Abstract

In this paper we present the algebraic-\(\lambda\)-cube, an extension of Barendregt’s \(\lambda\)-cube with first- and higher-order algebraic rewriting. We show that strong normalization is a modular property of all systems in the algebraic-\(\lambda\)-cube, provided that the first-order rewrite rules are non-duplicating and the higher-order rules satisfy the general schema of Jouannaud and Okada. This result is proven for the algebraic extension of the Calculus of Constructions, which contains all the systems of the algebraic-\(\lambda\)-cube.

We also prove that local confluence is a modular property of all the systems in the algebraic-\(\lambda\)-cube, provided that the higher-order rules do not introduce critical pairs. This property and the strong normalization result imply the modularity of confluence.

1 Introduction

Many different computational models have been developed and studied by theoretical computer scientists. One of the main motivations for the development of such models is no doubt that of isolating particular aspects of the practice of computing, in order to better investigate them, so allowing either to tune existing programming languages or to devise new ones. However, the study of computational models cannot exploit all its possibilities to help the development of actual computing tools unless also their interactions and possible (in)compatibilities are investigated. In this framework, many research efforts have been devoted in the last years to the study of the interactions between two closely related models of computation: the one based on \(\beta\)-reduction on \(\lambda\)-terms and the one formalized by means of rewrite rules on algebraic terms. These particular models are relevant for the study of two aspects of programming languages: higher-order programming and data types specification. The combination of these two models has also provided an alternative in the design of new programming languages: the algebraic functional languages [20]. These languages allow algebraic definitions of data types and operators (as in equational languages like OBJ) and definition of higher-order functions (as in functional languages like ML), in a unified framework.

The study of systems based on \(\lambda\)-calculi and algebraic rewriting has been carried out both in untyped and typed contexts. If no type discipline is imposed
on the languages the interactions between these computational models raise several problems, as shown in [21] and [14]. For typed languages (typed versions of \(\lambda\)-calculus and typed term rewriting systems) things work out nicely. In [8] and [24] it is shown that the system obtained by combining a terminating first-order many-sorted term rewrite system with the second order typed \(\lambda\)-calculus is again terminating with respect to \(\beta\)-reduction and the algebraic reductions induced by the rewrite rules, i.e., strong normalization is a modular property in this case. The same result is proven for confluence in [9]. In [20] both results are extended to combinations of first- and higher-order rewriting systems with second order \(\lambda\)-calculus, under certain conditions on the form of the rewrite rules.

The question that naturally arises is whether such a nice interaction between typed \(\lambda\)-calculi and algebraic rewriting is independent of the power of the type discipline. More precisely, the question is whether the existing results extend to higher-order type disciplines such as the Calculus of Constructions of Coquand and Huet [12]. Even more, one could wonder if such interaction is “well-behaved” also in case one considers not only first-order algebraic rewriting, but the powerful form of higher-order algebraic rewriting defined in [20] as well. Indeed, considering only first-order algebraic rewriting, this problem has already been addressed in [1]. A strong restriction was however imposed on the definition of the combined system: in the conversion rule only \(\beta\)-conversion \((\gamma = \beta)\) was considered as equality. So, even if there was a rewrite rule \(x + 0 \to x\) in the system, two types of the form \(P(x)\) and \(P(x + 0)\) were not considered to be the same (where \(P\) is a type depending on natural numbers). Such a choice was motivated mainly by the essential use of the property of confluence in the proof of the modularity of strong normalization: confluence does not hold in general for \(R\beta\)-equality, where \(R\) is the given set of first-order algebraic rules.

In this paper we extend the Calculus of Constructions, adding not only first- but also higher-order algebraic rewriting, and considering in the type conversion rule \((conv)\) the \(R\beta\)-equality generated by the algebraic reductions together with \(\beta\)-reduction. Considering \(R\beta\)-equality, a proof of strong normalization can no longer rely on the confluence property. Actually, also other properties of the metatheory of the system, like Subject Reduction, which in the case of the pure Calculus of Constructions are proven using confluence, will have to be proven independently of confluence in this extension.

In fact, using \(R\beta\)-equality in \((conv)\) even the definition of the system is more involved. Indeed the rule \((conv)\) is part of the definition of the terms of the system and one cannot define a notion of \(R\beta\)-equality to be used in the rule unless one knows what the terms are. We have then to cope with a circularity, which can be solved in two ways, either by defining the system by levels, starting from the pure Calculus of Constructions, or by defining algebraic rewriting on terms of the pure calculus enriched with algebraic constants, and using this relation on pseudoterms in the rule \((conv)\).

The second solution, to be better discussed later, is the one we have chosen in the present paper where, for sake of uniformity, we provide a definition of the extension with first- and higher-order (in the sense of [20]) algebraic rewriting on all the systems of the so-called \(\lambda\)-cube [6], [4]. This extension will be called algebraic-\(\lambda\)-cube \((\lambda R\beta\)-cube for short).

