ELSEVIER

Theoretical Computer Science 169 (1996) 201–220

Theoretical Computer Science

Combining algebraic rewriting, extensional lambda calculi, and fixpoints

Roberto Di Cosmo [a,*], Delia Kesner [b]

[a] *DMI-LIENS (CNRS URA 1347) Ecole Normale Supérieure - 45, Rue d'Ulm - 75230 Paris France*

[b] *CNRS and LRI - Bât 490, Université de Paris-Sud - 91405 Orsay Cedex, France*

Abstract

It is well known that confluence and strong normalization are preserved when combining *algebraic rewriting systems* with the simply typed lambda calculus. It is equally well known that confluence fails when adding either the usual contraction rule for η, or recursion together with the usual contraction rule for surjective pairing.

We show that *confluence* and *strong normalization* are *modular* properties for the combination of algebraic rewriting systems with typed lambda calculi enriched with *expansive* extensional rules for η and surjective pairing. We also show how to preserve confluence in a modular way when adding *fixpoints* to different rewriting systems. This result is also obtained by a simple translation technique allowing to simulate bounded recursion.

1. Introduction

Confluence and strong normalization for the combination of lambda calculus and algebraic rewriting systems have been the object of many studies [3, 15, 5, 12], where the *modularity* of these properties is studied. However, the simple extensional equality:

$$(\eta) \qquad \lambda x.Mx = M \quad \text{if } M : A \to B \quad \text{and} \quad x \notin FV(M)$$

when turned into a *contractive* rewrite rule

$$(\eta_c) \qquad \lambda x.Mx \to M \quad \text{if } M : A \to B \quad \text{and} \quad x \notin FV(M)$$

breaks the modularity of confluence, as the following example shows:

Example 1.1. Take a single algebraic rule $fx \xrightarrow{alg} a$ where f and a are algebraic symbols. Then the following diagram cannot be closed:

$$f \xleftarrow{\eta_c} \lambda x.fx \xrightarrow{alg} \lambda x.a$$

* Corresponding author. E-mail: dicosmo@ens.fr.

PII S0304-3975(96)00121-1

On the other hand, if we turn the η equality the other way around into an *expansion* instead of a contraction

(η_e) $\quad M \to \lambda x.Mx$ if $M:A \to B$ and $x \notin FV(M)$

the previous counterexample simply goes away as it becomes $f \xrightarrow{\eta_e} \lambda x.fx \xrightarrow{alg} \lambda x.a$.

The expansive interpretation of extensional equalities was pioneered by Mints [24] and in the last years there has been an increasing interest in this reading of extensional rules, (see for example [1, 11, 10, 7, 14], where the applicability of the η_e rule to a subterm M is restricted in order to guarantee strong normalization, as follows (see [10] for a detailed discussion):

(*structural condition*): M must not be a λ-abstraction,

(*contextual condition*): M must not be applied to an argument.

The rule obtained from η_e when enforcing these conditions is what we call here a *conditional expansion rule*, and will be referred to simply as η in the rest of the paper: even if it seems to be much more restrictive than the other ones, because of these conditions, we will see that together with β and a first-order algebraic system it generates the same equational theory as η_c, that is, we do not loose any equality.

It is then quite legitimate to ask if it is possible to recover the confluence property when combining *confluent* first-order algebraic rewriting systems with the typed lambda calculus and *expansive extensional rules*, and moreover to ask whether this property can be derived in a modular way. Notice that these conditional expansion rules do *not* fit even into the very general framework of higher-order systems [15, 20, 27], where higher-order variables are not allowed as left-hand sides of rules. Notice that if we want to get rid of the specific counterexample above, without using expansion rules, there is also the possibility to state that a function symbol f alone is not a well-formed term if it has not a base type. While this approach can also lead to a confluent system, as shown by [25], it is important to consider that the expansive reading of extensional rules was not born just for handling the counterexample above, but to solve many different problems (like the ones posed by the combination of λ-calculus with the terminal type [6], or with fixpoint operators [26]): all work done using contractions will inevitably be confronted with the same unsurmountable difficulties as soon as these further extensions are considered.

Another natural question arises when taking into consideration fixpoint operators: whenever we want to show confluence in the presence of fixpoints, we usually resort to the technique of labeled reductions that originated in Lévy's work on the untyped lambda-calculus [22]. Furthermore, it has already been shown that fixpoints are compatible with *expansion* rules for surjective pairing [10, 11], while recursion together with the surjective pairing equality oriented as a *contraction* rule causes confluence to fail [26]. This negative result has a similar flavor to the fact that recursion together with non-linear algebraic rules causes confluence (and even uniqueness of normal forms) to fail, as shown for example in [3]. It is quite reasonable to ask again for a more friendly

proof technique based on modular properties, possibly capable of handling conditional expansion rules.

In this paper we give a positive answer to these questions: confluence and normalization are modular properties when combining algebraic rewriting systems with typed lambda calculi featuring extensional rules, and confluence can be modularly derived when adding fixpoints, even in the presence of conditional expansion rules. To do so, we show that reduction to expansive normal form commutes with the other reductions, and this allows to reduce both confluence and strong normalization in presence of extensionality to the already known confluence and strong normalization properties of the system without extensionality.

For the fixpoint combinators we adopt a simulation technique that allows to reduce the confluence property of a quite broad class of reduction relations, including those generated by *left-linear combinatory reduction systems* (*CRSs*) [19] with fixpoints to the confluence of the system *with-out* fixpoints. The restriction of left-linear system is clearly necessary when we have fixpoints, as shown for example in the case of algebraic rewriting in [3]. We also show how to extend this result to the expansive interpretation of extensional rules: they do not fit into the general definition of a left-linear CRS, but can be handled separately.

The paper is logically divided into two main parts: we first show how the combination of first-order algebraic rewriting systems with the typed lambda-calculus preserves strong normalization and confluence even with expansion rules for η and surjective pairing, then we present a general technique for handling fixpoint combinators, even in the presence of expansion rules. These two results will give us the full picture: first-order algebraic rewriting systems can be added by preserving confluence to the extensional simply typed lambda calculus, while fixpoints can be added preserving confluence to any left-linear rewriting system possibly containing expansion rules.

These results provide a simple, clean and powerful way of incorporating extensionality into a higher-order language with algebraic data types, and have clearly immediate application in the field of automated theorem proving, by providing a simple way of deciding extensional equality that does not rely on ad hoc normalization strategies as it is done in previous works.

This paper presents substantial improvements over [9], where some of the results presented here appeared in a restricted form: we prove the preservation of confluence and strong normalization for the combination of typed λ-calculus with algebraic systems and expansion rules in a much simpler way and without requiring left-linearity of the algebraic system; we also present a more detailed and precise treatment of confluence preservation in the presence of fixpoints.

