POLYMORPHIC REWRITING CONSERVES ALGEBRAIC STRONG NORMALIZATION AND CONFLUENCE

Val Breazu-Tannen
and Jean Gallier

MS-CIS-89-27
LOGIC & COMPUTATION 06

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

May 1989

To appear in the proceedings of ICALP, Stresa, July 1989

Acknowledgements: This research was supported in part by ONR grants NOOO14-88-K-0634, NOOO14-88-K-0593, DARPA grant NOOO14-85-K-0018, NSF grants MCS-8219196-CER, IRI84-10413-AO2 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

Polymorphic Rewriting Conserves Algebraic Strong Normalization and Confluence

Val Breazu-Tannen[1] *Jean Gallier*[2]

Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St., Philadelphia, PA 19104, USA

Abstract. We study combinations of many-sorted algebraic term rewriting systems and polymorphic lambda term rewriting. Algebraic and lambda terms are mixed by adding the symbols of the algebraic signature to the polymorphic lambda calculus, as higher-order constants.

We show that if a many-sorted algebraic rewrite system R is strongly normalizing (terminating, noetherian), then $R + \beta + \eta +$ type-β + type-η rewriting of mixed terms is also strongly normalizing. We obtain this results using a technique which generalizes Girard's "candidats de reductibilité", introduced in the original proof of strong normalization for the polymorphic lambda calculus.

We also show that if a many-sorted algebraic rewrite system R has the Church-Rosser property (is confluent), then $R + \beta +$ type-β + type-η rewriting of mixed terms has the Church-Rosser property too. Combining the two results, we conclude that if R is canonical (complete) on algebraic terms, then $R + \beta +$ type-β + type-η is canonical on mixed terms.

η reduction does not commute with algebraic reduction, in general. However, using long η-normal forms, we show that if R is canonical then $R + \beta + \eta +$ type-β + type-η convertibility is still decidable.

To appear in the proceedings of ICALP, Stresa, July 1989

[1]Partially supported by ONR Grant NOOO14-88-K-0634 and by ARO Grant DAAG29–84–K–0061
[2]Partially supported by ONR Grant NOOO14-88-K-0593.

1 Introduction

From a very general point of view, this paper is about the interaction between "first-order computation" modeled by algebraic rewriting, and "higher-order polymorphic computation" modeled by reduction in the Girard-Reynolds polymorphic lambda calculus. Our results permit to conclude that this interaction is quite smooth and pleasant.

Changing the perspective, we regard algebraic rewrite systems as tools for the proof-theoretic analysis of algebraic equational theories, and we recall that such algebraic theories are used to model data type specifications [EM85]. Then, our results continue to confirm a thesis put forward in a series of papers [MR86, BM87, Bre88] , namely that *strongly normalizing type disciplines* interact nicely with algebraic data type specifications.

The preservation of the confluence of algebraic rewriting is a case in point. We show in this paper that the very powerful, impredicative, but strongly normalizing, polymorphic type discipline yields confluent rewriting when combined with confluent algebraic rewriting. In contrast, this fails for type disciplines which allow the type-checking of *fixed points*, as in lambda calculi with recursive types, in particular in the untyped lambda calculus. (A counterexample is furnished by Klop's result about the lambda calculus with surjective pairing; see [Bre88] for a simpler one.)

The first main result of this paper, (see section 4) states that combining a confluent many-sorted algebraic rewrite system with almost all kinds (except η) of polymorphic term reduction notions gives a system that, globally, is confluent. A comparison of such a result with the preservation of confluence results of [Toy87] and [Klo80] appears in [Bre88].

A brief summary of the technical setting for our result goes as follows. Given a many-sorted signature Σ, we construct *mixed* lambda terms with the sorts of Σ as constant "base" types and from the symbols in Σ seen, by currying, as higher-order constants. Then, given a set R of rewrite rules between algebraic Σ-terms, we show that if R is CR on algebraic Σ-terms, then $R + \beta +$ type-$\beta +$ type-η rewriting of mixed terms has the Church-Rosser property too. (Notice the absence of η; a counterexample appears in section 4.) An obvious, but important, feature of R-rewriting on mixed terms is that this is done such that the variables occurring in the algebraic rules can be instantiated with any mixed terms, as long as they are of the same "base" type as the variables they replace.

Our result and its proof are direct generalizations of the corresponding result for the simply typed lambda calculus presented in [Bre88]. However, since the publication of [Bre88], we have found an error in the proof of one of the lemmas (specifically lemma 2.2) used there for the confluence result. In this paper we correct the error, and generalize the statement of the lemma—from simply typed normal forms to arbitrary polymorphic terms (see theorem 3.5).

Our second main result is about preservation of strong normalization (SN). In the same setting as above, we show in section 6 that given a set R of rewrite rules between algebraic Σ-terms, if R is SN on algebraic Σ-terms, then $R + \beta + \eta +$ type-$\beta +$ type-η rewriting

of mixed terms is also SN (no problem with η here). This settles an open question posed in [Bre88], where some insight into the problem was also given.

Combinations of SN rewrite systems are notoriously impredictable. Toyama [Toy87] gives two SN algebraic rewrite systems whose direct sum is not SN. Results like ours in which SN is preserved in the combination (which is not even a direct sum, since application is shared) are therefore mathematically very interesting.

