Encoding Type Universes Without Using Matching Modulo Associativity and Commutativity

Frédéric Blanqui
Université Paris-Saclay, INRIA, ENS Paris-Saclay, CNRS, Laboratoire Méthodes Formelles, 4 avenue des Sciences 91190 Gif-sur-Yvette, France

Abstract

The encoding of proof systems and type theories in logical frameworks is key to allow the translation of proofs from one system to the other. The $\lambda\Pi$-calculus modulo rewriting is a powerful logical framework in which various systems have already been encoded, including type systems with an infinite hierarchy of type universes equipped with a unary successor operator and a binary max operator: Matita, Coq, Agda and Lean. However, to decide the word problem in this max-successor algebra, all the encodings proposed so far use rewriting with matching modulo associativity and commutativity (AC), which is of high complexity and difficult to integrate in usual algorithms for β-reduction and type-checking. In this paper, we show that we do not need matching modulo AC by enforcing terms to be in some special canonical form wrt associativity and commutativity, and by using rewriting rules taking advantage of this canonical form. This work has been implemented in the proof assistant Lambdapi.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Type theory; Theory of computation → Equational logic and rewriting

Keywords and phrases logical framework, type theory, type universes, rewriting

Digital Object Identifier 10.4230/LIPIcs.FSCD.2022.24

Supplementary Material *Software (Source Code)*: `https://github.com/Deducteam/lambdapi/blob/master/src/core/term.ml`
archived at `swh:1:cnt:4fe22b374435a880877522ae26106c089df55178`

Acknowledgements The author thanks Thiago Felicissimo for his testing and remarks on the implementation of the present work in `https://github.com/Deducteam/lambdapi`, Guillaume Genestier for his careful reading of a first version of this paper, Gaspard Férey for his remarks on a first version of this paper, as well as the anonymous reviewers for their suggestions.

1 Introduction

The complete formalization of important mathematical theorems or software is possible but still very costly in terms of time and expertise (seL4, compcert, odd-order theorem, etc.). Moreover, all these certifications are specific to a given prover, and rely on its implementation and maintenance. And it is currently very difficult to automatically translate developments done in one system to another system, especially if those systems are based on different, and possibly incompatible, foundations. Hence, there is a lot of work duplication, and it gets more and more difficult for new proof systems to emerge as the development of standard libraries is time-consuming and not very rewarding.

Logical frameworks. A way to improve this situation is to encode the axioms and rules of proof systems into a common language, called a logical framework, so that a feature (e.g. polymorphism) that is common to two different systems is encoded by the same construction [10]. Using a logical framework for n systems allows one to reduce the number of translators necessary to translate each system to all the others from $\mathcal{O}(n^2)$ to $\mathcal{O}(2n)$.

The $\lambda\Pi$-calculus modulo rewriting, $\lambda\Pi/\mathcal{R}$, is a good candidate for such a logical framework [10]. In [14] already, Cousineau and Dowek proved that any functional pure type system (PTS) [7] can be encoded in $\lambda\Pi/\mathcal{R}$. Then, several other systems have been encoded: higher-order logic and the OpenTheory format used by HOL-Light and HOL4, the calculus of inductive constructions and the proof systems of Matita [5], Coq [17] and Agda [19].

$\lambda\Pi/\mathcal{R}$ extends the logical framework LF [22] by allowing the definition of function symbols and types by a set $\mathcal{R}$ of rewriting rules [34]. LF itself extends Church's simply-typed λ-calculus with dependent types, that is, object-indexed type families. Given a type A, written $A : \text{Type}$, the product of an A-indexed family of types $(B(x))_{x\in A}$ is written $\Pi x : A, B(x)$, and simply $A \to B$ if $B(x)$ does not depend on x. In LF, types equivalent modulo β-conversion are identified while, in $\lambda\Pi/\mathcal{R}$, types equivalent modulo $\beta\mathcal{R}$-conversion are identified.

For the type conversion and type-checking of $\lambda\Pi/\mathcal{R}$ to be decidable, one usually requires the rewrite relation generated by β-reduction and rewrite rules, $\longrightarrow_\beta \cup \longrightarrow_\mathcal{R}$, to preserve typing, be confluent (the order of reductions does not matter) and terminating (there is no infinite rewrite sequence), and various criteria have been developed to check those properties (see for instance [9, 19, 17]).

Type universes are a way to reify types, that is, to see types as objects [28], which allows one to express polymorphism (quantification over types) and build models of type theory in type theory, like in set theory inaccessible cardinals allow one to build models of ZF. By iterating this process, we get an ω-indexed sequence of type-theoretic universes $U_0, U_1, \ldots$ with each one being an element of the next one, usually written $U_i : U_{i+1}$ in type theory. However, to keep the system consistent, some care must be taken when defining universe constructors. For instance, if $A : U_i$ and, for all $x : A$, $B(x) : U_j$, then, with predicative universes, we must have $(\Pi x : A, B(x)) : U_{\max(i,j)}$.

Following [14], one can easily encode such an infinite hierarchy of type universes in $\lambda\Pi/\mathcal{R}$, by using the following $\lambda\Pi/\mathcal{R}$ infinite signature and set of rules:[1]

- for each universe U_i, the symbols $U_i : \text{Type}$ and $T_i : U_i \to \text{Type}$;
- for each axiom $U_i : U_{i+1}$, the symbol $u_i : U_{i+1}$ and the rewrite rule $T_{i+1} u_i \longrightarrow U_i$;
- for each product from U_i to U_j, the symbol $\pi_{i,j} : \Pi x : U_i, (T_i\, x \to U_j) \to U_{\max(i,j)}$ and the rewrite rule $T_{\max(i,j)}(\pi_{i,j}\, x\, y) \longrightarrow \Pi z : T_i\, x, T_j(y\, z)$.

To get a finite signature, one can represent type universes in Peano arithmetic using the following algebra [5]:

▶ **Definition 1** (Max-successor algebra). *The max-successor algebra $\mathcal{L}$ is the first-order term algebra made of the symbols* $\mathtt{z}$ *of arity 0,* $\mathtt{s}$ *of arity 1 and $\sqcup$ of arity 2, written infix. We moreover take $\sqcup$ of smaller priority than* $\mathtt{s}$ *so that $\mathtt{s}x \sqcup y$ is the same as $(\mathtt{s}x) \sqcup y$. Then, let $\mathcal{C} = \mathcal{V} \cup \{\mathtt{z}\}$ where $\mathcal{V}$ some set of variables disjoint from function symbols.*

The interpretation of a term t wrt a valuation $\mu : \mathcal{V} \to \mathbb{N}$ is as expected:

- $\mathtt{z}$ *is interpretated as 0: $\mathtt{z}\mu = 0$,*
- $\mathtt{s}$ *is interpreted as the successor function: $(\mathtt{s}\, t)\mu = t\mu + 1$,*
- $\sqcup$ *is interpreted as the binary max function on $\mathbb{N}$: $(u \sqcup v)\mu = \max(u\mu, v\mu)$.*

Two terms t, u are equivalent, written $t \simeq u$, if, for all valuations μ, $t\mu = u\mu$.

In the following, we will denote by $\simeq_A$ the equational sub-theory of $\simeq$ generated by the equation $(t \sqcup u) \sqcup v = t \sqcup (u \sqcup v)$, and by $\simeq_{AC}$ the equational sub-theory of $\simeq$ generated by the equations $u \sqcup v = v \sqcup u$ and $(t \sqcup u) \sqcup v = t \sqcup (u \sqcup v)$.

[1] In $\lambda\Pi/\mathcal{R}$, rules are sometimes presented as part of the signature [13, 30].

By using this algebra, we can then encode in $\lambda\Pi/\mathcal{R}$ a type system with an infinite hierarchy of type universes using the following *finite* signature:

- the symbols $\mathcal{L}$: Type, $\mathtt{z} : \mathcal{L}$, $\mathtt{s} : \mathcal{L} \to \mathcal{L}$, $\sqcup : \mathcal{L} \to \mathcal{L} \to \mathcal{L}$ and the rules $\mathtt{z} \sqcup y \longrightarrow y$, $x \sqcup \mathtt{z} \longrightarrow x$, $(\mathtt{s}\,x) \sqcup (\mathtt{s}\,y) \longrightarrow \mathtt{s}\,(x \sqcup y)$;
- the symbols $U : \mathcal{L} \to$ Type and $T : \Pi i : \mathcal{L}, U\,i \to$ Type;
- the symbol $u : \Pi i : \mathcal{L}, U\,(\mathtt{s}\,i)$ and the rewrite rule $T\,\_\,(u\,i) \longrightarrow U\,i$;
- the symbol $\pi : \Pi i : \mathcal{L}, \Pi j : \mathcal{L}, \Pi x : U\,i, (T\,i\,x \to U\,j) \to U\,(i \sqcup j)$ and the rewrite rule $T\,\_\,(\pi\,i\,j\,x\,y) \longrightarrow \Pi z : T\,i\,x, T\,j\,(y\,z)$.

