Polymorphic Rewriting Conserves Algebraic Strong Normalization

Val Tannen[*] Jean H. Gallier[†]

[*]University of Pennsylvania, val@cis.upenn.edu

[†]University of Pennsylvania, jean@cis.upenn.edu

Polymorphic Rewriting Conserves Algebraic Strong Normalization

MS-CIS-90-36
LOGIC & COMPUTATION 19

Val Breazu-Tannen
Jean Gallier

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

June 1990

Polymorphic Rewriting Conserves Algebraic Strong Normalization [1]

Val Breazu-Tannen[2] *Jean Gallier*[3]

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. The result is obtained using a technique which generalizes Girard's "candidats de reductibilité", introduced in the original proof of strong normalization for the polymorphic lambda calculus.

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, the results in this paper together with the

[1]To appear in **Theoretical Computer Science**.
[2]Partially supported by ONR Grant NOOO14-88-K-0634 and by ARO Grant DAAG29-84-K-0061
[3]Partially supported by ONR Grant NOOO14-88-K-0593.

results in [BG89] 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.

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. 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 main result is about preservation of strong normalization (SN). In the setting described above, we show in section 5 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-β + type-η rewriting of mixed terms is also SN.

Combinations of SN rewrite systems are notoriously impredictable. Toyama [Toy87] gives two SN algebraic rewrite systems whose direct sum is not SN (see example 1.1). 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.

Combining the main result of this paper with one in [BG89], we obtain the following: if R is canonical (SN and CR) on algebraic terms, then $R + \beta$ + 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 (SN may still fail), as was shown by Barendregt and Klop (see the survey [Klo87]).

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] in the context of the simply typed lambda calculus. (His unary syntactic logical relations are simply typed versions of the sets of generalized candidates.) This idea is taken further, to the Girard-Reynolds polymorphic lambda calculus, 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],[4] and the technical conditions we use in section 4 owe to both Tait and Mitchell. In order to accomodate *many-sorted* algebraic rewriting we use a generalization of Girard's original *typed* candidates.

The main result of this paper settles an open question posed in [Bre88], where some insight into the problem was also given. Several related results have also been obtained recently.

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

Okada [Oka89] proves conservation of SN by the addition of simply typed β-reduction, gives a short sketch of an extension to polymorphic terms and type-β reduction, and claims further extensions to η-reduction. Dougherty [Dou89] proves conservation of SN when adding algebraic rewriting to certain SN terms of the untyped lambda calculus, using an analysis of the residuals of algebraic reduction on untyped lambda terms. Barbanera [Bar89] proves conservation of SN when adding algebraic rewriting to those terms of the untyped lambda calculus, which can be assigned conjunctive types, using an extension of Tait's method. While Barbanera's result strenghtens ours, Dougherty's uses sort-erasure and thus is applicable only to one-sorted algebraic systems: indeed, the following example shows that there are many-sorted algebraic rewrite systems which are SN, but which cease to be SN when the sorts are identified.

Example 1.1
Let i and j be two distinct sorts, and Σ_1 and Σ_2 be the following disjoint signatures:
$\Sigma_1 \overset{\text{def}}{=} \{f\colon i \times i \times i \rightarrow i,\ 0\colon i,\ 1\colon i\}$, and $\Sigma_2 \overset{\text{def}}{=} \{g\colon j \times j \rightarrow j\}$.
Let R and S be the following sets of equations over Σ_1 and Σ_2 respectively (these equations are due to Toyama [Toy87]):

$$\begin{array}{rcl} R & \overset{\text{def}}{=} & \{f(0,1,x) \rightarrow f(x,x,x)\} \\ S & \overset{\text{def}}{=} & \{g(u,v) \rightarrow u \\ & & \ \ g(u,v) \rightarrow v\} \end{array}$$

It is easily seen that both R and S are SN, and so is $R \cup S$, because the set of terms over $\Sigma_1 \cup \Sigma_2$ is the disjoint union of the sets of terms over Σ_1 and Σ_2, the sorts being distinct. However, if we identify the sorts i and j and consider the corresponding one-sorted signatures, then Toyama exhibits the mixed term $f(g(0,1), g(0,1), g(0,1))$, which rewrites to itself in three steps.

2 Mixing algebra and polymorphic lambda calculus

This section is devoted to a review of the concepts and notation needed for stating our results. We start with an arbitrary 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.

Definition 2.1 *(Algebraic signature)*
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* $s \in S$.

The intention is that each symbol in Σ names some heterogenous operation which takes arguments of sorts (in order) $s_1, \ldots, s_n$ and returns a result of sort s.

Definition 2.2 *(Types)*
Let $\mathcal{V}$ be a countably infinite set of *type variables. Type expressions (types)* are defined by the following grammar:

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

where s ranges over S, and $t \in \mathcal{V}$.

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 $\mathcal{T}$.

Definition 2.3 *(Terms)*
Let $\mathcal{X}$ be a countably infinite set of *term variables. Raw terms* are defined by the following grammar:

$$M ::= \; f \mid x \mid (MM) \mid (M\tau) \mid (\lambda x{:}\, \sigma.\, M) \mid (\lambda t.\, M)$$

where f ranges over the function or constant symbols from a signature Σ, and $x \in \mathcal{X}$.

We denote by $\mathcal{R}\Lambda$ the set of all raw terms. Free and bound variables are defined as usual. We denote by $FV(M)$ the set of free variables of M. We denote by $FTV(M)$ the set of free type variables of M. Again we identify the terms which differ only in the name of the bound variables and bound type variables. We also follow the convention that in a given mathematical context (*e.g.,* definition, proof) all bound variables and type variables (in terms or types) are chosen to be different from the free variables and type variables [Bar84].

In order to define what it means for a raw term to type-check, we need the concept of a type assignment.

Definition 2.4 *(Type assignment)*
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. We write $\Delta \leq \Delta'$ when $dom\Delta \subseteq dom\Delta'$ and $\Delta'(x) = \Delta(x)$ for every $x \in dom\Delta$.

Definition 2.5 *(Declared term)*
A *declared term* is a pair $\langle \Delta, M \rangle$ consisting of a type assignement Δ and a raw term M, written $\Delta \triangleright M$.

A declared term $\Delta \triangleright M$ may or may not type-check. In order to define which declared terms type-check, we give the following *typing rules*, which are used to derive *type-checking judgments* of the form $\Delta \triangleright M : \sigma$. (The name of each rule corresponds to the raw term construct that it helps type-check.)

Definition 2.6 *(Typing Rules)*

Variables.

$$\Delta \triangleright x : \sigma$$

where $x : \sigma \in \Delta$.

Constants. For any $f \in \Sigma$ of arity $s_1 \cdots s_n$ and sort s, and for any Δ,

$$\Delta \triangleright f : \sigma$$

where $\sigma \overset{\text{def}}{=} s_1 \to \cdots \to s_n \to s$.

Application.

$$\frac{\Delta \triangleright M : \sigma \to \tau \quad \Delta \triangleright N : \sigma}{\Delta \triangleright (MN) : \tau}$$

Abstraction.

$$\frac{\Delta, x : \sigma \triangleright M : \tau}{\Delta \triangleright (\lambda x : \sigma.\ M) : \sigma \to \tau}$$

Type application.

$$\frac{\Delta \triangleright M : \forall t.\ \sigma}{\Delta \triangleright (M\tau) : \sigma[\tau/t]}$$

for any $\tau \in \mathcal{T}$.

Type abstraction.

$$\frac{\Delta \triangleright M : \sigma}{\Delta \triangleright (\lambda t.\ M) : \forall t.\ \sigma}$$

where $t \notin FTV(ran\Delta)$.

Definition 2.7
Given a declared term $\Delta \triangleright M$ and a type σ we say that $\Delta \triangleright M$ *has type* σ if the judgement $\Delta \triangleright M : \sigma$ is derivable. We say that a declared term *type-checks* if it has some type.

Clearly, if $\Delta \triangleright M$ type-checks, then $FV(M) \subseteq dom\Delta$. If $x{:}\,\sigma \in \Delta$, we say that $x{:}\,\sigma$ is *declared* in $\Delta \triangleright M$. A declared term $\Delta \triangleright M$ can have declared variables which do not belong to $FV(M)$. The following fact is well-known.

Lemma 2.8
If $\Delta \triangleright M$ type-checks then it has a unique *type, σ. Moreover, the judgement $\Delta \triangleright M : \sigma$ has a* unique *derivation.*

As the reader must have observed, it is notationally rather cumbersome to manipulate declared terms $\Delta \triangleright M$. It is possible to adopt certain conventions that will allow us to alleviate this burden when no ambiguities arise. Often, we will write $\Delta \triangleright M$ simply as M. In the case of an application MN, we tacitly assume that M and N are in fact declared terms $\Delta \triangleright M$ and $\Delta \triangleright N$ with *the same* Δ. In the case of an application $(\lambda x{:}\,\sigma.\, M)N$, we tacitly assume that M and N are in fact declared terms $\Delta, x{:}\,\sigma \triangleright M$ and $\Delta \triangleright N$ with *the same* Δ.

