ELSEVIER

Theoretical Computer Science 170 (1996) 173–207

Theoretical Computer Science

Intersection type assignment systems with higher-order algebraic rewriting

Franco Barbanera[a,*], Maribel Fernández[b,1]

[a] *Dipartimento di Informatica, Corso Svizzera, 185, 10149 Torino, Italy*
[b] *DMI-LIENS (CNRS URA 1327), Ecole Normale Supérieure, 45, rue d'Ulm, 75005 Paris, France*

Received January 1994; revised November 1995
Communicated by C. Böhm

Abstract

General computational models obtained by integrating different specific models have recently become a stimulating and promising research subject. In particular, extensions of various λ-calculi with algebraic rewriting, either typed or untyped, have been deeply investigated. In the present paper this subject is addressed from the point of view of *type assignment*. A powerful type assignment system, the one based on intersection types, is extended with first- and higher-order algebraic rewriting, and relevant properties, like strong normalization and completeness, are proved.

1. Introduction

The interactions between term rewriting systems and λ- calculus have been widely studied over the last few years [5, 7, 10, 11, 17, 24, 26, 31]. An important motivation of these works is the need of establishing a formal setting integrating two closely-related models of computation: the one based on rewriting of algebraic terms, and the one based on β-reduction of λ-terms. Since these two models are based on term reduction, the most natural model including both of them is given by the combination of the two kinds of reductions on mixed terms, i.e. on algebraic-λ-terms. An interesting question to investigate is under which conditions the properties of each reduction relation transfer to their combination, or in other words, under which conditions the properties are *modular*.

If we consider untyped languages, the attempt to "amalgamate" algebraic reductions with β-reduction raises serious problems. Klop [26] showed that in order to add a confluent term rewriting system to the untyped λ-calculus preserving confluence, restrictions have to be imposed on the term rewriting system. If such restrictions

* Corresponding author. e-mail: barba@di.unito.it.
[1] Part of this work was done while the author was at LRI, Université de Paris-Sud.

PII S0304-3975(96)00263-4

are omitted, confluence and strong normalization (also called termination) of the combination hold only if restrictions are imposed on the set of mixed terms, as shown by Dougherty [17]. On the other hand, if we consider typed languages things work out nicely: confluence and termination are modular properties in this case. This result is shown by Tannen and Gallier [11] and Okada [31] for the polymorphic second-order λ-calculus extended with first-order algebraic rewriting, using the computability predicate method developed by Tait and Girard (see e.g. [20]).

Combinations of algebraic rewriting and typed λ-calculi have been studied also as an interesting alternative in the design of new programming languages. The *algebraic-functional paradigm* defined by Jouannaud and Okada [24] allows first-order definitions of abstract data types and operators, as in equational languages like OBJ [19], and the full power of λ-calculus for defining functions of higher types, as in functional languages like ML [21]. The following example, taken from [24], shows a simple program that cannot be written in one of the above-mentioned traditional languages alone, but can be written in the unified language.

$$\mathsf{append}\ \ \mathsf{nil}\, l = l$$

$$\mathsf{append}\,(\mathsf{cons}\ x\ l)\, l' = \mathsf{cons}\, x\,(\mathsf{append}\ l\ l')$$

$$\mathsf{append}\,(\mathsf{append}\ \ l l')\, l'' = \mathsf{append}\ l\,(\mathsf{append}\ l'\ l'')$$

$$\mathsf{map}\, X\ \mathsf{nil} = \mathsf{nil}$$

$$\mathsf{map}\, X\,(\mathsf{cons}\, x\ l) = \mathsf{cons}\,(X\, x)(\mathsf{map}\, X\ l)$$

Here, the first-order function **append** is defined algebraically (the third rule establishes the associativity of append on lists, and makes the definition nor primitive recursive), while the higher-order function **map** is defined recursively on the structure of lists.

Adding λ-expressions to this kind of *higher-order* rewrite systems (i.e. allowing to substitute λ-terms for higher-order variables) we obtain the usual paradigm of functional programming languages extended with algebraic definitions of operators and types. This is the essence of the algebraic functional paradigm: the combination of first-order algebraic equations and λ-calculus enriched with higher-order functional constants defined by higher-order equations.

Jouannaud and Okada [24] showed that algebraic functional languages built upon typed versions of λ-calculus (simply typed λ-calculus or polymorphic λ-calculus) are modular with respect to termination, provided the set of higher-order rules satisfies certain conditions (the so-called *general schema*) and the first-order rules are *non-duplicating* (that is, the number of occurrences of any variable in the right-hand side of a rule is less than or equal to the number of occurrences in the left-hand side[2]). In the above-mentioned work the authors left open the question of whether other type disciplines of λ-calculus interact as nicely with "wall-behaved" term rewriting systems.

[2] Although not explicitly indicated in [24], this restriction is necessary for the modularity of termination.

In this paper we address the problem of the modularity of strong normalization and completeness (strong normalization + confluence) of algebraic functional languages in a *type assignment* setting. Type assignment systems have several advantages in practice and are used in powerful functional programming languages, notably ML and Miranda[3] [40].

More precisely, we consider the extension of the intersection type assignment system for λ-calculus [9] (which is a very expressive and powerful system, since it can type all strongly normalizable λ-terms) with first- and higher-order algebraic rewriting. We prove that strong normalization is modular when first-order rules are non-duplicating and higher-order rules satisfy the general schema. We then show that when the higher-order rules do not introduce critical pairs, also completeness is modular. Our proof of strong normalization is based on the computability predicate method. For completeness we give a simple proof using the strong normalization result and Newman's Lemma.

Finally, we show that termination and completeness are modular properties also in combinations of *many-sorted* first- and higher-order rewrite systems with the intersection type assignment system for the λ-calculus. This result can be seen, on the one hand, to be in the direction of extending [24] because of the introduction of the intersection type assignment system, and, on the other hand, as a generalization of [5] by taking into account higher-order rewrite systems. Also, this is the first step in the proof of modularity of strong normalization in the algebraic extension of the Calculus of Constructions [7].

In Section 2 we present the definition of the intersection type assignment systems for the algebraic extensions of the λ-calculus. In Section 3 we prove Subject Reduction, and in Section 4 modularity of strong normalization and completeness. The conclusions are contained in Section 5. In the appendix we consider the case in which a many-sorted term-rewriting system is added to the intersection type assignment system.

Results similar to those presented here, but without considering higher-order rewriting, were proved in [5]. Ref [6] contains a preliminary version of the results of the present paper.

