Untyped Confluence in Dependent Type Theories

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

Untyped Confluence in Dependent Type Theories

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

Abstract

We investigate techniques based on van Oostrom's decreasing diagrams that reduce confluence proofs to the checking of critical pairs in the absence of termination properties, which are useful in dependent type calculi to prove confluence on untyped terms. These techniques are applied to a complex example originating from practice: a faithful encoding, in an extension of LF with rewrite rules on objects and types, of a subset of the calculus of inductive constructions with a cumulative hierarchy of predicative universes above Prop. The rules may be first-order or higher-order, plain or modulo, non-linear on the right or on the left. Variables which occur non-linearly in lefthand sides of rules must take their values in confined types: in our example, the natural numbers. The first-order rules are assumed to be terminating and confluent modulo some theory: in our example, associativity, commutativity and identity. Critical pairs involving higher-order rules must satisfy van Oostrom's decreasing diagram condition wrt their indexes taken as labels.

1. Introduction

The two essential properties of a type theory, consistency and decidability of type checking, follow from three simpler ones: type preservation, strong normalization and confluence. In dependent type theories however, confluence and type preservation are needed to build strong normalization models; confluence is needed to show preservation of product types by rewriting which is an essential ingredient of the type preservation proof; type preservation is needed to show that derivations issued from well-typed expressions are well-typed which is an essential ingredient of the confluence proofs. One can break this circularity in two ways : by proving all three properties together within a single huge induction [9]; or by proving confluence on untyped terms, which then allows to prove successively type preservation, confluence on typed terms, and strong normalization. The latter way is developed here

The confluence problem is indeed crucial for type theories allowing for user-defined computations such as in Dedukti (see `http://dedukti.gforge.inria.fr/`) and Agda. Current techniques for showing confluence by using van Oostrom theorem for higher-order rewrite systems [18], allow such an approach for simple type theories, in which the rules are left-linear, have development closed critical pairs, and do not build associativity and commutativity into pattern matching. But allowing for non-left-linear rules, or for non-trivial critical pairs or computing over non-free data structures, whether first-order like sets or higher-order like abstract syntax, is out of scope of current techniques. Such computations are however present in Dedukti. A main ambition of Dedukti is to serve as a common language for representing proof objects originating from different proof systems. Encoding these proof systems makes heavy use of the rewriting capabilities of $\lambda\Pi$Mod, the formal system on which Dedukti is based [4, 8].

We give here an encoding in $\lambda\Pi$Mod of the *Calculus of Constructions with Universes* $\mathrm{CCU}^{\infty}_{\subseteq}$, which uses both Nipkow's higher-order rewriting, non-left-linear rules, and associativity and commutativity. $\mathrm{CCU}^{\infty}_{\subseteq}$ is 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 [5] is relatively simple [6], we only allude to here. The major difficulty when encoding $\mathrm{CCU}^{\infty}_{\subseteq}$ is 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 [2] 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 attempt in [3] is confluent on ground terms only, hence restricting its use in $\lambda\Pi$Mod to encode type systems which do not include universe polymorphism, like Matita.

Our rewrite based encoding is confluent on terms with variables.

Because $\lambda\Pi$Mod includes beta-reduction, both on terms and types, the usual results for showing confluence of the above encoding on untyped terms do not apply. We have therefore developed a powerful result for showing confluence of our encoding, which applies to dependent type theories like those underlying Agda [14] and Dedukti. The main technical tool we use is van Oostrom's decreasing diagrams for labelled relations, which permits to prove confluence of rewrite systems that verify a kind of local confluence property called decreasing diagram [17]: local peak need not only to be joinable, but the labels of the joinable rewrites must be smaller than those of the local peak. In the case of $\lambda\Pi$Mod, we classify the user's rules in two categories: a set R_{fo} of non-erasing algebraic rules which is terminating and Church-Rosser modulo a set E_{fo} of algebraic equations; and a set R_{ho} of higher-order rules whose lefthand sides are patterns. Variables having multiple occurrences in lefthand sides of user's rules are guaranteed to operate on homogeneous algebraic terms by a syntactic assumption, *confinement*. Indeed, obtaining Church-Rosser calculi by putting together different confluent systems is known to be difficult in presence of non-left-linear rules [1]. Further, confluence of arbitrary non-left-linear rules is never preserved in presence of a fixpoint combinator [11], which can itself be encoded in the pure lambda calculus.