We will as much as possible avoid using explicitly declared terms and judgments except when necessary to avoid ambiguities. Unfortunately, there are a few cases where we will not be able to avoid declared terms.

Definition 2.9 *(Substitutions)*
A *substitution* is a map $\varphi : \mathcal{V} \cup \mathcal{X} \longrightarrow \mathcal{T} \cup \mathcal{R}\Lambda$, such that $\varphi(u) \neq u$ for finitely many $u \in \mathcal{V} \cup \mathcal{X}$, $\varphi(t) \in \mathcal{T}$ whenever $t \in \mathcal{V}$, and $\varphi(x) \in \mathcal{R}\Lambda$ whenever $x \in \mathcal{X}$. The *domain* of the substitution φ is the the set $dom\varphi = \{u \in \mathcal{V} \cup \mathcal{X} \mid \varphi(u) \neq u\}$.

A substitution $\varphi : \mathcal{V} \cup \mathcal{X} \longrightarrow \mathcal{T} \cup \mathcal{R}\Lambda$ can be uniquely extended (in the customary fashion, by recursion) to a map $\widehat{\varphi} : \mathcal{T} \cup \mathcal{R}\Lambda \longrightarrow \mathcal{T} \cup \mathcal{R}\Lambda$, which is a homomorphism with respect to the type and term structure. [5]

We define the result of applying φ to a (raw) term M or a type σ as $M[\varphi] \stackrel{\text{def}}{=} \widehat{\varphi}(M)$, and $\sigma[\varphi] \stackrel{\text{def}}{=} \widehat{\varphi}(\sigma)$.

A *type substitution* is a substitution φ such that $dom\varphi \subseteq \mathcal{V}$ (and then $\varphi : \mathcal{V} \longrightarrow \mathcal{T}$). A *term substitution* is a substitution φ such that $dom\varphi \subseteq \mathcal{X}$ (and then $\varphi : \mathcal{X} \longrightarrow \mathcal{R}\Lambda$).

If $dom\varphi = \{t_1, \ldots, t_m, x_1, \ldots, x_n\}$, $\varphi(t_i) = \sigma_i$, and $\varphi(x_j) = M_j$ $(t_i \in \mathcal{V}, x_j \in \mathcal{X})$, we also denote the substitution φ as $[\sigma_1/t_1, \ldots, \sigma_m/t_m, M_1/x_1, \ldots, M_n/x_n]$, (and we denote $M[\varphi]$ as $M[\sigma_1/t_1, \ldots, \sigma_m/t_m, M_1/x_1, \ldots, M_n/x_n]$).

[5]Strictly speaking, one must define substitution *before* identifying α-congruent expressions (*i.e.*, expressions which differ in the name of the bound variables) and then show that it can be extended to α-congruence classes, upon which it indeed acts like a homomorphism (see [Bar84], appendix C).

Since types do not contain term variables, note that $\sigma[\sigma_1/t_1, \ldots, \sigma_m/t_m, M_1/x_1, \ldots, M_n/x_n]$ is in fact equal to $\sigma[\sigma_1/t_1, \ldots, \sigma_m/t_m]$. Given a type substitution $\theta = [\sigma_1/t_1, \ldots, \sigma_m/t_m]$ and a term substitution $\varphi = [M_1/x_1, \ldots, M_n/x_n]$, we denote as $\theta \cup \varphi$ the substitution $[\sigma_1/t_1, \ldots, \sigma_m/t_m, M_1/x_1, \ldots, M_n/x_n]$, which is well defined since $\mathcal{V}$ and $\mathcal{X}$ are disjoint.

We will be considering substitutions with some type-preserving properties.

Definition 2.10
Let Δ and Δ' be two type assignments, and let φ be a substitution, $\varphi : \mathcal{V} \cup \mathcal{X} \longrightarrow \mathcal{T} \cup \mathcal{R}\Lambda$. We say that φ *type-checks between* Δ *and* Δ', iff $dom\varphi \cap \mathcal{X} = dom\Delta$, and $\Delta' \triangleright x[\varphi] : \Delta(x)[\varphi]$ is derivable for every $x \in dom\Delta$. We will sometimes abbreviate "φ type-checks between Δ and Δ'" by the notation $\varphi : \Delta \longrightarrow \Delta'$.

Note that the above definition makes sense, since $\Delta(x)$ is a type, and thus only the type components of φ are substituted in $\Delta(x)$. Also, when φ is a term substitution, $\varphi : \Delta \longrightarrow \Delta'$ simply means that $\varphi : \Delta \longrightarrow \Delta'$ is type-preserving (since in this case, $\Delta(x)[\varphi] = \Delta(x)$).

The following lemma is easily shown.

Lemma 2.11 *Given a substitution* $\varphi : \Delta \longrightarrow \Delta'$*, if* $\Delta \triangleright M : \sigma$*, then* $\Delta' \triangleright M[\varphi] : \sigma[\varphi]$.

We define the usual *reduction relations* at the level of raw terms. This is justified by lemma 2.13.

Definition 2.12 *(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

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

Lemma 2.13 *If $\Delta \rhd M$ type-checks and $M \xrightarrow{\lambda^{\forall}} N$ then $\Delta \rhd N$ also also type-checks and has the same type.*

We will also need

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

It is well-known that both $\lambda^{\forall}$-reduction and λ^{-}-reduction are canonical (*i.e.*, strongly normalizing and confluent) on all terms. In fact, the generalized method of candidates presented in section 4 can be used to prove this (see theorem 4.11). We denote by $\lambda^{\forall}nf(X)$ and $\lambda^{-}nf(X)$ the corresponding normal forms of X.

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

Definition 2.14 *(Algebraic terms)*
A type assignment is *algebraic* iff all the types occurring in it are sorts. Among the polymorphic declared terms that type-check, *algebraic* declared terms are defined inductively as follows:

- Any term of the form $\Delta \rhd x$, where Δ is algebraic and x is declared in Δ, is an algebraic term.
- If $\Delta \rhd f$ has type $s_1 \to \cdots \to s_n \to s$, where f is a symbol in Σ, the type assignement Δ is algebraic, and $\Delta \rhd A_1 : s_1, \ldots, \Delta \rhd A_n : s_n$ are algebraic terms, then $\Delta \rhd f\, A_1 \cdots A_n$ is an algebraic term.

Clearly, the types of algebraic terms are actually sorts.

Definition 2.15 *(Algebraic rewrite rules)*
An *algebraic rewrite rule*, written $r \equiv \Gamma \rhd A \to B : s$, is a pair r of algebraic terms, $\Gamma \rhd A$ and $\Gamma \rhd B$ which have the same type (sort) s, and such that $FV(B) \subseteq FV(A)$, and A is not a variable. [6]

Each algebraic rewrite rule determines a reduction relation on *all* declared terms that type-check, not only the algebraic ones. In order to precisely define this relation, we introduce *contexts with exactly one hole*, in the spirit of [Bar84].

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

Definition 2.16 *(Contexts)*
A *raw context* is a raw term in which an additional special constant $\bigcirc$ (called *hole*) can occur. Given a type σ, a *(type-checked declared) context with one hole of type σ* consists of a type assignment Δ and a raw context C in which the hole occurs exactly once, such that $\Delta \triangleright C$ type-checks if we add the *hole axiom scheme* $\Theta \triangleright \bigcirc : \sigma$ where Θ ranges over all type assignments. We use the notation $\Delta \triangleright C[\ : \sigma]$ for such a context.

By lemma 2.8, a context $\Delta \triangleright C[\ : \sigma]$ has a unique type τ and $\Delta \triangleright C : \tau$ has a unique derivation. In this derivation, there is exactly one instance of the hole axiom scheme. Say that this instance is $\Delta' \triangleright \bigcirc : \sigma$. Since the derivation is unique, Δ, C, and σ determine Δ'. Then, given a declared term $\Delta' \triangleright M$ of type σ, we can "plug the hole" in the context, by replacing $\Delta' \triangleright \bigcirc : \sigma$ with the derivation of $\Delta' \triangleright M : \sigma$. The resulting derivation type-checks an actual term (no holes), which we will denote by $\Delta \triangleright C[\Delta' \triangleright M]$. As opposed to terms, contexts are *not* considered modulo renaming of bound variables. In fact, their use is motivated precisely by the situations in which a binding λx in C captures a variable x that is free in M, something that cannot be simulated with substitution. In working with declared contexts, as with declared terms, we will omit the type assignments when no ambiguities arise.

Definition 2.17 *(Algebraic reduction)*
Given an algebraic rewrite rule $r \equiv \Gamma \triangleright A \rightarrow B : s$, we define a reduction relation on declared terms as follows

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

there exists a context $\Delta \triangleright C[\ : s]$ and a term substitution $\varphi : \Gamma \longrightarrow \Gamma'$, such that Γ' is the type assignment of the instance of the hole axiom scheme used to type-check the context, and such that [7]

$$\Delta \triangleright M \equiv \Delta \triangleright C[\Gamma' \triangleright A[\varphi]] \qquad\qquad \Delta \triangleright N \equiv \Delta \triangleright C[\Gamma' \triangleright B[\varphi]]$$

For simplicity, we write $M \xrightarrow{r} N$, tacitly assuming that M and N are declared terms with the same type assignment Δ. Clearly, from the definition, if $M \xrightarrow{r} N$ then M and N type-check and have the same type. One can easily check the following fact.

