Dependently typed lambda calculus with a lifting operator
Internship report

Damien Rouhling

May - August 2014

Abstract

This report is about my internship in the computer science department of Chalmers University of Technology in Gothenburg, Sweden, for the validation of my first year of masters in computer science. It deals with the definition of a lifting operator for a cumulative hierarchy of universes in a dependent type theory and a proof of (weak) normalization for the resulting calculus. We use normalization by evaluation (NbE) and prove its completeness and soundness respectively with a partial equivalence relation (PER) model and Kripke logical relations.

Contents

Introduction 2

1 Context 2
1.1 Motivations . . . 2
1.2 System . . . 3
1.3 Contributions . . . 5

2 Normalization by evaluation 5
2.1 Principle . . . 5
2.2 In presence of the lifting operator . . . 5

3 Completeness and soundness of NbE 7
3.1 PER model and completeness of NbE . . . 7
3.2 Kripke relations and soundness of NbE . . . 10

4 Implementation 11
4.1 Bidirectional type-checking . . . 11
4.2 Adaptation and integration of the calculus . . . 12

5 Further and related work 13

Conclusion 14

A Rules of the calculus 15

B Evaluation and read-back 17

References 19

Introduction

In the context of my first year of masters' internship I worked on dependent type theory under the supervision of Thierry Coquand in the computer science department of Chalmers in Gothenburg, Sweden.

Types were introduced to avoid Russell's paradox. This paradox implies that there is no set of all sets. One can define a universe type to express inside the system what it means to be a type. It is a type which contains every well-formed (or "small") type, introduced by Per Martin-Löf ([ML72], [ML84]). However, one has to be careful with the terminology: the universe is a type in the sense that it can be used in a typing judgment (I can find a term of type the universe) but it is not its own type. It is inconsistent to have a universe type whose type is itself (see Jean-Yves Girard's paradox [Gir72]). A solution to give a type to the universe is to introduce an infinite hierarchy of universes, each having the next one as type ([Pal98]).

Dependent types are types constructed from type families. They extend the correspondence between lambda calculus and logic, also known as the Curry-Howard correspondence. Dependent types correspond to quantifiers. Dependent function types can be seen as types for functions with a codomain depending on the term to which they are applied. They correspond to the universal quantifier of predicate calculus. For the existential quantifier, there are the dependent pairs: the type of the second element of the pair depends on the first element.

The goal of my internship was to show that it is possible to define a constructor for terms allowing to lift them from one universe to the next in a system with dependent types[1]. The next section gives the ideas behind this constructor and the chosen system. In particular, the calculus had to be proven normalizing.

For this purpose, we used normalization by evaluation, a method which is now well established ([CD97], [AAD07], [Abe13]). We needed to adapt it to our system, the lifting operator introducing some subtleties. The next step was to prove this normalization sound and complete. We also have a prototype implementation of such a system.

1 Context

This work is motivated by the wish for combining cumulativity and universe polymorphism in a proof assistant and results from a proposition of Conor McBride ([McB11]). We first detail his proposition and then explain our choices to achieve the definition of the lifting operator suggested by McBride.

1.1 Motivations

Combining cumulativity and universe polymorphism is an active subject of research. Cumulativity expresses the inclusion of universes in the higher ones, but can be extended by the use of a subtyping rule:

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash T \leq T'}{\Gamma \vdash t : T'}$$

Several versions of subtyping exist but the one we will consider here is the following (see Appendix A, Figure 6 for details): each universe is a subtype of the next one (which gives cumulativity by transitivity) and we have a subtyping rule for function types, contravariant in the domain and covariant in the codomain.

[1]Only dependent functions were considered.

$$\begin{array}{rcl}
\mathit{Name} \ni c & & \\
\mathit{Exp} \ni r, s, t & ::= & c \mid \mathsf{v}_i \mid \lambda.t \mid r\ s \mid \mathsf{U}_n \mid \Pi\ R\ S \mid t^+ \mid t\sigma \\
\mathit{ExpSub} \ni \sigma, \delta & ::= & \uparrow \mid \mathsf{id} \mid \langle\rangle \mid (\sigma, t) \mid \sigma\delta \\
\mathit{Nf} \ni v, w & ::= & u \mid \Pi\ V\ \ (\lambda.W) \mid \lambda.v \mid \mathsf{U}_n \\
\mathit{Ne} \ni u & ::= & c^n \mid \mathsf{v}_i \mid u\ v \\
\mathit{Ctx} \ni \Gamma & ::= & \diamond \mid \Gamma.T \\
\mathit{Def} \ni l & ::= & \diamond \mid l.c : T \mid l.c = t : T
\end{array}$$

Figure 1: Grammar of terms, substitutions, normal forms, contexts and definitions

In his blog post ([McB11]), McBride recalls that Coq ([Coq]) supports cumulativity, but not Adga ([Agd]). On the other side, Agda supports universe polymorphism, and not Coq. Universe polymorphism allows you to declare terms which are usable at different levels. An example is the case of the identity type, which should exist at every level.

The need for universe polymorphism is real: for his research, Vladimir Voevodsky had to duplicate code in an implementation in Coq[2]. He had two versions of the same file with different universe levels, which could be avoided by "a better universe management". Note that on the other branch of his project[3] Voevodsky removes the universe consistency checking by a "type in type patch", which is an even worse solution.

Robert Pollack and Robert Harper were the first to try to deal with universe polymorphism for the Calculus of Constructions ([HP91]). Their system has been refined by Hugo Herbelin ([Her05]). An extension of Coq based on these ideas has recently been implemented by Matthieu Sozeau and Nicolas Tabareau ([ST14]). However, the system is complicated because it deals with constraints solving for universe variables.

The suggestion of McBride is a simpler system in which the user has to specify the levels. His idea is the introduction of an operator $t \mapsto t^n$ which adds n to every level appearing in t. Using rules which are invariant under uniform increment of levels, this permits to emulate universe polymorphism (e.g. if t is derivable of type U_m, then t^n will be derivable of type U_{n+m}).

Let us consider the example of the polymorphic identity function. We have a term id of type $\Pi\ \ (X : \mathsf{U}_0)\ \ (X \to X)$, which means that $\mathit{id}\ X$ is a (non-dependent) function of type $X \to X$. The type of id^1 is then $\Pi\ \ (X : \mathsf{U}_1)\ \ (X \to X)$, which is the type of the polymorphic identity function at level 1. Thus, to get the identity function at level n, it is sufficient to provide the identity function at level 0 and to apply this lifting operator.

1.2 System

During my internship, I have designed a system in which the lifting operator suggested by McBride is a term constructor. Keeping in mind the practical applications, we want to allow the user of a proof assistant to declare a term at some level, and then to give explicitely the level at which he or she wants the term to be lifted each time it is used. We write t^+ for t lifted and we use the following notation: $t^0 := t$ and $t^{n+1} := (t^n)^+$.

The introduction of the lifting operator in the syntax brings some subtleties in the definition of a reduction. The constructor $.^+$ should commute with every other constructor except that U_n^+ reduces to U_{n+1} and that variables are unchanged by this operation, because the context is also lifted (if x is of type T, in the lifted context x will be of type T^+ so that we don't need to lift x).

The most significant point is the case of β-reduction: how do we define $t^+\,[x := u]$?

[2]See https://github.com/vladimirias/Foundations/tree/before_implicit
[3]https://github.com/vladimirias/Foundations/tree/master

We cannot replace directly x by u in t^+ as it is shown by the following example. If we give the type $\mathsf{U}_0 \to \mathsf{U}_0$ to $t := \lambda x.x$, then t^+ is of type $\mathsf{U}_1 \to \mathsf{U}_1$ and $t^+\ \mathsf{U}_0$ should be of type U_1. Since t reduces to $\lambda x.x^+$ (and then also to t!) we might need to compute $x^+\,[x := \mathsf{U}_0]$. By directly replacing x by U_0, the term will reduce to U_1, which is of type U_2, not U_1 (recall Girard's paradox).

To solve this problem, we introduce explicit substitutions ($t\delta$ is a term if t is a term and δ is a substitution). The grammar is given in Figure 1. $\uparrow$ is a weakening substitution. It is introduced because we use De Bruijn's indices for variables ($\mathsf{v}_i \uparrow$ reduces to v_{i+1}). The reductions rules for lifting with substitutions are the following: to reduce $t^+\delta$ one has first to reduce t^+ and $(t\delta)^+$ reduces to $t^+\delta^+$, where δ^+ is defined as follows: $\uparrow^+ := \uparrow$, $\mathsf{id}^+ := \mathsf{id}$, $\langle\rangle^+ := \langle\rangle$, $(\sigma, t)^+ := (\sigma^+, t^+)$ and $\sigma\delta^+ := \sigma^+\delta^+$.

Lifting on contexts is defined the obvious way: $\diamond^+ := \diamond$ and $(\Gamma.T)^+ := \Gamma^+.T^+$. We use the same notations δ^n and Γ^n as for terms.

We also introduce environments of global definitions. In these environments, constants are defined to be of a certain type, and one can give a term to which it should correspond. In this case, the constant plays the role of a name for the term. The definitions being global, only closed terms and types are used in them. Note that if the reduction eliminates the lifting constructor on variables, it cannot be the case for constants.