2. Systems $\vdash_R^{\wedge}$

Type assignment systems (also called type inference systems) are formal systems for assigning types to untyped terms. These systems are defined by specifying a set of terms, a set of types one assigns to terms, and a set of type inference rules. The rules are usually given in a natural deduction style. Here, we use a slight variation of the standard presentation, in order to keep track of the premises statements depend on. We shall refer to [23] for more details.

[3] Miranda is a trade mark of Research Software LTD.

The type assignment systems $\vdash_R^{\wedge}$ that we are going to define are algebraic extensions of a λ-calculus system with intersection types. Type assignment systems with intersection types for λ-calculus were originally devised in [13, 15], and deeply investigated afterwards in several papers, among which we recall [9, 37, 14, 1]. We refer to those papers and to the surveys [12, 22] for motivations and applications of intersection types.

We begin the description of system $\vdash_R^{\wedge}$ by considering a set $\mathscr{S}$ of *sorts* (i.e. names of domains), a set $\mathscr{V}$ of *type variables*, and a set $\mathscr{F}$ of (untyped) *function symbols*. Each function symbol is equipped with an *arity*, denoted by superscripts when not clear from the context.

Definition 1 (*Types*). The set $\mathscr{T}_{\wedge\mathscr{S}}$ of types of $\vdash_R^{\wedge}$ is defined inductively as follows:
- Constant types (sorts): if $s \in \mathscr{S}$ then $s \in \mathscr{T}_{\wedge\mathscr{S}}$.
- Type variables: If $\varphi \in \mathscr{V}$ then $\varphi \in \mathscr{T}_{\wedge\mathscr{S}}$.
- Arrow types: If $\sigma, \tau \in \mathscr{T}_{\wedge\mathscr{S}}$ then $\sigma \to \tau \in \mathscr{T}_{\wedge\mathscr{S}}$.
- Intersection types: If $\sigma, \tau \in \mathscr{T}_{\wedge\mathscr{S}}$ then $\sigma \wedge \tau \in \mathscr{T}_{\wedge\mathscr{S}}$.

A type is *algebraic* if it contains neither the type constructor $\wedge$ nor type variables. We denote by $\mathscr{T}_{\mathscr{S}}$ the set of algebraic types. The set of *base types* of $\mathscr{T}_{\wedge\mathscr{S}}$ is $\mathscr{S} \cup \mathscr{V}$.

We will consider types modulo associativity, commutativity and idempotency of the type operator $\wedge$. From the properties of $\wedge$ it is easy to deduce:

Lemma 1. *For each type ρ there exists a finite set I, integers n_i, types $\sigma_j^{(i)}$ $(0 \leqslant j \leqslant n_i)$ and τ_i $(i \in I)$ such that $\rho = \bigwedge_{i\in I}(\sigma_1^{(i)} \to \sigma_2^{(i)} \to \cdots \to \sigma_{n_i}^{(i)} \to \tau_i)$, where for all $i \in I, \tau_i$ is not an intersection type if $n_i = 0$.*

Definition 2 (*Terms*). The (untyped) terms of $\vdash_R^{\wedge}$ are defined by the following grammar:

$$\Lambda_{\mathscr{F}} ::= \mathbf{x} \mid \mathbf{f} \mid (\Lambda_{\mathscr{F}} \Lambda_{\mathscr{F}}) \mid \lambda\mathbf{x}.\Lambda_{\mathscr{F}}$$

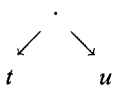
where $\mathbf{x}$ ranges over a set $\mathscr{X}$ of (untyped) term variables and $\mathbf{f}$ over $\mathscr{F}$.

Terms are then untyped λ-terms with constants and on them the usual notion of β-reduction is defined. We work modulo α-conversion as usual, i.e. terms that only differ in their bound variables are considered to be equal. In fact, we work with equivalence classes of terms (or representatives of such classes). For practical purposes we shall only work with representants in which all bound variables are pairwise distinct and different from the free variables. The set of free variables of a term t will be denoted by $FV(t)$.

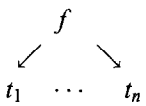
As usual terms can be viewed as trees. For example, the abstraction $\lambda x.t$ can be represented by

$$\begin{array}{l} \lambda x \\ | \\ t \end{array}$$

and the application tu (when t is not an element of $\mathscr{F}$) by



For terms of the form $ft_1 \cdots t_n$ ($f \in \mathscr{F}$) we will use the traditional representation where the root of the tree is f and the subterms are the trees corresponding to $t_1, \ldots, t_n$:



Subterms will be denoted by the so-called *positions*, which are sequences of natural numbers denoting the path from the root of the term to the subterms. The empty sequence (root position) is denoted by ε. The subterm of t at position p is denoted by $t|_p$ and $t[u]_p$ is the result of replacing the subterm of t at position p by u.

Definition 3. (i) A statement is an expression of the form $M : \sigma$ where $\sigma \in \mathscr{T}_{\wedge\mathscr{S}}$ and $M \in \Lambda_{\mathscr{F}}$. M is the *subject* of the statement.

(ii) A *basis* (the set of assumptions a statement depends on) is a set of statements with only variables as subjects. Moreover, there are no two statements with the same subject. If x does not occur in the basis B, then $B, x : \sigma$ denotes the basis $B \cup \{x : \sigma\}$. A basis is *algebraic* if all the types occurring in it are so. For B and B' bases, $B \subseteq_{\wedge} B'$ if for all $x : \sigma \in B$ there exists $x : \sigma' \in B'$ such that $\sigma' \equiv \sigma \wedge \tau$ for some τ.

(iii) A *set $\mathscr{AX}_{\mathscr{S}\mathscr{F}}$ of axiom statements* relative to a set $\mathscr{S}$ of sorts and a set of constants $\mathscr{F} = \{f_1, f_2, \ldots, f_n\}$, is a set of statements of the form

$$\mathscr{AX}_{\mathscr{S}\mathscr{F}} = \{f_1 : \sigma_1, f_2 : \sigma_2, \ldots, f_n : \sigma_n\}$$

where σ_i belongs to $\mathscr{T}_{\mathscr{S}}(1 \leqslant i \leqslant n)$ and it is of the form $\tau_1 \to \cdots \to \tau_m \to \tau$ where m is the arity of f_i. We shall often omit, when clear, the subscripts of $\mathscr{AX}_{\mathscr{S}\mathscr{F}}$.