Lemma 2.18 *If A is algebraic and $A \xrightarrow{r} M$, then M 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 $m \equiv p \rightarrow p'$ and any many-sorted Σ-terms q, q'

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

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

[7]Strictly speaking, we have to allow *variants* of a rule, that is, instances $\Gamma' \triangleright A[\nu] \rightarrow A'[\nu] : \tau$, where $\nu : \Gamma \longrightarrow \Gamma'$ is a renaming substitution which is a bijection bewteen $dom\Gamma$ and $dom\Gamma'$.

Definition 2.19
Let R be a set of algebraic rewrite rules. Define the following reduction relations on terms:

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

For any of these reduction relations, we will denote by $\twoheadrightarrow$ the reflexive and transitive closure of $\longrightarrow$.

Example 2.20
Consider the signature Σ defined by: $a, b, c\colon s, \quad f\colon s \rightarrow s \rightarrow s$ (where s is a sort),

and the rewrite rule: $x\colon s, y\colon s, z\colon s \rhd fx(fyz) \longrightarrow f(fxy)z\colon s$.

We have the following reduction sequence:

$$\begin{aligned} ((\lambda t.\, \lambda x\colon t.\, x)s)(fa(fbc)) &\longrightarrow ((\lambda t.\, \lambda x\colon t.\, x)s)(f(fab)c) \\ &\longrightarrow (\lambda x\colon s.\, x)(f(fab)c) \\ &\longrightarrow f(fab)c. \end{aligned}$$

Finally, we state precisely our main result:

(Conservation of Strong Normalization.) If $\xrightarrow{R}$ is strongly normalizing on algebraic terms then $\xrightarrow{\lambda^{\mathsf{v}} R}$ is strongly normalizing on all terms that type-check.

3 Algebraic rewriting of higher-order terms

In this section, we show that strong normalization of algebraic reduction on algebraic terms transfers to algebraic reduction on arbitrary terms. The section's main result, which will be proved later as theorem 3.10, can be stated as follows.

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

The proof of the main claim will require some auxiliary lemmas, and in order to understand why they are needed, we begin by sketching this proof.

Sketch of proof for the main claim. 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 out of M. Because any R-reduction from a term of the form $H\, T_1 \cdots T_k$ where H is an abstraction, a type abstraction, a variable, or a constant which takes $> k$ arguments (*i.e.,* the length of its arity is $> k$), must take place inside some term among the H and T_i's, by an argument involving a form of the "pigeonhole principle", we can show that one of the reduction sequences from some term among H and the T_i's must be infinite.[8] But the existence of an infinite reduction from some term among H and the T_i's contradicts the induction hypothesis. 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. $\square$

Definition 3.1 *(Algebraic trunk decomposition)*
An *algebraic trunk decomposition* of a declared term that type-checks $\Gamma \rhd M$ consists of an algebraic term $\Delta \rhd A$ (the "trunk") and a term substitution $\varphi\colon \Delta \longrightarrow \Gamma$ such that $M \equiv A[\varphi]$, $dom\varphi = FV(A)$, each variable in A occurs 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.

Strictly speaking, a trunk decomposition for $\Gamma \rhd M$ is a pair $\langle \Delta \rhd A, \varphi\colon \Delta \longrightarrow \Gamma \rangle$ with the above properties, but for simplicity of notation, we will often denote a trunk decomposition of M as $A[\varphi]$. Given φ and φ' with $dom\varphi = dom\varphi'$, the notation $\varphi \overset{R}{\twoheadrightarrow} \varphi'$ means that $\varphi(x) \overset{R}{\twoheadrightarrow} \varphi'(x)$ for every $x \in dom\varphi$.

The following terminology will also be useful. A term whose type is a sort and which 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, is called a *nontrunk term.* A term $f\,M_1 \ldots M_k$ whose type is a sort and where f is a constant taking k arguments, is called a *trunk term.*

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, and when M is a trunk term, A is not a variable.

Proof. Immediate. $\square$

Equipped with this tool, the last case in the proof of the main theorem follows from the following result, proved later as lemma 3.9.

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

Before proving the secondary claim, 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

[8]This argument will be presented more rigorously later when we prove theorem 3.10.

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 $A[\varphi] \overset{R}{\twoheadrightarrow} A'[\varphi']$ then for each $x' \in FV(A')$ there is an $x \in FV(A)$ such that $\varphi(x) \overset{R}{\twoheadrightarrow} \varphi'(x')$. 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, example 3.4 shows that $A[\varphi] \overset{R}{\twoheadrightarrow} A'[\varphi']$ does not necessarily imply $A \overset{R}{\twoheadrightarrow} A'$.

It will be useful to distinguish between algebraic trunk reduction steps and non-trunk reduction steps.

Definition 3.3
We shall denote algebraic *trunk reductions* by $\xrightarrow{tR}$, and algebraic reductions in the non-trunk part by $\xrightarrow{ntR}$ (*non-trunk reductions*).

It is important to note that if a nontrunk term M R-reduces to another term N, then N cannot be a trunk term. This implies that for a non-trunk reduction $M \overset{ntR}{\twoheadrightarrow} N$, if $M = A[\varphi]$ is a trunk decomposition of M, then $N = A[\varphi']$ for *the same trunk* A, i.e., the trunk does not grow in a non-trunk R-reduction. Unfortunately, the trunk can grow when some $\varphi(x)$ β-reduces.

Example 3.4

Let s be a sort, and let $f : s \to s \to s$, $g : s \to s \to s \to s$, and $a, b, c : s$ be constants. Consider the following set of rewrite rules $R = \{fxx \longrightarrow gxxx,\ a \longrightarrow b,\ b \longrightarrow c\}$ where $x : s$ is a first-order variable. Consider also the declared term $M = f(za)(zb)$, where $z : s \to s$ is a higher-order variable. While we have the rewrite sequence

$$\begin{aligned} M &\xrightarrow{ntR} f(zb)(zb) \\ &\xrightarrow{tR} g(zb)(zb)(zb) \\ &\xrightarrow{ntR} g(zb)(zc)(zb), \end{aligned}$$

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

A number of auxiliary lemmas will be needed in order to obtain a proof of the secondary claim (lemma 3.9).

Lemma 3.5
If $M \equiv A[\varphi] \xrightarrow{R} N$, then the following holds.

(1) If $M \xrightarrow{tR} N$, then $N \equiv A'[\varphi']$, where for every $y \in dom\varphi'$, there is some $x \in dom\varphi$ such that $\varphi'(y) = \varphi(x)$, and A' is some algebraic term, else

(2) $M \xrightarrow{ntR} N$, and $N \equiv A[\varphi']$, where $\varphi(x_i) \xrightarrow{R} \varphi'(x_i)$ for some $x_i \in dom\varphi$ and $\varphi'(x_j) = \varphi(x_j)$ for all $j \neq i$.

Proof. Immediate by a case analysis depending on which kind of redex is being contracted. □

Note that case (2) holds because a nontrunk term cannot rewrite to a trunk term. Thus, the trunk cannot grow.

Lemma 3.6
If $M \equiv A[\varphi] \overset{R}{\twoheadrightarrow} M' \equiv A'[\varphi']$, then for every $y \in dom\varphi'$, there is some $x \in dom\varphi$ such that $\varphi(x) \overset{R}{\twoheadrightarrow} \varphi'(y)$.

Proof. An easy induction on the number of rewrite steps using lemma 3.5. □

Next, we will exploit the observation made in example 3.4 about the positive effect of identifying the variables that occur in the trunk A of an algebraic trunk decomposition $A[\varphi]$.

Definition 3.7
For every sort s, let z_s be some designated variable of that sort. If A is an algebraic term, we let $A[\zeta]$ be the term obtained by replacing, for every sort s, all free variables of sort s in A by z_s. (Note that $A[\zeta]$ is also an algebraic term.)

Lemma 3.8
If $A[\varphi] \xrightarrow{tR} A'[\varphi']$ then $A[\zeta] \xrightarrow{R} A'[\zeta]$. If $A[\varphi] \xrightarrow{ntR} A'[\varphi']$ then $A[\zeta] \equiv A'[\zeta]$.

Proof. For the first part, let $A[\zeta] \equiv A[\nu]$, where $\nu(x) = z_s$ for every variable $x \in FV(A)$ of sort s.[9] Since $A[\varphi] \xrightarrow{tR} A'[\varphi']$, by case (1) of lemma 3.5, we have that for every $y \in dom\varphi'$, there is some $x \in dom\varphi$ such that $\varphi'(y) = \varphi(x)$. Thus, we can define a function $h: dom\varphi' \longrightarrow dom\varphi$ such that $\varphi'(y) = \varphi(h(y))$ for every $y \in dom\varphi'$, and it is easy to see that we have $A[\nu] \xrightarrow{tR} A'[\nu']$, where $dom\nu' = dom\varphi'$ and $\nu'(y) = \nu(h(y))$ for every $y \in dom\nu'$. But then, $\nu'(y) = z_s$ for every $y \in dom\nu'$ of sort s, and so $A[\nu'] \equiv A'[\zeta]$, as claimed.

