[17] H. Geuvers and M.J. Nederhof. A simple modular proof of the strong normalization for the calculus of constructions. *J. of Functional Programming*, vol.1, 1991.

[18] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* Thèse d'Etat, Univ. Paris VII, France, 1972.

[19] R. Harper, F. Honsell and G. Plotkin. A framework for defining logics. *Proceedings second symposium on Logic In Computer Science.*, IEEE, Washington DC, 1987.

[20] J.-P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam*, 1991.

[21] J. W. Klop. Term rewriting systems: a tutorial. *EATCS Bulletin*, 32:143–182, June 1987.

[22] M. H. A. Newman. On theories with a combinatorial definition of 'equivalence'. *Ann. Math.*, 43(2):223–243, 1942.

[23] T. Nipkow. Higher order critical pairs. *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.

[24] Mitsuhiro Okada. Strong normalizability for the combined system of the types lambda calculus and an arbitrary convergent term rewrite system. In *Proc. ISSAC 89, Portland, Oregon*, 1989.

[25] M. Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26:65–70, 1987.

[26] Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, April 1987.

[27] B. Werner. *Méta-théorie du Calcul des Constructions Inductives.* Thèse Univ. Paris VII, France.

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 λR-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 HOR. 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

[1] F. Barbanera. Adding algebraic rewriting to the calculus of constructions: Strong normalization preserved. In *Proc. of the 2nd Int. Workshop on Conditional and Typed Rewriting*, 1990.

[2] F. Barbanera and M. Fernández. Combining first and higher order rewrite systems with type assignment systems. *Proceedings of the international conference on Typed Lambda Calculi and Applications*, Utrecht, LNCS 664, Springer Verlag, 1993.

[3] F. Barbanera and M. Fernández. Modularity of termination and confluence in combinations of rewrite systems with λ_ω. *Proceedings of the 20th International Colloquium on Automata, Languages, and Programming*, Lund, A.Lingas, R.Karlsson and S.Carlsson eds., LNCS 700, Springer Verlag, 1993.

[4] H. Barendregt. Introduction to generalised type systems. *Journal of Functional Programming*, 1991.

[5] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter λ-model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[6] S. Berardi. *Type dependence and constructive mathematics.* Ph.D. Thesis, Mathematical Institute, University of Torino, 1990.

[7] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.

[8] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization. *Theoretical Computer Science*, 1990.

[9] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. To appear, 1993.

[10] M. Coppo, and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. of Formal Logic*, 21(4), 1980.

[11] Th. Coquand. Pattern matching with dependent types. In *Proceedings of the Workshop on Logical Frameworks*, 1992.

[12] Th. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76:95–120, February 1988.

[13] Th. Coquand and C. Paulin-Mohring. Inductively defined types. In *Proceedings of Colog'88, LNCS 417*, Springer-Verlag, 1990.

[14] D. J. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. In *Proc. 4th Rewriting Techniques and Applications, Como, LNCS 488*, 1991.

[15] H. Geuvers. The Church-Rosser property for $\beta\eta$-reduction in typed λ-calculi. *Proceedings seventh symposium on Logic In Computer Science.*, IEEE, Santa Cruz, 1992.

[16] H. Geuvers. *Logics and type systems.* Ph.D. Thesis, dept. of Computer Science, university of Nijmegen, 1993.

Definition 4.3 *The map* $[\ ] : \mathsf{Kind}(\lambda_{RC}) \cup \mathsf{Constr}(\lambda_{RC}) \cup \mathsf{Object}(\lambda_{RC}) \to \mathsf{Object}(\lambda_{R\omega})$ *is inductively defined by*

1. $[\star] = c^0$
2. $[x] = x \quad$ *if* $x \in Var^\star$
3. $[\alpha] = x^\alpha \quad$ *if* $\alpha \in Var^\Box$
4. $[s] = c^0 \quad$ *if* $s \in S$
5. $[f] = f \quad$ *if* $f \in \mathcal{F}$
6. $[\Pi x\!:\!M.N] = c^{0\to 0\to 0}[M][N][c^{\tau(M)}/x]$ *if* $\Pi x\!:\!M.N$ *is formed by* $(\star,\star)$ *or* $(\star,\Box)$