Given a set of axiom statements $\mathscr{AX}$, a function symbol $f \in \mathscr{F}$ is a *first-order function symbol* (with respect to $\mathscr{AX}$) if $f : s_1 \to \cdots \to s_m \to s \in \mathscr{AX}$ with $s_1, \ldots, s_m, s \in \mathscr{S}$, and m is the arity of f. Otherwise, f is called a *higher-order function symbol* (with respect to $\mathscr{AX}$). First-order functions operate on sort types only: they have inputs of sort types, and give values of sort type as result. Higher-order functions, also called "functionals", can, instead, manipulate objects that have an arrow type. We denote by Σ the set of all the first-order function symbols and by f, g its elements. We use F, G to denote higher-order function symbols. When it is clear from the context, we use f to denote a generic (first- or higher-order) element of $\mathscr{F}$.

Given a term M, a function symbol f is called *saturated* in M if each occurrence of f in M is applied at least to m arguments, with m the arity of f.

Definition 4 (*Inference rules*). Let $\mathscr{S}$ be a set of sorts, $\mathscr{F}$ a set of function symbols and $\mathscr{AX}$ a set of axiom statements. Let $\sigma, \tau \in \mathscr{T}_{\wedge,\mathscr{S}}$.

$$(Var) \quad B, x : \sigma \vdash^{\wedge}_{R} x : \sigma$$

$$(Ax) \quad B \vdash^{\wedge}_{R} f : \sigma \quad \text{for any } f : \sigma \in \mathscr{AX}$$

$$(\rightarrow I) \quad \frac{B, x : \sigma \vdash^{\wedge}_{R} M : \tau}{B \vdash^{\wedge}_{R} \lambda x.M : \sigma \rightarrow \tau}$$

$$(\rightarrow E) \quad \frac{B \vdash^{\wedge}_{R} M : \sigma \rightarrow \tau \qquad B \vdash^{\wedge}_{R} N : \sigma}{B \vdash^{\wedge}_{R} (MN) : \tau}$$

$$(\wedge I) \quad \frac{B \vdash^{\wedge}_{R} M : \sigma \qquad B \vdash^{\wedge}_{R} M : \tau}{B \vdash^{\wedge}_{R} M : \sigma \wedge \tau}$$

$$(\wedge E) \quad \frac{B \vdash^{\wedge}_{R} M : \sigma \wedge \tau}{B \vdash^{\wedge}_{R} M : \sigma}$$

The sets $\mathscr{S}, \mathscr{F}$ and $\mathscr{AX}$ are parameters system $\vdash^{\wedge}_{R}$ depends on. In what follows, we shall assume $\mathscr{S}, \mathscr{F}$ and $\mathscr{AX}$ to be given.

A term M will be called *typeable* if there exists a basis B and a type σ such that $B \vdash^{\wedge}_{R} M : \sigma$ is derivable by means of the above inference rules. We shall denote by $\Lambda_{\wedge R}$ the set of typeable terms. We shall express the fact that is possible to derive $B \vdash^{\wedge}_{R} M : \sigma$ in system $\vdash^{\wedge}_{R}$ by simply stating $B \vdash^{\wedge}_{R} M : \sigma$.

Lemma 2.

$$[B \vdash^{\wedge}_{R} M : \sigma, B \subseteq_{\wedge} B'] \Rightarrow B' \vdash^{\wedge}_{R} M : \sigma.$$

Proof. It is easy to obtain a derivation of $B' \vdash^{\wedge}_{R} M : \sigma$ from one of $B \vdash^{\wedge}_{R} M : \sigma$ by means of several applications of rule $(\wedge E)$. □

It is immediate to check that, in case one disregards rule (Ax) and does not consider sorts in types, the system described above is one of the fundamental systems with intersection types present in the literature. We denote by $\vdash^{\wedge}$ such a system. The set of terms typeable in $\vdash^{\wedge}$ will be called $\Lambda_{\wedge}$. System $\vdash^{\wedge}$ enables to characterize λ-terms having a relevant functional property, namely β-strong normalization.

Theorem 1 ([37, 28, 1, 2]). *Let M be a term of the pure λ-calculus:*

$$M \text{ is } \beta\text{-strongly normalizable} \Leftrightarrow M \in \Lambda_{\wedge}.$$

It is easy to check that from the above theorem the following lemma follows for $\vdash^{\wedge}_{R}$.

Lemma 3. *Let $M \in \Lambda_{\mathscr{F}}$:*

$$M \in \Lambda_{\wedge R} \Rightarrow M \text{ is } \beta\text{-strongly normalizable.}$$

This is because if we deal only with β-reduction, function symbols can be looked at as variables. Note that, since the function symbols can be given only a particular type (depending on $\mathscr{AX}$), the reverse of the lemma above does not hold.

We now complete the specification of systems $\vdash_R^{\wedge}$, by defining the notion of *algebraic rewriting*.

Definition 5 ($\wedge$-*algebraic terms*). Let $t \in \Lambda_{\mathscr{F}}$.

(i) t is a *quasi-$\wedge$-algebraic* term if it is built up using only variables and function symbols (i.e. without λ-abstractions).

(ii) t is a $\wedge$-*algebraic* term if

1. it is quasi-$\wedge$-algebraic,
2. any $f \in \mathscr{F}$ is saturated in t.

(iii) t is a *first-order* quasi-$\wedge$-algebraic term if it is quasi-$\wedge$-algebraic and

1. any $f \in \mathscr{F}$ occurring in t is first-order,
2. there is no subterm of t of the form xP.

Otherwise, t will be called *higher-order*.

We will denote by $\Lambda_{\wedge R}^{\mathrm{Alg}}$ the restriction of $\Lambda_{\wedge R}$ to $\wedge$-algebraic terms.

Definition 6 (*Rewritable terms*). A term t is *rewritable* if it is $\wedge$-*algebraic* and

(1) there exists an algebraic basis B and $\sigma \in \mathscr{T}_{\mathscr{S}}$ such that $B \vdash_R^{\wedge} t : \sigma$,

(2) for any $x \in FV(t)$ there exists a subterm $fP_1 \cdots P_k$ of t such that $f \in \mathscr{F}$ and $P_j \equiv x$ for some $1 \leqslant j \leqslant k$,

(3) t is not of the form xP.

It is straightforward to check that a variable cannot be rewritable algebraic term. Moreover, by definition of rewritable term, and since function symbols have algebraic (ground) types, it is easy to show the following.

Lemma 4. *Given a* rewritable *term t, there exists a unique (minimal with respect to $\subseteq_{\wedge}$) algebraic basis A_t and a unique algebraic type σ_t^{Alg} such that $A_t \vdash_R^{\wedge} t : \sigma_t^{\mathrm{Alg}}$, and*

$$B \vdash_R^{\wedge} t : \tau \;\Rightarrow\; [A_t \subseteq_{\wedge} B \;\&\; \tau \equiv \sigma_t^{\mathrm{Alg}}].$$

The above lemma shows that rewritable algebraic terms are somewhat independent from bases, so justifying (1) of Definition 6. Moreover, it is possible, in a rewritable term, to define the notions of the first- and higher-order variables. Let t be a rewritable term and $x \in FV(t)$; x is said to be first-order variable in t if the type of x in A_t belongs to $\mathscr{S}$.

