Semantics of Intensional Type Theory extended with Decidable Equational Theories [*]

Qian Wang[1,3,4,5,6] and Bruno Barras[1,2]

1 Laboratoire d'Informatique de L'École Polytechnique
2 INRIA Saclay – Île de France
3 Key Laboratory for Information System Security, Ministry of Education
4 Tsinghua National Laboratory for Information Science and Technology (TNList)
5 School of Software, Tsinghua University
6 Department of Computer Science and Technology, Tsinghua University

Abstract

Incorporating extensional equality into a dependent intensional type system such as the Calculus of Constructions (CC) provides with stronger type-checking capabilities and makes the proof development closer to intuition. Since strong forms of extensionality generally leads to undecidable type-checking, it seems a reasonable trade-off to extend intensional equality with a decidable first-order theory, as experimented in earlier work on CoqMTU and its implementation CoqMT.

In this work, CoqMTU is extended with strong eliminations. The meta-theoretical study, particularly the part relying on semantic arguments, is more complex. A set-theoretical model of the equational theory is the key ingredient to derive the logical consistency of the formalism. Strong normalization, the main lemma from which type-decidability follows, is proved by attaching realizability information to the values of the model.

The approach we have followed is to first consider an abstract notion of first-order equational theory, and then instantiate it with a particular instance, Presburger Arithmetic. These results have been formalized using Coq.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Calculus of Constructions, Extensional Type Theory, Intensional Type Theory, Model, Meta-theory, Consistency, Strong Normalization, Presburger Arithmetic

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.653

1 Introduction

Most proof assistants, such as Coq [19], implement intensional type theory because extensional type theory is usually undecidable. Coq implements ECIC, a type theory that extends CC [8] with two more features: inductive types in the style of the Calculus of Inductive Constructions (CIC) [12], and a predicative hierarchy of universes, in the style of the Extended Calculus of Constructions (ECC) [10].

The purely intensional version of type-theory may become awkward when it comes to programming with dependent types. In the well-known examples of "vectors", one has the type $\mathsf{Vect}(n)$ of lists of length n, a concatenation function @ such that $v_1@v_2$ has length

[*] Supported by National Science Foundation of China grant 61272002, Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation 2011-9, Major Research plan of the National Natural Science Foundation of China grant 91218302, National Basic Research Program of China (973 Program) grant 2010CB328003.

$n_1 + n_2$ whenever v_i has length n_i. But showing that $v@\mathsf{nil} = v$ is not possible because it is not even a well-typed statement: lengths $n + 0$ and n are not identified because $+$ is defined recursively on the first argument (n here) which is not in the form of 0 or successor.

Several works tried to fix this problem by either adding rewrite rules to increase the computing ability [5] or including in extensional equalities [14, 15, 11]. Unfortunately, such solutions have never been implemented as a proof assistant until CoqMT [17] which is evolved from [6, 7, 16] and further generalized by CoqMTU [4]. The idea is a trade-off between decidability and type-checking capabilities, only allowing decidable extensional theory and automatically checking the theory equality by a decision procedure. Though this solution looks nice, the meta-theory of CoqMTU is not well understood yet: confluence and subject reduction of the full calculus are proved, but strong normalization (SN) and consistency are only proved in absence of strong elimination.

Our first contribution, shown in Section 4, is that formalizing a new schema (CCT) incorporating an abstract decidable theory into CC family, which is not only ECIC implemented by Coq, but a further extension of it. The abstract theory can be instantiated by any concrete one of interest if it can be represented in CCT and satisfies special assumptions to ensure the main meta-theoretical properties. CCT captures many calculus, but we provide a uniform way of establishing the meta-theoretical properties proved semantically.

Our second contribution, shown in Section 5, is about proving the key properties of CCT: consistency and strong normalization (with strong elimination), the basis for proving the meta-theory of other calculus extending CC and admitting a decidable theory. As often, this requires a set-theoretical model. We have based our study on the work of the second author [3, 2], that provides a modular framework for modeling a wide range of type theories from CC to the formalism of Coq. In this article, we show that this framework can be reused and that it accommodates the extra features of CCT.

To implement CICUT as a proof assistant, we must investigate its syntactic properties, among which only Church-Rosser can not be proved in the usual way, because we do not embed extensional equations into ι-reduction as CoqMTU does. Nevertheless, it is not a disaster because we believe Church-Rosser still hold and will study the whole syntactic properties of CICUT in the future.

Finally, our last contribution, shown in Section 6, is to show that Presburger Arithmetic fits perfectly in this abstract notion of decidable first-order theory, which also demonstrates our abstraction philosophy works well.

In the following sections, most of the proofs will not be shown in the paper due to the page limit. They have been done in the Coq development and available to whom are interested[1]. Many similar notations are overloaded to avoid ambiguity, such as λ is overloaded by $\dot{\lambda}$ and $\check{\lambda}$ to represent abstractions of set and pure λ-term respectively. We will detail when we need. The notations without overloading are Coq primitives, such as $\forall$ for universal quantification because the whole work is formally done in Coq. To make the concepts easier to understand, we use $\rightarrow$ for function types only, and use $\Rightarrow$ for implication. We also try to hide De Bruijn index, though it is heavily used in the development.

2 CICUT

Since the focus of this paper is the semantic meta-theory, the syntax of CICUT will not be exhaustively introduced here. The core of CICUT is almost the same as that of the CC.

[1] The development is available at `https://github.com/superwalter/SETheory-dev`

$$\frac{}{\Gamma \vdash} \qquad \frac{\Gamma \vdash T : s}{\Gamma\,[x:T] \vdash} \qquad \frac{\Gamma \vdash, \quad \Gamma(n) = (x:T)}{\Gamma \vdash n :\uparrow^{n+1} T} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Prop} : \mathsf{Kind}}$$

$$\frac{\Gamma \vdash T : s_1, \quad \Gamma\,[x:T] \vdash U : s_2}{\Gamma \vdash \Pi x : T.\,U : s_2} \qquad \frac{\Gamma\,[x:T] \vdash M : U}{\Gamma \vdash \lambda x : T.M : \Pi x : T.U}$$

$$\frac{\Gamma \vdash M : \Pi x : T.U, \quad \Gamma \vdash N : T}{\Gamma \vdash M\,N : U[x\backslash N]} \qquad \frac{\Gamma \vdash M : T, \quad \Gamma \vdash T = T' : s}{\Gamma \vdash M : T'}$$

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash M = M : T} \qquad \frac{\Gamma \vdash M = M' : T}{\Gamma \vdash M' = M : T} \qquad \frac{\Gamma \vdash M_1 = M_2 : T, \quad \Gamma \vdash M_2 = M_3 : T}{\Gamma \vdash M_1 = M_3 : T}$$

$$\frac{\Gamma \vdash T = T' : s_1, \quad \Gamma\,[x:T] \vdash U = U' : s_2}{\Gamma \vdash \Pi x : T.U = \Pi x : T'.U' : s_2} \qquad \frac{\Gamma \vdash T = T' : s_1, \quad \Gamma\,[x:T] \vdash M = M' : U}{\Gamma \vdash \lambda x : T.M = \lambda x : T'.M' : \Pi x : T.U}$$

$$\frac{\Gamma \vdash M = M' : \Pi x : T.U, \quad \Gamma \vdash N = N' : T}{\Gamma \vdash M\,N = M'\,N' : U[x\backslash N]} \qquad \frac{\Gamma\,[x:T] \vdash M = M' : U, \quad \Gamma \vdash N = N' : T}{\Gamma \vdash (\lambda x : T.M)\,N = M'[x\backslash N'] : U[x\backslash N]}$$

Figure 1 Calculus of Constructions, judgmental presentation.

Figure 1 shows a judgmental presentation of the typing rules of CC, it is different from the usual one by replacing an equivalence relation on untyped λ-terms by equality judgments. $\Gamma \vdash M = M' : T$ expresses that M is equal to M', both being of type T in context Γ. [13] shows these two representations are equivalent, and we will extend this conclusion for all of the extensions considered in this article, leaving the proof to the future.

