# Dedukti: an Implementation of Set Theory with Pointed Graphs

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

We show how to implement a version of set theory in DEDUKTI. To do so, we adapt in $\lambda\Pi$–*calculus modulo theory* the guideline given in *Deduction modulo theory* by G. Dowek and A. Miquel –using a theory of pointed graphs– and we *formally* write the proofs in DEDUKTI. To achieve this goal, we develop a language of formulas and we use a structural induction.

## 1 Introduction

Several prover are based on set theory, such as MIZAR, ATELIER B and ISABELLE/ZF. The goal of this paper is to present an implementation of set theory in the logical framework DEDUKTI [2] expressed in the $\lambda\Pi$-*calculus modulo theory*. Our long term project is to use this implementation to enable interoperability between such provers based on set theory. We use here one of DEDUKTI's implementation, called LAMBDAPI.

The first way to implement set theory in DEDUKTI is to state each axiom in the framework. But such a method does not yield a cut-elimination theorem for set theory. In particular, it does not enable to extract witnesses from a constructive existence proof. An alternative is to orient these axioms as rewriting rules. For instance,

$$x \in \mathscr{P}(y) \Leftrightarrow x \subseteq y$$

is replaced with

$$x \in \mathscr{P}(y) \longrightarrow x \subseteq y$$

However, as pointed out by M. Crabbé [4], such a formulation of set theory does not have a cut-elimination property. An other solution is to define sets as *pointed graphs* [1]. With such a formulation, we can prove a cut-elimination theorem: for every proof in natural deduction that contains cuts there exists a cut-free proof. Such a theory expressed in *Deduction modulo theory* is presented in [5] with pencil and paper proofs. The goal of this paper is to present an implementation of this theory.

In 2007, the $\lambda\Pi$-*calculus modulo theory* had not been defined and this is why the original theory was defined in *Deduction modulo theory*. The first issue was to adapt the expression in *Deduction modulo theory* to the expression in $\lambda\Pi$-*calculus modulo theory*. This led to simplify the original formulation as classes of nodes of pointed graphs and relations on nodes can be naturally expressed in DEDUKTI while that required a specific sort and specific comprehension axiom in *Deduction modulo theory*. This enabled to reduce the size of the signature from 31 symbols to 26.

On the other hand, in the original formulation of the theory some lemmas were only valid for a specific subset of propositions. These lemmas were proved by induction over the structure of the propositions of this subset. Implementing these lemmas in DEDUKTI required to define a new inductive sort for

Submitted to:
LFMTP 2022

the formulas of this subset and an interpretation of these formulas into the general type of propositions, itself defined with rewriting rules.

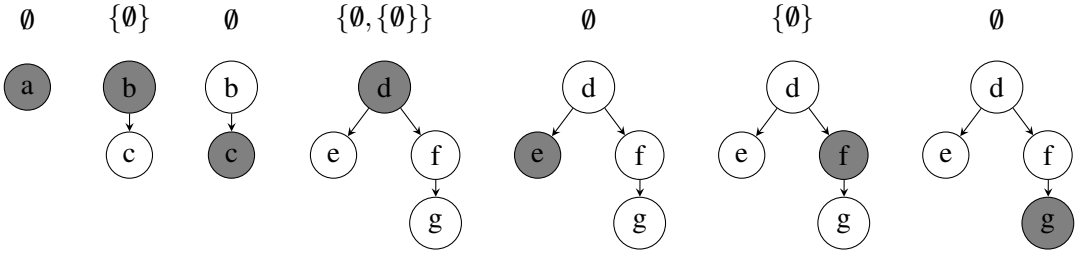
This development is the first, to our knowledge, to use such a reflection principle defined with rewriting rules in DEDUKTI. The generality of this method still remains to be investigated.