The rules defining $\sqcup$ are indeed sufficient to decide whether $t \simeq u$ when t and u are closed terms (i.e. terms with no variables), which is necessary for deciding the type conversion relation of $\lambda\Pi/\mathcal{R}$.

Universe variables. This representation with universe variables is also useful to represent systems with floating/elastic universes or universe polymorphism like in Coq or Agda [33, 32, 2]. However, in this case, the rules defining $\sqcup$ do not allow one to decide $\simeq$ on open terms (i.e. terms with variables), even if one adds the associativity and commutativity of $\sqcup$ in the type conversion because, for instance, $x \sqcup x = x$ ($\sqcup$ is idempotent), $x \sqcup \mathtt{s}x = \mathtt{s}x$, $x \sqcup \mathtt{s}(\mathtt{s}x) = \mathtt{s}(\mathtt{s}x)$, ...

The relation $\simeq$ on open terms is decidable though since it is a sub-theory of Presburger arithmetic [29]. So, one solution could be to use as logical framework not $\lambda\Pi/\mathcal{R}$ but an extension of LF with decision procedures, like CoqMT [8]. But the translation from such a logical framework to HOL-Light, Coq, Agda, etc. would be more difficult or introduce undesirable axioms in the target system.

In [6], Assaf and his coauthors introduced a presentation of the max-successor algebra to deal with universe variables. They replaced the successor symbol $\mathtt{s}$ by two new symbols: $\mathtt{1}$ of arity 0, and $+$ of arity 2. However, they had to use rewriting with matching modulo associativity and commutativity (AC) of $\sqcup$, and associativity, commutativity and unit (ACU) of $+$ (as $\mathtt{z}$ is a neutral element of $+$), and extend type conversion with those theories too. But matching modulo AC or ACU is NP-complete [24, 25].

Finally, in [20], Genestier introduced another presentation of the max-successor algebra that can be decided by using $\simeq_{AC}$ and matching modulo $\simeq_{AC}$ only (more details will be given in Section 3).

However, efficient implementations of matching modulo AC or AC-equivalence rely on data structures for representing terms that are different from the ones used for implementing β-reduction and type-checking in dependent type systems [4, 35, 12, 1]. For instance, in [15, 16], an AC symbol f is considered as varyadic (i.e. can take any number of arguments) and terms are "flattened" so that f has no argument headed by f. The addition of AC-matching and AC-equivalence in a type-checker for $\lambda\Pi/\mathcal{R}$ can therefore introduce inefficiencies and bugs, and greatly increase the size of the code. For instance, the addition of AC-matching and AC-equivalence in Dedukti doubled the size of the code[2].

We can therefore wonder whether there is another way to handle universe variables that is easier to implement in a type-checker for $\lambda\Pi/\mathcal{R}$.

[2] See `https://github.com/Deducteam/Dedukti/pull/219`.

Outline. In this paper, we give yet another presentation of the max-successor algebra together with a new convergent rewrite system for deciding it that does not use matching modulo AC. This can be achieved by keeping terms in some AC canonical form, following a technique introduced in [11].

We start by giving a direct proof of decidability of the word problem in the max-successor algebra. This will allow us to introduce some notions, like the one of canonical form, that is at the basis of our new presentation. For the sake of completeness, we then recall Genestier's rewrite system with matching modulo AC. Then, in Section 4, we give a new presentation of the max-successor algebra and a convergent rewrite system for deciding the equivalence of two AC-canonical terms of a shape ensured by our translation. Finally, in Section 5, we explain how to modify the code of a $\lambda\Pi/\mathcal{R}$ type-checker to ensure that every term can always be in AC-canonical form. This work has been implemented in the proof assistant Lambdapi and the code is freely accessible on `https://github.com/Deducteam/lambdapi`.

2 Word problem in the max-successor algebra

We first give a direct proof of decidability of $\simeq$ by recalling the notion of canonical form for the max-successor algebra introduced by Genestier in [20], by showing that two equivalent terms have equal canonical forms, and by providing a recursive functional program for computing the canonical form of a term. To this end, we reuse a terminology that is common in the study of hetegeneous signatures [18, 21]:

▶ **Definition 2** (Aliens, combs and caps). *Given a binary symbol f, let* $\mathrm{aliens}_f : \mathcal{L} \to \mathcal{L}^+$ *be the function mapping every term to a non-empty list of terms such that* $\mathrm{aliens}_f(t) = \mathrm{aliens}_f(u)\mathrm{aliens}_f(v)$ *(the list concatenation being written by juxtaposition) if $t = fuv$, and* $\mathrm{aliens}_f(t) = t$ *(the singleton list) otherwise.*

Conversely, let $\mathrm{comb}_f : \mathcal{L}^+ \to \mathcal{L}$ *be the function mapping a non-empty list of terms to a term such that* $\mathrm{comb}_f[t] = t$ *and, for all $n \geq 2$,* $\mathrm{comb}_f[t_1, \dots, t_n] = ft_1\mathrm{comb}_f[t_2, \dots, t_n]$.

Let an f-context be a term whose symbols are f or a distinguished variable $\square$. Given an f-context C with n occurrences of $\square$ at the respective (disjoint) positions[3] *$p_1 < \dots < p_n$ (ordered lexicographically*[4]*), and n terms $t_1, \dots, t_n$, let $C[t_1, \dots, t_n]$ be the term obtained by replacing the occurrence of $\square$ at position p_i by t_i for every i.*

Given a term t, let $\mathrm{cap}_f(t)$ *be the (unique) biggest f-context C such that* $t = C[\mathrm{aliens}_f(t)]$.

▶ **Example.** $\mathrm{aliens}_\sqcup((x \sqcup y) \sqcup z) = [x; y; z]$, $\mathrm{comb}_\sqcup[x; y; z] = x \sqcup (y \sqcup z)$, $\mathrm{cap}_\sqcup((x \sqcup y) \sqcup z) = ((\square \sqcup \square) \sqcup \square)$, $\mathrm{Pos}((x \sqcup y) \sqcup z) = \{\varepsilon, 1, 2, 11, 12\}$, *and* $\mathrm{cap}_\sqcup((x \sqcup y) \sqcup z)[t_1, t_2, t_3] = (t_1 \sqcup t_2) \sqcup t_3$.

▶ **Lemma 3.**

- *For all terms t,* $t \simeq_A \mathrm{comb}_\sqcup(\mathrm{aliens}_\sqcup(t))$.
- *For all sequences of terms l, m and terms t, u,* $\mathrm{comb}_\sqcup(ltum) \simeq_{AC} \mathrm{comb}_\sqcup(lutm)$.
- *For all terms $t_1, \dots, t_n$,* $\mathtt{s}(\mathrm{comb}_\sqcup[t_1, \dots, t_n]) \simeq \mathrm{comb}_\sqcup[\mathtt{s}(t_1), \dots, \mathtt{s}(t_n)]$

Proof.

- By definition, $t = \mathrm{cap}_\sqcup(t)[\mathrm{aliens}_\sqcup(t)]$. Let C be the canonical form of $\mathrm{cap}_\sqcup(t)$ wrt the convergent rewrite system made of the rewrite rule $(x \sqcup y) \sqcup z \to x \sqcup (y \sqcup z)$. We have $\mathrm{cap}_\sqcup(t) \simeq_A C$, $\mathrm{cap}_\sqcup(t)[\mathrm{aliens}_\sqcup(t)] \simeq_A C[\mathrm{aliens}_\sqcup(t)]$ and $C[\mathrm{aliens}_\sqcup(t)] = \mathrm{comb}_\sqcup(\mathrm{aliens}_\sqcup(t))$. Therefore, $t \simeq_A \mathrm{comb}_\sqcup(\mathrm{aliens}_\sqcup(t))$.

[3] The set $\mathrm{Pos}(t)$ of the positions in a term t is defined as usual as words on $\mathbb{N}$: $\mathrm{Pos}(x) = \{\varepsilon\}$ where ε is the empty word, and $\mathrm{Pos}(ft_1 \dots t_n) = \{\varepsilon\} \cup \{ip \mid 1 \leq i \leq n, p \in \mathrm{Pos}(t_i)\}$.