The main novel feature of CICUT and CoqMTU in the syntax of terms is the embedding of a type of first-order terms. Regarding the typing rules, the main difference is the extension of the definitional equality with a decidable theory $\sim_{\mathcal{T}}$ on these first-order terms. See [4] for a more detailed presentation. Let us just give the extra inference rules (besides Presburger arithmetic axioms) that need to be considered to have CICUT instantiated with Presburger arithmetic. It introduces three canonical constants and a defined symbol (Rec):

$$\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{nat} : \mathsf{Kind}} \qquad \frac{\Gamma \vdash}{\Gamma \vdash 0 : \mathsf{nat}} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathsf{S} : \mathsf{nat} \to \mathsf{nat}}$$

$$\frac{\Gamma \vdash P : \mathsf{nat} \to s, \quad \Gamma \vdash N : \mathsf{nat}, \quad \Gamma \vdash M_0 : P\,0, \quad \Gamma \vdash M_1 : \Pi n : \mathsf{nat}.\,P\,n \to P\,(\mathsf{S}\,n)}{\Gamma \vdash \mathsf{Rec}(P, N, M_0, M_1) : P\,N}$$

$$\frac{\Gamma \vdash \quad M \sim_{\mathcal{T}} N}{\Gamma \vdash M = N : \mathcal{T}} \qquad \frac{N \sim_{\mathcal{T}} 0}{\mathsf{Rec}(P, N, M_0, M_1) \sim_{\mathcal{T}} M_0}$$

$$\frac{N \sim_{\mathcal{T}} \mathsf{S}\,N'}{\mathsf{Rec}(P, N, M_0, M_1) \sim_{\mathcal{T}} M_1\,N'\,\mathsf{Rec}(P, N', M_0, M_1)}$$

The main difference between CICUT and CoqMTU is the abandon of incorporating definitional equalities into reductions and restore the strong elimination, which is witnessed by the fact that the eliminator of natural numbers (Rec) can be used with a term P belonging to any sort s. By contrast, weak eliminations are obtained by restricting s to the Prop, the sort of propositions. In other words, weak elimination provides the induction principle of the natural numbers, but not the possibility to define functions by structural induction.

3 Extensible set-theoretical realizability model of CC

This section gives a short introduction to a general method used to build consistency and strong normalization models of a wide range of type-theories with dependent types. See [2] for more details.

Consistency of such formalisms is often achieved by providing a set-theoretical model, that interprets terms and types as sets. The judgment that a term has a given type is interpreted by the proposition that the term is interpreted by a member of the interpretation of the type. The soundness of such a model implies the consistency of the formalism, as soon as one type (representing the absurd proposition) is interpreted by the empty set.

Strong normalization (SN) is the property that any well-typed term of a type system cannot be reduced ad infinitum. It often captures the logical strength of the type system seen as a logical formalism. This is why it generally requires a particular model construction. Types are not mere sets of values anymore, but they should also be interpreted as sets of strongly normalizing λ-terms, that represent the possible terms that have this type. The latter are often called “realizers”. The soundness of the model shall also require that the term can be interpreted by a realizer of its type. SN is thus a consequence of the construction of such a realizability model.

The main ingredient in this model construction is the notion of saturated sets due to Tait [18] (but Girard’s reducibility candidates serve the same purpose). They are sets of strongly normalizing λ-terms such that the type constructors (such as the arrow type or intersection) can be interpreted.

SN of CC and CIC in presence of weak elimination can be proved in this setting. However, to take care of strong elimination, we need a more subtle definition of realizers: a saturated set for the value of each type is not enough, each member of the value of a type should have its own saturated set of realizers (see the R function below). This is closely related to the notion of Λ-sets introduced by Altenkirch [1].

In the remainder of this section, we show how the model construction can be carried out, provided we can implement the two signatures below: the first one gathers what is required to build a set-theoretical model (the set-denotation, symbols will be overloaded with a $\dot{}$); the second one corresponds to the extra requirements to have a full realizability model (the term-denotation, symbols will be overloaded with a $\check{}$), from which SN will follow.

3.1 Abstract Parameterized Models

The first abstract model is designed to provide a uniform structure of set denotations to all the models, hiding the implementation until instantiation. It contains the set-denotation for all closed terms in the model together with the properties to ensure soundness. It uses a higher-order presentation: binding constructions are represented using functional arguments.

▶ **Definition 1** (Abstract model)**.** The model has the following signature:

$$\mathsf{X} : \mathit{Type} \qquad \dot{\in} : \mathsf{X} \to \mathsf{X} \to \mathit{Prop} \qquad \dot{=} : \mathsf{X} \to \mathsf{X} \to \mathit{Prop} \qquad \bigstar : \mathsf{X}$$

$$\dot{@} : \mathsf{X} \to \mathsf{X} \to \mathsf{X} \qquad \dot{\lambda} : \mathsf{X} \to (\mathsf{X} \to \mathsf{X}) \to \mathsf{X} \qquad \dot{\Pi} : \mathsf{X} \to (\mathsf{X} \to \mathsf{X}) \to \mathsf{X}$$

satisfying certain properties (the followings are some examples):

$$\frac{N \dot{\in} A}{(\dot{\lambda} x \dot{\in} A.f) \dot{@} N \doteq F(N)}[\beta] \quad \frac{\forall x \dot{\in} A.F(x) \dot{\in} \bigstar}{(\dot{\Pi} x \dot{\in} A.F) \dot{\in} \bigstar}[\textsc{Imp}] \quad \frac{\forall x \dot{\in} A.(f(x)) \dot{\in} (F(x))}{(\dot{\lambda} x \dot{\in} A.f) \dot{\in} (\dot{\Pi} x \dot{\in} A.F)}[\Pi\text{-I}]$$

where $\dot{\lambda} x \dot{\in} A.f$ stands for $\dot{\lambda}(A, x \mapsto f(x))$ and $\dot{\Pi} x \dot{\in} A.B$ for $\dot{\Pi}(A, x \mapsto B(x))$.

X is the type of values, that can be seen as a kind of set-theory since we assume we have relations $\dot{\in}$ and $\doteq$ (equality and membership). $\bigstar$ is the set of all the values. Other symbols and related properties are specific to CC, hence look similar to their counterpart in CC.

The second abstract model is an supplement of the first one when upgrading a consistency model to a SN model. The key parameter is the function R that takes a type and one of its elements, and returns the saturated set formed by the realizers of this element.

▶ **Definition 2** (Abstract supplement of SN model)**.** This model aims at building a saturated set for each type and indicates that each proposition is inhabited.

$$✠ : \mathsf{X} \qquad ✠ \in \dot{\Pi}(P \dot{\in} \bigstar).P \qquad \mathsf{R} : \mathsf{X} \to \mathsf{X} \to \mathsf{SAT}$$

$$\mathsf{R}(\dot{\Pi} x \dot{\in} A.B, f) = \bigcap_{x \dot{\in} A} \mathsf{R}(A, x) \overset{sat}{\to} \mathsf{R}(B(x), f \dot{@} x) \qquad \mathsf{R}(\bigstar, P) = \mathsf{SN}$$

where SAT and SN are the sets of all saturated sets and all SN pure λ-terms respectively. $\bigcap$ and $\overset{sat}{\to}$ are standard set intersection and product on saturated sets.

The instance ✠ (the *daimon*) ensures that any type is inhabited including *False*, such that SN is guaranteed in arbitrary type context. Consequently, consistency can not be proved as in the consistency model, but as a matter of fact, it still can be deduced as will be shown in Section 5.