# 2 The Theory of Pointed Graphs

The set theory with pointed graphs developed by G. Dowek and A. Miquel is called IZmod (Intuitionistic Zermelo set theory in *Deduction modulo*). 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. A pointed graph is a direct graph whose one of its node is identified as the root.




We have represented several pointed graphs where the root is indicated by the filled circle. We see that the location of the root is important as it changes the set represented by the pointed graph. In the third graph, the node $b$ has no effect because the root is $c$.

The operator *root* gives the root of a pointed graph. $a/x$ replace the root of $a$ by $x$. $x\ \eta_a\ y$ means there is a edge in $a$ from $y$ to $x$. We have the following rewriting rules:

$$x\ \eta_{a/z}\ y \longrightarrow x\ \eta_a\ y$$

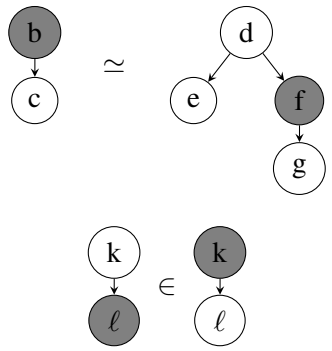
$$root(a/x) \longrightarrow x$$

$$(a/x)/y \longrightarrow a/y$$

We see in the examples that some different pointed graphs can represent the same set. To characterized this, a notion of bisimilarity, noted $\simeq$, is introduced between graphs with rewriting rule

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

In set theory, there only exists one sort: the sets. In the IZmod theory, there exists two sorts: the nodes and the pointed graphs. Therefore, there are in IZmod two constructions related to the inclusion




of sets. We can take the example of $\emptyset \in \{\emptyset\}$. We can represents this inclusion in two ways, either with a relation between pointed graphs or with a relation between nodes within a same graph.

When in a graph we have an edge from $k$ to $\ell$, the set represented by the pointed graph with the root $\ell$ is an element of the set represented by the pointed graph with root $k$. But any pointed graph bisimilar with pointed graph with root $\ell$ also represents a set that is an element of the set represented with the pointed graph with root $k$. This leads to the definition of membership noted $\in$, with rewriting rule $a \in b \longrightarrow \exists x\ (x\ \eta_b\ root(b)\ \wedge\ a \simeq (b/x))$.

The objective of the IZmod theory is to composed pointed graphs, for instance to *join* pointed graphs. The root of the new pointed graph is $o$. To guarantee that $o$ is not a node of one of the original pointed graphs, an injective function $i$ is introduced, with $i'$ its left inverse and $I$ the predicate of its image.

$$i'(i(x)) \longrightarrow x$$
$$I(x) \longrightarrow \top$$
$$I(o) \longrightarrow \bot$$

The same method is applied to constructors *pair*, *powerset*, *etc*.

## 2.2 Set Theories

These pointed graphs permit to prove the axioms of a theory IZst that lies between Zermelo theory (Z) and Zermelo-Fraenkel theory (ZF). This theory does not contain the Replacement scheme but it contains two additional axioms: the Strong Extensionality axiom and the Transitive Closure axiom.

**Strong Extensionality axiom.**

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

where $R(a,b)$ is a formula of free variables $x_1, ..., x_n$.

The hypothesis of the Strong Extensionality axiom imitates the structure of the rewriting rule of $\eta$. This can be explained with the following reasoning: in the Extensionality axiom, the hypothesis states

that two sets have the same elements, so the hypothesis of the strong extensionality axiom copy the structure of one of the equivalent to the inclusion in IZmod –that is to say the constructor $\eta$.

**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.

The Strong Extensionality axiom can be deduced from the Foundation axiom and the Transitive Closure axiom can be derived from the Replacement Scheme.

The Strong Extensionality axiom implies the Extensionality axiom[1].

The IZmod theory is an extension of IZst set theory with an encoding of pointed graphs.