We can now use the notion of rewritable term to define rewrite rules. We introduce first the notion of λ-rewrite rule and then that of (proper) rewrite rule. The former notion, more general of the usual one of algebraic reduction, is given since our results will hold also for a particular class of λ-rewrite rules.

Definition 7. A *λ-rewrite rule* is a pair of terms $\langle t, t' \rangle$ where t is $\wedge$-algebraic while t' can also contain abstractions, and such that

1. t is rewritable,[4]
2. $FV(t') \subseteq FV(t)$,
3. Let A_t be the unique (minimal) algebraic basis and σ the unique algebraic type such that $A_t \vdash_R^{\wedge} t : \sigma$ (such A_t and σ exist by Lemma 4), then $A_t \vdash_R^{\wedge} t' : \sigma$.

A (proper) rewrite rule is a λ-rewrite rule involving only $\wedge$-algebraic terms.

Definition 8 (*Rewrite rules*). (i) A *rewrite rule* is a λ-rewrite rule $\langle t, t' \rangle$ such that also t' is $\wedge$-algebraic.

(ii) A *first-order rewrite rule* is a rewrite rule $\langle t, t' \rangle$ such that both t and t' are first-order $\wedge$-algebraic terms.

(iii) A *higher-order rewrite rule* is a rewrite rule which is not first-order.

In the following $r : t \to t'$ denotes a $(\lambda-)$rewrite rule. Given a set R of rewriting rules, we denote by *FOR* and *HOR* the subsets of first-order and higher-order rules of R, respectively. When λ-rewrite rules are present, we assume them to be in the set *HOR*.

Before introducing the reduction relations for system $\vdash_R^{\wedge}$ we have to fix more notation: we shall denote substitutions by $\{x_1 \mapsto N_1, \ldots, x_n \mapsto N_n\}$,[5] postfix notation will be used for their application, e.g. $M\{x_1 \mapsto N_1, \ldots, x_n \mapsto N_n\}$ denotes the simultaneous substitution of N_i for $x_i (1 \leqslant i \leqslant n)$ in M. When applying a substitution to a λ-term we must take care of bound variables, as usual. We will use the symbol ϕ to denote a generic substitution, and $dom(\phi)$ will denote the domain of ϕ.

A rewrite rule induces a rewriting relation on $\Lambda_{\wedge R}$ as follows.

Definition 9 (*Algebraic rewriting*). Let $r : t \to t'$ be a rewrite rule. The reduction relation $\to_r$ on $\Lambda_{\wedge R}$ is defined as follows: $M \to^r M'$ if $M \equiv C[t\phi]$, for some context $C[-]$ and substitution ϕ, and $M' \equiv C[t'\phi]$.

The definition of the reduction relations *only for typeable terms* is crucial for strong normalization to hold. If we had defined algebraic reductions without taking care of the typability of terms we could have terms in $\Lambda_{\mathscr{F}}$ not strongly normalizable even if *FOR* and *HOR* are strongly normalizing systems: it is in fact easy to find strongly normalizing many-stored term rewrite systems which fail to be strongly normalizing if we do not take types into consideration, i.e. if all sorts are identified; see [39] for examples.

The Subject Reduction Theorem (Theorem 3 below) will guarantee that the definition of algebraic rewriting is correct.

Given a set R of rewrite rules we will write $M \to_R M'$ if, for some $r \in R$, $M \to_r M'$.

[4] As stated before, this condition subsumes the usual condition "t is not a variable".

[5] We follow the notation of [16]

As usual, $\rightarrow_{R\beta}$ denotes the union of the reduction relations $\rightarrow_R$ and $\rightarrow_\beta$, and $\rightarrow^*_{R\beta}$ its reflexive and transitive closure.

Many definitions of *higher-order rewriting* appear in the literature; none of them has been universally accepted, and ours is one among the many possible ones. Recently, a general definition of higher-order rewriting has been proposed by van Oostrom and van Raamsdonk [32, 33], which subsumes systems like Klop's CRSs [27] and Nipkow's HRSs [30]. Even if our higher-order rewriting rules can be looked at from van Oostrom and van Raamsdonk's point of view, it is not clear whether it is so also for the whole systems $\vdash^{\wedge}_R$.

Summing up what we have defined above, system $\vdash^{\wedge}_R$ is then completely specified by a quadruple

$$\vdash^{\wedge}_R = \langle \mathscr{S}, \mathscr{F}, \mathscr{AX}, R \rangle$$

where $\mathscr{S}$ is a set of sorts, $\mathscr{F}$ a set of function symbols, $\mathscr{AX}$ a set of axiom statements relative to $\mathscr{S}$ and $\mathscr{F}$, and R as set of rewrite rules.

As mentioned in the introduction, in this paper we are concerned with termination and confluence properties of system $\vdash^{\wedge}_R$. For what concerns termination, however, if unrestricted higher-order rewrite rules are considered, even if terminating, it can be easily shown that this property fails. For example, let HOR be the set

$$\{r\colon\ f(Xx)xX \rightarrow f(Xx)(Xx)X\}.$$

For such terminating set of rules $\vdash^{\wedge}_R$ is not strongly normalizing:

$$\begin{aligned} f((\lambda y.y)x)x(\lambda y.y) &\rightarrow_r f((\lambda y.y)x)((\lambda y.y)x)(\lambda y.y) \\ &\rightarrow_\beta f((\lambda y.y)x)x(\lambda y.y) \end{aligned}$$

In order to guarantee strong normalization of $\rightarrow_{R\beta}$ we will only allow definitions of higher-order functions by a generalization of primitive recursion, called *general schema* [24]. We will present a simplified version of the original schema of Jouannaud and Okada. In the definition of the general schema we will use the notation $t[\vec{v}]$ to indicate that t is a term and $v_1, \ldots, v_n$ are subterms of t. so $t[\vec{v}]$ is the same as t, the notation only makes appear explicitly some of the subterms of t. We will also assume that there are no mutually recursive definitions of higher-order functions.