3.2 Main Model Construction

The main model is called M, consisting in giving an account of all syntactic entities (terms, judgments, derivations) based on an instance of the signatures above. The denotations of open terms depend on the valuations of free variables, thus a term is encoded as a pair of functions each taking a valuation ($\mathsf{N} \to \mathsf{X}$ for set-denotation, or $\mathsf{N} \to \Lambda$ for term-denotation) as a parameter returning a set or a pure λ-term as denotation, with some requirements to ensure consistency. Since de Bruijn indices are used, the arguments of valuations are natural numbers. Λ is the type of pure λ-terms.

Due to space constraints, we do not expose the full details of how sorts are dealt with. In the following, we will only consider the sort of propositions. See the formal development or [2] for an exact account. And the symbols of M will be overloaded with a $\tilde{}$.

▶ **Definition 3** (Pseudo-terms)**.** A *term* is a pair of a set and term denotation:

$$\mathsf{Term} \triangleq \{f : (\mathsf{N} \to \mathsf{X}) \to \mathsf{X}\} \times \{g : (\mathsf{N} \to \Lambda) \to \Lambda \mid \mathsf{sub}(g) \wedge \mathsf{lift}(g)\}$$

where $\mathsf{sub}(g)$ and $\mathsf{lift}(g)$ assert that g commutes with substitution and relocation.

We use $\mathsf{Val}(t)_i \triangleq f(i)$ and $\mathsf{Tm}(t)_j \triangleq g(j)$ to denote the set-denotation and term-denotation of term t with certain valuations i and j.

▶ **Definition 4** (Explicit substitution). An *explicit substitution* is a pair:

$$\mathsf{Esub} \triangleq \{f : (\mathsf{N} \to \mathsf{X}) \to (\mathsf{N} \to \mathsf{X})\} \times \{g : (\mathsf{N} \to \Lambda) \to (\mathsf{N} \to \Lambda) | \mathsf{El}(g), \mathsf{Es}(g)\}$$

where $\mathsf{El}(g)$ and $\mathsf{Es}(g)$ are commutation properties similar to the previous definition.

We use $\sigma(i)$ and $\sigma(j)$ to represent $f(i)$ and $g(j)$ for an explicit substitution σ.

▶ **Definition 5** (Term constructors). *Constructors* in M are encoded as:

$$\begin{array}{rcll}
\mathsf{Prop} & \triangleq & \langle i \mapsto \bigstar, & j \mapsto \textsc{k}\rangle \\
\tilde{n} & \triangleq & \langle (i \mapsto i(n)), & j \mapsto j(n)\rangle \\
M \tilde{@} N & \triangleq & \langle i \mapsto \mathsf{Val}(m)_i \dot{@} \mathsf{Val}(N)_i, & j \mapsto \mathsf{Tm}(m)_j \check{@} \mathsf{Tm}(N)_j\rangle \\
\tilde{\lambda} A.t & \triangleq & \langle i \mapsto \dot{\lambda} x \dot{\in} \mathsf{Val}(A)_i.\mathsf{Val}(t)_{i'}, & j \mapsto \textsc{k} \check{@} (\check{\lambda} x.\mathsf{Tm}(t)_{j'}) \check{@} \mathsf{Tm}(A)_j\rangle \\
\tilde{\Pi} A.B & \triangleq & \langle i \mapsto \dot{\Pi} x \dot{\in} \mathsf{Val}(A)_i.\mathsf{Val}(B)_{i'}, & j \mapsto \textsc{k} \check{@} \mathsf{Tm}(A)_j \check{@} \check{\lambda} x.\mathsf{Tm}(B)_{j'}\rangle
\end{array}$$

where $\textsc{k}$ is the combinator $\check{\lambda} x.\check{\lambda} y.x$, i' is the function such that $i'(0) = x$ and $i'(n+1) = i(n)$, and j' is defined as $j'(0) = j(0)$ and $j'(n+1) = \check{\uparrow}^1 j(n)$.

Note that we just show the pair omitting the trivial proof of the assertions. The $\textsc{k}$ combinator used in the second component of $\tilde{\lambda}$ and $\tilde{\Pi}$ allows to simulate CC-reductions occurring in types. Many properties such as $\mathsf{Val}(M \tilde{@} N)_i = \mathsf{Val}(M)_i \dot{@} \mathsf{Val}(N)_i$ follow straightforwardly.

Similarly, we define several instances of explicit substitution, identity and cons:

$$\mathsf{id} \triangleq \langle i \mapsto i, j \mapsto j\rangle \qquad \sigma \cdot M \triangleq \langle i \mapsto \mathsf{Val}(M)_i \odot \sigma(i), j \mapsto \mathsf{Tm}(M)_j \odot \sigma(j)\rangle$$

where $i \mapsto x \odot \sigma \triangleq [0 \mapsto x, \ldots, n+1 \mapsto \sigma(n)](n \geq 0)$.

An *explicit substitution* σ applied to a term M is defined as:

$$M[\sigma] \triangleq \langle i \mapsto \mathsf{Val}(M)_{\sigma(i)}, j \mapsto \mathsf{Tm}(M)_{\sigma(j)}\rangle.$$

▶ **Definition 6** (Judgment). A *type judgment*, denoted by $M : T$, holds for valuations i j iff:

$$[M : T]_{i,j} \triangleq \mathsf{Tm}(M)_j \Vdash_{\mathsf{Val}(T)_i} \mathsf{Val}(M)_i$$

where $t \Vdash_T x$ stands for $x \dot{\in} T \wedge t \check{\in} \mathsf{R}(T, x)$, which reads as "$t$ realizes x of type T".

▶ **Definition 7** (Semantics of context). The denotation of a M-context Γ is a set of pairs of valuations defined as :

$$[\Gamma] = \{(i,j) \mid \forall n.[n : \tilde{\uparrow}^{n+1} \Gamma(n)]_{i,j}\}$$

We will write membership of (i,j) to $[\Gamma]$ as $(i,j) \tilde{\in} [\Gamma]$.

▶ **Definition 8** (Judgments with context). There are four kinds of judgments:

$$\begin{array}{rcll}
\tilde{\vdash} \Gamma & \triangleq & \exists (i,j).(i,j) \tilde{\in} [\Gamma] & \text{(Well-founded Judgment)} \\
\Gamma \tilde{\vdash} M \tilde{\in} T & \triangleq & \forall (i,j) \tilde{\in} [\Gamma].[M : T]_{i,j} & \text{(Typing Judgment)} \\
\Gamma \tilde{\vdash} M \doteq N & \triangleq & \forall (i,j) \tilde{\in} [\Gamma].\mathsf{Val}(M)_i \doteq \mathsf{Val}(N)_i & \text{(Equality Judgment)} \\
\Gamma_1 \tilde{\vdash} \sigma \triangleright \Gamma_2 & \triangleq & \forall (i,j) \tilde{\in} [\Gamma_1], (\sigma(i), \sigma(j)) \tilde{\in} [\Gamma_2] & \text{(Explicit Substitution Judgment)}
\end{array}$$

To express strong normalization, we need to express the notion of reduction between pseudo-terms. This can be based on the set-denotation since the $[\beta]$ parameter of Def. 1 assigns the same set to β-equivalent pseudo-terms. A pseudo-term reduces to another if the term-denotation of the former reduces to that of the latter, whatever the valuation:

▶ **Definition 9.** The *pseudo-reduction* (one step or more) in M is defined as:

$$M \tilde{\to} M' \triangleq \forall j.\mathsf{Tm}(M)_j \check{\to}^+ \mathsf{Tm}(M')_j$$

A pseudo-term is said strongly normalizing if there is no infinite chain of pseudo-reduction starting from it. The abstract SN lemma can be proved now:

▶ **Lemma 10** (Abstract SN). *For any well-typed term t in a valid context Γ, t is SN.*

$$\tilde{\vdash} \Gamma \ \wedge \ \Gamma \tilde{\vdash} t \tilde{\in} T \ \Rightarrow \ SN(t)$$

Proof. By $\tilde{\vdash} \Gamma$, we have $\exists (p,q) \tilde{\in} [\Gamma]$. By $\Gamma \tilde{\vdash} t \tilde{\in} T$, we have $\mathsf{Tm}(t)_j \check{\in} \mathsf{R}(\mathsf{Val}(T)_i, \mathsf{Val}(t)_i)$ for all $(i,j) \tilde{\in} [\Gamma]$. Since $\mathsf{R}(\mathsf{Val}(T)_i, \mathsf{Val}(t)_i)$ is a saturated set, we have $\mathsf{Tm}(t)_q$ is SN by applying (p,q). Any reduction from t can be simulated by a reduction from $\mathsf{Tm}(t)_q$, hence t is SN as a consequence of $\mathsf{Tm}(t)_q$ is SN. ◀

3.3 Soundness of the Main Model

There are four steps to investigate the meta-theories of CC using the model above. The first is to prove the existence of a model, that is, there are instances of the abstract models.