   $[\Pi\alpha\!:\!M.N] = c^{0\to 0\to 0}[M][N][c^{\tau(M)}/x], [c^{\rho(M)}/\alpha]$ *if* $\Pi\alpha\!:\!M.N$ *is formed by* $(\Box,\star)$ *or* $(\Box,\Box)$
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* $(\star,\star)$ *or* $(\star,\Box)$

   $[\lambda\alpha\!:\!M.N] = (\lambda z\!:\!0.\lambda\alpha\!:\!\rho(M).\lambda x^\alpha\!:\!\tau(M).[N])[M]$ *where* z *is a fresh variable, if* $\lambda\alpha\!:\!M.N$ *is formed by* $(\Box,\star)$ *or* $(\Box,\Box)$
8. $[MN] = [M][N] \quad$ *if* MN *is formed by* $(\star,\star)$ *or* $(\star,\Box)$

   $[MN] = [M]\tau(N)[N] \quad$ *if* MN *is formed by* $(\Box,\star)$ *or* $(\Box,\Box)$.

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 $\mathsf{Object}(\lambda_{R\omega})$, and Theorem 4.5 that the mapping preserves all possible reduction sequences.

Theorem 4.4 *Let* $\Gamma \in \mathsf{Context}(\lambda_{RC})$, $M, N \in \mathsf{Term}(\lambda_{RC})$.
If $\Gamma \vdash_{\lambda_{RC}} M : N$ *then* $\tau(\Gamma) \vdash_{\lambda_{R\omega}} [M] : \tau(N)$.

Theorem 4.5 *Let* $M, M' \in \mathsf{Term}(\lambda_{RC})$. *If* $M \to_{R\beta} M'$ *then* $[M] \to_{R\beta} [M']$.

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

Theorem 4.6 $\to_{R\beta}$*-strong normalization of* $\lambda_{R\omega}$ *implies* $\to_{R\beta}$*-strong normalization of* λ_{RC} .

5 Modularity of Confluence

For strongly normalizing relations, local confluence is equivalent to confluence (Newman's Lemma [22]). So, to prove confluence for $\to_{R\beta}$ in λ_{RC}, it is enough to prove local confluence. The following lemmas show that the confluence of FOR for algebraic terms transfers to λ_{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* λ_{RC} *then* $\to_{FOR}$ *is locally confluent on* λ_{RC}.

Lemma 5.2 *Let* HOR *be a higher order rewrite system satisfying the general schema. If there is no critical pair, then* $\to_{HOR}$ *is confluent on* λ_{RC}.

Now using the previous lemmas we can prove that $\to_{R\beta}$ is locally confluent.

Theorem 5.3 (Local Confluence of $\to_{R\beta}$ in λ_{RC}) *If* FOR *is confluent over the set of algebraic terms, and* HOR *does not introduce critical pairs then* $\to_{R\beta}$ *is locally confluent in* λ_{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 $\to_{R\beta}$ 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

method has been used by Harper, Honsell and Plotkin [19] to obtain SN of their system LF (roughly corresponding to λ_P) using SN of simply typed lambda calculus (corresponding to $\lambda_\rightarrow$).

The translation for proving $\lambda_{\wedge R} \models \mathsf{SN} \Rightarrow \lambda_{R\omega} \models \mathsf{SN}$ 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 $\lambda_{R\omega} \models \mathsf{SN} \Rightarrow \lambda_{RC} \models \mathsf{SN}$ will be described below.