The judgments of our system are the following:

- l ok $\quad l$ is a well-formed environment of global definitions.
- $\Gamma \vdash^l \quad \Gamma$ is a well-formed context.
- $\Gamma \vdash^l \delta : \Delta \quad \delta$ is a well-formed substitution from context Γ to context Δ.
- $\Gamma \vdash^l t : T \quad t$ is a well-formed term of type T in context Γ.
- $\Gamma \vdash^l T$ iff $\Gamma \vdash^l T : \mathsf{U}_n$ for some n and $\Gamma \vdash^l T = T'$ iff $\Gamma \vdash^l T = T' : \mathsf{U}_n$ for some n.
- $\Gamma \vdash^l R \leq S \quad R$ is a subtype of S in context Γ.
- $\Gamma \vdash^l \sigma = \delta : \Delta \quad \sigma$ and δ are equal substitutions from context Γ to context Δ.
- $\Gamma \vdash^l r = s : T \quad r$ and s are equal terms of type T in context Γ.

The rule for lifted terms is the following:

$$\frac{\Gamma \vdash^l t : T}{\Gamma^+ \vdash^l t^+ : T^+}$$

It corresponds to the idea of using rules which are invariant by uniform increment of levels, mentioned in the previous subsection.

Remark 1 (Link with presheaf models of type theory)**.** *This rule also has a strong link with the current work of Thierry Coquand on presheaf models of type theory.*

He defines a system in which the typing judgment has the form $\Gamma \vdash_I t : T$ where I is an object of a particular category. Then, for any morphism $f : I \to J$ we have the following rule:

$$\frac{\Gamma \vdash_I t : T}{\Gamma f \vdash_J tf : Tf}$$

In our case, the category is the monoid $(\mathbb{N}, +)$. Later on (Section 2), our notion of evaluation will also correspond to the one in presheaf models.

The rules for definitions, contexts, substitutions, typing and subtyping are given in Figure 6. Equality for terms is specified in Figure 7 and for substitutions in Figure 8. We use the following notations:

- $[t] := (\mathsf{id}, t)$.
- $(\Gamma.T)\,(0) := T \uparrow$ and $(\Gamma.T)\,(i+1) := \Gamma\,(i) \uparrow$.

$$
\begin{array}{rcl}
D \ni a, f & ::= & (\lambda.t)\,\rho \mid \uparrow^{A} e \mid \mathsf{U}_n \mid \mathsf{Fun}\ A\ F \\
Base \ni B & ::= & \mathsf{U}_n \mid \uparrow^{\mathsf{U}_n} E \\
Env \ni \rho & ::= & (n, \tau, l) \\
\tau \in Sub & := & \mathbb{N} \to D \\
D^{ne} \ni e & ::= & c^n \mid \mathsf{x}_k \mid e\ d \\
D^{nf} \ni d & ::= & \downarrow^{A} a
\end{array}
$$

Figure 2: Semantic domain, environments and normal values

1.3 Contributions

I have achieved a proof of weak normalization for the system in Appendix A, going through normalization by evaluation. I had first to learn the basics of normalization by evaluation and then to adapt the method to our case[4] (Section 2). It was necessary to prove the normalization by evaluation sound and complete with respect to the propositional equality of our system. Completeness is proven using a PER model (Subsection 3.1). Kripke logical relations were used to show the soundness (Subsection 3.2). I have implemented a bidirectional type checker in Haskell ([Has]) corresponding to this system (Section 4), with the help of Anders Mörtberg. Different extensions were considered (Section 5).

2 Normalization by evaluation

Normalization by evaluation is a well-developed method which has proven to be useful ([ACP11], [FP13], [AC14]). We explain here the principle of this method and the adaptation needed to use it in our framework.

2.1 Principle

Normalization by evaluation is achieved through two steps. First, the considered term will be *evaluated* in an environment. The environment should correspond to the context in which the term is typed because it gives a value to each (free) variable. The evaluation function's codomain is a set of semantics values. A particular case for the semantic domain is a partial applicative structure. It is a set equipped with an application function (see next subsection for an example).

Then, values are *reified* back to the set of terms, resulting in normal forms. Normalization can be typed or untyped ([Abe13]). Typed normalization by evaluation is used to obtain η-long β-normal forms. In this case, reification is led by type annotations. Those annotations can be introduced at the level of values ([Abe13]): we can use *reflection* ($\uparrow^{A} e$) to give the type of a neutral value, and *reification* ($\downarrow^{A} a$) to give the type of a value and get a normal value. A *read-back* function ($\mathsf{R}^{nf}_{\cdot}.$) is then used to send normal values to normal terms. It might call a particular read-back function for neutral values ($\mathsf{R}^{ne}_{\cdot}.$). This idea of having a reification in two phases (first η-expansion, then read-back) has been introduced by Andreas Abel, Thierry Coquand and Miguel Pagano ([ACP11]).

2.2 In presence of the lifting operator

We denote environments by ρ, and the evaluation of a term t in the environment ρ by $[\![t]\!]\,(\rho)$.

[4]Because of the previous remark about substitution.

Recall our discussion about substitution on a lifted term (Subsection 1.2). The term $t^+\delta$ cannot be reduced until t^+ is. We encounter the same difficulty when trying to define $[\![t^+]\!]\,(\rho)$.

Thierry Coquand has suggested introducing level annotations in environments: an environment ρ is then equal to a pair (n,τ) where the integer n is a level annotation and the substitution[5] τ corresponds to the usual structure of environments.

How does evaluation work, then? Most of the rules stay unchanged, we just need to pass the level annotation. However, three rules must draw our attention:

$$\begin{array}{rcl} [\![\mathsf{v}_i]\!]\,(n,\tau) & := & [i]\,(\tau) \\ [\![\mathsf{U}_n]\!]\,(m,\tau) & := & \mathsf{U}_{n+m} \\ [\![t^+]\!]\,(n,\tau) & := & [\![t]\!]\,(n+1,\tau) \end{array}$$

In the case of a variable, we only need to take the corresponding value in the substitution τ (denoted by $[i]\,(\tau)$ here). It should not be lifted in any way (see again our discussion in Subsection 1.2). Universes are shifted, since it is the meaning of this operator. Finally, the evaluation of a lifted term is the evaluation of this term in an environment with an incremented level annotation. As for the typing rules, this last evaluation rule is justified by presheaf models of type theory (recall Remark 1).

For defined constants, we also add the environment of global definitions to the structure of environments. Two cases appear:

$$\begin{array}{rcl} [\![c]\!]\,(n,\tau,l) & := & [\![t]\!]\,(n,\tau,l) \text{ if } c = t : T \in l \\ [\![c]\!]\,(n,\tau,l) & := & \uparrow^{[\![T]\!](n,\tau,l)} c^n \text{ if } c : T \in l \end{array}$$

If the constant is a name for a given term, its evaluation will be the evaluation of the term. If no term is given, it will be considered as a neutral value and we evaluate the type of the constant to build a reflection.

Since values are given only to free variables in the environments, evaluation is blocked on lambda abstractions: we introduce closures $(\lambda.t)\,\rho$ in our semantic domain. We can evaluate under the abstraction when it is applied to a value. That is why we have an applicative structure.

The grammar of our semantic domain is given in Figure 2. We denote by $\mathsf{add}\,(a,\tau)$ the substitution τ' such that:

$$\left\{ \begin{array}{rcl} \tau'\,(0) & \equiv & a \\ \tau'\,(n+1) & \equiv & \tau\,(n) \end{array} \right.$$

The complete set of rules for evaluation, application and read-back is given in Appendix B. Note that any substitution could be the evaluation of the empty explicit substitution: we will never try to evaluate a variable with it. Base types (B) are used later (Subsection 3.2).

To define the normalization by evaluation function, we first need to introduce a canonical substitution associated to a given context.

Definition 1 (Normalization by evaluation)**.** Define the n-lifted canonical substitution by:

$$\uparrow_n^{\diamond,l} := i \mapsto \mathsf{U}_0 \text{ and } \uparrow_n^{\Gamma.T,l} := \mathsf{add}\left(\uparrow^{[\![T]\!](n,\tau,l)} \mathsf{x}_{|\Gamma|}, \tau\right) \text{ where } \tau \equiv \uparrow_n^{\Gamma,l}$$

Then, $\mathsf{NbE}_{\Gamma,l}^{T}\,(n,t) := \mathsf{R}_{|\Gamma|}^{nf} \downarrow^{A} \left([\![t]\!]\left(n,\uparrow_n^{\Gamma,l},l\right)\right)$ where $A \equiv [\![T]\!]\left(n,\uparrow_n^{\Gamma,l},l\right)$.

Write then $\uparrow^{\Gamma,l}$ for $\uparrow_0^{\Gamma,l}$, and $\mathsf{NbE}_{\Gamma,l}^{T}\,(t)$ for $\mathsf{NbE}_{\Gamma,l}^{T}\,(0,t)$.

For types, we will use $\mathsf{NbE}_{\Gamma,l}\,(n,T)$ instead of $\mathsf{NbE}_{\Gamma,l}^{\mathsf{U}_i}\,(n,T)$.