We prove our conservation of SN result by generalizing a technique due to Girard [Gir72], the method of candidates of reducibility. For the simple type discipline the idea of associating certain sets of strongly normalizing terms to types to facilitate a proof by induction that all terms are SN already appears in [Tai67] but the situation is much more complicated for the polymorphic lambda calculus. The idea that such techniques could be used for proving other results than strong normalization with respect to β-reduction apparently originated with Statman [Sta85]. (His unary syntactic logical relations are simply typed versions of the sets of generalized candidates.) This idea is taken further, and very well articulated by Mitchell [Mit86] where most of the ingredients of the generalization we give here appear except that it works for proving properties of *type-erasures* of polymorphic lambda terms, and not all such properties reflect back to typed terms. Tait also uses the type-erasing technique just for strong normalization [Tai75],[3] and the technical conditions we use in section 5 owe to both Tait and Mitchell. In order to accomodate *many-sorted* algebraic rewriting we use a generalization of Girard's original *typed* candidates.

Working independently from us, Dougherty also gives an answer to [Bre88]'s open question on SN preservation [Dou89]. His method works for any strongly normalizing untyped terms, using an analysis of the residuals of algebraic reduction on untyped lambda terms. However, the use of type- and therefore sort-erasure limits its applicability to one-sorted algebraic systems: indeed, it is easy to construct an SN many-sorted algebraic rewrite system which ceases to be SN when the sorts are identified.

Combining our two results, we obtain the following: if R is canonical (SN and CR) on algebraic terms, then R + β + type-β + type-η is canonical on mixed terms. Again, we should point out that even direct sums of canonical systems are not necessarily canonical, as was shown by Barendregt and Klop [Klo87].

The reader may wonder what happens with η-reduction. An example is given in section 4 which shows that η-reduction does not commute even with the simplest kind of algebraic reduction. We do not regard this as a significant fact since the computational interpretation of η-reduction is quite unclear. However, η, regarded as an *equational axiom*, may be useful when reasoning about programs. In view of this, we examine the problem of deciding R + β + η + type-β + type-η convertibility. We show in section 7, by using long η-normal forms, that if R is canonical then convertibility is decidable.

[3] Mitchell's results were obtained independently of Tait's.

2 Mixing algebra and polymorphic lambda calculus

This section is devoted to developing the notation for stating our results. We start with an arbitary many-sorted algebraic signature and define *mixed terms i.e.,* polymorphic lambda terms constructed with the symbols of the signature seen as higher-order constants. In the process, we give a new, simpler, notation for polymorphic terms. The motivation for departing from the style of recent presentations [Mit86, BC88] is that the notation they offer is too cluttered. We sketch a new notation which handles polymorphic lambda terms with almost the same ease as the usual notation handles simply typed lambda terms [Sta82]. This is very helpful to the intuition needed in proofs depending heavily on the combinatorics of terms as is demonstrated very well by Statman's work on the simply typed lambda calculus. This notation deserves a detailed development, but because of space limitations we shall do it elsewhere. We conclude the section with a precise statement of the main theorems we prove in the paper.

Let S be a set of *sorts* and Σ an S-sorted algebraic signature. Each function symbol $f \in \Sigma$ has an *arity*, which is a string $s_1 \cdots s_n \in S^*$, $n \geq 0$, and a *sort* $s \in S$ intending to symbolize a heterogenous operation which takes arguments of sorts (in order) $s_1, \ldots, s_n$ and returns a result of sort s.

Type expressions (types) are defined by

$$\sigma ::= s \mid t \mid \sigma \rightarrow \sigma \mid \forall t.\, \sigma$$

where s ranges over S and t ranges over an infinite set $\mathcal{V}$ of *type variables.* Therefore, the "base" types are exactly the sorts of the signature. Free and bound variables are defined in the usual way. We denote by $FTV(\sigma)$ the set of type variables which are free in σ. We will identify the type expressions which differ only in the name of the bound variables. The set of type expressions will be denoted by $\mathcal{T}$.

A type substitution is a map $\theta : \mathcal{V} \longrightarrow \mathcal{T}$. The result of applying θ to σ is denoted $\sigma[\theta]$ and, if θ is the identity everywhere except $\theta(t) = \tau$, $\sigma[\tau/t]$.

Let $\mathcal{X}$ be an infinite set of *(term) variables.* A *type assignment* is a partial function $\Delta : \mathcal{X} \longrightarrow \mathcal{T}$ with *finite* domain. Alternatively, we will also regard type assignments as finite sets of pairs $x{:}\sigma$ such that no x occurs twice. We write $\Delta, x{:}\sigma$ for $\Delta \cup \{x{:}\sigma\}$ and, by convention, the use of this notation implies that $x \notin dom\Delta$. The *empty* type assignment is usually omitted. A *declaration* is a pair consisting of a type assignement and a type, written $\Delta \vdash \sigma$. Terms, together with their declarations, are defined inductively as follows

Variables. For any Δ and any $x{:}\sigma$ in Δ, the triple $\langle \Delta, x, \sigma \rangle$ is a term of declaration $\Delta \vdash \sigma$.

Constants. For any $f \in \Sigma$ of arity $s_1 \cdots s_n$ and sort s, and for any Δ, the triple $\langle \Delta, f, \sigma \rangle$ where $\sigma \stackrel{\text{def}}{=} s_1 \rightarrow \cdots \rightarrow s_n \rightarrow s$ is a term of declaration $\Delta \vdash \sigma$.

Application. If M is term of declaration $\Delta \vdash \sigma \to \tau$ and N is a term of declaration $\Delta \vdash \sigma$ then MN is a term of declaration $\Delta \vdash \tau$.

Abstraction. If M is a term of declaration $\Delta, x{:}\sigma \vdash \tau$ then $\lambda x{:}\sigma.M$ is a term of declaration $\Delta \vdash \sigma \to \tau$.

