Untyped Confluence In Dependent Type Theories
Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, Jiaxiang Liu

To cite this version:
Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, Jiaxiang Liu. Untyped Confluence In Dependent Type Theories: Full version. 2017. hal-01515505

HAL Id: hal-01515505
https://inria.hal.science/hal-01515505
Preprint submitted on 27 Apr 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Untyped Confluence In Dependent Type Theories
Ali Assaf, Gilles Dowek, Jean-Pierre Jouannaud, Jiaxiang Liu
Google Inc, INRIA&ENS-Cachan, Université Paris-Saclay and École Polytechnique, Tsinghua University
INRIA Project Deducteam at //dedukti.gforge.inria.fr/

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 taken from practice: a dependent type calculi to prove confluence on untyped terms. These pairs in the absence of termination properties, which are useful in

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, an essential ingredient of the type preservation proof; type preservation is needed to show that derivations issued from well-typed expressions are well-typed, an essential ingredient of the confluence proof. One can break this circularity in two ways: by proving all properties together within a single induction [13]; or by proving confluence on untyped terms first, and then 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 [5] and now Agda [24]. Current techniques for showing confluence by using van Oostrom theorem for higher-order rewrite systems [29] are restricted to 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, and 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, whose main ambition 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, 12].

In Section 6, we describe a rewrite-based encoding in $\lambda\Pi$Mod of the Calculus of Constructions with Cumulative Universes $\varPi^\infty_{\subseteq \infty}$, which uses Nipkow’s higher-order rewriting, non-left-linear rules, and associativity and commutativity. $\varPi^\infty_{\subseteq \infty}$ 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 [7] is relatively simple [9], we only allude to it here. The major difficulty when encoding $\varPi^\infty_{\subseteq \infty}$ is the treatment of universe cumulativity, which needs to be rendered explicit. Existing encodings of universe cumulativity in $\lambda\Pi$Mod have limitations. In [2], the infinite universe hierarchy is encoded by a set of function symbols indexed externally, hence infinite. The rewrite based attempt in [3] is confluent on ground terms only, restricting its use to encode type systems, like Matita, which do not include universe polymorphism. Our encoding is confluent on terms with variables, hence can support Coq’s universe polymorphism.

Our major contribution is developed in Section 5: a result reducing the Church-Rosser property of a $\lambda\Pi$Mod theory on untyped terms to typed critical pair computations, which goes far beyond the most advanced available technique based on development closed critical pairs [27], which cannot handle our example. This result is applied to the previous encoding in Section 6. It can be used more generally to show confluence of dependent type theories like those definable in Dedukti.

The main technical tool we use, recalled in Section 4, is van Oostrom’s decreasing diagrams for labelled relations, which permit to prove confluence of rewrite systems that verify a kind of local confluence property called decreasing diagram [28]: local peaks need to be joinable by rewrites whose labels are smaller, in some well-founded sense, than those of the local peak. This technique provides a modular analysis of local peaks by reflecting the various components of a rewrite system in the labels. In the case of $\lambda\Pi$Mod, we classify the rules and equations in three categories: the functional ones inherited from the $\lambda$-calculus; a set of user-defined first-order rules and equations forming a Church-Rosser, terminating, normal rewrite system [17]; and a set of user-defined higher-order rules whose left-hand sides are patterns [22]. Our definition of higher-order rewriting on untyped terms adapts Nipkow’s definition given for typed terms by replacing $\beta$-normalization with Miller’s $\beta^*$-normalization [22], which, unlike $\beta$, is terminating on untyped terms. Some $\beta$-steps that are implicit in Nipkow’s must therefore become explicit in our setting.

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 [19], which can itself be encoded in the pure lambda calculus. To eliminate this difficulty, variables having multiple occurrences in left-hand sides of user’s rules are guaranteed to operate on homogenous algebraic terms by a syntactic assumption, confinement: by ensuring
that no redex other than first-order ones may occur below a non-linear variable of a rule, confinement eliminates heterogeneous local peaks that would not have decreasing diagrams otherwise, like those occurring in Klop’s fixpoint combinator example. Confinement appears therefore to be a crucial original concept, which is built directly in the type system of $\lambda$IMod.

Because of right non-linearities of the user’s rules, and because $\beta$-rewriting may stack redexes that were previously disjoint, we analyze in Section 3 the local peaks of the relation union of first-order rewriting, $\beta$-rewriting at a set of parallel positions, and higher-order rewriting at a set of orthogonal positions. A related relation, multi-step rewriting, that extends Tait and Martin-Löf’s definition of parallel reductions initially designed for the $\lambda$-calculus [6], is used by van Oostrom for, in particular, analyzing confluence in first-order orthogonal systems via permutation equivalence [27]. Both relations use labelling techniques for describing how the relation can be seen as a succession of elementary steps. Unlike van Oostrom’s, our labelling technique is external to the terms involved in the rewriting step, hence makes orthogonal steps. Unlike van Oostrom’s, our labelling technique is external to the terms involved in the rewriting step, hence makes orthogonal steps. Moreover, it allows to solve local peaks, whose two steps use rules of different categories.

The label’s second component is used to solve local peaks, whose two steps use rules of different categories. Confining confluence in first-order orthogonal systems via permutation equivalence [27].

The head of a term is its ordinal symbol. Terms of the form $\lambda x : M.N$ are abstractions. Terms of the form $\Pi x : M.N$ or $\Pi y : C.A$ are products. Terms of the form $(M N)$ are applications. Application associates to the right. We write $(u \, v)$ for $\ldots (u \, v_1) \ldots v_n$. Terms of the form $f(\overline{v})$, with $f \in \Sigma$, are algebraic-headed, they must satisfy $\Xi[f] = \text{arity}(f)$. Terms built solely from the signature and variables are called algebraic. Algebraic terms built from the sub-signature $\Sigma_{\text{cd}} \cup \Sigma_{\text{cd}}$ are called algebraic.

Confined terms are particular first-order algebraic terms. As usual, we write $f$ instead of $f()$ when $\text{arity}(f) = 0$.

Note that $\lambda$IMod’s type constructors $*$ and $\sqcup$ are constants from $\Sigma$, regardless of their specific role and despite the fact that $\Sigma$ is called the user’s signature.

We use: $\text{Var}(M)$ and $\text{BV}(\text{Var}(M))$ for the sets of variables of $M$ occurring free and bound respectively; $z$ for fresh variables; $=$ for the syntactic equality of terms, $\equiv$ for definitional equality.

**Typing rules**

The use of $\lambda$IMod’s typing rules is marginal here. However, they are slightly different from those given in [12], because of confinement.

There are two forms of judgements, $\Gamma \vdash M : A$ meaning that the term $M$ has type $A$ in the context $\Gamma$, and $\Gamma \vdash \vdash$ meaning that the context $\Gamma$ is well-formed. In a context, variables from $\mathcal{Y}$ must be given a confined type. Those from $\mathcal{X}$ must not. Here are a few important rules, see [12] for the (classical) others:

**JP:** Give all typing rules here.

$\Gamma \vdash A : *; y \not\in \mathcal{Y}; y, A \text{ confined (resp. } x \not\in \Gamma; x, A \text{ non-confined)}$

$\Gamma, y : A \vdash (\text{resp. } \Gamma, x : A \vdash)$

$\Gamma \vdash \Pi x_1 : A_1, \ldots, \Pi x_n : A_n, B \in \Sigma$

$\Gamma \vdash M_1 : A_1, \ldots, \Gamma \vdash M_n : A_n \{x_1 \mapsto A_1, \ldots, x_n \mapsto A_n\}$

$\Gamma \vdash \{f(M_1, \ldots, M_n) = B\} \{x_1 \mapsto A_1, \ldots, x_n \mapsto A_n\}$

$\Gamma \vdash \{x : A \vdash B \mid x, A, B \text{ not confined}\}$

$\Gamma \vdash \lambda x : A.M : A, A.M : A.B$

$\Gamma \vdash M : \Pi x : A.B \vdash N : A \text{ non-confined}$

$\Gamma \vdash \Pi x : A.M : N \vdash \Pi x : A.B \text{ non-confined}$

$\Gamma \vdash M : N \vdash B(x \mapsto N)$

$\Gamma \vdash N : A \vdash B : * \quad A \equiv B$

$\Gamma \vdash M : B$

As in all extensions of LF, the conversion $\equiv$ is generated by all rewrite rules, whether built-in or user-defined.

This type system does enforce a stratification over the sublanguages of objects, types and kinds, objects being typed by types, types by kinds, and kinds (among which $\ast$ by $\sqcup$, see [12]. User-defined symbols must come along with a type or a kind. Symbols in $\Sigma_\text{cd}$ must either be type constants (of type $*$), or first-order function symbols, of type $\Pi x_1 : s_1, \ldots, s_n, s : A$, where $n$ is the arity of $f$ and $s_1, \ldots, s_n, s$ are type constants.

A major change wrt traditional type systems, including that of today’s Dedukti, are the rules for typing abstractions and applications: terms of a confined type cannot be abstracted over or argument of an application, nor applied since confined types are constants from $\Sigma_{\text{cd}}$. Note that confinement assumptions are not needed when the rules are used bottom-up: they result then from the definition of (untyped) terms. They are needed, on the other hand, when the rules are used top-down. They are also needed if, as is actually customary, the rules serve defining untyped terms and typed terms at the same time, without any additional reference to a grammar of untyped terms.

The following assumption ensures then that the set of typable confined terms is the set of typable terms of a confined type:

**Assumption 1:** (i) $\forall f : \Pi x : \pi.s : \Sigma_{\text{cd}} \cup \Sigma_{\text{ho}} \quad (s \not\in \Sigma_{\text{cd}})$

(ii) $\forall f : \Pi x : \pi.s : \Sigma_{\text{cd}} \quad (s, \pi \in \Sigma_{\text{cd}})$
Positions. Positions in terms are words over the natural numbers, using $·$ for concatenation, $λ$ for the empty word, $≥_P$ for the prefix order on positions (below), $≤_P$ for its inverse (above), $>_P$ for its strict part, and $p ≤_q$ for $\neg(p >_P q)$ (parallel or disjoint). We use $P$ for the set $\{p·q : p, q ∈ Q\}$. Identifying a position $p$ with the singleton set $\{p\}$, we use: given a set of parallel positions $P$ and an arbitrary set of positions $Q$, $Q >_P P$ for $∀q ∈ Q \exists p ∈ P (p >_P q)$; $Q_{\min}$ for the largest subset of minimal positions of $Q$; given a term $M$, $\text{Pos}(M)$, $\text{FP}(M)$, $\text{VP}(M)$ and $\text{VP}(\text{Pos}(M))$ for the following respective sets of positions of $M$: all positions, the non-variable positions, and the positions of all variables, and linear variables respectively. A term $M$ is linear if $\text{VP}(\text{Pos}(M)) = \text{VP}(M)$. We also denote by: $[M]_p$, the termset of $M$ at position $p ∈ \text{Pos}(M)$; $[M]_p$, where $P ⊂ \text{Pos}(M)$ is a list of parallel positions $\{N\} = \{P\}$, the term obtained by replacing in $M$ all subterms at positions in $P$ by the corresponding term in the list of terms $\vec{N}$, using the simplified form $M[N]_p$ in case all terms in $\vec{N}$ are equal to $N$. We use $\text{Var}(M) (\text{BV}(\text{Var}(M)))$, for the sets of variables of $M$ occurring free (bound).

Substitutions. Substitutions are sort-preserving, capture-avoiding homomorphisms, written $σ \triangleq \{z_1 \mapsto M_1, . . . , z_n \mapsto M_n\}$, of domain $\text{Dom}(σ) \triangleq \{z_1, . . . , z_n\}$ if $∀i (z_i ≠ M_i)$, such that $M_i$ is a confined term iff $z_i$ is a confined variable. The substitution $σ$ is algebraic-headed if all $M_i$‘s are algebraic-headed. We use post-fixed notation for the application of a substitution $σ$ to a term $t$ or a substitution $σ$, as in $σt$ and $στ$, called instances by $τ$ of $σ$ and $σ$. Given terms $s, t, u, v, w$, computing the substitution $γ$, whenever it exists, such that $t = sγ$, in which case $t$ is an instance of $s$, or $uγ = v$, in which case $γ$ is a plain unifier of $u, v$, is called plain pattern matching and plain unification respectively. Unification is sorted: confined variables must be replaced by confined terms. We use $\text{SU}(u, v)$ for the set of plain unifiers of $u, v$. It is well known that a most general unifier $γ$ exists for any two unifiable terms, that is $∀γ ∈ \text{SU}(u, v) \exists p (γ = σp)$. The word plain may be omitted.

Given a term $u$ and substitution $σ$ such that $\text{Dom}(σ) \cap \text{BV}(u) = \emptyset$, we define the term $u[σ] \triangleq u[x_1 σ][o_1] . . . [x_n σ][o_n]$, where $O_i \triangleq \{x : u[x_1 σ][o_1] . . . [x_n σ][o_n]\}$. The associated operation on terms is not a homomorphism. Called variable-replacement, it is nothing but a convenient notation for multiple replacements at positions of free variables in a term. It is of course not compatible with positions of bound variables in the term. Note that it coincides with instantiation when $\bigcup_{σ ∈ \text{Dom}(σ)} \text{Var}(σ) \cap \text{BV}(u) = \emptyset$.

Computation rules. In type theory, computation is based on rewrite rules. We use $→$ for a rewriting relation on terms, $←$ for its inverse, $→$ for its symmetric closure, $↔$ for its reflexive transitive closure, $⇔$ for its symmetric, reflexive, transitive closure called convertibility, $⇒$ for its parallel closure at a set of parallel positions, and $⇔$ for its nested closure to be defined later. An element $s$ such that $s \rightarrow t$ for no $t$ is in normal form. A normal form for $t$, written $t\uparrow$ is a term $s$ in normal form such that $t \rightarrow s$.

Definition 2.2. A plain rewriting system is a set $R$ of proper rules. A term $s$ (plain) -rewrites to $t$ at position $p$, written $s \rightarrow^R p t$ if $s|_p = τ[γ]$ for some rule $r \rightarrow τ$, and $t = σ[γ]|_p$. An equational rewriting system is made of a plain rewriting system $R$ and a set $E$ of regular equations. A term $s$ rewrites (modulo $E$) to a term $p$, written $s \rightarrow^E p t$ (or $s \rightarrow^E t$), if $s|_p = τ[γ]$ for some rule $r \rightarrow τ$, and $t = σ[γ]|_p$. An equational rewriting system $(R, E)$ is terminating if the relation $\rightarrow^E$ has no infinite chain.

