An Implementation of Set Theory with Pointed Graphs in Dedukti

Valentin Blot, Gilles Dowek, Thomas Traversié

An Implementation of Set Theory with Pointed Graphs in Dedukti

Valentin Blot
Inria, France
LMF, ENS Paris-Saclay, France
valentin.blot@inria.fr

Gilles Dowek
Inria, France
LMF, ENS Paris-Saclay, France
gilles.dowek@ens-paris-saclay.fr

Thomas Traversié
CentraleSupélec, France
thomas.traversie@inria.fr

DEDUKTI is a type-checker for the $\lambda\Pi$-*calculus modulo theory*, a logical framework that allows the extension of conversion with user-defined rewrite rules. In this paper, we present the implementation of a version of Dowek-Miquel's intuitionistic set theory in DEDUKTI. To do so, we adapt this theory — based on the concept of pointed graphs — from *Deduction modulo theory* to $\lambda\Pi$-*calculus modulo theory*, and we *formally* write the proofs in DEDUKTI. In particular, this implementation requires the definition of a deep embedding of a certain class of formulas, as well as its interpretation in the theory.

1 Introduction

During the last decades, theorem provers have attracted a lot of interest from various fields of science. Especially, the use of such tools has become more and more common in the areas of software verification and formalization of mathematics. As an example, verification has become mandatory for software running on aircrafts, and formalization of mathematical results allowed the scientific community to trust some complicated proofs of recent results, as well as identify and fix errors in some cases.

This growing interest has triggered the development of many theorem provers, with various focuses and based on a large range of theories. While the diversity of theorem provers provides users with a wide range of tools, this comes at the expense of a lack of interaction and reusability between proof fragments done in different tools. The $\lambda\Pi$-*calculus modulo theory* is a logical framework supporting dependent types and user-definable rewrite rules, and DEDUKTI [2] is a type-checker for this framework. The definition of well-chosen rewrite rules allows the encoding of various logics, and hence the translations between various theorem provers through the DEDUKTI tool.

Several theorem provers, such as MIZAR, ATELIER B and ISABELLE/ZF, are based on set theory. In order to extend the interoperability allowed by DEDUKTI to these provers, it is necessary to encode set theory in the $\lambda\Pi$-*calculus modulo theory*, and to implement this encoding in DEDUKTI. The goal of this paper is to present such an encoding and implementation. In order to facilitate the proof of its correctness, we implement this encoding in the tool LAMBDAPI that provides tactics to help the user in the production of proofs type-checkable in DEDUKTI.

Stating each axiom of set theory in DEDUKTI would lead to an implementation that does not satisfy a cut-elimination theorem. In particular, it would forbid extraction of witnesses from constructive existence proofs.

An alternative option would be to orient these axioms as rewriting rules. For instance the powerset axiom $x \in \mathscr{P}(y) \Leftrightarrow x \subseteq y$ would be replaced with the rewrite rule $x \in \mathscr{P}(y) \longrightarrow x \subseteq y$. However, as pointed out by Crabbé [4], such a formulation of set theory does not satisfy a cut-elimination property either. Indeed, if for a given set a we define the set $b \equiv \{x \in a \,|\, x \in x\}$ then $b \in b$ would get rewritten to $b \in a \wedge b \in b$, leading to an infinite reduction.

Logical Frameworks and Meta Languages:
Theory and Practice (LFMTP 2022)

In the current paper, we represent sets as *pointed graphs* [1]. With such a formulation, we can prove a proof normalization theorem: for every proof in natural deduction there exists a normal proof of the same statement. Such an encoding of set theory has been defined in the context of *Deduction modulo theory* [5], together with pencil and paper proofs. At that time, the $\lambda\Pi$-*calculus modulo theory* had not been defined yet, and the first contribution of the current paper is to adapt this encoding to the $\lambda\Pi$-*calculus modulo theory*. In particular, we avoid the original definition of classes of nodes by using the quantification on propositions permitted by $\lambda\Pi$-*calculus modulo theory*. This reduces the size of the signature from 31 to 26 symbols.

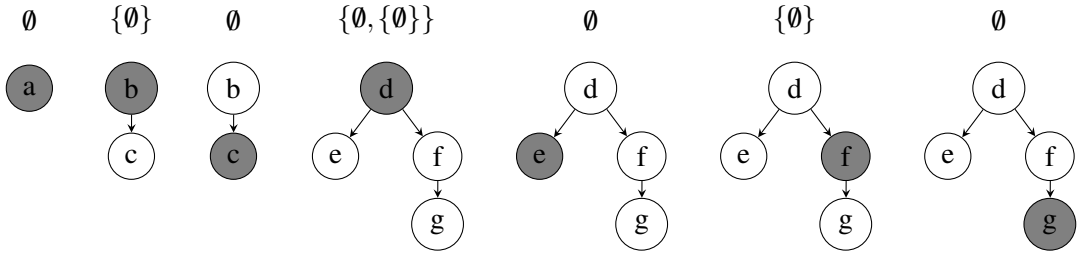
The second contribution consists of the implementation of this new theory in DEDUKTI. In the original formulation of the theory some lemmas are only valid for a specific class of formulas, as their proofs proceed by induction on the structure of these formulas. Implementing these lemmas in DEDUKTI requires the definition of an inductive sort of formulas together with an interpretation of these formulas into the general type of propositions. This interpretation is defined with rewriting rules. The current development is the first, to our knowledge, to use such a reflection principle defined with rewriting rules in DEDUKTI. The generality of this method still remains to be investigated.