4.1 $\lambda_{R\omega} \models \mathsf{SN} \Rightarrow \lambda_{RC} \models \mathsf{SN}$

The translation from terms in λ_{RC} to terms in $\lambda_{R\omega}$ is a simple generalization of that provided by Geuvers and Nederhof in [17] to prove strong normalization for λ_C . 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

$$\Gamma \vdash_{\lambda_{RC}} M{:}A \Rightarrow [\Gamma] \vdash_{\lambda_{R\omega}} [M]{:}[A]$$

i.e. $[-]$ cannot work uniformly on all the terms of λ_{RC}. One is then forced to define another map $\tau(-)$ from kinds and constructors to types and to prove that

$$\Gamma \vdash_{\lambda_{RC}} M{:}A \Rightarrow \tau(\Gamma) \vdash_{\lambda_{R\omega}} [M]{:}\tau(A).$$

Since also τ cannot work uniformly on constructors and kinds, in its definition we shall use another map, $\rho : \{\Box\} \cup \mathsf{Kind}(\lambda_{RC}) \rightarrow \mathsf{Kind}(\lambda_{R\omega})$ such that if M is a constructor of kind k in λ_{RC} then $\tau(M)$ is a constructor of kind $\rho(k)$ in $\lambda_{R\omega}$. $\rho(k)$ is just the $\lambda_{R\omega}$ -kind obtained by erasing from k all type dependencies.

Definition 4.1 *The map* $\rho : \{\Box\} \cup \mathsf{Kind}(\lambda_{RC}) \rightarrow \mathsf{Kind}(\lambda_{R\omega})$ *is inductively defined by:*

1. $\rho(\star) = \rho(\Box) = \star$
2. $\rho(\Pi\alpha{:}M.N) = \rho(M) \rightarrow \rho(N)$ *if* $\Pi\alpha{:}M.N$ *is formed by* $(\Box, \Box)$
3. $\rho(\Pi x{:}M.N) = \rho(N)$ *if* $\Pi x{:}M.N$ *is formed by* $(\star, \Box)$.

The definition by cases is correct by Lemma 3.5.

Now, we choose one of the variables of $Var^\Box$ 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* $\tau : \{\Box\} \cup \mathsf{Kind}(\lambda_{RC}) \cup \mathsf{Constr}(\lambda_{RC}) \rightarrow \mathsf{Term}(\lambda_{R\omega})$ *is inductively defined by:*

1. $\tau(\star) = \tau(\Box) = 0 : \star$
2. $\tau(\alpha) = \alpha$ *if* α *is a variable.*
   $\tau(s) = s$ *if* $s \in \mathcal{S}$.
3. $\tau(\Pi\alpha{:}M.N) = \Pi\alpha{:}\rho(M).\tau(M) \rightarrow \tau(N) : \star$ *if* $\Pi\alpha{:}M.N$ *is formed by* $(\Box, \Box)$ *or* $(\Box, \star)$.
   $\tau(\Pi x{:}M.N) = \Pi x{:}\tau(M).\tau(N)$ *if* $\Pi x{:}M.N$ *is formed by* $(\star, \Box)$ *or* $(\star, \star)$.
4. $\tau(\lambda\alpha{:}M.N) = \lambda\alpha{:}\rho(M).\tau(N)$ *if* $\lambda\alpha{:}M.N$ *is formed by* $(\Box, \Box)$.
   $\tau(\lambda x{:}M.N) = \tau(N)$ *if* $\lambda x{:}M.N$ *is formed by* $(\star, \Box)$.
5. $\tau(MN) = \tau(M)\tau(N)$ *if* MN *is formed by* $(\Box, \Box)$.
   $\tau(MN) = \tau(M)$ *if* MN *is formed by* $(\star, \Box)$.

The definition by cases is correct by Lemma 3.5.

In order to map $\mathsf{Context}(\lambda_{RC})$ into $\mathsf{Context}(\lambda_{R\omega})$ we choose for each variable $\alpha \in Var^\Box$ a connected variable $x^\alpha \in Var^\star$, such that no two variables of $Var^\Box$ are connected to the same variable of $Var^\star$. We extend now the map τ in such a way that it acts also on $\mathsf{Context}(\lambda_{RC})$ yielding elements of $\mathsf{Context}(\lambda_{R\omega})$:

1. Let $A \in \mathsf{Kind}(\lambda_{RC}) \cup \mathsf{Type}(\lambda_{RC})$.
   $\tau(x : A) = x : \tau(A)$ if $x \in Var^\star$.
   $\tau(\alpha : A) = \alpha : \rho(A), x^\alpha : \tau(A)$ if $\alpha \in Var^\Box$.
2. Let $\Gamma = \langle u_1 : A_1, u_2 : A_2, \ldots, u_n : A_n\rangle \in \mathsf{Context}(\lambda_{RC})$.
   $\tau(\Gamma) = \langle 0 : \star, d : \bot, \tau(u_1 : A_1), \tau(u_2 : A_2), \ldots, \tau(u_n : A_n)\rangle$.

