{X \times 1}) : X \times \mathbf{1} \to \Omega$ to represent this constant true function, so that $(\mathtt{t} \circ \pi_1)^\sharp : \mathbf{1} \to \Omega^X$. Finally, we want $\forall$ to always evaluate *that* function as true, and we want $\forall$ to be the unique function $\forall : \Omega^X \to X$ which makes the following diagram a pullback:

$$\begin{array}{ccc} \mathbf{1} & \xrightarrow{\beta_1} & \mathbf{1} \\ {\scriptstyle (\mathtt{t}\pi_1)^\sharp}\downarrow & & \downarrow{\scriptstyle \mathtt{t}} \\ \Omega^X & \xrightarrow[\forall_X]{} & \Omega \end{array}$$

where we write $\forall_X$ to emphasize the type of $\forall$. We note that the more natural choice of the constant true function on X is $\mathtt{t} \circ \beta_X : X \to \Omega$ however there is no natural way to turn this into an element of Ω^X like there is by sharping $\mathtt{t} \circ \pi_1$. We can justify the use of $\mathtt{t} \circ \pi_1$ with the following important theorem. Let $p : X \to \Omega$ be an arbitrary predicate, then:

$$(\forall_X \circ (p\pi_0)^\sharp = \mathtt{t}) \text{ if and only if } (\forall\, x \in X.\ p(x) = \mathtt{t})$$

where $\forall$ on the right hand side is a logical symbol of the metalogic *outside* of ETCS while $\forall_X$ is a function *inside* of ETCS. By applying the pullback property of the diagram above with $(p\pi_0)^\sharp : \mathbf{1} \to \Omega^X$ and $\mathrm{id}_{\mathbf{1}} : \mathbf{1} \to \mathbf{1}$, since we have

$$
\begin{aligned}
&\forall_X \circ (p\pi_0)^\sharp = \mathsf{t} \\
&\Leftrightarrow (p\pi_0)^\sharp = (\mathsf{t}\pi_1)^\sharp \\
&\Leftrightarrow e_\Omega \circ (\mathrm{id} \times (p\pi_0)^\sharp) = e_\Omega \circ (\mathrm{id} \times (\mathsf{t}\pi_1)^\sharp) \\
&\Leftrightarrow p\pi_0 = \mathsf{t}\pi_1 \\
&\Leftrightarrow p\pi_0 = \mathsf{t}\beta_X\pi_0 \\
&\Leftrightarrow p = \mathsf{t}\beta_X \\
&\Leftrightarrow \forall\, x \in X.\ p(x) = \mathsf{t}\beta_X(x) \\
&\Leftrightarrow \forall\, x \in X.\ p(x) = \mathsf{t}
\end{aligned}
$$

Finally, since the existential operator is often defined as $\exists \equiv \neg\forall\neg$, we should try to define $\exists_X$ similarly. However, since $\mathrm{dom}(\forall_X) = \Omega^X$, we should lift $NOT : \Omega \to \Omega$ to $NOT^X : \Omega^{\mathbb{N}} \to \Omega^X$ so that the types work out. Thus, we define $\exists_X = \mathrm{NOT} \circ \forall_X \circ \mathrm{NOT}^{\mathbb{N}} : \Omega^{\mathbb{N}} \to \Omega$. With this definition in place, we can similarly show that

$$(\exists_X \circ (p\pi_0)^\sharp = \mathsf{t}) \text{ if and only if } (\exists\, x \in X.\ p(x) = \mathsf{t}).$$

Likewise, we define OR and AND as the unique maps, χ, which make the following respective diagrams pullbacks:

$$
\begin{array}{ccc}
\mathbf{1} \amalg (\mathbf{1} \amalg \mathbf{1}) & \xrightarrow{\beta} & \mathbf{1} \\
{\scriptstyle \langle \mathsf{t},\mathsf{t}\rangle \amalg (\langle \mathsf{t},\mathsf{f}\rangle \amalg \langle \mathsf{f},\mathsf{t}\rangle)}\downarrow & & \downarrow{\scriptstyle \mathsf{t}} \\
\Omega \times \Omega & \xrightarrow[\chi]{} & \Omega
\end{array}
$$

and

$$
\begin{array}{ccc}
\mathbf{1} & \xrightarrow{\beta_{\mathbf{1}}} & \mathbf{1} \\
{\scriptstyle \langle \mathsf{t},\mathsf{t}\rangle}\downarrow & & \downarrow{\scriptstyle \mathsf{t}} \\
\Omega \times \Omega & \xrightarrow[\chi]{} & \Omega
\end{array}
$$