[5]Not to be mixed with explicit substitutions which are defined on terms. In environments, substitutions are defined on values.

3 Completeness and soundness of NbE

Completeness of normalization by evaluation can be proven by a PER model. It is motivated by the idea of realizability predicate. We then show soundness using Kripke logical relations.

3.1 PER model and completeness of NbE

Realizability predicates (or computability predicates) have been introduced and used by William Tait to model second order intuitionistic logic ([Tai75]). Catarina Coquand used them to prove normalization for a dependent type theory ([Coq98]). This notion has also been used by Pierre-Louis Curien ([Cur91]) and Thierry Coquand and Guilhem Jaber ([CJ10]).

The basic idea (in our framework) is to use predicates $Type_i$ and $Val_i\,(T)$. $Type_i\,(T)$ holds for "T is a good type at level i" and if $Type_i\,(T)$ is true, then $Val_i\,(T,t)$ holds for "t is a good term of type T at level i". These predicates can be defined inductive-recursively.

Cumulativity impose that if $Type_i\,(T)$ holds, then it is also the case for all $j > i$. Thus, we can define $Type\,(T)$ as $\exists i, Type_i\,(T)$ and have a similar definition for $Val\,(T,t)$.

One can then show that if $Type\,(T)$ then T is normalizable (and a similar statement for Val). Note that we can justify semantically the lifting operator by showing the following statement: if $Type\,(T)$ then for all n we have also $Type\,(T^n)$ (and similarly for Val).

However, in presence of a rule for η-conversion, predicates are not sufficient ([AC05]) and we need to introduce partial equivalence relations (PER), which are sufficient to prove the consistency of a system ([Abe10]) and further up the completeness of normalization by evaluation ([ACP11]). A partial equivalence relation is a symmetric and transitive binary relation.

Constants aside, there is no occurrence of the lifting constructor in the set of values so that a standard PER model will be sufficient. We first define PERs over neutral and normal semantic values[6]. Two values are related if they will be read back to the same normal form. Then, our goal will be to show that if we can derive two terms to be equal in our system, then the reifications of their evaluations will be related by these PERs.

Definition 2 (PERs over normal semantic values)**.**

- $d = d' \in \top$ iff $\forall n \exists v \in Nf, \mathsf{R}^{nf}_n d \equiv v \equiv \mathsf{R}^{nf}_n d'$.
- $e = e' \in \bot$ iff $\forall n \exists u \in Ne, \mathsf{R}^{ne}_n e \equiv u \equiv \mathsf{R}^{ne}_n e'$.

One can use the introduction rules in Figure 3.

Now we are ready to define the PERs corresponding to our idea of realizability predicates.

Definition 3. We define the following PERs over semantic values using the rules in Figure 4:

- $Type_i \in \mathsf{PER}\,(D)$.
- $Val_i\,(A) \in \mathsf{PER}\,(D)$ for $A \in Type_i$.
- $Type := \bigcup\limits_i Type_i$ and $Val := \bigcup\limits_i Val_i$.

We use the following notations:

- $\underline{E}_n := \left\{ \uparrow^{\uparrow^{\mathsf{U}_n}E_1} e_1 = \uparrow^{\uparrow^{\mathsf{U}_n}E_2} e_2 \mid E_1 = E_2 = E \in \bot \text{ and } e_1 = e_2 \in \bot \right\}$.
- $\Pi\ \mathcal{A}\ \mathcal{F} := \{ f = f' \mid \forall a = a' \in \mathcal{A}, f \cdot a = f' \cdot a' \in \mathcal{F}\,(a) \}$.

$$\frac{}{c^n = c^n \in \bot} \qquad \frac{}{\mathsf{x}_k = \mathsf{x}_k \in \bot} \qquad \frac{e = e' \in \bot \qquad d = d' \in \top}{e\ d = e'\ d' \in \bot} \qquad \frac{e = e' \in \bot}{\downarrow^{B_1}\uparrow^{B_2} e = \downarrow^{B'_1}\uparrow^{B'_2} e' \in \top}$$

$$\frac{}{\downarrow^{\mathsf{U}_k} \mathsf{U}_n = \downarrow^{\mathsf{U}_k} \mathsf{U}_n \in \top} \qquad \frac{\forall e = e' \in \bot, \downarrow^{F \cdot \uparrow^{A} e} \left(f \cdot \uparrow^{A} e\right) = \downarrow^{F' \cdot \uparrow^{A'} e'} \left(f' \cdot \uparrow^{A'} e'\right) \in \top}{\downarrow^{\mathsf{Fun}\ A\ F} f = \downarrow^{\mathsf{Fun}\ A'\ F'} f' \in \top}$$

$$\frac{\downarrow^{\mathsf{U}_i} A = \downarrow^{\mathsf{U}_i} A' \in \top \qquad \forall e = e' \in \bot, \downarrow^{\mathsf{U}_i} \left(F \cdot \uparrow^{A} e\right) = \downarrow^{\mathsf{U}_i} \left(F' \cdot \uparrow^{A'} e'\right) \in \top}{\downarrow^{\mathsf{U}_i} \mathsf{Fun}\ A\ F = \downarrow^{\mathsf{U}_i} \mathsf{Fun}\ A'\ F' \in \top}$$

Figure 3: PERs $\top$ and $\bot$

$$\frac{E = E' \in \bot \qquad j \leqslant i}{\uparrow^{\mathsf{U}_j} E = \uparrow^{\mathsf{U}_j} E' \in Type_i}\ (\textsc{neut}) \qquad \frac{j < i}{\mathsf{U}_j = \mathsf{U}_j \in Type_i}\ (\textsc{univ})$$

$$\frac{A = A' \in Type_i \qquad \forall a = a' \in Val_i(A), F \cdot a = F' \cdot a' \in Type_i}{\mathsf{Fun}\ A\ F = \mathsf{Fun}\ A'\ F' \in Type_i}\ (\textsc{fun})$$

- If $\uparrow^{\mathsf{U}_j} E = \uparrow^{\mathsf{U}_j} E' \in Type_i$ by (NEUT), then $Val_i\left(\uparrow^{\mathsf{U}_j} E\right) := \underline{E}_j$.
- If $\mathsf{U}_j = \mathsf{U}_j \in Type_i$ by (UNIV), then $Val_i(\mathsf{U}_j) := Type_j$.
- If $\mathsf{Fun}\ A\ F = \mathsf{Fun}\ A'\ F' \in Type_i$ by (FUN), then
  $Val_i(\mathsf{Fun}\ A\ F) := \Pi\ Val_i(A)\ \ (a \mapsto Val_i(F \cdot a))$.

Figure 4: PERs $Type_i$ and $Val_i(A)$

We also need to model subtyping. We use the definition given by Daniel Fridlender and Miguel Pagano ([FP13]).

Definition 4 (Semantic subtyping)**.** We define a relation $\preccurlyeq \subseteq Type \times Type$

1. If $E = E' \in \bot$, then $\uparrow^{\mathsf{U}_n} E \preccurlyeq \uparrow^{\mathsf{U}_n} E'$ for any n.
2. If $j \leqslant i$, then $\mathsf{U}_j \preccurlyeq \mathsf{U}_i$.
3. If $A' \preccurlyeq A$ and $\forall a = a' \in Val\left(A'\right), F \cdot a \preccurlyeq F' \cdot a'$, then $\mathsf{Fun}\ A\ F \preccurlyeq \mathsf{Fun}\ A'\ F'$.

Finally, we can model our judgments. We first define what it means to be a good substitution for a given context (which will be supposed valid) and then define the validity of judgments. Note that we need to quantify over n in the definition of validity and we don't get it as a consequence because the lifting is a rule of our system.

Definition 5.

1. $\tau = \tau' \in Subst\left(\diamond, l\right)$ for all τ, τ'.
2. If $\tau = \tau' \in Subst\left(\Gamma, l\right)$ and $a = a' \in Val\left(\llbracket T \rrbracket \left(0, \tau, l\right)\right)$, then $\mathsf{add}\left(a, \tau\right) = \mathsf{add}\left(a', \tau'\right) \in Subst\left(\Gamma.T, l\right)$.