The reason for putting $0 : \star$ and $d : \bot \equiv \Pi\alpha{:}\star.\alpha$ in the context is that in the following definition of the map $[-]$ on terms of λ_{RC} it will be necessary to have a canonical inhabitant for each type and kind. If $\tau(\Gamma) \vdash_{\lambda_{R\omega}} B : \star$ or $\tau(\Gamma) \vdash_{\lambda_{R\omega}} B : \Box$, we want $\tau(\Gamma) \vdash_{\lambda_{R\omega}} c^B : B$ for a c^B which does not depend on the structure of Γ.

Now, if $\tau(\Gamma) \vdash_{\lambda_{R\omega}} B : \star$ we shall put $c^B \equiv dB$ and if $\tau(\Gamma) \vdash_{\lambda_{R\omega}} B : \Box$, a canonical inhabitant of B is inductively defined by

1. If $B \equiv \star$ then $c^\star = 0$
2. If $B \equiv k_1 \rightarrow k_2$ then $c^{k_1 \rightarrow k_2} = \lambda\alpha : k_1.c^{k_2}$.

Then the systems of the λR-cube are strongly normalizable w.r.t. $\rightarrow_{R\beta}$.

The rest of the paper will be devoted to the proof of the main theorem. Since all the systems of λR-cube are subsystems of λ_{RC} the proof of the main theorem will be given for λ_{RC}.

3 Metatheory of the λR-cube

In this section we will deal with the main syntactical properties of λ_{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 λ-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, $\mathbf{SR}_R$)
For Γ 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 β 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:

$$\frac{\Gamma \vdash \lambda x{:}C.M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x{:}C.M)N : B[N/x]}$$

with $(\lambda x{:}C.M)N \rightarrow_\beta M[N/x]$ 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 $\Pi x{:}C.D =_{R\beta} \Pi x{:}A.B$, but we cannot conclude from this that $C =_{R\beta} A$ and $D =_{R\beta} B$, 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 β-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 $\Pi x{:}A.C =_{R\beta} \Pi x{:}B.D$ and all the terms on the reduction-expansion-path from $\Pi x{:}A.C$ to $\Pi x{:}B.D$ are types or kinds, then $\Pi x{:}A.C =_{R\beta} \Pi x{:}B.D$ via a path that only uses Π-terms.*

Corollary 3.3 *If $\Pi x{:}A.C =_{R\beta} \Pi x{:}B.D$, then $A =_{R\beta} B$ and $C =_{R\beta} D$.*

Proposition 3.4 (Subject Reduction for β) *For Γ and Γ' contexts, P, P' and D terms,*

$$\Gamma \vdash P : D \ \& \ P \twoheadrightarrow_\beta P' \ \Rightarrow \ \Gamma \vdash P' : D$$
$$\Gamma \vdash P : D \ \& \ \Gamma \twoheadrightarrow_\beta \Gamma' \ \Rightarrow \ \Gamma' \vdash P : D.$$

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 $\Pi x{:}A.C =_{R\beta} \Pi x{:}B.D$ implies $A =_{R\beta} B$ and $C =_{R\beta} D$, which is what the previous Corollary states. □

From Propositions 3.1 and 3.4 we obtain the Subject Reduction for $\rightarrow_{R\beta}$.

The following lemma will be used in the following section.

Lemma 3.5 (Uniqueness of formation) *Let Γ and Γ' be contexts and B a term.*
B formed by (p_1, p_2) in Γ & B formed by (p_1', p_2') in Γ' 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 implicitely assume conditions 1. and 2. of the Main Theorem (2.11) to be satisfied. $\chi \models SN$ will denote the fact that system χ is strongly normalizable.