The main result we prove for the systems of the \(\lambda R\beta\)-cube is the modularity of the strong normalization property, i.e., that the systems are strongly normalizing in case the first-order algebraic rules are so on algebraic terms (the higher-order rules we will use are strongly normalizing because of their structure).

As said before, we had to cope with the problem of not having at hand the property of confluence. We solved such a problem by extending some technical results devised in [15] (see also [16]). We prove strong normalization in three steps. By means of a reduction preserving translation we prove strong normalization of the extended Calculus of Constructions to be implied by the same property of system \(\lambda_{R\beta}\) (the extended polymorphic typed \(\lambda\)-calculus of order \(\omega\)). The strong normalization property of this last system was proven in [3] by a reduction preserving translation, showing it to be implied by the same property of system \(\lambda_{AR}\) (a type assignment system for \(\lambda\)-calculus with intersection types and algebraic rewriting), which in turn was proven strongly normalizable in [2].

Finally, we will prove that local confluence is a modular property provided that the higher-order rules do not introduce critical pairs. This, and the previous strong normalization result, imply the modularity of confluence in the systems of the \(\lambda R\beta\)-cube.

We assume the reader familiar with the basic notions and notations of Pure Type Systems and the \(\lambda\)-cubes as presented in [6], [16], [4]. The algebraic extension of the \(\lambda\)-cube will be discussed in Section 2. Section 3 will be devoted to the metatheory of the \(\lambda R\beta\)-cube. We will then outline the strong normalization proof in Section 4. In Section 5 we prove the modularity of confluence. Section 6 contains the conclusions.
2 Adding Algebraic Rewriting to the \(\lambda\)-cube

The \(\lambda\)-cube is a coherent collection of eight type systems. Each system (generically denoted by \(\lambda_n\)) is placed on a vertex of the cube in a way that geometrically exploits the possible "dependencies" between types and terms. Each of the possible directions in the three-dimensional space in which the cube is corresponds to a particular dependency.

As said in the introduction, we wish to modify the definition of the \(\lambda\)-cube in order to have also algebraic features. We begin by considering a denumerable set \(S\) of sorts: \(S = \{s_1, s_2, \ldots\}\). The elements of \(S\) denote algebraic base types. So, first of all, we add to the rules of the cube the following axiom: \((\text{alg1})\) \(\vdash s : x\), for each \(s_i \in S\). We define now, by induction, a set of algebraic types.

Definition 2.1 (Algebraic types) The set \(T_S\) of algebraic types on \(S\) is inductively defined as follows:

- If \(s \in S\) then \(s \in T_S\)
- If \(\sigma, \tau \in T_S\) then \(\Pi x:\sigma.\tau \in T_S\)

We will call first-order algebraic types the elements \(\sigma_1 \to \ldots \to \sigma_n \to \sigma \in T_S\) such that \(\sigma, \sigma_i \in S\) (1 \(\leq\) i \(\leq\) n). A context \(\Gamma = (x_1: A_1, \ldots, x_n: A_n)\) is called algebraic if \(A_i \in T_S\) (1 \(\leq\) i \(\leq\) n).

The next step is to consider, for each algebraic type, a set of function symbols of that type, i.e., a signature \(\mathcal{F} = \bigcup_{\tau \in T_S} \mathcal{F}_\tau\), where \(\mathcal{F}_\tau\) denotes the set of function symbols of type \(\tau\). We assume \(\mathcal{F}_\tau \cap \mathcal{F}_{\tau'} = \emptyset\) if \(\tau \neq \tau'\). Each function symbol \(f\) in \(\mathcal{F}\) is assumed to have an arity which, when it will be necessary, will be denoted by superscripts \((f^n)^1\). The introduction of the signature is naturally expressed in the framework of the cube by adding, for each \(f \in \mathcal{F}_\sigma\), the following axiom: \((\text{alg2})\) \(\vdash f : \sigma\). A function symbol \(f^n\) is called first-order if it has a first-order algebraic type \(s_1 \to \ldots \to s_n \to s\) and \(n \equiv m\). Function symbols which are not first-order will be called higher-order.

We have now to extend the notion of pseudoterm with sorts and function symbols, i.e., pseudoterms are defined by

\[
T ::= x \mid f \mid s \mid x \mid \square \mid TT \mid \lambda x:TT \mid \Pi x:TT
\]

where \(f\) and \(s\) range over \(\mathcal{F}\) and \(S\), respectively. Intuitively, \(x\) and \(\square\) denote respectively the set of types and kinds. The set \(\{x, \square\}\) will be called \(PK\).