Type application. If M is a term of declaration $\Delta \vdash \forall t.\, \sigma$ then for any $\tau \in \mathcal{T}$, $M\tau$ is a term of declaration $\Delta \vdash \sigma[\tau/t]$.

Type abstraction. If M is a term of declaration $\Delta \vdash \sigma$ and $t \notin FTV(ran\Delta)$, then $\lambda t.\, M$ is a term of declaration $\Delta \vdash \forall t.\, \sigma$.

For a term M of declaration $\Delta \vdash \sigma$ we define the type of M to be σ and we write $M : \sigma$.

We denote by Λ the set of all terms.

Free and bound variables are defined as usual. We denote by $FV(M)$ the set of free variables of M. Clearly if M has declaration $\Delta \vdash \sigma$ then $FV(M) \subseteq dom\Delta$. If $x{:}\tau \in \Delta$, we say that $x{:}\sigma$ is *declared* in M. A term can have declared variables which do not belong to $FV(M)$. The free and bound type variables of a term, and substitution of types for type variables in terms are defined such that free occurrences in type assignments and types count too. (We denote by $FTV(M)$ the set of free type variables of M.) For example, $FTV(\langle\Delta, x, \sigma\rangle) \stackrel{\text{def}}{=} FTV(ran\Delta) \cup FTV(\sigma)$. Again we identify the terms which differ only in the name of the bound variables and bound type variables.

In view of their inductive definition, we will regard terms as trees. Consequently, we can define *subterms* (as subtrees), occurrences of subterms in a term, and *replacement* of a subterm by another term.

Note that if a variable or a constant of declaration $\Delta \vdash \sigma$ occurs as a subterm of a term of declaration $\Delta' \vdash \tau$ then $\Delta' \subseteq \Delta$. Given a term N of declaration $\Delta' \vdash \tau$ and a type assignment Δ'' such that $\Delta' \subseteq \Delta''$, there is a canonical *expansion* of N to a term N' of declaration $\Delta'' \vdash \tau$ obtained by adding $\Delta'' \setminus \Delta'$ to the declarations of the variables and constants of N . (This may require some renaming of the bound variables.)

Substitution of terms for variables in terms can be defined via replacement of subterms. Let Δ and Δ' be two type assignments. A *substitution from* Δ *to* Δ' written $\varphi : \Delta \longrightarrow \Delta'$ is a map that associates to any $x \in dom\Delta$ a term $\varphi(x)$ of declaration $\Delta' \vdash \Delta(x)$. Let M be a term of declaration $\Delta \vdash \sigma$ and $\varphi : \Delta \longrightarrow \Delta'$ a substitution. Define the result of applying φ to M as the term of declaration $\Delta' \vdash \sigma$ obtained by replacing all the occurrences in M of subterms of the form $\langle\Delta'', x, \sigma\rangle$ where $x \in dom\Delta$ with corresponding expansions of $\varphi(x)$ from Δ' to $\Delta' \cup (\Delta'' \setminus \Delta)$ where the union is assumed disjoint (some renaming of bound variables may be necessary).

Notation for substitution: $M[\varphi]$ and $M[N/x]$ if φ is the identity everywhere except $\varphi(x) = N$.

This introduction to the notation is, by necessity, informal. In particular, many details and many tedious proofs are hidden behind the casual "we identify types and terms which differ

only in the name of the bound variables". But the rigorous treatment is similar to that of other lambda calculi and will be given elsewhere. The point of this notation is that once all these basic definitions are made precise, declarations can almost always be left implicit, as is the case with types in the simply typed lambda calculus. Taking advantage of this, except for the the basic definition of terms we just gave, we will not actually need to use the triple notation for variables and constants. This is well illustrated, for example, by the definition of the usual *notions of reduction*:

(β-reduction) $M \xrightarrow{\beta} N$ iff
N is obtained from M by replacing a subterm of the form $(\lambda x{:}\sigma.\ X)Y$ with $X[Y/x]$.

(η-reduction) $M \xrightarrow{\eta} N$ iff
N is obtained from M by replacing a subterm of the form $\lambda x{:}\sigma.\ Zx$ with Z, where $x \notin FV(Z)$.

(type-β reduction) $M \xrightarrow{\mathcal{T}\beta} N$ iff
N is obtained from M by replacing a subterm of the form $(\lambda t.\ X)\tau$ with $X[\tau/t]$.

(type-η reduction) $M \xrightarrow{\mathcal{T}\eta} N$ iff
N is obtained from M by replacing a subterm of the form $\lambda t.\ Zt$ with Z, where $t \notin FTV(Z)$.

Let

$$\xrightarrow{\lambda^{\forall}} \stackrel{\text{def}}{=} \xrightarrow{\beta} \cup \xrightarrow{\eta} \cup \xrightarrow{\mathcal{T}\beta} \cup \xrightarrow{\mathcal{T}\eta},$$

and we will also need

$$\xrightarrow{\lambda^{-}} \stackrel{\text{def}}{=} \xrightarrow{\beta} \cup \xrightarrow{\mathcal{T}\beta} \cup \xrightarrow{\mathcal{T}\eta}.$$

Next we will introduce algebraic terms and rewriting. There is a well-known transformation, known as *currying* that maps algebraic Σ-terms into Λ. This transformation is an injection. In view of that, we choose to talk directly about curried algebraic terms and define algebraic rewriting on them.