2. Basic definitions

We use the standard notions of substitutions, reduction, normal form, confluence, normalization, etc., from the theory of λ-calculus and rewriting systems [2, 8], and

we recall here the standard definitions concerned with first-order algebraic rewriting systems and extensional typed lambda calculus with pairs. We also fix the notations for the different reduction relations.

Definition 2.1 (*Signature*). A signature $\Sigma = \langle \mathscr{T}, \mathscr{F}, \mathscr{D} \rangle$ consists of

- A set $\mathscr{T}$ of *base types.*
- A set $\mathscr{F}$ of *function symbols*
- A set $\mathscr{D}$ of *declarations* of the form $f : \alpha_1 \to \cdots \to \alpha_n \to \alpha$, where $f \in \mathscr{F}$, $\alpha_1, \ldots, \alpha_n$, $\alpha \in \mathscr{T}$ and $n \geqslant 0$. We say that n is the *arity* of f.

Notice that in our presentation we use *curryfied* algebraic function symbols, while in the classical presentation those algebraic function symbols that are not constants are not terms themselves, but only used to build terms of base types.

We assume the sets $\mathscr{F}$ and $\mathscr{T}$ to be disjoint and we require every functional symbol in $\mathscr{F}$ to have exactly one declaration in $\mathscr{D}$. From now on, we suppose the signature Σ to be fixed. Given a signature Σ, we define the set of types A and terms M of our calculus by the following grammar:

$$A ::= \xi \mid A \times A \mid A \to A$$

$$M ::= f \mid x \mid \lambda x : A.M \mid MM \mid \langle M, M \rangle \mid \pi_1(M) \mid \pi_2(M)$$

where ξ ranges over $\mathscr{T}$, f ranges over $\mathscr{F}$ and x ranges over a countable set of variables. We will often write $MN_1 \cdots N_k$ for the term $((\ldots(MN_1)\ldots)N_k)$.

Variables and constants are typed by the following axioms:

$$x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i \quad (1 \leqslant i \leqslant n)$$

$$x_1 : A_1, \ldots, x_n : A_n \vdash f : A \quad \text{if } f : A \text{ is in the signature}$$

where the x_j's are pairwise distinct.

And terms are typed by the following rules:

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B}$$

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \times A_2} \qquad \frac{\Gamma \vdash M : B_1 \times B_2}{\Gamma \vdash \pi_1(M) : B_1} \qquad \frac{\Gamma \vdash M : B_1 \times B_2}{\Gamma \vdash \pi_2(M) : B_2}$$

Notice that in our typing system it is possible for an algebraic function symbol to be *partially* applied, i.e. to be applied to less arguments that its arity.

In the following, we will only work with well-typed terms, even if we will often omit types when they are clear from the context.

Definition 2.2 (*Context*). Let $\Box$ be a fresh symbol. A term with one occurrence of $\Box$ is called a *context*, and is written $C[\ ]$. The result of replacing the occurrence of $\Box$ in $C[\ ]$ with a term t is written $C[t]$.

Definition 2.3 (*Algebraic term*). A term is *algebraic* if it is either a variable of base type or has the form $f\ T_1 \cdots T_n$, where $f \in \mathscr{F}$ has arity n, and every T_i is an algebraic term. Note that an algebraic term is always of base type according to our definition.

Definition 2.4 (*Algebraic rewriting system*). A *first-order algebraic rewriting rule* is an ordered pair (T, U) of algebraic terms such that T is not a variable, and every variable of U also appears in T. An *algebraic rewriting system* $\mathscr{A}$ is a finite set $\{(T_i, U_i)\}_{i=1}^n$ of algebraic rewriting rules.

Definition 2.5 (*Left-linear algebraic rewriting system*). An algebraic rewriting system is *left-linear* if no variable occurs on the left-hand side of the same rule more than once.

We note $FV(M)$ the set of *free variables* of the term M. We write $[\overline{N}/\overline{x}]$ for the typed substitution mapping each variable $x_i : A_i$ to a term $N_i : A_i$ and $M[\overline{N}/\overline{x}]$ for the term M where each variable x_i free in M is replaced by N_i.

Definition 2.6 (*Algebraic reduction*). The algebraic reduction between terms is a binary relation $\xrightarrow{\mathscr{A}}$, defined in such a way that $C[T'] \xrightarrow{\mathscr{A}} C[U']$ if and only if there exists a substitution θ and a rule $(T, U) \in \mathscr{A}$ such that $T' = \theta(T)$ and $U' = \theta(U)$.

The higher-order rewrite relation. We now give a formal definition of the various rewrite relations we are interested in. First of all, notice that these relations will *not* be congruences (precisely because of the contextual condition on the expansion rules), so it will not be enough to just give the basic rewrite rules that operate on a root redex, as is done traditionally, but we will also need to explicitly say for which contexts the relation is closed.

We identify terms up to α-conversion and we consider the following set of basic rewrite rules:

$$
\begin{array}{lll}
(\beta) & (\lambda x : A.M)N \xrightarrow{\beta} M[N/x] & \\
(\pi_i) & \pi_i\langle M_1, M_2\rangle \xrightarrow{\pi_i} M_i, & \text{for } i = 1,2 \\
(SP) & M \xrightarrow{SP} \langle \pi_1(M), \pi_2(M)\rangle, & \text{if } \begin{cases} M : A \times B \\ M \text{ is not a pair} \end{cases} \\
(\eta) & M \xrightarrow{\eta} \lambda x : A.Mx, & \text{if } \begin{cases} x \notin FV(M) \\ M : A \to C \\ M \text{ is not a } \lambda\text{-abstraction} \end{cases}
\end{array}
$$

Definition 2.7 (*Relation* $\Rightarrow$). The one-step reduction relation between terms, denoted $\Rightarrow$, is defined to be the closure of the reduction rules $\beta, \eta, \pi_1, \pi_2, SP$ for all the contexts *except* application and projection, i.e.,

- If $M \Rightarrow M'$, then $MN \Rightarrow M'N$ except in the case $M \xrightarrow{\eta} M'$
- If $M \Rightarrow M'$, then $\pi_i(M) \Rightarrow \pi_i(M')$ except in the case $M \xrightarrow{SP} M'$

This amounts to forbidding expansions of terms that are either applied or projected.

Notation 2.8. The transitive and the reflexive transitive closure of $\Rightarrow$ are noted $\Rightarrow^+$ and $\Rightarrow^*$, respectively.

We use $\overset{B}{\Longrightarrow}$ to denote the reduction relation $\Rightarrow$ without extensional rules, and $\overset{E}{\Longrightarrow}$ to denote the relation $\Rightarrow$ with only extensional rules.
We will also write M^R for the R-normal form of a term M.

We denote by $\rightsquigarrow$ the reduction relation $\Rightarrow \cup \xrightarrow{\mathscr{A}}$, by $\rightsquigarrow_B$ the relation $\overset{B}{\Longrightarrow} \cup \xrightarrow{\mathscr{A}}$ and by $\rightsquigarrow_E$ the relation $\overset{E}{\Longrightarrow} \cup \xrightarrow{\mathscr{A}}$.