Definition 6 (Validity)**.**

1. Definitions: $\diamond \models$, if $\diamond \models^l t : T$ and $c \notin l$ then $l.c = t : T \models$ and if $\diamond \models^l T$ and $c \notin l$ then $l.c : T \models$.
2. Contexts: if $l \models$ then $\diamond \models^l$ and if $\Gamma \models^l T$, then $\Gamma.T \models^l$.
3. Types: $\Gamma \models^l T$ if $\Gamma \models^l T = T$.
4. Type equalities: $\Gamma \models^l T = T'$ if $\Gamma \models^l$ and $\forall n, \tau = \tau' \in Subst\left(\Gamma^n, l\right), \llbracket T \rrbracket \left(n, \tau, l\right) = \llbracket T' \rrbracket \left(n, \tau', l\right) \in Type$.
5. Subtyping: $\Gamma \models^l T \leq T'$ if $\Gamma \models^l T, \Gamma \models^l T'$ and $\forall n, \tau = \tau' \in Subst\left(\Gamma^n, l\right), \llbracket T \rrbracket \left(n, \tau, l\right) \preccurlyeq \llbracket T' \rrbracket \left(n, \tau', l\right)$.
6. Terms: $\Gamma \models^l t : T$ if $\Gamma \models^l t = t : T$.
7. Term equalities: $\Gamma \models^l t = t' : T$ if $\Gamma \models^l T$ and $\forall n, \tau = \tau' \in Subst\left(\Gamma^n, l\right), \llbracket t \rrbracket \left(n, \tau, l\right) = \llbracket t' \rrbracket \left(n, \tau', l\right) \in Val\left(\llbracket T \rrbracket \left(n, \tau, l\right)\right)$.
8. Substitutions: $\Gamma \models^l \delta : \Delta$ if $\Gamma \models^l \delta = \delta : \Delta$.
9. Substitution equalities: $\Gamma \models^l \delta = \delta' : \Delta$ if $\Gamma \models^l$, $\Delta \models^l$ and $\forall n, \tau = \tau' \in Subst\left(\Gamma^n, l\right), \llbracket \delta \rrbracket \left(n, \tau, l\right) = \llbracket \delta' \rrbracket \left(n, \tau', l\right) \in Subst\left(\Delta^n, l\right)$.

Proving completeness of NbE is done in a few simple steps:

- Relate $Type$ and Val to $\top$ and $\bot$ through reflection and reification (Lemma 1).
- Prove that if $\Gamma \vdash^l J$ then $\Gamma \models^l J$.
- Conclude, using the fact that $\uparrow_n^{\Gamma, l} \in Subst\left(\Gamma^n, l\right)$ (Theorem 1 and Corollary 1).

Lemma 1. *Suppose $A = A' \in Type$*

1. $\downarrow^{\mathsf{U}_i} A = \downarrow^{\mathsf{U}_i} A' \in \top$.
2. *If $e = e' \in \bot$ then $\uparrow^A e = \uparrow^{A'} e' \in Val\left(A\right)$.*
3. *If $a = a' \in Val\left(A\right)$ then $\downarrow^A a = \downarrow^{A'} a' \in \top$.*

Theorem 1.

- *If $\Gamma \vdash^l T = T'$ then $\mathsf{NbE}_{\Gamma, l}\left(n, T\right) \equiv \mathsf{NbE}_{\Gamma, l}\left(n, T'\right)$.*
- *If $\Gamma \vdash^l t = t' : T$, then $\mathsf{NbE}^T_{\Gamma, l}\left(n, t\right) \equiv \mathsf{NbE}^T_{\Gamma, l}\left(n, t'\right)$.*

Corollary 1 (Completeness of NbE)**.**

- *If $\Gamma \vdash^l T = T'$ then $\mathsf{NbE}_{\Gamma, l}\left(T\right) \equiv \mathsf{NbE}_{\Gamma, l}\left(T'\right)$.*
- *If $\Gamma \vdash^l t = t' : T$, then $\mathsf{NbE}^T_{\Gamma, l}\left(t\right) \equiv \mathsf{NbE}^T_{\Gamma, l}\left(t'\right)$.*

[6]This is where constants will be added.

3.2 Kripke relations and soundness of NbE

Kripke logical relations are often used to build models in intuitionistic logic and in type theory. I won't give a complete overview of logical relations but I will try to make understandable their use in our framework. I had to learn about them and did so by reading articles ([ACD08], [ACP11], [AS12], [FP13]) and an internship report ([Sch11]), and through discussions with Andreas Abel.

To show the soundness of NbE, we need to prove that for any well-typed term t we can derive its equality with the result of its normalization by evaluation. For this purpose, we relate terms to values (using a logical relation) and then show that if a term is related to a value, we can derive the equality of the term with a proper reification of the value. The last step is to show that if a term is well-typed then it is related to its evaluation.

As mentioned by Gabriel Scherer in his presentation of logical relations ([Sch11], Appendix A), negative occurrences of the relation (that is, on the left of an implication) have to be closed under weakening of contexts.

Definition 7 (Context weakening)**.** $\Delta \leq_i^l \Gamma$ iff $\Delta \vdash^l \uparrow_i : \Gamma$ where $\uparrow_0 := \mathsf{id}$ and $\uparrow_{i+1} := \uparrow_i \uparrow$.

Notation 1. $\downarrow_\Gamma^A a := \mathsf{R}_{|\Gamma|}^{nf} \downarrow^A a$.

For types: $\downarrow_\Gamma A$ is used instead of $\downarrow_\Gamma^{\mathsf{U}_n} A$.

Two logical relations are defined below:

- $\Gamma \vdash^l T \circledR A$ relates a type T to a value A.
- $\Gamma \vdash^l t : T \circledR a \in A$ relates a term t to a value a at a type T, which should be related to the value A.

Two cases occur: either the value is a base type or it is a function type. The former is fairly easy to handle but the latter introduces negative occurrences of the relations, hence the need for context weakening closure.

Definition 8 (Logical relations on types and terms)**.** $\Gamma \vdash^l T \circledR A$ and $\Gamma \vdash^l t : T \circledR a \in A$ are defined by induction on $A \in Type_k$:

- $\Gamma \vdash^l T \circledR B$ iff $\forall \Delta \leq_i^l \Gamma, \Delta \vdash^l T \uparrow_i = \downarrow_\Delta B$.
- $\Gamma \vdash^l T : S \circledR A \in \mathsf{U}_i$ iff $\Gamma \vdash^l S = \mathsf{U}_i$ and $\Gamma \vdash^l T \circledR A$.
- $\Gamma \vdash^l t : T \circledR a \in B$ for $B \not\equiv \mathsf{U}_i$ iff $\forall \Delta \leq_i^l \Gamma, \Delta \vdash^l t \uparrow_i = \downarrow_\Delta^B a : T \uparrow_i$.
- $\Gamma \vdash^l T \circledR \mathsf{Fun}\ A\ F$ iff $\Gamma \vdash^l T = \Pi\ R\ \ (\lambda.S)$, $\Gamma \vdash^l R \circledR A$ and $\forall \Delta \leq_i^l \Gamma, \Delta \vdash^l r : R \uparrow_i$ $\circledR\ a \in A \Rightarrow \Delta \vdash^l S\,(\uparrow_i, r)\ \circledR\ F \cdot a$ for some R, S.
- $\Gamma \vdash^l t : T \circledR f \in \mathsf{Fun}\ A\ F$ iff $\Gamma \vdash^l T = \Pi\ R\ \ (\lambda.S)$, $\Gamma \vdash^l R \circledR A$ and $\forall \Delta \leq_i^l \Gamma, \Delta \vdash^l$ $r : R \uparrow_i\ \circledR\ a \in A \Rightarrow \Delta \vdash^l (t \uparrow_i)\ \ r : S\,(\uparrow_i, r)\ \circledR\ f \cdot a \in F \cdot A$ for some R, S.

The next theorem is the first essential point in the proof of soundness of NbE: we can reify a value related to a term and then derive their equality.

Theorem 2. *Suppose* $\Gamma \vdash^l T \circledR A$.

- $\Gamma \vdash^l T = \downarrow_\Gamma A$.
- *If* $\Gamma \vdash^l t : T \circledR a \in A$ *then* $\Gamma \vdash^l t = \downarrow_\Gamma^A a : T$.
- *If* $\forall \Delta \leq_i^l \Gamma, \Delta \vdash^l t \uparrow_i = \mathsf{R}_{|\Delta|}^{ne} e : T \uparrow_i$ *then* $\Gamma \vdash^l t : T \circledR\ \uparrow^A e \in A$.

To end the proof, one needs to show that a well-typed term is related to its evaluation by the logical relation. It will be done by induction on the typing derivation. Because of the rules for explicit substitutions, we first need to relate them to the substitutions contained in the environments.

Definition 9 (Logical relation on substitutions)**.** We define $\Gamma \vdash^l \delta : \Delta \circledR \tau$ by induction on Δ:

- If $\Gamma \vdash^l \delta : \diamond$ then $\Gamma \vdash^l \delta : \diamond \circledR \tau$ for all τ.
- $\Gamma \vdash^l \sigma : \Delta.T \circledR \mathsf{add}(a, \tau)$ iff $\Gamma \vdash^l \sigma = (\delta, t) : \Delta.T$, $\Gamma \vdash^l \delta : \Delta \circledR \tau$, $\Gamma \vdash^l T\delta \circledR [\![T]\!](0, \tau, l)$ and $\Gamma \vdash^l t : T\delta \circledR a \in [\![T]\!](0, \tau, l)$.

We can now show the "fundamental theorem of logical relations". Knowing that Γ is a well-formed context implies $\Gamma \vdash^l \mathsf{id} : \Gamma \circledR \uparrow^{\Gamma, l}$, this theorem will complete the proof of soundness of NbE. Note again the quantification over n, which is a specific feature of our system.

Theorem 3.