2 The Theory of Pointed Graphs

The set theory with pointed graphs developed by Dowek and Miquel is called Intuitionistic Zermelo set theory in *Deduction modulo* (IZmod). We give here an informal presentation of the ideas developed in [5].

2.1 Sets as Pointed Graphs

In the IZmod theory, sets are represented by pointed graphs, that is, directed graphs with a distinguished node: the root. We give here the representation of several sets as pointed graphs. The root is indicated by the filled circle. The location of the root is important as it changes the set represented by the pointed graph. For example, in the third graph, the node b is irrelevant since the root is c. Distinct pointed graphs can represent the same set. For example, here are various representations of ordinals $0 = \emptyset$, $1 = \{\emptyset\}$ and $2 = \{\emptyset, \{\emptyset\}\}$ as pointed graphs.



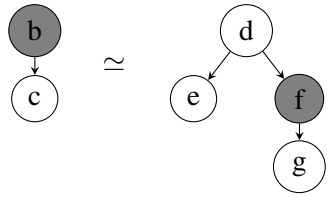
IZmod has a sort of pointed graphs and a sort of nodes. If a is a pointed graph, $root(a)$ is the root of a, which is a node. If x is a node of a, then a/x refers to the pointed graph a where x is the new root (this is a re-rooting operation). If y is also a node of a, $x\ \eta_a\ y$ is the proposition asserting that there is a edge in a from y to x. Following this interpretation, IZmod contains the following rewriting rules:

$$x\ \eta_{a/z}\ y \longrightarrow x\ \eta_a\ y \qquad root(a/x) \longrightarrow x \qquad (a/x)/y \longrightarrow a/y$$

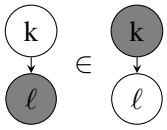
As noted above, distinct pointed graphs can represent the same set. Two pointed graphs representing the same set are related through a notion of bisimilarity, denoted $\simeq$, and defined with the rewrite rule:

$$\begin{aligned} a \simeq b \longrightarrow\ & \exists r,\ r\ root(a)\ root(b) \\ & \wedge\ \forall x \forall x'\ \forall y\ (x'\ \eta_a\ x\ \wedge\ r\ x\ y \Rightarrow \exists y'\ (y'\ \eta_b\ y\ \wedge\ r\ x'\ y')) \\ & \wedge\ \forall y \forall y'\ \forall x\ (y'\ \eta_b\ y\ \wedge\ r\ x\ y \Rightarrow \exists x'\ (x'\ \eta_a\ x\ \wedge\ r\ x'\ y')) \end{aligned}$$

As an example, the following pointed graphs are bisimilar, hence they represent the same set:



In set theory, there only exists one sort: the sets. The IZmod theory has two main sorts — nodes and pointed graphs — and two notions related to set membership — relations between pointed graphs and relations between nodes within the same graph. As an example, take $\emptyset \in \{\emptyset\}$:



Since there is an edge from k to ℓ, the set represented by the pointed graph with root ℓ is an element of the set represented by the pointed graph with root k. But any pointed graph bisimilar to the pointed graph with root ℓ also represents a set that is an element of the set represented by the pointed graph with root k. This leads to the definition of a membership relation $\in$ with the rewrite rule:

$$a \in b \longrightarrow \exists x\ (x\ \eta_b\ root(b)\ \wedge\ a \simeq (b/x))$$

In order to define set constructions such as pairs or powersets, we need a way of *joining* two existing graphs together, as well as a way of *adding* a new node to an existing graph. To this end, the IZmod theory has two *disjoint* injections i and j on nodes, and a symbol for some constant node o that is not in the image of i or j. There are also inverses i' and j' to i and j, and predicates I and J on the images of i and j:

$$\begin{array}{llll} i'(i(x)) \longrightarrow x & I(i(x)) \longrightarrow \top & I(j(x)) \longrightarrow \bot & I(o) \longrightarrow \bot \\ j'(j(x)) \longrightarrow x & J(j(x)) \longrightarrow \top & J(i(x)) \longrightarrow \bot & J(o) \longrightarrow \bot \end{array}$$

For example, we want to define a pointed graph $\{a,b\}$ that satisfies the axiom of pairing, that is to have $\forall x\ (x \in \{a,b\} \Leftrightarrow (x \simeq a \vee x \simeq b))$. The intuitive idea to build this pointed graph is to create a new node that will be parent of the root of a and of the root of b, and to define this new node to be the root of $\{a,b\}$. We need to ensure that all the nodes of $\{a,b\}$ are different from each other. To comply with this constraint, we choose o for the new node and we use injection i on the nodes of a and injection j on the nodes of b. This construction is formalized by the rewriting rules

$$\begin{aligned} \mathrm{root}(\{a,b\}) \longrightarrow o \qquad\qquad & x\ \eta_{\{a,b\}}\ x' \longrightarrow\ (\exists y \exists y'\ (x = i(y) \wedge x' = i(y') \wedge y\ \eta_a\ y')) \\ & \vee\ (\exists y \exists y'\ (x = j(y) \wedge x' = j(y') \wedge y\ \eta_b\ y')) \\ & \vee\ (x = i(\mathrm{root}(a)) \wedge x' = o) \\ & \vee\ (x = j(\mathrm{root}(b)) \wedge x' = o) \end{aligned}$$

We can proceed similarly for other constructions such that powerset or union.