A declaration is *algebraic* iff all the types occurring in it are sorts. Among polymorphic terms, *algebraic* terms are defined inductively by

- Any variable (term) of algebraic declaration is an algebraic term.
- If f is a constant (term) of declaration $\Delta \vdash s_1 \to \cdots \to s_n \to s$, $ran\Delta$ consists only of sorts, and $A_1 : s_1,\ \ldots\ A_n : s_n$ are algebraic terms, then $f\,A_1 \cdots A_n$ is an algebraic term.

As intended, it follows that any algebraic term has an algebraic declaration.

An algebraic rewrite rule is a pair r of algebraic terms, written $r \equiv A \to A'$, where A, A' have the same declaration, $FV(A') \subseteq FV(A)$, and A is not a variable.[4] Such a rule defines a reduction relation on *all* terms not only the algebraic ones:

$$M \xrightarrow{r} N \quad \text{iff}$$

there exists a substitution φ such that N is obtained from M by replacing an occurrence of $A[\varphi]$ as a subterm with $A'[\varphi]$.

Lemma 2.1 *If B is algebraic and $B \xrightarrow{r} Z$ then Z is algebraic.*

Thus, we can talk about algebraic rewriting on algebraic terms. It is easy to see that currying establishes the expected relation between many-sorted algebraic rewriting of Σ-terms [MG85] and our definition of algebraic rewriting. Indeed, for any many-sorted Σ-rewrite rule $m \equiv p \to p'$ and any many-sorted Σ-terms q, q'

$$q \xrightarrow{m} q' \quad \text{iff} \quad curry(q) \xrightarrow{c(m)} curry(q')$$

where $c(m) \equiv curry(p) \to curry(p')$.

Let R be a set of algebraic rewrite rules. Define the following notions of reduction on terms:

$$\xrightarrow{R} \stackrel{\text{def}}{=} \bigcup_{r \in R} \xrightarrow{r} \ , \quad \xrightarrow{\lambda^\forall R} \stackrel{\text{def}}{=} \xrightarrow{\lambda^\forall} \cup \xrightarrow{R} \ , \quad \xrightarrow{\lambda^- R} \stackrel{\text{def}}{=} \xrightarrow{\lambda^-} \cup \xrightarrow{R} \ .$$

For any of these notions of reduction we will denote by $\twoheadrightarrow$ the reflexive-transitive closure of $\longrightarrow$.

It is well-known that both $\lambda^\forall$ and λ^- reduction are canonical (*i.e.*, strongly normalizing and confluent) on all terms. In fact, the generalized method of candidates presented in section 5 can be used to prove this (see theorem 5.6). We denote by $\lambda^\forall nf(X)$ and $\lambda^- nf(X)$ the corresponding normal forms of X.

Finally, we state precisely our main results:

(Conservation of Strong Normalization.) If $\xrightarrow{R}$ is strongly normalizing on algebraic terms then $\xrightarrow{\lambda^\forall R}$ is strongly normalizing on all terms.

(Conservation of Confluence.) If $\xrightarrow{R}$ is confluent on algebraic terms then $\xrightarrow{\lambda^- R}$ is confluent on all terms.

[4] The results hold also if we have *degenerate* rules $z \longrightarrow A'$ where $FV(A') = \emptyset$ but their effect can be simulated with normal rules anyway.

3 Algebraic rewriting of higher-order terms

In this section, we show that the properties that algebraic reduction has on algebraic terms transfer to algebraic reduction on arbitrary terms.

Theorem 3.1
If $\xrightarrow{R}$ is strongly normalizing on algebraic terms then $\xrightarrow{R}$ is strongly normalizing on all terms.

Proof Sketch. We proceed by induction on the size of terms. The only case in which the induction hypothesis does not immediately apply is the case of an application term. Let $M \equiv H\,T_1 \cdots T_k$ be such that H is not an application and the T_i's are terms or types. Suppose there is an infinite R-reduction sequence out of M. If H is an abstraction, a type abstraction, a variable, or a constant which takes $> k$ arguments (*i.e.,* the length of its arity is $> k$), then each reduction in the sequence is inside some term among the H and T_i's, and since there are only finitely many such terms there must be an infinite reduction sequence from one of them, contradicting the induction hypothesis. (This kind or argument based on the pigeonhole principle will be invoked again.) The only complex case is when H is a constant which takes exactly k arguments, and in this case the type of M is a sort. We need to analyze algebraic reductions on such terms, in particular to separate "trunk" (close to the "root" of terms) algebraic reductions from other reductions.

An *algebraic trunk decomposition* of a term M consists of an algebraic term A (the "trunk") and a substitution φ such that $M \equiv A[\varphi]$, variables occur in A only once, and for all $x \in FV(A)$ the term $\varphi(x)$ has the form $H\,T_1 \cdots T_k$ where H is an abstraction, a type abstraction, or a variable and $T_1, \ldots, T_k$ are terms or types. Clearly the type of any term that has an algebraic trunk decomposition must be a sort, but in fact that's all it takes:

Lemma 3.2
Any term M whose type is a sort has an algebraic trunk decomposition $M \equiv A[\varphi]$. Moreover, this decomposition is unique up to renaming the free variables of A.

With this, the last case in the proof of the theorem follows from

Lemma 3.3
Let $\xrightarrow{R}$ be SN on algebraic terms. Let $A[\varphi]$ be an algebraic trunk decomposition. If for each $x \in FV(A)$ $\xrightarrow{R}$ is SN on $\varphi(x)$ then $\xrightarrow{R}$ is SN on $A[\varphi]$.