A function symbol \(f^n\) is said to be saturated in a pseudoterm \(M\) if any occurrence of its appears in subterms of the form \(f_{P_1} \ldots P_m\) with \(m \equiv n\).

We have now to define the notion of algebraic term, i.e., the natural translation of the notion of term rewriting systems. This notion has to be defined in the setting of the \(\lambda\)-cube since its algebraic extension is being defined.

Definition 2.2 (Algebraic Terms) (i) A pseudoterm is algebraic if it is formed only by variables and function symbols of the signature.

(ii) An algebraic term in \(\Gamma\) (in a system \(\lambda_\gamma\)), for \(\Gamma \in \text{Context}(\lambda_\gamma)\), is an algebraic pseudoterm \(t\) such that \(\Gamma \vdash t : A\). Any \(f \in \mathcal{F}\) is saturated in \(t\), and

\[
\forall x \in \text{FV}(t), [x : B \in \Gamma \Rightarrow \exists \sigma \in T_S. B \equiv_\gamma \sigma].
\]

(iii) A first-order algebraic term \(t\) (in \(\Gamma\)) is an algebraic term (in \(\Gamma\)) such that any \(f \in \mathcal{F}\) occurring in \(t\) is first-order and there is no subterm of \(t\) of the form \(xP\).

It is easy to see that the \(\sigma\) of (ii) of the above definition is unique. Notice that it is not possible to speak of algebraic terms independently of contexts. However, if we restrict to first-order algebraic terms we can avoid contexts, as the following lemma shows.

Lemma 2.3 In any system \(\lambda_\gamma\), if \(t\) is a first-order algebraic term in \(\Gamma\) (and \(t\) is not a variable) then for any \(\Gamma'\) such that \(\Gamma' \vdash t : A, t\) is algebraic in \(\Gamma'\).

We will now define the notion of rewrite rule and on top of that the notion of rewrite relation. A rewrite rule will be a pair of algebraic terms. Since we are going to use the induced rewrite relation as part of the definition of term (in the conversion rule), then, in order not to create a circularity, we will define it on pseudoterms\(^2\). Dealing with pseudoterms the context \(\Gamma\) turns out to have no sense, so we will impose that \(r : t \to t'\) is a rewrite rule only if, for any context \(\Gamma\), \(t\) is algebraic in \(\Gamma\) whenever it is typable in it.

\(^2\) Of course one could define the reduction relation on terms of the algebraic extension of the \(\lambda\)-cube simply by stratifying its definition, starting from the \(\lambda\)-cube and defining the rule (conv) for the level \(i\) using the rewrite relation defined at level \(i - 1\). The final system would then be the limit of such a chain of systems. Our choice is however motivated, with respect to this one, by its simplicity.
is not a real restriction, since all first-order terms and higher-order terms of a relevant class satisfy it. We will call such terms “rewritable”.

**Definition 2.4** A term is rewriting if it is algebraic in some context \( \Gamma \) and for any \( x \in \text{FV}(t) \) there exists a subterm \( fP_1 \ldots P_k \) of \( t \) such that \( f \in \mathcal{F} \) and \( P_j \equiv x \) for some \( 1 \leq j \leq k \).

We can now use the notion of rewritable term to define that of rewrite rule.

**Definition 2.5** A rewrite rule \( r \) is a pair \( (t,t') \), such that \( t,t' \) are algebraic terms (for some contexts), \( t \) is rewritable\(^3\), \( \text{FV}(t') \subseteq \text{FV}(t) \), and, for any context \( \Gamma \) and pseudoterm \( A, \Gamma \vdash t : A \Rightarrow \Gamma \vdash t' : A \). A rewrite rule will be denoted by \( r : t \to t' \).

A first-order rewrite rule is a rewrite rule \( r : t \to t' \) where both \( t \) and \( t' \) are first-order algebraic terms. A higher-order rewrite rule is a rewrite rule which is not first-order. We will generalize the notion of rewrite rule by allowing also \( \lambda \)-abstractions in the right-hand side. We have then \( \lambda \)-higher-order rewrite rules\(^4\).

Given a set \( R \) of rewriting rules, we denote by \( \text{FOR} \) and \( \text{HOR} \) the subsets of first-order and higher-order rules of \( R \), respectively. A rewrite rule induces a rewriting relation on pseudoterms as follows.

**Definition 2.6** Let \( M \) and \( N \) be pseudoterms. \( M \to_r N \) iff there exists a rewrite rule \( r : t \to t' \), a context \( C[t]\) and a substitution \( \varphi \) such that \( M \equiv C[t] \varphi \) and \( N \equiv C[t'] \varphi \). \( \to_r \) denotes the reflexive and transitive closure of \( \to_r \).