In a similar fashion we can define $XOR, NAND, IFF$, and $IMPLIES$. We summarise their definitions within the Isabelle environment here:

```
definition NOT :: "cfunc" where
  "NOT = (THE χ. is_pullback 1 1  Ω Ω β_1 t f χ)"
definition AND :: "cfunc" where
  "AND = (THE χ. is_pullback 1 1 (Ω × Ω) Ω β_1 t ⟨t,t⟩ χ)"

definition NOR :: "cfunc" where
"NOR = (THE χ. is_pullback 1 1 (Ω × Ω) Ω β_1 t ⟨f,f⟩ χ)"

definition OR :: "cfunc" where
```

```
"OR = (THE χ. is_pullback (1⨿(1⨿1)) 1 (Ω × Ω) Ω β(1⨿(1⨿1)) t
(⟨t,t⟩⨿(⟨t,f⟩⨿⟨f,t⟩)) χ)"

definition XOR :: "cfunc" where
"XOR = (THE χ. is_pullback (1⨿1) 1 (Ω × Ω) Ω β(1⨿1) t
(⟨t,f⟩⨿⟨f,t⟩) χ)"

definition NAND :: "cfunc" where
"NAND = (THE χ. is_pullback (1⨿(1⨿1)) 1 (Ω × Ω) Ω β(1⨿(1⨿1))
    t
(⟨f,f⟩⨿(⟨t,f⟩⨿⟨f,t⟩)) χ)"

definition IFF :: "cfunc" where
"IFF = (THE χ. is_pullback (1⨿1) 1 (Ω × Ω) Ω β(1⨿1) t
(⟨t,t⟩⨿⟨f,f⟩) χ)"

definition IMPLIES :: "cfunc" where
"IMPLIES = (THE χ. is_pullback (1⨿(1⨿1)) 1 (Ω × Ω) Ω β(1⨿(1
    ⨿1)) t
(⟨t,t⟩⨿(⟨f,f⟩⨿⟨f,t⟩)) χ)"
```

Listing 1.26. "Internal" Boolean Operators

With this internal logic in place, we are finally in a position to say what exactly $n \leq m$ means. We formally define $\leq\colon \mathbb{N} \times \mathbb{N} \to \Omega$ as:

```
definition leq :: "cfunc" where
  "leq = EXISTS ℕ ∘ (eq_pred ℕ ∘ (add2 × id ℕ) ∘ associate_left
    ℕ ℕ ℕ)^♯"
```

Listing 1.27. Less-than-or-equal-to Operator

From here we are able to prove the main fact about $\leq$, namely its usual definition:

```
lemma leq_true_implies_exists:
  assumes m_type: "m ∈ ℕ" and n_type: "n ∈ ℕ"
  assumes leq_true: "leq ∘ ⟨m, n⟩ = t"
  shows "∃ k. k ∈ ℕ ∧ k +  m = n"

lemma exists_implies_leq_true:
  assumes m_type: "m ∈ ℕ" and n_type: "n ∈ ℕ"
  assumes leq_eqn: "∃ k. k ∈ ℕ ∧ k + m = n"
  shows "leq ∘ ⟨ m, n⟩ = t"
```

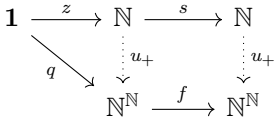
Listing 1.28. The usual definition as a corollary

One can further easily see, for example, that $m < n$ if and only if $n \not\leq m$, thus we can define the predicate $\texttt{lt} = \text{NOT} \circ \texttt{leq}$. Thus, we can see that our internal logical system allows us to bridge meta and object level truth values, and that

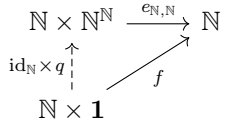
this is not merely an exercise in logic; rather, we are able to turn intuitively clear predicates into ETCS functions that capture their essence.