The second part follows from case (2) of lemma 3.5. □

Lemma 3.9
Let $\xrightarrow{R}$ be SN on algebraic terms. Let $A[\varphi]$ be an algebraic trunk decomposition. If $\xrightarrow{R}$ is SN on $\varphi(x)$ for each $x \in FV(A)$, then $\xrightarrow{R}$ is SN on $A[\varphi]$.

[9]This is necessary because ζ being infinite, strictly speaking, it is not a substitution. However, ν is a substitution agreeing with ζ on $FV(A)$.

Proof. First, observe that if $M \equiv A[\varphi] \overset{R}{\twoheadrightarrow} M' \equiv A'[\varphi']$, then $\overset{R}{\twoheadrightarrow}$ is SN on $\varphi'(y)$ for every $y \in dom\varphi'$. This follows from lemma 3.6, since for every $y \in dom\varphi'$ there is some $x \in dom\varphi$ such that $\varphi(x) \overset{R}{\twoheadrightarrow} \varphi'(y)$. Assume there is an infinite reduction from M. There are two cases.

Case 1. The infinite reduction sequence $M \equiv A[\varphi] \overset{R}{\twoheadrightarrow} \ldots$ contains only a finite number of trunk rewrites. This means that the reduction is of the form $M \equiv A[\varphi] \overset{R}{\twoheadrightarrow} M' \equiv A'[\varphi'] \overset{ntR}{\twoheadrightarrow} \ldots$, where the infinite reduction from M' does not contain any trunk rewrites. Then, for every M'' such that $M' \equiv A'[\varphi'] \overset{ntR}{\twoheadrightarrow} M'' \equiv A'[\varphi'']$ in this infinite reduction, we have $dom\varphi'' = dom\varphi'$ and $\varphi' \overset{R}{\twoheadrightarrow} \varphi''$. Letting $dom\varphi' = \{x_1, \ldots, x_m\}$, if there is some $k \geq 0$ such that each reduction sequence from $\varphi'(x_i)$ is of length bounded by k, then any reduction sequence $M' \equiv A'[\varphi'] \overset{ntR}{\twoheadrightarrow} \ldots$ has length bounded by mk. Thus, there must be an infinite reduction from $\varphi'(x_i)$ for some $x_i \in dom\varphi'$, contradicting the fact that $\varphi'(y)$ is SN for every $y \in dom\varphi'$.

Case 2. The infinite reduction sequence $M \equiv A[\varphi] \overset{R}{\twoheadrightarrow} \ldots$ contains an infinite number of trunk rewrites. In view of lemma 3.8, we transform each term $B[\psi]$ in the infinite reduction sequence out of M into a corresponding algebraic term $B[\zeta]$. Since there are infinitely many trunk rewrite steps, the result will be an infinite sequence of R-reductions on algebraic terms, contradicting the assumption that $\overset{R}{\twoheadrightarrow}$ is SN on algebraic terms. $\square$

We can now prove the main theorem of this section.

Theorem 3.10
If $\xrightarrow{R}$ is strongly normalizing on algebraic terms then $\xrightarrow{R}$ is strongly normalizing on all terms that type-check.

Proof. 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, and suppose that there is an infinite R-reduction sequence $M \overset{R}{\twoheadrightarrow} M_1 \overset{R}{\twoheadrightarrow} \ldots M_n \overset{R}{\twoheadrightarrow} M_{n+1} \overset{R}{\twoheadrightarrow} \ldots$ out of M. There are two cases.

Case 1. The term H in $M \equiv H\,T_1 \cdots T_k$ is not a constant taking k arguments. In this case, because any R-reduction from a term of the form $H\,T_1 \cdots T_k$ where H is an abstraction, a type abstraction, a variable, or a constant which takes $> k$ arguments (*i.e.,* the length of its arity is $> k$), must take place inside some term among the H and T_i's, it is easily seen by induction on n that each term M_n is of the form $H^n\,T_1^n \cdots T_k^n$ with $H \overset{R}{\twoheadrightarrow} H^n$ and $T_i \overset{R}{\twoheadrightarrow} T_i^n$, for $i = 1, \ldots, k$. Then, one of the reduction sequences from some term among H and the T_i's must be infinite, since otherwise, if m is an upper bound on the length of these reduction sequences, the length of the reduction sequence $M \overset{R}{\twoheadrightarrow} M_1 \overset{R}{\twoheadrightarrow} \ldots M_n \overset{R}{\twoheadrightarrow} M_{n+1} \overset{R}{\twoheadrightarrow} \ldots$ is at most $(k+1)m$. [10] But the existence of an infinite reduction from some term among H and the T_i's contradicts the induction hypothesis.

Case 2. The term H is a constant which takes exactly k arguments, and the type of M is a sort. But then, M can be decomposed as $M \equiv A[\varphi]$ where A is not a variable. Thus, each $\varphi(x_i)$ has size strictly smaller that the size of M, and by the induction hypothesis, $\varphi(x_i)$ is SN for every $x_i \in dom\varphi$. We conclude by applying lemma 3.9. □

4 Generalized candidates of reducibility

In this section, we present our generalization of Girard's 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 4.8. We choose to present a version using so-called saturated sets. Another version using Girard sets (sets satisfying conditions given in Girard's thesis [Gir72] and in [GLT89]) is possible. For a presentation of this other version and a detailed comparison of the various conditions involved, we refer the reader to [Gal90].

Let P be a property of declared terms that type-check. For each type σ, let P_σ be the set of all declared terms of type σ which have the property P.

Definition 4.1 *(Sets of P-candidates)*
The *family of sets of P-candidates* is the $\mathcal{T}$-indexed family $\mathcal{C} = (\mathcal{C}_\sigma)_{\sigma\in\mathcal{T}}$, where each $\mathcal{C}_\sigma$ consists of all sets C (called *P-candidates*) of declared terms of type σ having the property P (*i.e.,* $C \subseteq P_\sigma$), and such that the following conditions hold.

(Cand 1) If x is a variable, $T_1, \ldots, T_k$ $(k \geq 0)$ are either declared terms that type-check 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, $N_1, \ldots, N_k$ $(k \geq 0)$ are declared terms that type-check which have the property P, and $f\,N_1 \cdots N_k$ has type σ, then $f\,N_1 \cdots N_k \in C$. (Note that the length of the arity of f may differ from k.)

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

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

[10]This argument uses a form of the "pigeonhole principle". A similar kind of argument already occurred in the proof of lemma 3.9 and will occur a few more times.

(Cand 5) Whenever $\Delta \triangleright M \in C$ and $\Delta \leq \Delta'$, then $\Delta' \triangleright M \in C$. [11]

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

(Clo 1a) If $\Delta \triangleright M$ type-checks and if $\Delta, x{:}\,\sigma \triangleright Mx$ has property P (in particular, also type-checks), then $\Delta \triangleright M$ 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 set P_σ is itself a P-candidate (*i.e.,* $P_\sigma \in \mathcal{C}_\sigma$).

Observe that in stating the above conditions, except for conditions (Clo 1a) and (Cand 5) where this is not possible, rather than using declared terms (requiring the Δ part), we have dropped the Δ part, making use of the tacit assumptions discussed in section 2.

The main theorem of this section (theorem 4.8) will state the following fact:

Claim. If P is candidate-closed, then every declared term that type-checks has property P.

The proof of this claim requires defining a sort of semantic interpretations of the types involving the family $\mathcal{C}$ of sets of P-candidates. First, we need the concept of a candidate assignment.

Definition 4.2 *(Candidate assignment)*
Let P be a property of declared terms that type-check. A *candidate assignment* (with respect to P) is map $\rho{:}\, \mathcal{V} \longrightarrow \mathcal{T} \times \mathcal{C}$ that associates to each type variable t a pair $\langle \tau, C\rangle$, where $\tau \in \mathcal{T}$ is some type, and C is a P-candidate such that $C \in \mathcal{C}_\tau$. Furthermore, denoting the map such that $t \mapsto \tau$ as $\rho_\mathcal{T}$, we assume that the set $\{t \in \mathcal{V} \mid \rho_\mathcal{T}(t) \neq t\}$ is finite. Thus, $\rho_\mathcal{T}$ is a type substitution. The map such that $t \mapsto C$ is denoted by $\rho_\mathcal{C}$. With a slight abuse of notation, we will sometimes denote $\rho_\mathcal{T}$ or $\rho_\mathcal{C}$ simply by ρ.

We associate to each type σ and each candidate assignment ρ a set of declared terms that type-check, denoted $[\![\sigma]\!]\rho$, as follows.

