Universe Polymorphism Expressed as a Rewriting System

Guillaume Genestier[123]

[1] Université Paris-Saclay, ENS Paris-Saclay, Inria, CNRS, LSV
[2] MINES ParisTech, PSL University
[3] This work was supported by the Cost Action EUTypes CA15123

The $\lambda\Pi$-calculus modulo rewriting ($\lambda\Pi/\mathcal{R}$ for short) is a system of dependent types where types are identified modulo the β-reduction of λ-calculus and rewriting rules given by the user to define not only functions but also types.

Cousineau and Dowek [3] showed that $\lambda\Pi/\mathcal{R}$ is well-suited to encode a whole class of rich logics: *Functional Pure Type System* (PTS) [2]. To do so, they use a symbol `Univ`$_s$ for each sort s, which contains the codes of elements of this sort and the associated decoder `T`$_s$. Then `code` and `prod` reflect the PTS axioms and rules, respectively. For the simply typed λ-calculus, which is the PTS with $\mathcal{S} = \{*, \square\}$, $\mathcal{A} = \{(*, \square)\}$ and $\mathcal{R} = \{(*, *, *)\}$, the encoding is:

```
symbol Univ□ : TYPE.    symbol T□ : Univ□ ⇒ TYPE.   symbol code*□ : Univ□.
symbol Univ* : TYPE.    symbol T* : Univ* ⇒ TYPE.   T□ code*□ ⟶ Univ*.
symbol prod*** : (A : Univ*) ⇒ (T* A ⇒ Univ*) ⇒ Univ*.
T* (prod*** A B) ⟶ (x : T* A) ⇒ T* (B x).
```

In their encoding, every sort has its own symbol, and every rule has its associated product symbol. However, having an infinite number of symbols and rules is not well-suited for practical implementations. Hence, to encode PTS with an infinite number of sorts, Assaf suggested to have a type `Sort` for sorts and a single symbol for products [1]. For *Full Pure Type Systems*[1] this extension is quite direct: `Univ`, `T`, `code` and `prod` are now symbols in the syntax and the meta-arguments of type `Sort` are now real arguments in the syntax. The peculiarity of each PTS is reflected in the encoding of `Sort` and of the functions `axiom` and `rule`.

Let us suppose that all sorts are of the form Set_ℓ with $\ell \in \mathbb{L}$ called a level [2]. It is common to enrich PTS with *Universe Polymorphism* [4], *i.e.* add the possibility for the user to quantify over universe levels, introducing $\forall \ell, \mathrm{Set}_\ell$ among the terms. Indeed, just like we use polymorphism to avoid declaring a type of lists for each type of elements, we do not want to declare a new type for each level. Hence, we want to declare `List` in $\forall \ell, (A : \mathrm{Set}_\ell) \to \mathrm{Set}_\ell$.

To assign a type to $\forall \ell, \mathrm{Set}_\ell$, a new sort Set_ω is introduced, which is not typable, is the type of no sort and over which one cannot quantify. This sort is for internal purposes only, it is not in the syntax of the system we are encoding (even if it is in the syntax of the encoded version of the system). In addition to this new sort, we add to the encoding a new symbol $\forall_{\mathbb{L}}$ which reprents this universal quantification.

```
symbol setOmega : Sort.                          symbol set : 𝕃 ⇒ Sort.
symbol ∀𝕃 : (f : (𝕃⇒Sort)) ⇒ ((l:𝕃) ⇒ Univ (f l)) ⇒ Univ setOmega.
T _ (∀𝕃 f t) ⟶ (l : 𝕃) ⇒ T (f l) (t l).
```

For instance, the encoding of $\forall \ell, \mathrm{Set}_\ell$ is `∀𝕃 (λ l, axiom (set l)) (λ l, code (set l))`. And its decoding (when applying `T setOmega`) is, as expected, `(l:𝕃) -> Univ (set l)`.

Definition 1 (Translation)**.** Given a well-typed term t in a Universe Polymorphic Full Pure Type System, we translate it by: $|x|$=`x` $\quad |\mathrm{Set}_\ell|$=`code` $\|\mathrm{Set}_\ell\|$; $\quad \|\mathrm{Set}_\omega\|$=`setOmega`;

---

[1] A PTS is called *full* if axioms and rules are total functions, respectively from $\mathcal{S}$ and $\mathcal{S} \times \mathcal{S}$ to $\mathcal{S}$. This definition is more restrictive than the one given in [5], where axioms are not enforced to be total.

[2] We could also, without difficulty, consider several hierarchies sharing the same levels, like Set_ℓ and Prop_ℓ.

$\|\text{Set}_\ell\| = \texttt{set}\ |\ell|_{\mathbb{L}}$, if $\ell \neq \omega$; $\quad |(x : A) \to B| = \texttt{prod}\ \|s\|\ \|s'\|\ |A|\ (\lambda \texttt{x} : \texttt{T}\ \|s\|\ |A|.|B|)$;
$|\lambda x^A.t| = \lambda(\texttt{x} : \texttt{T}\ \ \|s\|\ |A|).|t|$; $\quad |\forall \ell, A| = \forall_{\mathbb{L}}\ (\lambda \ell : \mathbb{L}.\ \|s\|)\ (\lambda \ell : \mathbb{L}.\ |A|)$.