If \( R \) is a set of rewrite rules we define

\[
M \to_R N \Leftrightarrow \exists r \in R. M \to_r N
\]

and

\[
M \to_{\beta} N \Leftrightarrow M \to_R N \vee M \to_{\beta} N.
\]

Once one specifies a set \( \mathcal{S} \) of sorts, a signature \( \mathcal{F} \) and a set \( R \) of rewriting rules, it would seem that to define the algebraic extension of the \( \lambda \)-cube specified by \( (\mathcal{S}, \mathcal{F}, R) \), it suffices, besides the additional axioms for sorts and function symbols given before, to replace the rule (conv) of the pure \( \lambda \)-cube by the following one

\[
\Gamma \vdash A : B \quad \Gamma \vdash B' : p \quad B \equiv_{\beta} B'
\]

where \( \equiv_{\beta} \) is the least congruence containing \( \to_{\beta} \).

This, however, would not work. In the pure cube if we have \( A \equiv_{\beta} B \), then the Church-Rosser property of \( \equiv_{\beta} \), together with the property of subject-reduction, ensures that \( A \) and \( B \) are always equal via \( \beta \)-reductions and \( \beta \)-expansions that remain inside the set of well-typed terms. It is easy to realize, however, that we cannot rely, in general, on the Church-Rosser property for \( \equiv_{\beta} \). Therefore, we cannot consider \( \equiv_{\beta} \) in the (conv) rule. We have instead to consider the \( \beta \)-reduction relation.

The complete definition of algebraic extension of the \( \lambda \)-cube runs now as follows.

**Definition 2.7 (The \( \lambda R \)-cube)** Let \( \mathcal{S} = \{ s_1, s_2, \ldots \} \) be a set of sorts, \( \mathcal{F} = \{ f_1, f_2, \ldots \} \) a signature on \( \mathcal{S} \) and \( R \) a set of rewriting rules\(^5\).

The \( \lambda (\mathcal{S}, \mathcal{F}, R) \)-cube (\( \lambda R \)-cube for short) is defined by adding the following axioms to the axioms of the \( \lambda \)-cube

\[
(alg 1) \quad t : \ast \quad \text{for any } s \in \mathcal{S}
\]

\[
(alg 2) \quad f : \sigma \quad \text{for any } f \in \mathcal{F}
\]

and by replacing the following rule for the (conv) rule

\[
(\text{red}_{\beta}) \quad \Gamma \vdash M : A \quad \Gamma \vdash B : s \quad A \to_{\beta} B \text{ or } B \to_{\beta} A
\]

The eight systems of the \( \lambda R \)-cube will be called \( \lambda R_{\ast}, \lambda R_2, \lambda R_{\ast 2}, \lambda R_{\ast \omega}, \lambda R_{2 \omega}, \lambda R_{\ast 2 \omega} \) and \( \lambda R_{2 \omega} \) (or \( \lambda R_{\mathcal{C}} \)).

Then, graphically the \( \lambda R \)-cube turns out to be as follows

---

\(^3\)This condition subsumes the usual condition “\( t \) not a variable” for rewrite rules.

\(^4\)The notion of \( \lambda \)-higher-order rewrite rule does not match the usual notion of rewrite rule, being more general. We introduced it since the result we will obtain holds also for a particular class of such rewrite rules.

\(^5\)Recall that the notion of rewriting rule is independent from contexts and systems.
\(\lambda\)-cube. We will denote by \(\lambda_{R^\bullet}\) a generic system of the \(\lambda R\)-cube.

Some systems of the \(\lambda R\)-cube are already present in the literature. In particular, when \(\text{HOR}\) is empty, \(\lambda_{R^\bullet}\) is the system studied in [7] and [24], while \(\lambda R_3\) is equivalent to the system defined by Breau-Tannen and Gallier in [8]. The systems of [20] correspond to \(\lambda_{R^\bullet}\) and \(\lambda R_m\). We have already mentioned in the introduction which results were proved for these systems.

Now that we have defined the \(\lambda R\)-cube, we can say what is an algebraic term (in \(\Gamma\)) in the \(\lambda R\)-cube (the notion of algebraic term used to define it was for the \(\lambda\)-cube).

**Definition 2.8** An algebraic term (in \(\Gamma\)) in the \(\lambda R\)-cube is an algebraic pseudoterm such that \(\Gamma \vdash_{R^\bullet} t : A\) and

\[
\forall x \in \text{FV}(t). [x : B \in \Gamma \Rightarrow B = R_3 \tau \in \text{Ts}].
\]

Moreover each occurrence of function symbols has to be saturated in \(t\).

We are interested in the strong normalization property for the systems of the \(\lambda R\)-cube. However, if unrestricted terminating higher-order rewrite rules are considered it can be easily shown that this property fails. Then, following [20], we consider higher-order rules that always terminate on algebraic terms thanks to their structure: a generalization of primitive recursion called general schema.