A normal rewriting system is made of a plain rewriting system $R$ and a terminating equational rewriting system $(S, E)$ whose rules are called simplifiers. Let $S \subseteq S\uparrow^{-1} E$. A term $s$ (normal) -rewrites to $t$ at position $p$, written $s \rightarrow^{S\uparrow E} p t$ if $s|_p = τ[γ]$ for some rule $r \rightarrow τ$, and $t = σ[γ]|_p$. We write $s \rightarrow^{S\uparrow E} t$ if $s \rightarrow^{S\uparrow E} p t$ for some $p ∈ \text{Pos}(s)$.

The position or set of positions at which a rewrite takes place may be omitted, or replaced by a condition it satisfies, like $p ≤_q q$ or $P ≥_q q$. We may record various information in the upper index of the rewriting arrow. This applies to the normal form notation too.

A first essential remark is that all kinds of rewriting we just defined preserve our two sorts of confined terms and non-confined terms, under our definitions of a rewrite rule and a substitution. A second is that normal rewriting is not stable by substitution instance, since terms to be rewritten need to be normalized beforehand at the rewritten position. The situation is even worse with variable replacement which will always require dedicated lemmas.

Definition 2.3. Given a normal rewriting system $(R, S, E)$, we call conversion: a pair $(s, t)$ such that $s \leftrightarrow (s, t)$ is the peak: a triple $(s, u, t)$ such that $s \leftrightarrow t$, $(R, S, E)$ is the local peak: a triple $(s, u, t)$ such that $s \leftrightarrow (s, u, t)$ where $i : l \rightarrow r, j : q \rightarrow d ∈ R, k : l \rightarrow r ∈ S$, and $l$ is an ancestor of $q$ if $p ≤_q q$; $i : j : q \rightarrow d ∈ R, k : l \rightarrow r ∈ S$. A local peak or cliff is disjoint if $p ≥_q q$; $\text{Mod}(s) \cap \text{Var}(t) = \emptyset$ and critical if $q ∉ \text{FP}(t)$ (the latter two comparisons may make no sense when $S ∪ E = \emptyset$).

A normal rewriting system $(R, S, E)$ is Church-Rosser if every conversion $(s, t)$ is normal-joinable: $s \rightarrow^R (s, t)$, and strongly normalizing $(SN, or terminating)$ if $R \rightarrow^R (R, S, E)$ terminates in $E$-equivalence classes of terms, i.e., $\rightarrow^E (R, S, E)$ terminates.

Plain rewriting and rewriting modulo are particular cases of normal rewriting when $S = E = \emptyset$ and $S = \emptyset$ respectively, allowing us to factor out many definitions. Cases where normal rewriting is necessary are rewriting modulo associativity, commutativity and identity [16], implemented in the programming system MAUDE [11], and higher-order rewriting [23]. The general framework is introduced in [17], improving [2]. Plain rewriting enjoys two key properties implying that disjoint and ancestor rewriting peaks are always joinable [14]. More generally, given two normal rewriting systems $(R, S, E)$ and $(R′, S′, E′)$, we have the following easy joinability properties:

- $\rightarrow_{R,S,E} \supseteq \rightarrow_{R′,S′,E′}$ (DP)
- $\rightarrow_{R,S,E} \supseteq \rightarrow_{R′,S′,E′}$ (AP)

Where $O, P$ are sets of parallel positions.
Note that (DP) would not hold if the result of a normal rewriting step were normalized up to its redex, as in [21].

(AP) specializes to (LAP) when \( P \not\supseteq Q \). (LAP) generalizes to rewriting at a set of parallel positions \( P, Q \) such that \( P \not\supseteq Q \). (LAP) generalizes to rewriting at a set of parallel positions such that \( Q \supseteq P \cdot \lambda \mathcal{P} \mathcal{O}(l) \). Both are proved from (DP) and (LAP) by induction on \([P]+[Q]\) and \([Q]\) respectively.

(AP)/LAP use plain rewriting at position \( p \). Using instead rewriting modulo or normal rewriting would not give a valid property, since the modulo part below could interact with the rewrite at \( q \). There are cases, however, where there are no interactions, in particular when the modulo part cannot apply above \( q \) and does not need to apply below \( q \), in which case we say that rewriting at \( p \) preserves the redex at \( q \).

**Definition 2.4** (Preservation). Given a normal rewrite system \((R, S, E)\), a normal rewrite \( u \xrightarrow{p} v \) preserves a position \( q \in \mathcal{P} \mathcal{O}(u) \) such that \( q > p \) if \( u[z]_p \xrightarrow{p} w[z]_p \) \( \mathcal{O} \) for some set of parallel positions \( O \) \( s = w[v]_p \mathcal{O} \) and \( V \in \mathcal{O} (v|_p \mathcal{E} = E u|_q) \).

Note that the occurrences of \( z \) in \( w \) need not be replaced by exactly the same term to get \( v \). Note also that \( u|_p \) and \( v|_p \) are in \( S E \cdot \mathcal{O} \)-normal form by definition of normal rewriting. This explains that \( v|_p \mathcal{O} \) and \( v|_p \mathcal{E} \) may be different if \( p \neq o \), and \( E \)-equal to \( u|_q \) rather than \((S \cup E)\)-equal to \( u|_q \).

The absence of interactions above \( q \) results in practice from signature considerations that forbid any application of an equation or simplifier on the path from \( p \) to \( q \), while the absence of interactions below \( q \) results from the fact that the path from \( p \) to \( q \) goes through a linear variable of the left-hand side of rule used at \( p \).

The linear ancestor peak property becomes:

\[
v \xrightarrow{p} u \xrightarrow{q} w \subseteq \xrightarrow{p} o \xrightarrow{q} \quad (LAP)
\]

where \( O \) is a set of parallel positions.

**Proof.** By definition of preservation, \( u[z]_q \xrightarrow{p} s[z]_p \mathcal{O} \) and \( \mathcal{O} \in O \) \( (v|_p \mathcal{O} \xrightarrow{p} E u|_q) \), hence \( v|_p \mathcal{E} = g \cup E \mathcal{O} \mathcal{E} \) \( w|_q \), and thus \( v|_p \mathcal{O} \mathcal{E} = g \cup E \mathcal{O} \mathcal{E} \) \( w|_q \), hence \( v \xrightarrow{p} s|w[n]_p \mathcal{O} \). Now \( w = u[w|_q] \), hence \( w \xrightarrow{s} s[w[n]_p |o} \) and we are done.

(LAP) generalizes to rewrites at a set of parallel positions provided \( u \xrightarrow{p} v \) preserves all positions in that set. In this context it is sometimes useful to write \( p \xrightarrow{o} q \) instead of \( p \xrightarrow{o} q \).

We now turn to a last classical property of rewrites:

**Definition 2.5.** Given a (rewrite) relation \( \equiv \) and an equational theory \( = \), we say that \( \equiv \) commutes over \( = \). If for all \( x, y, z \) such that \( x \equiv y \), then there exists \( s \) such that \( u \equiv s \equiv w \).

### 2.1 Functional computations in \( \lambda \text{I} \text{M} \text{od} \)

\( \lambda \text{I} \text{M} \text{od} \) comes equipped with two rewriting schemas, \( \alpha \)-conversion and \( \beta \)-reduction:

\[
\lambda x : U.M = \lambda z : U.M \{ x \mapsto z \} \quad \text{if } z \text{ fresh or } z = x \quad (\alpha)
\]

\[
\Pi x : U.V = \Pi z : U.V \{ x \mapsto z \} \quad \text{if } z \text{ fresh or } z = x \quad (\alpha)
\]

\[
(\lambda x : U.M) N \rightarrow M \{ x \mapsto N \} \quad (\beta)
\]

Substituting \( N \) to each occurrence of \( x \) in \( M \) involves renaming the bound variables of \( M \) away from those of \( N \) in order to avoid captures: \( (\beta) \) rewrites modulo \( \alpha \), and is accordingly denoted by \( \xrightarrow{\beta} \), (in short, \( \xrightarrow{\beta} \)). Note that \( \alpha \) and \( \beta \) preserve sorts because, on the one hand a confined term cannot be abstracted over, and on the other hand, a term in argument position of an application cannot be confined. Hence, a reduction like \( (\lambda z.z) u \xrightarrow{\beta} u \) with \( u \) confined does not exist in the untyped syntax.

Major properties of \( \beta \)-rewrites are that: \( \xrightarrow{\beta} \), hence \( \equiv \), is Church-Rosser modulo \( \alpha \); commutes over \( = \), as well as over any equational theory generated by a set of algebraic equations whose non-linear variables are confined; and enjoys (LAP).

But since disjoint redexes in a term may become stacked residuals after \( \beta \)-rewriting, (LAP) does not extend to parallel rewriting.

A particular case of repeated \( \beta \)-rewrites plays a key role here:

\[
(\lambda x.u \pi) \xrightarrow{\beta} u[\pi \mapsto \pi] \quad (\beta^0, \text{as dubbed by Miller})
\]

\( \beta^0 \)-rewrites have a key property for our setting: by decreasing the size of terms, whether typed or not, they are confluent and terminating on all terms. Further, they need no \( \alpha \)-conversion and enjoy the generalized form of (LAP).

As anticipated, we need to characterize the behavior of \( \beta \)-reductions in presence of variable-replacement:

**Lemma 2.6.** Let \( u = (\lambda x.s \ t) \) be a term and \( \sigma \) a substitution at \( x \notin \text{Dom}(\sigma) \). Then, \( u[\sigma] \equiv (\lambda x.s \ t)[\sigma] \equiv ((\lambda x.s)[\sigma] \ t[\sigma]) = (\lambda x.s[\sigma] \ t[\sigma]) \) by assumption that \( x \notin \text{Dom}(\sigma) \). Hence \( u[\sigma] \equiv (\lambda x.s[\sigma] \ t[\sigma]) \).

**Proof.** Use \( u[\sigma] = (\lambda x.s \ t)[\sigma] = ((\lambda x.s)[\sigma] \ t[\sigma]) = (\lambda x.s[\sigma] \ t[\sigma]) \) by assumption that \( x \notin \text{Dom}(\sigma) \). Hence \( u[\sigma] \equiv (\lambda x.s[\sigma] \ t[\sigma]) \).

### 2.2 First-order computations in \( \lambda \text{I} \text{M} \text{od} \)

We assume given three sets of first-order algebraic rules: rules \( R_{fo} \), simplifiers \( S_{fo} \) and equations \( E_{fo} \). Simplifiers and equations have confined lefthand and righthand sides. Rules in \( R_{fo} \) are confined/non-confined depending on the sort of their both sides.

**Assumption 2:** \( (R_{fo}, S_{fo}, E_{fo}) \) is a terminating, Church-Rosser, normal rewriting system.

This assumption isolates the Church-Rosser property of \( (R_{fo}, S_{fo}, E_{fo}) \). It implies in particular that \( (S_{fo}, E_{fo}) \) is a terminating Church-Rosser system. To check it, we need [17]:

**Assumption 3:** \( E_{fo} \)-unification and \( E_{fo} \cup S_{fo} \)-unification are finite and complete.

A common example is provided by the two equations of associativity and commutativity for \( E_{fo} \) and the identity rule for \( S_{fo} \).

In order to ensure preservation of other rewrites, we also need:

**Assumption 4:** Variables in rules from \( R_{fo} \) are either

(i) confined or (ii) linear (on both sides).

This assumption is automatically satisfied by the rules from \( R_{fo} \), which are confined, since they have no non-confined variables.

By definition of normal rewriting, firing rules of \( R_{fo} \) requires pattern matching their lefthand side \( l \) modulo \( S_{fo} \cup U_{E_{fo}} \). Since, however, the non-confined variables of \( l \) are linear, pattern matching can be assumed plain below them. It follows that first-order rewrites with rule \( l \rightarrow r \) preserve non-confined redexes, if any, below \( l \).
We end up with instantiation and variable replacement by normal substitutions, that is in normal form wrt \( (S_d,E_d) \).

**Lemma 2.7.** Let \( u \stackrel{q}{\rightarrow} v \) and \( \sigma \) a normal substitution st \( \text{Dom}(\sigma) \subseteq X \). Then \( u \sigma \stackrel{q}{\rightarrow} v \sigma \) and \( u[\sigma] \stackrel{q}{\rightarrow} v[\sigma] \).

**Proof.** Instantiation and variable replacement by non-confined terms do not impact \((S_d,E_d)\)-normal forms. \( \square \)

**Example 2.1.** (extracted from Section 6). Consider the confined signature made of the constants 0, 1 and the binary symbols \( +, \cdot, \max, \uparrow \) and \( \ll \). Letters \( i, j \) denote confined variables while \( a \) is non-confined.

Let \( E_d \) be made of associativity and commutativity for \( + \), \( S_d \) be the identity for \( + \), and \( R_0 \) be the following set of rules:

\[
\begin{align*}
1: & \quad \text{max}(i, i + j) \rightarrow i + j \\
2: & \quad \text{max}(i + j, j) \rightarrow i + j \\
3: & \quad \text{rule}(i, 0) \rightarrow 0 \\
4: & \quad \text{rule}(i, j + 1) \rightarrow \max(i, j + 1) \\
5: & \quad \ll(0, a) \rightarrow a \\
6: & \quad \ll(i + 1, a) \rightarrow \ll(i, \ll(i, a))
\end{align*}
\]

Assumptions 3 and 4 are satisfied, since, in particular, the variable \( a \) appears linearly in rules. For Assumption 2, we must show termination in \( AC \) equivalence classes and check that the ACZ-critical pairs are joinable by rewriting. Termination can be achieved easily by Rubio’s fully syntactic \( AC \) path ordering [25]. This order works very 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 rules whose status must be lexicographic while the precedence can be \( \ll \gg \gg \gg \), \( \ll \gg \gg \gg \), \( \ll \gg \gg \gg \), \( \ll \gg \gg \gg \). Routine calculations then do the job. Checking the ACZ-critical pairs routine too, is only a trivial one between the two max rules.