The proof of the Main Theorem consists in three main steps

- $\lambda_{\wedge R} \models \mathsf{SN}$
- $\lambda_{\wedge R} \models \mathsf{SN} \Rightarrow \lambda_{R\omega} \models \mathsf{SN}$
- $\lambda_{R\omega} \models \mathsf{SN} \Rightarrow \lambda_{RC} \models \mathsf{SN}$

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

For $\lambda_{\wedge R} \models \mathsf{SN}$ 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

λ-cube. We will denote by $\lambda_{R\_}$ a generic system of the λR-cube.

Some systems of the λR-cube are already present in the literature. In particular, when HOR is empty, i.e.we have only first-order rewriting, $\lambda_{R\rightarrow}$ is the system studied in [7] and [24], while λ_{R2} is equivalent to the system defined by Breazu-Tannen and Gallier in [8]. The systems of [20] correspond to $\lambda_{R\rightarrow}$ and λ_{R2}. We have already mentioned in the introduction which results were proved for these systems.

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

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

$$\forall x \in FV(t).[x : B \in \Gamma \Rightarrow B =_{R\beta} \tau \in \mathsf{T}_{\mathcal{S}}].$$

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 λR-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*

$$F\vec{l}[\vec{X},\vec{x}]\vec{Y} \rightarrow v[(Fr_1^{\rightarrow}[\vec{X},\vec{x}]\vec{Y}),\ldots,(Fr_m^{\rightarrow}[\vec{X},\vec{x}]\vec{Y}),\vec{X},\vec{x},\vec{Y}]$$

where $\vec{X}$ and $\vec{Y}$ are sequences of higher-order variables and $\vec{x}$ is a sequence of first-order variables, and such that

1. $\vec{X} \subseteq \vec{Y}$[6]
2. *F is function symbol that can appear neither in $\vec{l}, r_1^{\rightarrow}, \ldots, r_m^{\rightarrow}$, nor in the rules of FOR, and its occurrences in v are only the ones explicitly indicated*
3. *$\vec{l}, r_1^{\rightarrow}, \ldots, r_m^{\rightarrow}$ are terms of sort type*
4. *$\forall i \in [1..m]$, $\vec{l} \rhd_{mul} r_i^{\rightarrow}$ (where $\lhd$ denotes strict subterm ordering and $\rhd_{mul}$ denotes the multiset extension of $\rhd$)*

A set HOR of higher-order rewrite rules satisfies the general schema *(w.r.t. FOR) if each rule $r \in HOR$ 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 $\vec{X} \subseteq \vec{Y}$ 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. $\rightarrow_{FOR}$*
2. *HOR satisfies the general schema (w.r.t. FOR).*

[6] Note that this condition ensures that $F\vec{l}[\vec{X},\vec{x}]\vec{Y}$ is rewritable.

[7] Also called *conservative* in the literature.

[8] By Lemma 2.3 we can avoid referring to a context.

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 t is* rewritable *if it is algebraic in some context Γ and for any $x \in FV(t)$ 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 $\langle t, t' \rangle$, such that t, t' are algebraic terms (for some contexts), t is rewritable*[3]*, $FV(t') \subseteq FV(t)$, and, for any context Γ 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 λ-abstractions in the right-hand side. We have then* λ-higher-order rewrite rules[4]*.*

Given a set R of rewriting rules, we denote by FOR and 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 $C[\,]$ and a substitution φ such that $M \equiv C[t\varphi]$ and $N \equiv C[t'\varphi]$. $\twoheadrightarrow_r$ denotes the reflexive and transitive clusure 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

$$M \to_{R\beta} N \;\Leftrightarrow\; M \to_R N \vee M \to_\beta N.$$

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 λ-cube specified by $\langle \mathcal{S}, \mathcal{F}, R \rangle$, it suffices, besides the additional axioms for sorts and function symbols given before, to replace the rule $(conv)$ of the pure λ-cube by the following one

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : p \qquad B =_{R\beta} B'}{\Gamma \vdash A : B'}$$

where $=_{R\beta}$ is the least congruence containing $\to_{R\beta}$.