- *If* $\Delta \vdash^l T$ *then* $\Gamma \vdash^l \delta : \Delta^n \circledR \tau \Rightarrow \Gamma \vdash^l T^n\delta \circledR [\![T]\!](n, \tau, l)$.
- *If* $\Delta \vdash^l R \leq S$ *then* $\Gamma \vdash^l \delta : \Delta^n \circledR \tau \Rightarrow \Gamma \vdash^l R^n\delta \circledR [\![R]\!](n, \tau, l)$ *and* $\Gamma \vdash^l S^n\delta \circledR [\![S]\!](n, \tau, l)$.
- *If* $\Delta \vdash^l t : T$ *then* $\Gamma \vdash^l \delta : \Delta^n \circledR \tau \Rightarrow \Gamma \vdash^l t^n\delta : T^n\delta \circledR [\![t]\!](n, \tau, l) \in [\![T]\!](n, \tau, l)$.
- *If* $\Delta \vdash^l \sigma : \Sigma$ *then* $\Gamma \vdash^l \delta : \Delta^n \circledR \tau \Rightarrow \Gamma \vdash^l \sigma^n\delta : \Sigma^n \circledR [\![\sigma]\!](n, \tau, l)$.

Theorem 4 (Soundness of NbE)**.**

- *If* $\Gamma \vdash^l T$ *then* $\Gamma \vdash^l T = \mathsf{NbE}_{\Gamma,l}(T)$.
- *If* $\Gamma \vdash^l t : T$ *then* $\Gamma \vdash^l t = \mathsf{NbE}^T_{\Gamma,l}(t) : T$.

4 Implementation

This work having a practical goal, implementing the system was a natural step. We explain here where we started from and give some details about the actual implementation.

4.1 Bidirectional type-checking

A system such as ours cannot be implemented as it is. To check the type of a term, a computer needs either help from the user or rules it can use without having to guess which one to apply.

For this purpose, bidirectional type systems fit well. These are systems in which two kinds of judgments exist: type checking and type inference. For type checking, the user give a term t and a type T and the computer checks that t is actually of type T. Type inference only requires a term as input and outputs a type. This gives an algorithm which is easy to implement on normal forms.

Here, we write $\Gamma \vdash^l v \Leftarrow V$ for "v checks against the type V" and $\Gamma \vdash^l u \Rightarrow T$ for "u is inferred of type T"[7]. The system is given in Figure 5. We use an algorithmic version of subtyping, given below.

Definition 10 (Algorithmic subtyping)**.** Suppose $\Gamma \vdash^l V$ and $\Gamma \vdash^l V'$. We define then $V \trianglelefteqslant V'$ as follows:

- $u \trianglelefteqslant u'$ iff $u \equiv u'$.
- $\mathsf{U}_i \trianglelefteqslant \mathsf{U}_j$ iff $i \leqslant j$.
- $\Pi\ V\ \ (\lambda.W) \trianglelefteqslant \Pi\ V'\ \ (\lambda.W')$ iff $V' \trianglelefteqslant V$ and $W \trianglelefteqslant W'$.

Theorem 5. *Suppose* $\Gamma \vdash^l R$ *and* $\Gamma \vdash^l S$*, then*
$\Gamma \vdash^l R \leq S$ *iff* $\mathsf{NbE}_{\Gamma,l}(R) \trianglelefteqslant \mathsf{NbE}_{\Gamma,l}(S)$.

Note that the rules about definitions correspond to our initial goal: we can declare a term at a certain type and later use it with any shift when it is needed.

[7]Recall v is for normal terms and u for neutral terms.

$$\frac{\Gamma \vdash^l V \Leftarrow \mathsf{U}_n \qquad \Gamma.V \vdash^l W \Leftarrow \mathsf{U}_n}{\Gamma \vdash^l \Pi\ V\ \ (\lambda.W) \Leftarrow \mathsf{U}_n} \qquad \frac{n < m}{\Gamma \vdash^l \mathsf{U}_n \Leftarrow \mathsf{U}_m} \qquad \frac{\Gamma.V \vdash^l v \Leftarrow W}{\Gamma \vdash^l \lambda.v \Leftarrow \Pi\ V\ \ (\lambda.W)}$$

$$\frac{\Gamma \vdash^l u \Rightarrow T \qquad \mathsf{NbE}_{\Gamma,l}\left(T\right) \trianglelefteq V}{\Gamma \vdash^l u \Leftarrow V} \qquad \frac{}{\Gamma.T_i \ldots T_0 \vdash^l \mathsf{v}_i \Rightarrow T_i \uparrow_{i+1}}$$

$$\frac{\Gamma \vdash^l u \Rightarrow T \qquad \mathsf{NbE}_{\Gamma,l}\left(T\right) \equiv \Pi\ V\ \ (\lambda.W) \qquad \Gamma \vdash^l v \Leftarrow V}{\Gamma \vdash^l u\ v \Rightarrow W\left[v\right]} \qquad \frac{c : T \in l}{\Gamma \vdash^l c^n \Rightarrow T^n} \qquad \frac{c = t : T \in l}{\Gamma \vdash^l c^n \Rightarrow T^n}$$

Figure 5: Bidirectional type-checking system

One can show that this system is correct and complete with respect to our original system. Completeness requires a result of compatibility with subtyping which itsefl requires the notion of subtyping of contexts. We refer again to the work of Daniel Fridlender and Miguel Pagano ([FP13]).

Definition 11 (Subtyping of contexts)**.** $\Gamma \leq^l \Delta$ by induction on the contexts:

- $\diamond \leq^l \diamond$.
- If $\Gamma \leq^l \Delta$, $\Delta \vdash^l S$ and $\Gamma \vdash^l R \leq S$ then $\Gamma.R \leq^l \Delta.S$.

Theorem 6 (Correctness of bidirectional type-checking)**.**

- *If* $\Gamma \vdash^l V$ *and* $\Gamma \vdash^l v \Leftarrow V$ *then* $\Gamma \vdash^l v : V$.
- *If* $\Gamma \vdash^l$ *and* $\Gamma \vdash^l u \Rightarrow T$ *then* $\Gamma \vdash^l u : T$.

Theorem 7 (Completeness of bidirectional type-checking)**.** *If* $\Gamma \vdash^l v : T$ *then* $\Gamma \vdash^l v \Leftarrow \mathsf{NbE}_{\Gamma,l}\left(T\right)$.

4.2 Adaptation and integration of the calculus

To implement this system, we started from a type-checker for Mini-TT ([CKNT09]) written in Haskell ([Has]). It can be found in the branch "cleantt" of the "cubical" project, by Cyril Cohen, Thierry Coquand, Simon Huber and Anders Mörtberg[8].

The implementation is however quite different from the bidirectional system in the previous subsection. Indeed, types are evaluated and terms are checked against these values. Thus, we also need to keep track of the environment for evaluations. Type checking is then written $\rho, \Gamma \vdash t \Leftarrow A$ (and similarly for inference), which should be understood as $[\![t]\!]\left(\rho\right) \in Val\left(A\right)$. The context Γ is only kept to have a trace of the variables' type.

The adaptation of the structure of environments aside, we needed to give a rule to infer the type of t^+. The following has been implemented:

$$\frac{\left(n+1, \tau\right), \Gamma \vdash t \Rightarrow A}{\left(n, \tau\right), \Gamma \vdash t^+ \Rightarrow A}$$

This rule corresponds exactly to our notion of evaluation: $[\![t^+]\!]\left(n, \tau\right) \equiv [\![t]\!]\left(n+1, \tau\right)$.

The implementation can be found in the branch "minittplus" of the "cubical" project[9]. Two example files are relevant. The file "test.tt" contains the example of the polymorphic

[8] https://github.com/simhu/cubical/tree/cleantt

[9] https://github.com/simhu/cubical/tree/minittplus

identity, natural numbers, lists and magmas. There is also a proof by reflexivity that a lifted variable is equal to the variable. The file "equality.cub" contains an expression of the univalence axiom (see [Uni13] for an overview of homotopy type theory).

5 Further and related work

We considered two extensions of the system. Even if we have a system in which one can freely declare and lift constants, it would be more useful if it contained what Catarina Coquand calls ""real" dependent types" ([Coq98]). For this purpose, our first extension introduces natural numbers in the system: Nat is the type of natural numbers, we have the constructors Zero and Succ and the primitive recursion Natrec. The lifting operator commutes with Succ and Natrec and is eliminated on Nat and Zero. These rules aside, introducing natural numbers is done the usual way (see [ACP11] or [Abe13] for example).

The second extension deals with an inverse operation for the lifting. Universe levels can be shifted down as well. However, this is not as easy to write as for the lifting operator. With the present hierarchy of universes, it is impossible to have a generic rule like

$$\frac{\Gamma \vdash t : T}{\Gamma^- \vdash t^- : T^-}$$

We get stuck when we try to define U_0^-. It would be inconsistent to reduce it to U_0, since we could derive $\vdash \mathsf{U}_0 : \mathsf{U}_0$. A possibility, briefly mentioned by McBride in the comments of his blogpost ([McB11]), would be to have a hierarchy with levels over $\mathbb{Z}$. This option is totally unexplored to our knowledge. Let us however remark that it is unclear whether we can define a model for such a hierarchy, since the absence of a bottom universe forbids the usual inductive-recursive definitions.

Instead, we defined a system in which we have to seek the first constructor which is not $.^-$ in the considered term to know if we are allowed to write it. The evaluation is extended with $\llbracket t^- \rrbracket (n, \tau) \equiv \llbracket t \rrbracket (n-1, \tau)$ and the evaluation of universes becomes partial $\llbracket \mathsf{U}_n \rrbracket (m, \tau) \equiv \mathsf{U}_{n+m}$ if $n+m$ is non negative.