### 2.3 Higher-order computations in \( \lambda \Pi \text{Mod} \)

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 algebraic rules having a non-linear non-confined variable (such as \( x \cdot 0 \rightarrow 0 \) with a non-confined variable \( x \) on left lost on the right).

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 [22], or algebraic expressions as in recursive rules [7] or non-regular algebraic rules as above. Miller’s patterns require using some form of higher-order pattern matching to fire rules, while first-order pattern matching suffices for algebraic lefthand sides.

**Definition 2.8 (Pattern [22]).** A pattern is a \( \Sigma_{ho} \)-headed term \( L \) in \( \beta \)-normal form whose all non-confined free variables occur in pre-redexes, that is, in maximal non-applied subterms of the form \( (X \, F) \), \( n \geq 0 \), where \( X \) is a vector of \( n \) distinct variables bound above in \( L \). A pre-redex is trivial if reduced to a variable.

The lefthand side \( \text{rec}(S(x), X, Y) \) of one of the two classical recursive rules over natural numbers, is a pattern: algebraic terms are indeed patterns whose variables are not applied, possibly because they are first-order (\( X \) above may be, \( Y \) is not). In particular, all first-order algebraic terms are patterns in our sense.

Unlike in Miller’s original definition [22], patterns must be \( \Sigma_{ho} \)-headed and need not be in \( \eta \)-long form: the lefthand sides of the above recursive rule would not be a pattern otherwise. More generally, the lefthand side of any recursive rule as defined in [7], is then a pattern. As already pointed out in [17], the notion of pattern is quite robust. We could indeed require that patterns are in normal form for \((R_0, S_d, E_d)\). This would be practically relevant, but is not needed here.

The main property of a pattern is that it generates by instantiation very specific \( \beta \)-redexes and reducts at predictable positions. Consider the subterm \( (X \, F) \) at position \( p \) in a pattern \( L \), and a substitution \( \sigma = \{ X \mapsto \lambda z u \} \). Then, \( (\text{rec})(u)[\sigma] \rightarrow_{\beta_{\sigma}} u[\tau[\sigma]] \). So, the instance of a pattern by such a substitution \( \beta \)-reduces to a term obtained by replacing the subterms of the form \( (X \, F) \) by terms of the form \( u[\tau[\sigma]] \) where \( u \) is a term which does not contain any free occurrence of variables in \( \tau \).

**Assumption 5:** Lefthand sides of higher-order rules are patterns whose all non-linear variables are confined.

Note that linear variables can be either confined or non-confined. The following assumption simplifies the coming confluence proof and diagrams and is satisfied by our example:

**Assumption 6:** Righthand sides of higher-order rules are not (i) variables, nor (ii) abstractions.

Nipkow’s definition of higher-order rewriting based on higher-order pattern matching was elaborated for terminating computations [23]. It operates on terms in \( \beta \)-normal form and \( \beta \)-normalizes the result. Our definition for non-terminating computations operates instead on redexes in \( \beta^0 \)-normal form and \( \beta^0 \)-normalizes their contractum: higher-order rewriting is seen here as normal rewriting with \((R_{ho}, \beta^0 \cup S_d, \alpha \cup E_d)\), inheriting its properties. In particular, as for normal rewriting, a term need not be in normal form wrt \((\beta^0 \cup S_d, \alpha \cup E_d)\) in order to be rewritten at some position \( p \). It needs to be in normal form up to \( p \) only. This subtle difference with Nipkow’s definition is instrumental for enjoying (DP) and (LAP).

Let therefore \( S = \beta^0 \cup S_d, E = \alpha \cup E_d, \) and \( SE = S \cup E \).

**Lemma 2.9.** The equational rewriting system \((S, E)\) is Church-Rosser and terminating.

**Proof.** We interpret a higher-order term by the pair made of the multiset of its confined subterms first, and then its size. Confined terms are compared in the associated termination order, and sizes with the natural order on natural numbers. Termination follows from the facts that \( \beta^0 \) decreases the size of terms, and the order on confined terms is stable by instantiation, hence by \( \alpha \)-conversion. Since critical peaks and cliffs are those of \((S_d, E_d)\), the Church-Rosser property follows. \( \square \)

This justifies our intuition of higher-order rewriting being an instance of normal rewriting. We say that \( u \) is normal if \( u = u[\xi_{SE}] \).

We need naming positions of variables in a pattern:

**Definition 2.10 (Fringe).** The fringe \( F_L \) of a higher-order pattern \( L \) is made of three sets of parallel positions: its plain fringe \( F_L^{\text{plain}} \) is \( \{ p \in F_{\text{Pos}}(L) : L|_p \) is a trivial pre-redex \}, its functional fringe \( F_L^{\text{fun}} \) is \( \{ p \in F_{\text{Pos}}(L) : L|_p \) is a non-trivial pre-redex \}, and its confined fringe \( F_L^{\text{cd}} \) is \( \{ p \in F_{\text{Pos}}(L) : L|_p \) is a maximal confined term \}.

Higher-order rewriting can now be formulated “à la Nipkow”:

**Definition 2.11.** Given a term \( u \) normal below \( p \) in \( \text{Pos}(u) \) and a higher-order rule \( i : L \rightarrow R \) in \( R_{ho} \), \( \forall \text{Var}(L) \land \forall \text{Var}(u) = \emptyset \), we say that \( u \) rewrites with \( i \) at \( p \), written \( u \rightarrow_i^p \) or simply \( u \rightarrow_i \), iff

\[
\text{if } u \uparrow_p \Rightarrow q \cdot F_L^{\text{cd}}_{E_d} = \big( \text{rec}(S(x), X, Y) \big) \ll \big( L \big) \| \ll \big( F_L^{\text{cd}}_{E_d} \| v \big) \quad \text{and } v = u[\xi_{SE}] \| .
\]
Note that all confined steps apply below the confined fringe: confined subterms of a non-confined term $X \gamma$ remain identical in $X \gamma$ and $u$. Let us then consider an algebraic pattern $L$. Since there are no applications in $L$, then $F^{\text{conf}}_L = \emptyset$ and $(L \gamma) \downarrow = L \gamma \uparrow_{(\alpha \beta \gamma) E_d}$. If $L$ is the left-hand side of a recursive rule, then $u_{L} = L \gamma$ as usual, hence higher-order rewriting coincides with plain rewriting in this case. This allows us to treat all rules in $R_{ho}$ uniformly, including recursive rules if any.

Firing a higher-order rule involves two distinct algorithms — higher-order pattern matching and pattern matching modulo $S_d \cup E_d$ — taking place below the functional fringe of the left-hand side pattern $L$ for the first, and below its confined fringe for the second. Since no position can be below both fringes at the same time, this makes firing higher-order rules modular with respect to each matching algorithm.

This is true of unification too: $S_d \cup E_d \cup \beta^0 \cup \alpha$-unification of two patterns reduces to higher-order unification of patterns on the one hand, and $S_d \cup E_d$-unification of confined terms on the other. The reason is that confined terms can only be equated to other confined terms, otherwise unification must fail. This is yet another, very practical, implication of our notion of confined expressions.

Since $\beta^0$-normalization occurs below $p$ only, normal rewriting is monotonic. Stability, on the other hand, requires normalization:

**Lemma 2.12.** Let $u$ be a term and $\sigma$ a substitution, both normal, at $u \overset{p}{\rightarrow}_{R_{ho}} v$. Then $u\sigma|_{S_d \cup E_d} \overset{p}{\rightarrow}_{R_{ho}} v\sigma|_{S_d \cup E_d}$ and $u[\sigma]^p|_{S_d \cup E_d} \overset{p}{\rightarrow}_{R_{ho}} v[\sigma]^p|_{S_d \cup E_d}$.

Usual stability is recovered for a large class of substitutions:

**Definition 2.13.** A substitution $\sigma$ is preserving if $\forall x \in \text{Dom}(\sigma), x$ is not confined, $\sigma x$ is normal, and is not an abstraction.

Algebraic-headed normal substitutions are therefore preserving.

**Lemma 2.14.** Let $u$ be a normal term and $\sigma$ a preserving substitution. Then $u\sigma|_{S_d \cup E_d} \overset{p}{\rightarrow} R_{ho}$ and $u[\sigma]^p \overset{p}{\rightarrow} R_{ho}$.

**Proof.** Since $x\sigma$ is not confined, the confined subterms of $u\sigma$ and $u[\sigma]^p$ are those of $u$ or $x\sigma$, hence are normal. And since $x\sigma$ is not an abstraction, no $\beta^0$-redex, hence no $\beta^0$-redex, can be created.

In the sequel, we manage, most of the time, to use preserving substitutions so that unwanted normalization steps do not pop up along computations. In particular, preserving substitutions allow us to recover important properties of rewriting modulo that are not true of normal rewriting in general:

**Corollary 2.15 (Stability).** Let $u \overset{p}{\rightarrow}_{R_{ho}} v$ and $\sigma$ a preserving substitution. Then $u\sigma|_{S_d \cup E_d} \overset{p}{\rightarrow}_{R_{ho}} v\sigma|_{S_d \cup E_d}$ and $u[\sigma]^p \overset{p}{\rightarrow}_{R_{ho}} v[\sigma]^p$.

**Lemma 2.16.** Let $u$ normal such that $u \overset{p}{\rightarrow}_{R_{ho}} v$. Then $v$ is normal.

**Proof.** Let $w = u[z]_{\sigma}$ for some fresh variable $z$, and $\sigma(z) = u_{i,j}$. Hence $u = w[\sigma]$. By assumption, $w$ and $\sigma(z)$ are normal. Let now $\tau(z) = w_{i,j}$. By definition of higher-order rewriting, $\tau(z)$ is normal. Since rules are sort preserving, $\tau(z)$ is non-confined. Further, $\tau(z)$ cannot be a variable nor an abstraction by Assumption 6(ii). Hence $\tau$ is preserving and $v = w[\tau]$ is normal by Lemma 2.14.

**Example 2.2.** Here is a rule (in untyped format) which is used in our encoding of $CC\mu^e_{\alpha\lambda}$ described at Section 6:

\[
\begin{align*}
10 : & \quad \pi(i, i + j + 1, a, \lambda x, \tau(i + j, (b x))) \\
& \quad \rightarrow \overset{p_4}{\tau(i + j, \pi(i, i + j, a, b))}
\end{align*}
\]

We see that rule 10 satisfies all our assumptions, in particular the non-confined variables $a, b$ occur linearly on both sides of the rule.
3. Orthogonal higher-order rewriting

A parallel rewrite step is a sequence of reductions at a set $P$ of parallel positions, (DP) ensuring that the result does not depend upon a particular sequentialization of $P$. An orthogonal rewrite step is a sequence of reductions at some set $P$ of positions, both parallel and nested, (DP) and (LAP) ensuring that the result does not depend upon a particular sequentialization of $P$. Our definition is based on splitting a term $s$ into a term $u$ and a preserving substitution $\sigma$ such that $s = u[\sigma]$, therefore ensuring that (DP), (LAP) and (Perm) apply. Then $u$ is rewritten at some set of orthogonal positions $O$, while $\sigma$ is rewritten at a family of orthogonal positions $Q = \{ Q_x \}_{x \in \text{Dom}(\sigma)}$ indexed by the domain of the substitution $\sigma$. The term $u$ and the capturing substitution $\sigma$ are not unique, provided they define the same term $u[\sigma]$ and sets of positions in it. The fact that the resulting term is then the same is a major property of orthogonal rewriting, and expresses the fact above that it does not depend upon a particular sequentialization of its set of positions. A particular case of course obtained when $O$ is a set of parallel positions and $Q$ is the emptyset, or vice versa, resulting then in parallel rewriting. An important attribute of our formulation is that no hypothesis is made on the underlying rewrite relation, which can be plain, modulo or normal, and, more precisely, higher-order as is the case here. Another distinctive feature is that our relation is ternary: the orthogonal rewriting arrow is decorated with a label that internalizes the rewrite history of its construction.

The idea of orthogonal rewriting appears in the literature under at least two different names, parallel reductions and multi-step rewriting. Parallel reductions, defined by induction on the term structure, were introduced by Tait and Martin-Löf to show confluence of the pure $\lambda$-calculus [6]. Multi-step rewriting generalizes the construction for both concrete and abstract rewriting relations. All these generalizations are extensively studied in [27], where they are used for analyzing orthogonal rewrite relations as well as, more generally, orthogonal rewrite steps of non-orthogonal rewrite relations, whether operating on first-order terms, higher-order terms or term-graphs. The idea of labelling rewrites is not new either, although the major trend in the literature is to incorporate the history of reductions in the rewritten term, rather than making the relation ternary, by decorating the rewrite system itself instead of the rewrite relation. We refer to [27] for a precise account of the literature and extensive bibliographic notes.

A major trait of both orthogonal and multi-step rewriting is that the underlying rewrite relation need not be orthogonal. Rewrite rules may have critical pairs, as well as repeated variables in their left-hand sides. Only the sequence of steps used in an orthogonal or multi-step rewrite needs to be orthogonal. The fact that it is still possible to analyze the Church-Rosser property of the underlying relation by studying its orthogonal extension relies on two properties: confinement, which takes care of repeated variables in left-hand sides of the rules, and the internalization of its construction, via a label in the case of orthogonal rewriting, that appears to be instrumental, as we shall see, to factor out critical peaks from two diverging orthogonal rewrite steps.

**Definition 3.1.** Orthogonal rewriting with higher-order rule $i$ at positions in $O$, written $\stackrel{\scriptscriptstyle O}{\Rightarrow}_i$, is the smallest relation containing $\stackrel{\scriptscriptstyle O}{\Rightarrow}_i$, and closed under vertical product (in short, product), written $\sigma \overset{P \otimes_o Q}{\Rightarrow}_i \tau$, defined as

(i) $u \overset{P}{\Rightarrow}_i v$;

(ii) $\sigma$ is a preserving substitution at $Q = \{ Q_x \}_{x \in \text{Dom}(\sigma)}$ and $\forall x \in X (x \overset{P}{\Rightarrow}_i x \tau)$, abbreviated as $\sigma \overset{P}{\Rightarrow}_i \tau$;

(iii) $P \otimes_o Q \overset{\scriptscriptstyle O}{\Rightarrow}_i \{ o \cdot q : x \in \text{Var}(u), u|_o = x \text{ and } q \in Q_x \}$.