Theorem 2.9. *The reflexive, symmetric and transitive closure of $\Rightarrow$ generates the same equational theory as the one obtained using the traditional equalities for β, π, SP and η. The same holds for $\rightsquigarrow$.*

Proof. By a simple case analysis. This is done for $\Rightarrow$ in Theorem 3.3 of [10], but the same proof holds for $\rightsquigarrow$, because the conditions imposed on expansions only involve higher-order terms, and do not interfere with $\xrightarrow{\mathscr{A}}$. □

3. Modularity of confluence and strong normalization

We know from [3, 5] that combining the non-extensional simply typed lambda calculus with a confluent first-order algebraic rewriting system preserves confluence. On the other hand, this combination yields a strongly normalizing system when the algebraic one is [4, 28]. When one also adds non-extensional pairs [12] shows, adapting the techniques in [3], that confluence and strong normalization are modular properties for the combination with *left-linear* algebraic rewriting systems. These prior results are used by our proof technique for incorporating the expansion rules.

3.1. An overview of the proof technique

To do so, we use a technique originally developed in [10] for a specific typed lambda calculus with expansion rules, that is general enough to be applied in this context. This amounts to find a translation between rewriting systems with some special properties.

Definition 3.1 (*Simulation*). Given any reduction relations R_1 and R_2, a translation $-^\circ$ from $R_1 \cup R_2$ to R_2 has the *simulation property* if $M \xrightarrow{R_1 \cup R_2} N$ implies $M^\circ \xrightarrow{R_2} {}^+N^\circ$. Also, we say that $-^\circ$ from $R_1 \cup R_2$ to R_2 has the *weak simulation property* if $M \xrightarrow{R_1 \cup R_2} N$ implies $M^\circ \xrightarrow{R_2} {}^*N^\circ$.

We will often call (*weak*) *simulation* a translation with the (weak) simulation property.

The *simulation* property alone is sufficient to show that strong normalization is preserved when adding expansion rules, while a *weak simulation* which is the identity on expansive normal forms allows to derive the preservation of confluence.

Proposition 3.2 (Normalization via translation (I)). *Given two reduction relations R_1, R_2 on a given set of terms, and a simulation from $R_1 \cup R_2$ to R_2, then if R_2 is strongly normalizing, also $R_1 \cup R_2$ is strongly normalizing.*

Proof. Straightforward since any infinite reduction sequence of $R_1 \cup R_2$ can be turned via simulation into an infinite reduction of R_2, leading to a contradiction with the hypothesis. □

Proposition 3.3 (Normalization via translation (II)). *Let R_1, R_2 be two strongly normalizing reduction relations on a given set of terms. If any reduction step $M \xrightarrow{R_2} N$ can be simulated by a reduction sequence $M^{R_1} \xrightarrow{R_2}{}^{+} N^{R_1}$, then if R_1 is confluent, also $R_1 \cup R_2$ is strongly normalizing.*

Proof. Suppose that $R_1 \cup R_2$ is not strongly normalizing. Then there exists an infinite reduction sequence of $R_1 \cup R_2$, and this sequence must contain an infinitely many R_1-steps as well as R_2-steps (because both R_1 and R_2 are strongly normalizing). Using the confluence of R_1, this sequence can be turned via simulation into an infinite reduction of R_2, leading to a contradiction with the hypothesis. □

Proposition 3.4 (Confluence via translation). *Given two reduction relations R_1, R_2 on a given set of terms, and a translation $-^\circ$ s.t.*

- *Any reduction step $M \xrightarrow{R_1 \cup R_2} N$ can be (weakly) simulated by a reduction $M^\circ \xrightarrow{R_2}{}^{*} N^\circ$*
- *The translation is the identity on the R_1-normal-forms*

then if R_2 is confluent and R_1 is weakly normalizing, $R_1 \cup R_2$ is confluent.

Proof. Fig. 1 shows how to close any diagram of $R_1 \cup R_2$. The dotted line pinpoints the first translation step from M to M°, while $P_i^{R_1}$ is any R_1-normal form of P_i, and $\equiv$ is equality up to α-conversion.

The weak normalization property for R_1 ensures the existence of the R_1-normal forms $P_1^{R_1}$ and $P_2^{R_1}$. The link between the outer and inner diagrams can be done by the second property since translation does not affect R_1-normal forms. Finally, the inner diagram can be closed by the confluence property of the reduction relation R_2. □

Remark 3.5 (*A simple generalization*). It is easy to obtain a simple generalization of the previous proposition. If one takes any reduction $R' \subseteq (R_1 \cup R_2)^*$ which weakly simulates $(R_1 \cup R_2)^*$, then confluence of R' still implies confluence of $(R_1 \cup R_2)^*$. We decided, however, to present the proposition with $R' = R_2$, since this is the form that we will need all along the paper.

In the special case where the translation $-^\circ$ is exactly the R_1-normal form, then we can prove a stronger abstract result, which generalizes the *generalized interpretation method* of [16].

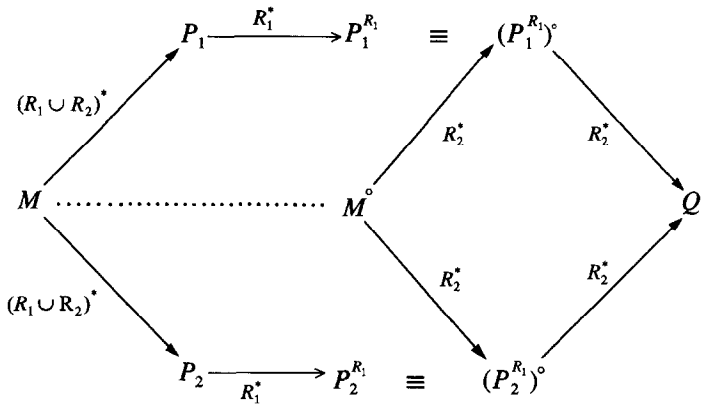

Fig. 1.

Proposition 3.6. *Let R_1 and R_2 be two reduction systems such that R_1 is weakly normalizing and f is a deterministic normalization strategy taking any term M to an R_1-normal form, named $f(M)$. Let R' be another reduction system on the set of R_1-normal forms satisfying the following properties:*

- $R' \subseteq (R_1 \cup R_2)^*$
- *Any reduction step $M \xrightarrow{R_1 \cup R_2} N$ can be (weakly) simulated by $f(M) \xrightarrow{R'} {}^* f(N)$*

then R' is confluent if and only if $R_1 \cup R_2$ is confluent.