Moreover, our system seems to be related to Quine's New Foundations ([Qui37]). Indeed, Quine's system is known to be equiconsistent with TST+ (Typed Set Theory with typical ambiguity) ([Spe90]). In TST+, there is an operation which increments every type index in a formula, denoted Φ^+. Typical ambiguity expresses the logical equivalence between Φ and Φ^+ for any Φ. TST+ has recently been proven consistent by Murdoch Gabbay ([Gab14]), using nominal sets (see [Pit13] for an overview of nominal sets).

Conclusion

The goal of this work was to make precise the proposition of Conor McBride for a lifting operator for a cumulative hierarchy of universes ([McB11]). The purpose of this operator is to allow in a simple way the combination of cumulativity and universe polymorphism in a proof assistant.

The introduction of this operator in the syntax of terms brought complications in the handling of variables and substitutions. However, they are solved by using adapted structures such as explicit substitutions and level annotations in environments.

We proved weak normalization for the resulting system using normalization by evaluation. A partial equivalence relation model was built to prove the completeness of this normalization. For its soundness, we needed Kripke logical relations. An important remark is that we used the standard instantiations of these structures, the adaptation of environments put aside. Only the inductive hypotheses were to be adapted to get our results.

We designed a type checking algorithm and implemented an adapted version of it. The adaptation is based on the link with presheaf models of type theory, a field still in development. We provided some examples we believe to be relevant.

A natural extension to the system, defining an inverse operation for the lifting, raised the question of a hierarchy with levels over $\mathbb{Z}$. The usual method for building models seems to be unusable in this context.

The link between our system and Quine's New Foundations still has to be explored. Especially, "consistent renamings of levels", which are used to express typical ambiguity in Gabbay's proof of consistency for TST+ ([Gab14]), might prove useful in a simpler expression of both the lifting operator and its inverse.

A Rules of the calculus

Definitions.

$$\frac{}{\diamond\ \mathsf{ok}}\ (\textsc{empty-def}) \qquad \frac{l\ \mathsf{ok} \qquad \diamond \vdash^l T \qquad c \notin l}{l.c : T\ \mathsf{ok}}\ (\textsc{prim-def})$$

$$\frac{l\ \mathsf{ok} \qquad \diamond \vdash^l t : T \qquad c \notin l}{l.c = t : T\ \mathsf{ok}}\ (\textsc{def})$$

Contexts.

$$\frac{l\ \mathsf{ok}}{\diamond \vdash^l}\ (\textsc{empty-ctx}) \qquad \frac{\Gamma \vdash^l \qquad \Gamma \vdash^l T}{\Gamma.T \vdash^l}\ (\textsc{ext-ctx})$$

Substitutions.

$$\frac{\Gamma \vdash^l}{\Gamma \vdash^l \mathsf{id} : \Gamma}\ (\textsc{id-subs}) \qquad \frac{\Sigma \vdash^l \delta : \Delta \qquad \Gamma \vdash^l \sigma : \Sigma}{\Gamma \vdash^l \delta\sigma : \Delta}\ (\textsc{comp-subs})$$

$$\frac{\Gamma \vdash^l}{\Gamma \vdash^l \langle\rangle : \diamond}\ (\textsc{ept-subs}) \qquad \frac{\Gamma \vdash^l T}{\Gamma.T \vdash^l \uparrow : \Gamma}\ (\textsc{fst-subs})$$

$$\frac{\Gamma \vdash^l \delta : \Delta \qquad \Delta \vdash^l T \qquad \Gamma \vdash^l t : T\delta}{\Gamma \vdash^l (\delta, t) : \Delta.T}\ (\textsc{ext-subs})$$

Terms.

$$\frac{\Gamma \vdash^l}{\Gamma \vdash^l \mathsf{U}_n : \mathsf{U}_{n+1}}\ (\textsc{u-u-f}) \qquad \frac{\Gamma \vdash^l R : \mathsf{U}_n \qquad \Gamma.R \vdash^l S : \mathsf{U}_n}{\Gamma \vdash^l \Pi\ R\ (\lambda.S) : \mathsf{U}_n}\ (\textsc{fun-u-f})$$

$$\frac{\Gamma \vdash^l \qquad \Gamma(i) \equiv T}{\Gamma \vdash^l \mathsf{v}_i : T}\ (\textsc{hyp}) \qquad \frac{\Gamma \vdash^l R \qquad \Gamma.R \vdash^l S \qquad \Gamma.R \vdash^l t : S}{\Gamma \vdash^l \lambda.t : \Pi\ R\ (\lambda.S)}\ (\textsc{fun-i})$$

$$\frac{\Gamma \vdash^l R \qquad \Gamma.R \vdash^l S \qquad \Gamma \vdash^l t : \Pi\ R\ (\lambda.S) \qquad \Gamma \vdash^l r : R}{\Gamma \vdash^l t\ r : S[r]}\ (\textsc{fun-el})$$

$$\frac{\Delta \vdash^l T \qquad \Delta \vdash^l t : T \qquad \Gamma \vdash^l \delta : \Delta}{\Gamma \vdash^l t\delta : T\delta}\ (\textsc{subs-term}) \qquad \frac{\Gamma \vdash^l t : R \qquad \Gamma \vdash^l R \leq S}{\Gamma \vdash^l t : S}\ (\textsc{sub})$$

$$\frac{\diamond \vdash^l t : T \qquad \Gamma \vdash^l \langle\rangle : \diamond}{\Gamma \vdash^l t : T}\ (\textsc{closed-term}) \qquad \frac{\Gamma \vdash^l t : T}{\Gamma^+ \vdash^l t^+ : T^+}\ (\textsc{lift-term})$$

$$\frac{\Gamma \vdash^l \qquad c : T \in l}{\Gamma \vdash^l c : T}\ (\textsc{prim-def-term}) \qquad \frac{\Gamma \vdash^l \qquad c = t : T \in l}{\Gamma \vdash^l c : T}\ (\textsc{def-term})$$

Subtypes.

$$\frac{\Gamma \vdash^l R = S}{\Gamma \vdash^l R \leq S}\ (\textsc{refl-sty}) \qquad \frac{\Gamma \vdash^l}{\Gamma \vdash^l \mathsf{U}_n \leq \mathsf{U}_{n+1}}\ (\textsc{u-sty})$$

$$\frac{\Gamma \vdash^l R' \leq R \qquad \Gamma.R \vdash^l S \qquad \Gamma.R' \vdash^l S \leq S'}{\Gamma \vdash^l \Pi\ R\ (\lambda.S) \leq \Pi\ R'\ (\lambda.S')}\ (\textsc{fun-sty})$$

$$\frac{\Gamma \vdash^l R \leq R' \qquad \Gamma \vdash^l R' \leq R''}{\Gamma \vdash^l R \leq R''}\ (\textsc{trans-sty}) \qquad \frac{\Delta \vdash^l R \leq S \qquad \Gamma \vdash^l \delta : \Delta}{\Gamma \vdash^l R\delta \leq S\delta}\ (\textsc{subs-sty})$$

$$\frac{\Gamma \vdash^l R \leq S}{\Gamma^+ \vdash^l R^+ \leq S^+}\ (\textsc{lift-sty})$$

Figure 6: Typing rules

$$\frac{\Gamma \vdash^{l} t : T}{\Gamma \vdash^{l} t = t : T}\ (\textsc{refl-et}) \qquad \frac{\Gamma \vdash^{l} t = t' : T \qquad \Gamma \vdash^{l} t = t'' : T}{\Gamma \vdash^{l} t' = t'' : T}\ (\textsc{sym-tran-et})$$

$$\frac{\Gamma \vdash^{l} t : T}{\Gamma \vdash^{l} t\mathsf{id} = t : T}\ (\textsc{neut-et}) \qquad \frac{\Gamma \vdash^{l} \delta : \Delta}{\Gamma \vdash^{l} \mathsf{U}_n\delta = \mathsf{U}_n : \mathsf{U}_{n+1}}\ (\textsc{u-subs-et})$$

$$\frac{\Gamma \vdash^{l} \delta : \Delta \qquad \Delta \vdash^{l} T \qquad \Gamma \vdash^{l} t : T\delta}{\Gamma \vdash^{l} \mathsf{v}_0\,(\delta, t) = t : T\delta}\ (\textsc{snd-et}) \qquad \frac{\Gamma \vdash^{l} \qquad \Gamma\,(i+1) \equiv T}{\Gamma \vdash^{l} \mathsf{v}_i \uparrow = \mathsf{v}_{i+1} : T}\ (\textsc{w-var-et})$$

$$\frac{\Gamma \vdash^{l} \delta : \Delta \qquad \Gamma \vdash^{l} s : S \qquad \Delta\,(i) \equiv T}{\Gamma \vdash^{l} \mathsf{v}_{i+1}\,(\delta, s) = \mathsf{v}_i\delta : T}\ (\textsc{sub-var-et})$$

