Encoding Proofs in Dedukti: the case of Coq proofs

Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, Jiaxiang Liu

Encoding Proofs in Dedukti: the case of Coq proofs

Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud and Jiaxiang Liu

1. Introduction

A main ambition of the Inria project Dedukti described at `http://dedukti.gforge.inria.fr/` is to serve as a common language for representing and type checking proof objects originating from other proof systems. Encoding these proof objects makes heavy use of the rewriting capabilities of $\lambda\Pi$Mod, the formal system on which Dedukti is based [4, 9]. So far, the proofs generated by two automatic proof systems, Zenon and iProver, have been encoded, and can therefore be read and checked by Dedukti. But Dedukti goes far beyong this so-called *hammering* technique of sending goals to automated provers. Proofs from HOL and Matita can be encoded as well [1, 4]. Some Coq's proofs can be encoded already, when they do not use universe polymorphism [3]. Our ambition here is therefore to close this remaining gap.

To this end , we describe in Section 3, a rewrite-based encoding in $\lambda\Pi$Mod of the *Calculus of Constructions with Universes* $\mathrm{CCU}^{\infty}_{\subseteq}$, a generalization of the calculus of constructions with an infinite hierarchy of predicative universes above the impredicative universe Prop. Together with inductive types, it forms the core of the *Calculus of Inductive Constructions* as is implemented in the proof system Coq. Encoding inductive types in the style of Blanqui [6] is relatively simple [7]. The major difficulty when encoding $\mathrm{CCU}^{\infty}_{\subseteq}$ is indeed the treatment of *universe cumulativity*, which needs to be rendered explicit. Existing encodings of universe cumulativity in $\lambda\Pi$Mod have limitations:

- The encoding of [1] is purely equational, resulting in complex proofs. Further, the universe hierarchy is encoded by a set of function symbols indexed *externally*, which is therefore *infinite*, yet another source of complexity in proofs.
- Although rewrite based, the more elaborated attemp in [3] is confluent on ground terms only, hence restricting its use in $\lambda\Pi$Mod to encode type systems wich do not include universe polymorphism, like Matita.

Our encoding is based on a finite set of rewrite rules which is confluent on terms with variables, the key to universe polymorphism. This encoding uses, in addition to β-rewrites, non-left-linear first-order and higher-order rewrite rules. Further, some first-order rewrite rules use pattern matching modulo associativity and commutativity, and even identity, making confluence a main theoretical difficulty. A second major contribution of this work is therefore the development of a new technique for showing confluence of such complex rewrite systems *on untyped terms*.

2. $\lambda\Pi$Mod

Terms. Let $\mathcal{X}$, Σ_{fo} and Σ_{ho} be pairwise disjont sets of *variables*, *first-order* and *higher-order* symbols respectively, the latter two equipped with a fixed arity. Let also $\mathcal{Y}$ and Σ_{cd} be subsets of $\mathcal{X}$ and Σ_{fo} respectively, of elements we call *confined*. The set of (untyped) terms is defined by the grammar rules:

$$\begin{array}{rcl} M, N & \stackrel{\text{def}}{=} & x{\in}\mathcal{X} \mid \lambda x : M.N \mid M\ N \mid \Pi x : M.N \mid f(\overline{M}) \mid U \\ & & \text{with } f \in (\Sigma_{fo} \setminus \Sigma_{cd}) \cup \Sigma_{ho} \\ U & \stackrel{\text{def}}{=} & y{\in}\mathcal{Y} \mid g(\overline{U}), \text{ with } g \in \Sigma_{cd} \end{array}$$

The set $\Sigma = \Sigma_{fo} \cup \Sigma_{ho}$ is called the user's *signature*. Confined expressions are first-order. Type constructors * and $\Box$ are symbols from Σ, regardless of their specific role. We write f instead of $f()$. The *head* of a term is its outermost symbol. An abstraction $\lambda y : U.M$ and a product $\Pi y : M.V$ are headed by the binary symbols $\lambda y{:}\_.\_$ and $\Pi y{:}\_.\_$ respectively. Both these and application are the *functional* symbols. We use $=$ for the syntactic equality of terms. Given a set or term A, we denote by $|A|$ its size.