[4] $ip < jq$ if $i < j$ or else $i = j$ and $p < q$.

- By induction on l.
  - Case l empty. If m is empty, $\mathrm{comb}_\sqcup(tu) \simeq_{AC} \mathrm{comb}_\sqcup(ut)$. Otherwise, $\mathrm{comb}_\sqcup(tum) = t \sqcup (u \sqcup \mathrm{comb}_\sqcup(m)) \simeq_{AC} u \sqcup (t \sqcup \mathrm{comb}_\sqcup(m)) = \mathrm{comb}_\sqcup(utm)$.
  - Case $l = al'$. $\mathrm{comb}_\sqcup(ltum) = a \sqcup \mathrm{comb}_\sqcup(l'tum)$. By induction hypothesis, $\mathrm{comb}_\sqcup(l'tum) \simeq_{AC} \mathrm{comb}_\sqcup(l'utm)$. Therefore, $\mathrm{comb}_\sqcup(ltum) \simeq_{AC} \mathrm{comb}_\sqcup(lutm)$.
- First note that, for all x and y, $\mathtt{s}(x \sqcup y) \simeq (\mathtt{s}x) \sqcup (\mathtt{s}y)$. We then proceed by induction on n. If $n = 1$, this is immediate since $\mathrm{comb}_\sqcup[t] = t$. If $n \geq 2$, $\mathtt{s}(\mathrm{comb}_\sqcup[t_1, \dots, t_n]) = \mathtt{s}(t_1 \sqcup \mathrm{comb}_\sqcup[t_2, \dots, t_n]) \simeq (\mathtt{s}t_1) \sqcup (\mathtt{s}(\mathrm{comb}_\sqcup[t_2, \dots, t_n]))$. By induction hypothesis, $\mathtt{s}(\mathrm{comb}_\sqcup[t_2, \dots, t_n]) \simeq \mathrm{comb}_\sqcup[\mathtt{s}(t_2), \dots, \mathtt{s}(t_n)]$. Therefore, $\mathtt{s}(\mathrm{comb}_\sqcup[t_1, \dots, t_n]) \simeq \mathrm{comb}_\sqcup[\mathtt{s}(t_1), \dots, \mathtt{s}(t_n)]$. ◀

▶ **Definition 4** (s-terms, S-function and total order on s-terms)**.** *A term is an* s*-term if it contains no $\sqcup$ symbol.*

For all s*-terms t, there is a unique pair $(k, x) \in \mathbb{N} \times \mathcal{C}$ such that $t = S\,k\,x$, where $S : \mathbb{N} \to \mathcal{L} \to \mathcal{L}$ is the (meta-level) function such that $S\,0\,t = t$ and, for all $n \geq 1$, $S\,n\,t = S\,(n-1)\,(\mathtt{s}\,t)$.*

Assuming that $\mathcal{C}$ is totally ordered, we define a total order on s*-terms by taking $Spx \leq Sqy$ iff $x \leq y$ or else $x = y$ and $p \leq q$.*

▶ **Definition 5** (Canonical forms)**.** *A term $t \in \mathcal{L}$ is in canonical form if:*
- $t = \mathrm{comb}_\sqcup[\mathrm{aliens}_\sqcup(t)]$,
- $\mathrm{aliens}_\sqcup(t)$ *is a strictly increasing list of* s*-terms (in the order of Definition 4),*
- *t is linear (every variable occurs at most once),*
- *if $S\,k\,\mathtt{z}$ and $S\,l\,x$ are aliens of t then $k > l$.*

▶ **Lemma 6.**
- *Two equivalent canonical forms are equal.*
- *Every term is equivalent to a canonical form.*

Proof.
- Let t and u be two equivalent canonical forms. t and u have the same variables $x_1, \dots, x_n$ since, otherwise, they could not have the same interpretation for all valuations. Let $Sk_1x_1, \dots, Sk_nx_n$ be the aliens of t not of the form $Sk\mathtt{z}$, and $Sl_1x_1, \dots, Sl_nx_n$ be the aliens of u not of the form $Sk\mathtt{z}$.
  Assume that t has an alien of the form $Sk_0\mathtt{z}$ and u has no alien of the form $Sk\mathtt{z}$. Then, $n > 0$ and $0 \leq k_n < k_0$. But, by taking $x_i\mu = 0$ for all i, we get $t\mu = k_0$ and $u\mu = 0$, which is not possible since $t \simeq u$.
  Assume that t has an alien of the form $Sk_0\mathtt{z}$ and u has an alien of the form $Sl_0\mathtt{z}$. By taking $x_i\mu = 0$ for all i, we get $t\mu = k_0$ and $u\mu = l_0$. Therefore, $k_0 = l_0$.
  Let now $M = \max(\{k_i | 1 \leq i \leq n\} \cup \{l_i | 1 \leq i \leq n\})$, $N = \max(M, k)$ if t and u have an alien of the form $Sk\mathtt{z}$, and $N = M$ otherwise. For all $i \geq 1$, let μ_i be the valuation mapping x_i to N and all other variables to 0. Then, $t\mu = N + k_i$ and $u\mu = N + l_i$. Therefore, $k_i = l_i$ for all i, and $t = u$.
- We prove that, for all terms t, there is a canonical form t' such that $t \simeq t'$, by induction on the size of t.
  - Case t is a variable or $\mathtt{z}$. This is immediate since t is in canonical form.
  - Case $t = \mathtt{s}u$. By induction hypothesis, $u \simeq u'$ in canonical form. Let $[u_1, \dots, u_n]$ be the aliens of u'. We have $t \simeq \mathtt{s}u' = \mathtt{s}(\mathrm{comb}_\sqcup[u_1, \dots, u_n]) \simeq \mathrm{comb}_\sqcup[\mathtt{s}(u_1), \dots, \mathtt{s}(u_n)]$. $[\mathtt{s}(u_1), \dots, \mathtt{s}(u_n)]$ is a strictly increasing list of $\mathtt{s}$-terms. Moreover, if $Sk\mathtt{z}$ and Slx are elements of this list, then $k > l$. Therefore, $\mathrm{comb}_\sqcup[\mathtt{s}(u_1), \dots, \mathtt{s}(u_n)]$ is a canonical form.

- Case $t = u \sqcup v$. By induction hypothesis, $u \simeq u'$ in canonical form, and $v \simeq v'$ in canonical form. Given a list of $\mathtt{s}$-terms, let $\mathrm{sort}(l)$ be the function putting the elements of l in increasing order. We have $\mathrm{comb}_\sqcup(l) \simeq_{AC} \mathrm{comb}_\sqcup(\mathrm{sort}(l))$. Given an increasing list of $\mathtt{s}$-terms, let $\mathrm{merge}(l)$ be the function that, starting from l:
  - replaces any two (adjacent) terms Spx, Sqx by the single term $S(p \sqcup_{\mathbb{N}} q)x$,
  - removes any term $Sp\mathtt{z}$ if there is also some term Sqx with $p \leq q$.

  We have $\mathrm{comb}_\sqcup(l) \simeq \mathrm{comb}_\sqcup(\mathrm{merge}(l))$ since $Spx \sqcup Sqx \simeq S(p \sqcup_{\mathbb{N}} q)x$ and $Sp\mathtt{z} \sqcup Sqx \simeq Sqx$ if $p \leq q$. Let now $l = \mathrm{aliens}_\sqcup(u')$ and $m = \mathrm{aliens}_\sqcup(v')$. Then, $t \simeq u' \sqcup v' = \mathrm{comb}_\sqcup(\mathrm{aliens}_\sqcup(u' \sqcup v')) = \mathrm{comb}_\sqcup(lm) \simeq_{AC} \mathrm{comb}_\sqcup(\mathrm{sort}(lm)) \simeq \mathrm{comb}_\sqcup(\mathrm{merge}(\mathrm{sort}(lm)))$, which is in canonical form. ◀

It follows that, for checking whether $t \simeq u$, it suffices to compute and syntactically compare the canonical forms of t and u. This could be easily done in any programming language. However, we are interested in implementing this in the logical framework $\lambda\Pi/\mathcal{R}$ and its implementation Lambdapi, which allows one to define functions by using rewriting rules with syntactic matching only. However, before showing that this can indeed be done, we are first going to see a solution using rewriting with matching modulo AC proposed in [20] and implemented in Dedukti thanks to the addition of matching modulo AC in Dedukti by Gaspard Férey (see p. 92 in [17]).