$$\frac{\diamond \vdash^{l} t : T \qquad \Gamma \vdash^{l} \langle\rangle : \diamond}{\Gamma \vdash t\langle\rangle = t : T\langle\rangle}\ (\textsc{ept-closed-et})$$

$$\frac{\Gamma \vdash^{l} R = R' : \mathsf{U}_n \qquad \Gamma.R' \vdash^{l} S' : \mathsf{U}_n \qquad \Gamma.R \vdash^{l} S = S' : \mathsf{U}_n}{\Gamma \vdash^{l} \Pi\ R\ \ (\lambda.S) = \Pi\ R'\ \ \left(\lambda.S'\right) : \mathsf{U}_n}\ (\textsc{cong-fun-et})$$

$$\frac{\Delta \vdash^{l} R : \mathsf{U}_n \qquad \Delta.R \vdash^{l} S : \mathsf{U}_n \qquad \Gamma \vdash^{l} \delta : \Delta}{\Gamma \vdash^{l} (\Pi\ R\ \ (\lambda.S))\,\delta = \Pi\ \ (R\delta)\ \ (\lambda.S\,(\delta\uparrow, \mathsf{v}_0)) : \mathsf{U}_n}\ (\textsc{fun-subs-et})$$

$$\frac{\Gamma \vdash^{l} R \qquad \Gamma.R \vdash^{l} S \qquad \Gamma.R \vdash^{l} t = t' : S}{\Gamma \vdash^{l} \lambda.t = \lambda.t' : \Pi\ R\ \ (\lambda.S)}\ (\textsc{xi-et})$$

$$\frac{\Delta \vdash^{l} R \qquad \Delta.R \vdash^{l} S \qquad \Delta.R \vdash^{l} t : S \qquad \Gamma \vdash^{l} \delta : \Delta}{\Gamma \vdash^{l} (\lambda.t)\,\delta = \lambda\,(t\,(\delta\uparrow, \mathsf{v}_0)) : (\Pi\ R\ \ (\lambda.S))\,\delta}\ (\textsc{abs-subs-et})$$

$$\frac{\Gamma \vdash^{l} R \qquad \Gamma.R \vdash^{l} S \qquad \Gamma \vdash^{l} t = t' : \Pi\ R\ \ (\lambda.S) \qquad \Gamma \vdash^{l} r = r' : R}{\Gamma \vdash^{l} t\ r = t'\ r' : S\,[r]}\ (\textsc{cong-app-et})$$

$$\frac{\Delta \vdash^{l} R \qquad \Delta.R \vdash^{l} S \qquad \Delta \vdash^{l} t : \Pi\ R\ \ (\lambda.S) \qquad \Delta \vdash^{l} r : R \qquad \Gamma \vdash^{l} \delta : \Delta}{\Gamma \vdash^{l} (t\ r)\,\delta = t\delta\ \ (r\delta) : (S\,[r])\,\delta}\ (\textsc{app-subs-et})$$

$$\frac{\Delta \vdash^{l} t = t' : T \qquad \Gamma \vdash^{l} \delta = \delta' : \Delta}{\Gamma \vdash^{l} t\delta = t'\delta' : T\delta}\ (\textsc{cong-subs-et})$$

$$\frac{\Delta \vdash^{l} T \qquad \Delta \vdash^{l} t : T \qquad \Sigma \vdash^{l} \delta : \Delta \qquad \Gamma \vdash^{l} \sigma : \Sigma}{\Gamma \vdash^{l} (t\delta)\,\sigma = t\,(\delta\sigma) : (T\delta)\,\sigma}\ (\textsc{assoc-et})$$

$$\frac{\Gamma \vdash^{l} R \qquad \Gamma.R \vdash^{l} S \qquad \Gamma.R \vdash^{l} t : S \qquad \Gamma \vdash^{l} r : R}{\Gamma \vdash^{l} (\lambda.t)\ \ r = t\,[r] : S\,[r]}\ (\textsc{beta-et})$$

$$\frac{\Gamma \vdash^{l} t = t' : R \qquad \Gamma \vdash^{l} R \leq S}{\Gamma \vdash^{l} t = t' : S}\ (\textsc{sub-et}) \frac{\Gamma \vdash^{l} R \qquad \Gamma.R \vdash^{l} S \qquad \Gamma \vdash^{l} t : \Pi\ R\ \ (\lambda.S)}{\Gamma \vdash^{l} t = \lambda.\,(t\uparrow\ \ \mathsf{v}_0) : \Pi\ R\ \ (\lambda.S)}\ (\textsc{eta-et})$$

$$\frac{\Gamma \vdash^{l} t = t' : T}{\Gamma^{+} \vdash^{l} t^{+} = t'^{+} : T^{+}}\ (\textsc{lift-et}) \qquad \frac{\Gamma \vdash^{l} T \qquad \Gamma \vdash^{l} \mathsf{v}_i : T}{\Gamma^{+} \vdash^{l} \mathsf{v}_i^{+} = \mathsf{v}_i : T^{+}}\ (\textsc{lift-var-et})$$

$$\frac{\Gamma \vdash^{l} R \qquad \Gamma.R \vdash^{l} S \qquad \Gamma \vdash^{l} \lambda.t : \Pi\ R\ \ (\lambda.S)}{\Gamma^{+} \vdash^{l} (\lambda.t)^{+} = \lambda.\left(t^{+}\right) : (\Pi\ R\ \ (\lambda.S))^{+}}\ (\textsc{lift-abs-et})$$

$$\frac{\Gamma \vdash^{l} R \qquad \Gamma.R \vdash^{l} S \qquad \Gamma \vdash^{l} t : \Pi\ R\ \ (\lambda.S) \qquad \Gamma \vdash^{l} r : R}{\Gamma^{+} \vdash^{l} (t\ r)^{+} = t^{+}\ r^{+} : (S\,[r])^{+}}\ (\textsc{lift-app-et})$$

$$\frac{\Gamma \vdash^{l} R : \mathsf{U}_n \qquad \Gamma.R \vdash^{l} S : \mathsf{U}_n}{\Gamma^{+} \vdash^{l} (\Pi\ R\ \ (\lambda.S))^{+} = \Pi\ R^{+}\ \ \left(\lambda.S^{+}\right) : \mathsf{U}_n^{+}}\ (\textsc{lift-fun-et})$$

$$\frac{\Delta \vdash^{l} T \qquad \Delta \vdash^{l} t : T \qquad \Gamma \vdash^{l} \delta : \Delta}{\Gamma^{+} \vdash^{l} (t\delta)^{+} = t^{+}\delta^{+} : (T\delta)^{+}}\ (\textsc{lift-subs-et})$$

$$\frac{\Gamma \vdash^{l}}{\Gamma^{+} \vdash^{l} \mathsf{U}_n^{+} = \mathsf{U}_{n+1} : \mathsf{U}_{n+2}}\ (\textsc{lift-u-et}) \qquad \frac{\Gamma \vdash^{l} \qquad c = t : T \in l}{\Gamma \vdash^{l} c = t : T}\ (\textsc{def-et})$$

Figure 7: Equality rules for terms

$$\frac{\Gamma \vdash^l \delta : \Delta}{\Gamma \vdash^l \delta = \delta : \Delta}\ (\textsc{refl-es}) \qquad \frac{\Gamma \vdash^l \delta : \Delta}{\Gamma \vdash^l \mathsf{id}\delta = \delta : \Delta}\ (\textsc{neut-l-es})$$

$$\frac{}{\diamond \vdash^l \mathsf{id} = \langle\rangle : \diamond}\ (\textsc{id-ept-es}) \qquad \frac{\Gamma \vdash^l \delta : \Delta}{\Gamma \vdash^l \langle\rangle\delta = \langle\rangle : \diamond}\ (\textsc{comp-ept-es})$$

$$\frac{\Gamma \vdash^l \delta = \delta' : \Delta \qquad \Gamma \vdash^l \delta = \delta'' : \Delta}{\Gamma \vdash^l \delta' = \delta'' : \Delta}\ (\textsc{sym-trans-es}) \qquad \frac{\Gamma \vdash^l \delta : \Delta}{\Gamma \vdash^l \delta\mathsf{id} = \delta : \Delta}\ (\textsc{neut-r-es})$$

$$\frac{\Gamma \vdash^l T}{\Gamma.T \vdash^l \mathsf{id} = (\uparrow, \mathsf{v}_0) : \Gamma.T}\ (\textsc{id-ext-es})$$

$$\frac{\Sigma \vdash^l \delta : \Delta \qquad \Delta \vdash^l T \qquad \Sigma \vdash^l t : T\delta \qquad \Gamma \vdash^l \sigma : \Sigma}{\Gamma \vdash^l (\delta, t)\,\sigma = (\delta\sigma, t\sigma) : \Delta.T}\ (\textsc{ext-es})$$

$$\frac{\Theta \vdash^l \delta : \Delta \qquad \Sigma \vdash^l \theta : \Theta \qquad \Gamma \vdash^l \sigma : \Sigma}{\Gamma \vdash^l (\delta\theta)\,\sigma = \delta\,(\theta\sigma) : \Delta}\ (\textsc{assoc-es})$$

$$\frac{\Gamma \vdash^l \delta : \Delta \qquad \Delta \vdash^l T \qquad \Gamma \vdash^l t : T\delta}{\Gamma \vdash^l \uparrow (\delta, t) = \delta : \Delta}\ (\textsc{fst-es})$$

$$\frac{\Gamma \vdash^l \sigma = \sigma' : \Sigma \qquad \Sigma \vdash^l \delta = \delta' : \Delta}{\Gamma \vdash^l \delta\sigma = \delta'\sigma' : \Delta}\ (\textsc{cong-comp-es})$$

$$\frac{\Gamma \vdash^l \delta = \delta' : \Delta \qquad \Delta \vdash^l T \qquad \Gamma \vdash^l t = t' : T\delta}{\Gamma \vdash^l (\delta, t) = (\delta', t') : \Delta.T}\ (\textsc{cong-ext-es})$$