Terms built solely from the signature and variables are called *algebraic*, whose subset of *confined* algebraic terms is generated from the non-terminal U. Confinement of untyped terms should of course follow from confinement of typed terms, hence be enforced, for a given specification in $\lambda\Pi$Mod, by the typing rules.

Typing rules There are two forms of judgements, $\Gamma \vdash M : A$ meaning that the term M *has type* A in the context Γ, and $\Gamma \vdash$ meaning that the context Γ is *well-formed*. nil is the empty context.

$$\frac{}{nil \vdash}$$

$$\frac{\Gamma \vdash A : * \qquad x \notin \Gamma}{\Gamma, x : A \vdash} \qquad \frac{\Gamma \vdash \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\left\{\begin{array}{l} f : \Pi x_1 : A_1. \cdots \Pi x_n : A_n.B \in \Sigma \\ \Gamma \vdash \\ \Gamma \vdash M_1 : A_1 \\ \vdots \\ \Gamma \vdash M_n : A_n\{x_1 \mapsto A_1, \ldots, x_{n-1} \mapsto A_{n-1}\} \end{array}\right.}{\Gamma \vdash f(M_1, \ldots, M_n) : B\{x_1 \mapsto A_1, \ldots, x_n \mapsto A_n\}}$$

$$\frac{\Gamma, x : A \vdash M : B \qquad B \text{ is not confined}}{\Gamma \vdash \lambda x : A.M : \Pi x : A.B}$$

$$\frac{\Gamma \vdash M : \Pi x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B\{x \mapsto N\}}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : * \qquad A \equiv B}{\Gamma \vdash M : B}$$

Note the specific form of the abstraction rule which does not allow to abstract over confined expressions.

Rewrite rules available in $\lambda\Pi$Mod are described in detail later.

3. Encoding Coq's universes in $\lambda\Pi$Mod

The signature. The encoding uses function symbols to represent sorts, types and terms of CIC. The arity of a function symbols is indicated in superscript position in the declaration of the type of that symbol. Arity 0 is omitted. Knowledge of $\mathrm{CCU}^{\infty}_{\subseteq}$ is assumed to understand the rules. Appropriate expositions are [1, 3, 16].

The specification has 3 type constructors, Sort, U and T. Other declared symbols allow us to build objects and the dependent type they inhabit.

Sort is the type for universes in our encoding, starting with the impredicative universe Prop and continuing with the predicative

ones. For convenience, we simply call them $0, 1, \ldots$, so as to be represented by a copy of the natural numbers generated by three constructors, $0, 1$ and $+$. This perhaps unusual encoding is instrumental in obtaining a finite Church-Rosser system.

The symbols U and T represent, respectively, the type of *codes* and the *decoding function* of universes [9], with u, $\uparrow$, $\Uparrow$ and π being codes for the type $\mathrm{U}(0)$, for the types lifted respectively one and n levels using cumulativity, and for Π-types.

Variables i, j are of type Sort, variable a has type$\mathrm{U}(i)$ for some i, while variable b has a more complex Π-type (see π's type).

$$\begin{array}{lcl}
\mathrm{Sort} & : & * \\
0, 1 & : & \mathrm{Sort} \\
+^2 & : & \Pi i : \mathrm{Sort} \,.\Pi j : \mathrm{Sort} \,.\, \mathrm{Sort} \\
\max^2 & : & \Pi i : \mathrm{Sort} \,.\Pi j : \mathrm{Sort} \,.\, \mathrm{Sort} \\
\mathrm{rule}^2 & : & \Pi i : \mathrm{Sort} \,.\Pi j : \mathrm{Sort} \,.\, \mathrm{Sort}
\end{array}$$

$$\begin{array}{lcl}
\mathrm{U}^1 & : & \Pi i : \mathrm{Sort} \,.\, * \\
\mathrm{T}^2 & : & \Pi i : \mathrm{Sort} \,.\Pi a : \mathrm{U}(i). * \\
\mathrm{u}^1 & : & \Pi i : \mathrm{Sort} \,.\, \mathrm{U}(0) \\
\uparrow^1 & : & \Pi i : \mathrm{Sort} \,.\, \mathrm{U}(+(i, 1)) \\
\Uparrow^2 & : & \Pi i : \mathrm{Sort} \,.\Pi j : \mathrm{Sort} \,.\Pi a : \mathrm{U}(i).\, \mathrm{U}(\max(i, j)) \\
\pi^4 & : & \Pi i : \mathrm{Sort} \,.\Pi j : \mathrm{Sort} \,.\Pi a : \mathrm{U}(i). \\
 & & \qquad \Pi b : (\Pi x : \mathrm{T}(i, a).\, \mathrm{U}(j)).\, \mathrm{U}(\mathrm{rule}(i, j))
\end{array}$$