Before we sketch the proof of this lemma, we give a motivating discussion. For an algebraic trunk decomposition $M \equiv A[\varphi]$, an algebraic redex must occur either entirely within one of the subterms $\varphi(x)$, or "essentially" within the trunk part. More precisely, we say that

$A[\varphi] \xrightarrow{R} A'[\varphi']$ is an *algebraic trunk reduction step* if the R-redex is *not* a subterm of one of the $\varphi(x)$'s. It is easy to see that if $A[\varphi] \xrightarrow{R}\!\!\!\!\!\rightarrow A'[\varphi']$ then for each $x' \in FV(A')$ there is an $x \in FV(A)$ such that $\varphi(x) \xrightarrow{R}\!\!\!\!\!\rightarrow \varphi'(x')$. However, separating the trunk reductions is somewhat subtle because algebraic rewrite rules may be non-linear, or may erase some of their arguments. In particular, the following example shows that $A[\varphi] \xrightarrow{R}\!\!\!\!\!\rightarrow A'[\varphi']$ does not necessarily imply $A \xrightarrow{R}\!\!\!\!\!\rightarrow A'$. (We shall denote algebraic trunk reductions by $\xrightarrow{tR}$ and algebraic reductions in the non-trunk part by $\xrightarrow{ntR}$.)

Example 3.4
Let $R = \{fxx \longrightarrow gxxx,\ a \longrightarrow b,\ b \longrightarrow c\}$, *and* $M = f(Fa)(Fb)$, *where* F *is a higher-order variable. While we have the rewrite sequence*

$$\begin{array}{rl} M & \xrightarrow{ntR} f(Fb)(Fb) \\ & \xrightarrow{tR} g(Fb)(Fb)(Fb) \\ & \xrightarrow{ntR} g(Fb)(Fc)(Fb), \end{array}$$

we do not have that $fx_1x_2 \xrightarrow{R}\!\!\!\!\!\rightarrow gy_1y_2y_3$ *even if we rename the* y*'s. However, note that* $fzz \xrightarrow{R}\!\!\!\!\!\rightarrow gzzz$.

Sketch of Proof for Lemma 3.3. Suppose there is an infinite R-reduction sequence out of $A[\varphi]$. If this sequence has only finitely many trunk reduction steps, let $A'[\varphi']$ be term obtained after the last trunk step. By a pigeonhole principle kind of argument, some $\varphi'(x')$ is not SN hence some $\varphi(x)$ is not SN, contradiction. If this sequence has infinitely many trunk reduction steps then we get an infinite sequence of R-reductions on *algebraic terms* (hence a contradiction) from the following observation: if $A_1[\varphi_1] \xrightarrow{ntR}\!\!\!\!\!\rightarrow A_2[\varphi_2] \xrightarrow{tR} A_3[\varphi_3]$ then $A_1[\zeta] \xrightarrow{R} A_3[\zeta]$ where ζ is the substitution that takes all variables of a sort into some fixed variable of that sort.

We now turn to the confluence result.

Theorem 3.5
If $\xrightarrow{R}$ *is confluent on algebraic terms then* $\xrightarrow{R}$ *is confluent on all terms.*

Proof Sketch. We show by induction on the size of M that R-confluence holds from M. Again, the only case in which the induction hypothesis does not immediately apply is the case of an application term. For application terms $M \equiv H\,T_1 \cdots T_k$ such that H is an abstraction, a type abstraction, a variable, or a constant which takes $> k$ arguments, each R-reduction out of M is completely inside H or inside one of the T_i's. By induction hypothesis, confluence holds from each of these, thus confluence holds from M.

This leaves only case when H is a constant which takes exactly k arguments. Note that the example 3.4 also shows that nontrunk rewrite steps and trunk rewrite steps cannot always be permuted. The problem is caused by non-linear rewrite rules. Part of the proof of lemma 2.2 (page 85) of [Bre88] is invalidated by this problem. However, the argument can be repaired, but the technical details are surprisingly involved. The key is to realize that on terms of type sort, $\overset{R}{\longrightarrow\!\!\!\!\!\rightarrow}$ is the transitive reflexive closure of $\overset{ntR}{\longrightarrow\!\!\!\!\!\rightarrow} \circ \overset{tR}{\longrightarrow\!\!\!\!\!\rightarrow}$. then, with the observation that if for each $x \in FV(A)$ $\overset{R}{\longrightarrow}$ is CR on $\varphi(x)$ and if $A[\varphi] \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} A'[\varphi']$ then for each $x' \in FV(A')$ $\overset{R}{\longrightarrow}$ is CR on $\varphi'(x')$, the R-confluence needed in the last case of the proof of the theorem follows from

Lemma 3.6
Let $\overset{R}{\longrightarrow}$ be CR on algebraic terms. Let $A[\varphi]$ be an algebraic trunk decomposition. If for each $x \in FV(A)$ $\overset{R}{\longrightarrow}$ is CR on $\varphi(x)$ and if $N \overset{tR}{\longleftarrow\!\!\!\!\!\leftarrow} N' \overset{ntR}{\longleftarrow\!\!\!\!\!\leftarrow} A[\varphi] \overset{ntR}{\longrightarrow\!\!\!\!\!\rightarrow} P' \overset{tR}{\longrightarrow\!\!\!\!\!\rightarrow} P$ then there is a Q such that $N \overset{ntR}{\longrightarrow\!\!\!\!\!\rightarrow} N'' \overset{tR}{\longrightarrow\!\!\!\!\!\rightarrow} Q \overset{tR}{\longleftarrow\!\!\!\!\!\leftarrow} P'' \overset{ntR}{\longleftarrow\!\!\!\!\!\leftarrow} P$.

The proof of this lemma is inspired by some key ideas of Toyama [Toy87] and is omitted here.

4 Conservation of the Church-Rosser property

Let R be a set of algebraic rewrite rules.