Proof. The only if part can be proven as in Proposition 3.4: assume R' is confluent and suppose $P^* \xleftarrow{R_1 \cup R_2} M \xrightarrow{R_1 \cup R_2} {}^* N$. Using the fact that f is a deterministic weak simulation we have $f(P)^* \xleftarrow{R'} f(M) \xrightarrow{R'} {}^* f(N)$. Since R' is confluent, we have $f(P) \xrightarrow{R'} {}^* Q^* \xleftarrow{R'} f(N)$ form some R_1-normal form Q. As $R' \subseteq (R_1 \cup R_2)^*$, we obtain $f(P) \xrightarrow{R_1 \cup R_2} {}^* Q^* \xleftarrow{R_1 \cup R_2} f(N)$. Finally, f takes a term to its R_1-normal form, so we have $N \xrightarrow{R_1} {}^* f(N)$ and $P \xrightarrow{R_1} {}^* f(P)$ which allows us to close the diagram.

As for the if part, assume $R_1 \cup R_2$ is confluent and let M, P and Q be R_1-normal forms. Then any divergence $P^* \xleftarrow{R'} M \xrightarrow{R'} {}^* Q$ is a divergence $P^* \xleftarrow{R_1 \cup R_2} M \xrightarrow{R_1 \cup R_2} {}^* Q$ by the first hypothesis so that the diagram can be closed by $P \xrightarrow{R_1 \cup R_2} {}^* N^* \xleftarrow{R_1 \cup R_2} Q$. By the weak simulation hypothesis we have $f(P) \xrightarrow{R'} {}^* f(N)^* \xleftarrow{R'} f(Q)$, but P and Q are R_1-normal forms so $P = f(P)$ and $Q = f(Q)$ which makes it possible to conclude the proof. □

3.2. Combining first-order algebraic rewriting systems with λ-calculus and extensional rules

In this section we show that confluence and strong normalization are modular properties of the combination of first-order algebraic rewriting systems with the extensional typed lambda calculus with pairing and η based on *expansion rules*. The results presented here generalize the ones originally presented in [9], as we can show modularity

of confluence and strong normalization in the combination of first-order algebraic rewriting systems with λ-calculus and extensional rules, *without* requiring left-linearity of the algebraic rewriting systems.

We proceed now to show that the translation from $\rightsquigarrow$ into $\rightsquigarrow_B$ that sends a term M to its expansive normal M^E form has the following properties:

- any reduction step $M \rightsquigarrow_B N$ can be simulated by a reduction sequence $M^E \rightsquigarrow_B^+ N^E$
- any reduction step $M \rightsquigarrow N$ can be simulated by a reduction sequence $M^E \rightsquigarrow_B^* N^E$
- the translation is, by definition, the identity on the (η, SP)-normal forms

As a consequence, the desired modularity results will be derived from the above Propositions 3.3 and 3.4, using the following well-known results about expansions in simply typed λ-calculus.

Theorem 3.7 (Expansions are SN and CR). *The relation* $\overset{E}{\Longrightarrow}$ *is strongly normalizing and confluent.*

Proof. Several proofs of strong normalization can be found, either direct, using a decreasing measure as in [23, 17], or indirect as in [9]. Confluence then follows by Newman's Lemma. □

Theorem 3.8 (Simulation over expansive-normal forms). *If* $M \overset{B}{\Longrightarrow} N$, *then* $M^E \overset{B}{\Longrightarrow}^+ N^E$.

Proof. This is done by an analysis of expansive normal forms in Lemma 14 of [1]. A clean proof for a more complex calculus can also be found in [18]. □

We now prove a similar result for first-order algebraic rewriting vs. expansive normal forms.

Lemma 3.9. *For every substitution* θ *and every algebraic term* T *there is a substitution* φ *such that*:

1. $\varphi(T) = \theta(T)^E$;
2. *for all algebraic term* U *with* $FV(U) \subseteq FV(T)$, $\varphi(U) = \theta(U)^E$.

Proof. First of all, for every variable y, and every substitution θ we define $\varphi(y) = \theta(y)^E$.

1. Let now T be any algebraic term; we proceed by induction on the structure of T.

- $T \equiv x$. Then we have $\varphi(x) = \theta(x)^E$.
- $T \equiv f\ T_1 \cdots T_n$. Then $\theta(T) \equiv f\ \theta(T_1) \cdots \theta(T_n)$ and by i.h. there is a substitution φ_i such that $\varphi_i(x) = \theta(x)^E$ for all $x \in FV(T_i)$ and $\varphi_i(T_i) = \theta(T_i)^E$ for $i = 1 \cdots n$. Since the expansive normal form of a term is unique (because of Theorem 3.7), all the φ_i's agree on their common variables, so we can define φ as $\varphi_1 \cup \cdots \cup \varphi_n$ and then $\varphi(f\ T_1 \cdots T_n) = f\ \varphi_1(T_1) \cdots \varphi_n(T_2) = f\ \theta(T_1)^E \cdots \theta(T_n)^E = (f\ \theta(T_1) \cdots \theta(T_n))^E = \theta(T)^E$.

2. As for the second property, it is enough to remark that the expansive normal form of $\theta(U)$, when U is algebraic, is uniquely determined by its value on $\theta(x)$ for all variable x in U and this is precisely what φ gives. □

Theorem 3.10 (Algebraic simulation over expansive-normal forms). *If* $M \xrightarrow{\mathscr{A}} N$, *then* $M^E \xrightarrow{\mathscr{A}}{}^{+} N^E$.

Proof. By induction on the structure of M, using the fact that an algebraic term has base type, so it cannot be expanded at the root. We just show here the case when $\xrightarrow{\mathscr{A}} N$ is a root reduction step.

Let $M = \theta(T)$, where $(T, U) \in \mathscr{A}$ and $N = \theta(U)$. Now by Lemma 3.9 there is a substitution φ such that $\varphi(T) = \theta(T)^E$. So we can apply the algebraic rule on the E-normal form of M and get $M^E = \theta(T)^E = \varphi(T) \xrightarrow{\mathscr{A}} \varphi(U)$. But Lemma 3.9 also says $\varphi(U) = \theta(U)^E$, so $M^E \xrightarrow{\mathscr{A}} N^E$ as needed. □

We can then conclude that

Theorem 3.11 (Simulation for expansions).
- *If* $\Gamma \vdash M : A$ *and* $M \rightsquigarrow_B N$, *then* $M^E \rightsquigarrow_B^+ N^E$.
- *If* $\Gamma \vdash M : A$ *and* $M \rightsquigarrow N$, *then* $M^E \rightsquigarrow_B^* N^E$.