5 Constructing Functions within ETCS

We briefly mention that within ETCS there is a general strategy to construct functions by combining Axioms 9 and 10 and carefully choosing auxiliary functions corresponding to the solid arrows of those diagrams to get the desired dashed arrows. This is by no means an easy task but often involves making guesses and "type chasing"; to illustrate the method we show how one can define addition on the naturals. We will use Axiom 10 to construct the curried version of addition, say $u_+ : \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$, so that $u_+^\flat$ will be ordinary binary addition (i.e. $u_+^\flat\langle n, m\rangle = n + m$), which is to say we set $X = \mathbb{N}^{\mathbb{N}}$ in Axiom 10:



Intuitively, since u_+ represents curried addition, it must be that $u_+(m) \equiv \lambda\, n.n + m$ or that $u_+(m)$ is the "plus m" function. It follows that $u_+(0) = n + 0$ corresponds to the constant function $n \mapsto n$. Precisely, our first step to constructing u_+ is to find a $q : 1 \to \mathbb{N}^{\mathbb{N}}$ such that $q = u_+(z)$. Now comes the second half of our general strategy, we use Axiom 9 to find $q : 1 \to \mathbb{N}^{\mathbb{N}}$ since it (uniquely) outputs a function $f^\sharp : Y \dashrightarrow Z^X$. We determine how to apply Axiom 9 by aligning its output with the required input for q in Axiom 10. Since $q : 1 \to \mathbb{N}^{\mathbb{N}}$ is our input to Axiom 10, it must be that $f^\sharp$ of Axiom 9 is q from Axiom 10 and hence $Y = \mathbf{1}$ and $X = Z = \mathbb{N}$. That is,



But since $f = (f^\sharp)^\flat$ and $q = f^\sharp$, by considering how f acts on elements from its domain, it follows that:

$$\begin{aligned} f\langle n, \mathrm{id}_{\mathbf{1}}\rangle &= q^\flat\langle n, \mathrm{id}_{\mathbf{1}}\rangle \\ &= (u_+ z)^\flat\langle n, \mathrm{id}_{\mathbf{1}}\rangle \\ &= u_+^\flat \circ (\mathrm{id}_{\mathbb{N}} \times z)\langle n, \mathrm{id}_{\mathbf{1}}\rangle \\ &= +\langle n, 0\rangle \\ &= n \end{aligned}$$

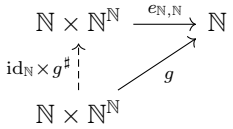
That is, we need f to be defined as the map $\langle n, \mathrm{id}_{\mathbf{1}}\rangle \mapsto n$ and this is precisely the left projection on $\mathbb{N} \times \mathbf{1}$. To conclude, in Axiom 10 we set $q := (\pi_0^{\mathbb{N}\times\mathbf{1}})^\sharp$.

Now we seek to determine $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ from Axiom 10 so that we can uniquely and correctly define u_+. First, we note from Axiom 10 that we require

$$f\,(\lambda\, n.n + m) = f \circ u_+(m) = u_+(sm) = (\lambda\, n.n + sm)$$

for every $m \in \mathbb{N}$ equivalently $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ represents a function $(\lambda\, n.\ n + m) \mapsto (\lambda\, n.\ n + (m + 1))$. That is, we want a function that given an "add m" function, returns an "add $m + 1$" function.

Thus, we will again use Axiom 9, to produce a function $g^\sharp = f$ where $f : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ so that in Axiom 9 $X = Z = \mathbb{N}$ and $Y = \mathbb{N}^{\mathbb{N}}$. We summarize this as:

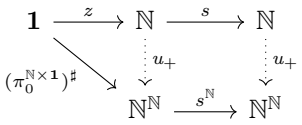


We can solve for $f = g^\sharp$ algebraically since we know that $u_+(sm) = fu_+m$, we simply need to apply $\flat$ to both sides in an attempt to invert f. This leads to the following chain of equalities:

$$\begin{aligned} u_+^\flat \circ (\mathrm{id}_{\mathbb{N}} \times sm) &= \big[u_+\big(sm\big)\big]^\flat \\ &= \big[f\ u_+\ m\big]^\flat \\ &= f^\flat \circ (\mathrm{id}_{\mathbb{N}} \times u_+\ m) \\ &= f^\flat \circ (\mathrm{id}_{\mathbb{N}} \times u_+)\big(\mathrm{id}_{\mathbb{N}} \times m\big) \end{aligned}$$

Applying both sides to $\langle n, \mathrm{id}_{\mathbb{N}}\rangle$ shows that the left-hand side evaluates to $n + (m + 1)$. By guessing $f := \big(s \circ e_{\mathbb{N},\mathbb{N}}\big)^\sharp = s^{\mathbb{N}}$, this right-hand side evaluates to $s\big(n + m\big)$, matching the left-hand side as desired.