Note that $P \otimes_o Q$ is defined for empty sets $P, Q$ (by $Q$ being empty, we mean that $Q$ is a set of empty sets), $P = \emptyset$ corresponding to monotonicity and $Q = \emptyset$ to stability by preserving substitutions. Orthogonal rewriting reduces to the identity if $P, Q$ are both empty, and to parallel rewriting if $P$ is a set of parallel positions and $Q$ is empty or vice versa.

Note further that $(P \otimes_o Q)_{\text{min}}$ is in general different from $P_{\text{min}}$, since there may be no position from $P$ on a path from the root of $u$ to some variable $x \in X$. This implies that our product construction is both horizontal and vertical, and that using a single rewrite step $u \overset{P}{\Rightarrow}_i v$ would define the very same binary relation between terms.

Note also that $u[\sigma]$ and $v[\tau]$ are normal below all positions in $(P \otimes_o Q)_{\text{min}}$ under assumption (ii) by Lemmas 2.14 and 2.16 — whithout that assumption, the definition of $P \otimes_o Q$ would be more complicated, since normalization steps would need to be reflected in (iii). We occasionally omit $u$ in $\otimes_o$. Using non-capturing substitutions in this definition would raise difficulties, because writing $u[\sigma]$ instead of $u[\sigma]$ implies that no variable of $\sigma x$ for $x \in \text{Var}(u)$ can be bound above in $u$, hence constraining $u$. There is a way out, though, by splitting $u[\sigma]$ into a term $t$ and a substitution $\tau$ such that $\tau = \rho[u[\sigma]]$, and replacing in the definition $u[\sigma]$ by $(t \rho)\tau_0$. For example, if $\text{Dom}(\sigma) = \{ x \}$ and $u[p] = x$, we can take $t \overset{P}{=} u[(x y)]_p$ and $\tau(x) \overset{P}{=} \lambda_{p,x}$. The obtained substitution $\tau$ is an abstraction, hence does not satisfy the assumptions of Lemmas 2.14 and 2.16. Although we believe that the entire theory can be carried out without these assumptions, this provisional choice makes the technical developments much easier.

**Example 3.1.** Consider the rewrite system $\{ f(f(x)) \rightarrow x \}$. Then $f(f(g(f(f(x))))) \overset{(A,1,1,1)}{\Rightarrow}_P f(g(x))$.

Product extends to substitutions by writing $\sigma[\gamma] \overset{P \otimes_o Q}{\Rightarrow}_i \tau[\theta]$. Then $P$ is a set of sets of positions, while $Q$ can be considered as a set of sets of positions indexed successively by the variables of $\text{Dom}(\sigma)$, then those of $\text{Dom}(\gamma)$, or, equivalently, by a set of positions indexed by their union. A routine check shows:

**Lemma 3.2.** Let $u, \sigma, O, P, Q$ st $O \subseteq P \otimes_o Q(u[\sigma])$ and $\forall x \in \text{Dom}(\sigma)(P_x \subseteq P \otimes_o Q(u[\sigma]))$. Then, $O \otimes_o (P \otimes_o Q) = (O \otimes_o P) \otimes_o Q$.

**Proof.** Follows directly from the associativity of concatenation for positions.

We now show a key property of orthogonal rewrites: the result of an orthogonal rewrite depends only on the input term $u$ and the set $P$ of orthogonal positions used, but not on their decomposition into a given product. To this end, we show that the term to be rewritten can always be decomposed into a term and a substitution whose frontier is defined by the set $(P \setminus P_{\text{min}})$.

**Lemma 3.3.** Let $P \subseteq P \otimes_o u[\gamma]$ for some linear term $u$ and $O = FP \otimes_o u[\gamma] \cap P$. Then, $P = O \otimes_o Q$ for some $Q$.

**Proof.** Take $Q \overset{\scriptscriptstyle O}{\Rightarrow}_i \{ q : o \cdot q \in P \land u|_o = x \}$.

This lemma will be mostly used as a way to define $Q$ when given $P$ and $u$, in which case we will write $P \overset{O}{\Rightarrow}_i Q$, using indeed the equation as an implicit definition of $Q$.

**Lemma 3.4.** Let $u \overset{O}{\Rightarrow}_i v$, $u'[\gamma] \overset{P \otimes_o Q}{\Rightarrow}_i v'[\theta] \overset{\scriptscriptstyle O}{\Rightarrow}_i v$. Then, $u'[\gamma] \overset{P_{\text{min}} \otimes_o Q}{\Rightarrow}_i v'[\theta] \overset{\scriptscriptstyle O}{\Rightarrow}_i v$.
Proof. By induction on the sum of length of sets in the set $P$. There are three cases according to $P$.

If $u \to^i v$, then $Q = \emptyset$, the result holds with $u' = u$ and $v' = v$.

Otherwise, $u = s[\sigma] \otimes \tau$, $t[\sigma] = v$. If $P' = \emptyset$, then $t$ is $\sigma \otimes \tau$. By induction hypothesis, $\sigma = \sigma'[\gamma] \otimes \tau[\theta] = \tau$.

By Definition 3.1, $u = s[\sigma] = s[\sigma'[\gamma]] \otimes \tau[\theta] = \tau$. Let $i \in \mathbb{N}$, $Q$ is a vector of fresh variables, hence $\sigma' \otimes \tau'$. Let now $P = P_{\text{def}} \otimes \sigma$ and $Q' = Q_1 \otimes O$. Accordingly, we split $\sigma$ as its restrictions $\sigma_1$ and $\sigma_2$, respectively the variables of $s$ which occur below $P_{\text{def}}$ and those which do not. Let $\tau_1$ and $\tau_2$ such that $\sigma_1 \otimes \tau_1$ and $\sigma_2 \otimes \tau_2$. Hence $\tau = \tau_1 \otimes \tau_2$. Let now $\nu$ such $u' = u'[\nu] = (s[\sigma])[\nu] = \nu[s[\sigma]]$,

$s' = s'[\nu] = (s'[\nu])[\nu] = \nu[s'[\nu]]$.

Let finally $\sigma_2 = \nu[\sigma]$, where $\nu$ is a (new) vector of fresh variables. Note that $u' = u'[\nu] = \nu[s[\sigma]]$.

By induction hypothesis, $s = s' \otimes \nu[\tau] = t$ and $\nu[\tau] = \nu[\tau_1] \otimes \nu[\tau_2]$. Remark that $u = s[\sigma] = s'[\nu][\nu] \otimes \nu[\tau] = u'[\nu]$, defining $\gamma \equiv \gamma_1 \otimes \gamma_2$, with $\gamma_1 \equiv \sigma_1 \otimes \nu[\tau_1]$. Likewise, $v = v[\nu] = v'[\nu][\nu] \otimes \nu[\tau_2] = v'[\nu][\nu] \otimes \nu[\tau_2]$, where $v[\nu] \equiv v'[\nu]$. Hence $\gamma \equiv \gamma_1 \otimes \nu[\tau_1] \otimes \nu[\tau_2]$, with $\gamma_1 \equiv \sigma_1 \otimes \nu[\tau_1]$. Finally $\nu[\tau_1] = \nu[\tau_1] \otimes \nu[\tau_2]$, further, $\nu[\tau_1] \equiv \nu[\tau_1]$. Since $P_{\text{def}} = P_{\text{def}} \otimes Q_2$, we get the result by choosing $\nu[Q_1] \equiv (Q_2 \otimes O')$.

Because splitting $P$ into $P_{\text{def}}$ and $Q$ depends solely on $P$ as a set, the result of an orthogonal rewrite step is entirely determined by its source and set of orthogonal positions:

Let $s[\sigma] = t[\sigma], s[\sigma] \otimes \nu[\tau] = v$ and $t[\sigma] \otimes \nu[\tau] = w$. Then $v = w$.

These results mean that permutation equivalence (we refer to [27]) for a precise definition) is built in the definition of orthogonal rewriting. Then, orthogonal rewriting inherits many properties of higher-order rewriting.

Corollary 3.6. Let $s = \emptyset \otimes \nu[\tau]$. Then $s \otimes \nu[\tau] = \emptyset \otimes \nu[\tau]$.

Corollary 3.7. Let $u$ a term and $\alpha$ a preserving substitution $s t u[\sigma]$ is normal, $u \to^i \rho \otimes \nu[\tau]$, and $s t u[\sigma]$ is normal, $u \to^i \rho \otimes \nu[\tau]$. Then, $\exists \alpha Q \sigma (\text{LAP}) v[\sigma] \otimes \nu[\tau] \to^i v[\sigma] \otimes \nu[\tau]$, and (Perm) $u[\sigma] \otimes \nu[\tau] w[\sigma] \otimes \nu[\tau]$. By Lemma 3.2, and since $\exists \alpha \otimes \nu[\tau]$,

$u[\sigma] \otimes \nu[\tau] w[\sigma] \otimes \nu[\tau]$. Finally, $u[\sigma] \otimes \nu[\tau] v[\sigma] \otimes \nu[\tau]$. We now consider a kind of converse. Given an orthogonal rewrite step $s \otimes \nu[\tau]$, we can arbitrarily split $s$ into $u$ and $v$ and $P$ into $O$ and $P'$ such that $u[\sigma] \otimes \nu[\tau] = v[\sigma] \otimes \nu[\tau]$? This is of course not possible in general, we need to take care of splitting away from all redex patterns involved in the orthogonal step (the notion of a redex pattern is not defined here, but is intuitively the part of a redex that is replaced—and not copied—when applying a rewrite rule):

Lemma 3.8. Let $u \overset{P}{\to} v$, and $Q$ a set of parallel positions $s t = u, O \otimes s \overset{P}{\to} v$. Then, $s[\sigma] \otimes \nu[\tau] v[\sigma] \otimes \nu[\tau]$.

Proof. The case $P = \emptyset$ follows from the definition. Otherwise, we proceed by induction on $|u|$. There are three cases:

1. $Q = \emptyset$. Then $s = s[\nu] = \emptyset$, and $Q = \emptyset$.

2. Let $u = \emptyset$, hence $P_{\text{def}} = \emptyset$, and wlog $\emptyset = \emptyset$.

3. By Lemma 3.3, $P = \emptyset \otimes u \overset{P_{\text{def}} \otimes u}{\to} v$, and $P = \emptyset \otimes u \overset{P_{\text{def}} \otimes u}{\to} v$. By definition of orthogonal reductions, $\emptyset \otimes u \overset{P_{\text{def}} \otimes u}{\to} v$. By easy calculations, $u = \emptyset \otimes u \overset{P_{\text{def}} \otimes u}{\to} v$. Otherwise, we recursively cut away the remaining positions of $Q$. Let $Z = u[\sigma]$, for some $o \in O$. Then $Z \overset{P_{\text{def}} \otimes u}{\to} Z[\sigma]$. Let also $Q_1 = \emptyset \otimes u \otimes (P_{\text{def}} \otimes u)$.

4. By induction hypothesis, $Z \overset{\emptyset \otimes P_{\text{def}} \otimes u}{\to} Z[\sigma]$. Let $z \overset{\emptyset \otimes u[\sigma]}{\to} z[\sigma]$, for some $\sigma \in Q_1$. Then $P_{\text{def}} \otimes u[\sigma] = P_{\text{def}} \otimes u[\sigma]$. By Lemma 3.2, closure of orthogonal rewrites under product, and Lemma 3.4 yield the result.
3. If $\Lambda \not\in P \cup Q$, then by induction hypothesis, the result holds for the immediate subterms $u_i$ of $u$ wrt to the positions $P_i$ and $Q_i$, belonging to $P_{\text{pos}}(u_i)$. Putting back the head symbol of $u$ gives then the result by monotonicity of orthogonal rewrites.

3.1 Nested higher-order critical peaks

Generating the minimal nested critical peaks that reduce the confluence of higher-order orthogonal rewriting to the existence of decreasing diagrams for these peaks requires computing the overlaps of two orthogonal rewriting steps. The lefthand sides of the rules they use may overlap horizontally, as in parallel critical pairs, but also vertically, forming vertical chains of overlaps. Unlike traditional critical pairs, vertical overlaps between two lefthand sides may take place inside the substitution associated with a partial vertical chain, while self-overlaps involving the same lefthand side must be discharged. Computing these overlaps requires then some bookkeeping, both in terms of substitutions and overlapping positions. This is the role of the notion of seed introduced now:

**Definition 3.9 (Seeds).** Given two rules $k: L \rightarrow R_1; G \rightarrow D$ from $R_w$, the set $pS_k^i$ of $(k,i)$-pre-seeds is the smallest set of tuples $(s,\sigma, P, Q)$, where $P$ and $Q$ are lists of positions in $P_{\text{pos}}(s_{\sigma} \downarrow_{SE} S_k)$ of L-redexes and G-redexes respectively, such that

(i) $pS_k^i$ contains the trivial pre-seed $\{x. x \rightarrow L ; \{\}, \{\}\}$;

(ii) $pS_k^i$ is closed by nested overlapping: given $(s,\sigma, P, Q) \in pS_k^i$, two lists of parallel positions $\{p_i \in F_{\text{pos}}(s_{\sigma} \downarrow_{SE} S_k) : p_i \gg \sigma \cdot F_x \}_{i \in I}$ and $\{q_j \in F_{\text{pos}}(s_{\sigma} \downarrow_{SE} S_k) : q_j \gg \sigma \cdot F_G \}_{j \in J}$ which are not both empty and whose elements are pairwise incomparable, renamings $\{L_i\}_{i \in I}$ of $L$ and $\{G_j\}_{j \in J}$ of $G$ such that $\forall i,j, \sigma \cdot V_a(L_i), \sigma \cdot V_a(G_j)$ and $\sigma \cdot V_a$ are pairwise disjoint sets, and $\tau$ a most general SE-unifier of the equation $\wedge_{i \in I}(\sigma \downarrow_{SE} S_k) p_i = L_i \wedge \wedge_{j \in J}(\sigma \downarrow_{SE} S_k) q_j = G_j$, the non-trivial pre-seed $(\sigma \downarrow_{SE} S_k, \tau, p \cup \{p_i \}_{i \in I}, Q \cup \{q_j \}_{j \in J})$ belongs to $pS_k^i$.

The set of seeds is defined as $S_k^i = \{ (u \downarrow_{SE} P, Q) : (u, \tau, P, Q)$ is a non-trivial pre-seed $\}$.