Lemma 4.1
Let $X, Y \in \Lambda$ and $r \in R$. If $X \overset{r}{\longrightarrow} Y$ then $\lambda^- nf(X) \overset{r}{\longrightarrow\!\!\!\!\!\rightarrow} \lambda^- nf(Y)$.

The proof is essentially the same as that of lemma 2.1 in [Bre88] with the minor addition that one checks that the form of certain subterms is also preserved by $\mathcal{T}\beta$ and $\mathcal{T}\eta$ reduction. This is where the proof breaks down for η. This lemma is false if we replace λ^- with $\lambda^\forall$ as can be seen from the simple example $r \equiv fx \longrightarrow a$ and $X \equiv \lambda z.\, fz$.

Theorem 4.2
If R-reduction is confluent on algebraic terms then $\lambda^- R$-reduction is confluent on all terms.

Proof. (The same as the proof of theorem 2.3 in [Bre88].) Suppose that $Y \overset{\lambda^- R}{\longleftarrow\!\!\!\!\!\leftarrow} X \overset{\lambda^- R}{\longrightarrow\!\!\!\!\!\rightarrow} Z$. By taking everything to λ^--normal form, we obtain from lemma 4.1 that $\lambda^- nf(Y) \overset{R}{\longleftarrow\!\!\!\!\!\leftarrow} \lambda^- nf(X) \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} \lambda^- nf(Z)$. Then, by theorem 3.5, there exists a W such that $\lambda^- nf(Y) \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} W \overset{R}{\longleftarrow\!\!\!\!\!\leftarrow} \lambda^- nf(Z)$. Thus $Y \overset{\lambda^-}{\longrightarrow\!\!\!\!\!\rightarrow} \lambda^- nf(Y) \overset{R}{\longrightarrow\!\!\!\!\!\rightarrow} W \overset{R}{\longleftarrow\!\!\!\!\!\leftarrow} \lambda^- nf(Z) \overset{\lambda^-}{\longleftarrow\!\!\!\!\!\leftarrow} Z$.

5 Generalized candidates of reducibility

We give here a brief development of our generalization of Girard's typed candidates of reducibility technique. We also state that the technique can be applied to obtain some well-known SN and CR results, in addition to Girard's original SN result. We begin with the defininition of the generalized candidates. For the intuition behind the definition the reader may consult [GLT89]. The technical use of the candidates should be evident from the proof of theorem 5.1.

Let P be a property of terms. For each type σ, let P_σ be the set of all terms of type σ which have the property P. A *P-candidate* is a pair $\langle \sigma, C \rangle$ where $\sigma \in \mathcal{T}$ and C is a set of terms of type σ having the property P (*i.e.*, $C \subseteq P_\sigma$) such that the following hold.

(Cand 1) If x is a variable, $T_1, \ldots, T_k$ $(k \geq 0)$ are either terms which have the property P or types, and $x\,T_1 \cdots T_k$ has type σ, then $x\,T_1 \cdots T_k \in C$.

(Cand 2) If $f \in \Sigma$ is a constant, $T_1, \ldots, T_k$ $(k \geq 0)$ are either terms which have the property P or types, and $f\,T_1 \cdots T_k$ has type σ, then $f\,T_1 \cdots T_k \in C$. (Note that the length of the arity of f may differ from k.)

(Cand 3) If M, N are terms which have the property P, $T_1, \ldots, T_k$ $(k \geq 0)$ are either terms which have the property P or types, $x{:}\tau$ is declared in M, and $M[N/x]\,T_1 \cdots T_k \in C$ then $(\lambda x{:}\tau.\, M)\,N\,T_1 \cdots T_k \in C$.

(Cand 4) If M is a term which has the property P, $T_1, \ldots, T_k$ $n \geq 0$ are either terms which have the property P or types, τ is a type, and $M[\tau/t]\,T_1 \cdots T_k \in C$ then $(\lambda t.M)\,\tau\,T_1 \cdots T_k \in C$.

The property P is *candidate-closed* iff the following hold.

(Clo 1a) If Mx (where x is a variable) has property P, then M has property P.

(Clo 1b) If Mt (where t is a type variable) has property P, then M has property P.

(Clo 2) For any type σ, the pair $\langle \sigma, P_\sigma \rangle$ is itself a P-candidate.

Theorem 5.1
If P is candidate-closed then all terms have property P.

Proof Sketch. Assume P is candidate-closed.

A *candidate assignment* is map γ that associates to each type variable a P-candidate. Taking the first projection, we can regard any candidate assignment also as a type substitution, and write $\sigma[\gamma]$ for any type σ.

We associate to each type σ and each candidate assignment γ a pair consisting of a type and a set of terms, denoted $[\![\sigma]\!]\gamma$, as follows

$$\begin{array}{rcl}
[\![s]\!]\gamma & \stackrel{\text{def}}{=} & \langle s, P_s\rangle \\
[\![t]\!]\gamma & \stackrel{\text{def}}{=} & \gamma(t) \\
[\![\sigma\to\tau]\!]\gamma & \stackrel{\text{def}}{=} & \langle \sigma[\gamma]\to\tau[\gamma]\,,\ \{M \mid \forall N,\ N\in[\![\sigma]\!]\gamma \implies MN\in[\![\tau]\!]\gamma\}\rangle \\
[\![\forall t.\,\sigma]\!]\gamma & \stackrel{\text{def}}{=} & \langle \forall t.\,\sigma[\gamma]\,,\ \{M \mid \forall\langle\tau,C\rangle\,\text{P}-\text{cand.},\ M\tau\in[\![\sigma]\!]\gamma\{t:=\langle\tau,C\rangle\}\ \}\rangle
\end{array}$$

$$\text{where}\quad \gamma\{t:=\langle\tau,C\rangle\}(t') \stackrel{\text{def}}{=} \begin{cases} \langle\tau,C\rangle & t'=t \\ \gamma(t') & t'\neq t \end{cases}$$

Lemma 5.2
$[\![\sigma]\!]\gamma$ is a P-candidate of type $\sigma[\gamma]$

All this is then used to show that any term belongs to some P-candidate, and thus has the property P. One uses induction on terms, strengthening the induction hypothesis as follows.