Proof. By combining the previous Theorems 3.7, 3.8 and 3.10. □

This is enough to get our modularity results:

Theorem 3.12 (Modularity of the strong normalization property). *Let* $\mathscr{A}$ *be any strongly normalizing first-order algebraic rewriting system. Then* $\mathscr{A}$ *plus the simply typed lambda calculus with pairing and expansion rules for* η *and SP is strongly normalizing.*

Proof. We know by [4] that the combination of a strongly normalizing first-order algebraic rewriting system with the lambda calculus yields a strongly normalizing reduction, and the technique used there extends easily to handle pairing (actually, since pairs are definable in the simply typed lambda calculus, one can also get this result by a simple simulation). By Proposition 3.3 (taking as R_1 the reduction $\xRightarrow{E}$, which is confluent and strongly normalizing by Theorem 3.7, and $\rightsquigarrow_B$ as R_2) and Theorem 3.11, our reduction $\rightsquigarrow$ is strongly normalizing. □

Theorem 3.13 (Modularity of the confluence property). *Let* $\mathscr{A}$ *be any confluent first-order algebraic rewriting system. Then* $\mathscr{A}$ *plus the simply typed lambda calculus with pairing and expansion rules for* η *and SP is confluent.*

Proof. We know by [3, 5] that the combination of a confluent first-order algebraic rewriting system with the lambda calculus yields a confluent reduction, and the technique extends easily to handle pairing. So, our reduction relation $\rightsquigarrow_B$ is confluent. In view of Proposition 3.4 (taking $\overset{E}{\Longrightarrow}$ as R_1, $\rightsquigarrow_B$ as R_2 and the reduction to $\overset{E}{\Longrightarrow}$-normal form as the translation), and using Theorems 3.11 and 3.7 this is enough to deduce that $\rightsquigarrow$ is also confluent. □

4. Adding recursion to a rewriting system

We focus now on adding fixpoint operators to a rewriting system S. We assume a new constant $fix_A : (A \to A) \to A$ for each type A, with the reduction rule [1]

$$(fix) \qquad fix_A \xrightarrow{fix} \lambda f.f(fix_A f)$$

We also assume that the fix constant does not occur in any rule of S. We will drop the type subscript in the rest of this section, as it is not necessary for the proofs to go through. Indeed, all the proofs hold also in an untyped setting.

4.1. The traditional approach

Let us recall here a proof technique essentially due to Lévy (see [22]) that is used quite often to prove that a specific confluent calculus stays confluent when combined with a fixpoint operator. Usually, one considers an auxiliary reduction relation with *bounded* fixpoint operators fix^n and the more restrictive reduction rule [2]

$$(\overline{fix}) \qquad fix^n \xrightarrow{\overline{fix}} \lambda f.f(fix^{n-1} f) \quad n > 0$$

where fix^0 can be taken as a fixed fresh variable y not occurring anywhere else. Essentially, this puts a bound on the depth of any recursive call, so to eliminate infinite sequences of fix reductions: sometimes, this is enough to preserve strong normalization of the original reduction relation S. If it happens that local confluence also still holds, then by Newman's Lemma we have confluence of this auxiliary reduction relation $\xrightarrow{S \cup \overline{fix}}$. Then for the specific system under consideration one has to show the following facts:

(*Erasing*): If $M \xrightarrow{S \cup \overline{fix}} N$, then $|M| \xrightarrow{S \cup fix} |N|$, where $|M|$ is obtained from M by removing all the superscripts from the fix terms.

(*Bounding*): For any reduction sequence $M_0 \xrightarrow{S \cup fix} M_1 \xrightarrow{S \cup fix} \cdots \xrightarrow{S \cup fix} M_n$, there exists an indexed computation $N_0 \xrightarrow{S \cup \overline{fix}} N_1 \xrightarrow{S \cup \overline{fix}} \cdots \xrightarrow{S \cup \overline{fix}} N_n$ such that $|N_i| = M_i$, for $i = 0 \cdots n$

---

[1] We choose to work with this rule instead of the more common one $fix_A M \xrightarrow{fix} M(fix_A M)$, because it is computationally equivalent but allows to simplify the proofs a bit. This version is also used for example in [12].

[2] The corresponding bounded version for the more common fixpoint rule is $fix^n M \xrightarrow{\overline{fix}} M(fix^{n-1} M)$ $n > 0$.

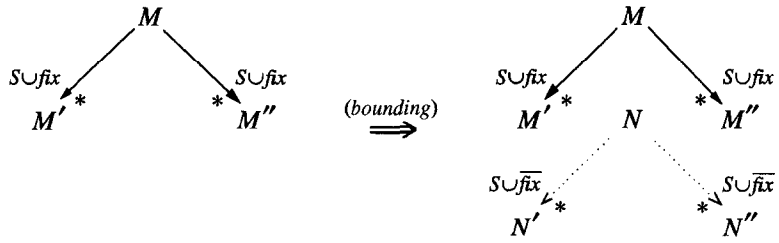

Fig. 2.

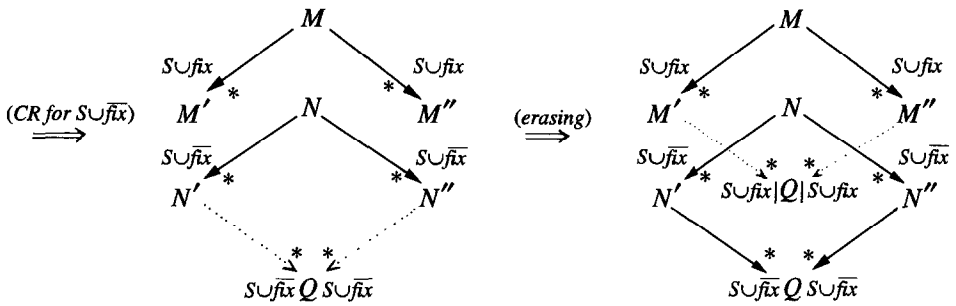

Fig. 3.

(usually, but not always, as we will see in Example 4.8, it suffices to index all the fix terms in M_0 by a number $k \geqslant n$).

Finally, using just the confluence property for $\xrightarrow{S\cup\overline{fix}}$, the confluence of the reduction relation $\xrightarrow{S\cup fix}$ can be derived using the following general proposition.

Proposition 4.1. *If* $\xrightarrow{S\cup fix}$ *and* $\xrightarrow{S\cup\overline{fix}}$ *satisfy the erasing and bounding properties, then confluence of* $\xrightarrow{S\cup\overline{fix}}$ *implies confluence of* $\xrightarrow{S\cup fix}$.

Proof. Take any diagram $M'^* \xleftarrow{S\cup fix} M \xrightarrow{S\cup fix} {}^*M''$. By bounding we get $N'^* \xleftarrow{S\cup\overline{fix}} N \xrightarrow{S\cup\overline{fix}} {}^*N''$ with $M = |N|$, $M' = |N'|$ and $M'' = |N''|$ (see Fig. 2). Then by confluence of $\xrightarrow{S\cup\overline{fix}}$ there exists a Q s.t. $N' \xrightarrow{S\cup\overline{fix}} {}^*Q^* \xleftarrow{S\cup\overline{fix}} N''$ and finally by erasing we get that $M' \xrightarrow{S\cup fix} {}^*|Q|^* \xleftarrow{S\cup\overline{fix}} M''$ (see Fig. 3).