3 Decision procedure using matching modulo AC

In this section, we recall the rewriting system using matching modulo AC proposed by Genestier in [20] for deciding $\simeq$. The idea is to represent the terms of $\mathcal{L}$ as the maximum of a natural number and of a finite set of expressions corresponding to the terms $S\,l\,x$ with x a variable. To do so, Genestier uses a multi-sorted term algebra with three sorts:[5]

- The sort $\mathtt{N}$ with the constructors $0 : \mathtt{N}$, $\mathtt{s} : \mathtt{N} \to \mathtt{N}$, $+ : \mathtt{N} \times \mathtt{N} \to \mathtt{N}$ and $\oplus : \mathtt{N} \times \mathtt{N} \to \mathtt{N}$ written infix, with $\oplus$ of priority smaller than $\mathtt{s}$, to represent arithmetic expressions on natural numbers. The sort $\mathtt{N}$ is interpreted as $\mathbb{N}$, 0 as 0, $\mathtt{s}$ as the successor function, $+$ as the addition, and $\oplus$ as the maximum.
- The sort $\mathtt{E}$ with the constructors $\emptyset : \mathtt{E}$, $\mathtt{a} : \mathtt{N} \times \mathtt{L} \to \mathtt{E}$, $\cup : \mathtt{E} \times \mathtt{E} \to \mathtt{E}$ written infix, and $\mathtt{A} : \mathtt{N} \times \mathtt{E} \to \mathtt{E}$, to represent the maximum of a finite set of arithmetic expressions. The sort $\mathtt{E}$ is interpreted as $\mathbb{N} \cup \{-\infty\}$, $\emptyset$ as $-\infty$, $\mathtt{a}$ and $\mathtt{A}$ as the addition with $x + (-\infty) = -\infty$, and $\cup$ as the maximum. $\mathtt{a}\,k\,t$ represents the singleton set $\{k + t\}$, and the auxiliary function $\mathtt{A}\,k\,E$ (called $\mathtt{mapPlus}$ in [20]) adds k to every element of E.
- The sort $\mathtt{L}$ with the constructor $\mathtt{m} : \mathtt{N} \times \mathtt{E} \to \mathtt{L}$. The sorts $\mathtt{L}$ is interpreted as $\mathbb{N}$, and $\mathtt{m}$ as the maximum.

A term of $\mathcal{L}$ is translated to a term of sort $\mathtt{L}$ with the same interpretation as follows:

- $|x| = x$,
- $|\mathtt{z}| = \mathtt{m}\,0\,\emptyset$,
- $|\mathtt{s}\,t| = \mathtt{m}\,(\mathtt{s}\,0)\,(\mathtt{a}\,(\mathtt{s}\,0)\,|t|)$
- $|u \sqcup v| = \mathtt{m}\,0\,((\mathtt{a}\,0\,|u|) \cup (\mathtt{a}\,0\,|v|))$

Then, Genestier introduces a rewrite system, that we will call $\mathcal{G}$, made of the rewrite rules of Figure 1 and of the rewrite rules of Figure 2. The second rule for $\cup$ corresponds to the equation $(p + x) \oplus (q + x) = (p \oplus q) + x$. It allows one to have at most one occurrence of

[5] In [20], $\sqcup$ is denoted by $\mathtt{max}$, $\mathtt{E}$ by $\mathtt{LSet}$, $\mathtt{a}$ by $\oplus$, $\mathtt{A}$ by $\mathtt{mapPlus}$, and $\mathtt{m}$ by $\mathtt{Max}$.

$$\begin{array}{rcl} 0+q & \longrightarrow & q \\ \mathtt{s}\,p+q & \longrightarrow & \mathtt{s}\,(p+q) \\[4pt] p\oplus 0 & \longrightarrow & p \\ 0\oplus q & \longrightarrow & q \\ \mathtt{s}\,p\oplus\mathtt{s}\,q & \longrightarrow & \mathtt{s}\,(p\oplus q) \end{array}$$

Figure 1 Rewrite rules for addition and maximum on natural numbers.

$$\begin{array}{rcl} X\cup\emptyset & \longrightarrow & X \\ (\mathtt{a}\,p\,x)\cup(\mathtt{a}\,q\,x) & \longrightarrow & \mathtt{a}\,(p\oplus q)\,x \\[4pt] \mathtt{A}\,p\,\emptyset & \longrightarrow & \emptyset \\ \mathtt{A}\,p\,(\mathtt{a}\,q\,x) & \longrightarrow & \mathtt{a}\,(p+q)\,x \\ \mathtt{A}\,p\,(X\cup Y) & \longrightarrow & (\mathtt{A}\,p\,X)\cup(\mathtt{A}\,p\,Y) \\[4pt] \mathtt{m}\,0\,(\mathtt{a}\,0\,x) & \longrightarrow & x \\ \mathtt{m}\,p\,(\mathtt{a}\,q\,(\mathtt{m}\,r\,X)) & \longrightarrow & \mathtt{m}\,(p\oplus(q+r))\,(\mathtt{A}\,q\,X) \\ \mathtt{m}\,p\,((\mathtt{a}\,q\,(\mathtt{m}\,r\,X))\cup Y) & \longrightarrow & \mathtt{m}\,(p\oplus(q+r))\,((\mathtt{A}\,q\,X)\cup Y) \end{array}$$

Figure 2 The system $\mathcal{G}$ for computing canonical forms with matching modulo AC includes the above rules as well as the rules of Figure 1.

every variable x. The second rule of $\mathtt{A}$ corresponds to the equation $p+(q+x)=(p+q)+x$, while the third rule of $\mathtt{A}$ corresponds to the equation $p+(x\oplus y)=(p+x)\oplus(p+y)$. The rules of $\mathtt{m}$ are the main rules for computing the canonical form. The first rule corresponds to the equation $0\oplus(0+x)=x$. The second rule corresponds to the equation $p\oplus(q+(r\oplus(k_1+x_1)\oplus\ldots\oplus(k_n+x_n)))=(p\oplus(q+r))\oplus(q+k_1+x_1)\oplus\ldots(q+k_n+x_n)$. The last rule is similar.

Genestier then proves the following properties:

- The rewrite relation $\longrightarrow_{\mathcal{G},AC}$ generated by $\mathcal{G}$ using matching modulo associativity and commutativity of $\cup$ is not confluent on terms with variables of sort $\mathtt{N}$ or $\mathtt{E}$.
- For all terms t in $\mathcal{L}$, any $\longrightarrow_{\mathcal{G},AC}$-normal form of $|t|$ is either a variable or of the form $\mathtt{m}\,p\,((\mathtt{a}\,q_1\,x_1)\cup\ldots\cup(\mathtt{a}\,q_n\,x_n))$ with $x_1,\ldots,x_n$ distinct variables and, for all k, $q_k\le p$.
- Two such normal forms are equal modulo associativity-commutativity of $\cup$.

To these results, we can add:

▶ **Lemma 7.** *The relation $\longrightarrow_{\mathcal{G}/AC}=\simeq_{AC}\longrightarrow_{\mathcal{G}}\simeq_{AC}$ generated by $\mathcal{G}$ on AC-equivalence classes, which contains $\longrightarrow_{\mathcal{G},AC}$, terminates.*

Proof. It can be automatically proved by, for instance AProVE [3], using 3 consecutive strictly monotone polynomial interpretations on $\mathbb{N}$, and then formally certified in Isabelle/HOL by CeTA[6]. ◀

[6] `http://cl-informatik.uibk.ac.at/software/ceta/`

4 Getting rid of matching modulo AC

In this section, we present our main contribution: a new presentation of $\mathcal{L}$ and a new rewrite system not using matching modulo AC. It is inspired by the decidability proof of Section 2.

The main problem for computing the canonical form of a term is to be able to replace an expression of the form $Spx \sqcup (Sry \sqcup Sqx)$ by $S(p \oplus q)x \sqcup Sry$. One way to do it is by using the rule (4) of Figure 3 with matching modulo AC. Indeed, we have $Spx \sqcup (Sry \sqcup Sqx) \simeq_{AC} Spx \sqcup (Sqx \sqcup Sry) \longrightarrow_{\mathcal{R}} S(p \oplus q)x \sqcup Sry$. Another way to do it is to make sure that the aliens of a term are always ordered so that two aliens Spx and Sqx sharing the same variable x are always put side by side. Following [11], this can be achieved by replacing constructors by construction functions, that is here, $\sqcup$ by some new function symbol $\sqcup'$ which will rearrange its aliens so as to get such an AC-canonical form. Hence, we get $Spx \sqcup' (Sry \sqcup' Sqx) \simeq_{AC} Spx \sqcup (Sqx \sqcup Sry) \longrightarrow_{\mathcal{R}} S(p \oplus q)x \sqcup Sry$.