Higher order rewrite rules satisfying the general schema are of wide use in the practice of higher-order rewriting and can be considered as definitions of new functionals of a language.

**Definition 2.9 (The general schema [20])**

A higher-order rewrite rule \(r : t \rightarrow t'\) satisfies the general schema w.r.t. FOR if it is of the form

\[
F[\tilde{X}, \tilde{Z}]\tilde{Y} \rightarrow v[(Fr_1[\tilde{X}, \tilde{Z}]\tilde{Y}), \ldots, (Fr_m[\tilde{X}, \tilde{Z}]\tilde{Y})], \tilde{X}, \tilde{Z}, \tilde{Y}]
\]

where \(\tilde{X}\) and \(\tilde{Y}\) are sequences of higher-order variables and \(\tilde{Z}\) is a sequence of first-order variables, and such that

1. \(\tilde{X} \subseteq \tilde{Y}\)

2. \(F\) is function symbol that can appear neither in \(\tilde{t}\), \(\tilde{r}_1\), \ldots, \(\tilde{r}_m\), nor in the rules of FOR, and its occurrences in \(v\) are only the ones explicitly indicated

\[\text{Note that this condition ensures that } F[\tilde{X}, \tilde{Z}]\tilde{Y} \text{ is rewritable.}\]

3. \(\tilde{r}_1, \ldots, \tilde{r}_m\) are terms of sort type

4. \(\forall i \in [1..m], \tilde{t} \triangleright_{\text{mul}} \tilde{r}_i\) (where \(\triangleright\) denotes strict subterm ordering and \(\triangleright_{\text{mul}}\) denotes the multiset extension of \(\triangleright\))

A set \(\text{HOR}\) of higher-order rewrite rules satisfies the general schema (w.r.t. FOR) if each rule \(r \in \text{HOR}\) satisfies the general schema and there are not mutually recursive definitions.

Some of the conditions given in the definition of the general schema can be loosened. The condition \(\tilde{X} \subseteq \tilde{Y}\) could be removed by reasoning on a transformed version of \(F\), while mutually recursive definitions can be managed by introducing product types and packing them together in the same product. Although restricted, the general schema is interesting from a practical point of view: it allows the introduction of functional constants of higher-order types by primitive recursion on a first-order data structure. We refer to [20] for examples and applications of the general schema.

A restriction is also to be imposed on the first-order rewrite rules: FOR must be non-duplicating.

**Definition 2.10** A first-order rewrite rule \(r : t \rightarrow t'\) is non-duplicating\(^7\) if for any variable \(x\) the number of its occurrences in \(t\) is less than or equal to the number of its occurrences in \(t'\). A set of rewrite rules is non-duplicating if each of them is so.

The restriction to non-duplicating first-order rules is necessary to get strong normalization also if we consider algebraic terms only (otherwise we could easily code Toyama’s example of non-termination [26]). In [25] it was shown that strong normalization is a modular property of disjoint unions of non-duplicating first-order term rewriting systems. In practice, however, the restriction to non-duplicating rules is not a real constraint, since most implementations of rewrite systems use sharing, and shared-reductions are always non-duplicating.

We can now state our main result.

**Theorem 2.11 (Main Theorem)** Let \(R\) be a set of rewrite rules such that

1. FOR is non-duplicating and first-order algebraic terms\(^8\) are strongly normalizable w.r.t. \(\rightarrow_{\text{FOR}}\)

2. \(\text{HOR}\) satisfies the general schema (w.r.t. FOR).

\[\text{Note that this condition ensures that } F[\tilde{X}, \tilde{Z}]\tilde{Y} \text{ is rewritable.}\]

\[\text{Also called conservative in the literature.}\]

\[\text{By Lemma 2.3 we can avoid referring to a context.}\]
Then the systems of the \( \lambda R \)-cube are strongly normalizable w.r.t. \( \Rightarrow_{R_3} \).

The rest of the paper will be devoted to the proof of the main theorem. Since all the systems of \( \lambda R \)-cube are subsystems of \( \lambda_{RC} \) the proof of the main theorem will be given for \( \lambda_{RC} \).

3 Metatheory of the \( \lambda R \)-cube

In this section we will deal with the main syntactical properties of \( \lambda_{RC} \) that will be used in the proof of Theorem 2.11. The proofs of some of them are straightforward extensions of the corresponding proofs for the \( \lambda \)-cube, but other properties, like Subject Reduction, require the development of some technical machinery.