In this recursive definition of pre-seeds, the last overlapping substitution $\tau$ tells us where to overlap next, while the maximal positions in $P$ and $Q$ of these overlaps for both rules tells us where to not overlap. In particular, the very first overlap is impossible with $L$, unless $k = 1$: our notation $S^i_k$ is meant to suggest that there is a $k$-redex above the $i$-redexes. The fact that overlapping at some recursive call may take place inside the substitution obtained at the previous recursive call explains the need of a recursive computation of the pre-seeds.

Note that, for the first nested overlapping computation, the set $\{p_i \}_{i \in I}$ is empty, so that the first (parallel) overlap is always with the sole rule $G \rightarrow D$ overlapping $L$ at the set of parallel positions $\{q_j \}_{j \in J}$.

Since the sets $P, Q$ are monotonically increasing (wrt to the order on positions), our computation of pre-seeds avoids too many redundancies. Filtering out the remaining redundancies would require some more bookkeeping.

One may wonder why we call these critical peaks nested rather than orthogonal. First, orthogonality refers explicitly to the absence of critical pairs, so an orthogonal critical pair would be kind of self-contradicting. Another reason is that there is a single rule lefthand side sitting at the top of a seed. Therefore, all redexes occurring in a seed are nested inside that lefthand side, whether they extend the seed construction vertically or horizontally.

There is no reason why the set of pre-seeds, hence the set of seeds, should be finite. In case it is finite, then confluence is checkable. Otherwise, confluence needs to be proved. Here is such an example:

**Example 3.2.** Consider the rules $1: f(f(x)) \rightarrow z$ and $2: f(y) \rightarrow y$, and let us compute first the pre-seeds between rule 1 and 2.

Init: $(\{z \rightarrow f(f(z))\}, \{\}, \{\})$ is a seed.

Step 1: the set $\{p \in \{\Lambda\} : p \gg \sigma \cdot \{\} \cdot \{\Lambda\} = \{\Lambda\}\}$ is empty. The set $\{q \in \{\Lambda\} : q \gg \sigma \cdot \{\} \cdot \{\Lambda\} = \{\Lambda\}\}$ and we choose the list made of the single position $\Lambda$. The unifier of the equation $f(f(z)) = f(y)$ yields $(y \rightarrow z)$. Therefore $(f(f(z)), \{y \rightarrow f(z)\}, \{\}, \{\})$ is a seed.

Step 2: the first set remains the same while the second becomes $\{1\}$. We can then choose the list made of that position. The unifier of the equation $f(z) = f(y)$ yields $(y \rightarrow z)$. Therefore $(f(f(z)), \{y \rightarrow z\}, \{\}, \{\})$ is a seed.

Step 4: the computation stops with empty sets of $p$'s and $q$'s.

Since this computation is non-deterministic in the choice of positions, we could actually have done step 3 directly, instead of step 2. This would have given the very same seed as in step 3 (because the substitution obtained at step 2 had no effect on the overlap term).

Let us now compute the pre-seeds between rule 2 and rule 1.

Init: $(z, z \rightarrow f(z))$, $\{\}, \{\}$ is a pre-seed.

Step 1: The sets of $p$'s and $q$'s are respectively $\{\}$ and $\{\}$.

Choosing $q = \Lambda$, we get $\tau = (z \rightarrow f(z))$, hence, renaming $x$ in $z$, we get $(f(z), \{z \rightarrow f(f(z))\}, \{\}, \{\})$ is a pre-seed.

Step 2 yields the pre-seed $(f(f(z)), \{\}, \{\})$, and the computation stops at step 3 with no further seed.

Computing now the pre-seeds between rule 1 and itself, we get the single pre-seed $(f(f(z)), \{\}, \{\})$.

We can now compute the pre-seeds between rule 2 and itself and get infinitely many elements of one of the following two sets: $\{(f^{n+1}(z), \{z \rightarrow f(f(z))\}, \{1\}^{|n+2|} \times \{1\}^{|n+1|} \times \{1\}^{|n|})\}_{n \geq 0}$ and $\{(f^n(z), \{z \rightarrow f(f(z))\}, \{1\}^{|n+1|} \times \{1\}^{|n|} \times \{1\}^{|n-1|})\}_{n \geq 0}$.

A straightforward induction on the definition of pre-seeds shows:

**Lemma 3.10.** Let $(u, P, Q) \in S_k^i$ be a seed. Then, $u = L_k \varphi_{SE}^i$ for some $\varphi$ and $v \varphi_{SE}^i u \varphi_{SE}^i w$.

**Definition 3.11.** Let $(u, P, Q) \in S_k^i$ and $\theta$ an arbitrary substitution. Then, $u \varphi_{SE}^i u \varphi_{SE}^i w$, the tuple $(u \varphi_{SE}^i, k, P, l, Q)$, and $(v \varphi_{SE}^i, k, P, l, Q)$ are respectively a nested critical peak a nested overlap, and a nested peak.

Note that Lemma 2.12 is implicitly used in these definitions. Note also that the quadruple $(k, P, l, Q)$ defines already a family of nested overlaps, since the terms $u = L_k \varphi_{SE}^i$ can be recomputed from it, $\theta$ being a most general $SE$-unifier among possibly many that satisfy the corresponding set of equations given in the definition of the set of pre-seeds. In case $\theta$ is the identity, we then get a nested critical overlap.

We now give a useful characterization of nested overlaps.

**Lemma 3.12.** Assume $(u, k, P = \{p_i\}_{i \in I}, I, Q = \{q_j\}_{j \in J})$ is a nested overlap. Then, $\wedge_{i \in I, j \in P} u_{|SE} I, \sigma_l$ and $\wedge_{i \in I, j \in Q} u_{|SE} G_l, \tau_j$, where $\{L_i\}_{i \in I}$ and $\{G_j\}_{j \in J}$ are variable renamings of $L, G$ sharing no variables among themselves nor with $u$.

**Proof.** We assume implicitly here that the positions in $P, Q$ are listed in non-decreasing order wrt $\gg$. The proof is by induction on $P \cup Q$. Let $O = \max(P)$, $O' = \max(Q)$, $P' = P \setminus O$ and $Q' = Q \setminus O'$. It follows from the definition of nested overlaps that $(u, k, P' = \{p_i\}_{i \in I}, Q' = \{q_j\}_{j \in J})$ is a nested overlap. By induction hypothesis, it satisfies the property $\wedge_{i \in I, j \in Q'} u_{|SE} L_i, \sigma_l$ and $\wedge_{j \in J, q \in Q} u_{|SE} G_j, \tau_j$, where $\{L_i\}_{i \in I}$ and $\{G_j\}_{j \in J}$ are variable renamings of $L, G$ sharing no variables among them-
Theorem 3.1

Consider the plain rewrite system not preserving in general, but does not rewrite. On the other hand, is important to keep them separate. In particular, the substitution is then obtained by expressing the property that is a preserving substitution such that implies that is a nested critical peak. Since functional rewrites operate modulo variable sharing no variables among themselves nor with . Hence the conditions on variables of , the corresponding pre-seeds unification problem is therefore unifiable modulo . By property of most general SE-unifiers, and for some substitutions of domain of and of the Church-Rosser property of (, ) holds. Thanks to the fact that the substitutions from the set and have pairwise disjoint domains. Let . By Lemma 3.10, unless . We use the fact that the substitutions from the set and have pairwise disjoint domains. This ends the proof.

4. Decreasing diagrams for labelled rewriting

Our goal is to prove confluence on untyped terms. Since beta-reductions do not terminate on their set, we shall use van Oostrom’s technique relying on the existence of a decreasing diagram for each local peak [28]. Since functional rewrites operate modulo variable renaming, we shall also use Jouannaud and Liu’s extension of van Oostrom’s decreasing diagram completeness property, for which commutation plays a central role [18].

Labelled rewriting. Labelled rewriting is a basic notion underlying decreasing diagrams. A labelled binary relation on an abstract set is denoted by , where is an element of a set of labels equipped with a partial quasi-order which strict part is well-founded. We write for (resp., ) if and only if for all in the multiset (or sequence) of labels.

Decreasing diagrams [30].

Definition 4.1 (Local diagram). Given a labelled relation on an abstract set, a local diagram is a pair made of a local peak and a conversion .

Definition 4.2 (Decreasing local diagram). A local diagram with peak is decreasing if its joining conversion has the form with labels in (resp., ) strictly smaller than in (resp., ) and .
Decreasing diagrams are abbreviated as DDs. A facing step of a decreasing diagram may be missing, its side steps are then absorbed by the middle ones. This is the case if \( m > n \) or \( n > m \).

**Theorem 4.1** ([30]). A labelled, abstract rewriting relation is Church-Rosser if all its local peaks have a decreasing diagram. Conversely, any confluent relation on a countable set can be labelled so that all its local peaks have decreasing diagrams.

**The modulo case.** van Oostrom’s theorem generalizes to rewrite relations modulo an equational theory. Its converse requires two further properties: that rewriting commutes over the equational theory and that the set of labels is increased by a new minimum label reserved for the equational steps [18]. This generalization of van Oostrom’s theorem will be applied later to \( \Longrightarrow_{\beta_n} \).

## 5. Confluence in \( \Lambda \Pi \text{Mod} \)

We now consider the Church-Rosser property of the rewrite relation used in \( \Lambda \Pi \text{Mod} \) on untyped terms, generated by the rewriting system whose pieces have been described in Section 2. As usual, it is essential to choose carefully the relation to work with. For the method to be sound, it must contain rewriting and define the same convertibility relation as the one generated by the set of rules/equations. In case of non-terminating rewrites, non-linearities make it difficult to get decreasing diagrams for ancestor peaks since there can only be one facing step on each side of the joining conversion. The usual way out is to impose left-linearity (for non-confined variables is actually enough) and use some form of parallel rewriting to handle non-right-linearities. First-order rewrites having a small label will not need parallel rewriting, but \( \beta \)-rewrites will in case they take place below a first-order rewrite. Higher-order rewrite steps will require a label bigger than the other steps, so that the latters can be neglected when needed. Parallel higher-order rewriting will be needed again, but is no more sufficient: parallel higher-order rewrites taking place below a beta-step may now become both duplicated and nested, making orthogonal rewriting necessary to get decreasing diagrams. The various rewrite steps to be considered in an arbitrary conversion, from which all local peaks and cliffs must be replaced by decreasing diagrams, are therefore:

\[
(\mathcal{R}_o, S_{cd}, E_{cd}) \cup \xleftarrow{} \cup \xrightarrow{} \cup \xrightarrow{} \cup \xrightarrow{} \cup \xrightarrow{\beta} (\mathcal{R}_o, S, E)
\]

We shall also use when needed the notation \( \xrightarrow{\beta} \) as an abbreviation for

\[
(\mathcal{R}_o, S_{cd}, E_{cd}) \cup \xleftarrow{} \cup \xrightarrow{} \cup \xrightarrow{} \cup \xrightarrow{} \cup \xrightarrow{\beta} (\mathcal{R}_o, S, E).
\]

### 5.1 Labelling rewrites

We label each rewrite or equational step by a pair \( \langle m, n, p \rangle \), where \( m \) is a natural number indicating the rewrite category, \( n \) is a natural number or a term, and \( p \) is the rewrite (non)position or set of positions, which is used here for convenience, not for comparing labels. Labels do not depend upon the direction of a rewrite step. Labels are compared lexicographically, using the order on natural numbers for all their components but terms, which are compared in the order defined by the termination property of \( (\mathcal{R}_o, S_{cd}, E_{cd}) \).

<table>
<thead>
<tr>
<th>Rewrite</th>
<th>( m )</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u =_n v )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( u \xleftarrow{p} E_{cd} v )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( u \xrightarrow{\beta} (\mathcal{R}<em>o, S</em>{cd}, E_{cd}) v )</td>
<td>1</td>
<td>( u )</td>
</tr>
<tr>
<td>( u \xrightarrow{\beta} )</td>
<td>2</td>
<td>// canonical labelling</td>
</tr>
<tr>
<td>( u \xrightarrow{\beta} )</td>
<td>3</td>
<td>( i )</td>
</tr>
</tbody>
</table>

There are three different categories of rewrite steps: first-order, functional and higher-order, and two kinds of equational steps: first-order and functional. The label’s structure obeys two important principles. First, only the second label matters for homogeneous local peaks, which use a rewrite of the same category on both sides of the peak. On the other hand, finding decreasing diagrams for heterogeneous local peaks relies on the first label’s component.

Equalities in \( E_{cd} \) and \( \alpha \)-conversion are treated very differently. While the later is considered as an equational theory for which all rewrites commute over, \( \xrightarrow{} \) is considered instead as a symmetric rewrite relation whose local peaks with the other three rewrite relations, called cliffs, must have decreasing diagrams like the other local peaks. This subtlety ease the use of the completeness theorem for decreasing diagrams modulo, for which \( =_\alpha \)-steps must have the same label, minimum among all labels [18].

### 5.2 Decreasing reductions

So far, we have used essentially preserving substitutions replacing subterms at variable positions, since this is the essential notion needed in the definition of orthogonal rewriting. However, Theorem 3.1 introduces substitutions that are not preserving. Their use requires a specific treatment carried out now.

**Definition 5.1.** Given a normal term \( u \), a rewrite step \( u \xrightarrow{A} v \) is collapsing if \( v \) is a non-confined variable or an abstraction.

The following property follows directly from the definition:

**Lemma 5.2.** Non-collapsing rewrites preserve \( \beta^0 \)-normal forms.

As a preparation for the study of critical higher-order peaks, we define below in two steps, reductions which are intended to be decreasing for all instantiations.

**Definition 5.3.** A derivation is called eager-normal if of the form \( u \xrightarrow{\sigma} s \) or \( u \xrightarrow{\sigma} s \xrightarrow{t} v \) where \( s \xrightarrow{t} v \) is an eager-normal derivation, and \( s \) is a non-confined variable or an abstraction.

Note that the essential step cannot be a simplification step, since \( (S, E) \) is Church-Rosser and terminating, and \( u \xrightarrow{\sigma} s \) is normal.

**Lemma 5.4.** Let \( u \) be a normal term \( s \) \( u \xrightarrow{\sigma} s \xrightarrow{t} v \) is an eager-normal derivation whose essential steps are non-collapsing. Then, given \( \sigma \), there is an eager-normal derivation from \( u \) to \( v \) whose essential steps use the same rules and are non-collapsing.