Figure 8: Equality rules for substitutions

B Evaluation and read-back

Evaluation and application:

$$\begin{array}{rcl}
[i]\,(\tau) & := & \tau\,(i) \\
[\![\mathsf{v}_i]\!]\,(n, \tau, l) & := & [i]\,(\tau) \\
[\![c]\!]\,(n, \tau, l) & := & \uparrow^{[\![T]\!](n,\tau,l)} c^n \text{ if } c : T \in l \\
[\![c]\!]\,(n, \tau, l) & := & [\![t]\!]\,(n, \tau, l) \text{ if } c = t : T \in l \\
[\![\lambda.t]\!]\,(\rho) & := & (\lambda.t)\,\rho \\
[\![r\ s]\!]\,(\rho) & := & [\![r]\!]\,(\rho) \cdot [\![s]\!]\,(\rho) \\
[\![\mathsf{U}_n]\!]\,(m, \tau, l) & := & \mathsf{U}_{n+m} \\
[\![t^+]\!]\,(n, \tau, l) & := & [\![t]\!]\,(n+1, \tau, l) \\
[\![\Pi\ R\ S]\!]\,(\rho) & := & \mathsf{Fun}\ [\![R]\!]\,(\rho)\ \ [\![S]\!]\,(\rho) \\
[\![t\sigma]\!]\,(n, \tau, l) & := & [\![t]\!]\,(n, [\![\sigma]\!]\,(n, \tau, l)\,, l) \\
[\![\uparrow]\!]\,(n, \mathsf{add}\,(a, \tau)\,, l) & := & \tau \\
[\![\mathsf{id}]\!]\,(n, \tau, l) & := & \tau \\
[\![\langle\rangle]\!]\,(\rho) & := & i \mapsto \mathsf{U}_0 \\
[\![(\sigma, t)]\!]\,(\rho) & := & \mathsf{add}\,([\![t]\!]\,(\rho)\,, [\![\sigma]\!]\,(\rho)) \\
[\![\sigma\delta]\!]\,(n, \tau, l) & := & [\![\sigma]\!]\,(n, [\![\delta]\!]\,(n, \tau, l)\,, l) \\
\left(\uparrow^{\mathsf{Fun}\ A\ F} e\right) \cdot a & := & \uparrow^{F \cdot a} (e\ d) \text{ where } d \equiv \downarrow^A a \\
(\lambda.t)\,(n, \tau, l) \cdot a & := & [\![t]\!]\,(n, \mathsf{add}\,(a, \tau)\,, l)
\end{array}$$

Read-back:

$$\begin{array}{rcl}
\mathsf{R}^{nf}_{n} \downarrow^{\mathsf{Fun}\ A\ F} f & := & \lambda.\mathsf{R}^{nf}_{n+1} \downarrow^{F \cdot a} (f \cdot a) \ \text{ where } a \equiv \uparrow^{A} \mathsf{x}_n \\
\mathsf{R}^{nf}_{n} \downarrow^{\mathsf{U}_k} \mathsf{U}_i & := & \mathsf{U}_i \\
\mathsf{R}^{nf}_{n} \downarrow^{\mathsf{U}_i} \mathsf{Fun}\ A\ F & := & \Pi\ \left(\mathsf{R}^{nf}_{n} \downarrow^{\mathsf{U}_i} A\right)\ \left(\mathsf{R}^{nf}_{n} \downarrow^{\mathsf{Fun}\ A\ (\lambda.\mathsf{v}_1)(0,\mathsf{add}(\mathsf{U}_i,i\mapsto\mathsf{U}_0),\diamond)} F\right) \\
\mathsf{R}^{nf}_{n} \downarrow^{B} \uparrow^{B'} e & := & \mathsf{R}^{ne}_{n} e \\
\mathsf{R}^{ne}_{n} c^k & := & c^k \\
\mathsf{R}^{ne}_{n} \mathsf{x}_k & := & \mathsf{v}_{n-(k+1)} \\
\mathsf{R}^{ne}_{n} (e\ d) & := & \left(\mathsf{R}^{ne}_{n} e\right)\ \left(\mathsf{R}^{nf}_{n} d\right)
\end{array}$$

References

[AAD07] Andreas Abel, Klaus Aehlig, and Peter Dybjer. Normalization by evaluation for martin-löf type theory with one universe. *Electr. Notes Theor. Comput. Sci.*, 173:17–39, 2007.

[Abe10] Andreas Abel. On parametric polymorphism and irrelevance in martin-löf type theory, 2010.

[Abe13] Andreas Abel. Normalization by evaluation – dependent types and impredicativity. Habilitation thesis, Institut für Informatik, Ludwig-Maximilians-Universität München, 2013.

[AC05] Andreas Abel and Thierry Coquand. Untyped algorithmic equality for martin-löf's logical framework with surjective pairs. In Pawel Urzyczyn, editor, *TLCA*, volume 3461 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2005.

[AC14] Andreas Abel and James Chapman. Normalization by evaluation in the delay monad: A case study for coinduction via copatterns and sized types. In Paul Levy and Neel Krishnaswami, editors, *MSFP*, volume 153 of *EPTCS*, pages 51–67, 2014.

[ACD08] Andreas Abel, Thierry Coquand, and Peter Dybjer. Verifying a semantic beta-eta-conversion test for martin-löf type theory. In Philippe Audebaud and Christine Paulin-Mohring, editors, *MPC*, volume 5133 of *Lecture Notes in Computer Science*, pages 29–56. Springer, 2008.

[ACP11] Andreas Abel, Thierry Coquand, and Miguel Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. *Logical Methods in Computer Science*, 7(2), 2011.

[Agd] Agda. http://wiki.portal.chalmers.se/agda/pmwiki.php.

[AS12] Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science*, 8(1), 2012.

[CD97] Thierry Coquand and Peter Dybjer. Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science*, 7(1):75–94, 1997.

[CJ10] Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundam. Inform.*, 100(1-4):43–52, 2010.

[CKNT09] Thierry Coquand, Yoshiki Kinoshita, Bengt Nordström, and Makoto Takeyama. A simple type-theoretic language: Mini-tt. In Yves Bertot, Gérard Huet, Jean-Jacques Lévy, and Gordon Plotkin, editors, *From Semantics to Computer Science*, pages 139–164. Cambridge University Press, 2009. Cambridge Books Online.

[Coq] Coq. http://coq.inria.fr/.

[Coq98] Catarina Coquand. A realizability interpretation of martin-löf's type theory, 1998.

[Cur91] Pierre-Louis Curien. An abstract framework for environment machines. *Theor. Comput. Sci.*, 82(2):389–402, 1991.

[FP13] Daniel Fridlender and Miguel Pagano. A type-checking algorithm for martin-löf type theory with subtyping based on normalisation by evaluation. In Masahito Hasegawa, editor, *TLCA*, volume 7941 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 2013.

[Gab14] Murdoch James Gabbay. Consistency of quine's new foundations using nominal techniques. *CoRR*, abs/1406.4060, 2014.

[Gir72] J. Y Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université de Paris 7, 1972.

[Has] Haskell. http://www.haskell.org.

[Her05] Hugo Herbelin. Type inference with algebraic universes in the calculus of inductive constructions, 2005.

[HP91] Robert Harper and Robert Pollack. Type checking with universes. *Theor. Comput. Sci.*, 89(1):107–136, 1991.

[McB11] Conor McBride. Crude but effective stratification, 2011. http://mazzo.li/epilogue/index.html%3Fp=857&cpage=1.html.

[ML72] Per Martin-Löf. An intuitionistic theory of types, 1972. Technical report.

[ML84] Per Martin-Löf. Intuitionistic type theory, 1984. Bibliopolis.

[Pal98] Erik Palmgren. On universes in type theory. Oxford University Press, 1998.

[Pit13] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

[Qui37] W. V. Quine. New foundations for mathematical logic. *The American Mathematical Monthly*, 44(2):pp. 70–80, 1937.

[Sch11] Gabriel Scherer. Universe subtyping in martin-löf type theory, 2011. Internship report.

[Spe90] Ernst Specker. Typical ambiguity. In Gerhard Jäger, Hans Läuchli, Bruno Scarpellini, and Volker Strassen, editors, *Ernst Specker Selecta*, pages 193–201. Birkhäuser Basel, 1990.

[ST14] Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in coq. In Gerwin Klein and Ruben Gamboa, editors, *ITP*, volume 8558 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2014.

[Tai75] WilliamW. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer Berlin Heidelberg, 1975.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.