The rewrite system. The constructor $+$ is associative and commutative, and has 0 as identity element. We will take the liberty to use an infix notation for $+$, the abbreviation 2 for $1 + 1$, and a varyadic number of arguments to ease the readability of sums.

$$\begin{array}{lrcl}
1: & \max(i, i + j) & \underset{m_1}{\to} & i + j \\
2: & \max(i + j, j) & \underset{m_2}{\to} & i + j \\
3: & \mathrm{rule}(i, 0) & \underset{m_3}{\to} & 0 \\
4: & \mathrm{rule}(i, j + 1) & \underset{r_1}{\to} & \max(i, j + 1) \\
5: & \Uparrow(0, a) & \underset{l_1}{\to} & a \\
6: & \Uparrow(i + 1, a) & \underset{l_2}{\to} & \uparrow(i, \Uparrow(i, a))
\end{array}$$

The rules for $\max$, rule and $\Uparrow$ are self explanatory. The rule function symbol is used to account for the impredicativity of Prop encoded here as the universe 0: rule behaves as $\max$ when its second argument is a predicative universe.

$$\begin{array}{lrcl}
7: & \pi(i+1, i+j+1, \uparrow(i, a), b) & \underset{p_1}{\to} & \pi(i, i+j+1, a, b) \\
8: & \pi(i+j+2, j+1, \uparrow(i+j+1, a), b) & \underset{p_2}{\to} & \uparrow(i+j+1, \pi(i+j+1, j+1, a, b)) \\
9: & \pi(i+j+2, j+2, a, \uparrow(j+1, b)) & \underset{p_3}{\to} & \pi(i+j+2, j+1, a, b) \\
10: & \pi(i, i+j+1, a, \uparrow(i+j, b)) & \underset{p_4}{\to} & \uparrow(i+j, \pi(i, i+j, a, b)) \\
11: & \pi(i+1, 1, a, \uparrow(0, b)) & \underset{p_5}{\to} & \Uparrow(i+1, \pi(i+1, 0, a, b)) \\
12: & \pi(0, 1, a, \uparrow(0, b)) & \underset{p_5}{\to} & \uparrow(0, \pi(0, 0, a, b)) \\
13: & \pi(i+1, 0, \uparrow(i, a), b) & \underset{p_6}{\to} & \pi(i, 0, a, b) \\
14: & \mathrm{T}(i+1, \mathrm{u}(i)) & \underset{t_1}{\to} & \mathrm{U}(i) \\
15: & \mathrm{T}(i+1, \uparrow(i, a)) & \underset{t_2}{\to} & \mathrm{T}(i, a) \\
16: & \mathrm{T}(i, \Uparrow(i, a)) & \underset{t_3}{\to} & \mathrm{T}(0, a) \\
17: & \mathrm{T}(0, \pi(i, 0, a, b)) & \underset{t_4}{\to} & \Pi x : T(i, a).T(0, b) \\
18: & \mathrm{T}(i+j, \pi(i, i+j, a, b)) & \underset{t_5}{\to} & \Pi x : T(i, a).T(i+j, b) \\
19: & \mathrm{T}(i+j+1, \pi(i+j+1, j+1, a, b)) & \underset{t_6}{\to} & \Pi x : T(i+j+1, a).T(j+1, b)
\end{array}$$

The rules for T are standard decoding rules [9]. The rules for π are most delicate and are chosen to ensure that types have a unique encoding, a property that is crucial for the preservation of typing [1, 3]. Their design obtained by comparing (via $+$) the first two arguments of π ensures that they have very few critical pairs (eliminating them all would require a richer language involving a comparison operator, which would raise other confluence problems).

A faithfull encoding. We denote the obtained signature and rewrite system by Σ_{CIC} and R_{CIC} respectively.