To recap, after a fair bit of algebraic manipulation and some guess work, we arrive at the pair of functions $q := (\pi_0^{\mathbb{N}\times\mathbf{1}})^\sharp$ and $f := s^{\mathbb{N}}$ to define $u_+ : \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$ finally we uncurry this operation to define orderinary addition as $u_+^\flat : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$. In summary:



with $\mathsf{add} := u_+^\flat$, and conventionally we write $n + m$ for $\mathsf{add}\langle n, m\rangle$.

6 Recursion

6.1 General Recursive Functions

Lawvere in his original paper on ETCS, proved the Primitive Recursion theorem [5], leaving the General Recursion theorem open. The general recursive functions

are the smallest class of partial functions that includes the initial functions and is closed under composition, primitive recursion, and the minimization operator μ.

Theorem 1 (Minimisation). *Given a function $f : \mathbb{N} \times A \to \mathbb{N}$, there exists a function $\mu_f : A \to \mathbb{N} \coprod \mathbf{1}$ such that, for all $a \in A$:*

$$\mu_f(a) = i_0 n \iff \big(f\langle n, a\rangle = 0 \ \wedge \ (\forall\ m < n\,.\, f\langle m, a\rangle > 0)\big).$$

and $\mu_f(a) = i_1 \mathrm{id}_1$ if there is no such n that would give $f\langle n, a\rangle = 0$.

Proof. We seek to construct a characteristic function which lifts the external predicate

$$f\langle n, a\rangle = 0 \ \wedge \ \forall\, m < n.\, f\langle m, a\rangle > 0$$

to an internal predicate $\chi_f : \mathbb{N} \times A \to \Omega$ which represents it. We do this in parts. First the truth-value of the expression "$f\langle n, a\rangle = 0''$ is equivalent to the valuation of $\mathtt{eq\_pred}_{\mathbb{N}}\langle f\langle n, a\rangle, 0\rangle$. Let $Z_f : \mathbb{N} \times A \to \Omega$ denote the predicate "constant 0 on $\mathbb{N} \times A$" then

$$Z_f := \mathtt{eq\_pred}_{\mathbb{N}}\langle f, 0 \circ \beta_{\mathbb{N}\times A}\rangle\,.$$

Similarly, the expression "$f\langle m, a\rangle > 0$" is simply "$f\langle m, a\rangle \neq 0$" and hence its predicate always has the opposite valuation of the last predicate; therefore, we represent this expression valuating $P_f := \mathrm{NOT} \circ Z_f$ against $\langle m, a\rangle$.

Next we work on the second half of the main predicate, observe the following equivalence:

$$(\forall\, m < n. f\langle m, a\rangle > 0) \equiv (\forall\, m \in \mathbb{N}.(m < n \Rightarrow f\langle m, a\rangle > 0))$$

Thus, we want an internal predicate

$$P(m, n, a) \ :\equiv \ \big(m < n \ \Rightarrow \ f(m, a) > 0\big)$$

as a map

$$P : \mathbb{N} \times (\mathbb{N} \times A) \ \to \ \Omega.$$

After a moment's reflection it is clear that

$$P \ := \ \mathrm{IMPLIES} \circ \Big\langle \mathrm{lt} \circ \langle \pi_0,\, \pi_0 \circ \pi_1\rangle,\ P_f \circ \langle \pi_0,\, \pi_1 \circ \pi_1\rangle \Big\rangle \ : \ \mathbb{N} \times (\mathbb{N} \times A) \longrightarrow \Omega.$$

Now we want, for each $\langle n, a\rangle$ the truth value of $\forall\, m.P(m, n, a)$ and by Axiom 9 its transpose $P^\sharp : (\mathbb{N} \times A) \to \Omega^{\mathbb{N}}$ represents, for each $\langle n, a\rangle$ the predicate $m \mapsto P(m, n, a)$. We "quantify" this predicate by composing it with $\forall_{\mathbb{N}}$. Therefore let $\chi_f = \mathrm{AND} \circ \langle Z_f, \forall_{\mathbb{N}} \circ P^\sharp\rangle : \mathbb{N} \times A \to \Omega$, it follows by construction:

$$\chi_f\langle n, a\rangle = \mathtt{t} \text{ if and only if } "f\langle n, a\rangle = 0 \ \wedge \ \forall\, m < n.\, f\langle m, a\rangle > 0''\,.$$

Now fix $a \in A$, and define $\chi_f^a := \chi_f \circ \langle \mathrm{id}_{\mathbb{N}},\ a \circ \beta_{\mathbb{N}} \rangle : \mathbb{N} \to \Omega$, then $\chi_f^a(n) = \mathtt{t}$ if and only if the main predicate holds for (n, a).