Definition 10 (*The general schema*). (i) A λ-higher-order rewrite rule $r : t \rightarrow t'$ satisfies *the general schema* if it is of the form

$$F\vec{l}[\vec{X},\vec{x}]\vec{Y} \rightarrow v[(F\vec{q}_1[\vec{X},\vec{x}]\vec{Y}),\ldots,(F\vec{q_m}[\vec{X},\vec{x}]\vec{Y})]$$

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

(1) $\vec{X} \subseteq \vec{Y}$,

(2) F is a higher-order function symbol not appearing in the sequences of terms $\vec{l}, \vec{q_1}, \ldots, \vec{q_m}$, and its occurrences in v are only the ones explicitly indicated,

(3) $\vec{l}, \vec{q_1}, \ldots, \vec{q_m}$ are terms with variables in $\vec{X}, \vec{x}$,

(4) $\forall i \in \{1..m\}$, $\vec{q_i} \lhd_{\text{mul}} \vec{l}$ where $\lhd$ denotes strict subterm ordering and $\lhd_{\text{mul}}$ its multiset extension, defined as usual. (If $<$ is a partial ordering on S, then the ordering $<_{\text{mul}}$ on multisets of elements of S is the transitive closure of the replacement of an element with any finite number, including zero, of elements that are smaller under $<$.)

(ii) A set *HOR* of higher-order rewrite rules *satisfies the general schema* if

(1) each rule $r \in HOR$ satisfies the general schema,

(2) there is no mutual recursion.

Condition (ii)(2) guarantees that the higher-order function symbols in R can always be indexed $(F_1, \ldots, F_n)$ in such a way that if i is the index of the F in

$$F\vec{l}[\vec{X}, \vec{x}]\vec{Y} \to v[(F\vec{q_1}[\vec{X}, \vec{x}]\vec{Y}, \ldots, (F\vec{q_m}[\vec{X}, \vec{x}]\vec{Y})],$$

then the indexes of the other function symbols in the rule are strictly less than i. In the following, we shall consider only such sort of indexing for higher-order symbols.

Although restricted, the general schema is interesting from a practical point of view: it allows the introduction of functional constants of higher-order types by primitive recursion on a first-order data structure. Let us show some examples.

Example 1. Consider the signature of lists, with constructors cons and nil. The function append (that concatenates two lists) can be defined by a set *FOR* of first-order rules:

$$\mathsf{append}\,\mathsf{nil}\,l \to l$$

$$\mathsf{append}\,(\mathsf{cons}\,x\,l)\,l' \to \mathsf{cons}\,x\,(\mathsf{append}\,l\,l')$$

$$\mathsf{append}\,(\mathsf{append}\,l\,l')\,l'' \to \mathsf{append}\,l\,(\mathsf{append}\,l'\,l'')$$

The functional map, which applies a function to all the elements of a list, can be defined recursively on the first-order structure of lists:

$$\mathsf{map}\,X\,\mathsf{nil} \to \mathsf{nil}$$

$$\mathsf{map}\,X\,(\mathsf{cons}\,x\,l) \to \mathsf{cons}\,(X\,x)\,(\mathsf{map}\,X\,l)$$

This definition satisfies the general schema.

We will show another example using lists:

Example 2. foldr is a very useful higher-order function, whose informal meaning is the following: Let $\langle x_1, \ldots x_n \rangle$ denote the list containing the elements $x_1, \ldots, x_n$; then

$$\mathsf{foldr}\,a\langle x_1, \ldots x_n\rangle f = f x_1 (f x_2 (\cdots (f x_n a) \cdots))$$

where f is a function and a is a constant.

It is easy to define **foldr** by a set of higher-order rules satisfying the general schema:

$$\mathsf{foldr}\, a\, \mathsf{nil}\, X \to a$$

$$\mathsf{foldr}\, a\, (\mathsf{cons}\, x\, l)\, X \to X\, x\, (\mathsf{foldr}\, a\, l\, X)$$

Now, using **foldr**, and assuming that $+, \times, 0$ and 1 are already defined, we can define the functions

$$\mathsf{sum} \to \mathsf{foldr}\ +\ 0$$

$$\mathsf{product} \to \mathsf{foldr}\ \times\ 1$$

The function **sum** adds the elements of a list of numbers, while **product** multiplies them. Moreover, assume that **append** is defined as in the previous example, then we can define

$$\mathsf{concat} \to \mathsf{foldr}\, \mathsf{append}\, \mathsf{nil}$$

The function **concat** concatenates a list of lists into one long list.

The higher-order rewrite rules defining **foldr**, **sum**, **product** and **concat** satisfy the general scheme, then, as a consequence of the "main theorem" that we will prove later, the union of the above-defined rewrite systems is strongly normalizing.

We have seen that unrestricted terminating higher-order rewrite rules could prevent strong normalization of $\to_{R\beta}$ to hold, which led us to impose some restrictions on *HOR*. This, however, is not sufficient. In fact, without imposing a restriction also on first-order rewrite rules, termination can fail even on terms in $\Lambda^{\mathrm{Alg}}_{\wedge R}$. It is, in fact, possible to code Toyama's example of non-termination [39], as it can be seen below.

Example 3. Consider the system $FOR \cup HOR$:

$$FOR = \{f01x \to fxxx\}$$
$$HOR = \{FX \to 1, FX \to 0\}$$

where F is a higher-order function symbol, f is a first-order function symbol, 0,1, are first-order constants and X,x are variables. *FOR* is terminating and *HOR* satisfies the general scheme; however, there is an infinite reduction sequence out of the term $f(FX)(FX)(FX)$:

$$f(FX)(FX)(FX) \to f0(FX)(FX) \to f01(FX) \to f(FX)(FX)(FX) \cdots$$

The cause of non-termination in the above example is that the rule in *FOR* is *duplicating*, i.e. there are more occurrences of the variable x on its right-hand side than on its left-hand side. We will show that the restriction of *FOR* to (terminating) non-duplicating rules, together with the general schema condition for *HOR*, imply Strong normalization in system $\vdash^{\wedge}_{R}$.

Definition 11. (i) A rewrite rule $r : t \to t'$ is *non-duplicating* if for any variable x the number of its occurrences in t is greater than or equal to the number of occurrences in t'.

(ii) A set of rewrite rules in non-duplicating if each of them is so.

Example 4. The first-order system defining the function append in Example 1 is non-duplicating.

The restriction to non-duplicating rules was first proposed by Rusinowitch [38] to ensure modularity of termination of disjoint unions of first-order term rewriting systems.

Definition 12 (*Safeness*). A set of rewrite rules $R = FOR \cup HOR$ is *safe* if

1. FOR is non-duplicating and strongly normalizing on first-order terms in $\Lambda^{\mathrm{Alg}}_{\wedge R}$.
2. HOR satisfies the general schema.

A system $\vdash^{\wedge}_{R}$ is safe if its set of rewrite rules is so.