2.2 Set Theories

The membership $\in$ and bisimilarity $\simeq$ relations define a set theory IZst that lies between Zermelo (Z) and Zermelo-Fraenkel (ZF). This theory does not include the Replacement scheme but it contains two additional axioms: Strong Extensionality and Transitive Closure:

Strong Extensionality axiom.

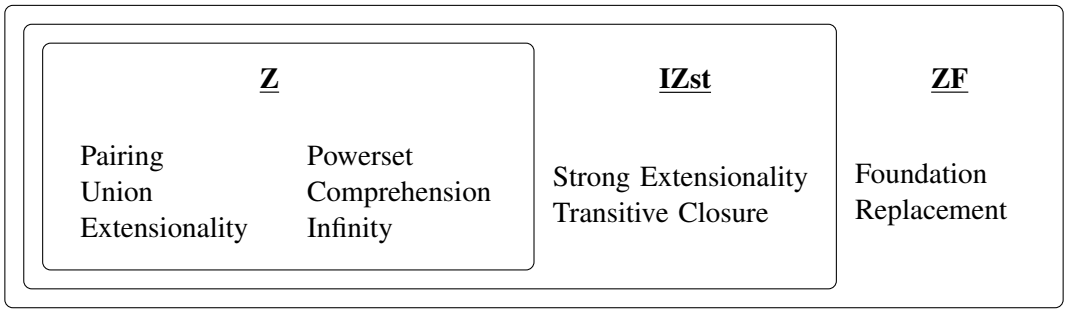
$$\begin{aligned}\forall x_1 ... \forall x_n \forall a \forall b\ (R(a,b)\\ &\wedge\ \forall x \forall x' \forall y\ (x' \in x \wedge R(x,y) \Rightarrow \exists y'\ (y' \in y \wedge R(x',y')))\\ &\wedge\ \forall y \forall y' \forall x\ (y' \in y \wedge R(x,y) \Rightarrow \exists x'\ (x' \in x \wedge R(x',y')))\\ &\Rightarrow a \simeq b)\end{aligned}$$

where $R(a,b)$ is a formula with free variables $x_1, ..., x_n$. Note that the hypothesis of the Strong Extensionality axiom mimics the structure of the rewrite rule for $\simeq$.

Transitive Closure axiom. $\forall a \exists e\ (a \subseteq e \wedge \forall x \forall y\ (x \in y \wedge y \in e \Rightarrow x \in e))$

The Transitive Closure axiom conveys the idea that every set is included in a transitive set.

Strong Extensionality can be deduced from Foundation and Transitive Closure can be derived from Replacement. Moreover, Strong Extensionality implies Extensionality[1]. Therefore we have the following picture:



The IZmod theory is an extension of IZst set theory with an encoding of pointed graphs. Indeed, the axioms of IZst are provable in IZmod [5, see Section 3.3].

3 The Language of Pointed Graphs

3.1 Sorts

There are four sorts in IZmod [5, see Section 3.2]: graphs, nodes, classes of nodes and binary relations between nodes. We implemented the first two via the standard technique [3] of defining a universe *Set* : `TYPE` of sorts, a function *El* of type *Set* $\rightarrow$ `TYPE` and two constants *graph* and *node* of type *Set*:

```
constant symbol graph : Set;
constant symbol node : Set;
```

[1] See Section 5.3

We also introduce a constant *arrow* of type $Set \to Set \to Set$, together with its associated rewrite rule $El\ (x\ arrow\ y) \longrightarrow (El\ x) \to (El\ y)$.

Contrary to *Deduction modulo*, classes of nodes and binary relations on nodes can be expressed primitively via quantifications on propositions in $\lambda\Pi$-*calculus modulo theory*. The sort of classes is defined in DEDUKTI as $El\ node \to El\ prop$ and that of binary relations as $El\ node \to El\ node \to El\ prop$.

The symbols *graph* and *node* are specific to the expression of IZmod in DEDUKTI, while in contrast the symbols *Set*, *El* and *prop* are part of the standard library of DEDUKTI.

3.2 Signature

The signature of IZmod contains 33 symbols (the 31 from [5, see Table 2], plus the symbol of empty set [5, see Section 3.2] and the symbol of inductive set [5, see Section 2.1]). As we have replaced the sorts for classes and relations with primitive DEDUKTI types, we do not need specific predicate symbols *mem* and *rel* to apply a class to a node or a relations to two. By virtue of the encoding of classes of nodes as elements of $El\ node \to El\ prop$, $mem(x,P)$ can simply be expressed as $P\ x$, and similarly $rel(x,y,r)$ can simply be expressed as $r\ x\ y$. In the same way, we do not need symbols $g_{x,y_1,\ldots,y_n,P}$ and $g'_{x,x',y_1,\ldots,y_n,P}$ to build classes and relations. Finally, the equality symbol is part of the standard library of DEDUKTI. The signature is thus reduced to 28 symbols. Below are the first 25 symbols, the specific case of the comprehension symbol is treated later. Some symbols are defined as `constant` since they are not subject to rewriting rules.

```
symbol eta : El graph → El node → El node → El prop;
symbol root : El graph → El node;
symbol cr : El graph → El node → El graph; // change of root

constant symbol o : El node;
symbol i : El node → El node;
symbol i' : El node → El node;
symbol j : El node → El node;
symbol j' : El node → El node;
symbol I : El node → El prop;
symbol J : El node → El prop;

// injection from graphs to nodes (for powerset) and its inverse
symbol ρ : El graph → El node;
symbol ρ' : El node → El graph;

// natural numbers
constant symbol 0 : El node;
symbol Succ : El node → El node;
symbol Pred : El node → El node;
symbol Null : El node → El prop;
symbol Nat : El node → El prop;
symbol < : El node → El node → El prop;

symbol simeq : El graph → El graph → El prop; // bisimilarity
symbol ∈ : El graph → El graph → El prop;
```

```
symbol join : El graph → El graph;
symbol pair : El graph → El graph → El graph;
symbol powerset : El graph → El graph;
symbol omega : El graph; // set of natural numbers
symbol closure : El graph → El graph;
```