There are functions $[M]_A$ and $[\![A]\!]$ that faithfully translate the terms of $\mathrm{CCU}^\infty_\subseteq$ into the terms of $\lambda\Pi\mathrm{Mod}$ with signature Σ_{CIC} :

THEOREM 3.1 (Preservation of typing). *For any Γ, M, and A in $\mathrm{CCU}^\infty_\subseteq$, if $\Gamma \vdash M : A$ then $[\![\Gamma]\!] \vdash [M]_A : [\![A]\!]$ in $\lambda\Pi\mathrm{Mod}$ with signature Σ_{CIC}.*

The reader might ask, rightfully, about the converse of the theorem above. Indeed, if we can prove $[\![\bot]\!]$ in $\lambda\Pi\mathrm{Mod}$ then the encoding would be useless. Preservation of inhabitation (also called *conservativity*) will ensure that this is not the case.

CLAIM 3.1 (Preservation of inhabitation). *For any Γ, M, and A in $\mathrm{CCU}^\infty_\subseteq$, if $[\![\Gamma]\!] \vdash [M]_A : [\![A]\!]$ then $\Gamma \vdash M : A$.*

A Church-Rosser encoding. This set of rules and equations R_{CIC} has been obtained with the MAUDE system [8] –using MAUDE has been instrumental in this quest–, by hiding the higher-order aspects of some of the rules. Further, we also used MAUDE to compute all critical overlaps (modulo associativity, commutativity and identity) of the set of rules. Because the β-rule is non-terminating on untyped terms, we had to show that each critical pair has a decreasing diagram in the sense of van Oostrom [17]. MAUDE does not support confluence proofs based on decreasing diagrams, which forced us to do these computations by hand.

On the other hand, van Oostrom's theorem is abstract, its application to particular rewrite systems is non-trivial, and, indeed, no result prior to ours could show the Church-Rosser property of R_{CIC}. This is so because some first-order rules use pattern matching modulo commutativity, associativity, and identity, and higher-order rules are non-left-linear. We had therefore to design our own application of van Oostrom's theorem in order to show the Church-Rosser property of R_{CIC} (augmented with the β-rule).

Let us explain how the confluence proof goes. First, we split the signature into first-order and higher-order. The rules for Π and T contain functional symbols, they are higher-order. Since the other rules contain no functional symbols, nor Π, T, they can be taken as being first-order. This defines the first-order signature $\Sigma_{fo} = \{0, 1, +, \max, \mathrm{rule}, \uparrow, \Uparrow\}$, and the higher-order one $\Sigma_{ho} = \{\mathrm{T}, \pi, \mathrm{U}\}$. The signature Σ_{fo} needs then to be confined : universe calculations operate on first-order expressions and subexpressions. These signatures split the set R_{CIC} into two subsets, R_{fo} and R_{ho} of respectively first-order and higher-order rewrite rules.

For R_{fo}, following [11], we must first show termination of normal rewriting in ACI equivalence classes. Termination can indeed be achieved easily by Rubio's fully syntactic AC path ordering [15]. This order works much like Dershowitz's recursive path ordering, but has a specific status for AC operators that performs additional monotonicity checks. All symbols can have a multiset status but rule, whose status must be lexicographic while the precedence can be $\mathrm{rule} > \max > + > 1 > 0$. Routine calculations show termination. There is actually a little difficulty here: termination is shown by an AC-compatible ordering, and not an ACI-compatible ordering. This is however sufficient, since terms to be rewritten must be in normal form wrt identity as a rule.

We must then check that the ACI-critical pairs between R_{fo}-rules are joinable by rewriting. This is routine too, there is only a trivial one between the two $\max$ rules.

We are left with our last task, showing that critical pairs involving at least one higher-order rule have decreasing diagrams. All these critical pairs have been be computed with MAUDE, while their decreasing diagrams have been computed by hand, since MAUDE cannot do that. These computations are summerized in Figure 1. The first column lists the critical pairs as given by MAUDE, the second column lists the real overlaps, the third the joinability diagram for the corresponding critical pair, and the last the constraints generated in order to obtain a decreasing diagram. These constraint operate on the rules' *labels*, which is indicated as a subscript of the rewriting arrow.