Note that there is no need to normalize \( u \sigma \) and \( v \sigma \), since this is done anyway in the definition of an eager-normal derivation.

Note also that the assumption that essential steps are non-collapsing is crucial: otherwise, instantiation of variables by abstractions could create new \( \beta^0 \)-redexes.

**Proof.** By induction on the sequence, assuming wlog that \( \sigma \) is normal. Let \( u \xrightarrow{id} s \xrightarrow{\sigma} t \xrightarrow{\sigma} v \) be the \( S_\sigma \)-normalization prefix of the reduction. Since \( (S, E) \) is Church-Rosser, then \( u \sigma \xrightarrow{S_\sigma} v \sigma \xrightarrow{S_\sigma} \). If \( s \xrightarrow{\sigma} v \), we are done. Otherwise, \( s \xrightarrow{\sigma} t \) with \( i \in \{ \beta \} \cup \mathcal{R}_o \cup \mathcal{R}_{\delta_o} \), and \( t \xrightarrow{\sigma} v \) is eager-normal. First, \( \sigma \xrightarrow{S_\sigma} = (\sigma[s|\sigma[p]]p \xrightarrow{S_\sigma} (\sigma[s|\sigma[p]]p \xrightarrow{S_\sigma} (\sigma[s|\sigma[p]]p \xrightarrow{S_\sigma} (\sigma[s|\sigma[p]]p \xrightarrow{S_\sigma} \). There are three cases:

- \( i = \beta \). Then \( |p| = (\lambda w.w') \) and \( s \xrightarrow{\sigma} w \sigma \xrightarrow{\sigma} w' \sigma \xrightarrow{\sigma} v \sigma \xrightarrow{\sigma} \). Hence \( s \sigma \xrightarrow{\sigma} v \sigma \xrightarrow{\sigma} \). There are three cases:

  - \( i = \beta \). Then \( |p| = (\lambda w.w') \) and \( s \xrightarrow{\sigma} w \sigma \xrightarrow{\sigma} w' \sigma \xrightarrow{\sigma} v \sigma \xrightarrow{\sigma} \). Hence \( s \sigma \xrightarrow{\sigma} v \sigma \xrightarrow{\sigma} \).
Given a local peak \( f \) folding them again. in this case. This is because normalization as well by our example.

\[ \begin{align*}
\text{Definition 5.5. } & \text{Given a peak } (v, u, w) \text{ labelled } \langle m \in \{1, 3\}, i, P \rangle \text{ and } (3, j, Q) \text{ from } u \text{ to } v \text{ and } u \text{ to } w \text{ respectively, } \\
& \text{a decreasing reduction is a derivation from some term } s \text{ st } \\
&(i) s \in \{v, w\} \text{ and all steps have a label strictly smaller than either } (m, i) \text{ or } (3, j); \text{ or} \\
&(ii) m = 3 \text{ and } s = v \cdots \overset{(3, i)}{j} \cdots \overset{(3, i)}{k} \cdots \overset{(3, i)}{l} \cdots \text{ with } \ell < (3, i) \text{ and } l' < (3, \max(i, j)) \text{ (the case } s = w \text{ is symmetrical).} \\
\end{align*} \]

A non-collapsing eager-normal decreasing reduction is called strongly decreasing (SDR).

The main restriction of an SDR is that the first form applies when \( m = 1 \) and \( s = v \), and therefore no facing step is allowed in this case. This is because normalization \( \beta \)-steps would otherwise be necessary before the facing step, which would contradict van Oostrom’s condition for decreasing diagrams. Although this restriction is satisfied by our example, it is not very satisfactory, and believe it is non-necessary. The reason is that, given a higher-order step at position \( p \), it is possible to hide the \( \beta \)-steps taking place before at positions below \( p \), by including them to the definition of the higher-order rewrite at \( p \).

We have not tried this idea yet because our example satisfies the restriction.

Being an eager-normal decreasing reduction is of course no real restriction, but the absence of collapsing essential steps is. However, since higher-order rewrites are non-collapsing by Assumption 6 (i.ii), the only further restriction is that \( \beta \)-steps and first-order steps are non-collapsing. And since typed first-order rewrites cannot pop-up abstractions, only collapsing \( \beta \)-steps are indeed forbidden, a quite weak restriction for practice. This restriction is satisfied as well by our example.

\[ \begin{align*}
\text{Example 5.1. } & \text{An example of SDR which is not a development.} \\
\text{Given the previous local peak } (v, u, w), \text{ we use the notation } \\
\end{align*} \]

As a consequence of Lemma 5.4, we get:

\[ \text{Corollary 5.6 (Stability of SDRs). Let } u \triangleleft \llcorner m, i; n, j \triangleright \text{ for SDRs originating from } u, \text{ and } \llcorner m, i; n, j \triangleright \text{ for those originating from } w. \text{ We sometimes omit unnecessary components of these 4-tuples.} \]

\[ \text{By the Church-Rosser assumption, } s \triangleright \llcorner m, i; n, j \triangleright \text{ for both } s \text{ and } w. \text{ Hence, } s \triangleright \llcorner m, i; n, j \triangleright \text{ for both } s \text{ and } w. \text{ Hence, } \]

An important property for constructing decreasing diagrams for higher-order local peaks is the ability to permute steps taking place in the term \( u \) and the capturing substitution \( \sigma \) of a term \( u[\sigma] \) so as to bring together distant orthogonal \( i \)-steps taking place respectively in \( u \) and \( \sigma \) and merge them in a single orthogonal \( i \)-step taking place in \( u[\sigma] \). This is possible for strongly decreasing reductions, thanks again to the non-collapsing assumption (for the essential steps behind the facing higher-order step would suffice here).

\[ \text{Lemma 5.7. Let } \sigma \text{ preserving such that } s[\sigma] \overset{p}{\rightarrow} t[\sigma] \overset{i}{\rightarrow} t[r]. \text{ Then, } \]

\[ \text{Proof. The case where } j \text{ a higher-order rewrite follows from Corollary 3.7. If } j \text{ is a } \beta \text{-rewrite or a first-order rewrite and } i \text{ is a higher-order rewrite, the use of Lemma 2.12 introduces extra rewrites steps with } \sigma E. \text{ Other cases are straightforward.} \]

\[ \text{Lemma 5.8. Let } s \text{ be a normal term and } \sigma \text{ is a preserving substitution such that } s \triangleleft \llcorner k; j; 3; i \triangleright \llcorner \sigma \text{ and } \llcorner k; j; 3; i \triangleright \theta \text{ are two SDRs. Then, there is an SDR } s[\sigma] \triangleleft \llcorner k; j; 3; i \triangleright \llcorner \theta \text{.} \]

\[ \text{Proof. By definition of an SDR, } \]

\[ \text{In all cases but the first, we conclude by induction hypothesis.} \]

\[ \text{In the sequel, we call strongly decreasing diagrams (SDDs) those that use strongly decreasing reductions on both sides.} \]

\[ \text{5.3 DDs for first-order local peaks and cliffs} \]

\[ \text{Decreasing diagrams result from our assumption that the equational rewrite system } (R_{\sigma}, S_{\sigma}, E_{\sigma}) \text{ is Church-Rosser and terminating:} \]

\[ \text{Corollary 5.9. Homogeneous first-order local peaks and cliffs have decreasing diagrams.} \]

\[ \text{There is a subtlety in the proof. Consider a local peak } s[\sigma] \overset{p}{\rightarrow} u \overset{j}{\rightarrow} t. \]

\[ \text{By the Church-Rosser assumption, } s \rightarrow v \triangleleft E_a \text{ for } \sigma \text{ strictly smaller than } u. \text{ Since the labels of the peak are both equal to } \{1, u\}, \text{ it does not matter whether the first component of the label of the step originating from } r \text{ is } 0 \text{ or } 1. \text{ And since one of these two labels suffices to make the diagram decreasing, the argument applies as well if } i \in R_{\sigma}, j \in S_{\sigma} \text{ or } i \in R_{\sigma}, j \in E_{\sigma}. \]

\[ \text{Then, all rewrites must be confined, and hence have } 0 \text{ as their label’s first component.} \]

\[ \text{5.4 DDs for functional local peaks} \]

\[ \text{By parallel canonical labelling, we mean the canonical labelling of the relation } \Rightarrow_{\beta \alpha} \text{, which is Church-Rosser modulo } \alpha \text{ on untyped terms since this is already true of } \Rightarrow_{\beta \alpha}. \text{ We therefore apply the generalisation of van Oostrom’s theorem rather than the original version, using the property that } \Rightarrow_{\beta} \text{ commutes over } \Rightarrow_{\alpha}. \]

\[ \text{Note that we need to repeat the same reasoning done previously to account for the fact that the labels of } \beta \text{-steps and the other } \beta \text{-steps are different. Again, the key is that the former are smaller.} \]
The fact that they are plain steps rather than parallel steps does not harm because plain steps are particular parallel steps.

Labelling the functional rewrite steps requires further use of an erasing interpretation before to take the parallel canonical labelling, to ensure its invariance by a first-order rewrite taking place below, which will be needed for the analysis of functional/first-order local peaks. Our interpretation is a homomorphism from untyped terms of \(\lambda\text{I}Mod\) to terms of a lambda calculus enriched with a list construction of variable arity denoted by \(\{\ldots\}\). The vector operation over lists is interpreted as concatenation. Lists of lists are flattened. Given a term in which no variable is bound twice,
\[
\| (s \triangleright t) \| \overset{def}{=} \{ [\{s\} [t]] \} \quad \| f(s) \| \overset{def}{=} \{ f[s] \} \quad \| x \in X \| \overset{def}{=} \{ x \} \\
\| \lambda x : t. s \| \overset{def}{=} \{ \lambda x : [t]. [s] \} \quad \| [\Pi x : t. s] \| \overset{def}{=} \{ [\Pi ([s], [t])] \}
\]

Positions in a list \(\{\pi\}\) are as expected, as are the extension of \(\lambda\text{I}\)-constructors from terms to lists of terms.

Being non-erasing homomorphisms by Assumption 4(ii), interpretations are compatible with labelling local functional peaks:

**Lemma 5.10.** Assume \(s | p = (x_\vdash v \vdash w)\) for some \(p \in \mathcal{P}(q,s)\). Then, \(\[s\|/\beta \equiv [p]\|/\beta\] \equiv \{\{x \mapsto [w]\}\}_{\beta}/\beta\) for some \(q \in \mathcal{P}(\{s\|)\).

This justifies the use of the completeness theorem for the lambda-calculus of interpreted terms.

**Corollary 5.11.** Homogeneous functional local peaks and cliffs have decreasing diagrams.

**5.5 DDs for functional/first-order peaks and cliffs**

Interpretations are compatible with \(R_{\beta}\)-rewriting as well:

**Lemma 5.12.** Let \(s \xrightarrow{R_{\beta}} t\). Then, \(\|s\| = \|t\|\).

**Proof.** Follows from the assumption that non-confined variables in first-order rules appear linearly. Note that algebraic terms, in particular confined ones, are interpreted by the empty list. \(\Box\)

**Corollary 5.13.** Heterogeneous functional/first-order local peaks and cliffs have decreasing diagrams.

**Proof.** Consider the local peak \(v \xrightarrow{(2,m,p)} u \xrightarrow{(1,n,p)} w\) with \(i \vdash l \rightarrow r \in R_{\beta}\). Since no critical peak is possible here, \(P = P_{\#} \uplus Q\) where \(P_{\#} \# p\) is maximal. By (DP) \(\|w\|/\beta \equiv s \equiv v\|/\beta\) with \(v \equiv u \equiv s \equiv w\) if \(P_{\#} \# 0\). If \(Q \equiv 0\), we are done. Otherwise, there are two cases:
- \(P = P_{\#} \uplus \{q\}\) with \(p \# p\). Since \(\beta\) is left-linear, \(\rightarrow t \xrightarrow{\beta} s\) for some \(t\) by (LAP). Hence \(t \xrightarrow{\beta} w\).
- \(P = P_{\#} \uplus Q\), with \(Q \equiv q\). Since \(Q\) is not empty, the \(\beta\) redexes at \(Q\) are below non-confined variables of \(l\), hence linear by Assumption 4(ii). By (LAP), \(\rightarrow t \xrightarrow{Q_{\#}} s\) for some \(t\) where \(Q' \# p\), hence \(Q' \# P_{\#}\). Therefore, we obtain \(t \xrightarrow{\beta} Q_{\#} w\).

In both cases, the label of the parallel \(\beta\)-step from \(w\) is \((2,m,Q')\) by Lemma 5.12, resulting in a decreasing diagram from \(v\) to \(w\).

The case of cliffs is similar, even simpler: a \(\beta\)-step cannot occur below an equational step since equations in \(E_{\text{cd}}\) are confined. \(\Box\)

**5.6 DDs for higher-order local peaks**

Since typed first-order rewrites cannot pop-up abstractions, the following assumption is not very different from expected:

**Critical peak assumption 1:** Higher-order nested critical peaks have strongly decreasing diagrams.
Since \( u \) is normal by assumption that \( \Lambda \in P \), then \( u' \) and \( \theta \) are normal, hence \( r \gamma \) is normal by Lemma 2.9. It follows that \( r \) and \( \gamma \) are normal. By Lemma 2.16, \( s, t \) are normal. However, \( s \gamma, t \gamma \) need not be normal since there is no reason why \( \gamma \) should be preserving. Hence \( r \gamma \stackrel{O}{\Rightarrow} s \gamma \downarrow s \mathcal{E}_{E} \) by Lemma 2.12.

Likewise, \( r \gamma \stackrel{O}{\Rightarrow} t \gamma \downarrow s \mathcal{E}_{E} \). Further, by Definition 3.1, \( s \gamma \downarrow s \mathcal{E}_{E} [\sigma] \) and \( t \gamma \downarrow s \mathcal{E}_{E} [\sigma] \) by \( r \gamma \stackrel{O}{\Rightarrow} t \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By Critical pair assumption 1, there is a strongly decreasing diagram for the nested critical peak \( s \gamma \downarrow s \mathcal{E}_{E} [\sigma] \) and \( t \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

Because \( \theta \) is preserving, all our rewrite rules are stable, hence \( s \gamma \downarrow s \mathcal{E}_{E} [\sigma] \), \( t \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By Corollary 3.7(LAP), \( s \gamma \downarrow s \mathcal{E}_{E} [\sigma] \), \( t \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This yields a new local peak \( m \gamma \downarrow s \mathcal{E}_{E} [\sigma] \), which is strictly smaller than the one we started from.