3.3 Rewriting Rules

Among the rewrite rules [5, see Table 3], we can drop the compatibility of equality with *mem* as it comes for free with the Leibniz equality of DEDUKTI's standard library. We can also drop the rewrite rules determining the behavior of $g_{x,y_1,\ldots,y_n,P}$ and $g'_{x,x',y_1,\ldots,y_n,P}$, as they are superseeded by our encoding of classes and binary relations. The remaining rewrite rules are easy to implement in DEDUKTI:

General

```
rule eta (cr $a $z) $x $y ↪ eta $a $x $y;
rule root (cr $a $x) ↪ $x;
rule (cr (cr $a $x) $y) ↪ cr $a $y;
```

Relocations

```
rule i' (i $x) ↪ $x;
rule j' (j $x) ↪ $x;
rule ρ' (ρ $x) ↪ $x;
rule I (i $x) ↪ ⊤;
rule J (j $x) ↪ ⊤;
rule I (j $x) ↪ ⊥;
rule J (i $x) ↪ ⊥;
rule I (o) ↪ ⊥;
rule J (o) ↪ ⊥;

rule Pred (Succ $x) ↪ $x ;
rule Null 0 ↪ ⊤;
rule Nat 0 ↪ ⊤;
rule Null (Succ $x) ↪ ⊥;
rule Nat (Succ $x) ↪ Nat $x;

rule $x < 0 ↪ ⊥;
rule $x < (Succ $y) ↪ ($x < $y) ∨ ($x = $y);
```

Equality and Membership

```
rule $a simeq $b ↪ '∃ r : El (node arrow (node arrow prop)),
    r (root $a) (root $b)
    ∧ ('∀ x, '∀ x', '∀ y,
        eta $a x' x ∧ r x y
            ⇒ '∃ y', eta $b y' y ∧ r x' y')
    ∧ ('∀ y, '∀ y', '∀ x,
        eta $b y' y ∧ r x y
```

```
            ⇒ ‘∃ x’, eta $a x’ x ∧ r x’ y’);

rule $a ∈ $b ↪ ‘∃ x, ((eta $b x (root $b)) ∧ ($a simeq cr $b x));
```

Constructions

```
rule eta (join $a) $x $x’ ↪
  (‘∃ y, ‘∃ y’, ($x = i y) ∧ ($x’ = i y’) ∧ eta $a y y’)
    ∨ (‘∃ y, ‘∃ z, ($x = i y)
      ∧ ($x’ = o)
      ∧ eta $a y z
      ∧ eta $a z (root $a));

rule eta (pair $a $b) $x $x’ ↪
  (‘∃ y, ‘∃ y’, (($x = i y) ∧ ($x’ = i y’) ∧ eta $a y y’))
    ∨ (‘∃ y, ‘∃ y’, ($x = j y) ∧ ($x’ = j y’) ∧ eta $b y y’)
    ∨ (($x = i (root $a)) ∧ ($x’ = o))
    ∨ (($x = j (root $b)) ∧ ($x’ = o));

rule eta (powerset $a) $x $x’ ↪
  (‘∃ y, ‘∃ y’, ($x = i y) ∧ ($x’ = i y’) ∧ eta $a y y’)
    ∨ (‘∃ y, ‘∃ c, ($x = i y)
      ∧ ($x’ = j (ρ c))
      ∧ (eta $a y (root $a))
      ∧ ((cr $a y) ∈ c))
    ∨ (‘∃ c, ($x = j (ρ c)) ∧ ($x’ = o));

rule eta omega $x $x’ ↪
  (‘∃ y, ‘∃ y’, ($x = i y) ∧ ($x’ = i y’) ∧ (y < y’))
    ∨ (‘∃ y, ($x = i y) ∧ ($x’ = o) ∧ Nat y);

rule eta (closure $a) $x $x’ ↪
  (‘∃ y, ‘∃ y’, (($x = i y) ∧ ($x’ = i y’) ∧ eta $a y y’))
  ∨ (‘∃ y, ($x = i y)
       ∧ ($x’ = o)
       ∧ (‘∀ c : El (node arrow prop),
              ((‘∀ z, eta $a z (root $a) ⇒ c z)
              ∧ (‘∀ z, ‘∀ z’, (eta $a z z’) ∧ (c z’) ⇒ (c z)))
          ⇒ c y));

rule root (join $a) ↪ o;
rule root (pair $a $b) ↪ o;
rule root (powerset $a) ↪ o;
rule root omega ↪ o;
rule root (closure $a) ↪ o;
```

The pointed graph `omega` represents the set of von Neumann ordinals [5, see Table 2]. Indeed, the rewriting rule of `eta omega x x’` states that in `omega` the edges correspond to the order relation $<$ and

that there is an edge from every natural number to the root of `omega`. Therefore, the elements of `omega` are the natural numbers, and the elements of some natural number are all the natural numbers strictly below it, which corresponds to the definition of von Neumann ordinals.