Local peaks may be of various kinds depending on which category each rule belongs to, and then closed in various ways. Homogeneous local peaks between two R_{fo}-rewrites may be closed because we assume their confluence and so are those between two beta rewrites because beta-reduction is known to be confluent. Closing homogeneous local peaks between R_{ho}-rewrites relies upon several technical novelties, among which a definition of higher-order rewriting for untyped terms which adapts Nipkow's definition given for typed terms by replacing beta-normalization by Miller's beta^0-normalization [12]. Closing heterogeneous ancestor peaks relies on another novelty, the generalization to rewriting modulo of Huet's ancestor peak property, under some *preservation* condition. Finally, by ensuring that only first-order redexes may occur below a non-linear variable of another redex, confinement eliminates local peaks that would not have decreasing diagrams otherwise.

We are not going to describe the confluence result in detail here, but give instead the rewrite rules encoding $\mathrm{CCU}^{\infty}_{\subseteq}$ in $\lambda\Pi$Mod. To have a feeling of the strength of the confluence result, remember that beta-reduction comes along with $\mathrm{CCU}^{\infty}_{\subseteq}$.

2. $\lambda\Pi$**Mod**

Terms and typing rules. Let $\mathcal{X}$, Σ_{fo} and Σ_{ho} be pairwise disjoint 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 a 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.

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

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

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

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

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

where $\equiv$ is joinability with the set of all rewrite rules to be described later. Confinement is built in the abstraction rule by forbidding to abstract over confined expressions.

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 CCU$^\infty_\subseteq$ is assumed to understand the rules. Appropriate expositions are [2, 3, 16].

The specification has 3 type constructors, Sort, U and T. Other symbols allow building objects and dependent types 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 [8], while u, $\uparrow$, $\Uparrow$ and π are codes for, the type $\mathrm{U}(0)$, the type lifted one level from some type $\mathrm{U}(i)$, the type lifted n level from the type $U(0)$, 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. The latter two types are the types of the third and fourth argument of π respectively.

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

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

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 variadic 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 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. It's name comes from PTS's.

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

The rules for T are standard decoding rules [8]. 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 [2, 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 faithful 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 $\text{CCU}^\infty_\subseteq$ into the terms of $\lambda\Pi$Mod with signature Σ_{CIC} :

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

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

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

4. Confluence via decreasing diagrams

Van Oostrom's confluence theorems are abstract, their application to particular rewrite systems is non-trivial, and, indeed, no result prior to ours could show the Church-Rosser property of R_{CIC}: first-order rules use pattern matching modulo ACI on terms in normal form for identity, and higher-order rules are non-left-linear, use higher-order confined pattern matching, and have critical pairs.

First-order rewriting We assume a normal rewriting system (R_{fo}, S_{fo}, E_{fo}) [10], the simplifiers in S_{fo} and equations in E_{fo} being confined, while the non-confined variables of the rules in R_{fo} must appear linearly on both sides.

Higher-order rewriting

DEFINITION 4.1. *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.

Lefthand sides of higher-order rules must be patterns, and their non-left-linear variables must be confined. 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 the main recursor rule over natural numbers, is a pattern as well. Both are linear, hence satisfy our assumption. More generally, algebraic terms whose non-linear variables are confined, are patterns whose variables are not applied, hence plain first-order pattern matching suffices for firing those patterns.

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 with a particular case of beta-reduction in which the argument is a variable. This is called beta^0-rewriting by Miller who defined higher-order pattern [12]. Beta^0-rewrites have a property which is crucial in our setting: by decreasing the size of terms, whether typed or not, they terminate on all terms. We denote by $u{\downarrow_{\beta^0}}$ the beta^0-normal form of term u.

DEFINITION 4.2. *The* fringe *F_L of a higher-order pattern L is made of two pairwise disjoint sets of positions, its* functional fringe *$F^{fun}_L = \{p \in \mathcal{FP}os(L) : L|_p = (X\,\overline{y}), X \in \mathcal{V}ar(L), |\overline{y}| > 0\}$ and its* confined fringe *$F^{cd}_L = \{p \in \mathcal{P}os(L) : L|_p$ is confined$\}$.*

Note that higher-order E-unification of patterns is modular when E is a set of confined equations: it reduces to higher-order unification of patterns on the one hand, and E-unification of confined terms on the other hand. The reason is that confined variables can only be equated to confined terms, otherwise unification fails.