Let

$$M_a = \{n \in \mathbb{N} \mid f\langle n, a\rangle = 0 \wedge \forall\, m < n.f(m, a) > 0\}$$

which is officially defined as the equalizer of χ_f^a and $\mathtt{t}$. Observe that M_a can have at most one element. For suppose by contradiction that $n, n' \in M_a$ with $n < n'$ then $f\langle n', a\rangle = 0$ and $\forall\, m < n'.f(m, a) > 0$. But $n < n'$ so $f(n, a) > 0$ however $n \in M_a$ so $f(n, a) = 0$, a contradiction!

We want to define the minimisation operator

$$\mu_f : A \longrightarrow \mathbb{N} \coprod \mathbf{1}$$

by

$$\mu_f(a) \ = \ \begin{cases} i_0(n), & \text{if } M_a = \{n\}, \\ i_1(\mathrm{id}_{\mathbf{1}}), & \text{if } M_a = \emptyset\,, \end{cases}$$

but we must do so internal to our theory. Recall that $\exists_{\mathbb{N}} : \Omega^{\mathbb{N}} \to \Omega$ has the property that $\exists_{\mathbb{N}}(p) = \mathrm{t}$ if and only if $\exists\, n \in \mathbb{N}.p(n) = \mathtt{t}$. Further, observe that $\chi_f^\sharp \circ a : 1 \to \Omega^{\mathbb{N}}$ is the curried version of the predicate $n \mapsto \chi_f(n, a)$, so we may rightly define a predicate $\exists_f : A \to \Omega$ by

$$\exists_f \ := \ \exists_{\mathbb{N}} \circ \chi_f^\sharp \ : \ A \longrightarrow \Omega.$$

and it has the following semantic property:

$$\exists_f(a) = \mathtt{t} \iff \exists\, n \in \mathbb{N}.\ \chi_f(n, a) = \mathtt{t} \iff M_a \neq \emptyset.$$

We now internalise the family of sets M_a and use $\exists_f$ to obtain a canonical "domain of definition" for the minimisation operator.

Let $m : M \hookrightarrow \mathbb{N} \times A$ be the subobject classified by χ_f, i.e. $M \subseteq \mathbb{N} \times A$ is the relation

$$M \ = \ \{\, (n, a) \in \mathbb{N} \times A \mid \chi_f\langle n, a\rangle = \mathtt{t} \,\}.$$

We write $\pi_0^M : M \to \mathbb{N}$ and $\pi_1^M : M \to A$ for the composites $\pi_0 \circ m$ and $\pi_1 \circ m$, respectively. Then $\pi_1^M : M \to A$ is the projection $(n, a) \mapsto a$ restricted to M.

By the argument above, for each fixed $a \in A$ the fibre

$$M_a = \{\, n \in \mathbb{N} \mid (n, a) \in M \,\}$$

has *at most* one element, and M_a is nonempty if and only if $\exists_f(a) = \mathtt{t}$. In particular, π_1^M is functional[3]: for each a there is at most one n with $(n, a) \in M$.

Next, let $D \hookrightarrow A$ be the subobject classified by $\exists_f$, i.e. the mono $d : D \hookrightarrow A$ such that

$$\chi_D \ = \ \exists_f : A \to \Omega,$$

[3] Recall that a *relation* $R \subset X \times Y$ is said to be **functional** just in case for each $x \in X$ there is a unique $y \in Y$ such that $\langle x, y\rangle \in R$. But as $\pi_1^M : M \to A$ this isn't quite *relational* in the same sense but they are related.

where χ_D is the characteristic map of D. Concretely, D is the object of all those $a \in A$ such that $M_a \neq \emptyset$; that is, D is the "domain of definition" of the minimisation operator.

By the defining property of the existential quantifier, the image of $\pi_1^M : M \to A$ is exactly the subobject $d : D \hookrightarrow A$, and the epi–mono factorisation of π_1^M has the form

$$M \xrightarrow{e} D \xrightarrow{d} A$$

with e epi and d mono. On each fibre $D_a = \{a\}$, the fibre M_a is nonempty and has at most one element; therefore e is also mono, hence an isomorphism. Thus there is an isomorphism

$$e : M \xrightarrow{\cong} D$$

with inverse $e^{-1} : D \to M$.