4 The Language of Formulas

This section is devoted to the operator that builds sets by comprehension. Indeed, in IZmod, the only valid sets of the form $\{x \in a \,|\, P(x)\}$ are those for which P does not contain elements specific to the implementation of sets as pointed graphs, that is, all quantifiers in P must be on *graph* and P can only use symbols $\in$, $\simeq$ and the logical connectives [5, see Table 5]. Taking this restriction into account is the main challenge of our implementation. In order to achieve this goal, we define a datatype representing the class of formulas just described, as well as an interpretation of this datatype into *El prop*.

4.1 Formulas

We define the constant *formula* of type *Set* and the operators for building elements of *El formula*. Variables are represented as natural numbers and we do not need to handle α-conversion as we do not need substitution.

```
constant symbol formula : Set;
constant symbol eqF : El nat → El nat → El formula;
constant symbol inF : El nat → El nat → El formula;
constant symbol andF : El formula → El formula → El formula;
constant symbol orF : El formula → El formula → El formula;
constant symbol allF : El nat → El formula → El formula;
constant symbol exF : El nat → El formula → El formula;
constant symbol impF : El formula → El formula → El formula;
constant symbol fF : El formula;
constant symbol tF : El formula;
```

We also define an induction principle over first-order formulas:

```
constant symbol recF : Π (P : El formula → El prop),
π(‘∀ x, ‘∀ y, P (eqF x y))
→ π(‘∀ x, ‘∀  y, P (inF x y))
→ π(‘∀ f, ‘∀ g, (P f ∧ P g) ⇒ (P (andF f g)))
→ π(‘∀ f, ‘∀ g, (P f ∧ P g) ⇒ (P (orF f g)))
→ π(‘∀ f, ‘∀ g, (P f ∧ P g) ⇒ (P (impF f g)))
→ π(‘∀ f, (P f) ⇒ (‘∀ x, P (allF x f)))
→ π(‘∀ f, (P f) ⇒ (‘∀ x, P (exF x f)))
→ π(P tF)
→ π(P fF)
→ π(‘∀ f, P f);
```

The symbol π is used to map each proposition (of sort *El prop*) to the type of its proofs (of sort `TYPE`).

4.2 Interpretation

In this section, we define the interpretation of the datatype of formulas into the type of propositions. Since formulas can contain variables (represented as natural numbers), the interpretation depends on a valuation from natural numbers to graphs. Hence we have:

```
symbol interpretation : (El nat → El graph) → El formula → El prop;
```

The interpretation of quantifiers requires an operator for updating a valuation, taking as arguments the initial valuation, the variable that needs to be updated, and the new value of this variable:

```
symbol update : (El nat → El graph) → El nat → El graph
                                          → (El nat → El graph)
```

The computation of $update\ \sigma\ x\ a\ y$ relies on a decision procedure for the equality on natural numbers: if $x = y$ then $update\ \sigma\ x\ a\ y$ is a, other wise it is $\sigma\ y$. Rather than implementing such a decision procedure separately, we embed it in an auxiliary function:

```
symbol update1 : (El nat → El graph) → El nat → El graph → El nat
                                          → (El nat → El graph)
```

$update\ \sigma\ x\ a\ y$ reduces to $update1\ \sigma\ x\ a\ y\ y$, and $update1\ \sigma\ x\ a\ y\ z$ reduces to a if $x = y$, and $\sigma\ z$ otherwise, by decreasing x and y until one (or both) becomes *zero*. The last argument to $update1$ is thus used for storing the variable at which $update\ \sigma\ x\ a$ was initially called. The rewrite rules are as follows:

```
rule update $σ $x $a $y ↪ update1 $σ $x $a $y $y;
rule update1 $σ zero $a zero $z ↪ $a;
rule update1 $σ zero $a (s $y) $z ↪ $σ $z;
rule update1 $σ (s $x) $a zero $z ↪ $σ $z;
rule update1 $σ (s $x) $a (s $y) $z ↪ update1 $σ $x $a $y $z;
```

We can prove that *update* and $update1$ satisfy the intended properties:

Theorem 4.1

$(i)\ \forall\sigma\forall x\forall y\forall z\forall a\ ((x = y \Rightarrow update1\ \sigma\ x\ a\ y\ z = a) \wedge (\neg\ x = y \Rightarrow update1\ \sigma\ x\ a\ y\ z = \sigma\ z))$

$(ii)\ \forall\sigma\forall\sigma'\forall x\forall a\forall b\ (a \simeq b \Rightarrow (\forall y\ (\sigma\ y \simeq \sigma'\ y)) \Rightarrow \forall z\ (update\ \sigma\ x\ a\ z \simeq update\ \sigma'\ x\ b\ z))$