Again, we translate terms of $\mathcal{L}$ into a multi-sorted term algebra. However, our algebra is simpler than Genestier's algebra. Like [20], we distinguish expressions representing natural numbers from the other expressions by using distinct sorts. However, we do not introduce a new sort for sets but simply extend $\mathcal{L}$-terms with a new symbol $\mathtt{S}$ corresponding to the (meta-level) function S of Definition 4.

We consider the multi-sorted term algebra $\mathcal{I}$ with two sorts $\mathtt{N}$ and $\mathtt{L}$, and the constructors $0 : \mathtt{N}$, $\mathtt{s} : \mathtt{N} \to \mathtt{N}$, $+ : \mathtt{N} \times \mathtt{N} \to \mathtt{N}$ and $\oplus : \mathtt{N} \times \mathtt{N} \to \mathtt{N}$ written infix, $\mathtt{z} : \mathtt{L}$, $\mathtt{S} : \mathtt{N} \times \mathtt{L} \to \mathtt{L}$ and $\sqcup : \mathtt{L} \to \mathtt{L} \to \mathtt{L}$. Again, we assume that $\oplus$ is of priority smaller than $\mathtt{s}$. All the sorts are interpreted as $\mathbb{N}$, 0 as 0, $\mathtt{s}$ as the successor function, $+$ and $\mathtt{S}$ as the addition, and $\oplus$ as the maximum.

▶ **Definition 8** (Guarded terms). *An $\mathcal{I}$-term is guarded if every occurrence of an element $x \in \mathcal{C}$ of sort* L *is in a subterm of the form $\mathtt{S}\,p\,x$.*

The idea behind guarded terms is to represent an $\mathcal{L}$-term of the form $S\,k\,x$ by the $\mathcal{I}$-term $\mathtt{S}\,\overline{k}\,x$, where $\overline{k}$ is the representation of k in $\mathtt{N}$.

An $\mathcal{L}$-term is translated into a guarded $\mathcal{I}$-term of sort $\mathtt{L}$ with the same interpretation in $\mathbb{N}$ as follows:

- $|x| = \mathtt{S}\,0\,x \sqcup \mathtt{S}\,0\,\mathtt{z}$
- $|\mathtt{z}| = \mathtt{S}\,0\,\mathtt{z}$
- $|\mathtt{s}\,t| = \mathtt{S}\,(\mathtt{s}\,0)\,|t|$
- $|u \sqcup v| = |u| \sqcup |v|$

For each occurrence of a variable, we add an occurrence of $\mathtt{z}$ so that, after normalization (see below), we get a term of the form $\mathtt{S}\,p_1\,x_1 \sqcup \ldots \sqcup \mathtt{S}\,p_n\,x_n \sqcup \mathtt{S}\,q\,\mathtt{z}$ with $p_i \leq q$.

▶ **Definition 9** (AC-canonical forms). *Let $\leq$ be any total order on $\mathcal{I}$-terms such that $\mathtt{S}\,p\,x \leq \mathtt{S}\,q\,y$ iff $x < y$ or else $x = y$ and $p \leq q$.*[7]

An $\mathcal{I}$-term t is in AC-canonical form if $t = \mathrm{comb}_{\sqcup}[\mathrm{sort}(\mathrm{aliens}_{\sqcup}(t))]$ and every element of $\mathrm{aliens}_{\sqcup}(t) - \{t\}$ is in AC-canonical form, where $\mathrm{sort}(l)$ is the elements of l in increasing order wrt $\leq$.

Let $\twoheadrightarrow^{AC}$ be the relation mapping every term t to its unique AC-canonical form $[t]$.

Two terms are AC-equivalent iff their AC-canonical forms are equal.

[7] Take for instance the lexicographic path ordering generated by any total precedence on function symbols and variables, and right-to-left comparison of the arguments of $\mathtt{S}$.

Note that AC-canonization is a canonizer in the sense of Shostak [31]. It satisfies the properties (CAN-1) to (CAN-5) explicited in [26]: (CAN-1) it is idempotent; (CAN-2) it decides $\simeq_{AC}$; (CAN-3) it preserves variables; (CAN-4) every subterm of a canonical term is canonical; and (CAN-5) it commutes with order-preserving variable renamings.

We now introduce the rewrite relation that we will use to decide $\simeq$:

▶ **Definition 10** (Rewriting modulo AC-canonization). *Let $\longrightarrow_{\mathcal{R}}^{AC} = \longrightarrow_{\mathcal{R}}\twoheadrightarrow^{AC}$, where $\mathcal{R}$ is made of the rewrite rules of Figures 1 and 3.*

An $\longrightarrow_{\mathcal{R}}^{AC}$ step is a standard $\longrightarrow_{\mathcal{R}}$ step with syntactic matching followed by AC-canonization. We will see in Section 5 that AC-canonization is easily implemented by replacing constructors by construction functions, so that AC-canonization is implicitly done at term construction time [11]. In other words, our decision procedure reduces to standard rewriting with syntactic matching but on a restricted set of terms, namely the terms in AC-canonical form.

This notion of rewriting is close to the notion of normal rewriting [27], which consists in applying a standard rewrite step after normalization wrt a convergent rewrite system $\mathcal{S}$. The difference is that AC-canonization cannot defined by a convergent rewrite system.

One can easily check that the rules of $\mathcal{R}$ preserve guardedness (if t is guarded and $t \longrightarrow_{\mathcal{R}}^{AC} u$, then u is guarded too) and are semantically correct ($\longrightarrow_{\mathcal{R}}^{AC} \subseteq \simeq$). Indeed, the first rule corresponds to the associativity of $+$: $p+(q+x) = (p+q)+x$. The second rule corresponds to the distributivity of $+$ over $\oplus$: $p+(x\oplus y) = (p+x)\oplus(p+y)$. On the contrary, the last two rules factorize identical monoms that are side by side: $(p+x)\oplus(q+x) = (p\oplus q)+x$.

$$\begin{array}{lrcl}
(1) & \mathtt{S}\,p\,(\mathtt{S}\,q\,x) & \longrightarrow & \mathtt{S}\,(p+q)\,x \\
(2) & \mathtt{S}\,p\,(x \sqcup y) & \longrightarrow & \mathtt{S}\,p\,x \sqcup \mathtt{S}\,p\,y \\
(3) & \mathtt{S}\,p\,x \sqcup \mathtt{S}\,q\,x & \longrightarrow & \mathtt{S}\,(p \oplus q)\,x \\
(4) & \mathtt{S}\,p\,x \sqcup (\mathtt{S}\,q\,x \sqcup y) & \longrightarrow & \mathtt{S}\,(p \oplus q)\,x \sqcup y
\end{array}$$

Figure 3 Rewrite system on canonical forms.

We now prove that the relation $\longrightarrow_{\mathcal{R}}^{AC}$ terminates and is confluent on guarded terms with no variables of sort $\mathtt{N}$.

▶ **Lemma 11.** *The relation $\longrightarrow_{\mathcal{R}/AC} = \simeq_{AC}\longrightarrow_{\mathcal{R}}\simeq_{AC}$, which contains $\longrightarrow_{\mathcal{R}}^{AC}$, terminates.*

Proof. AProVE[8] automatically proves the termination of $\longrightarrow_{\mathcal{R}/AC}$ by a succession of 3 strictly monotone polynomial interpretations on $\mathbb{N}$, and its result can be formally checked by CeTA:

- $P_{\mathtt{S}}x_1x_2 = 3 + x_1 + 3x_1x_2 + 3x_2$
- $P_{+}x_1x_2 = x_1 + 2x_1x_2 + x_2$
- $P_{\sqcup}x_1x_2 = 3 + x_1 + x_2$
- $P_{\mathtt{s}}x_1 = x_1$
- $P_{\oplus}x_1x_2 = 1 + x_1 + x_2$
- $P_0 = 1$

validates all the rules as well as the AC axioms of $\sqcup$[9] and strictly orients all the rules except the last rules of $+$ and $\oplus$.

[8] `http://aprove.informatik.rwth-aachen.de/`
[9] A polynomial Pxy validates the AC axioms iff $Pxy = axy + b(x+y) + c$ with $b(b-1) = ac$.

- $P_+ x_1 x_2 = x_1 + x_2$
- $P_\sqcup x_1 x_2 = 3 + 3x_1 + 2x_1x_2 + 3x_2$
- $P_{\mathtt{s}} x_1 = 3 + x_1$
- $P_\oplus x_1 x_2 = 1 + x_1 + 2x_2$

validates all the rules and equations and strictly orients the last rule of $\oplus$.