It is easy to show Subject Reduction for \( \rightarrow_r \):

**Proposition 3.1 (Subject Reduction Lemma for rewriting, SR\(_R\))**

For \( \Gamma \) a context, \( P, P' \) and \( D \) terms and \( r \in R \),

\[
\Gamma \vdash P : D \& P \rightarrow_r P' \Rightarrow \Gamma \vdash P' : D.
\]

It turns out that Subject Reduction for \( \beta \) is a much harder nut to crack. The standard proof is by induction on the derivation. Here we run into a problem when we consider the base case:

\[
\Gamma \vdash \lambda x : C. M : \Pi x : A, B \quad \Gamma \vdash N : A
\]

with \( (\lambda x : C. M) \downarrow N \beta \Rightarrow M[N/x] \), and we want to show that \( \Gamma \vdash M[N/x] : B[N/x] \). By Stripping we conclude that \( \Gamma \vdash \lambda x : C. M : \Pi x : C.D \) with \( \Pi x : C.D \Rightarrow_{R_3} \Pi x : A.B \), but we cannot conclude from this that \( C \Rightarrow_{R_3} A \) and \( D \Rightarrow_{R_3} B \), because we do not have the Church-Rosser property.

Following [15] and [16], we solved this problem by showing first the commutativity of weak-head-reduction and \( \beta \)-reduction, and then the commutativity of weak-head-reduction and \( R \)-reduction. We also proved that we have postponement of \( R \)-reduction with respect to weak-head-reduction on types and kinds. Using this properties we can prove the lemma:

**Lemma 3.2** If \( \Pi x : A.C \Rightarrow_{R_3} \Pi x : B.D \) and all the terms on the reduction-expansion-path from \( \Pi x : A.C \) to \( \Pi x : B.D \) are types or kinds, then \( \Pi x : A.C \Rightarrow_{R_3} \Pi x : B.D \) via a path that only uses \( \Pi \)-terms.

**Corollary 3.3** If \( \Pi x : A.C \Rightarrow_{R_3} \Pi x : B.D \), then \( A \Rightarrow_{R_3} B \) and \( C \Rightarrow_{R_3} D \).

**Proposition 3.4 (Subject Reduction for \( \beta \))** For \( \Gamma \) and \( \Gamma' \) contexts, \( P, P' \) and \( D \) terms,

\[
\Gamma \vdash P : D \& P \rightarrow_{\beta} P' \Rightarrow \Gamma \vdash P' : D \quad \Gamma \vdash P : D \& \Gamma \rightarrow_{\beta} \Gamma' \Rightarrow \Gamma' \vdash P : D.
\]

**Proof** By induction on the derivation one proves the statement for a one step reduction. The only interesting case is when the last rule is \( (app) \) and \( P \equiv (\lambda x : A.B)C, P' \equiv B[C/x] \). We then use the fact that \( \Pi x : A.C \Rightarrow_{R_3} \Pi x : B.D \) implies \( A \Rightarrow_{R_3} B \) and \( C \Rightarrow_{R_3} D \), which is what the previous Corollary states.

From Propositions 3.1 and 3.4 we obtain the Subject Reduction for \( \rightarrow_{R_3} \).

The following lemma will be used in the following section.

**Lemma 3.5 (Uniqueness of formation)** Let \( \Gamma \) and \( \Gamma' \) be contexts and \( B \) a term.

\( \Gamma \vdash B \) formed by \( (p_1, p_2) \) in \( \Gamma \) and \( \Gamma' \vdash B \) formed by \( (p'_1, p'_2) \) in \( \Gamma' \) implies \( p_1 \equiv p'_1, p_2 \equiv p'_2 \).

4 The proof of the Main Theorem

From now on, when dealing with a set \( R \) of rewriting rules, we shall implicitly assume conditions 1. and 2. of the Main Theorem (2.11) to be satisfied.

\( \chi \models SN \) will denote the fact that system \( \chi \) is strongly normalizable.

The proof of the Main Theorem consists in three main steps

- \( \lambda_{SR} \models SN \)
- \( \lambda_{SR} \models SN \Rightarrow \lambda_{Rw} \models SN \)
- \( \lambda_{Rw} \models SN \Rightarrow \lambda_{RC} \models SN \)

where system \( \lambda_{SR} \) is a type assignment system consisting in the extension of the intersection system of [5] [10] with a set \( R \) of rewriting rules.

For \( \lambda_{SR} \models SN \) we refer to [2], where a proof based on the Tait-Girard computability predicate method is given.

The proofs of the other two steps are based instead on a method that, together with that of Tait-Girard, is among the most used in proofs of strong normalization: the method of reduction-preserving translations. The implications are proved by providing a translation from the terms of the former system to the terms of the latter such that reductions are preserved, i.e., reducible terms are mapped to reducible terms. This
method has been used by Harper, Honsell and Plotkin [19] to obtain SN of their system LF (roughly corresponding to λF) using SN of simply typed lambda calculus (corresponding to λω).

The translation for proving λλR ⊢ SN ⇒ λλRw ⊢ SN is nothing but a type-erasing function. For its definition and the proof of reduction-preservation we refer to [3].