Nipkow's definition of higher-order rewriting based on higher-order pattern matching was elaborated for terminating computations [13]. 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 4.3. *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) $L\gamma \xrightarrow[\beta^0]{\geq_{\mathcal{P}} F^{fun}_L} w \xleftarrow[S_{fo}\cup E_{fo}]{\geq_{\mathcal{P}} F^{cd}_L} u|_p$, and

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

Our definition coincides with plain rewriting when L is a linear, algebraic pattern without confined subterms. This allows us to treat all rules in R_{ho} uniformly, including the recursor rules if any.

Checking confluence of $\lambda\Pi$Mod We use normal rewriting with first-order rules [10], parallel rewriting at disjoint positions for functional computations, and higher-order rewriting at an arbitrary non-empty set of non-overlapping positions for higher-order rules.

Here are the rewrite steps considered in an arbitrary conversion:

$$\underset{R_{fo}}{\longrightarrow} \cup \underset{R_{fo}}{\longleftarrow} \cup \underset{S_{fo}}{\longrightarrow} \cup \underset{S_{fo}}{\longleftarrow} \cup \underset{\beta}{\Longrightarrow} \cup \underset{\beta}{\Longleftarrow} \cup \underset{R_{ho}}{\oplus\!\!\Longrightarrow} \cup \underset{R_{ho}}{\Longleftarrow\!\!\oplus} \cup \underset{E_{fo}}{\longleftrightarrow} \cup \underset{\alpha}{\longleftrightarrow}$$

We use van Oostrom's decreasing diagrams technique (rewrite diagrams suffice here [17]). Labels are pairs $\langle m, n\rangle$, of which the first component m is a natural number to prioritize the rules:

Rewrite	m	n
$u \underset{\alpha}{\longleftrightarrow}^p v$	0	0
$u \underset{E_{fo}}{\longleftrightarrow}^p v$	0	1
$u \underset{R_{fo}\,S_{fo}\,E_{fo}{\downarrow}}{\longrightarrow}^p v$	1	$u_{S_{fo}\cup E_{fo}}$
$u \underset{S_{fo}\,E_{fo}}{\longrightarrow}^p v$	1	$u_{E_{fo}}$
$u \overset{P}{\underset{\beta}{\Longrightarrow}} v$	2	parallel canonical labelling
$u \underset{i\in R_{ho}}{\overset{Q}{\oplus\!\!\Longrightarrow}} v$	3	i

We assume that the normal rewriting system (R_{fo}, S_{fo}, E_{fo}) is terminating and Church-Rosser. Checking the Church-Rosser assumption can be done using the techniques described in [10].

We need now take care of functional computations. Because first-order rules may have non-linear righthand sides, we need using rewrites at disjoint positions for β to ensure that ancestor peaks decrease. We know that the beta-rule is confluent on arbitrary terms, hence rewriting at a set of disjoint positions with beta is confluent as well: we use as second component the label that provided by van Oostrom's completeness theorem.

By the same token as above, higher-order computations need rewriting at disjoint positions. And because functional rewrites may stack redexes below that were previously disjoint, we even need higher-order rewriting at non-overlapping positions.

The label's first component allows to localize the reasoning for homogeneous peaks. We are therefore left with higher-order homogeneous peaks and heterogeneous peaks. Ancestor peaks of all kinds (but homogeneous first-order and functional) are taken care of by a new concept, *preservation*, which allows to generalize Huet's ancestor peak joinability argument. Further, confinement forbidding higher-order/functional rewrites below a confined variable of a lefthand side of rule, and non-confined variables being linear, the usual linearity arguments apply.

We are left showing that the higher-order critical pairs, as well as the critical pairs of the first-order rules inside the higher-order ones, have decreasing diagrams. All these critical pairs have been computed with MAUDE, while their decreasing diagrams have been computed by hand, since MAUDE cannot do that. These computations are summarized in a table. 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 constraints operate on the rules' *labels* indicated as a subscript of the rewriting arrow.

THEOREM 4.1. *The dependent type theory* $\lambda\Pi$Mod *equipped with a set of rules satisfying our assumptions is Church-Rosser.*

5. A Church-Rosser encoding of CCU$^{\infty}_{\subseteq}$.

The set of rules and equations R_{CIC} has been obtained with the MAUDE system [7] –using MAUDE has been instrumental in this quest–, by hiding the higher-order aspects of the rules.

To show that R_{CIC} satisfies our assumptions, we first 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}, S_{fo}, E_{fo}) and R_{ho} of respectively first-order and higher-order rewrite rules. Checking that they satisfy our assumptions is routine.

For R_{fo}, we must first show it is a terminating Church-Rosser normal rewriting system. Termination of normal rewriting in AC equivalence classes can be shown by Rubio's fully syntactic AC path ordering [15]. 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, and it can be done by the confluence checker available in MAUDE.