The safeness condition will be shown to imply strong normalization for $\vdash^{\wedge}_{R}$. We extend now the notion of safeness in order to obtain a condition sufficient to obtain completeness (strong normalization + confluence) for $\vdash^{\wedge}_{R}$. We first recall the notion of critical pair.

Definition 13. If $l \to r$ and $s \to t$ are two rewrite rules (we assume that the variables of $s \to t$ were renamed so that there is no common variable with $l \to r$), p is the position of a non-variable subterm of s and μ is a most general unifier of $s|_p$ and l, then $(t\mu, s\mu[r\mu]_p)$ is a *critical pair* formed from those rules. Note that $s \to t$ may be a renamed version of $l \to r$. In this case a superposition in the root position is not considered a critical pair.

Thus, a critical pair arises from a most general non-variable overlap between two left-hand sides. Overlaps of higher-order variables applied to some arguments do generate critical pairs (these are non-variable overlaps due to the application operator).

Example 5. Consider the β-rule: $(\lambda x.M)N \to M\{x \mapsto N\}$.[6] If a rule r in HOR contains the term Xt (where X is a higher-order variable and t is an arbitrary term) as a subterm of the left-hand side – for instance, consider $r : F(X0) \to X$ – then there is a critical pair between r and β: $(\lambda x.M, F(M\{x \mapsto 0\}))$.

Definition 14 (*Complete safeness*). A set of rewrite rules $R = FOR \cup HOR$ is *completely safe* if

(1) it is safe, and

[6] This is actually a "meta-rule", or a rule schema. Although one cannot write this rule in HOR, it is possible to compute the critical pairs generated by the superpositions of this rule scheme on the left-hand sides of HOR.

(2) *FOR* is locally confluent on $\Lambda^{\mathrm{Alg}}_{\wedge R}$, and *HOR* does not introduce critical pairs (i.e. there is no critical pair between rules of *HOR*, between *FOR* and *HOR*, between β and *HOR*[7])

We shall prove in the rest of the paper that safeness is a sufficient condition for strong normalization. Using this result we will deduce in a straightforward manner that complete safeness is sufficient for completeness.

Theorem 2 (Main Theorem). *Let $\vdash^{\wedge}_R$ be the system defined by $\langle \mathscr{S}, \mathscr{F}, \mathscr{AX}, R\rangle$.*
(i) *R is safe $\Rightarrow$ $\rightarrow_{R\beta}$ is strongly normalizing on $\Lambda_{\wedge R}$.*
(ii) *R is completely safe $\Rightarrow$ $\rightarrow_{R\beta}$ is complete on $\Lambda_{\wedge R}$.*

3. The generalized subject reduction theorem

In this section we show an important property of $\rightarrow_{R\beta}$, the *Subject Reduction* property, stating that $\rightarrow_{R\beta}$ preserves types. In order to prove this theorem we first need some technical lemmas.

Lemma 5. *If we get $B \vdash^{\wedge}_R M : \tau$ from $B \vdash^{\wedge}_R M : \tau_1, \ldots, B \vdash^{\wedge}_R M : \tau_n$ $(n > 0)$ using only rules $(\wedge I)$ and $(\wedge E)$, then $\tau \equiv \bigwedge_{i\in I} \rho_i$ where all ρ_i $(i \in I)$ are not intersections and for all $i \in I$ there is some $1 \leqslant j \leqslant n$ such that either $\tau_j \equiv \rho_i$ or $\tau_j \equiv \rho_i \wedge \sigma_j$ for some σ_j.*

Proof. Easy, by the fact that only rules $(\wedge I)$ and $(\wedge E)$ are used after $B \vdash^{\wedge}_R M : \tau_1, \ldots, B \vdash^{\wedge}_R M : \tau_n$. □

Lemma 6. *If $B \vdash^{\wedge}_R MN : \bigwedge_{i\in I} \rho_i$ (where, for any $i \in I$, ρ_i is not on intersection type) then for all $i \in I$ there exist μ_i and σ_i such that $B \vdash^{\wedge}_R N : \mu_i$ and either $B \vdash^{\wedge}_R M : \mu_i \rightarrow \rho_i$ or $B \vdash^{\wedge}_R M : \mu_i \rightarrow \rho_i \wedge \sigma_i$ for some σ_i.*

Proof. Any derivation of $B \vdash^{\wedge}_R MN : \bigwedge_{i\in I} \rho_i$ can be decomposed into
(a) n derivations $(n > 0)$ of $B \vdash^{\wedge}_R MN : \tau_1, \ldots, B \vdash^{\wedge}_R MN : \tau_n$, each of them terminating with an application of rule $(\rightarrow E)$;
(b) a derivation of $B \vdash^{\wedge}_R MN : \bigwedge_{i\in I} \rho_i$ from $B \vdash^{\wedge}_R MN : \tau_1, \ldots, B \vdash^{\wedge}_R MN : \tau_n$ using only rules $(\wedge I)$ and $(\wedge E)$ in between.
Then, for all τ_j there exists μ_j such that $B \vdash^{\wedge}_R M : \mu_j \rightarrow \tau_j$ and $B \vdash^{\wedge}_R N : \mu_j$. The thesis now follows from Lemma 5. □

Lemma 7. *If $B \vdash^{\wedge}_R \lambda x.M : \bigwedge_{i\in I} \rho_i$ (where, for any $i \in I$, ρ_i is not an intersection type) then for all $i \in I$ there exist σ_i and τ_i such that $\rho_i = \sigma_i \rightarrow \tau_i$ and $B, x : \sigma_i \vdash^{\wedge}_R M : \tau_i$.*

[7] See Example 5.

Proof. Any derivation of $B \vdash^{\wedge}_R \lambda x.M : \bigwedge_{i\in I} \rho_i$ can be decomposed into

(a) n derivations ($n > 0$) of $B \vdash^{\wedge}_R \lambda x.M : \tau_1, \ldots, B \vdash^{\wedge}_R \lambda x.M : \tau_n$, each of them terminating with an application of rule $(\rightarrow I)$;

(b) a derivation of $B \vdash^{\wedge}_R \lambda x.M : \bigwedge_{i\in I} \rho_i$ from $B \vdash^{\wedge}_R \lambda x.M : \tau_1, \ldots, B \vdash^{\wedge}_R \lambda x.M : \tau_n$ using only rules $(\wedge I)$ and $(\wedge E)$ in between.