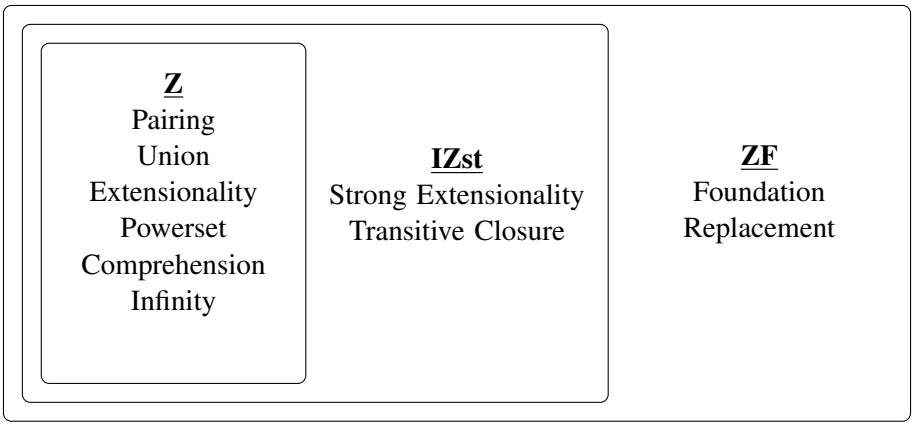



Figure 1: View of the different set theories

# 3 The Language of Pointed Graphs

## 3.1 Sorts

The language of the theory IZmod uses four sorts [5, see Section 3.2]. The first two are for the pointed graphs and for the nodes of the pointed graphs. In DEDUKTI, we would need two universal quantifiers and two existential quantifiers, one for each sort. We rather use another solution [3] that is to declare a constant *Set* of type `TYPE` for codes of sorts, a function *El* of type *Set* $\rightarrow$ `TYPE`, two constants *graph* and *node* of sort *Set*.

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

The two other sorts of the theory IZmod are for classes of nodes and for binary relations on nodes. In DEDUKTI, the sort of classes is just *El node* $\rightarrow$ *El prop* and that of binary relations *El node* $\rightarrow$ *El node* $\rightarrow$ *El prop*. To quantify on such sorts, we introduce constant *arrow* of type *Set* $\rightarrow$ *Set* $\rightarrow$ *Set* and rewrite rule

$$El\ (x\ arrow\ y) \longrightarrow (El\ x) \rightarrow (El\ y)$$

[1] See Section 5.3

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

### 3.2 Signature

The signature of IZmod contains 31 symbols [5, see Table 2]. As we have replaced the sorts for classes and relations with the DEDUKTI types *El node* → *El prop* and *El node* → *El node* → *El prop*, we do not need specific predicate symbols to apply a class to a node or a relations to two. In the same way, we do not need comprehension axioms for classes and relations.

Similarly, the equality symbol is part of the standard library of DEDUKTI.

The signature is thus reduced to 26 symbols. The specific case of the comprehension symbol is treated later.

```
symbol eta : El graph → El node → El node → El prop;
symbol root : El graph → El node;
symbol cr : El graph → El node → El graph;
constant symbol o : El node;
constant symbol 0 : 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;
symbol ρ : El graph → El node;
symbol ρ' : El node → El graph;
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;
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;
symbol closure : El graph → El graph;
```

### 3.3 From *Deduction modulo theory* to *λΠ-calculus modulo theory*

Unlike *Deduction modulo theory*, *λΠ-calculus modulo theory* allows quantification over propositions.

Thus the symbols *mem* and *rel* [5, see Table 2] are not needed anymore. Indeed, a class of nodes can simply be expressed as a term $P$ of type *El node* → *El prop*, and the fact that a node $x$ is an element of this class can be expressed as $P\ x$ and not as $mem(x,P)$. In a similar way, the fact that two nodes $x$ and $y$ are related by a relation $r$ is now simply expressed with $r\ x\ y$.