4.2. Our approach

Only two points of the traditional approach are really needed to prove confluence with unbounded recursion:

1. The erasing and bounding properties.
2. Bounded fixpoints preserve confluence of the original specific rewriting system.

Indeed, the strong normalization property of $\xrightarrow{\overline{fix}}$ is used only as a technical tool to prove confluence of $\xrightarrow{\overline{fix}}$, and does not have real interest: we show that bounded fixpoints preserve confluence for *any* reduction relation, and we do not need at all the strong normalization property of $\xrightarrow{\overline{fix}}$ in the proof. Surprisingly enough, only the bounding property puts some constraint on the rewriting system under consideration (as we will see later, we have to forbid non-left-linear reduction rules for bounding to hold).

4.3. Confluence with bounded fixpoints

Instead of adapting an existing confluence proof to take into account the bounded fixpoint reduction rule, as is usually done, it is better to look for a more modular approach. It turns out that such an approach exists indeed, and is based on the existence of a translations of the fix^n constants that enjoys the weak simulation property and is the identity on $\overline{fix}$-normal forms. This is enough to prove confluence of the mixed system without requiring any kind of information on the termination properties of the original rewriting system.

Proposition 4.2 (Bounded recursion preserves confluence). *Let S be any rewriting system. If there exists a translation $[\![\ ]\!]$ from $S \cup \overline{fix}$ to S s.t.*

- *$[\![\ ]\!]$ is a weak simulation from $S \cup \overline{fix}$ to S, (i.e. $M \xrightarrow{S\cup\overline{fix}} N$ implies $[\![M]\!] \xrightarrow{S} {}^*[\![N]\!]$)*
- *$[\![\ ]\!]$ is the identity on $\overline{fix}$-normal forms*

then if S is confluent so is S plus bounded recursion.

Proof. Assume S is confluent. We apply Proposition 3.4 taking $\overline{fix}$ as R_1 and S as R_2. The required properties of $[\![\ ]\!]$ there, are satisfied by hypothesis here. Also, the strong normalization of $\xrightarrow{\overline{fix}}$ alone is trivial since each step strictly decreases the simple measure on terms given by the sum of all indexes n_i of all occurrences of bounded fixpoints. We can then conclude that S plus bounded recursion is confluent. □

We can now state our general theorem for the preservation of confluence in the presence of fixpoints.

Theorem 4.3 (Unbounded fixpoints vs. confluence). *Given an arbitrary confluent rewriting relation S, then $S \cup fix$ is confluent whenever the following conditions hold.*

(Erasing): If $M \xrightarrow{S\cup\overline{fix}} N$, then $|M| \xrightarrow{S\cup fix} |N|$, where $|M|$ is obtained from M by removing all the superscripts from the fix terms.

(Bounding): For any reduction sequence $M_0 \xrightarrow{S\cup fix} M_1 \xrightarrow{S\cup fix} \cdots \xrightarrow{S\cup fix} M_n$, there exists an indexed computation $N_0 \xrightarrow{S\cup\overline{fix}} N_1 \xrightarrow{S\cup\overline{fix}} \cdots \xrightarrow{S\cup\overline{fix}} N_n$ such that $|N_i| = M_i$, for $i = 0 \cdots n$ (usually it suffices to index all the fix terms in M_0 by a number $k \geqslant n$).

(*Weak simulation*): *There exists a weak simulation from* $S \cup \overline{fix}$ *to* S *which is the identity on* $\overline{fix}$*-normal forms.*

Proof. This is a direct consequence of Propositions 4.1 and 4.2. □

Remark 4.4. These very simple arguments hold for whatever abstract relation S and whatever definition of $S \cup fix$ we may choose, but in most cases the erasing property holds trivially and one can use as the required simulation an appropriate context closure of this very simple and natural translation:

$$[\![fix^0]\!] = fix^0 \equiv y \text{ (the fixed fresh variable in the def.)}$$
$$[\![fix^n]\!] = \lambda f.f([\![fix^{n-1}]\!]f)$$

which has the crucial property that

$$[\![fix^n]\!] = \lambda f.f([\![fix^{n-1}]\!]f) = [\![\lambda f.f(fix^{n-1}f)]\!]$$

The only problems arise in the verification of the bounding property due to some form of non-left-linearity.

We will try to substantiate this general remark, in what follows, by considering some very powerful framework for defining term rewriting systems.

4.4. Fixpoints and CRSs

When the reduction system S is a CRS as defined in [21], the fixpoint rule is presented as

$$(fix) \qquad fix \to \lambda([f]@(f, @(fix, f)))$$

It is immediate to verify the *erasing* property, and the simple translation suggested above becomes:

Definition 4.5 (*Simple translation for fixpoints in CRSs*). Let λ be a unary operator used for λ-abstraction and @ be a binary operator used for application.

$$[\![fix^0]\!] = fix^0 \equiv Y \text{ (the } fix\text{ed fresh variable in the def.)}$$
$$[\![fix^n]\!] = \lambda([f]@(f, @([\![fix^{n-1}]\!], f)))$$

One can extend $[\![\ ]\!]$ to all the terms of the CRS by defining $[\![M]\!] = M[[\![\overrightarrow{fix^i}]\!] / \overrightarrow{fix^i}]$, that is, by just replacing all fix^i constants with their translations.

Example 4.6. Let M be *pair* $(@(fix^2, H), fix^0)$, where *pair* is a binary operator used for pairs. Then its translation is

$$pair(@(\lambda([f]@(f, @(\lambda([g]@(g, @(Y, g)))), f)), H), Y)$$

This translation has the weak simulation property: indeed, one-step reduction of bounded recursion can be simulated by an empty step in the reduction system without recursion as follows:

$$[\![fix^n]\!] = \lambda([f]@(f, @([\![fix^{n-1}]\!], f))) = [\![\lambda([f]@(f, @(fix^{n-1}, f)))]\!]$$

Also, the translation is the identity on $\overline{fix}$-normal forms, so the hypothesis of Theorem 4.3 is then verified.

This is different from what is done in [11, 9], where to simulate a step of bounded recursion one needs a β reduction step, and hence β must be part of the original rewriting system.

Using this translation and Proposition 4.2, we immediately have:

Corollary 4.7 (Bounded recursion preserves confluence of CRSs). *If S is a confluent CRS, then so is $S \cup \overline{fix}$.*

Left-linearity and the *bounding* property. When we turn to the unbounded fixpoint operator, though, we face a difficulty that limits our results: for a general reduction relation $\xrightarrow{S}$, the confluence (with or without normalization) of $\xrightarrow{S \cup \overline{fix}}$ does not imply confluence for $\xrightarrow{S \cup fix}$. The reason is that if there is some rule (like the contractive version of surjective pairing) where some metavariable appears more than once, it is easy to build counterexamples, like the following one, to the bounding property that is crucial for Lévy's trick.