We can show that the abstract models can be instantiated in intuitionistic Zermelo-Frankel set theory with replacement (IZF_R). Types are encoded as a couples formed of a set of values and a function from this set towards saturated set.

All propositions have the same set-denotation $\{\emptyset\}$, and associate a saturated set to $\emptyset$. So the type of all propositions is defined as

$$\bigstar \triangleq (\{(\emptyset, \_ \mapsto S) | S \check{\in} \mathsf{SAT}\}, \_ \mapsto \mathsf{SN}).$$

Dependent product is based on an alternative encoding of functions due to Aczel (see [2] for details), in order to interpret the impredicativity of Prop. The term-denotation of products is fixed by the abstract model. Finally, the daimon ✠ has to be taken as $\emptyset$.

The second step is to prove that the model interprets CC correctly. Syntax maps to semantic terms in the model straightforwardly.

▶ **Theorem 11** (Soundness). *If $\Gamma \vdash t : T$ (in CC, see Fig. 1), then $\tilde{\Gamma} \tilde{\vdash} \tilde{t} \tilde{\in} \tilde{T}$ (in M).*

The third step is the prove the consistency and SN in M. Consistency can now be proved independently from strong normalization:

▶ **Theorem 12** (Consistency). *The model M is consistent.*

$$\forall M, \neg([\ ] \tilde{\vdash} M \tilde{\in} (\tilde{\Pi} P : \mathsf{Prop}.P))$$

Proof. Assume there is closed proof t of $False \triangleq \tilde{\Pi}\mathsf{Prop}.\tilde{0}$ in the model. Then, the term interpretation of the proof t should be closed by commutation with substitution. By the properties of $\tilde{\Pi}$, $\mathsf{Tm}(t)_j \check{@} u$, where u and j are respectively a closed term and a closed valuation, should be in all saturated sets. As a consequence, it must contain free variables which must be in $\mathsf{Tm}(t)_j$ since u is closed. Since both t and j are closed, $\mathsf{Tm}(t)_j$ should be closed, a contradiction. ◀

▶ **Theorem 13** (Strong normalization). *Well typed M-terms are SN.*

Finally, by soundness result, consistency and SN of CC is an immediate consequence of the consistency and SN of M.

4 A Sound Model of Abstract First Order Theory

Compared with sticking to some specific first-order theory, we are more interested in investigating an abstract model ($\mathsf{M_T}$) such that the abstract meta-theoretical results established in that model can apply to any of its instance.

There are three principles to design such an abstract model. Firstly, this model should capture as many theories as possible which leads to an *abstract expression* of its signature and axioms. Secondly, this model should provide enough evidence to ensure only the valid theories are included : including in such a theory would not break the meta-theories established in M. The evidence can be taken as abstract property about the abstract expressions, which we call *assumption*. At last, we want to carry out the tough work as much as possible, such that it is not so hard to instantiate this model later for any allowable specific theory. There should be as few assumptions as possible and each assumption should be the familiar concept. Many decidable theories are first order, hence we restrict us to first-order theories.

To prove the soundness, we also need to abstractly formalize the syntax of the theory as well as the interpretation rules following the abstraction schema of $\mathsf{M_T}$. There are two kinds of abstraction: abstract expressions and assumptions. When instantiating, all abstract expressions and assumptions should be specified and proved respectively.

Following Coq's convention, environment **Parameter** and **Axiom** are used for abstract expression and assumption respectively. Environment **Definition** and **Lemma** are only used for concrete definition and property. Since $\mathsf{M_T}$ is an extension of M, the symbols in $\mathsf{M_T}$ are also reloaded by $\tilde{}$. Syntactic symbols are overloaded by $\hat{}$ for formulas and $\bar{}$ for terms.

4.1 Syntax of the Abstract Theory

Classically, first-order theory consists in four parts: the signature, the formulas, the axioms and the inference rules.

▶ **Parameter 14** (Signature). The domain and the operations of signature are:

$$\mathsf{T} : \mathsf{Set}; \quad \bar{\uparrow} : \mathsf{T} \to \mathsf{N} \to \mathsf{N} \to \mathsf{T}; \quad \bar{\Theta} : \mathsf{T} \to \mathsf{T} \to \mathsf{N} \to \mathsf{T}; \quad \bar{\Phi} : \mathsf{T} \to \mathsf{N} \to [..\mathsf{N}..]$$

where Set is the Coq's primitive type, $[..\mathsf{N}..]$ is the list of natural numbers. $\bar{\Theta}^u_k t$ is the substitution operation on t, $\bar{\Phi}^k t$ the function collecting the free variables in t, and $\bar{\uparrow}^n_k t$ the operation relocating the free variables in t. The latter three operations are indexed by some number k corresponding to the depth at which the operator is applied.

▶ **Definition 15** (Formulas). Formulas are defined inductively upon signature:

$$\mathsf{F} ::= \hat{\top} \mid \hat{\bot} \mid f \hat{\to} g \mid f \hat{\vee} g \mid f \hat{\wedge} g \mid t \sim_{\mathcal{T}} u \mid \hat{\neg} f \mid \hat{\forall} f \mid \hat{\exists} f$$

where f and g are formulas, and t and u are first-order terms.

Note that equality is the only predicate we are interested in. We shall (again) use $\hat{\Phi}^k f$, $\hat{\uparrow}^n_k f$ and $\hat{\Theta}^N_k f$ to denote respectively the set of free variables, the relocation of free variables and the substitution operating on formulas.

▶ **Definition 16** (Context). The *context* (notated as C) is a list of declarations corresponding to either a term variable or an assumption:

$$\mathsf{C} \triangleq [..(\mathsf{T} + \{f : \mathsf{F}\})..].$$

Since theories considered here are first-order, a well-formed formula should contain term variables only.

▶ **Definition 17** (Well-formed term and formula)**.** Given a context H,

$$\mathsf{Wf_t}(H,t) \triangleq \forall n \in_l \bar{\Phi} t, H(n) = \mathsf{T}; \quad \mathsf{Wf_f}(H,g) \triangleq \forall n \in_l \hat{\Phi} g, H(n) = \mathsf{T}$$

A valid formula should be a well-formed formula justified from the axioms by a succession of judgments. We define first the (abstract) notion of axiom, which should be well-formed formulas. Inference rules for generating theorems from the axioms come next. They are defined here in terms of introduction rules, elimination rules and judgmental rules.