By induction hypothesis, the local peaks \( \sigma \gamma \downarrow s \mathcal{E}_{E} [\sigma] \), \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \) have SDDs \( \sigma \gamma \downarrow s \mathcal{E}_{E} [\sigma] \), \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \) for some substitution \( \phi \). We have now got two successive of derivations from \( v \) : \( v \downarrow s \mathcal{E}_{E} [\sigma] \), \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By Lemma 2.9, the initial \( v \gamma \)-step can be eliminated to give an SDR \( v \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

Lemma 5.8 then provides us with an SDR \( v \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

Likewise, there is an SDR \( w \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

Together, these SDRs constitute a strongly decreasing diagram for \( v \gamma \)

We now consider the general case where \( O = (P \cup Q)_{\min} = \{a_{k}\} \), \( P_{k} = \{p \in P : p \geq a_{k}\} \), \( Q_{k} = \{q \in Q : q \geq a_{k}\} \), and \( O_{k} = P_{k} \cup Q_{k} \).

We can then apply the previous result to each \( O_{k} \) and then combine the resulting strongly decreasing diagrams for peaks \( v \) \( (3, j, p_{k}) \) \( (3, j, q_{k}) \) \( R_{w} \) \( R_{w} \) by using (DP) and Lemma 5.8 to get a strongly decreasing diagram. Here, the facing \( i \)-steps (resp. \( j \)-steps) will be combined into a single \( i \)-step (resp. \( j \)-step) since they occur at parallel positions.

**Corollary 5.14.** Homogeneous higher-order peaks have decreasing diagrams under Critical peak assumption 1.

### 5.7 DDs for first-order/higher-order local peaks and cliffs

Because pattern matching the lefthand side of a higher-order rule preserves its top function symbol—which belongs to \( \Sigma_{Q} \) by assumption 5—, it cannot overlap the lefthand side of a first-order rule which is entirely built from the first-order signature.

Simplification rules can of course overlap higher-order rules at strict subterms, but they need not be taken into account since higher-order rewriting uses pattern matching modulo the confined theory. Critical pairs are therefore obtained, as for rewriting modulo, by unifying modulo \( SE \) the left-hand side \( l \) of a rule \( l \mapsto r \in R_{w} \) with a strict subterm \( L_{l} \) of a rule \( L \mapsto R \in R_{w} \). A key property originating from the definition of higher-order rewriting is that no equation from \( E \) can take place on the path from its root to the position \( p \) when firing the rule \( L \mapsto R \), a crucial condition for these critical pairs modulo to make sense.

**Critical peak assumption 2:** Critical peaks of first-order rules onto higher-order rules have strongly decreasing diagrams.

Let \( v \) \( (1, m, p) \) \( (3, j, Q) \) \( R_{w} \) \( R_{w} \) \( w \). The construction of a decreasing diagram is by induction on the size of \( u \) modified as before. Let \( P = (Q \cup \{p\})_{\min} \). We consider first the case where \( P \) is a singleton. There are three cases:

- \( P = \{p\} \) No overlap of \( L \) onto \( l \) is possible, as already noticed.
- \( s = u(\exists) \) for fresh \( s \) and \( s(\sigma) = u \). Since non-confined variables in \( l \) are linear, \( s \) rewrites at \( p \) with rule \( j \).
- \( P = (Q \cup \{p\}) \) \( Q \subseteq \sigma \) \( Q' \). Therefore, by Lemma 3.8, \( v \downarrow e[\sigma]^j_s \) \( s \gamma \downarrow s \mathcal{E}_{E} [\sigma] \) \( \sigma \) \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By Lemma 3.3, \( \sigma = \emptyset \) \( Q \subseteq \sigma \) \( Q' \).

By Lemma 3.8, \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This case is shown at Figure 5.

Let \( r \) \( i \) \( r \mapsto r \) \( i \) \( (Q \cup \{p\})_{\min} \).

By Lemma 3.7, \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This case is shown at Figure 7.

Let \( r \) \( u \) \( (Q \cup \{p\})_{\min} \).

By Lemma 3.7, \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This case is shown at Figure 7.

Let \( r \gamma \) \( j \) \( r \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By induction hypothesis, we have \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This case is shown at Figure 7.

Let \( r \gamma \) \( j \) \( r \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By induction hypothesis, we have \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This case is shown at Figure 7.

Let \( r \gamma \) \( j \) \( r \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

By induction hypothesis, we have \( \gamma \downarrow s \mathcal{E}_{E} [\sigma] \).

This case is shown at Figure 7.
Let \( r \overset{\text{def}}{=} u[\gamma(Qv_{s(\gamma)})_{\alpha}] \). By the same token again, \( r \overset{p \beta}{\rightarrow} s \), hence \( r[\gamma] \overset{p \beta}{\rightarrow} s[\gamma] \) by stability under substitution of non-confined variables. By Lemma 3.3, \( Q = \{ q \} \otimes_r Q' \). By Lemma 3.8, \( u \overset{r[\gamma] \overset{p \beta}{\rightarrow} s[\gamma]}{\rightarrow} t[\gamma] \overset{\beta}{\rightarrow} t = w \).

Since symbols at \( q, p \) are different by signature assumption, then \( p \neq q \). Then, \( p \in q(\mathcal{F} \mathcal{P} \mathcal{O} \mathcal{S}(L)/(\{A_k (\geq p) (P_i^n) \})) \). Therefore, by definition of higher-order rewriting, \( L[\gamma] \overset{\beta}{\rightarrow} L[\gamma] \) for some \( \theta, p' \) with \( p = q' \), assuming \( L \) and \( t \) share no variable, and therefore \( \theta = E \sigma \), where \( \sigma \) is a most general \( E \)-unifier of the equation \( L[\gamma] = l \), and \( R \sigma \overset{t}{\rightarrow} L[\sigma] \) is a critical peak of \( j \) onto \( i \). By assumption, this peak has a critical diagram. The proof then follows the case of nested critical peak.

The case where \((P \cup \{ p \})_{\alpha} \) is not a singleton follows the same scheme as for HO/HO local peaks, using now (DP).

**Corollary 5.15.** Heterogeneous first-order/higher-order local peaks have decreasing diagrams under Critical peak assumption 2.

### 5.8 DDs for functional/higher-order local peaks

We omit here the treatment of plain \( \beta^0 \)-steps which are a particular case of that of parallel \( \beta \)-step.

Let \( v \overset{\beta^0}{\rightarrow} \overset{\beta}{\rightarrow} w \). By induction on \( |P| \), we prove the existence of a decreasing form of the diagram \( v \overset{Q'}{\rightarrow} \overset{\beta}{\rightarrow} w \). Since \( \beta \)-rewrites have a smaller category, this diagram is decreasing. By definition of parallel rewriting, \( u \overset{p \beta}{\rightarrow} v \overset{P \beta}{\rightarrow} w \). Assume the result is true for a single beta-reduction considered as a parallel one. Then \( v' \overset{\beta}{\rightarrow} \overset{\beta}{\rightarrow} s \overset{\beta}{\rightarrow} w \). We then conclude by induction hypothesis.

We now sketch the case of a single beta-rewrite from \( u \) at some position \( p \), and assume that \( p \geq P q \), where \( q \) is the position of a higher-order rewrite \( L \rightarrow R \). Since left-hand sides of higher-order rules are beta-normal, \( p \) must be below the fringe of \( L \). Then, by Lemma 2.17, the position is preserved by the higher-order rewrite at \( q \). If instead \( q \geq P p \), we know that \( q \) is preserved by the rewrite at \( p \). It follows that the situation is very much like the first-order/higher-order heterogeneous peaks, with the exception that a beta-rewrite at \( p \) may stack redexes occurring below \( p \). This is the reason for introducing orthogonal higher-order redexes.

**Corollary 5.16.** Heterogeneous functional/higher-order local peaks have decreasing diagrams.

### 5.9 The confluence theorem

**Theorem 5.1.** \( \lambda\Pi Mod \) equipped with a set of rules and equations \((R_{\alpha}, S_{\alpha}, E_{\alpha}, R_{\beta})\) satisfying all our assumptions is Church-Rosser modulo \( \alpha \cup E_{\alpha} \) on untyped terms.

### 6. Encoding Coq’s universes in \( \lambda\Pi Mod \)

Encodings follow a well-understood schema [12]: statements and proofs in a given system are encoded in Dedukti so as to preserve the provability relationship. Further, encodings must be shallow so as to get proof terms. This is especially easy in Dedukti thanks to the user-defined syntax and rewrite rules.

**The signature.** The encoding uses function symbols to represent sorts, types and terms of CIC. Knowledge of CCU is assumed to understand the rules. Appropriate expositions are [2, 3, 26].

The specification has 3 type constructors, Sort, U and T. The name Sort comes from PTSs. Other declared symbols allow us to build objects and the 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 call them 0, 1, ..., so as to be represented by a copy of the natural numbers generated by three constructors, 0, 1 and +. This perhaps uncommon presentation of the natural numbers 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. Finally, \( u(i) \), \( \tilde{\eta}(i, a) \), \( \tilde{\gamma}(i, a) \) and \( \pi(i, j, a, b) \) are codes for respectively, the type \( U(i + 1) \), the type lifted one level from some type \( U(i) \), the type lifted \( n \) level from the type \( U(0) \), and for \( U \)-types.

Variables \( i, j \) are of type Sort. Variables \( a \) and \( b \) are used as the third and fourth argument of \( \pi \) respectively.

\[
\begin{align*}
\text{Sort} & : * \\
0, 1 & : \text{Sort} \\
+^2 & : \Pi i : \text{Sort} . \Pi j : \text{Sort} . \text{Sort} \\
\text{max} & : \Pi i : \text{Sort} . \Pi j : \text{Sort} \rightarrow \text{Sort} \\
\text{rule} & : \Pi i : \text{Sort} . \text{Sort} \\
U^1 & : \Pi i : \text{Sort} . * \\
T^2 & : \Pi i : \text{Sort} . \Pi a : U(i) . * \\
U^1 & : \Pi i : \text{Sort} . U(i + 1) \\
\tilde{\eta}^2 & : \Pi i : \text{Sort} . \Pi a : U(i). U(i + 1) \\
\tilde{\gamma}^2 & : \Pi i : \text{Sort} . \Pi a : U(0). U(i) \\
\pi^4 & : \Pi i : \text{Sort} . \Pi j : \text{Sort} . \Pi a : U(i). \Pi b : (\Pi x : T(i, a). U(j)). U(\text{rule}(i, j)) \\
\text{The rewrite system.} & \\
\text{The constructor} + & \text{is associatif, commutatif, and has 0 as identity element. Since it is a constructor, that is, it never occurs as the head of a lefthand side of rules, no rule extension is needed [15]. We take the liberty to use an infix notation for +, the abbreviation 2 for 1 + 1, and a varadic number of arguments to ease the readability of sums.}
\end{align*}
\]

\[
\begin{align*}
1 : & \text{max} (i, i + j) \rightarrow i + j \\
2 : & \text{max} (i + j, j) \rightarrow i + j \\
3 : & \text{rule} (i, 0) \rightarrow 0 \\
4 : & \text{rule} (i, j + 1) \rightarrow \text{max} (i, j + 1) \\
5 : & \tilde{\eta}(0, a) \rightarrow i \\
6 : & \tilde{\gamma}(i + 1, a) \rightarrow \tilde{\eta}(i, \tilde{\eta}(i, a))
\end{align*}
\]

The rules for \( \text{max} \), \( + \) and \( \tilde{\eta} \) are self explanatory. Note that the rewrite rules for rule encode the impredicativity of Prop, encoded by 0 of type Sort. The name rule comes from PTSs again.
7: \( \pi(i + 1, i + j + 1, \uparrow(i, a), b) \rightarrow \pi(i, i + j + 1, a, b) \)
8: \( \pi(i + j + 2, j + 1, \uparrow(i + j + 1, a, b) \rightarrow \uparrow(i + j + 1, \pi(i + j + 1, j + 1, a, b)) \)
9: \( \pi(i + j + 2, j + 2, a, \lambda x : T(i + j + 2, a), \uparrow(j + 1, (b x)) \rightarrow \pi(i + j + 2, j + 1, a, b) \)
10: \( \pi(i, i + j + 1, a, \lambda x : T(i + j + 1, a), \uparrow(i + j, (b x)) \rightarrow \pi(i + j, \pi(i, i + j, a, b)) \)
11: \( \pi(i + 1, 1, a, \lambda x : T(1, a), \uparrow(0, (b x)) \rightarrow \uparrow(i + 1, \pi(i + 1, 0, a, b)) \)
12: \( \pi(0, 1, a, \lambda x : T(1, a), \uparrow(0, (b x)) \rightarrow \pi(0, 0, a, b) \)
13: \( \pi(i + 1, 0, \uparrow(i, a), b) \rightarrow \pi(i, 0, a, b) \)

The rules for \( \pi \) are delicate and chosen so as to ensure that types have a unique encoding, a property that is crucial for showing the preservation of typing [2, 3]. Their design obtained by comparing (via \( \pi \)) the first two arguments of \( \pi \) ensures that they have few critical pairs (eliminating them all would require a richer language involving a comparison operator and conditional rewrite rules — whose confluence checking is surely less understood. On the other hand, the complex type of \( b \) which depends on both first arguments of \( \pi \) impacts their formulation as explained now.

In rules 7, 8 and 13, the first argument of \( \pi \) decreases by 1 in the recursive call while the second argument is unchanged. Typing these rules requires then using conversions with rule 15. For an example, consider rule 7. The type of \( \pi \) tells us that in the lefthand side, \( b : \Pi \) \( X \rightarrow T(i + 1, \uparrow(i, a)).U(i + j + 1) \). In the righthand side, we get \( b : \Pi \) \( X \rightarrow T(i, a).U(i + j + 1) \). These two types are different, but convertible thanks to rule 15.

In rules 9, 10, 11 and 12, the second argument of \( \pi \) decreases by 1 in the recursive call while the first remains unchanged. Typing these rules requires having an explicit abstraction in the lefthand side, so that \( b \) can have the appropriate type in the righthand side.