Then, for all $0 < i \leqslant n$ there exist σ_i, μ_i such that $\tau_i = \sigma_i \rightarrow \mu_i$ and $B, x : \sigma_i \vdash^{\wedge}_R M : \mu_i$. By Lemma 5 and the fact that each τ_i is not an intersection, $\{\rho_i \mid i \in I\} \subseteq \{\tau_1, \ldots, \tau_n\}$, then, for all $i \in I$, $\exists \sigma_i$, μ_i such that $\rho_i = \sigma_i \rightarrow \mu_i$ and $B, x : \sigma_i \vdash^{\wedge}_R M : \mu_i$. □

Lemma 8 (Substitution Lemma). *If* $B, x : \rho \vdash^{\wedge}_R M : \sigma$ *and* $B \vdash^{\wedge}_R N : \rho$ *then* $B \vdash^{\wedge}_R M\{x \mapsto N\} : \sigma$.

Proof. By induction on the structure of M.

- If $M \equiv y \not\equiv x$, $M \equiv f \in \mathscr{F}$ or $M \equiv \lambda x.M'$ the lemma holds because $B, x : \rho \vdash^{\wedge}_R M : \sigma$, $x \notin FV(M) \Rightarrow B \vdash^{\wedge}_R M : \sigma$.
- If $M \equiv x$, then $B, x : \rho \vdash^{\wedge}_R x : \sigma$ is obtained from (copies of) $B, x : \rho \vdash^{\wedge}_R x : \rho$ by means of applications of rules $(\wedge E)$ and $(\wedge I)$. These can be applied in the same way to (copies of) $B \vdash^{\wedge}_R N : \rho$ in order to get a derivation of $B \vdash^{\wedge}_R N : \sigma$.
- If $M \equiv \lambda y.M'$ then by Lemma 1, $\sigma = \bigwedge_{i\in I} \rho_i$, and by Lemma 7, for all $i \in I$ there exist σ_i, τ_i such that $\rho_i = \sigma_i \rightarrow \tau_i$ and $B, x : \rho, y : \sigma_i \vdash^{\wedge}_R M' : \tau_i$. Then, by induction, $B, y : \sigma_i \vdash^{\wedge}_R M'\{x \mapsto N\} : \tau_i$. Using rule $(\rightarrow I)$ we obtain $B \vdash^{\wedge}_R \lambda y.M'\{x \mapsto N\} : \sigma_i \rightarrow \tau_i$ for all $i \in I$, and, using rule $(\wedge I)$, $B \vdash^{\wedge}_R \lambda y.M'\{x \mapsto N\} : \bigwedge_{i\in I} \sigma_i \rightarrow \tau_i \equiv \sigma$, where $\lambda y.M'\{x \mapsto N\} \equiv (\lambda y.M')\{x \mapsto N\}$.
- If $M \equiv M_1 M_2$ then by Lemma 1, $\sigma = \bigwedge_{i\in I} \rho_i$, and by Lemma 6, for all $i \in I$ there exist μ_i, σ_i such that $B, x : \rho \vdash^{\wedge}_R M_1 : \mu_i \rightarrow \rho_i \wedge \sigma_i$ (or simply $B, x : \rho \vdash^{\wedge}_R M_1 : \mu_i \rightarrow \rho_i$) and $B, x : \rho \vdash^{\wedge}_R M_2 : \mu_i$. By induction, $B \vdash^{\wedge}_R M_1\{x \mapsto N\} : \mu_i \rightarrow \rho_i \wedge \sigma_i$ and $B \vdash^{\wedge}_R M_2\{x \mapsto N\} : \mu_i$, for all $i \in I$. Then applying rule $(\rightarrow E)$ and by the fact that $M_1\{x \mapsto N\}M_2\{x \mapsto N\} \equiv M_1 M_2\{x \mapsto N\}$ we get $B \vdash^{\wedge}_R M\{x \mapsto N\} : \rho_i \wedge \sigma_i$ for all $i \in I$. By applying $(\wedge E)$ and $(\wedge I)$ we obtain $B \vdash^{\wedge}_R M\{x \mapsto N\} : \bigwedge_{i\in I} \rho_i \equiv \sigma$. □

Lemma 9. *Let* t *be rewritable and let* $x : \tau \in A_t$. *Then, given a derivation for* $B \vdash^{\wedge}_R t\{x \mapsto N\} : \sigma$, *all the maximal subderivations having* N *as subject have conclusion* $B \vdash^{\wedge}_R N : \tau$.

Proof. Easy, by definition of rewritable term and Lemma 4. □

Lemma 10. *Let* $r : t \rightarrow t'$ *be a* (λ-)*rewrite rule in* R:

$$B \vdash^{\wedge}_R t\{\vec{x} \mapsto \vec{N}\} : \sigma \;\Rightarrow\; B \vdash^{\wedge}_R t'\{\vec{x} \mapsto \vec{N}\} : \sigma.$$

Proof. Without loss of generality, we can assume that the variables $\vec{x}$ do not appear in B. Let $\vec{N}$ be the sequence $N_1, \ldots, N_m$ and, for $0 \leqslant i \leqslant m$, let $x_i : \tau_i \in A_t$. By

Lemma 9, all the maximal subderivations of $B \vdash^{\wedge}_R t\{\vec{x} \mapsto \vec{N}\} : \sigma$ with N_i as subject, have conclusion $B \vdash^{\wedge}_R N_i : \tau_i$. We can then replace (for any $0 \leqslant i \leqslant m$) those subderivations by $B, x_i : \tau_i \vdash^{\wedge}_R x_i : \tau_i$ and modify the derivation accordingly (adding $x_i : \tau_i$ to all the bases), getting a derivation of $B, \vec{x} : \vec{\tau} \vdash^{\wedge}_R t : \sigma$. By Lemma 4, for the unique (minimal) algebraic basis A_t we have that $A_t \vdash^{\wedge}_R t : \sigma$ and $A_t \subseteq_{\wedge} B, \vec{x} : \vec{\tau}$. By definition of rewrite rule we can infer $A_t \vdash^{\wedge}_R t' : \sigma$ and, by Lemma 2, $B, \vec{x} : \vec{\tau} \vdash^{\wedge}_R t' : \sigma$. Therefore, since we have also that $B \vdash^{\wedge}_R \vec{N} : \vec{\tau}$, by the Substitution Lemma it follows $B \vdash^{\wedge}_R t'\{\vec{x} \mapsto \vec{N}\} : \sigma$. □

Lemma 11 (Rewriting preserves types). *If $B \vdash^{\wedge}_R M : \tau$ and $M \to_R M'$ then $B \vdash^{\wedge}_R M' : \tau$.*