▶ **Parameter 18** (Axioms)**.** Axioms are Coq's predicates which judges special formulas in a context, further they should be well-formed :

$$\widehat{\mathsf{Ax}} : \mathsf{C} \to \mathsf{F} \to \mathit{Prop}; \quad \widehat{\mathsf{Ax}}(H,f) \Rightarrow \mathsf{Wf_f}(H,f)$$

▶ **Definition 19** (Derivation rules)**.** A derivation rule ($\hat{\vdash}$: $\mathsf{C} \to \mathsf{F} \to$ Prop) is a Coq's predicate with arity two defined inductively as followings:

$$\frac{H(n) = f, \quad \mathsf{Wf_f}(H, \hat{\uparrow}^{(S\ n)} f)}{H \mathrel{\hat{\vdash}} \hat{\uparrow}^{(S\ n)} f} \quad \frac{\widehat{\mathsf{Ax}}(H,f)}{H \mathrel{\hat{\vdash}} f} \quad \frac{}{H \mathrel{\hat{\vdash}} \hat{\top}} \quad \frac{H \mathrel{\hat{\vdash}} \bot, \quad \mathsf{Wf_f}(H,f)}{H \mathrel{\hat{\vdash}} f}$$

$$\frac{H \mathrel{\hat{\vdash}} f \hat{\to} \hat{\bot}}{H \mathrel{\hat{\vdash}} \hat{\neg} f} \quad \frac{H \mathrel{\hat{\vdash}} \hat{\neg} f}{H \mathrel{\hat{\vdash}} f \hat{\to} \hat{\bot}} \quad \frac{H \mathrel{\hat{\vdash}} f_1, \quad H \mathrel{\hat{\vdash}} f_2}{H \mathrel{\hat{\vdash}} f_1 \hat{\wedge} f_2} \quad \frac{H \mathrel{\hat{\vdash}} f_1 \hat{\wedge} f_2}{H \mathrel{\hat{\vdash}} f_1} \quad \frac{H \mathrel{\hat{\vdash}} f_1 \hat{\wedge} f_2}{H \mathrel{\hat{\vdash}} f_2}$$

$$\frac{H \mathrel{\hat{\vdash}} f_1, \mathsf{Wf_f}(H, f_2)}{H \mathrel{\hat{\vdash}} f_1 \hat{\vee} f_2} \quad \frac{H \mathrel{\hat{\vdash}} f_2, \mathsf{Wf_f}(H, f_1)}{H \mathrel{\hat{\vdash}} f_1 \hat{\vee} f_2} \quad \frac{H \mathrel{\hat{\vdash}} f_1 \hat{\vee} f_2, (f_1 :: H) \mathrel{\hat{\vdash}} \hat{\uparrow}^1 f_3, (f_1 :: H) \mathrel{\hat{\vdash}} \hat{\uparrow}^1 f_3}{H \mathrel{\hat{\vdash}} f_3}$$

$$\frac{\mathsf{Wf_f}(H, f_1), (f_1 :: H) \mathrel{\hat{\vdash}} \hat{\uparrow}^1 f_2}{H \mathrel{\hat{\vdash}} f_1 \hat{\to} f_2} \quad \frac{H \mathrel{\hat{\vdash}} f_1 \hat{\to} f_2, H \mathrel{\hat{\vdash}} f_1}{H \mathrel{\hat{\vdash}} f_2} \quad \frac{(\mathsf{T} :: H) \mathrel{\hat{\vdash}} f}{H \mathrel{\hat{\vdash}} \hat{\forall} f} \quad \frac{H \mathrel{\hat{\vdash}} \hat{\forall} f, \mathsf{Wf_t}(H,t)}{H \mathrel{\hat{\vdash}} \hat{\Theta}^t f}$$

$$\frac{H \mathrel{\hat{\vdash}} \hat{\Theta}^t f \quad \mathsf{Wf_t}(H,t)}{H \mathrel{\hat{\vdash}} \hat{\exists} f} \quad \frac{H \mathrel{\hat{\vdash}} \hat{\exists} f \quad (f :: \mathsf{T} :: H) \mathrel{\hat{\vdash}} \hat{\uparrow}^2 g}{H \mathrel{\hat{\vdash}} g}$$

Valid formulas (or theorems) are those formulas that can be derived by application of the above rules. Valid formulas should be well-formed:

▶ **Lemma 20.** *If* $H \mathrel{\hat{\vdash}} f$*, then* $\mathsf{Wf_f}(H,f)$

4.2 The abstract model of the theory

$\mathsf{M_T}$ is another formalization of the theory using the material provided by M. The domain of first-order term is encoded by a constant $\mathcal{S} : \mathsf{Term}$ in $\mathsf{M_T}$. First-order terms also encoded by Term should satisfy the following assumptions:

▶ **Axiom 21.** *The following assumptions aims at ensuring the meta-theory:*

$$\forall \Gamma, \Gamma \mathrel{\tilde{\vdash}} \mathcal{S} \mathrel{\tilde{\in}} \mathsf{Kind}\ [1] \quad \tilde{\uparrow}^n_k \mathcal{S} \mathrel{\tilde{\doteq}} \mathcal{S} \wedge \tilde{\Theta}^N_k \mathcal{S} \mathrel{\tilde{\doteq}} \mathcal{S}\ [2]$$

$$\frac{\mathsf{Val}(x)_i \dot{\in} \mathsf{Val}(\mathcal{S})_i \quad \mathsf{Val}(y)_i \dot{\in} \mathsf{Val}(\mathcal{S})_i}{\mathsf{Val}(x)_i \dot{=} \mathsf{Val}(y)_i \vee \neg \mathsf{Val}(x)_i \dot{=} \mathsf{Val}(y)_i}\ [3]$$

$$\frac{\mathsf{Val}(x)_i \dot{\in} \mathcal{S} \quad \mathsf{Val}(y)_i \dot{\in} \mathcal{S} \quad \neg \mathsf{Val}(x)_i \dot{=} \mathsf{Val}(y)_i}{\exists P, P \dot{\in} (\dot{\Pi} x \dot{\in} \mathcal{S}.\bigstar) \wedge P \dot{@} \mathsf{Val}(x)_i \dot{=} \mathit{true} \wedge P \dot{@} \mathsf{Val}(x)_i \dot{=} \mathit{false}}\ [4]$$

where $\mathit{true} \triangleq \dot{\Pi} P \dot{\in} \bigstar. \dot{\Pi} p \dot{\in} P.P$ *and* $\mathit{false} \triangleq \dot{\Pi} P \dot{\in} \bigstar.P.$

Assumption [1] and [2] assert that $\mathcal{S}$ is a closed object of type Kind. Assumption [3] asserts that equation of set-denotations of the first-order term should be decidable. The last assumption is included to ensure we could define the equality of the theory as Leibniz equality later. The idea is that there exists a predicate that discriminates between different values of $\mathcal{S}$.

We further assume that axioms actually hold in the model M_{T}:

▶ **Axiom 22.** *Axioms are formulas interpreted by provable* M*-terms.*

The encoding of formulas is given by a standard impredicative encoding due to Girard [9].

▶ **Definition 23** (Formulas)**.** The formulas are defined impredicatively:

$$x\tilde{\doteq}y \triangleq \tilde{\Pi}p : (\tilde{\Pi}x : \mathcal{S}.\mathsf{Prop}).\tilde{\Pi}t : (p\tilde{@}x).(p\tilde{@}y) \qquad \tilde{\neg}f \triangleq f\tilde{\to}\tilde{\bot} \qquad A\tilde{\to}B \triangleq \tilde{\Pi}x : A.B$$

$$\tilde{\bot} \triangleq \tilde{\Pi}P : \mathsf{Prop}.P \qquad A\tilde{\vee}B \triangleq \tilde{\Pi}P : \mathsf{Prop}.((A\tilde{\to}P)\tilde{\to}(B\tilde{\to}P)\tilde{\to}P)$$

$$\tilde{\top} \triangleq \tilde{\Pi}P : \mathsf{Prop}.P\tilde{\to}P \qquad A\tilde{\wedge}B \triangleq \tilde{\Pi}P : \mathsf{Prop}.((A\tilde{\to}B\tilde{\to}P)\tilde{\to}P)$$

$$\tilde{\forall}f \triangleq \tilde{\Pi}x : \mathcal{S}.f \qquad \tilde{\exists}f \triangleq \tilde{\Pi}P : \mathsf{Prop}.((\tilde{\Pi}n : \mathcal{S}.f[x/n])\tilde{\to}P)$$

We can prove similar properties to those in Definition 19, but in the format of typing judgments. Among the 26 properties, we only show the following by using Rule [3] and [4] of the previous Assumption 21.