Lemma 5.3
For any term M of declaration $\Delta\vdash\sigma$, for any candidate assignment γ, for every substitution $\varphi:\Delta\longrightarrow\Delta[\gamma]$ such that $\forall x\in dom\Delta$, $\varphi(x)\in[\![\Delta(x)[\gamma]]\!]\gamma$, we have $M[\gamma][\varphi]\in[\![\sigma]\!]\gamma$.

The theorem now follows by applying the previous lemma to $\gamma(t)\stackrel{\text{def}}{=}\langle t,P_t\rangle$ and $\varphi(x)\stackrel{\text{def}}{=}x$. We give without proof some applications. While all these results are certainly well-known, apparently the Church-Rosser results for polymorphic terms have not been proved by the "candidates" method before (but this path started in [Sta85, Mit86]).

Theorem 5.4 (Girard) *" M is $\xrightarrow{\beta T\beta}$-strongly normalizing "*
is a candidate-closed property of terms $M\in\Lambda$.

Theorem 5.5 (Girard) *" $\xrightarrow{\beta T\beta}$-confluence holds from M "*
is a candidate-closed property of terms $M\in\Lambda$.

Theorem 5.6
The following are also candidate-closed properties of terms $M\in\Lambda$:

- *" M is $\xrightarrow{\lambda^\forall}$-strongly normalizing "*
- *" $\xrightarrow{\lambda^\forall}$-confluence holds from M "*
- *" M is $\xrightarrow{\lambda^-}$-strongly normalizing "*
- *" $\xrightarrow{\lambda^-}$-confluence holds from M "*

6 Conservation of strong normalization

Let R be a set of algebraic rewrite rules such that $\xrightarrow{R}$ is strongly normalizing on algebraic terms. In view of theorem 5.1, the desired result follows from

Theorem 6.1 *" M is $\xrightarrow{\lambda^\forall R}$-strongly normalizing "*
is a candidate-closed property of terms $M \in \Lambda$.

Proof Sketch. (Clo 1a) and (Clo 1b) are immediate. For (Clo 2) we need to check that the set of strongly normalizing terms of a certain type satisfies (Cand 1)–(Cand 4). (Cand 1) is immediate by the pigeonhole principle kind of argument (see the proof of theorem 3.1). Checking (Cand 3) is a bit of work but the presence of algebraic rules makes no difference compared to theorems 5.4 and 5.6 so we choose to omit it due to space limitations. Checking (Cand 4) is an easier version of checking (Cand 3). The really new situation appears in checking (Cand 2).

Suppose that $N_1 \cdots N_k$ are all $\xrightarrow{\lambda^\forall R}$-strongly normalizing and that there is an infinite reduction sequence from $M \equiv f\, N_1 \cdots N_k$. Let the length of the arity of f be n. Since M type-checks $k \leq n$. If $k < n$ the pigeonhole principle kind of argument applies.

If $k = n$ then the type of M and that of all the terms in the reduction sequence is a sort, so we can find algebraic trunk decompositions for them. From here we distinguish two cases.

Case 1. The reduction sequence out of M contains only finitely many algebraic trunk reduction steps.

Let then $M' \equiv A'[\varphi']$ be the term in the sequence obtained through the last algebraic trunk reduction step. Then, any further reduction step in the sequence is non-trunk and therefore is inside one of the $\varphi'(x')$, $x' \in FV(A')$. By a pigeonhole principle kind of argument, one of these is not strongly normalizing. Since we can also show

Lemma 6.2
Let $A[\varphi]$ be an algebraic trunk decomposition. If $A[\varphi] \xrightarrow{\lambda^\forall R}\!\!\!\!\rightarrow A'[\varphi']$ then for any $x' \in FV(A')$ there exists an $x \in FV(A)$ and a subterm N' of $A'[\varphi']$ such that $\varphi'(x')$ is a subterm of N' and $\varphi(x) \xrightarrow{\lambda^\forall R}\!\!\!\!\rightarrow N'$.

It follows that one of the $\varphi(x)$, $x \in FV(A)$ is not strongly normalizing. Since each of these is a subterm of one of the N_i's we have a contradiction again.

Case 2. The reduction sequence out of M contains infinitely many algebraic trunk reduction steps.

In this case the idea is to take all the terms in the sequence to $\lambda^\forall$-normal form but this does not quite work because of the bad interaction between η and R. Instead we use the following:

A *long normal form* is (recursively) a term of the form $\lambda\vec{v}.\, h\, Z_1 \cdots Z_n$ where h is a variable or a constant, each Z_i is a long normal form, and the type of $h\, Z_1 \cdots Z_n$ is either a sort, or a type variable, or of the form $\forall t.\, \sigma$. While such a term is in general *not* in η-normal form, the name is justified by the fact that any term, X, is $\lambda^{\forall}$-convertible to a *unique* long normal form, $lnf(X)$; to effectively obtain it, take the term to λ^{-}-normal form and then perform (if needed) some η-expansions. With this, we have a result very similar to lemma 4.1 (the proof is also similar), and we strengthen it for algebraic trunk reduction steps:

Lemma 6.3
Let $X, Y \in \Lambda$ and $r \in R$. If $X \xrightarrow{r} Y$ then $lnf(X) \xrightarrow{r}\!\!\!\!\rightarrow lnf(Y)$. Moreover, if $X \xrightarrow{r} Y$ is actually an algebraic trunk reduction step then $lnf(X) \xrightarrow{r} lnf(Y)$.