Definition 4.3

$$\begin{aligned}
[\![s]\!]\rho &\overset{\text{def}}{=} P_s\\
[\![t]\!]\rho &\overset{\text{def}}{=} \rho_\mathcal{C}(t)\\
[\![\sigma \rightarrow \tau]\!]\rho &\overset{\text{def}}{=} \{\Delta \triangleright M \mid \Delta \triangleright M{:}\,(\sigma \rightarrow \tau)[\rho_\mathcal{T}],\ \forall \Delta' \triangleright N,\ \Delta \leq \Delta' \textit{ and}\\
&\qquad \Delta' \triangleright N \in [\![\sigma]\!]\rho \Longrightarrow \Delta' \triangleright MN \in [\![\tau]\!]\rho\}\\
[\![\forall t.\, \sigma]\!]\rho &\overset{\text{def}}{=} \{\Delta \triangleright M \mid \Delta \triangleright M{:}\,(\forall t.\, \sigma)[\rho_\mathcal{T}],\ \forall \tau \in \mathcal{T}\ \forall C \in \mathcal{C}_\tau,\ \Delta \triangleright M\tau \in [\![\sigma]\!]\rho\{t{:}= \langle\tau, C\rangle\}\ \}
\end{aligned}$$

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

[11]The need for (Cand 5) appeared when the proof of lemma 4.7 was written in full detail. It seems that (Cand 5) has been overlooked in previous work involving typed candidates.

It is easy to see that if $\Delta \triangleright M \in [\![\sigma]\!]\rho$, then $\Delta \triangleright M\colon \sigma[\rho_{\mathcal{T}}]$.

The next lemma shows that the closure conditions on P-candidates are sufficient to insure that the sets $[\![\sigma]\!]\rho$ are already in $\mathcal{C}$.

Lemma 4.4
Assume that P is candidate-closed. For every type σ and every candidate assignment ρ, $[\![\sigma]\!]\rho \in \mathcal{C}_{\sigma[\rho]}$, i.e., *$[\![\sigma]\!]\rho$ is a P-candidate of type $\sigma[\rho]$.*

Proof. The proof is by induction on the size of σ. Such a proof is given in [Gal90], although for a slightly different notation. For the benefit of the readers who are not familiar with this kind of argument, we prove closure under (Cand 1), (Cand 5), and that every declared term in $[\![\sigma]\!]\rho$ has property P. First, we prove that (Cand 1) holds.

That (Cand 1) holds when σ is a variable or a constant is trivial, since each $\rho(t)$ is a P-candidate, and P_σ itself is a P-candidate by (Clo 2).

Assume that $\Delta \triangleright x\,T_1 \cdots T_k\colon (\sigma \rightarrow \tau)[\rho]$, where $T_1, \ldots, T_k$ $(k \geq 0)$ are either declared terms which have the property P or types. Let $\Delta' \triangleright N$ be any declared term such that $\Delta' \triangleright N \in [\![\sigma]\!]\rho$, with $\Delta \leq \Delta'$. Then $\Delta' \triangleright N\colon \sigma[\rho]$, and so $\Delta' \triangleright x\,T_1 \cdots T_k N\colon \tau[\rho]$. By the induction hypothesis applied to τ, since (Cand 1) holds, we have $\Delta' \triangleright x\,T_1 \cdots T_k N \in [\![\tau]\!]\rho$. But then, by the definition of $[\![\sigma \rightarrow \tau]\!]\rho$, we have $\Delta \triangleright x\,T_1 \cdots T_k \in [\![\sigma \rightarrow \tau]\!]\rho$.

Finally, assume that $\Delta \triangleright x\,T_1 \cdots T_k\colon (\forall t.\, \sigma)[\rho]$, where $T_1, \ldots, T_k$ $(k \geq 0)$ are either declared terms which have the property P or types. Let $\tau \in \mathcal{T}$ be any type. We can assume by α-renaming that t is not free in τ and that no capture takes place when ρ is applied, and thus, $(\forall t.\sigma)[\rho] = \forall t.\sigma[\rho]$, $(\sigma[\rho])[\tau/t] = \sigma[\rho\{t\colon= \tau\}]$, and $\Delta \triangleright x\,T_1 \cdots T_k \tau\colon \sigma[\rho\{t\colon= \tau\}]$. By the induction hypothesis applied to σ, $[\![\sigma]\!]\rho' \in \mathcal{C}_{\sigma[\rho']}$ for every ρ', and in particular, for every ρ' of the form $\rho\{t\colon= \langle\tau, C\rangle\}$, where $\tau \in \mathcal{T}$ and $C \in \mathcal{C}_\tau$. Thus, $\Delta \triangleright x\,T_1 \cdots T_k \tau \in [\![\sigma]\!]\rho\{t\colon= \langle\tau, C\rangle\}$ for all $\tau \in \mathcal{T}$ and $C \in \mathcal{C}_\tau$, and by the definition of $[\![\forall t.\, \sigma]\!]\rho$, this means that $\Delta \triangleright x\,T_1 \cdots T_k \in [\![\forall t.\, \sigma]\!]\rho$. Thus, we have proved (Cand 1). The proof for (Cand 2), (Cand 3), and (Cand 4), is very similar. (Cand 5) follows immediately by inspection of the clauses of definition 4.3.

Finally, we prove that every declared term in $[\![\sigma]\!]\rho$ has property P. This is obvious when σ is a variable or a constant, since each $\rho(t)$ is a P-candidate, and P_σ itself is a P-candidate by (Clo 2).

Let $\Delta \triangleright M \in [\![\sigma \rightarrow \tau]\!]\rho$. Note that for every variable $x \notin dom\Delta$, since $\Delta, x\colon \sigma \triangleright x\colon \sigma$, by (Cand 1), $\Delta, x\colon \sigma \in [\![\sigma]\!]\rho$, and by the definition of $[\![\sigma \rightarrow \tau]\!]\rho$, we have $\Delta, x\colon \sigma \triangleright Mx \in [\![\tau]\!]\rho$. Applying the induction hypothesis to τ, the term $\Delta, x\colon \sigma \triangleright Mx$ has property P, and by (Clo 1a), this implies that $\Delta \triangleright M$ has property P.

Let $\Delta \triangleright M \in [\![\forall t.\, \sigma]\!]\rho$. By the definition of $[\![\forall t.\, \sigma]\!]\rho$, we have $\Delta \triangleright Mt \in [\![\sigma]\!]\rho\{t\colon= \langle t, P_t\rangle\}$. Applying the induction hypothesis to σ, the term $\Delta \triangleright Mt$ has property P, and by (Clo 1b), this implies that $\Delta \triangleright M$ has property P. This concludes the induction showing that every declared term in $[\![\sigma]\!]\rho$ has property P. $\square$

We also need the following technical lemmas.

Lemma 4.5
For every types σ, τ, for every ρ, we have

$$[\![\sigma[\tau/t]]\!]\rho = [\![\sigma]\!]\rho\{t := \langle \tau[\rho], [\![\tau]\!]\rho \rangle\}.$$

Proof. By induction on σ. $\square$

Lemma 4.6
Given any two candidate assignments ρ_1 and ρ_2, for every type σ, if $\rho_1(t) = \rho_2(t)$ for all $t \in FTV(\sigma)$, then $[\![\sigma]\!]\rho_1 = [\![\sigma]\!]\rho_2$

Proof. By induction on σ. $\square$

All this is then used to show that every term that type-checks belongs to some P-candidate, and thus has the property P. One uses induction on deductions, strengthening the induction hypothesis as shown in lemma 4.7. Given a candidate assignment ρ and a term substitution φ, we will continue to slightly abuse the notation and write $\rho \cup \varphi$ for the substitution $\rho_{\mathcal{T}} \cup \varphi$.

Lemma 4.7
*For every candidate assignment ρ, for every term substitution φ, for every Δ, for every declared term that type-checks $\Gamma \triangleright M$, if $\rho \cup \varphi$ type-checks between Γ and Δ (*i.e., $\rho \cup \varphi : \Gamma \longrightarrow \Delta$*), and if $\Delta \triangleright \varphi(x) \in [\![\Gamma(x)]\!]\rho$ for every $x \in dom\Gamma$, then we have $\Delta \triangleright M[\rho \cup \varphi] \in [\![\sigma]\!]\rho$, where σ is the type of $\Gamma \triangleright M$.*

Before giving a proof, note that Lemma 4.7 has the flavor of a Kripke-style soundness result. Indeed, if we think of the Δ's as worlds (ordered by inclusion $\leq$), then we can think of the sets $[\![\sigma]\!]\rho$ as the carriers of some sort of Kripke structure. The Kripke-style nature of theorem 4.7 can be made more explicit if we introduce the following definitions.

Say that $\Gamma[\rho \cup \varphi]$ is satisfied in Δ, denoted $\Delta \models \Gamma[\rho \cup \varphi]$, iff $\rho \cup \varphi$ type-checks between Γ and Δ and $\Delta \triangleright \varphi(x) \in [\![\Gamma(x)]\!]\rho$ for every $x \in dom\Gamma$. Also say that $M{:}\,\sigma$ is satisfied in Δ at $\rho \cup \varphi$, denoted $\Delta \models (M{:}\,\sigma)[\rho \cup \varphi]$, iff $\Delta \triangleright M[\rho \cup \varphi] \in [\![\sigma]\!]\rho$. Then, lemma 4.7 can be stated as follows:

For every Δ, ρ, and φ, if $\Delta \models \Gamma[\rho \cup \varphi]$ and $\Gamma \triangleright M{:}\,\sigma$, then $\Delta \models (M{:}\,\sigma)[\rho \cup \varphi]$.