▶ **Lemma 24.** *Equality of theory can be embedded into the typed equality.*

$$\frac{\mathsf{WFCE}(\Gamma) \quad \Gamma\tilde{\vdash} x\tilde{\in}\mathcal{S} \quad \Gamma\tilde{\vdash} y\tilde{\in}\mathcal{S} \quad \Gamma\tilde{\vdash} t\tilde{\in}(x\tilde{\doteq}y)}{\Gamma\tilde{\vdash} x\dot{=}y}\ [\tilde{\doteq}\text{-E}]$$

where $\mathsf{WFCE}(\Gamma)$*, standing for* well-formed closed environment*, is defined as:*

$$\forall(i,j')\tilde{\in}[\Gamma], \exists j, (i,j)\tilde{\in}[\Gamma] \wedge \mathsf{CPT}(j)$$

$\mathsf{CPT}(j)$ *stands for* Closed Pure Term*, means j always returns a closed term to any index.*

WFCE looks strange but necessary. In M, the value of any proposition is the singleton of empty, the distinction between true propositions and false propositions reflects in their realizers. False proposition contains open realizers only, while true proposition contains at least one close realizer. A well-formed context Γ can contain false propositions and of course false proposition $(x\tilde{\doteq}y)$ is derivable from Γ indicating that x and y may have different values. That's why we need a constraint WFCE on the context showing that the term-valuation of each variable should be the closed to ensure that Γ does not contain false propositions.

4.3 Interpretation Rules and Soundness

Soundness of M_{T} can be proved by defining abstract interpretation rules.

▶ **Parameter 25** (Signature mapping)**.** There exists a function $\mathsf{I}_{\mathsf{T}} : \mathsf{T} \to \mathsf{Term}$ which assigns a M-term for each first-order term.

▶ **Definition 26** (Formula mapping)**.** The function $\mathsf{I}_{\mathsf{F}} : \mathsf{F} \to \mathsf{Term}$, which assigns a M-term to each formula, is defined as:

$$\mathsf{I}_{\mathsf{F}}(x \sim_{\mathcal{T}} y) \triangleq \mathsf{I}_{\mathsf{T}}(x)\tilde{\doteq}\mathsf{I}_{\mathsf{T}}(y) \qquad \mathsf{I}_{\mathsf{F}}(\hat{\bot}) \triangleq \tilde{\bot} \qquad \mathsf{I}_{\mathsf{F}}(\hat{\top}) \triangleq \tilde{\top}$$

$$\mathsf{I}_{\mathsf{F}}(\hat{\neg}f) \triangleq \tilde{\neg}\mathsf{I}_{\mathsf{F}}(f) \qquad \mathsf{I}_{\mathsf{F}}(f\hat{\wedge}g) \triangleq \mathsf{I}_{\mathsf{F}}(f)\tilde{\wedge}\mathsf{I}_{\mathsf{F}}(g) \qquad \mathsf{I}_{\mathsf{F}}(f\hat{\vee}g) \triangleq \mathsf{I}_{\mathsf{F}}(f)\tilde{\vee}\mathsf{I}_{\mathsf{F}}(g)$$

$$\mathsf{I}_{\mathsf{F}}(f\hat{\to}g) \triangleq \mathsf{I}_{\mathsf{F}}(f)\tilde{\to}\mathsf{I}_{\mathsf{F}}(g) \qquad \mathsf{I}_{\mathsf{F}}(\hat{\forall}f) \triangleq \tilde{\forall}\mathsf{I}_{\mathsf{F}}(f) \qquad \mathsf{I}_{\mathsf{F}}(\hat{\exists}f) \triangleq \tilde{\exists}\mathsf{I}_{\mathsf{F}}(f)$$

The interpretation of the context is a combination of I_T and I_F:

▶ **Definition 27.** The function $\mathsf{I}_\Gamma : \mathsf{C} \to [..\mathsf{Term}..]$ is defined as:

$$\mathsf{I}_\Gamma(e) \triangleq \begin{cases} [\,] & (e = [\,]) \\ f(x) :: \mathsf{I}_\Gamma(l) & (e = x :: l) \end{cases} \text{ and } f(x) = \begin{cases} \mathsf{I}_\Gamma(x) & ((x = f) : \mathsf{F})) \\ \mathcal{S} & (x = \mathsf{T}) \end{cases}$$

We now need to assume or prove that each one of these semantic mappings produces expressions of the expected type ($\mathcal{S}$ for terms and Prop for formulas). Omitting the trivial case of contexts, this yields:

▶ **Axiom 28.** $\mathsf{Wf}_\mathsf{t}(\Gamma, t) \Rightarrow \mathsf{I}_\Gamma(\Gamma)\,\tilde{\vdash}\,\mathsf{I}_\mathsf{T}(t)\tilde{\in}\mathcal{S}$

▶ **Lemma 29.** $\mathsf{Wf}_\mathsf{f}(\Gamma, f) \Rightarrow \mathsf{I}_\Gamma(\Gamma)\,\tilde{\vdash}\,\mathsf{I}_\mathsf{F}(f)\tilde{\in}\mathsf{Prop}$

All axioms are formulas provable in the model.

▶ **Axiom 30.** $\widehat{\mathsf{Ax}}(\Gamma, f) \Rightarrow \exists t, \mathsf{I}_\Gamma(\Gamma)\,\tilde{\vdash}\,t\tilde{\in}\mathsf{I}_\mathsf{F}(f)$

Soundness of M_T is not a trivial result:

▶ **Theorem 31** (Soundness). M_T *is sound, that is any derivable formula can be proved in* M_T *by induction on the syntactic derivation rules (Definition 19):*

$$\Gamma\,\hat{\vdash}\,P \Rightarrow \exists p, \mathsf{I}_\Gamma(\Gamma)\,\tilde{\vdash}\,p\tilde{\in}\mathsf{I}_\mathsf{F}(P)$$

5 Soundness of CCT and CICUT

Having a sound model of the theory, the work remaining to ensure the meta-theory of CCT is to prove the soundness of the conversion rule extended with first-order equations.

The theorem is introduced step by step. When the original conversion checking fails to check $\Gamma \vdash A \simeq B$, a decision procedure $\mathsf{Preprocess}$ is called to check whether the theory can be used to check this equation. If the theory is applicable, $\mathsf{Preprocess}$ will refine the context and relocate the variables and the theory will check whether A' and B' are equal in the context Γ'. If any of the above steps fails, then it makes no difference to Coq. Hence, we can start the proof from the conditions that the equality is derivable in the theory:

$$\mathsf{Preprocess}(\Gamma, A, B) = (\Gamma', A', B')[1] \quad \text{and} \quad \Gamma'\,\hat{\vdash}\,A' \sim_\mathcal{T} B'[2]$$

The $\mathsf{Preprocess}$ function should maintain some invariants:

$$\mathsf{I}_\Gamma(\Gamma)\,\tilde{\vdash}\,\sigma \triangleright \mathsf{I}_\Gamma(\Gamma')\,[3]; \quad \mathsf{I}_\mathsf{T}(A) = (\mathsf{I}_\mathsf{T}(A'))[\sigma]\,[4]; \quad \mathsf{I}_\mathsf{T}(B) = (\mathsf{I}_\mathsf{T}(B'))[\sigma]\,[5]$$

By Lemma 20, any derivable formula should be a well-formed formula, particularly for equation in condition [2], its sub-term A' and B' should be well-formed in Γ'. Further, by Axiom 28, they should be interpreted in the scope $\mathcal{S}$:

$$\mathsf{I}_\Gamma(\Gamma')\,\tilde{\vdash}\,\mathsf{I}_\mathsf{T}(A')\tilde{\in}\mathcal{S} \quad [6] \qquad \mathsf{I}_\Gamma(\Gamma')\,\tilde{\vdash}\,\mathsf{I}_\mathsf{T}(B')\tilde{\in}\mathcal{S} \quad [7]$$