The symbols $g_{x,y_1,\ldots,y_n,P}$ and $g'_{x,x',y_1,\ldots,y_n,P}$ had been created as specific constructors that allow to introduce in *mem* a proposition $P$ by convert it into a class. These symbols are now not necessary in DEDUKTI since *mem* is deleted.

### 3.4 Rewriting Rules

All the other rewriting rules [5, see Table 3], except the ones involving the symbol associated with the construction of sets by comprehension (noted *comp*), 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));

symbol omega : El graph;
rule eta omega $x $x' ↪
  (‘∃ y, ‘∃ y', ($x = i y) ∧ ($x' = i y') ∧ (y < y'))
    ∨ (‘∃ y, ($x = i y) ∧ ($x' = o) ∧ Nat y);

symbol closure : El graph → El graph;
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 [6, see Table 2]. Indeed, the informal presentation of the representation of sets by pointed graphs given in 2.1 corresponds to Von

Neumann ordinals. The rewriting rule of `eta omega x x'` states that in `omega` the edges represents the order relation $<$ and that there is an edge from every natural number to the root of `omega`.

# 4 The Language of Formulas

A key point in the original formulation of the theory is that the lemmas in which the *comp* symbol – enabling to construct a set by comprehension– is used are only valid for a subset of formulas [5, see Table 5]. The formulas need to have all of its quantifiers of sort *El graph* and can only use the language $\in$, $\simeq$ and the classic logical connectives. Taking this restriction into account was the main challenge of this implementation.

Thus we need to introduce a set to characterize the validity domain of such lemmas.

## 4.1 Formulas

In order to achieve this goal, we define the constant *formula* of type *Set* and the logical connectives related to this constant.

```
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;
```

Then we are able to define an induction over formulas, using the language $\in$, $\simeq$ and the classic logical connectives.

```
constant symbol recF : Π (P : El formula → 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);
```

## 4.2 Interpretation

The next step is to interpret an object of type *formula* into *Prop*. We introduce the constant *interpretation* which receives a valuation of type *El nat* → *El graph* and a formula of type *El formula* and return a *El prop*.

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

We need to have a tool to update a valuation when we assign a variable. To do so, we introduce the constant $update$ of type $(El\ nat \to El\ graph) \to El\ nat \to El\ graph \to (El\ nat \to El\ graph)$ which takes as arguments a valuation $\sigma$, a natural number $x$ and a graph $a$ and returns a new valuation $(update\ \sigma\ x\ a)$ that substitutes $x$ by $a$ and acts like $\sigma$ for the other natural numbers.

To write a rewriting rule upon $update$, we need to be able to check if we apply $(update\ \sigma\ x\ a)$ to $y = x$ (which is substituted by $a$) or to $y \neq x$ (which is substituted by $\sigma\ y$.

We define the symbol $update1$ of type $(El\ nat \to El\ graph) \to El\ nat \to El\ graph \to El\ nat \to (El\ nat \to El\ graph)$.

The technique we use to rewrite $(update\ \sigma\ x\ a)\ y$ is the following:

- We keep in memory $y$ in the new argument $z$ of $update1$

$$(update\ \sigma\ x\ a)\ y \longrightarrow update1\ \sigma\ x\ a\ y\ y$$

- We decrement $x$ and $y$ until one of them equals zero

$$update1\ \sigma\ (s\ x)\ a\ (s\ y)\ z \longrightarrow update1\ \sigma\ x\ a\ y\ z$$

- If both are equal to zero, then $x$ and $y$ are equal and we return $a$.

$$update1\ \sigma\ zero\ a\ zero\ z \longrightarrow a$$

If only one equals zero, then they are different and we return $\sigma\ z$.