Formulated this way, the theorem looks like a Kripke-style soundness result. However, this analogy will not be pursued further in this paper.

Proof of lemma 4.7. The proof proceeds by induction on the depth of the proof of the judgment $\Gamma \triangleright M{:}\,\sigma$. Such a proof is given in [Gal90], although for a slightly different notation. For the benefit of the readers who are not familiar with this kind of argument, we consider two cases, abstraction, and type application.

Case 1. (Abstraction)

$$\frac{\Gamma, x{:}\,\sigma \triangleright M{:}\,\tau}{\Gamma \triangleright (\lambda x{:}\,\sigma.\, M){:}\,\sigma \rightarrow \tau}$$

Let φ be any term substitution, ρ any candidate assignment, and Δ any type assignment such that $\rho \cup \varphi$ type-checks between Γ and Δ and $\Delta \triangleright \varphi(x) \in [\![\Gamma(x)]\!]\rho$ for every $x \in dom\Gamma$. Let Δ' be any type assignment such that $\Delta \leq \Delta'$, and let $\Delta' \triangleright N$ any declared term such that $\Delta' \triangleright N \in [\![\sigma]\!]\rho$. We claim that $\rho \cup \varphi\{x{:}= N\}$ type-checks between $\Gamma, x{:}\,\sigma$ and Δ'.

For every $y \in dom\Gamma$, since $\rho \cup \varphi$ type-checks between Γ and Δ and $\Delta \leq \Delta'$, we have $\Delta' \triangleright y[\varphi]{:}\,\Gamma(y)[\rho]$. We also have $\Delta' \triangleright x[\varphi\{x{:}= N\}]{:}\,\sigma[\rho]$, since $x[\varphi\{x{:}= N\}] = N$, and $\Delta' \triangleright N \in [\![\sigma]\!]\rho$. Thus, $\rho \cup \varphi\{x{:}= N\}$ type-checks between $\Gamma, x{:}\,\sigma$ and Δ'.

We also claim that $\Delta' \triangleright \varphi\{x{:}= N\}(y) \in [\![(\Gamma, x{:}\,\sigma)(y)]\!]\rho$ for every $y \in dom(\Gamma, x{:}\,\sigma)$. This is true for the following two reasons: (1) $\Delta \triangleright \varphi(x) \in [\![\Gamma(x)]\!]\rho$ for every $x \in dom\Gamma$, and by (Cand 5), we have $\Delta' \triangleright \varphi(x) \in [\![\Gamma(x)]\!]\rho$; (2) We also have $\Delta' \triangleright N \in [\![\sigma]\!]\rho$, and $\varphi\{x{:}= N\}(x) = N$.

Thus, we can apply the induction hypothesis to $\Gamma, x{:}\,\sigma \triangleright M{:}\,\tau$, ρ, $\varphi\{x{:}= N\}$, and Δ', and we have

$$\Delta' \triangleright M[\rho \cup \varphi\{x{:}= N\}] \in [\![\tau]\!]\rho.$$

However, by α-renaming if necessary, it can be assumed that x is not free in $FV(N)$ and not free in any $\varphi(x)$, where $x \in dom\Gamma$, and so

$$M[\rho \cup \varphi\{x{:}= N\}] = M[\rho \cup \varphi][N/x].$$

From $\Delta' \triangleright M[\rho \cup \varphi\{x{:}= N\}] \in [\![\tau]\!]\rho$ and $M[\rho \cup \varphi\{x{:}= N\}] = M[\rho \cup \varphi][N/x]$, we obtain

$$\Delta' \triangleright M[\rho \cup \varphi][N/x] \in [\![\tau]\!]\rho.$$

In particular, by setting $\Delta' \equiv \Delta, x{:}\,\sigma$ and $N \equiv x$, we have

$$\Delta, x{:}\,\sigma \triangleright M[\rho \cup \varphi] \in [\![\tau]\!]\rho,$$

and since $\Delta \leq \Delta'$, by (Cand 5), we have

$$\Delta', x{:}\,\sigma \triangleright M[\rho \cup \varphi] \in [\![\tau]\!]\rho.$$

Thus, by lemma 4.4, since $\Delta' \triangleright N \in [\![\sigma]\!]\rho$ and $\Delta', x{:}\,\sigma \triangleright M[\rho \cup \varphi] \in [\![\tau]\!]\rho$, both $\Delta' \triangleright N$ and $\Delta', x{:}\,\sigma \triangleright M[\rho \cup \varphi]$ have property P. Since we also have $\Delta' \triangleright M[\rho \cup \varphi][N/x] \in [\![\tau]\!]\rho$, we are in a position to apply (Cand 3), and we have

$$\Delta' \triangleright (\lambda x{:}\,\sigma[\rho].\, M[\rho \cup \varphi])N \in [\![\tau]\!]\rho.$$

Since $(\lambda x{:}\,\sigma[\rho].\, M[\rho \cup \varphi]) = (\lambda x{:}\,\sigma.\, M)[\rho \cup \varphi]$, we get

$$\Delta' \triangleright ((\lambda x{:}\,\sigma.\, M)[\rho \cup \varphi])N \in [\![\tau]\!]\rho,$$

and this for all $\Delta' \rhd N \in [\![\sigma]\!]\rho$. By the definition of $[\![\sigma \rightarrow \tau]\!]\rho$, this shows that

$$\Delta \rhd (\lambda x\colon \sigma.\, M)[\rho \cup \varphi] \in [\![\sigma \rightarrow \tau]\!]\rho,$$

as desired.

Case 2. (Type application)

$$\frac{\Delta \rhd M\colon \forall t.\, \sigma}{\Delta \rhd (M\tau)\colon \sigma[\tau/t]}$$

By the induction hypothesis, we have

$$\Delta \rhd M[\rho \cup \varphi] \in [\![\forall t.\, \sigma]\!]\rho.$$

By the definition of $[\![\forall t.\, \sigma]\!]\rho$, we have

$$\Delta \rhd (M[\rho \cup \varphi]\gamma) \in [\![\sigma]\!]\rho\{t\colon = \langle \gamma, C\rangle\},$$

for every $\gamma \in \mathcal{T}$ and $C \in \mathcal{C}_\gamma$. In particular, we can choose $\gamma \equiv \tau[\rho]$ and $C \equiv [\![\tau]\!]\rho$. By α-renaming if necessary, it can be assumed that t is not free in $FTV(\tau[\rho])$ and not free in $[\rho](t)$ for every $t \in \mathcal{V}$, and so, $(\forall t.\, \sigma)[\rho] = \forall t.\, \sigma[\rho]$, and $(\sigma[\rho])[\tau[\rho]/t] = \sigma[\rho\{t\colon = \tau[\rho]\}] = \sigma[\tau/t][\rho]$. We also have

$$\Delta \rhd (M[\rho \cup \varphi]\tau[\rho]) = \Delta \rhd (M\tau)[\rho \cup \varphi],$$

and so,

$$\Delta \rhd (M\tau)[\rho \cup \varphi] \in [\![\sigma]\!]\rho\{t\colon = \langle \tau[\rho], [\![\tau]\!]\rho\rangle\}.$$

However, by lemma 4.5, we have

$$[\![\sigma[\tau/t]]\!]\rho = [\![\sigma]\!]\rho\{t\colon = \langle \tau[\rho], [\![\tau]\!]\rho\rangle\},$$

and so, we obtain

$$\Delta \rhd (M\tau)[\rho \cup \varphi] \in [\![\sigma[\tau/t]]\!]\rho,$$

as desired. $\square$

Finally, we obtain the main theorem of this section.

Theorem 4.8
If P is candidate-closed, then every declared term that type-checks has property P.

Proof. Apply lemma 4.7 by choosing ρ such that $\rho(t) \overset{\text{def}}{=} \langle t, P_t\rangle$ for all $t \in \mathcal{V}$, and $\varphi(x) \overset{\text{def}}{=} x$ for all $x \in \mathcal{X}$. $\square$

We give some applications without proof. For more details and proofs, we refer the reader to [Gal90]. 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 4.9 (Girard) *" M is $\xrightarrow{\beta T \beta}$-strongly normalizing "*
is a candidate-closed property of terms M that type-check.

Theorem 4.10 (Girard) *" $\xrightarrow{\beta T \beta}$-confluence holds from M "*
is a candidate-closed property of terms M that type-check.

Theorem 4.11
The following are also candidate-closed properties of terms M that type-check:

- *" 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 "*

5 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 4.8, the main result of this paper (the conservation of the SN-property) will hold if we can show the following claim (proved later as theorem 5.6):

Claim. " M is $\xrightarrow{\lambda^{\forall} R}$-strongly normalizing " is a candidate-closed property of terms M that type-check.