In order to check the critical pairs of the first-order rules inside the higher-order ones, or between the higher-order rules, we used MAUDE to compute the critical overlaps (modulo associativity, commutativity and identity) listed in the Appendix next page. MAUDE does not support confluence proofs based on decreasing diagrams, which forced us to do these computations by hand, as shown. It should be noted that MAUDE is the only available, maintained system which implements (normal) rewriting modulo associativity, commutativity and idempotency. We tried MAUDE with identity used as a rule, but were not able to obtain a confluent system that way. This sophistication has a pratical impact: the need to implement more general rewriting mechanisms in Dedukti in order to type-check Coq-proofs using universe polymorphism.

THEOREM 5.1. *The dependent type theory* $\lambda\Pi$Mod *equipped with the encoding of the cumulative hierarchy of predicative universes is Church-Rosser.*

6. Conclusion

The confluence theorem is complex. It allows mixing beta-reduction, a terminating, Church-Rosser, normal rewriting system, and higher-order rules (using higher-order pattern matching and the confined theory) that are possibly non-terminating. There is therefore a mix of traditional techniques based on termination, with van Oostrom's technique based on decreasing diagrams. In particular, we use the completeness of decreasing diagrams for beta-reductions, allowing parallel beta-rewrites at disjoint positions, in order to hide that particular confluence proof. For higher-order reductions, we use full parallel reductions with R_{ho} at non-overlapping positions, the labels being provided by the rule names. Further, confinement allows to have non-linear variables in lefthand sides of rules. Other variables must be linear in the lefthand sides of both R_{fo} and R_{ho}, and appear linearly in the righthand sides of R_{fo}.

References

[1] Claus Appel, Vincent van Oostrom, and Jakob Grue Simonsen. Higher-order (non-)modularity. In *Proc. RTA* 2010, *LIPIcs* 6, pp 17–32. 2010.

[2] Ali Assaf. A calculus of constructions with explicit subtyping. In *Proc. TYPES* 2014, *LIPIcs* 39, pp 27–46. 2014.

[3] Ali Assaf. A framework for defining computational higher-order logics. École Polytechnique, 2015.

[4] Ali Assaf and Guillaume Burel. Translating HOL to Dedukti. In *Proc. PxTP* 2015, *EPTCS* 186, pp 74–88, 2015.

[5] Frédéric Blanqui, Jean-Pierre Jouannaud, and Mitsuhiro Okada. Inductive-data-type systems. *Theoretical Computer Science*, 272:41–68, 2002.

[6] Mathieu Boespflug and Guillaume Burel. CoqInE: Translating the calculus of inductive constructions into the lambda-Pi-calculus modulo. In *Proc. PxTP*, pp 44, 2012.

[7] 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*, *LNCS* 4350. Springer, 2007.

[8] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-Pi-calculus modulo. In *Proc. TLCA* 2007, *LNCS* 4583, pp 102–117. Springer, 2007.

[9] Healfdene Goguen. The metatheory of UTT. In *Proc. TYPES* 1994, *LNCS* 996, pp 60–82. Springer, 1994.

[10] Jean-Pierre Jouannaud and Jianqi Li. Church-Rosser properties of normal rewriting. In *Proc. CSL* 12, *LIPIcs* 16, pp 350–365. 2012.

[11] Jan Willem Klop. *Combinatory reduction systems*. PhD thesis, CWI tracts, 1980.

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

[13] Tobias Nipkow. Higher-order critical pairs. In *Proc. LICS* 1991, pp 342–349. 1991.

[14] Ulf Norell. Dependently typed programming in Agda. In *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, *LNCS* 5832, pp 230–266. 2008.

[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 *Proc. ITP* 2014, Vienna Summer of Logic 2014, *LNCS* 8558, pp. 499–514. 2014.

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

[18] Vincent van Oostrom. Developing developments. *Theor. Comput. Sci.*, 175(1):159–181, 1997.

Appendix: critical pair calculations

MAUDE tells us that the input first-order specification encoding R_{CIC} is Church-Rosser, but it is not enough for our purpose: we need to further check all critical pairs involving higher-order rules and show that they have decreasing diagrams. They are listed below. The overlaps have been computed by MAUDE while the joinability calculations have been carried out by hand.

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

A solution to the ordering constraints is $p_5 > p_1 > p_2 > p_3 > p_4 > p_6 >$ other rules. This terminates the confluence verification.