$$update1\ \sigma\ zero\ a\ (s\ y)\ z \longrightarrow \sigma\ z$$

$$update1\ \sigma\ (s\ x)\ a\ zero\ z \longrightarrow \sigma\ z$$

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

$$interpretation\ \sigma\ (eqF\ x\ y) \longrightarrow (\sigma\ x) \simeq (\sigma\ y)$$

$$interpretation\ \sigma\ (inF\ x\ y) \longrightarrow (\sigma\ x) \in (\sigma\ y)$$

$$interpretation\ \sigma\ (andF\ f\ g) \longrightarrow (interpretation\ \sigma\ f) \wedge (interpretation\ \sigma\ g)$$

$$interpretation\ \sigma\ (orF\ f\ g) \longrightarrow (interpretation\ \sigma\ f) \vee (interpretation \sigma\ g)$$

$$interpretation\ \sigma\ (impF\ f\ g) \longrightarrow (interpretation\ \sigma\ f) \Rightarrow (interpretation\ \sigma\ g)$$

$$interpretation\ \sigma\ (allF\ x\ f) \longrightarrow \forall a, interpretation\ (update\ \sigma\ x\ a)\ f$$

$$interpretation\ \sigma\ (exF\ x\ f) \longrightarrow \exists a, interpretation\ (update\ \sigma\ x\ a)\ f$$

$$interpretation\ \sigma\ fF \longrightarrow \bot$$

$$interpretation\ \sigma\ tF \longrightarrow \top$$

### 4.3 Results concerning Valuation

Thanks to the introduction of *interpretation*, we can deduce five theorems.

The first theorem is used to simplify the terms when updating a valuation.

**Theorem 4.1** $\forall\sigma\forall x\forall y\forall z\forall a\,((x=y \Rightarrow update1\ \sigma\ x\ a\ y\ z \simeq a) \wedge (\neg\, x=y \Rightarrow update1\ \sigma\ x\ a\ y\ z \simeq \sigma\ z))$

*Proof.* The first term of the conjunction is proved by simple recurrence over natural numbers. The second term of the conjunction is proved by double recurrence. □

The second theorem conveys the idea that if two graphs are bisimilar then it is identical to update a valuation by either of theses two graphs.

**Theorem 4.2** $\forall\sigma\forall x\forall a\forall b\,(a \simeq b \Rightarrow \forall y\,(update\ \sigma\ x\ a\ y \simeq update\ \sigma\ x\ b\ y))$

**Theorem 4.3** $\forall\sigma\forall x\forall y\forall a\forall b\forall c\,(a \simeq b \Rightarrow \forall z\,(update\ (update\ \sigma\ x\ a)\ y\ c\ z \simeq update\ (update\ \sigma\ x\ b)\ y\ c\ z))$

The fourth theorem states that if two valuations are equal they keep being equal after an update.

**Theorem 4.4** $\forall\sigma\forall\sigma'\forall x\forall c\,((\forall y\,(\sigma\ y \simeq \sigma'\ y)) \Rightarrow \forall z\,(update\ \sigma\ x\ c\ z \simeq update\ \sigma'\ x\ c\ z))$

**Theorem 4.5** $\forall f\forall\sigma\forall\sigma'\,((interpretation\ \sigma\ f \wedge \forall x\,(\sigma\ x \simeq \sigma'\ x)) \Rightarrow interpretation\ \sigma'\ f)$