THEOREM 3.2. *The dependent type theory* $\lambda\Pi$Mod *equipped with the encoding of the cumulative hierarchy of predicative universes is Church-Rosser.*

4. Computation rules in $\lambda\Pi$Mod and their confluence

In all type theories, computation is based on rewrite rules.

DEFINITION 4.1. *A* rule *is a triple* $i : l \to r$*, whose possibly omitted name* i *is a natural number and lefthand and righthand side terms* l, r *satisfy* $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$*. A variable is* left linear *in* $i : l \to r$ *if it occurs exactly once in* l*, and regular if it occurs in* r *whenever it occurs in* l*. A* rewrite system *is a set of rules. An* equational system E *is a symmetric set of rules:* $i : r \to l \in E$ *iff* $i : l \to r \in E$*. Its rules, called* equations*, are also written* $i : l = r$*.*

Different rules may share the same index (so do the two rules of an equation). All variables of an equation are regular by definition. Rules and equations are *algebraic* when both sides are algebraic. So are associativity and commutativity (AC).

Functional computations in $\lambda\Pi$**Mod.** $\lambda\Pi$Mod comes with two rewriting schemas, α-conversion and β-reduction:

$$\begin{array}{lll} \lambda y : U \,.\, M = \lambda z : U \,.\, M\{y \mapsto z\} & \text{if } z \notin \mathcal{BV}ar(U, M) & (\alpha) \\ \Pi y : U \,.\, V = \Pi z : U \,.\, V\{y \mapsto z\} & \text{if } z \notin \mathcal{BV}ar(U, V) & (\alpha) \\ (\lambda y : U \,.\, M)\, N \to M\{y \mapsto N\} & & (\beta) \end{array}$$

Since substitution is not part of our language, (α) and (β) are equation/rule schemas. Further, substituting N to each occurrence of y in M involves renaming the bound variables of M away from those of N in order to avoid captures: (β) rewrites *modulo* α, which is therefore the set of equations attached to functional computations.

Two major properties are that $\longrightarrow_{\beta_\alpha}$ is Church-Rosser modulo α and commutes over $=_\alpha$, as well as over any equational theory generated by a set of algebraic equations whose non-linear variables are confined.

First-order computations in $\lambda\Pi$**Mod.** We assume for simplicity that all first-order terms are confined, which is the case of our example. We are therefore given two sets of algebraic rules R_{fo} and equations E_{fo} built from the signature Σ_{fo}. Rewriting with R_{fo} is therefore modulo E_{fo}.

Major assumption 1: R_{fo} is terminating in E_{fo}-equivalence classes and Church-Rosser modulo E_{fo}.

Testing for the Church-Rosser property will classically require another assumption:

Major assumption 2: E_{fo}-unification is finite and complete.

Higher-order computations in $\lambda\Pi$**Mod.** We assume now given a set of *higher-order* rules R_{ho}. Rules that contain functional symbols on either side must be in R_{ho}, as well as all algebraic rules having non-confined variables that occur in the lefthand side only.

Higher-order computations result from two kinds of rules, depending on the structure of their lefthand sides, either higher-order patterns in Miller's sense [13], or algebraic expressions as in recursor rules [5] or in non-regular algebraic rules. Patterns require using (some form of) higher-order pattern matching while first-order pattern matching suffices for algebraic lefthand sides.

DEFINITION 4.2 (Patterns [13]). *A (higher-order)* pattern *is a term* L *in* β*-normal form, headed by a symbol in* Σ_{ho}*, such that every free variable* X *occurs in a subterm* $(\dots(X\, x_1)\dots x_n)$*,* $n \geq 0$*, written* $(X\ \overline{y})$ *when* $\overline{y}$ *is a vector of* $n \neq 0$ *distinct variables bound above in* L*, and* X *otherwise, in which case* X *is* first-order*.*

For example, the term $\mathsf{diff}(\lambda x.\, \mathsf{sin}(F\ x))$ is a pattern. The term $\mathsf{rec}(S(x), X, X')$, lefthand side of one of the two classical recursor rules over natural numbers, is a pattern as well. More generally, algebraic terms are patterns whose variables are not applied, hence are first-order (despite possibly having a higher-order type like X, X' above –our vocabulary here is non-standard).