Each time it is used, s is the sort of A and s' the one of B.

It can be noted that the translation $|.|_{\mathbb{L}}$ of levels is not given yet. Indeed, with universe polymorphism, universe levels are open terms, hence, convertibility between universe levels is now an issue. Fortunately, it is the last one, since once this issue is overcomed, the encoding has one of the expected properties: we check at least as much as in the original system.

Theorem 2 (Correctness)**.** If the translation function is such that equality of levels implies convertibility of their translations, if $\Gamma \vdash_P t : A$, in a Universe Polymorphic Full Pure Type System P, then $|\Gamma| \vdash_{\lambda\Pi/P} |t| : \texttt{T}\ \|s\|\ |A|$, where s is the sort of A.

Of course, collapsing all levels satisfies the first hypothesis of the theorem. However it is not satisfactory, since it comes down to do an encoding in the inconsistent PTS with only one sort.

We present a correct and complete rewriting system modulo associativity and commutativity (AC), to decide level equality for the PTS where levels are natural numbers and axioms and rules are respectively the functions successor and max [3] [4]. The whole encoding, written in Dedukti, can be found in github.com/Deducteam/Agda2Dedukti, in the files `theory/Agda.dk` and `theory/univ.dk`.

More formally, given the grammar $t, u ::= x \in \mathcal{X} \mid 0 \mid s\,t \mid \max t\,u$, every term t in this grammar has a unique normal form denoted $t\downarrow$, such that $t\downarrow \equiv_{AC} u\downarrow$ if and only if for all $\sigma : \mathcal{X} \to \mathbb{N}$, $[\![t]\!]_\sigma = [\![u]\!]_\sigma$, where the interpretation of 0, s and max are the expected ones.

We must note that having a confluent system is not an issue here, since we desire the unique normal form property only for some specific terms. We obtain this thanks to the guarantee that all variables are of type $\mathbb{L}$.

With our system a normal form is either a variable, or of the form Max $i\ \{j_k + x_k\}_k$ with $x_1, x_2, \ldots$ distinct variables, and $i, j_1, j_2, \ldots$ ground natural numbers such that for all k, $i \geqslant j_k$. It must be noted, that we do not have $+$ in our original grammar, however encoding $s^n(x)$ as $n + x$ avoids to duplicate infinitely rewrite rules, depending on the number of s applied. The first argument is counting iterations, it is why it is restricted to be a ground natural number.

So that they are not confused with levels, a separate type $\mathbb{N}$ of ground natural numbers is introduced[5]. To encode sets, we use symbols modulo AC, since a set is either empty, a singleton of the form $\{i + x\}$, or the union of two sets. The only non-left-linear rule of the encoding eliminates redundancies, ensuring that all variables in the normal forms are distinct.

```
symbol ∅ : LSet.     infix ⊕ : ℕ⇒𝕃⇒LSet.     infix ac ∪ : LSet⇒LSet⇒LSet.
x ∪ ∅ ⟶ x.          (i ⊕ l) ∪ (j ⊕ l) ⟶ maxℕ i j ⊕ l.
```

The rule stating that $i + \max(t, u) = \max(i + t, i + u)$ must not break the ordering invariant. Hence it has to update the natural number at the head of `Max : ℕ ⇒ LSet ⇒ 𝕃` [6].

```
Max i  (j ⊕ Max k l)        ⟶ Max (maxℕ i (j +ℕ k)) (mapPlus j l).
Max i ((j ⊕ Max k l) ∪ tl)  ⟶ Max (maxℕ i (j +ℕ k)) (mapPlus j l ∪ tl).
```

We can now reflect the syntax we are interested in, using `Max`.

```
0 ⟶ Max 0ℕ ∅.                        s x ⟶ Max 1ℕ (1ℕ ⊕ x).
max x y ⟶ Max 0ℕ ((0ℕ ⊕ x) ∪ (0ℕ ⊕ y)).
```

[3] It is the level hierarchy behind the proof-assistant Agda, which has two families of sorts Prop_ℓ and Set_ℓ.
[4] The impredicative version, behind Coq and Lean, can also be encoded using a similar technique.
[5] Symbols of type $\mathbb{N}$ are in red and indexed with $\mathbb{N}$.
[6] Rules defining `mapPlus` of type `ℕ ⇒ LSet ⇒ LSet` can easily be inferred and is not detailed.

References

[1] A. Assaf. *A Framework for Defining Computational Higher-Order Logics.* PhD thesis, École polytechnique, 2015.

[2] H. Barendregt. Lambda calculi with types. In *Handbook of logic in computer science. Volume 2. Background: computational structures*, p. 117–309. Oxford University Press, 1992.

[3] D. Cousineau and G. Dowek. *Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo.* TLCA, LNCS 4583:102-117, 2007.

[4] R. Harper and R. Pollack. *Type Checking with Universes.* TCS 89:107-136, 1991.

[5] L. van Benthem Jutting, J. McKinna and R. Pollack. *Checking Algorithms for Pure Type Systems.* Types, LNCS 806:19-61, 1993.