We can now define the *witness* map

$$\nu_f \;:=\; \pi_0^M \circ e^{-1} : D \longrightarrow \mathbb{N}.$$

For $a \in D$ this sends a to the unique $n \in \mathbb{N}$ such that $\chi_f(n, a) = \mathtt{t}$, i.e. the unique element of M_a.

Recall every subobject has a complement[4]. Let $d^{\complement} : D^{\complement} \hookrightarrow A$ be the mono classified by $\mathrm{NOT} \circ \exists_f : A \to \Omega$; thus $D^{\complement}$ is the subobject of those a for which $M_a = \emptyset$. By the usual subobject decomposition, we obtain a coproduct diagram

$$D \xrightarrow{i_0} D \coprod D^{\complement} \xleftarrow{i_1} D^{\complement}$$

together with an isomorphism

$$\sigma_f : D \coprod D^{\complement} \xrightarrow{\cong} A$$

such that $\sigma_f \circ i_0 = d$ and $\sigma_f \circ i_1 = d^{\complement}$. We write $\sigma_f^{-1} : A \to D \coprod D^{\complement}$ for its inverse.

We now define a map

$$\widehat{\mu}_f : D \coprod D^{\complement} \longrightarrow \mathbb{N} \coprod \mathbf{1}$$

by case distinction on the coproduct:

$$\widehat{\mu}_f \;:=\; (\, i_0 \circ \nu_f \,) \;\amalg\; (\, i_1 \circ \beta_{D^{\complement}} \,)$$

Finally, we transport $\widehat{\mu}_f$ along σ_f to obtain the desired minimisation operator

$$\mu_f \;:=\; \widehat{\mu}_f \circ \sigma_f^{-1} \;:\; A \longrightarrow \mathbb{N} \coprod \mathbf{1}.$$

Unwinding the definitions, for any $a \in A$ we have two cases:

[4] By "complement" we mean that given a monomorphism $d : D \to A$ we form the complement pair $(d^{\complement}, D^{\complement})$ which is defined in the Isabelle as `complement_morphism`(d) : $A \backslash (D, d) \to A$ and has the essential property that $\chi_d \circ d^{\complement} = \mathtt{f} \circ \beta_A \circ d^{\complement}$.

1. If $a \in_A D$, then $\exists_f(a) = \mathtt{t}$ and there is a unique n with $\chi_f(n, a) = \mathtt{t}$, i.e. $M_a = \{n\}$. In this case $\sigma_f^{-1}(a)$ lies in the left summand D, so

$$\mu_f(a) \;=\; \widehat{\mu}_f(\sigma_f^{-1}(a)) \;=\; i_0(\nu_f(a)) \;=\; i_0(n),$$

   and by construction n satisfies

$$f\langle n, a\rangle = 0 \quad \text{and} \quad \forall\, m < n.\, f\langle m, a\rangle > 0.$$

2. If $a \in_A D^{\mathbb{C}}$, then $\exists_f(a) = \mathtt{f}$ and $M_a = \emptyset$. In this case $\sigma_f^{-1}(a)$ lies in the right summand $D^{\mathbb{C}}$, so

$$\mu_f(a) \;=\; \widehat{\mu}_f(\sigma_f^{-1}(a)) \;=\; i_1(\beta_{D^{\mathbb{C}}}(a)) \;=\; i_1(\mathrm{id}_{\mathbf{1}}),$$

   expressing that there is no n with $f\langle n, a\rangle = 0$.

This is exactly the behaviour required in the statement of the theorem, so μ_f is the desired minimisation operator.

7 Conclusion

In this paper we have given a precise, machine-checked axiomatization of the Elementary Theory of the Category of Sets (ETCS) inside Isabelle/HOL. Working with the abstract types `cset` and `cfunc`, we formalised the standard ETCS axioms for the category **Sets**: categorical structure, finite products and coproducts, terminal and initial objects, equalizers, quotients by equivalence relations, a truth object Ω, exponential objects, a natural number object $\mathbb{N}$, and the axiom of choice. On top of these axioms we developed a substantial library of derived constructions, including products and bowtie products of morphisms, associativity rebracketing maps, complements, and properties of mono-, epi-, and isomorphisms, so that arguments carried out in the prover closely resemble familiar categorical proofs on paper.