The main property of a pattern is that it generates by instantiation very specific β-redexes and reducts, at predictable positions, which can be beta-reduced. Consider the subterm $(X\ \overline{y})$ at position q in a pattern L, and a substitution $\sigma = \{X \mapsto \lambda\overline{z}.u\}$. Then, $(L\sigma)|_q \longrightarrow\!\!\!\!\!\rightarrow_\beta u\{\overline{z} \mapsto \overline{y}\}$. So, the instance of a pattern by such a substitution beta-reduces to a term obtained by replacing the subterms of the form $(X\ \overline{y})$ by terms of the form $u\{\overline{z} \mapsto \overline{y}\}$ where u is a term which does not contain any free occurrence of variables in $\overline{y}$. This particular case of beta-rewriting restricted to the case where the argument N is a variable is called beta^0-rewriting by Miller who introduced it [13]. Beta^0-rewrites have a property which is crucial in our setting: since they decrease the size of terms, whether typed or not, beta^0-rewrites terminate on all terms. We denote by $u{\downarrow_{\beta^0}}$ the beta^0-normal form of a term u.

We need names for the positions of variables in a pattern [10]:

DEFINITION 4.3. *The* fringe F_L *of a higher-order pattern* L *is the union of three pairwise disjoint sets of positions: (i) its* functional fringe $F_L^{fun} = \{p \in \mathcal{FP}os(L) : L|_p = (X\,\overline{y}), X \in \mathcal{V}ar(L), |\overline{y}| > 0\}$*; (ii) its* confined fringe $F_L^{cd} = \{p \in \mathcal{P}os(L) : L|_p$ *is confined*$\}$*; and (iii) its* variable fringe F_L^{var}*, set of positions of its non-confined first-order variables.*

Note that algebraic patterns have an empty functional fringe, and that the fringe of a linear algebraic pattern is the set of positions of its variables, in which case $(\geq_{\mathcal{P}} F_L) = (\geq_{\mathcal{P}} \mathcal{VP}os(L))$.

Major assumption 3: Lefthand sides of rules in R_{ho} are patterns whose non-linear variables are confined.

Miller showed that higher-order pattern matching and unification of patterns is decidable [13] (in linear time).

An important remark here is that higher-order E_{fo}-unification of two patterns is modular, that is, it reduces to higher-order unification of patterns on the one hand, and E_{fo}-unification of confined terms on the other hand. The reason is that confined variables can only be equated to confined terms, otherwise unification fails. This is yet another, very practical, implication of our notion of confined variable.

Nipkow's definition of higher-order rewriting based on higher-order pattern matching was elaborated for terminating computations [14]. It requires in particular that terms to be rewritten are in beta-normal form, and beta-normalizes the result to preserve that property. Our definition for non-terminating computations operates instead on terms in beta^0-normal form. It also allows to control how higher-order pattern matching operates on higher-order patterns by using beta^0-rewriting instead of beta^0-conversion, and allows confined equalities to operate below the confined variables:

DEFINITION 4.4. *A term u rewrites with a higher-order rule i : $L \to R \in R_{ho}$ at position $p \in \mathcal{P}os(u)$, written $u \xrightarrow[i]{p} v$, if*

(i) $u|_p$ is in beta^0-normal form,

(ii) $L\gamma \xrightarrow[\beta^0]{\geq_{\mathcal{P}} F_L^{fun}} w \xleftarrow[E_{fo}]{\geq_{\mathcal{P}} F_L^{cd}} u|_p$, and

(iii) $v = u[R\gamma\downarrow_{\beta^0}]_p$.

Note that our definition coincides with plain rewriting when L is a linear, algebraic pattern. It will therefore allow us to treat all rules in R_{ho} uniformly, including the recursor rules if any.

The confluence proof is based on a carefull analysis of the rewriting peaks, in order to show that they all have decreasing diagrams. There are two kinds of peaks, homogeneous and heterogeneous. Heterogeneous peaks require using parallel functional computations at disjoint positions, and parallel higher-order computations at non-overlapping positions. Homogeneous first-order peaks can be solved easily thanks to our assumption that R_{fo} is Church-Rosser modulo E_{fo}. Homogeneous parallel functional peaks can be solved because (parallel) beta-rewriting is Church-Rosser. Homogeneous higher-order peaks need assuming that the corresponding critical pairs have deceasing diagrams. Because these critical pairs are based on rewriting in parallel at disjoint positions, a further assumption (DO) is needed in order to ensure that they are finitely many, by restricting the possibility to iterate overlaps:

$$
\begin{array}{lcl}
\text{(DO)} & \stackrel{\text{def}}{=} & (\forall L_i \to R_i \in R_{ho})\,(\forall p <_{\mathcal{P}} F_{L_i}) \\
 & & (\forall L_j \to R_j \in R_{ho} \text{ s.t. } \mathcal{V}ar(L_i) \cap \mathcal{V}ar(L_j) = \emptyset) \\
 & & (\forall \sigma \text{ s.t. } L_i|_p\sigma =_{\beta^0 \cup E_{fo}} L_j\sigma)\, \mathrm{SOF}_j(L_i|_p) \wedge \mathrm{SOF}_i(L_j) \\
\mathrm{SOF}_i(u) & \stackrel{\text{def}}{=} & (\forall q \in \mathcal{FP}os(u)\backslash\{\Lambda\})\, \mathrm{OF}_i(u|_q) \\
\mathrm{OF}_i(v) & \stackrel{\text{def}}{=} & (\forall L_i \to R_i \in R_{ho} \text{ s.t. } \mathcal{V}ar(v) \cap \mathcal{V}ar(L_i) = \emptyset) \\
 & & (\forall o \in \mathcal{FP}os(v))(\forall \sigma)\ v|_o\sigma \neq_{\beta^0 \cup E_{fo}} L_i\sigma
\end{array}
$$

SOF stands for *strict subterm overlap-free*, and OF for *overlap-free*.

Major assumption 4: R_{ho} satisfies (DO).

Assumption (DO) is quite liberal, and indeed much weaker than the layering assumption from which it is inspired [12].

THEOREM 4.1. λΠMod *equipped with a set of rules and equations (R_{fo}, E_{fo}, R_{ho}) satisfying our assumptions is Church-Rosser modulo E_{fo} on untyped terms if its critical pairs involving higher-order rules are joinable via decreasing diagrams.*

References

[1] Ali Assaf. A calculus of constructions with explicit subtyping. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*, volume 39 of *LIPIcs*, pages 27–46. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[2] Ali Assaf. Conservativity of embeddings in the lambda pi calculus modulo rewriting. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, volume 38 of *LIPIcs*, pages 31–44. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[3] Ali Assaf. *A framework for defining computational higher-order logics*. PhD thesis, École polytechnique, Paris, 2015.

[4] Ali Assaf and Guillaume Burel. Translating HOL to dedukti. In Cezary Kaliszyk and Andrei Paskevich, editors, *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, volume 186 of *EPTCS*, pages 74–88, 2015.

[5] F. Blanqui, J.-P. Jouannaud, and M. Okada. The calculus of algebraic constructions. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 9th International Conference on Rewriting Techniques and Applications (RTA '99)*, number 1631 in LNCS, pages 301–316, Trento, Italy, July 1999. Springer Verlag.

[6] F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type systems. *Theoretical Computer Science*, 272:41–68, 2002.

[7] Mathieu Boespflug and Guillaume Burel. CoqInE: Translating the calculus of inductive constructions into the lambda-Pi-calculus modulo. In *Proof Exchange for Theorem Proving—Second International Workshop, PxTP*, page 44, 2012.

[8] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[9] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2007.

[10] J.-P. Jouannaud and C. Kop. Church-rosser properties of terminating rewriting computations. Draft.

[11] Jean-Pierre Jouannaud and Jianqi Li. Church-Rosser properties of normal rewriting. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 350–365. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[12] Jiaxiang Liu, Jean-Pierre Jouannaud, and Mizuhito Ogawa. Confluence of layered rewrite systems. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 423–440. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[13] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[14] T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, Amsterdam, The Netherlands, July 1991.

[15] Albert Rubio. A fully syntactic AC-RPO. *Inf. Comput.*, 178(2):515–533, 2002.

[16] Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in coq. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2014.

[17] Vincent van Oostrom. Confluence by decreasing diagrams. *Theor. Comput. Sci.*, 126(2):259–280, 1994.