We are left with the rules for \( T \), which are standard decoding rules whose lefthand and righthand sides have type \( \ast \):

14: \( T(i + 1, u(i)) \rightarrow U(i) \)
15: \( T(i + 1, \uparrow(i, a)) \rightarrow T(i, a) \)
16: \( T(0, \uparrow(i, a)) \rightarrow T(0, a) \)
17: \( T(0, \pi(i, 0, a, b)) \rightarrow \Pi \) \( X : T(i, a).U(0, (b x)) \)
18: \( T(i + j, \pi(i, i + j, a, b)) \rightarrow \Pi \) \( X : T(i, a).U(i + j + 1, a, b) \)
19: \( T(i + j + 1, \pi(i + j + 1, j + 1, a, b)) \rightarrow \Pi \) \( X : T(i + j + 1, a, b) \)

These rewrite rules have been obtained with the crucial help of MAUDE. They differ slightly from those presented in [3], where an infinite number of symbols is used.

The symbols \( \text{Sort}, 0, 1, +, \max, \text{rule}, U, T, u, \pi \) and the rules 1–4, 14, 17–19 define an embedding of the system CCU\( ^\infty \) (the calculus of constructions with universes but without cumulativity).

Since terms without universe variables in the embedding presented here are clearly in exact correspondence with those in the embedding presented there, Theorem 5.2.11 and 6.2.27 of [3] prove that this embedding is faithful, that is, there are functions \( M \) and \( A \) that faithfully and conservatively translate the terms of CCU\( ^\infty \) into the terms of \( \Lambda \)\( \Lambda \)Mod with this signature.

The symbols \( \uparrow, \uparrow \) and the rules 5–13, 15–16 add the cumulativity to the previous system. Therefore, according to Theorem 9.2.16 of [3], the obtained full system is an embedding of CCU\( ^\infty \) which faithfulness is still a very likely conjecture tackled in a forthcoming paper.

**A Church-Rosser encoding.** This set of rules (and equations for \( + \)) has been obtained with the crucial help of the MAUDE system by hiding the higher-order aspects of some of the rules, as described in Appendix where the MAUDE input file is given. Finding the appropriate arithmetic (the first 6 rules) for universes was the most delicate part. The other rules were either given or resulted from the use of MAUDE’s confluence checker.

We now show that these rules satisfy our assumptions.

First, we split the signature and rules according to our categories. The rules for \( \Pi \) and \( T \) contain functional symbols, they are higher-order. Since the other rules contain no functional symbols, nor \( \Pi, T \), they can be taken as being first-order. Our choice is then to take \( \Sigma = \{ 0, 1, +, \max, \text{rule} \} \), \( \Sigma_0 = \{ \uparrow, \uparrow \} \), and \( \Sigma_C = \{ T, \pi, u, U \} \).

Assumptions 3 and 4 are then satisfied as well. For Assumption 2, we must show termination in AC equivalence classes and check that the ACZ-critical pairs are joinable by rewriting. Termination can be achieved easily by Rubio’s fully syntactic AC path ordering [25]. This order works much like Dershowitz’s rewrite ordering, but has a specific status for AC operators that performs additional monotonicity checks. All symbols can have a multi-status, but rule whose status must be lexicographic while the precedence can be rule \( \max \) \( + \) \( > \) \( 1 \) \( > \) \( 0 \). Routine calculations show termination. Checking the CPs of \( R_0 \) is routine too, there is only a trivial one between the two max rules.

We are left with the assumptions about the higher-order rules.

Apart from \( a, b, \) all variables in the higher-order rules are of type \( \text{Sort} \), hence are confined. While \( b \) occurs linearly in their lefthand sides, \( a \) occurs non-linearly in lefthand sides of rules 9-12, hence violating Assumption 5. Since types need only be convertible in \( \Lambda \)\( \Lambda \)Mod, we replace these rules by rules 9' to 12' obtained by moving the type expression in a condition, for example:

\[ 9' : \quad \pi(i + j + 2, j + 2, a, \lambda x : Y.\uparrow(j + 1, (b x)) \rightarrow \pi(i + j + 2, j + 1, a, b) \text{ if } Y = \cdots T(i + j + 2, a) \]

where \( = \cdots \) is interpreted as convertibility. Since any rewrite operating on \( T(i + j + 2, a) \) preserves convertibility, confluence of the conditional typed rules reduces to the confluence of the untyped rules 9" to 12", where these conditions are dropped, as shown with:

\[ 9'' : \quad \pi(i + j + 2, j + 2, a, \lambda x : Y.\uparrow(j + 1, (b x)) \rightarrow \pi(i + j + 2, j + 1, a, b) \]

What this informal argument shows is that replacing the types of the bound variables in the rules is faithful with respect to confluence, that is, the confluence of the transformed rules implies the confluence of the typed rules.

We are left with our last assumption that the higher-order critical pairs, as well as the mixed higher-order/first-order pairs, of these modified rules have (strongly) decreasing diagrams. All these critical pairs have been computed with MAUDE, while their decreasing diagrams have been computed by hand. These computations are summarized in Figures 6 and 7. The first column lists the critical pairs as given by MAUDE (see Appendix), the second column lists the real overlaps, the third the joinability diagram for the corresponding critical pair, and the last the constraints generated on the rules’ indices in order to obtain a decreasing diagram.

Note that the higher-order rewrites with rules 17, 18 and 19, for which \( (b x) \) occur in the righthand side, cause a beta’-normalization step in case the variable \( b \) is instantiated by an ab-
Critical overlaps

<table>
<thead>
<tr>
<th>Equation</th>
<th>Critical top overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 = p_3 )</td>
<td>( \pi(i + 2, i + 1, a, \lambda x : Y. \hat{\pi}(i + 1, (b x))) )</td>
</tr>
<tr>
<td>( p_2 = p_3 )</td>
<td>( \pi(i + j + 3, i + 2, \hat{\pi}(i + j + 1, a), \lambda x : Y. \hat{\pi}(i + 1, (b x))) )</td>
</tr>
<tr>
<td>( p_4 = p_1 )</td>
<td>( \pi(i + 1, i + j + 2, \hat{\pi}(i, a), \lambda x : Y. \hat{\pi}(i + 1, (b x))) )</td>
</tr>
<tr>
<td>( p_5 = p_1 )</td>
<td>( \pi(1, 1, \hat{\pi}(0, a), \lambda x : Y. \hat{\pi}(0, b)) )</td>
</tr>
<tr>
<td>( p_5 = p_2 )</td>
<td>( \pi(i + j + 1, 1, \hat{\pi}(i + 1, a), \lambda x : Y. \hat{\pi}(0, b)) )</td>
</tr>
<tr>
<td>( t_4 = t_5 )</td>
<td>( T(0, \pi(0, a, b)) )</td>
</tr>
<tr>
<td>( t_5 = t_6 )</td>
<td>( T(j + 1, \pi(j + 1, j + 1, a, b)) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equation</th>
<th>Critical subterm overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(0, t_1) = t_3 )</td>
<td>( T(0, \hat{\pi}(0, a)) )</td>
</tr>
<tr>
<td>( T(i + 1, t_2) = t_3 )</td>
<td>( T(i + 1, \hat{\pi}(i + 1, a)) )</td>
</tr>
<tr>
<td>( T(0, p_7) = t_4 )</td>
<td>( T(0, \pi(i + 1, 0, \hat{\pi}(i, a), b) )</td>
</tr>
<tr>
<td>( T(i + j + 1, p_1) = t_5 )</td>
<td>( T(i + j + 1, \hat{\pi}(i, a), b) )</td>
</tr>
<tr>
<td>( T(i + 2, p_3) = t_5 )</td>
<td>( T(i + 2, \pi(i + 2, i + 2, a, \lambda x : Y. \hat{\pi}(i + 1, (b x))) )</td>
</tr>
<tr>
<td>( T(i + j + 1, p_4) = t_5 )</td>
<td>( T(i + j + 1, \hat{\pi}(i + j + 1, a, \lambda x : Y. \hat{\pi}(i + 1, (b x))) )</td>
</tr>
<tr>
<td>( T(1, p_5) = t_6 )</td>
<td>( T(1, \pi(1, i, a, \lambda x : Y. \hat{\pi}(0, b))) )</td>
</tr>
<tr>
<td>( T(i + 1, p_1) = t_6 )</td>
<td>( T(i + 1, \pi(i + 1, a, \lambda x : Y. \hat{\pi}(0, b))) )</td>
</tr>
<tr>
<td>( T(i + j + 2, p_2) = t_6 )</td>
<td>( T(i + j + 2, \pi(i + j + 2, i + j + 1, \hat{\pi}(i + j + 1, a, b))) )</td>
</tr>
<tr>
<td>( T(i + j + 2, p_3) = t_6 )</td>
<td>( T(i + j + 2, \pi(i + j + 2, i + j + 1, \hat{\pi}(i + j + 1, a, b))) )</td>
</tr>
<tr>
<td>( T(i + 1, p_5) = t_6 )</td>
<td>( T(i + 1, \pi(i + 1, i, 1, a, \lambda x : Y. \hat{\pi}(0, (b x))) )</td>
</tr>
</tbody>
</table>

A partial order making all these local peaks decreasing is the following: \( p_1 > p_5 \neq 1 > l_j > t_k \).

**Figure 6.** Decreasing diagrams for critical pairs

All these advances were however needed by our example of encoding a cumulative hierarchy of universes in a dependent type theory. The solutions were actually dictated by a careful analysis of the example itself, especially the notion of confinement which would hardly have popped up otherwise.

Along this work, we kept adding assumptions that we believe are non-essential, in order to facilitate technicalities, in particular the proof of the confluence theorem. The assumptions that we know or believe are non-essential are the following:

1. There is a single non-terminal in the grammar describing the set of untyped terms. Having several would not make any change.

2. First-order equations and simplifiers are built from the confined sub-signature and there are no higher-order equations. There could be additional non-confined equations in the first-order signature, this extension should be slightly more complex to carry out since equotons would now operate on non-confined sub-expressions. Similarly, adding equations for higher-order expressions, like associativity and commutativity of some higher-order symbol, would impact both the proofs, for the same reason as before, and the algorithmics: higher-order AC-unification would then be needed [10], which is not as bad as it looks since we have restricted ourselves to linear patterns in lefthand sides of higher-order rules.

3. Non-confined variables appear linearly on both sides of first-order rules. We conjecture that non-confined variables may appear non-linearly in righthand sides, but do not know how to adapt the current proof to this weaker assumption.

**Theorem 6.1.** The dependent type theory ΛIMod equipped with the encoding of the cumulative hierarchy of predicative universes is Church-Rosser.

**7. Conclusion**

We have described a complex, powerful confluence result for dependent type theories, which, like the one underlying Dedukti, admits user-defined rewrite rules. Our target accumulates difficulties: rewriting modulo an algebraic theory, higher-order rewriting based on higher-order pattern matching, combinations of these, non-linear variables in both the lefthand and righthand side of rules, and critical pairs modulo various theories. Many of these features alone are new in the context of β-rewriting untyped terms. The latter two, in particular, are known to raise important difficulties regarding confluence tests.
4. Righthand sides of higher-order rules cannot be variables nor abstractions. Lifting this restriction would require important technical changes. A weaker definition of higher-order rewriting a term $u$ with rule instance $L \gamma \rightarrow R \gamma$ in which the $\beta$-steps from $L \gamma$ to $u$ are exactly those needed to eliminate the head applications of the pre-reduces $(X \gamma \rightarrow \pi)$ should help.

5. More generally, many assumptions aim at preserving rewrites under substitution instance, that is, eliminating the need of normalizations (with $\beta$- or $\eta$-rules) each time a rewrite step is instantiated (preserving substitutions, strongly decreasing reductions, etc). We believe it is possible to carry around these normalizations in proofs by using the property that the rewrite system $(S, E)$ is Church-Rosser and terminating. This would then force us to use the general version of decreasing diagrams for which rewrites can go both ways.

6. We have not considered the $\eta$-rule in this work. Adding it should not raise many difficulties since algebraic symbols have a fixed arity in $\lambda\Pi\text{Mod}$.

We believe that future encodings will force us to remove or weaken these assumptions in turn.

On the other hand, we doubt that any of the other assumptions can be removed. The termination assumption for $R_0$, although natural and required for type checking, came in late. It relates to the existence of non-linear (hence confined) variables in both the lefthand and righthand sides of higher-order rules, making it impossible to find decreasing diagrams for some peaks without it.

Finally, one may question our encoding of Coq’s universes in the first place. In Coq’s implementation, universes are floating. We could mimick floating universes by using rewriting under constraints. Unfortunately, confluence under constraints is hard, although restricting the constraints to confined types would surely help. Similarly, we did not use conditional rules, whose confluence is even more delicate, although this would have simplified our definition of $\pi$. Results exist in the first-order case [20], but are very limited in the higher-order one [8].

References


Appendix: Using MAUDE

Checking the local confluence of a rewrite system modulo associativity, commutativity, and identity is a long and delicate task that is prone to errors when done manually. We verified the local confluence of our rewrite system with the help of the MAUDE Formal Environment. Since MAUDE only supports first-order rewrite systems, we slightly adapted our rewrite system to hide the higher-order nature of some symbols. This transformation does not impact the computation of critical pairs in the sense that the higher-order critical pairs can then be easily computed from the first order ones computed by MAUDE (there are many more). Here is the resulting MAUDE specification:

```maude
fmod LPM is
  sort Nat .
  sort Term .
  sort Type .
  op 0 : -> Nat [ctor] .
  op 1 : -> Nat [ctor] .
  op _+_: Nat Nat -> Nat [comm assoc id: 0 ctor] .
  op max : Nat Nat -> Nat .
  op rule : Nat Nat -> Nat .
  op u : Nat -> Term .
  op lift : Nat Term -> Term .
  op liftn : Nat Term -> Term .
  op pi : Nat Nat Term Term -> Term .
  op U : Nat -> Type .
  op T : Nat Term -> Type .
  op Pi : Type Type -> Type .
vars i j k l m x : Nat .
vars a b : Term .
edfm
```

This system can then be checked using the Checker tool of MAUDE: `(select tool CRC .)` and `(ccr LPM .)`.

The versions of the tools we used in our stack are: MAUDE 2.6 (Oct 13 2011) ; Full MAUDE 2.6.1e (July 13th 2012) ; MAUDE Formal Environment 1.0 ; Church-Rosser Checker 3m (July 7th 2012).