Now we have all the tools to define the rewriting rules of the interpretation of formulas:

```
rule interpretation $σ (eqF $x $y) ↪ ($σ $x) simeq ($σ $y);
rule interpretation $σ (inF $x $y) ↪ ($σ $x) ∈ ($σ $y);
rule interpretation $σ (andF $f $g)
                ↪ (interpretation $σ $f) ∧ (interpretation $σ $g);
rule interpretation $σ (orF $f $g)
                ↪ (interpretation $σ $f) ∨ (interpretation $σ $g);
rule interpretation $σ (impF $f $g)
                ↪ (interpretation $σ $f) ⇒ (interpretation $σ $g);
rule interpretation $σ (allF $x $f)
                ↪ ‘∀ a, interpretation (update $σ $x a) $f;
rule interpretation $σ (exF $x $f)
                ↪ ‘∃ a, interpretation (update $σ $x a) $f;
rule interpretation $σ fF ↪ ⊥;
rule interpretation $σ tF ↪ ⊤;
```

We can finally prove that *interpretation* is invariant under $\simeq$-equivalent valuations:

Theorem 4.2 $\forall f \forall \sigma \forall \sigma' \, ((interpretation \; \sigma \; f \; \wedge \; \forall x \, (\sigma \; x \; \simeq \; \sigma' \; x)) \; \Rightarrow \; interpretation \; \sigma' \; f)$

Proof. By induction over formula f. □

4.3 Comprehension, Empty Set and Inductive Set

With this syntax for restricted formulas at hand together with the interpretation in *prop*, we are finally able to define the comprehension construction in DEDUKTI:

```
symbol comp : El graph → (El nat → El graph) → El formula
                                                → El graph;
```

together with its rewrite rules

```
rule eta (comp $a $σ $f) $x $x' ↪
  (‘∃ y, ‘∃ y', (($x = i y) ∧ ($x' = i y') ∧ eta $a y y'))
  ∨ (‘∃ y, ($x = i y) ∧ ($x' = o) ∧ (eta $a y (root $a))
          ∧ (interpretation (update $σ zero (cr $a y)) $f));
rule root (comp $a $σ $f) ↪ o;
```

Moreover, we are now able to define two symbols related to the lemmas of the Infinity section [5, see Table 5]: the empty set symbol and the symbol for inductive sets *Ind* [5, see Section 2.1]. We implement *empty_set* of type *El graph* and *Ind* of type *El graph* $\rightarrow$ *El prop*. We use comprehension with the set of natural numbers and the formula fF to define the rewrite rule of the empty set:

```
rule empty_set ↪ comp omega (λ _, omega) fF;
```

We can now define inductive sets:

```
rule Ind $c ↪ (empty_set ∈ $c)
  ∧ (‘∀ a, (a ∈ $c) ⇒ ((join (pair a (pair a a))) ∈ $c));
```

Inductive sets are used in the axiom of infinity. The previous rule states that c is inductive if (i) it contains the empty set and (ii) if $a \in c$ then $a \cup \{a\} \in c$. Here we use the fact that $\cup\{a,\{a,a\}\} = \cup\{a,\{a\}\} = a \cup \{a\}$.

5 Lemmas

Dowek and Miquel proved that the theory IZmod of pointed graphs does validate all the axioms of IZst. This required 53 lemmas that were *informally* proved: each axiom of IZst corresponds to a lemma in IZmod, and we have additional intermediary lemmas on the structure of pointed graphs [5, see Tables 4 and 5]. For instance, we want to prove the axiom of pairing, that is to say that our construction $\{a,b\}$ indeed satisfies $\forall x \, (x \in \{a,b\} \Leftrightarrow (x \simeq a \vee x \simeq b))$ (lemma 43). In particular, this requires $(\{a,b\}/i(\text{root}(a))) \simeq a$ and $(\{a,b\}/j(\text{root}(b))) \simeq b$ (lemmas 36 and 37).

We *formally* prove in our implementation the 53 lemmas. Some of our proofs just follow the informal ones, while some others rely on the type of formulas and its embedding into propositions. The complete proofs can be found at `https://github.com/Deducteam/dedukti_set_theory/`.

The first two lemmas are immediate consequences of the definition of Leibniz equality in the standard library of DEDUKTI:

```
constant symbol = [s] : El s → El s → El prop;
notation = infix 4;
rule π (@= $s $x $y) ↪ Π (P : El $s → El prop), π(P $x) → π(P $y);
```

All the other lemmas of IZmod not involving *comp* (and therefore restricted formulas) are proved using the blueprint [6, see Proposition 1].

5.1 An Example of Proof

To show the way lemmas are proved in DEDUKTI we will take the example of lemma 30 and comment its proof. This lemma states that

$$a \in b \wedge a \simeq c \Rightarrow c \in b$$

Proof. We first assume graphs a, b and c and H the proof of $a \in b \wedge a \simeq c$. $a \in b$ rewrites to $\exists x\,(x\ \eta_b\ root\ b \wedge a \simeq (b/x))$.

We decompose the left part of H as x and Hx which is a proof of $x\ \eta_b\ root\ b \wedge a \simeq (b/x)$.

As the goal is to prove $c \in b$, that is, $\exists y\,(y\ \eta_b\ root\ b \wedge c \simeq (b/y))$, we need to find a suitable y. We take x and now have two goals: $x\ \eta_b\ root\ b$ and $c \simeq (b/x)$.