By the soundness theorem (Theorem 31) the interpretation of this equality should be inhabited in M_T (condition [8]). If Γ' does not contain false formula, then we have $\mathsf{WFCE}(\mathsf{I}_\Gamma(\Gamma'))$, hence the equality judgment can correctly interpret the equality in theory by Lemma 24 using conditions [6], [7] and [8]:

$$\exists t, \mathsf{I}_\Gamma(\Gamma')\,\tilde{\vdash}\,t\tilde{\in}(\mathsf{I}_\mathsf{T}(A')\tilde{\doteq}\mathsf{I}_\mathsf{T}(B'))\,[8] \qquad \mathsf{I}_\Gamma(\Gamma')\,\tilde{\vdash}\,\mathsf{I}_\mathsf{T}(A')\doteq\mathsf{I}_\mathsf{T}(B')\,[9]$$

Finally, by the substitution lemma applied to conditions [2], [3], [4] and [5], we derive:

$$\mathsf{I}_\Gamma(\Gamma)\,\tilde{\vdash}\,\mathsf{I}_\mathsf{T}(A)\doteq\mathsf{I}_\mathsf{T}(B)$$

Following the above analysis, the new conversion rule is justified by proving:

▶ **Theorem 32.** *Equal terms in theory have the same values in the model:*

$$\frac{\mathsf{WFCE}(\mathsf{I}_\Gamma(\Gamma')) \quad \Gamma\,\tilde{\vdash}\,\sigma \triangleright \mathsf{I}_\Gamma(\Gamma') \quad \Gamma'\,\hat{\vdash}\,A' \sim_{\mathcal{T}} B' \quad x = \mathsf{I}_\mathsf{T}(A')[\sigma] \quad y = \mathsf{I}_\mathsf{T}(B')[\sigma]}{\Gamma\,\tilde{\vdash}\,x\doteq y}$$

Since we require $\mathsf{WFCE}(\mathsf{I}_\Gamma(\Gamma'))$, not all first-order objects (terms and formulas) can be extracted from Γ. That means we may not extract satisfiable equations to Γ' and use these equations to check another equation as SMT solvers do. Nevertheless, it is still an significant improvement of conversion checking.

The sound model of CCT can be built now by adding a sound model for the abstract theory and proving the extended conversion rule. Further, since we no longer incorporate the equality of the theory into ι-reduction, we can absorb the models of inductive types and universes built by Barras to yield a sound model for CICUT. All the meta-theoretical properties that we have established for CC can be established for CCT and CICUT as well.

6 Example: Presburger Arithmetic

In this section, we take Presburger Arithmetic as an example to illustrate how to instantiate the above abstract setting for a specific theory.

Firstly, we define the signature and axioms of Presburger, and prove it correctly instantiates the abstract syntax of theory in Subsection 4.1.

6.1 Formalization of Presburger Arithmetic

The definition of the signature and axioms instantiate Parameter 14 and 18.

▶ **Definition 33** (Signature). Presburger signature is defined inductively:

$$\mathsf{T} \triangleq \bar{n}|\mathsf{C0}|\mathsf{C1}|(x:\mathsf{T})\bar{+}(y:\mathsf{T})$$

where $\bar{n}$ indicates the free variables, $\mathsf{C0}$ and $\mathsf{C1}$ are two constants representing zero and one respectively, $\bar{+}$ is the addition relation.

With this definition, variable relocation, substitution and free variable operations can be defined by recursion on the structure of T.

▶ **Definition 34** (Axioms). An axiom is a special formula presented by a predicate in Coq taking a context and formula as arguments and checking whether this formula is an axiom:

$$\widehat{\mathsf{Ax}}(H,f) \triangleq \begin{cases} f = \hat{\forall} x.\hat{\neg}(\mathsf{C0} \sim_{\mathcal{T}} (x\bar{+}\mathsf{C1})) & \vee \\ f = \hat{\forall} xy.((x\bar{+}\mathsf{C1} \sim_{\mathcal{T}} y\bar{+}\mathsf{C1})\hat{\to}(x \sim_{\mathcal{T}} y)) & \vee \\ f = \hat{\forall} x.x \sim_{\mathcal{T}} (x\bar{+}\mathsf{C0}) & \vee \\ f = \hat{\forall} xy.((x\bar{+}y)\bar{+}\mathsf{C1}) \sim_{\mathcal{T}} (x\bar{+}(y\bar{+}\mathsf{C1})) & \vee \\ \exists g, \mathsf{Wf_f}(\mathsf{T} :: H, g) \wedge (f = g[\mathsf{C0}]\hat{\to}(\hat{\forall} n.g[n]\hat{\to}g[n\bar{+}\mathsf{C1}])\hat{\to}(\hat{\forall} n.g[n])) & \end{cases}$$

The assumption in Parameter 18 can be proved, because g is well-typed in $(\mathsf{T} :: H)$.

Secondly, we give a brief introduction to the formalization of natural number in [2]: useful definition (maybe abstract) and properties without proof. Then the M_T will be instantiated by the these definitions and properties. The abstract interpretation rules are instantiated accordingly which will not be detailed here.

6.2 Formalization of Natural Numbers

As mentioned in Section 3, the first step is to define the set-values of natural numbers (NAT), its constructors (ZERO and SUCC) and eliminator (NREC) in set-theory. Proving that this setting interprets all of the axioms of Presburger arithmetic is straightforward.

The second step is to define the realizers (saturated set) of the type and its constructors. More details can be found in [2].

▶ **Definition 35** (Realizer of constructors)**.** The realizers of natural number are:

$$\mathsf{ZE} \triangleq \check{\lambda} x.\check{\lambda} f.x \qquad \mathsf{SU} \triangleq \check{\lambda} n.\check{\lambda} x.\check{\lambda} f.f \check{@} n \check{@} (n \check{@} x \check{@} f)$$

We then define the saturated set $\mathsf{cNAT}(k)$, the realizers of number k. The idea is to follow the impredicative definition of natural numbers, but we need the dependent version. So we take the least fixpoint of a function fNAT defined as follows:

▶ **Definition 36** (Saturated set associated to natural numbers)**.**

$$\begin{aligned}
\mathsf{fNAT}(A,k) &\triangleq \bigcap_{P:\mathsf{X}\to\mathsf{SAT}} P(\mathsf{ZERO}) \to \left[\bigcap_{n\in\mathsf{NAT}} A(n) \to P(n) \to P(\mathsf{SUCC}(n))\right] \to P(k) \\
\mathsf{cNAT}(k) &\triangleq \bigcap_{A \text{ s.t. } \mathsf{fNAT}(A)\subseteq A} A
\end{aligned}$$

and these definitions satisfy following properties :

$$\frac{}{\mathsf{ZE} \check{\in} (\mathsf{cNAT}(\mathsf{ZERO}))} \qquad \frac{n \dot{\in} \mathsf{NAT} \quad t \check{\in} \mathsf{cNAT}(n)}{\mathsf{SU} \check{@} t \check{\in} \mathsf{cNAT}(\mathsf{SUCC}(n)))}$$

Then, natural numbers can be defined in M:

▶ **Definition 37** (Natural Number)**.** The type of natural number and its constructors are defined in M as follows:

$$\begin{aligned}
\mathsf{Nat} &\triangleq \langle i \mapsto (\mathsf{NAT}, \mathsf{cNAT}), j \mapsto \mathrm{K} \rangle \\
\mathsf{Zero} &\triangleq \langle i \mapsto \mathsf{ZERO}, j \mapsto \mathsf{ZE} \rangle \\
\mathsf{Succ} &\triangleq \langle i \mapsto \mathsf{SUCC}, j \mapsto \mathsf{SU} \rangle \\
\mathsf{NatRec}(f,g,n) &\triangleq \left\langle \begin{array}{l} i \mapsto \mathsf{NREC}(\mathsf{Val}(f)_i, (n, y \mapsto \mathsf{Val}(g)_i \dot{@} n \dot{@} y), \mathsf{Val}(n)_i), \\ j \mapsto \mathsf{Tm}(n)_j \check{@} \mathsf{Tm}(f)_j \check{@} \mathsf{Tm}(g)_j \end{array} \right\rangle
\end{aligned}$$

▶ **Lemma 38.** *Typing rules of the natural numbers can be proved:*

$$\Gamma \tilde{\vdash} \mathsf{Zero} \tilde{\in} \mathsf{Nat} \quad \Gamma \tilde{\vdash} \mathsf{Succ} \tilde{\in} (\tilde{\Pi} n \tilde{\in} \mathsf{Nat}.\mathsf{Nat}) \quad \Gamma \tilde{\vdash} \mathsf{Nat} \tilde{\in} \mathsf{Kind}$$

$$\frac{\Gamma \tilde{\vdash} n \tilde{\in} \mathsf{Nat} \quad \Gamma \tilde{\vdash} f \tilde{\in} (P \tilde{@} \mathsf{Zero}) \quad \Gamma \tilde{\vdash} g \tilde{\in} (\tilde{\Pi} n \tilde{\in} \mathsf{Nat}.\tilde{\Pi}(P \tilde{@} n).P \tilde{@} (\mathsf{Succ} \tilde{@} n))}{\Gamma \tilde{\vdash} \mathsf{NatRec}(f,g,n) \tilde{\in} (P \tilde{@} n)}$$

6.3 Instantiation the Model of Abstract Theory by Presburger

The $\mathcal{S}$ in M_T is instantiated by Nat. Furthermore, we need to provide enough constructors to interpret Presburger Signature (Definition 33).

▶ **Definition 39** (Presburger Semantic)**.** The semantic of Presburger is:

$$\mathcal{S} \triangleq \mathsf{Nat} \quad \mathsf{Zero} \quad \mathsf{Succ}(\mathsf{Zero}) \quad \tilde{+} \triangleq \tilde{\lambda}(x \tilde{\in} \mathsf{Nat}).\tilde{\lambda}(y \tilde{\in} \mathsf{Nat}).\mathsf{NatRec}(x, \tilde{\lambda}(z \tilde{\in} \mathsf{Nat}).\mathsf{Succ}, y)$$

By the induction scheme provided in the values of natural numbers, we can prove the last two assumptions in Axiom 21, while others are trivial according to the definition.

Instantiating Presburger axioms in M is just a translation work according to the interpretation rules, but proving them requires more efforts: the main task is to construct the proof term for each axiom. We have all the detailed proofs in our development to ensure:

▶ **Lemma 40.** $\mathsf{M_T}$ *instantiated by Presburger arithmetic is sound.*

Then, all the meta-theoretical properties holding in $\mathsf{M_T}$ preserve after instantiation by Presburger arithmetic, therefore we can conclude:

▶ **Lemma 41.** *Presburger arithmetic is a safe theory to be embedded, hence* CoqMT *is safe.*

Let us finish with an important remark. We have proven the strong normalization for CICUT instantiated with Presburger arithmetic, but the reduction considered includes β- and ι-reduction (the reduction of NatRec when its main argument is either 0 or successor), but not the reduction associated to the last two rules of Section 2. The main difficulty is that the latter is not sequential (both $\mathsf{S}(n) + m$ and $n + \mathsf{S}(m)$ reduce to a successor), while the β-reduction of λ-calculus is sequential. We thus cannot claim yet the decidability of type-checking for the presented version of CICUT.

7 Conclusion

In this paper, we give an abstract proof of consistency and SN to CCT, which extends CC family by incorporating extensional equalities from an abstract theory. Presburger arithmetic is proved to correctly instantiate the abstract theory which, on the one hand, ensures Presburger arithmetic is safe to be embedded, on the other hand demonstrates that our abstraction strategy works well for adding a new theory of interest.

Actually, this proof is applicable to a richer type system contains more features if each feature has a sound model and it does not have interference with other features, such as CICUT. That's our original motivation to do this research: improving the Coq by incorporating decidable extensional equalities.

We don't embed theory in the ι-reduction to ensure the consistency and SN, the side effect is that Church-Rosser can not be proved in the usual way, as well as the decidability of type checking (DoTC). However we have another way to prove DoTC, that's our next work : formalize the syntax of CCT or CICUT, and build a complete formal proof of all the properties required.

Acknowledgements. The authors would like to thank Jean-Pierre Jouannaud and Pierre-Yves Strub for many useful discussions. The authors would also like to thank Jean-Pierre Jouannaud and anonymous reviewers for their useful comments and language editing which have greatly improved the mansucript.

References

1 Thorsten Altenkirch. Proving strong normalization of cc by modifying realizability semantics. In Henk Barendregt and Tobias Nipkow, editors, *TYPES*, volume 806 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 1993.

2 Bruno Barras. Semantical investigations in intuitionistic set theory and type theories with inductive families. In preparation habilitation thesis, http://www.lix.polytechnique.fr/ barras/habilitation/.

3 Bruno Barras. Sets in coq, coq in sets. *Journal of Formalized Reasoning*, 3(1):29–48, 2010.

4 Bruno Barras, Jean-Pierre Jouannaud, Pierre-Yves Strub, and Qian Wang. Coqmtu: A higher-order type theory with a predicative hierarchy of universes parametrized by a decidable first-order theory. In *LICS*, pages 143–151. IEEE Computer Society, 2011.

5 Frédéric Blanqui. Inductive types in the calculus of algebraic constructions. In Martin Hofmann, editor, *TLCA*, volume 2701 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2003.

6 Frédéric Blanqui, Jean-Pierre Jouannaud, and Pierre-Yves Strub. From formal proofs to mathematical proofs: a safe, incremental way for building in first-order decision procedures. *CoRR*, abs/0804.3762, 2008.

7 Frédéric Blanqui, Jean-Pierre Jouannaud, and Pierre-Yves Strub. From formal proofs to mathematical proofs: A safe, incremental way for building in first-order decision procedures. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *IFIP TCS*, volume 273 of *IFIP*, pages 349–365. Springer, 2008.

8 Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988.

9 J. L. Krivine. *Lambda-calculus, types and models.* Ellis Horwood, Upper Saddle River, NJ, USA, 1993.

10 Zhaohui Luo. Ecc, an extended calculus of constructions. In *LICS*, pages 386–395. IEEE Computer Society, 1989.

11 Nicolas Oury. Extensionality in the calculus of constructions. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2005.

12 Christine Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In Marc Bezem and Jan Friso Groote, editors, *TLCA*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993.

13 Vincent Siles and Hugo Herbelin. Equality is typable in semi-full pure type systems. In *LICS*, pages 21–30. IEEE Computer Society, 2010.

14 Mark-Oliver Stehr. The open calculus of constructions (part i): An equational type theory with dependent types for programming, specification, and interactive theorem proving. *Fundam. Inform.*, 68(1-2):131–174, 2005.

15 Mark-Oliver Stehr. The open calculus of constructions (part ii): An equational type theory with dependent types for programming, specification, and interactive theorem proving. *Fundam. Inform.*, 68(3):249–288, 2005.

16 Pierre-Yves Strub. *Théories des Types et Procédures de Décisions.* These, Ecole Polytechnique X, July 2008.

17 Pierre-Yves Strub. Coq modulo theory. In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2010.

18 William Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, chapter 7, pages 240–251–251. Springer Berlin / Heidelberg, Berlin, Heidelberg, 1975.

19 The Coq Development Team. The Coq Proof Assistant, Reference Manual, Version 8.4. Technical report, INRIA, Roquencourt, France, 2012.