This, however, would not work. In the pure cube if we have $A =_\beta B$, then the Church-Rosser property of $=_\beta$, together with the property of subject-reduction, ensures that A and B are always equal via β-reductions and β-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 $=_{R\beta}$ Therefore we cannot consider $=_{R\beta}$ in the $(conv)$ rule. We have instead to consider the $R\beta$-reduction relation.

The complete definition of algebraic extension of the λ-cube runs now as follows.

Definition 2.7 (The λR-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 $\lambda\langle \mathcal{S}, \mathcal{F}, R \rangle$-*cube *($\lambda R$-cube for short) is defined by adding the following axioms to the axioms of the λ-cube*

$$(alg1) \quad \vdash s : \star \qquad \text{for any } s \in \mathcal{S}$$

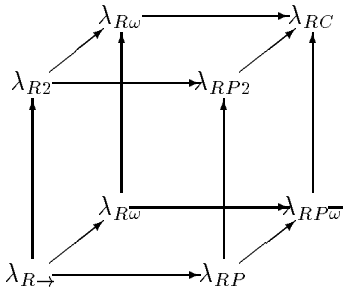
$$(alg2) \quad \vdash f : \sigma \qquad \text{for any } f \in \mathcal{F}_\sigma$$

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

$$(red_{R\beta}) \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B} \; A \to_{R\beta} B \text{ or } B \to_{R\beta} A$$

The eight systems of the λR-cube will be called $\lambda_{R\to}$, λ_{R2}, λ_{RP}, $\lambda_{R\overline{\omega}}$, λ_{RP2}, $\lambda_{R\omega}$, $\lambda_{RP\overline{\omega}}$ and $\lambda_{RP\omega}$ (or λ_{RC}).

Then, graphically the λR-cube turns out to be as follows



All the definitions which are not affected by the introduction of the algebraic features, like that of kind, object and so on, remain the same as for the

[3] This condition subsumes the usual condition "t not a variable" for rewrite rules.

[4] The notion of λ-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.

2 Adding Algebraic Rewriting to the λ-cube

The λ-cube is a coherent collection of eight type systems. Each system (generically denoted by $\lambda\_$) 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 λ-cube in order to have also algebraic features. We begin by considering a denumerable set $\mathcal{S}$ of *sorts*: $\mathcal{S} = \{s_1, s_2, \ldots\}$. The elements of $\mathcal{S}$ denote *algebraic base types*. So, first of all, we add to the rules of the cube the following axiom: $(alg1) \quad \vdash s_i : \star$, for each $s_i \in \mathcal{S}$. We define now, by induction, a set of algebraic types.

Definition 2.1 (Algebraic types) *The set* $\mathsf{T}_{\mathcal{S}}$ *of* algebraic types *on* $\mathcal{S}$ *is inductively defined as follows:*

- *If* $s \in \mathcal{S}$ *then* $s \in \mathsf{T}_{\mathcal{S}}$
- *If* $\sigma, \tau \in \mathsf{T}_{\mathcal{S}}$ *then* $\Pi x{:}\sigma.\tau \in \mathsf{T}_{\mathcal{S}}$

We will call *first-order algebraic types* the elements $\sigma_1 \to \ldots \to \sigma_n \to \sigma \in \mathsf{T}_{\mathcal{S}}$ such that $\sigma, \sigma_i \in \mathcal{S}$ $(1 \leq i \leq n)$. A context $\Gamma = \langle x_1{:}A_1, \ldots, x_n{:}A_n \rangle$ is called algebraic if $A_i \in \mathsf{T}_{\mathcal{S}}$ $(1 \leq i \leq n)$.