The translation and the reduction preservation proof for λλRw ⊢ SN ⇒ λλR ⊢ SN will be described below.

4.1 λλRw ⊢ SN ⇒ λλR ⊢ SN

The translation from terms in λRC to terms in λλRw is a simple generalization of that provided by Geuvers and Nederhof in [17] to prove strong normalization for λRC. Geuvers and Nederhof’s translation can be seen as a higher order version of the map defined by Harper, Honsell and Plotkin in [19].

As in [19] and [17], it is not possible here to define a reduction-preserving map [-] such that

Γ ⊢ M : A ⇒ [Γ] ⊢ [M] : [A]

i.e. [-] cannot work uniformly on all the terms of λRC. One is then forced to define another map τ(-) from kinds and constructors to types and to prove that


Since also τ cannot work uniformly on constructors and kinds, in its definition we shall use another map, ρ : {[]} ⊔ Kind(λRC) → Kind(λλRw) such that if M is a constructor of kind k in λRC then τ(M) is a constructor of kind ρ(k) in λλRw. ρ(k) is just the λRC-kind obtained by erasing from k all type dependencies.

Definition 4.1 The map ρ : {[]} ⊔ Kind(λRC) → Kind(λλRw) is inductively defined by:

1. ρ(□) = ρ(□) = □
2. ρ(Πα : M, N) = ρ(M) ⇒ ρ(N) if Πα : M, N is formed by (□, □)
3. ρ(Πx : M, N) = ρ(N) if Πx : M, N is formed by (∗, □).

The definition by cases is correct by Lemma 3.5.

Now, we choose one of the variables of VarB to act as a fixed constant, i.e. it will not be used as a bound variable in an abstraction. This variable will be denoted by 0.

Definition 4.2 The map τ : {[]} ⊔ Kind(λRC) ⊔ Constr(λRC) → Term(λλRw) is inductively defined by:

1. τ(□) = τ(□) = 0 : *
2. τ(α) = α if α is a variable.
3. τ(Πα : M, N) = Πα : ρ(M), τ(N) if Πα : M, N is formed by (□, □) or (□, ∗).
4. τ(Πx : M, N) = Πx : τ(M), τ(N) if Πx : M, N is formed by (∗, □) or (∗, ∗).
5. τ(MN) = τ(M), τ(N) if MN is formed by (□, □).

The definition by cases is correct by Lemma 3.5.

In order to map Context(λRC) into Context(λλRw) we choose for each variable α ∈ VarB a connected variable xα ∈ Var*, such that no two variables of VarB are connected to the same variable of Var*. We extend now the map τ in such a way that it acts also on Context(λRC) yielding elements of Context(λλRw):

1. Let A ∈ Kind(λRC) ⊔ Type(λRC).
   τ(x : A) = x : τ(A) if x ∈ Var*.
   τ(α : A) = α : ρ(A), xα : τ(A) if α ∈ VarB.
   τ(Γ) = ⟨0 : ∗, d : ∗, τ(u1 : A1), τ(u2 : A2), ..., τ(un : An)⟩.

The reason for putting 0 : ∗ and d : ∗ ∈ Πα : xα in the context is that in the following definition of the map [-] on terms of λRC it will be necessary to have a canonical inhabitant for each type and kind. If τ(Γ) ⊢ λλRw B : ∗ or τ(Γ) ⊢ λλRw B : □, we want τ(Γ) ⊢ λλRw cB : B for a cB which does not depend on the structure of Γ.

Now, if τ(Γ) ⊢ λλRw B : ∗ we shall put cB ≡ dB and if τ(Γ) ⊢ λλRw B : □, a canonical inhabitant of B is inductively defined by

1. If B ≡ ∗ then c∗ ≡ 0
2. If B ≡ k1 → k2 then ck1 → k2 = λα : k1, cαk2.
Theorem 4.3 The map \([\ ]\) : Kind(\(\lambda_{RC}\)) \cup Constr(\(\lambda_{RC}\)) \cup Object(\(\lambda_{RC}\)) \rightarrow Object(\(\lambda_{Rd}\)) is inductively defined by

1. \([\ast]\) = \(c^0\)
2. \([x]\) = \(x\) if \(x \in Var^*\)
3. \([\alpha]\) = \(\boxed{a^0}\) if \(\alpha \in Var^\mathbb{N}\)
4. \([s]\) = \(c^0\) if \(s \in S\)
5. \([f] = f\) if \(f \in \mathcal{F}\)
6. \([\Pi x : M. N]\) = \(c^{0 \rightarrow 0 \rightarrow 0}[M\{N\}][\boxed{c^{(\ast)}}/x]\) if \(\Pi x : M. N\) is formed by \((\ast, \ast)\) or \((\ast, \square)\)
7. \([\lambda x : M. N]\) = \((\lambda z : 0. \lambda x : \tau(M).[N]\{M\})\) where \(z\) is a fresh variable, if \(\lambda x : M. N\) is formed by \((\ast, \ast)\) or \((\ast, \square)\)
8. \([M\ N]\) = \([M]\{N\}\) if \(M \ N\) is formed by \((\ast, \ast)\) or \((\ast, \square)\)
9. \([M\ N]\) = \([M]\{\tau(N)\}\{N\}\) if \(M \ N\) is formed by \((\square, \ast)\) or \((\square, \square)\).