Let us first sketch the structure of the proof of this claim. Of course, we need to check that the conditions (Cand 1)–(Cand 5), (Clo 1a), (Clo 1b), and (Clo 2) hold. In fact, the only difficult case is to check (Cand 2) for terms of the form $M \equiv f\, N_1 \cdots N_k$, where f is a constant taking k arguments. In this case, the type of M and that of all the terms in any reduction sequence from M is a sort, and 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 $\varphi'(x')$, $x' \in FV(A')$. Unfortunately, there is a small complication: it would be tempting to claim that whenever $M' \equiv A'[\varphi'] \overset{\lambda^{\forall} R}{\twoheadrightarrow} M'' \equiv A''[\varphi'']$ and no trunk-reductions take place, then $A'' = A'$. However, this is not necessarily true because we may have steps $M_1 \equiv A_1[\varphi_1] \xrightarrow{\beta} M_2 \equiv A_2[\varphi_2]$ in which $\varphi_1(y) \xrightarrow{\beta} N$ for some $y \in FV(A_1)$, and

N has a nontrivial trunk decomposition itself, which implies that A_2 is strictly larger than A_1. Fortunately, it is possible to show that each $\varphi'(x')$ is SN: see lemma 5.2.

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 algebraic reduction [BG89]. Instead we will use *long normal forms* (see definition 5.3 below).

We now state and prove the auxiliary lemmas needed for proving the claim. First, we need a more general version of lemma 3.5.

Lemma 5.1
If $M \equiv A[\varphi] \xrightarrow{\lambda^{\forall}R} N$, then the following holds.

(1) If $M \xrightarrow{tR} N$, then $N \equiv A'[\varphi']$, where for every $y \in dom\varphi'$, there is some $x \in dom\varphi$ such that $\varphi'(y) = \varphi(x)$, else

(2) $M \xrightarrow{\lambda^{\forall}R} N$, where $N \equiv A'[\varphi']$ and for every $y \in FV(A')$, either there is some $x \in dom\varphi$ and some subterm N_x of $N \equiv A'[\varphi']$ such that $\varphi(x) \xrightarrow{\lambda^{\forall}R} N_x$ and $\varphi'(y)$ is a subterm of N_x, or there is some $x \in dom\varphi$ such that $\varphi'(y) = \varphi(x)$.

Proof. Immediate by a case analysis depending on which kind of redex is being contracted. $\square$

Lemma 5.2
If $M \equiv A[\varphi] \overset{\lambda^{\forall}R}{\twoheadrightarrow} M' \equiv A'[\varphi']$ and $\varphi(x)$ is SN for every $x \in dom\varphi$, then $\varphi'(y)$ is SN for every $y \in dom\varphi'$.

Proof. An easy induction on the number of reduction steps using lemma 5.1 and the fact that a subterm of an SN term must be SN. $\square$

We will also need the concept of a *long normal form.* This is a straightforward generalization of the *η-expanded normal form* in [Hue75] called a *long $\beta\eta$-normal form* in [Sta82].

Definition 5.3 *(Long normal form)*
A term M is in long normal form if $M \equiv \lambda v_1. \cdots. \lambda v_k. h\, T_1 \cdots T_m$ where the v_i's are either type variables or of the form $y{:}\tau$, h is a variable or a constant, the T_j's are either type expressions or (inductively) terms in long normal form, we do *not* have $v_k \equiv T_m \equiv t$ for some type variable t (to avoid having a $\mathcal{T}\eta$-redex), and the type of $h\, T_1 \cdots T_m$ is either a sort, or a type variable, or of the form $\forall t.\, \sigma$. (We will often use the shorter notation $\lambda\vec{v}.\, h\, T_1 \cdots T_m \overset{\text{def}}{=} \lambda v_1. \cdots. \lambda v_k.\, h\, T_1 \cdots T_m$).

While long normal forms are in general *not* in η-normal form, the name is justified by the following result.

Lemma 5.4
Any term is $\lambda^{\forall}$-convertible to a unique long normal form.

Proof. Since every long normal form is also a λ^{-}-normal form, it is sufficient to show how to η-convert any λ^{-}-normal forms to a unique long normal form. If M is in λ^{-}-normal form then already $M \equiv \lambda v_1. \cdots. \lambda v_k. h\, T_1 \cdots T_m$ where the v_i's are either type variables or of the form $y{:}\tau$, h is a variable or a constant, the T_j's are either type expressions or terms in λ^{-}-normal form, and, we do *not* have $v_k \equiv T_m \equiv t$ for some type variable t. Suppose that in $\lambda\vec{v}.\, h\, T_1 \cdots T_m$ we have already (recursively and, for the uniqueness, inductively) η-converted those T_j which are terms (and therefore λ^{-}-normal forms of strictly smaller size) to their unique long normal form. Let the type of $h\, T_1 \cdots T_m$ be $\sigma_1 \to \cdots \to \sigma_n \to \tau$ where $n \geq 0$ and τ is either a sort, or a type variable, or of the form $\forall t.\, \sigma$ (any type is of this form). From this, the unique long normal form is reached by performing the η-expansions that give $\lambda\vec{v}.\, \lambda x_1{:}\sigma_1. \cdots. \lambda x_n{:}\sigma_n.\, h\, T_1 \cdots T_m\, U_1 \cdots U_n$ where U_i is the long normal form of x_i. $\square$

We denote by $\mathit{lnf}(M)$ the long normal form of M. It turns out that while algebraic reduction does not commute in general with η-reduction [BG89], it does "commute" with $\lambda^{\forall}$-conversion to long normal form, in the following sense.

Lemma 5.5
Let $r \in R$, and let M, N be two terms that type-check. If $M \xrightarrow{r} N$ then $\mathit{lnf}(M) \overset{r}{\twoheadrightarrow} \mathit{lnf}(N)$. Moreover, if $M \xrightarrow{r} N$ is actually an algebraic trunk reduction step then $\mathit{lnf}(M) \xrightarrow{r} \mathit{lnf}(N)$.

Proof. We consider first the general case (no restrictions on where the r-redex appears).

Let $r \equiv x_1{:}s_1, \ldots, x_n{:}s_n \rhd A \longrightarrow B : s$. Since $M \xrightarrow{r} N$, there exists a context with one hole $C[\ : s\ ]$ and a substitution φ such that $M \equiv C[A[\varphi]]$ and $N \equiv C[B[\varphi]]$. Let $P_i \overset{\text{def}}{=} \varphi(x_i)$ $(i = 1, \ldots, n)$. Then, we can write

$$M \equiv C[A[P_1/x_1, \ldots, P_n/x_n]] \qquad\qquad N \equiv C[B[P_1/x_1, \ldots, P_n/x_n]]$$

Without loss of generality, we can assume that the x_i's do not occur in M, N. Let us introduce the notation $\lambda\vec{x}{:}\vec{s}.\, D \overset{\text{def}}{=} \lambda x_1{:}s_1. \cdots. \lambda x_n{:}s_n.\, D$ where D is A or B. Then,

$$M \overset{\beta}{\twoheadleftarrow} M' \overset{\text{def}}{=} C[(\lambda\vec{x}{:}\vec{s}.\, A)P_1 \cdots P_n] \qquad\qquad N \overset{\beta}{\twoheadleftarrow} N' \overset{\text{def}}{=} C[(\lambda\vec{x}{:}\vec{s}.\, B)P_1 \cdots P_n]$$

Let z be a fresh variable of type $s_1 \to \cdots \to s_n \to s$. Then

$$M' \equiv C[z\, P_1 \cdots P_n][\lambda\vec{x}{:}\vec{s}.\, A \,/\, z] \qquad\qquad N' \equiv C[z\, P_1 \cdots P_n][\lambda\vec{x}{:}\vec{s}.\, B \,/\, z]$$

Let $Q \overset{\text{def}}{=} \mathit{lnf}(C[z\, P1 \cdots P_n])$. We claim that Q has the following property:

($*$) Any occurrence of z is at the head of a subterm of the form $z\,P'_1\cdots P'_n$ where P'_i has type s_i $(i=1,\ldots,n)$ and $z\,P'_1\cdots P'_n$ has type s (and thus cannot be further applied to terms or types).

Indeed, property ($*$) holds for $C[z\,P1\cdots P_n]$ and it is easy to check that it is preserved under β-reduction, $\mathcal{T}\beta$-reduction, and $\mathcal{T}\eta$-reduction. Moreover, while property ($*$) is not preserved under arbitrary η-expansions, it is preserved under the kind of η-expansion that are used to reach long normal form (see the proof of lemma 5.4). To see this, let Q' be a term of the form $\lambda\vec{v}.\,h\,T_1\cdots T_m$ and such that the type of $h\,T_1\cdots T_m$ is $\tau\rightarrow\tau'$, and assume that Q' has property ($*$). We can rule out the case $h\,T_1\cdots T_m\equiv z$ since by property ($*$) it implies that the type of z is a sort, and not $\tau\rightarrow\tau'$. For all the other possible occurrences of z it is easily seen that $\lambda\vec{v}.\,\lambda y{:}\,\tau.\,h\,T_1\cdots T_m\,y$ also has the property ($*$).