A central theme of our development is the internalisation of logical reasoning. Using the truth object Ω and the subobject classifier, we defined characteristic maps and then built an internal Boolean algebra of connectives (NOT, AND, OR, XOR, NAND, IFF, IMPLIES) entirely out of categorical structure. We further constructed internal quantifiers $\forall_X, \exists_X : \Omega^X \to \Omega$ and proved that, for suitable predicates $p : X \to \Omega$, internal truth of $\forall_X$ and $\exists_X$ coincides with the intended external, meta-level semantics. This bridge between object-level and meta-level truth underpins our development of arithmetic within ETCS: we defined order on $\mathbb{N}$ via $\leq : \mathbb{N} \times \mathbb{N} \to \Omega$, proved its usual characterisation in terms of the existence of a witness $k \in \mathbb{N}$ with $n + k = m$, and showed how familiar inequalities can be recovered as ETCS morphisms.

From the point of view of computability, we pushed Lawvere's original treatment of primitive recursion further by giving a fully internal construction of minimisation. For any $f : \mathbb{N} \times A \to \mathbb{N}$ we defined a morphism, $\mu_f : A \longrightarrow \mathbb{N} \coprod \mathbf{1}$,

that returns the least n with $f\langle n, a\rangle = 0$, when such an n exists, and a distinguished "undefined" value otherwise. The construction is carried out entirely within ETCS using the truth object, characteristic maps, quantifiers, quotient and image factorisation, and coproduct decompositions. In particular, the Isabelle/HOL proofs show that the general minimisation scheme can be formulated and verified in ETCS without appealing to an external, set-theoretic encoding of partial functions. This provides a concrete categorical realisation of general recursion on top of the primitive recursion principles already present in ETCS.

A further contribution of this work is infrastructural. The `typecheck_cfuncs` ML tactic (and related proof procedures) systematically discharge domain–codomain side conditions and associativity reshufflings, generating typing lemmas for all subexpressions involving `cfunc`. This significantly reduces the bookkeeping overhead typical of categorical developments and makes it realistic to scale ETCS proofs in Isabelle to nontrivial metatheoretic arguments. In effect, the mechanisation turns ETCS into a usable working environment for doing categorical foundations, rather than a bare collection of axioms.

There are several directions for future work. On the foundational side, our Isabelle development provides a concrete setting in which to investigate the relationship between ETCS and traditional set theories, for example by formalising interpretations of fragments of ZFC in ETCS (and conversely) and studying bi-interpretability questions. On the mathematical side, the present library offers a basis for building larger swathes of ordinary mathematics internally to ETCS—algebra, analysis, and category theory—while maintaining a clear separation between object-level reasoning and meta-level logic. Finally, because the formalisation is integrated into the Isabelle ecosystem, it can serve as a categorical foundation for other verification projects (for instance, those based on UTP-style semantics or iterated monadic constructions), opening the door to further connections between category-theoretic foundations, formal verification, and mechanised metatheory.

References

1. Cesare Burali-Forti. Una questione sui numeri transfiniti. Rendiconti del Circolo Matematico di Palermo (1884-1940), 11:154–164, 1897.
2. Alonzo Church. An unsolvable problem of elementary number theory. American Journal of Mathematics, 58:345–363, 1936.
3. Hans Halvorson. The Logic in Philosophy of Science. Cambridge University Press, Cambridge, 2019.
4. William S. Hatcher. The Logical Foundations of Mathematics. Pergamon, 1982.
5. F. William Lawvere. An elementary theory of the category of sets. Proceedings of the National Academy of Sciences of the United States of America, 52:1506–1511, 1964.
6. Tom Leinster. Rethinking set theory. The American Mathematical Monthly, 121:403–415, 2014.
7. Øystein Linnebo and Richard Pettigrew. Category theory as an autonomous foundation. Philosophia Mathematica, 19:227–254, 2011.

8. Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. Isabelle/HOL: A Proof Assistant for Higher-Order Logic, volume 2283 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2002.
9. Gerhard Osius. Categorical set theory: A characterization of the category of sets. Journal of Pure and Applied Algebra, 4:79–119, 1974.
10. Giuseppe Peano. Arithmetices principia: Nova methodo exposita. Fratres Bocca, 1889.
11. John von Neumann. Zur einführung der transfiniten zahlen. Acta Litterarum ac Scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae, 1:199–208, 1923.