The first one is proved by applying the left part of Hx.

The second one is obtained by applying lemma 5 (transitivity of $\simeq$) to c, a and b/x. To apply lemma 5, we need to prove $c \simeq a \wedge a \simeq b/x$. $c \simeq a$ is proved by applying lemma 4 (symmetry of $\simeq$) to a, c and the right part of H that is a proof of $a \simeq c$. $a \simeq b/x$ derives from the right part of Hx. □

This proof is written in DEDUKTI with the following code:

```
opaque symbol lemma30 : π(‘∀ a, ‘∀ b, ‘∀ c,
  ((a ∈ b) ∧ (a simeq c)) ⇒ (c ∈ b))
≔begin
assume a b c H;
refine ex_e node _ (and_el _ _ H) _ _;
assume x Hx;
refine ex_i node x _ _;
refine and_i _ _ _ _
{refine and_el _ _ Hx}
{refine lemma5 c a (cr b x)
  (and_i _ _ (lemma4 a c (and_er _ _ H)) (and_er _ _ Hx))}
end;
```

5.2 Lemmas involving Formulas

Lemma 32 is as follows:

$$(P(z \leftarrow a) \wedge a \simeq b) \Rightarrow P(z \leftarrow b)$$

where P is a restricted formula. Therefore, we formulate lemma 32 using the type of formulas, as well as its interpretation in *prop*. The valuation $update\ \sigma\ z\ a$ represents the assignment of variable $z \leftarrow a$.

```
opaque symbol lemma32 : Π (z : El nat), Π (f : El formula),
  π(‘∀ a, ‘∀ b, (‘∀ σ : (El nat → El graph),
  ((interpretation (update σ z a) f) ∧ (a simeq b))
  ⇒ (interpretation (update σ z b) f)))
```

The proof proceeds by induction over formulas: each case is proved easily, using the lemmas that have already been checked by DEDUKTI.

Strong extensionality (lemma 41) is implemented similarly:

```
opaque symbol lemma41 : Π (x y : El nat), Π (f : El formula),
  Π (c d : El graph), π(‘∀ σ : (El nat → El graph),
  ((interpretation (update (update σ x c) y d) f)
  ∧ (‘∀ a, ‘∀ a’, ‘∀ b,
     ((a’ ∈ a)
     ∧ (interpretation (update (update σ x a) y b) f))
     ⇒ (‘∃ b’, ((b’ ∈ b)
       ∧ (interpretation (update (update σ x a’) y b’) f))))
  ∧ (‘∀ b, ‘∀ b’, ‘∀ a,
     ((b’ ∈ b)
     ∧ (interpretation (update (update σ x a) y b) f))
     ⇒ (‘∃ a’, ((a’ ∈ a)
       ∧ (interpretation (update (update σ x a’) y b’) f)))))
  ⇒ (c simeq d))
```

5.3 Weak Extensionality

The proofs of lemmas 44, 47 and 48 require the use of the axiom of (weak) extensionality [6]. Therefore, we prove in DEDUKTI that extensionality is a consequence of strong extensionality (lemma 41):

Weak extensionality. $\forall c \forall d\ (\forall z\ (z \in c \Leftrightarrow z \in d) \Rightarrow c \simeq d)$

Proof. We assume that $\forall z\ (z \in c \Leftrightarrow z \in d)$. In order to prove $c \simeq d$, we follow the blueprint given by Dowek and Miquel [5, see Proposition 1] by using an instance of the strong extensionality axiom where $R(x,y)$ is $(x \simeq c\ \wedge\ y \simeq d)\ \vee\ x \simeq y$. We are left to prove the three premises of this instance of strong extensionality:

- $(c \simeq c\ \wedge\ d \simeq d)\ \vee\ c \simeq d$ derives from the reflexivity of bisimilarity (lemma 3).
- We need to prove:

  $$\forall a \forall a' \forall b\ (a' \in a\ \wedge\ ((a \simeq c\ \wedge\ b \simeq d)\ \vee\ a \simeq b) \Rightarrow (\exists b'\ (b' \in b\ \wedge\ ((a' \simeq c\ \wedge\ b' \simeq d)\ \vee\ a' \simeq b'))$$

  We assume $(a' \in a\ \wedge\ ((a \simeq c\ \wedge\ b \simeq d)\ \vee\ a \simeq b)$ and we choose b' equal to a', so that $(a' \simeq c\ \wedge\ a' \simeq d)\ \vee\ a' \simeq a'$ is a tautology and we are left to prove $a' \in b$. If $a \simeq b$, then by lemma 31 this is a consequence of $a' \in a$. Otherwise we have $a \simeq c\ \wedge\ b \simeq d$ so with lemma 31 applied twice we are left to prove $a' \in c \Rightarrow a' \in d$, which is a consequence of $\forall z\ (z \in c \Leftrightarrow z \in d)$.
- We proceed similarly for the third premise. □