Let

$$M''\overset{\text{def}}{=}\beta nf(Q[\lambda\vec{x}{:}\,\vec{s}.\,A\,/\,z])\qquad\qquad N''\overset{\text{def}}{=}\beta nf(Q[\lambda\vec{x}{:}\,\vec{s}.\,B\,/\,z])\ .$$

We will show that M'' is in long normal form and since clearly M $\lambda^{\forall}$-converts to M'', we must have $M''\equiv lnf(M)$. Similarly, $N''\equiv lnf(N)$. With this, we need also show that $M''\overset{r}{\twoheadrightarrow}N''$. Both facts follow from the following claim.

Claim. If Z is a term in long normal form having property ($*$) then

$$X\overset{\text{def}}{=}\beta nf(Z[\lambda\vec{x}{:}\,\vec{s}.\,A\,/\,z])\qquad\qquad Y\overset{\text{def}}{=}\beta nf(Z[\lambda\vec{x}{:}\,\vec{s}.\,B\,/\,z])$$

are in long normal form and $X\overset{r}{\twoheadrightarrow}Y$.

The proof of the claim is by induction on the size of Z. Let $Z\equiv\lambda\vec{v}.\,h\,T_1\cdots T_m$. We distinguish two cases.

($h\not\equiv z$) Let D be A or B. Then, $\beta nf(Z[\lambda\vec{x}{:}\,\vec{s}.\,D\,/\,z])\equiv\lambda\vec{v}.\,h\,T'_1\cdots T'_m$ where $T'_j\overset{\text{def}}{=}T_j$ if T_j is a type expression and $T'_j\overset{\text{def}}{=}\beta nf(T_j[\lambda\vec{x}{:}\,\vec{s}.\,D\,/\,z])$ if T_j is a term. In the latter case, T_j is a long normal form of strictly smaller size than Z. Since property ($*$) is inherited by subterms, we can apply the induction hypothesis and the statement of the claim for Z follows easily.

($h\equiv z$) In this case, by property ($*$), $Z\equiv\lambda\vec{v}.z\,Z_1\cdots Z_n$ where Z_i has type s_i $(i=1,\ldots,n)$. Each of the Z_i's is a long normal form having property ($*$) and of strictly smaller size than Z so the induction hypothesis applies. Let

$$X_i\overset{\text{def}}{=}\beta nf(Z_i[\lambda\vec{x}{:}\,\vec{s}.\,A\,/\,z])\qquad Y_i\overset{\text{def}}{=}\beta nf(Z_i[\lambda\vec{x}{:}\,\vec{s}.\,B\,/\,z])\qquad(i=1,\ldots,n)$$

Consider $X'\overset{\text{def}}{=}\lambda\vec{v}.\,A[X_1/x_1,\ldots,X_n/x_n]$. A is an algebraic term, thus already in long normal form. By induction hypothesis, the X_i's are in long normal form and since their types are sorts, X' is in long normal form, in particular also in β-normal form. Since $Z[\lambda\vec{x}{:}\,\vec{s}.\,A\,/\,z]$ β-reduces to X' we have $X\equiv X'$. Similarly, $Y\equiv\lambda\vec{v}.\,B[Y_1/x_1,\ldots,Y_n/x_n]$ and Y is in long normal form. Moreover, by induction hypothesis $X_i\overset{r}{\twoheadrightarrow}Y_i$ $(i=1,\ldots,n)$, hence $X\overset{r}{\twoheadrightarrow}Y$.

This ends the proof of the claim and that of the first part of the lemma.

For the second part, we consider the restricted case in which the r-reduction is an algebraic trunk reduction. Using the same notation as before, we write again the reduction $M \xrightarrow{r} N$ as

$$C[A[P_1/x_1, \ldots, P_n/x_n]] \xrightarrow{r} C[B[P_1/x_1, \ldots, P_n/x_n]]$$

This being a trunk reduction however, the hole in the context ;$C[\ ]$ does not occur within a β, $\mathcal{T}\beta$, or $\mathcal{T}\eta$ redex. Consequently

$$\lambda^{-}nf(M) \equiv \lambda^{-}nf(C[A[P_1/x_1, \ldots, P_n/x_n]]) \equiv \lambda^{-}nf(C)[A[\lambda^{-}nf(P_1)/x_1, \ldots, \lambda^{-}nf(P_n)/x_n]]$$

and similarly for $\lambda^{-}nf(N)$. Because the type of the hole and those of the $\lambda^{-}nf(P_i)$'s are sorts and because A is already in long normal form, we further have

$$lnf(M) \equiv lnf(C)[A[lnf(P_1)/x_1, \ldots, lnf(P_n)/x_n]]$$

and similarly for $lnf(N)$. It follows that $lnf(M) \xrightarrow{r} lnf(N)$. □

We can now prove the claim stated at the beginning of this section.

Theorem 5.6 *" M is $\xrightarrow{\lambda^{\forall}R}$-strongly normalizing "*
is a candidate-closed property of terms M that type-check.

Proof. (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 5). (Cand 1) is immediate by the familiar kind of argument.[12] The verification of (Cand 5) is trivial. Checking (Cand 3) is a bit of work but the presence of algebraic rules makes no difference compared to theorems 4.9 and 4.11. The details of this verification can be found in [Gal90].

Checking (Cand 4) is an easier version of checking (Cand 3). The only 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, we have $k \leq n$. If $k < n$, the familiar kind of argument applies.[13]

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

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

Let $M' \equiv A'[\varphi']$ 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

[12]By the pigeonhole principle kind of argument used in the proof of theorem 3.10.
[13]By the pigeonhole principle kind of argument used in the proof of theorem 3.10.

is inside one of the $\varphi'(x'),\ \ x' \in FV(A')$. By the familiar kind of argument, one of these is not strongly normalizing.[14] However, by lemma 5.2, since every N_i is SN, every $\varphi'(x'),\ \ x' \in FV(A')$ is also SN, a contradiction.

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

In view of lemma 5.5, we convert all the terms of the infinite reduction sequence out of M to long normal form. Since there are infinitely many algebraic trunk rewrite steps, the result will be an infinite sequence of R-reductions. By theorem 3.10, this is impossible. Thus, in both cases, the assumption that there is an infinite reduction sequence from M leads to a contradiction, which implies that M is $\xrightarrow{\lambda^{\forall}R}$-SN. □

Finally, we obtain the main result of this paper.

Theorem 5.7 *(Conservation of Strong Normalization)*
If $\xrightarrow{R}$ is strongly normalizing on algebraic terms, then $\xrightarrow{\lambda^{\forall}R}$ is strongly normalizing on all terms that type-check.

Proof. Apply theorem 4.8 and theorem 5.6. □

6 Directions for Further Research

The results of this paper and those of [BG89] 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 has 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). However, one should be careful, the situation is more complex, as demonstrated by the following example due to Okada.

[14]By the pigeonhole principle kind of argument used in the proof of theorem 3.10.

Example 6.1
Let $f : s \rightarrow s \rightarrow s$ be a binary operation symbol (s is a sort), and consider the following higher-order rewrite rule

$$f\,(zx)\,x \xrightarrow{r} f\,(zx)\,(zx)$$

where $z : s \rightarrow s$ is a higher-order variable and $x : s$ is a first-order variable. To r-rewrite an algebraic term we allow the instantiation of z by terms of type $s \rightarrow s$ obtained by application from first-order variables and f. Clearly, $\xrightarrow{r}$ is SN on algebraic terms. However, we have the following infinite reduction if z is instantiated to $\lambda y{:}\, s.\, y$:

$$f\,((\lambda y{:}\, s.\, y)x)\,x \xrightarrow{r} f\,((\lambda y{:}\, s.\, y)x)\,((\lambda y{:}\, s.\, y)x) \xrightarrow{\beta} f\,((\lambda y{:}\, s.\, y)x)\,x \xrightarrow{r} \ldots$$

Thus, the interaction between β-conversion and higher-order algebraic rewriting seems quite subtle. Actually, it is not quite clear what is meant by algebraic rewriting in the presence of higher-order variables, and this should be investigated further. In any case, it would be interesting to find sufficient conditions on higher-order rewrite rules that would allow conservation results of the kind presented in this paper to hold.

References

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Volume 103 of *Studies in Logic and the Foundations of Mathematics*, North-Holland, Amsterdam, second edition, 1984.

[Bar89] F. Barbanera. Combining term rewriting and type assignment systems. Manuscript, to appear in Proc. Conf. Italian chapter of EATCS. 1989.

[BG89] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic confluence. Manuscript, submitted for publication. 1989.

[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.

[Gal90] J. Gallier. On Girard's "Candidats de Reductibilité.". In P. Odifreddi, editor, *Logic and Computer Science*, pages ??–??, Academic Press, New York, 1990.

[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.

[GS89a] J. Gallier and W. Snyder. Complete sets of transformations for general E-Unification. *Theoretical Computer Science*, 67:203–260, 1989.

[GS89b] J. Gallier and W. Snyder. Higher-order unification revisited: complete sets of transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.

[Hue75] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[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.

[Oka89] M. Okada. Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. Manuscript, to appear in Proc. ISSAC. 1989.

[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.