Example 4.8. Let $I = \lambda x.x$, and consider the reduction sequence $(\lambda p.\langle \pi_1(p), \pi_2(p)\rangle)$ $(fix\, I) \xrightarrow{\beta} \langle \pi_1(fix\, I), \pi_2(fix\, I)\rangle \xrightarrow{fix} \langle \pi_1(I(fix\, I)), \pi_2(fix\, I)\rangle \xrightarrow{\beta} \langle \pi_1(fix\, I), \pi_2$ $(fix\, I)\rangle \xrightarrow{SP_{contr}} fix\, I$.

Whatever index n we associate to the original fix operator, there is no way to simulate the SP_{contr} reduction step in the labeled calculus as required by Proposition 4.1, because the occurrences of fix in the first component of the pair $\langle \pi_1(fix\, I), \pi_2(fix\, I)\rangle$ and the occurrences of fix in the second component of the same pair will always have labels differing by 1 (such counterexamples are easy to build for any nonleft-linear rewriting rule).

This very same counterexample can be encoded in a CRS having the following reduction rules:

(β) $\quad @(\lambda([x]A(x)), B) \to A(B)$
(SP) $\quad pair(pi_1(M), pi_2(M)) \to M$
(fix) $\quad fix \to \lambda([f]@(f, @(fix, f)))$

Indeed, we know from [26] that Klop's counterexample [19] can be adapted to show that the contractive version of surjective pairing is not confluent in the presence of a fixpoint operator; we also know from [3] that nonlinear algebraic rules cause confluence (and even uniqueness of normal forms) to fail in the presence of fixpoints.

Nevertheless, if the CRS S is *left-linear*, then it is straightforward to verify that $\xrightarrow{S\cup\overline{fix}}$ can simulate $\xrightarrow{S\cup fix}$. On the other hand, also the erasing property is trivially verified (even if the system is not left-linear) since a reduction step of bounded recursion can be simulated by a reduction step of unbounded recursion by just removing the superscripts from the fix constants. We can then conclude that:

Theorem 4.9 (Recursion preserves confluence of left-linear CRSs). *Let S be any left-linear CRS. If S is confluent, then so is S plus unbounded recursion.*

One important remark here is that even if unbounded recursion is orthogonal with all the other rules of the confluent system S (since we have added a *new* constant fix), the system S is not necessarily *orthogonal*, so the result of the previous theorem is *not* a particular case of the confluence proof in [21] for orthogonal CRSs.

4.5. Rewriting with fixpoints and expansion rules

A natural question that arises now is whether we can extend this rather general result to the case of the expansionary reductions we used in the first part of this paper. Indeed, these *conditional* rewriting rules, do *not* fit in the general schema for CRSs.

We claim that the preservation of confluence holds in general for left-linear systems extended with these expansion rules (where the notion of left linearity depends on the particular formalism we choose to describe the rewriting systems). Again, we can substantiate this claim by showing how to proceed in the very general case of left-linear CRSs extended with expansion rules, but for the sake of readability, we will write down λ-terms in the usual notation, and not in the CRS formalism as in the previous section.

The proof technique relies on a translation, but we cannot proceed exactly as we did in the absence of expansion rules, because now the conditional nature of the expansions can prevent the translation of an expandable term from being expandable: for example, fix^n can be expanded to $\lambda g.fix^n g$, but its translation $\lambda f.f([\![fix^{n-1}]\!]f)$ cannot be expanded to $\lambda g.(\lambda f.f([\![fix^{n-1}]\!]f))g$ since it is already a λ-abstraction. Notice that since fix^n constants have always functional types, this problem arises only to accommodate η, while SP expansions are not problematic.

To avoid this pitfall, it is enough to translate the fixpoints into something that is not a λ-abstraction, and there are many possibilities to do so. Let us see, for example, the following slight variation of the original translation, using $I = \lambda x.x$:

Definition 4.10 (*Expansion compatible translation*).

$$\ll fix^0 \gg = fix^0 \equiv y \text{ (the fixed fresh variable in the def.)}$$

$$\ll fix^n \gg = I(\lambda f.f(\ll fix^{n-1} \gg f))$$

The translation $[\![M]\!]$ of a term M is M where every occurrence of a fix^i constant is replaced by $[\![fix^i]\!]$.

For this translation, η expansions are no longer problematic: indeed, the translation of a fix^n is always a variable or an application, that can be expanded. Furthermore, fix^n reductions can be simulated on the translation using β, so we can prove the following:

Proposition 4.11 (Simulation for bounded fixpoints with expansions). *Let S be any rewriting system containing β and the expansion rules. If $M \xrightarrow{S \cup \overline{fix}} N$, then $\ll M \gg \xrightarrow{S}{}^{+} \ll N \gg$.*

Proof. We can simulate fix^n reductions using β:

$$\ll fix^n \gg = I(\lambda f.f(\ll fix^{n-1} \gg f)) \xrightarrow{\beta} \lambda f.f(\ll fix^{n-1} \gg f)$$
$$= \ll \lambda f.f(fix^{n-1} f) \gg$$

And the simulation of η expansion of fix^n constants is now possible:

$$\ll fix^n \gg = I(\lambda f.f(\ll fix^{n-1} \gg f)) \xrightarrow{\eta} \lambda g.(I(\lambda f.f(\ll fix^{n-1} \gg f)))g$$
$$= \ll \lambda g.fix^n g \gg$$

Since all other reduction rules do not invole fix, this concludes the proof. □

Remark 4.12 (*Strong normalization with expansions and bounded fixpoints*). An immediate consequence of Proposition 4.11 is that bounded fixpoints preserve strong normalization even in the presence of expansion rules.

Finally, we apply again Theorem 4.3 and conclude that

Theorem 4.13 (Modularity of confluence with fix and expansions). *Let L be any left-linear CRS containing β and extended with expansion rules for η and surjective pairing. If L is confluent, then so is L plus unbounded recursion.*

Proof. Erasing is trivially verified, and we have just given the weak simulation which is the identity on $\overline{fix}$-normal forms. Bounding can easily be proved due to the left-linearity of L. □

Remark 4.14 (*Avoiding β*). It is not necessary to require the system L to really contain the β rule: the proofs above go through unchanged if we assume that I, instead of being an abbreviation for the term $\lambda x.x$, is a fresh combinator with the associated rewriting rule

$$IM \to M$$

It is then enough to show that adding this rule to the original system preserves confluence, which is in general fairly easy to check, since I is a new combinator and it does not appear in any other rule.

4.6. *Strong normalization with β and bounded fixpoints*

For the sake of completeness, we briefly remark here that, as noticed in [11, 9], it is also possible to translate bounded fixpoints in a way that a one-step fixpoint reduction can be simulated with at least one step of the β rule of the simply typed λ-calculus. This is the case, for example, if we take the very same translation we used above (cf. Definition 4.10) for dealing with expansion rules.