In order to implement this result in DEDUKTI we first prove the following intermediary lemma:

```
opaque symbol lemmaHypExt : Π (c d : Graph),
  π((‘∀ z, (z ∈ c) ⇔ (z ∈ d)) ⇒
  ((((c simeq c) ∧ (d simeq d)) ∨ (c simeq d))
  ∧ (‘∀ a, ‘∀ a’, ‘∀ b,
    ((a’ ∈ a) ∧ (((a simeq c) ∧ (b simeq d)) ∨ (a simeq b)))
    ⇒ (‘∃ b’, ((b’ ∈ b)
    ∧ (((a’ simeq c) ∧ (b’ simeq d)) ∨ (a’ simeq b’))))))
  ∧ (‘∀ b, ‘∀ b’, ‘∀ a,
    ((b’ ∈ b) ∧ (((a simeq c) ∧ (b simeq d)) ∨ (a simeq b)))
     ⇒ (‘∃ a’, ((a’ ∈ a)
    ∧ (((a’ simeq c) ∧ (b’ simeq d)) ∨ (a’ simeq b’)))))))
```

Then in order to prove *weak extensionality*, we assume graphs c and d, and H the hypothesis $\forall z\ (z \in c) \Leftrightarrow (z \in d)$. Then we apply lemma 41 to:

- natural numbers *zero* and *one*
- the formula $(orF\ (andF\ (eqF\ zero\ two)\ (eqF\ one\ three))\ (eqF\ zero\ one))$
- graphs c and d
- the valuation $(update\ (update\ (\lambda\_, empty\_set)\ two\ c)\ three\ d)$
- the proof $lemmaHypExt\ c\ d\ H$.

Indeed, in the formula $(orF\ (andF\ (eqF\ zero\ two)\ (eqF\ one\ three))\ (eqF\ zero\ one))$, *two* will be interpreted by c and *three* by d, thanks the valuation $(update\ (update\ (\lambda\_, empty\_set)\ two\ c)\ three\ d)$. Finally we obtain extensionality:

```
opaque symbol lemmaExt : Π (c d : El graph),
  π((‘∀ x, (x ∈ c) ⇔ (x ∈ d)) ⇒ (c simeq d))
≔begin
assume c d H;
refine lemma41 zero one
  (orF (andF (eqF zero two) (eqF one three)) (eqF zero one))
  c d (update (update (λ _, empty_set) two c) three d)
  (lemmaHypExt c d H)
end;
```

5.4 The Axioms of IZst Theory

We have now encoded in DEDUKTI all the axioms of IZst set theory: the strong extensionality axiom corresponds to lemma 41, the axiom of the union is implemented by lemma 42, the pairing axiom corresponds to lemma 43, the axiom of the power set is encoded by lemma 44, the comprehension scheme is implemented by lemma 45, the axiom of infinity corresponds to lemma 51 and the transitive closure axiom is encoded by lemmas 52 and 53.

6 Conclusion

We have implemented in DEDUKTI a version of set theory — IZst — that corresponds to Zermelo set theory, with the Strong Extensionality axiom and the Transitive Closure axiom. To do so, we have adapted the work by G. Dowek and A. Miquel [5] from *Deduction modulo theory* to $\lambda\Pi$-*calculus modulo theory* and have encoded sets with a structure of pointed graphs of the IZmod theory. We have *formally* written all the proofs of the lemmas allowing us to implement set theory in DEDUKTI.

To define and prove the lemmas corresponding to the Comprehension axiom, we have developed a language of formulas, along with operators *interpretation* and *update*. In particular, the language of formulas allows us to prove that the Extensionality axiom derives from the Strong Extensionality axiom.

Historically, the encoding of sets by pointed graphs had been designed to enjoy the normalization property. IZmod expressed in *Deduction modulo theory* does so, but the case of our implementation in $\lambda\Pi$-*calculus modulo theory* remains to be investigated.

The implementation of IZmod theory represents the first significant corpus of formal proofs in LAMBDAPI. This implementation paves the way for interoperability between DEDUKTI and theorem provers based on set theory, such as ISABELLE/ZF.

Acknowledgments

The authors would like to thank Frédéric Blanqui for his precious insight on the internal functioning of LAMBDAPI.

References

[1] P. Aczel (1988): *Non well-founded sets*. Center for the Study of Language and Information, Stanford.

[2] A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant & R. Saillard (2016): *Dedukti: a Logical Framework based on the* $\lambda\Pi$*-Calculus Modulo Theory*. Manuscript.

[3] F. Blanqui, G. Dowek, E. Grienenberger, G. Hondet & F. Thiré (2021): *Some Axioms for Mathematics*. *Formal Structures for Computation and Deduction*.

[4] M. Crabbé (1974): *Non-normalisation de la théorie de Zermelo*. Manuscript.

[5] G. Dowek & A. Miquel (2007): *Cut elimination for Zermelo set theory*. Manuscript.

[6] G. Dowek & A. Miquel (2007): *Cut elimination for Zermelo set theory: Proof of 53 easy lemmas*. Manuscript.

Appendix

The complete proofs can be found at `https://github.com/Deducteam/dedukti_set_theory/`.

Lemma	Number of lines in the proof	Lemma	Number of lines in the proof
3	26	29	17
4	14	30	10
5	37	31	12
6	33	32	49
7	12	33	33
8	5	34	33
9	12	35	9
10	12	36	9
11	5	37	9
12	5	38	9
13	5	39	9
14	37	40	9
15	40	41	42
16	48	Weak extensionality	47
17	48	42	49
18	38	43	39
19	44	44	133
20	90	45	46
21	34	46	11
22	35	47	18
23	34	48	165
24	38	49	11
25	31	50	23
26	38	51	6
27	29	52	17
28	33	53	31