Now convert all the terms of the infinite reduction sequence out of M to long normal form. Since there are infinitely many algebraic trunk steps, the result will be an infinite sequence of R-reductions. By theorem 3.1, this is impossible.

7 Deciding convertibility in the presence of η

In view of the counterexample involving η prsented in section 4 there are algebraic rewrite systems R which are canonical such that $\lambda^{\forall}R$ is not confluent, and thus not canonical. Nonetheless, lemma 6.3 provides a satisfactory solution:

Theorem 7.1
If R is canonical on algebraic terms then $\lambda^{\forall}R$ convertibility is decidable.

Proof. Since R is canonical on algebraic terms it also canonical on all terms, by theorems 3.1 and 3.5. Let $Rnf(X)$ be the R-normal form of a term X.

The algorithm is the following: to decide if M and N are convertible test if $Rnf(lnf(M)) \equiv Rnf(lnf(N))$.

Indeed, if M, N are convertible to each other by a chain of $\lambda^{\forall}R$ conversion steps then take all the terms in this chain to long normal form. By lemma 6.3 $lnf(M)$ and $lnf(N)$ are R-convertible so their R-normal forms coincide. The converse is trivial.

8 Directions for Further Research

Of course, one would also like to know what to do in the absence of an equivalent canonical rewrite system. We conjecture that the proof-theoretic reduction from simply typed

theories with algebraic axioms to algebraic theories, given in [Bre88], can be generalized to polymorphic theories.

Our results show that some important properties of algebraic systems are preserved when algebraic rewriting and polymorphic lambda-term rewriting are mixed. As applications to the results of this paper, we intend to investigate higher-order unification modulo an algebraic theory. For the simply-typed lambda calculus, we conjecture that adding the lazy paramodulation rule investigated in [GS89a] to the set of higher-order transformations investigated in [GS89b] yields a complete set of transformations for higher-order E-unification. Such a result would have several applications in automated theorem proving. We also intend to investigate the possibility of extending Knuth-Bendix completion procedures to polymorphic theories with algebraic axioms.

Another direction of investigation is to consider more complicated type disciplines, such as that of the Calculus of Constructions [CH88].

More generally, we feel that the results of this paper are only a first step towards extending the important field of term rewriting systems to include higher-order rewriting. One of our main goals is to provide rigorous methods for understanding higher-order functional and logic programming. In particular, one is interested in rules which describe the behaviour of higher-order operations (such as *maplist*, for example). In any case, a lot of care will be needed with higher-order rules because, for example, fixed points are also described this way: $YF = F(YF)$.

References

[BC88] V. Breazu-Tannen and T. Coquand. Extensional models for polymorphism. *Theoretical Computer Science*, 85–114, 1988.

[BM87] V. Breazu-Tannen and A. R. Meyer. Computable values can be classical. In *Proceedings of the 14th Symposium on Principles of Programming Languages*, pages 238–245, ACM, January 1987.

[Bre88] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings of the Symposium on Logic in Computer Science*, pages 82–90, IEEE, July 1988.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Control*, 76:95–120, 1988.

[Dou89] D. Dougherty. Adding algebraic rewriting to the untyped lambda calculus. Manuscript, Wesleyan University. March 1989.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of algebraic specification 1: equations and initial semantics.* Springer-Verlag, 1985.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieure.* PhD thesis, Université Paris VII, 1972.

[GLT89] J. Y. Girard, Y. Lafont, and P. Taylor. *Typed lambda calculus.* Cambridge University Press, 1989. Forthcoming.

[GS89a] J. Gallier and W. Snyder. Complete sets of transformations for general *E*-Unification. *Theoretical Computer Science*, 1989. To appear.

[GS89b] J. Gallier and W. Snyder. Higher-order unification revisited: complete sets of transformations. *Journal of Symbolic Computation*, 1989. To appear.

[Klo80] J. W. Klop. *Combinatory reduction systems.* Tract 129, Mathematical Center, Amsterdam, 1980.

[Klo87] J. W. Klop. Term rewriting systems: a tutorial. *Bull. EATCS*, 32:143–182, June 1987.

[MG85] J. Meseguer and J. Goguen. *Deduction with many-sorted rewrite.* Technical Report 42, CSLI, Stanford, 1985.

[Mit86] J. C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In *Proceedings of the LISP and Functional Programming Conference*, pages 308–319, ACM, New York, August 1986.

[MR86] A. R. Meyer and M. B. Reinhold. 'Type' is not a type: preliminary report. In *Conf. Record Thirteenth Ann. Symp. Principles of Programming Languages*, pages 287–295, ACM, January 1986.

[Sta82] R. Statman. Completeness, invariance and λ-definability. *Journal of Symbolic Logic*, 47:17–26, 1982.

[Sta85] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

[Tai67] W. W. Tait. Intensional interpretations of functionals of finite type i. *Journal of Symbolic Logic*, 32:198–212, 1967.

[Tai75] W. W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Proceedings of the Logic Colloqium '73*, pages 240–251, *Lecture Notes in Mathematics*, Vol. 453, Springer-Verlag, 1975.

[Toy87] Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, January 1987.