- $P_+ x_1 x_2 = 3 + 3x_1 + 2x_1x_2 + 2x_2$
- $P_\sqcup x_1 x_2 = 3 + 3x_1 + 2x_1x_2 + 3x_2$
- $P_{\mathtt{s}} x_1 = 3 + 2x_1$

validates all the rules and equations and strictly orients the last rule of $+$. ◀

▶ **Lemma 12.** *The rewrite relation $\longrightarrow_{\mathcal{N}}$ generated by the rules of Figure 1 terminates and is confluent. Moreover, for all closed terms p, q, r of sort N, the following pairs of terms are joinable with $\longrightarrow_{\mathcal{N}}$:*

- $(p + q) + r = p + (q + r)$
- $p + q = q + p$
- $(p \oplus q) \oplus r = p \oplus (q \oplus r)$
- $p \oplus q = q \oplus p$
- $p + (q \oplus r) = (p + q) \oplus (p + r)$

Proof. The relation $\longrightarrow_{\mathcal{N}}$ terminates since it is included in the lexicographic path ordering with $+, \oplus > \mathtt{s}$. It is confluent since it is weakly orthogonal. So, every term of sort N has a unique normal form. Hence, it is sufficient to prove that the above equations are valid in the equational theory generated by $\mathcal{N}$.

A closed term of sort N in normal form wrt $\longrightarrow_{\mathcal{N}}$ cannot contain a subterm of the form $p + q$ or $p \oplus q$ since, otherwise, the smallest such subterm would be reducible by one of the rules of $\mathcal{N}$. Hence, every closed term of sort N in normal form wrt $\longrightarrow_{\mathcal{N}}$ is of the form $Sk0$ with $k \in \mathbb{N}$, where the (meta-level) function S is defined in Definition 4.

It therefore suffices to prove the above equations by using only induction on natural numbers and the rules of $\mathcal{N}$. This can easily be done in Lambdapi for instance. See `https://github.com/fblanqui/lib`. ◀

▶ **Lemma 13.** *$\longrightarrow_{\mathcal{R}}^{AC}$ is locally confluent on AC-canonical guarded terms with no variables of sort N.*

Proof. We show that every critical pair is joinable using $\longrightarrow_{\mathcal{R}}^{AC}$ and Lemma 12. In the following, the terms that are not between square brackets are in AC-canonical form. We also write $[p \oplus q]$ to denote either $p \oplus q$ or $q \oplus p$.

(1) $\mathtt{S}\,p\,(\mathtt{S}\,q\,x) \longrightarrow \mathtt{S}\,(p+q)\,x$ is overlapped by:

(1) By taking $x = \mathtt{S}rx$. We have
$t = \mathtt{S}p(\mathtt{S}q(\mathtt{S}rx)) \longrightarrow_1^{AC} \mathtt{S}(p+q)(\mathtt{S}rx) \longrightarrow_1^{AC} \mathtt{S}((p+q)+r)x$
and $t \longrightarrow_1^{AC} \mathtt{S}p(\mathtt{S}(q+r)x) \longrightarrow_1^{AC} \mathtt{S}(p+(q+r))x$.

(2) By taking $x = x \sqcup y$. We have
$t = \mathtt{S}p(\mathtt{S}q(x \sqcup y)) \longrightarrow_1^{AC} \mathtt{S}(p+q)(x \sqcup y) \longrightarrow_2^{AC} \mathtt{s}(p+q)x \sqcup \mathtt{S}(p+q)y$
and $t \longrightarrow_2^{AC} \mathtt{S}p(\mathtt{S}qx \sqcup \mathtt{S}qy) \longrightarrow_2^{AC} \mathtt{S}p(\mathtt{S}qx) \sqcup \mathtt{S}p(\mathtt{S}qy)$
$\longrightarrow_1^{AC} [\mathtt{S}(p+q)x \sqcup \mathtt{S}p(\mathtt{S}qy)] \longrightarrow_1^{AC} \mathtt{S}(p+q)x \sqcup \mathtt{S}(p+q)y$.

(2) $\mathtt{S}\,p\,(x \sqcup y) \longrightarrow \mathtt{S}\,p\,x \sqcup \mathtt{S}\,p\,y$ is overlapped by:

(3) By taking $x = \mathtt{S}qx$ and $y = \mathtt{S}rx$. We have
$t = \mathtt{S}p(\mathtt{S}qx \sqcup \mathtt{S}rx) \longrightarrow_2^{AC} \mathtt{S}p(\mathtt{S}qx) \sqcup \mathtt{S}p(\mathtt{S}rx) \longrightarrow_1^{AC} [\mathtt{S}(p+q)x \sqcup \mathtt{S}p(\mathtt{S}rx)]$
$\longrightarrow_1^{AC} [\mathtt{S}(p+q)x \sqcup \mathtt{S}(p+r)x] \longrightarrow_3^{AC} \mathtt{S}[(p+q) \oplus (p+r)]$
and $t \longrightarrow_3^{AC} \mathtt{S}p(\mathtt{S}(q \oplus r)x) \longrightarrow_1^{AC} \mathtt{S}(p+(q \oplus r))x$.

(4) By taking $x = \mathtt{S}qx$ and $y = \mathtt{S}rx \sqcup y$. We have
$t = \mathtt{S}p(\mathtt{S}qx \sqcup (\mathtt{S}rx \sqcup y)) \longrightarrow_2^{AC} [\mathtt{S}p(\mathtt{S}qx) \sqcup \mathtt{S}p(\mathtt{S}rx \sqcup y)]$
$\longrightarrow_1^{AC} [\mathtt{S}(p+q)x \sqcup \mathtt{S}p(\mathtt{S}rx \sqcup y)] \longrightarrow_2^{AC} [\mathtt{S}(p+q)x \sqcup (\mathtt{S}p(\mathtt{S}rx) \sqcup \mathtt{S}py)]$
$\longrightarrow_1^{AC} [\mathtt{S}(p+q)x \sqcup (\mathtt{S}(p+r)x \sqcup \mathtt{S}py)] \longrightarrow_4^{AC} [\mathtt{S}[(p+q) \oplus (p+r)]x \sqcup \mathtt{S}py]$
and $t \longrightarrow_4^{AC} [\mathtt{S}p(\mathtt{S}(q \oplus r)x \sqcup y)] \longrightarrow_2^{AC} [\mathtt{S}p(\mathtt{S}(q \oplus r)x) \sqcup \mathtt{S}py]$
$\longrightarrow_1^{AC} [\mathtt{S}(p + (q \oplus r))x \sqcup \mathtt{S}py]$.

(3) $\mathtt{S}\,p\,x \sqcup \mathtt{S}\,q\,x \longrightarrow \mathtt{S}\,(p \oplus q)\,x$ is overlapped by:

(1) By taking $x = \mathtt{S}rx$. We have
$t = \mathtt{S}p(\mathtt{S}rx) \sqcup \mathtt{S}q(\mathtt{S}rx) \longrightarrow_3^{AC} \mathtt{S}(p \oplus q)(\mathtt{S}rx) \longrightarrow_1^{AC} \mathtt{S}((p \oplus q) + r)x$
and $t \longrightarrow_1^{AC} [\mathtt{S}(p+r)x \sqcup \mathtt{S}q(\mathtt{S}rx)] \longrightarrow_1^{AC} [\mathtt{S}(p+r)x \sqcup \mathtt{S}(q+r)x]$
$\longrightarrow_3^{AC} \mathtt{S}[(p+r) \oplus (q+r)]x$.

(2) By taking $x = x \sqcup y$. We have
$t = \mathtt{S}p(x \sqcup y) \sqcup \mathtt{S}q(x \sqcup y) \longrightarrow_3^{AC} \mathtt{S}(p \oplus q)(x \sqcup y) \longrightarrow_2^{AC} [\mathtt{S}(p \oplus q)x \sqcup \mathtt{S}(p \oplus q)y]$
and $t \longrightarrow_2^{AC} [(\mathtt{S}px \sqcup \mathtt{S}py) \sqcup \mathtt{S}q(x \sqcup y)] \longrightarrow_2^{AC} [(\mathtt{S}px \sqcup \mathtt{S}py) \sqcup (\mathtt{S}qx \sqcup \mathtt{S}qy)]$
$\longrightarrow_3^{AC} [\mathtt{S}px \sqcup (\mathtt{S}qx \sqcup \mathtt{S}(p \oplus q)y)] \longrightarrow_4^{AC} [\mathtt{S}(p \oplus q)x \sqcup \mathtt{S}(p \oplus q)y]$.