This definition by cases is correct by Lemma 3.5. The following theorems state that \([\ ]\) satisfies the required conditions: Theorem 4.4 says that the range of \([\ ]\) is really Object(\(\lambda_{Rd}\)), and Theorem 4.5 that the mapping preserves all possible reduction sequences.

Theorem 4.4 Let \(\Gamma \in Context(\lambda_{RC})\), \(M, N \in Term(\lambda_{RC})\).
If \(\Gamma \vdash_{\lambda_{RC}} M : N\) then \(\tau(\Gamma) \vdash_{\lambda_{Rd}} M : \tau(N)\).

Theorem 4.5 Let \(M, M' \in Term(\lambda_{RC})\). If \(M \rightarrow_{Rd} M'\) then \([M] \rightarrow_{Rd} [M']\).

Using the previous theorems we can now easily prove the main result of this section.

Theorem 4.6 \(\rightarrow_{Rd}\)-strong normalization of \(\lambda_{Rd}\) implies \(\rightarrow_{Rd}\)-strong normalization of \(\lambda_{RC}\).

5 Modularity of Confluence

For strongly normalizing relations, local confluence is equivalent to confluence (Newman's Lemma [22]). So, to prove confluence for \(\rightarrow_{Rd}\) in \(\lambda_{RC}\), it is enough to prove local confluence. The following lemmas show that the confluence of FOR for algebraic terms transfers to \(\lambda_{RC}\)-terms, and that for the class of higher order rewrite systems which we consider, the absence of critical pairs implies confluence (note that this is not true for arbitrary higher-order rewrite systems, as shown in [23]).

Lemma 5.1 If FOR is confluent on the set of algebraic terms of \(\lambda_{RC}\) then \(\rightarrow_{FOR}\) is locally confluent on \(\lambda_{RC}\).

Lemma 5.2 Let HOR be a higher order rewrite system satisfying the general schema. If there is no critical pair, then \(\rightarrow_{HOR}\) is confluent on \(\lambda_{RC}\).

Now using the previous lemmas we can prove that \(\rightarrow_{Rd}\) is locally confluent.

Theorem 5.3 (Local Confluence of \(\rightarrow_{Rd}\) in \(\lambda_{RC}\))
If FOR is confluent over the set of algebraic terms, and HOR does not introduce critical pairs then \(\rightarrow_{Rd}\) is locally confluent in \(\lambda_{RC}\).

For example, the class of higher order rewriting systems defining higher order functions by primitive recursion (structured recursion) on first order data structures, verify the required hypothesis and then \(\rightarrow_{Rd}\) is confluent in this case.

6 Conclusions

We have extended the Calculus of Constructions with algebraic types and rewrite rules. Our system is closely related to the Calculus of Constructions with inductive types (CCI) defined by Th. Coquand and C. Paulin-Mohring [13], since CCI can be seen as an extension of the Calculus of Constructions with a particular class of higher order rewrite rules. The strong normalization of CCI was recently proved by B. Werner [27]. The problem of extending the CCI with pattern-matching definitions was studied by Th. Coquand in [11]. In particular, in [11] there is a notion of recursive schema ensuring strong normalization, and rewrite rules are assumed critical-pair free. In our framework these restrictions apply only to
higher order rules (first order rules are simply required to be non-duplicating).

Confluence and strong normalization are essential properties of logical systems, since they ensure the consistence of the system. Proving these properties is in general a difficult task, so it is important to study under which conditions these proofs are modular. Our results show that in order to prove strong normalization of any of the systems in the $\lambda R$-cube it is sufficient to prove termination of the first order rewrite rules in $R$ on algebraic terms, provided that $R$ satisfies certain syntactical conditions, namely non-duplication for FOR and the general schema for HOF. As a consequence, we get the strong normalization of a restriction of CCI (with pattern-matching) where the inductive types are defined by structural induction. The restriction on first order rules is not important in practice, since most implementations of rewriting use sharing, and shared reductions are always conservative. The general schema, however, limits the power of the higher order rules. The generalization of the proof of strong normalization to wider classes of higher order rules will be the subject of future work.

Acknowledgements

We wish to thank Jean-Pierre Jouannaud and Mariangiola Dezani for their scientific support. The first author is also grateful to Margherita Lombardi for her constant encouragement.

References