*Proof.* The fifth theorem is proved by induction over formulas. □

### 4.4 Comprehension, Empty Set and Inductive Set

Henceforth, we are able to define in DEDUKTI

$$comp : El\ graph \to (El\ nat \to El\ graph) \to El\ formula \to El\ graph$$

and its rewriting 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;
```

When it comes to the symbol related to the Infinity section of [5, see Section 2.1], we implement *empty_set* of type *El graph* and *Ind* of type *El graph* → *El prop*.

To define the empty set, we use *comp* with the formula $fF$:

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

Then we implement

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

# 5 Lemmas

A first result on the original formulation of the theory was to prove that the axioms of IZst are theorems in IZmod. This required 53 lemmas that were *informally* proved in [5, see Tables 4 and 5] and that we prove *formally* in this paper.

Some of these proofs just follow the informal ones. Some others require the use of the type of formula and its embedding into propositions.

The first lemma $x = x$ does not need to be implemented since it is already part of the standard library of DEDUKTI under the name `refl` (which is polymorphic).

The second lemma is already a consequence of the rewriting rule of the polymorphic $=$ implemented 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 except the ones where *comp* is involved are proved using the blueprint in [6, see Proposition 1]. The complete proofs can be found in `https://github.com/Deducteam/dedukti_set_theory/`.

## 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 make appear $x$ and $Hx$ the proof $x\ \eta_b\ root\ b \wedge a \simeq (b/x)$.

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

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

The second one is obtained by applying lemma 5 to $c$, $a$ and $b/x$. To apply lemma 5, we need to prove $c \simeq a \Rightarrow a \simeq b/x$. $c \simeq a$ is proved applying reflexivity to $a \simeq c$ (i.e. applying lemma 4 to $a$, $c$ and the right part of $H$). $a \simeq b/x$ 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

Now that the language of formulas have been designed along with the implementation of the *comp* symbol, lemma 32 can be implemented.

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

We implement it thanks to the *interpretation* symbol. 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 of this `opaque symbol` is done by induction over formulas: each case is proved easily, using the lemmas that have already been checked by DEDUKTI.

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 Extensionnality

We notice in [6] the use of weak extensionality (simply called extensionality) to prove lemmas 44, 47 and 48. We want to deduce weak extensionality from strong extensionality (i.e. lemma 41).

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

We follow the blueprint given by G. Dowek and A. Miquel [5, see Proposition 1] : we use the strong extensionality axiom where $R(x,y)$ is $(x \simeq c\ \wedge\ y \simeq d)\ \vee\ x \simeq y$.

*Proof.* We want to prove that $\forall c \forall d\ (\forall z\ (z \in c \Leftrightarrow z \in d) \Rightarrow c \simeq d)$. We assume that $\forall z\ (z \in c \Leftrightarrow z \in d)$. We want to apply strong extensionality to deduce $c \simeq d$. To do so, we need to prove the three terms of the hypothesis of strong extensionality.

$(c \simeq c \wedge d \simeq d) \vee c \simeq d$ is a tautology.

We want to prove that $\forall a \forall a' \forall b\ (a' \in a \wedge ((a \simeq c \wedge b \simeq d) \vee a \simeq b) \Rightarrow (\exists b'\ (b' \in b \wedge ((a' \simeq c \wedge b' \simeq d) \vee a' \simeq b'))$. We assume $(a' \in a \wedge ((a \simeq c \wedge b \simeq d) \vee a \simeq b)$. If $(a \simeq c \wedge b \simeq d)$, then we choose $b' \simeq a'$. We have $b' \in a$ because $a' \in a$. Yet, $a \simeq c$, $c \simeq d$ and $d \simeq b$. Then $b' \in b$. If $a \simeq b$, we choose $b' \simeq a'$. We have $b' \in a$ because $a' \in a$. Yet, $a \simeq b$. Thus $b' \in b$.

We proceed similarly for the third term. □

We implement this theorem in DEDUKTI:

```
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’)))))))
```

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 of the left hand term $lemmaHypExt\ c\ d\ H$.

Indeed, in the formula

$$(orF\ (andF\ (eqF\ zero\ two)\ (eqF\ one\ three))\ (eqF\ zero\ one))$$

*zero* and *one* will be interpreted by $c$ and $d$ thanks to lemma 41 and *two* will be interpreted by $c$ and *three* by $d$ thanks the valuation

$$(update\ (update\ (\lambda\_, empty\_set)\ two\ c)\ three\ d)$$

The proposition obtained corresponds to the proposition in lemmaHypExt.

```
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 a significant corpus of formal proofs in LAMBDAPI.

## Annex

| 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 |

## 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 λΠ-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.