(4) $\mathtt{S}\,p\,x \sqcup (\mathtt{S}\,q\,x\, \sqcup y) \longrightarrow \mathtt{S}\,(p \oplus q)\,x\, \sqcup y$ is overlapped by:

(1) By taking $x = \mathtt{S}rx$. We have
$t = \mathtt{S}p(\mathtt{S}rx) \sqcup (\mathtt{S}q(\mathtt{S}rx) \sqcup y) \longrightarrow_4^{AC} [\mathtt{S}(p \oplus q)(\mathtt{S}rx) \sqcup y]$
$\longrightarrow_1^{AC} [\mathtt{S}((p \oplus q) + r)x \sqcup y]$
and $t \longrightarrow_1^{AC} [\mathtt{S}(p+r)x \sqcup (\mathtt{S}q(\mathtt{S}rx) \sqcup y)]$
$\longrightarrow_1^{AC} [\mathtt{S}(p+r)x \sqcup (\mathtt{S}(q+r)x \sqcup y)] \longrightarrow_4^{AC} [\mathtt{S}[(p+r) \oplus (q+r)]x \sqcup y]$.

(2) By taking $x = x_1 \sqcup x_2$. We have
$t = \mathtt{S}p(x_1 \sqcup x_2) \sqcup (\mathtt{S}q(x_1 \sqcup x_2) \sqcup y) \longrightarrow_4^{AC} [\mathtt{S}(p \oplus q)(x_1 \sqcup x_2) \sqcup y]$
$\longrightarrow_2^{AC} [\mathtt{S}(p \oplus q)x_1 \sqcup (\mathtt{S}(p \oplus q)x_2 \sqcup y)] = u$
and $t \longrightarrow_2^{AC} [(\mathtt{S}px_1 \sqcup \mathtt{S}px_2) \sqcup (\mathtt{S}q(x_1 \sqcup x_2) \sqcup y)]$
$\longrightarrow_2^{AC} [(\mathtt{S}px_1 \sqcup \mathtt{S}px_2) \sqcup ((\mathtt{S}qx_1 \sqcup \mathtt{S}qx_2) \sqcup y)] = v$.
Since t is guarded, wlog we can assume that
$\mathrm{aliens}_\sqcup(y) = l_1, \mathtt{S}r_1x_1, .., \mathtt{S}r_mx_1, l_2, \mathtt{S}s_1x_2, .., \mathtt{S}s_nx_2, l_3$.
Then, u can be reduced to $\mathrm{comb}_\sqcup[l_1, \mathtt{S}ax_1, l_2, \mathtt{S}bx_2, l_3]$, where
$a = \mathrm{comb}_\oplus[r_1, .., p \oplus q, .., r_m]$ and $b = \mathrm{comb}_\oplus[s_1, .., p \oplus q, .., s_n]$,
by applying $m + n$ times $\longrightarrow_4^{AC}$,
and v can be reduced to $\mathrm{comb}_\sqcup[l_1, \mathtt{S}a'x_1, l_2, \mathtt{S}b'x_2, l_3]$, where
$a' = \mathrm{comb}_\oplus[r_1, .., p, .., q, .., r_m]$ and $b' = \mathrm{comb}_\oplus[s_1, .., p, .., q, .., s_n]$,
by applying $m + n + 2$ times $\longrightarrow_4^{AC}$.

(3) By taking $y = \mathtt{S}rx$. We have
$t = \mathtt{S}px \sqcup (\mathtt{S}qx \sqcup \mathtt{S}rx) \longrightarrow_4^{AC} [\mathtt{S}(p \oplus q)x \sqcup \mathtt{S}rx] \longrightarrow_3^{AC} \mathtt{S}((p \oplus q) \oplus r)x$
and $t \longrightarrow_3^{AC} [\mathtt{S}px \sqcup \mathtt{S}(q \oplus r)x] \longrightarrow_3^{AC} \mathtt{S}(p \oplus (q \oplus r))x$.

(4) By taking $y = \mathtt{S}rx \sqcup y$. We have
$t = \mathtt{S}px \sqcup (\mathtt{S}qx \sqcup (\mathtt{S}rx \sqcup y)) \longrightarrow_4^{AC} [\mathtt{S}(p \oplus q)x \sqcup (\mathtt{S}rx \sqcup y)] = u$
and $t \longrightarrow_4^{AC} [\mathtt{S}px \sqcup (\mathtt{S}(q \oplus r)x \sqcup y)] = v$.
Since t is guarded, wlog we can assume that $\mathrm{aliens}_\sqcup(y) = \mathtt{S}r_1x, .., \mathtt{S}r_mx, l$.
Then, u can be reduced to $\mathrm{comb}_\sqcup[\mathtt{S}ra, l]$, where
$a = \mathrm{comb}_\oplus[r_0, .., p \oplus q, .., r_m]$ and $r_0 = r$, by applying $m + 1$ times $\longrightarrow_4^{AC}$,
and v can be reduced to $\mathrm{comb}_\sqcup[\mathtt{S}a'x, l]$,
where $a' = \mathrm{comb}_\oplus[r_0, .., p, .., q, .., r_m]$, by applying $m + 2$ times $\longrightarrow_4^{AC}$. ◀

Hence, every $\mathcal{L}$-term has, after translation into an $\mathcal{I}$-term, a unique normal form wrt $\longrightarrow_{\mathcal{R}}^{AC}$. We now prove that this normal form is almost a canonical form, and that it is sufficient to decide $\simeq$.

▶ **Lemma 14.** *For all $\mathcal{L}$-terms t and u, we have $t \simeq u$ iff $[|t|]$ and $[|u|]$ have the same normal form wrt $\longrightarrow_{\mathcal{R}}^{AC}$, where $[|t|]$ is the AC-canonical form of the translation of t in $\mathcal{I}$.*

Proof. Wlog we can assume that $x \leq \mathtt{z}$ for all x.

Let $\mathcal{T}$ be the set of $\mathcal{I}$-terms containing $\mathtt{z}$ that are guarded and have no variable of sort $\mathtt{N}$.

First note that every $\mathcal{T}$-term that is in normal form wrt $\longrightarrow_{\mathcal{R}}^{AC}$ is of the form $\mathtt{S}\,p_1\,x_1 \sqcup \ldots \sqcup \mathtt{S}\,p_n\,x_n \sqcup \mathtt{S}\,q\,\mathtt{z}$ with $x_1 < \ldots < x_n < \mathtt{z}$ and $p_i \leq q$ for all i. Hence, the $\longrightarrow_{\mathcal{R}}^{AC}$-normal form of $[|t|]$ is $t' = \mathtt{S}\,p_1\,x_1 \sqcup \ldots \sqcup \mathtt{S}\,p_m\,x_m \sqcup \mathtt{S}\,q\,\mathtt{z}$ with $x_1 < \ldots < x_m < \mathtt{z}$ and $p_i \leq q$, and the $\longrightarrow_{\mathcal{R}}^{AC}$-normal form of $[|u|]$ is $u' = \mathtt{S}\,p'_1\,x'_1 \sqcup \ldots \sqcup \mathtt{S}\,p'_n\,x'_n \sqcup \mathtt{S}\,q'\,\mathtt{z}$ with $x'_1 < \ldots < x'_n < \mathtt{z}$ and $p'_i \leq q'$.

Note also that $t \simeq |t| \simeq [|t|] \simeq t'$, and similarly for u and u'.

Hence, if $t' = u'$ then $t \simeq u$.

Conversely, assume that $t \simeq u$. Then, $t' \simeq u'$, and t' and u' have the same canonical form. But a $\longrightarrow_{\mathcal{R}}^{AC}$-normal form $\mathtt{S}\,p_1\,x_1 \sqcup \ldots \sqcup \mathtt{S}\,p_n\,x_n \sqcup \mathtt{S}\,q\,\mathtt{z}$ with $x_1 < \ldots < x_n < \mathtt{z}$ and $p_i \leq q$ is almost a canonical form: it is a canonical form iff $n = 0$ or $p_n < q$. Moreover, if it is not canonical, then $n > 0$ and $p_n = q$, and its canonical form is $\mathtt{S}\,p_1\,x_1 \sqcup \ldots \sqcup \mathtt{S}\,p_n\,x_n$. So, $m = n$ and, for all i, $p_i = p'_i$ and $x_i = x'_i$. Moreover, since $t' \simeq u'$, we have $p_n < q$ iff $p'_n < q'$. Therefore, $q = q'$ and $t' = u'$. ◀

▶ Remark. The function mapping every $\mathcal{L}$-term t to the unique $\longrightarrow_{\mathcal{R}}^{AC}$ normal form of $[|t|]$ is not a canonizer in the sense of Shostak as it is not an endofunction. On the other hand, the function mapping every term of $\mathcal{T}$ (guarded terms containing $\mathtt{z}$ with no variable of sort $\mathtt{N}$) to its $\longrightarrow_{\mathcal{R}}^{AC}$ normal form is a canonizer in the sense of Shostak as it satisfies the following properties [26]: (CAN-1) it is idempotent; (CAN-2) it decides $\simeq$ on $\mathcal{T}$; (CAN-3) it preserves variables; (CAN-4) every subterm of a canonical term is canonical; and even (CAN-5) canonization commutes with order-preserving variable renamings.