Equation	Overlap	Joinability	DD Constraint
$p_1 = p_3$	$\pi(i+2, i+2, \uparrow(i+1, a), \uparrow(i+1, b))$	$\underset{t_2}{\longrightarrow} \underset{t_6}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_1 > t_2, t_6$
$p_2 = p_3$	$\pi(i+j+3, i+2, \uparrow(i+j+1, a), \uparrow(j+1, b))$	$\underset{t_6}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_2 > t_2, t_6$
$p_4 = p_1$	$\pi(i+1, i+j+2, \uparrow(i, a), \uparrow(i+j+1, b))$	$\underset{p_1}{\longrightarrow} = \underset{p_4}{\longleftarrow}$	
$p_5 = p_1$	$\pi(1, 1, \uparrow(0, a), \uparrow(0, b))$	$\underset{p_4}{\longrightarrow} = \emptyset$	$p_5 > p_4$
$p_5 = p_2$	$\pi(i+j+1, 1, \uparrow(i+1, a), \uparrow(0, b))$	$\underset{p_6}{\longrightarrow} \underset{l_2}{\longrightarrow} = \underset{p_5}{\longleftarrow}$	$p_5 > p_6, l_2$
$t_4 = t_5$	$\mathrm{T}(0, \pi(0, 0, a, b))$	$\emptyset = \emptyset$	
$t_5 = t_6$	$\mathrm{T}(j+1, \pi(j+1, j+1, a, b))$	$\emptyset = \emptyset$	
$\mathrm{T}(0, l_1) = t_3$	$\mathrm{T}(0, \Uparrow(0, a))$	$\emptyset = \emptyset$	
$\mathrm{T}(1+i, l_2) = t_3$	$\mathrm{T}(i+1, \Uparrow(i+1, a))$	$\underset{t_2}{\longrightarrow} \underset{t_3}{\longrightarrow} = \emptyset$	$p_4 > t_2$
$\mathrm{T}(0, p_6) = t_4$	$\mathrm{T}(0, \pi(i+1, 0, \uparrow(i, a), b)$	$\underset{t_4}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_6 > t_2$
$\mathrm{T}(i+j+1, p_1) = t_5$	$\mathrm{T}(i+j+1, \pi(i+1, i+j+1, \uparrow(i, a), b))$	$\underset{t_5}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_1 > t_2$
$\mathrm{T}(i+2, p_3) = t_5$	$\mathrm{T}(i+2, \pi(i+2, i+2, a, \uparrow(i+1, b)))$	$\underset{t_6}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_3 > t_2, t_6$
$\mathrm{T}(i+j+1, p_4) = t_5$	$\mathrm{T}(i+j+1, \pi(i, i+j+1, a, \uparrow(i+j, b)))$	$\underset{t_2}{\longrightarrow} \underset{t_5}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_4 > t_2, t_5$
$\mathrm{T}(1, p_5) = t_5$	$\mathrm{T}(1, \pi(1, 1, a, \uparrow(0, b)))$	$\underset{t_3}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_5 > t_2, t_3$
$\mathrm{T}(i+1, p_1) = t_6$	$\mathrm{T}(i+1, \pi(i+1, i+1, \uparrow(i, a), b))$	$\underset{p_4}{\longrightarrow} = \underset{p_2}{\longleftarrow}$	$p_1 > p_2, p_4$
$\mathrm{T}(i+j+2, p_2) = t_6$	$\mathrm{T}(i+j+2, \pi(i+j+2, j+1, \uparrow(j+1, a), b))$	$\underset{p_3}{\longrightarrow} = \underset{p_2}{\longleftarrow}$	$p_2 > p_3$
$\mathrm{T}(i+j+2, p_3) = t_6$	$\mathrm{T}(i+j+2, \pi(i+j+2, j+2, a, \uparrow(j+1, b)))$	$\underset{t_6}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_3 > t_2$
$\mathrm{T}(i+1, p_5) = t_6$	$\mathrm{T}(i+1, \pi(i+1, 1, a, \uparrow(0, b)))$	$\underset{l_2}{\longrightarrow} \underset{t_6}{\longrightarrow} = \underset{t_2}{\longleftarrow}$	$p_5 > l_2, t_2$

Figure 1. Decreasing diagrams computations

Church-Rosser calculations A solution to the ordering constraints is $p_5 > p_1 > p_2 > p_3 > p_4 > p_6 >$ other rules, which terminates the Church-Rosser proof.