The next step is to consider, for each algebraic type, a set of function symbols of that type, i.e. a *signature* $\mathcal{F} = \bigcup_{\tau \in \mathsf{T}_{\mathcal{S}}} \mathcal{F}_\tau$, where $\mathcal{F}_\tau$ denotes the set of function symbols of type τ. We assume $\mathcal{F}_\tau \cap \mathcal{F}_{\tau'} = \emptyset$ if $\tau \not\equiv \tau'$. 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: $(alg2) \quad \vdash f : \sigma$. A function symbol f^m is called *first-order* if it has a first-order algebraic type $s_1 \to, \ldots, s_n \to s$ 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

$$T ::= \mathbf{x} \mid \mathbf{f} \mid \mathbf{s} \mid \star \mid \Box \mid TT \mid \lambda \mathbf{x}{:}T.T \mid \Pi \mathbf{x}{:}T.T$$

where $\mathbf{f}$ and $\mathbf{s}$ range over $\mathcal{F}$ and $\mathcal{S}$, respectively. Intuitively, $\star$ and $\Box$ denote respectively the set of types and kinds. The set $\{\star, \Box\}$ 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 $fP_1 \ldots P_m$ with $m \equiv n$.

We have now to define the notion of algebraic term, i.e. the natural translation of the notion of term of term rewriting systems. This notion has to be defined in the setting of the λ-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 Γ *(in a system* $\lambda\_$*), for* $\Gamma \in \mathsf{Context}(\lambda\_)$*, is an algebraic pseudoterm* t *such that* $\Gamma \vdash_{\lambda\_} t : A$*, any* $f \in \mathcal{F}$ *is saturated in* t*, and*

$$\forall x \in FV(t).[x : B \in \Gamma \;\Rightarrow\; \exists \sigma \in \mathsf{T}_{\mathcal{S}}.B =_\beta \sigma].$$

(iii) A first-order algebraic term t *(in* Γ*) is an algebraic term (in* Γ*) 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 σ 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* $\lambda\_$*, if* t *is a first-order algebraic term in* Γ *(and* t *is not a variable) then for any* Γ' *such that* $\Gamma' \vdash t : A$*,* t *is algebraic in* Γ'*.*

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 Γ turns out to have no sense, so we will impose that $r : t \to t'$ is a rewrite rule only if, for any context Γ, t is algebraic in Γ whenever it is typable in it. This

[1] The motivation for the introduction of arities is that a symbol f with type $s_1 \to \ldots \to s_n \to s$ could be otherwise considered, at the same time, first- and higher-order. Arities ensure that such ambiguities cannot arise.

[2] Of course one could define the reduction relation on terms of the algebraic extension of the λ-cube simply by stratifying its definition, starting from the λ-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.

on the languages the interactions between these computational models raise several problems, as shown in [21] and [14]. For typed languages (typed versions of λ-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 λ-calculus is again terminating with respect to β-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 λ-calculus, under certain conditions on the form of the rewrite rules.

The question that naturally arises is whether such a nice interaction between typed λ-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 β-conversion ($=_\beta$) 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 β-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 of all the systems of the so-called λ-cube [6], [4]. This extension will be called *algebraic-λ-cube* (λR-cube for short).

The main result we prove for the systems of the λR-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 $\lambda_{R\omega}$ (the extended polymorphic typed λ-calculus of order ω). 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 $\lambda_{\wedge R}$ (a type assignment system for λ-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 λR-cube.

We assume the reader familiar with the basic notions and notations of Pure Type Systems and the λ-cube as presented in [6], [16], [4]. The algebraic extension of the λ-cube will be discussed in Section 2. Section 3 will be devoted to the metatheory of the λR-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.

Modularity of Strong Normalization and Confluence in the algebraic-λ-cube

Franco Barbanera

Dipartimento di Informatica
Universita' di Torino
Corso Svizzera 185, 10149 Torino
Italy
barba@di.unito.it

Maribel Fernández

CNRS
Université de Paris-Sud
91405 Orsay Cedex
France
maribel@lri.fr

Herman Geuvers

Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen
The Netherlands
herman@cs.kun.nl

Abstract

In this paper we present the algebraic-λ-cube, an extension of Barendregt's λ-cube with first- and higher-order algebraic rewriting. We show that strong normalization is a modular property of all systems in the algebraic-λ-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-λ-cube.

We also prove that local confluence is a modular property of all the systems in the algebraic-λ-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 β-reduction on λ-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 λ-calculi and algebraic rewriting has been carried out both in untyped and typed contexts. If no type discipline is imposed

*This research was partially supported by ESPRIT Basic Research Action "TYPES".