This means that any strongly normalizing rewriting system that contains β or some kind of combinator that can be used to code the identity combinator I seen above, stays strongly normalizing when bounded fixpoints are added.

This very simple and general result does not have real interest because we have already shown that strong normalization is not necessary at all to prove confluence with fixpoints, but we consider it worth mentioning here in view of the fact that it does not seem to be very well known: in many works normalization with bounded fixpoints is proven again and again without noticing that it is a fairly general property [29, 12, 10].

5. Conclusions

We have shown that extensional equalities can easily be incorporated in higher-order programming languages with algebraic data-types: the problems encountered in previous work to accomplish this goal were not due to extensional equalities themselves, but to the wrong choice of orientation of the associated rewrite rules. The well-known modularity results for confluence and normalization extend naturally once we choose expansion rules instead of the traditional contractive ones, thus providing a fully satisfactory solution to this long standing problem. In particular, this provides a simple way of deciding extensional equality that does not rely on ad hoc normalization strategies. We believe that this new approach can be successful also in the framework of polymorphic calculi, where the contractive interpretation of extensional equalities poses similar problems.

We have also shown how to deal in full generality with fixpoint combinators: confluence is preserved under very permissive hypotheses, so that for many interesting calculi it is now possible to focus on the recursion-free fragment.

References

[1] Y. Akama, On Mints' reductions for ccc-Calculus, in: *Typed Lambda Calculus and Applications*, Vol. 664 (Springer, Berlin, 1993) 1–12.

[2] H. Barendregt, *The Lambda Calculus; Its syntax and Semantics* (North-Holland, Amsterdam, revised edn., 1984).

[3] V. Breazu-Tannen, Combining algebra and higher order types, in: *Proc. Symp. on Logic in Computer Science* (LICS), (IEEE, New York, 1988) 82–90.

[4] V. Breazu-Tannen and J. Gallier, Polymorphic rewriting preserves algebraic strong normalization, *Theoret. Comput. Sci.* **83** (1991) 3–28.

[5] V. Breazu-Tannen and J. Gallier, Polymorphic rewriting preserves algebraic confluence, *Inform. Comput.* **114** (1994) 1–24.

[6] P.-L. Curien and R. Di Cosmo, A confluent reduction system for the λ-calculus with surjective pairing and terminal object, in: L. Monien and Artalejo ed., *Internat. Conf. on Automata, Languages and Programming* (*ICALP*), Lecture Notes in Computer Science, Vol. 510 (Springer, Berlin, 1991) 291–302.

[7] D. Cubric, On free CCC, Distributed on the types mailing list, 1992.

[8] Na. Dershowitz and J.-P. Jouannaud, Rewrite systems, in: J. Van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. B, Ch. 6, Formal Models and Semantics (MIT Press, Cambridge, 1990) 243–320.

[9] R. Di Cosmo and D. Kesner, Combining first order algebraic rewriting systems, recursion and extensional lambda calculi, in: S. Abiteboul and E. Shamir, ed., *Internat. Conf. on Automata, Languages and Programming* (*ICALP*), Lecture Notes in Computer Science, Vol. 820 (Springer, Berlin, 1994) 462–472.

[10] R. Di Cosmo and D. Kesner, Simulating expansions without expansions, *Math. Struct. Comput. Sci.* **4** (1994) 1–48; A preliminary version is available as Tech. Report LIENS-93-11/INRIA 1911.

[11] D.J. Dougherty, Some lambda calculi with categorical sums and products, in: *Proc. 5th Internat. Conf. on Rewriting Techniques and Applications* (*RTA*) (1993).

[12] B. Howard and J. Mitchell, Operational and axiomatic semantics of pcf, in: *Proc. LISP and Functional Programming Conf.* (ACM, New York, 1990) 298–306.

[13] C.B. Jay and N. Ghani, The virtues of eta-expansion, Tech. Report ECS-LFCS-92-243, LFCS, 1992, Univ. of Edinburgh, preliminary version of [14].

[14] C.B. Jay and N. Ghani, The virtues of eta-expansion, *J. Functional Programming* **5** (1995) 135–154.

[15] J.-P. Jouannaud and M. Okada, A computation model for executable higher-order algebraic specification languages, in: *Proc. 6th Ann. IEEE Symp. on Logic in Computer Science*, Amsterdam, The Netherlands, July 15–18 1991 (IEEE Computer Society Press, Silver Spring, MD, 1991) 350–361.

[16] F. Kamareddine and Alejandro Ríos, The confluence of the λs_e-calculus via a generalized interpretation method, 1995, Draft.

[17] D. Kesner, La définition de fonctions par cas à l'aide de motifs dans des langages applicatifs, Thèse de doctorat, Université de Paris XI, Orsay, 1993.

[18] D. Kesner, Reasoning about layered, wildcard and product patterns, in: R.-A. Levi, ed. *Internat. Conf. on Algebraic and Logic Programming* (*ALP*), LNCS, Vol. 850 (Springer, Berlin, 1994).

[19] J.W. Klop, Combinatory reduction systems, *Math. Center Tracts* **27** (1980).

[20] J.W. Klop, V. van Oostrom and F. van Raamsdonk, Combinatory reduction systems: introduction and survey, *Theoret. Comput. Sci.* **121** (1993) 279–308; Bohm's Festschrift.

[21] J.W. Klop, V. van Oostrom and F. van Raamsdonk, Combinatory reduction systems: introduction and survey, Tech. Report CS-R9362, CWI, Amsterdam, September 1993.

[22] J.J. Lévy, An algebraic interpretation of the $\lambda\beta\kappa$-calculus and a labelled λ-calculus, *Theoret. Comput. Sci.* **2** (1976) 97–114.

[23] G. Mints, Closed categories and the theory of proofs, *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im. V.A. Steklova AN SSSR*, **68** (1977) 83–114.

[24] G. Mints, Teorija categorii i teoria dokazatelstv. I, *Aktualnye problemy logiki i metodologii nauky* (1979) 252–278.

[25] C. Necula, Algebraic rewriting preserves (β,η) confluence in the typed lambda calculus, Draft Manuscript, Pol. Inst. of Bucharest, e-mail: George-Ciprian-Necula @PL1.FOX.CS.CMU.EDU, 1994.

[26] D. Nesmith, An application of klop's counterexample to a higher-order rewrite system, Draft Paper, 1989.

[27] T. Nipkow, Higher-order critical pairs, in: *Proc. Symp. on Logic in Computer Science* (*LICS*), Vol. 6 (1991) 342–349.

[28] M. Okada, Strong normalizability for the combined system of the types lambda calculus and an arbitrary convergent term rewrite system, in: *Proc. 20th Internat. Symp. on Symbolic and Algebraic Computation*, Portland, Oregon (1989).

[29] A. Poigné and J. Voss, On the implementation of abstract data types by programming language constructs, *J. Comput. System Sci.* **34** (1987) 340–376.

[30] V. van Oostrom, Developing developments, Draft, 1994.