5 Implementation of AC-canonization

To implement AC-canonization in Lambdapi [23], we use an approach introduced in [11]. AC-canonization is done at term construction time. More precisely, we use the mechanism of private data type of OCaml. A private data type is a semi-abstract data type: it is defined as an inductive data type so that users can pattern-match on values of this type but, to build values of this type, one needs to use construction functions. With this mechanism, one can easily enforce some invariant like, here, to have only terms in AC-canonical form. To do so, we only have to replace constructors by construction functions, which is easy and does not require big changes in the code, and implement those construction functions[10]. Moreover, to implement them, we can take advantage of the fact that their arguments are themselves already in AC-canonical form. Finally, note that, by doing so, we get AC-equivalence in the type conversion of Lambdapi for free. On the other hand, we had to slightly adapt the normalization algorithm of Lambdapi [23] to take into account the fact that terms are now put in AC-canonical form after each rewriting step, which may generate new redexes.

[10] See `https://github.com/Deducteam/lambdapi/pull/639`.

References

1 B. Accattoli and B. Barras. Environments and the complexity of abstract machines. In *Proceedings of the 19th International Conference on Principles and Practice of Declarative Programming*, 2017. doi:10.1145/3131851.3131855.

2 Agda sort system. https://agda.readthedocs.io/en/latest/language/sort-system.html.

3 http://aprove.informatik.rwth-aachen.de/.

4 A. Asperti, W. Ricciotti, C. Sacerdoti Coen, and E. Tassi. A bi-directional refinement algorithm for the calculus of (co)inductive constructions. *Logical Methods in Computer Science*, 8:1–49, 2012. doi:10.2168/LMCS-8(1:18)2012.

5 A. Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École Polytechnique, France, 2015. URL: https://tel.archives-ouvertes.fr/tel-01235303/.

6 A. Assaf, G. Dowek, J.-P. Jouannaud, and J. Liu. Encoding proofs in Dedukti: the case of Coq proofs, 2016. Presented at the First International Workshop on Hammers for Type Theories (HaTT). URL: https://hal.inria.fr/hal-01330980.

7 H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of logic in computer science. Volume 2. Background: computational structures*, pages 117–309. Oxford University Press, 1992.

8 B. Barras, J.-P. Jouannaud, P.-Y. Strub, and Q. Wang. CoqMTU: a higher-order type theory with a predicative hierarchy of universes parameterized by a decidable first-order theory. In *Proceedings of the 26th IEEE Symposium on Logic in Computer Science*, 2011. doi:10.1109/LICS.2011.37.

9 F. Blanqui. Type safety of rewrite rules in dependent types. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020. doi:10.4230/LIPIcs.FSCD.2020.13.

10 F. Blanqui, G. Dowek, E. Grienenberger, G. Hondet, and F. Thiré. Some axioms for mathematics. In *Proceedings of the 6th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 195, 2021. doi:10.4230/LIPIcs.FSCD.2021.20.

11 F. Blanqui, T. Hardin, and P. Weis. On the implementation of construction functions for non-free concrete data types. In *Proceedings of the 16th European Symposium on Programming*, Lecture Notes in Computer Science 4421, 2007. 15 pages. doi:10.1007/978-3-540-71316-6_8.

12 M. Boespflug, M. Dénès, and B. Grégoire. Full reduction at full throttle. In *Proceedings of the 1st International Conference on Certified Programs and Proofs*, Lecture Notes in Computer Science 7086, 2011. doi:10.1007/978-3-642-25379-9_26.

13 J. Chrząszcz. Modules in Coq are and will be correct. In *Proceedings of the International Workshop on Types for Proofs and Programs*, Lecture Notes in Computer Science 3085, 2003. doi:10.1007/978-3-540-24849-1_9.

14 D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 4583, 2007. doi:10.1007/978-3-540-73228-0_9.

15 S. Eker. Fast matching in combinations of regular equational theories. In *Proceedings of the 1st International Workshop on Rewriting Logic and Applications*, Electronic Notes in Theoretical Computer Science 4, 1996. doi:10.1016/S1571-0661(04)00035-0.

16 S. Eker. Associative-commutative rewriting on large terms. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003. doi:10.1007/3-540-44881-0_3.

17 G. Férey. *Higher-Order Confluence andUniverse Embedding in theLogical Framework*. PhD thesis, Université Paris-Saclay, France, 2021.

18 M. Fernández and J.-P. Jouannaud. Modular termination of term rewriting systems revisited. In *Proceedings of the 10th International Workshop on Specification of Abstract Data Types*, Lecture Notes in Computer Science 906, 1994. doi:10.1007/BFb0014432.

19 G. Genestier. *Dependently-Typed Termination and Embedding of Extensional Universe-Polymorphic Type Theory using Rewriting*. PhD thesis, Université Paris-Saclay, 2020. URL: `https://hal.inria.fr/tel-03167579`.

20 G. Genestier. Encoding agda programs using rewriting. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020. `doi:10.4230/LIPIcs.FSCD.2020.31`.

21 B. Gramlich. Modularity in term rewriting revisited. *Theoretical Computer Science*, 464:3–19, 2012. `doi:10.1016/j.tcs.2012.09.008`.

22 R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993. `doi:10.1145/138027.138060`.

23 G. Hondet and F. Blanqui. The new rewriting engine of dedukti. In *Proceedings of the 5th International Conference on Formal Structures for Computation and Deduction*, Leibniz International Proceedings in Informatics 167, 2020. `doi:10.4230/LIPIcs.FSCD.2020.35`.

24 D. Kapur and P. Narendran. NP-completeness of the associative-commutative unification and related problems. Unpublished Manuscript. Computer Science Branch, General Electric Corporate Research and Development, Schenectady, NY. See [25], 1986.

25 D. Kapur and P. Narendran. Matching, unification and complexity. *SIGSAM Bull.*, 21(4):6–9, 1987. `doi:10.1145/36330.36332`.

26 S. Krstić and S. Conchon. Canonization for disjoint unions of theories. *Information and Computation*, 199(1-2):87–106, 2005. `doi:10.1016/j.ic.2004.11.001`.

27 C. Marché. Normalized rewriting: an alternative to rewriting modulo a set of equations. *Journal of Symbolic Computation*, 21(3):253–288, 1996. `doi:10.1006/jsco.1996.0011`.

28 P. Martin-Löf. An intuitionistic theory of types: predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Proceedings of the 1973 Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1975. URL: `http://archive-pml.github.io/martin-lof/pdfs/An-Intuitionistic-Theory-of-Types-Predicative-Part-1975.pdf`.

29 M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu Matematykow Krajow Slowcanskich, Warszawa, Poland*, 1929.

30 R. Saillard. *Type checking in the Lambda-Pi-calculus modulo: theory and practice*. PhD thesis, Mines ParisTech, France, 2015. URL: `https://pastel.archives-ouvertes.fr/tel-01299180`.

31 R. Shostak. Deciding combination of theories. *Journal of the ACM*, 31(1):1–12, 1984.

32 M. Sozeau. Polymorphic universes.
`https://coq.inria.fr/refman/addendum/universe-polymorphism.html`.

33 M. Sozeau and N. Tabareau. Universe polymorphism in Coq. In *Proceedings of the 5th International Conference on Interactive Theorem Proving*, Lecture Notes in Computer Science 8558, 2014. `doi:10.1007/978-3-319-08970-6_32`.

34 TeReSe. *Term rewriting systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

35 B. Ziliani and M. Sozeau. A comprehensible guide to a new unifier for CIC including universe polymorphism and overloading. *Journal of Functional Programming*, 27(E10), 2017. `doi:10.1017/S0956796817000028`.