Proof. By definition, if $M \to_R M'$ then there exists $r : t \to t'$ in R such that $M \equiv C[t\{\vec{x} \mapsto \vec{N}\}]$ and $M' \equiv C[t'\{\vec{x} \mapsto \vec{N}\}]$. It holds then that, for a basis $B' \supseteq B$, $B' \vdash^{\wedge}_R t\{\vec{x} \mapsto \vec{N}\} : \tau'$. It is immediate to check that we can split a derivation of $B \vdash^{\wedge}_R M : \tau$ in two subderivations: one for $B' \vdash^{\wedge}_R t\{\vec{x} \mapsto \vec{N}\} : \tau'$ (for some $B' \supseteq B$ and τ') and one for $B, z : \tau' \vdash^{\wedge}_R C[z] : \tau$. By Lemma 10 we have $B' \vdash^{\wedge}_R t'\{\vec{x} \mapsto \vec{N}\} : \tau'$ and from that we can reconstruct a derivation of $B \vdash^{\wedge}_R C[t'\{\vec{x} \mapsto \vec{N}\}] : \tau$. □

Note that the assumption of ground types for the symbols in $\mathscr{F}$ (used in Lemma 4) is crucial for Lemma 11 to hold, as we said before. If type variables (and type substitution) were allowed in the signature, then rewriting would not be type preserving in general, as shown in [4].

Theorem 3 (Subject reduction). *If $B \vdash^{\wedge}_R M : \tau$ and $M \to^*_{R\beta} N$ then $B \vdash^{\wedge}_R N : \tau$.*

Proof. It suffices to prove the theorem for $\to_{R\beta}$. The thesis follows by a simple induction on the number of steps of the reduction sequence.

We proceed by structural induction on M.

- If M is variable, then $M \equiv N$.
- If $M \equiv f \in \mathscr{F}$ and $M \to_{R\beta} N$ then $M \to^r N$ for some $r \in R$, and the theorem holds because rewriting preserves types (Lemma 11).
- If $M \equiv \lambda x.M'$, the only possible reductions are in M' and then, by induction, types are preserved.
- If $M \equiv M_1 M_2$, by Lemma 1, $\tau = \bigwedge_{i \in I} \tau_i$, and by Lemma 6, for all $i \in I$ there exist μ_i, σ_i such that $B \vdash^{\wedge}_R M_1 : \mu_i \to \tau_i \wedge \sigma_i$ and $B \vdash^{\wedge}_R M_2 : \mu_i$. Let us consider all possible one-step reductions out of M:
  1. $M_1 \to_{R\beta} M_1'$ or $M_2 \to_{R\beta} M_2'$ then types are preserved by induction.
  2. $M_1 \equiv \lambda x.M_3$ and $(\lambda x.M_3)M_2 \to_{\beta} M_3\{x \mapsto M_2\}$. Then by Lemma 7, $B, x : \beta_i \vdash^{\wedge}_R M_3 : \tau_i \wedge \sigma_i$ and by the Substitution Lemma, $B \vdash^{\wedge}_R M_3\{x \mapsto M_2\} : \tau_i \wedge \sigma_i$ for all $i \in I$. Using $(\wedge E)$ and $(\wedge I)$ we get $B \vdash^{\wedge}_R M_3\{x \mapsto M_2\} : \tau$.
  3. $M \to_R M'$ then the thesis follows by Lemma 11. □.

4. Strong normalization and completeness

In this section we shall prove that all terms in $\Lambda_{\wedge R}$ are $\rightarrow_{R\beta}$-strongly normalizable when the set R of rewrite rules is safe according to Definition 12. We will also prove that $\rightarrow_{R\beta}$ is complete (strongly normalizing + confluent) for all the terms of $\Lambda_{\wedge R}$ when R is completely safe.

Dougherty [17] showed that for combinations of untyped λ-calculus and rewrite systems, strong normalization and confluence are modular properties on sets of "stable terms", and remarked that the modularity of strong normalization and confluence in combinations of typed λ-calculus with rewrite systems follows from his results. Intuitively, stability means that algebraic rewriting cannot create new β-redexes. Formally, if a set T of λ-algebraic terms is R-stable then for all $M \in T$ such that M is in β-normal form $M \rightarrow_R N$, N is in β-normal form.

Notice that our results are not consequence of [17], since $\Lambda_{\wedge R}$ is not R-stable in general. Consider the system with $\mathscr{S} = \{int\}$, $\mathscr{F} = \{F, 0\}$, $\mathscr{AX} = \{F : (int \rightarrow int) \rightarrow (int \rightarrow int), 0 : int\}$ and containing only one rule

$$FX \rightarrow X.$$

This is a terminating rule that satisfies general schema. The term $F(\lambda x.x)0 \in \Lambda_{\wedge R}$ is in β-normal form, but $F(\lambda x.x)0 \rightarrow_R (\lambda x.x)0 \rightarrow_\beta 0$.

4.1. Strong normalization of $\rightarrow_{R\beta}$

Our proof of strong normalization follows the Girard–Tait method [20], and is inspired by the proof given in [24]. It consists of two parts: in the first one we give the definition of a predicate *Red* on types and terms, usually called *computability predicate*, and prove some properties of *Red*, the most important one stating that if *Red* holds for a term then the term is strongly normalizable. In the second part *Red* is shown to hold for each term in $\Lambda_{\wedge R}$.

Before giving the definition of *Red*, we recall that, by Lemma 3, each term of $\Lambda_{\wedge R}$ is β-strongly normalizable.

Definition 15 (*Neutral terms*). A term is *neutral* if it is not of the form $\lambda x.t$ or $ft_1 \cdots t_n$ where f is a function symbol of arity m and $n < m$.

Recall that a *base type* is a type in $\mathscr{S} \cup \mathscr{V}$.

Definition 16. Let t be a term and τ a type. $Red(t, \tau)$ holds if there exists a basis B such that $B \vdash_R^{\wedge} t : \tau$ and

1. if τ is a base type then t is strongly normalizable
2. if $\tau = \tau_1 \wedge \tau_2$ then $Red(t, \tau_1)$ and $Red(t, \tau_2)$;

3. if $\tau = \sigma \to \rho$ then
   (a) if t is not neutral then for all w such that $Red(w, \sigma)$, $Red(tw, \rho)$;
   (b) if t is neutral then t is strongly normalizable and for all non-neutral t' such that $t \to_{R\beta}^{+} t'$, $Red(t', \tau)$.

We say that a term t is *computable* if there exists a type τ such that $Red(t, \tau)$.

Now, we will show that *Red* satisfies the standard properties of computability predicates.

Lemma 12. *$Red(x, \tau)$ for any variable x and type τ.*

Proof. By induction on τ using the definition of *Red* and the fact that $x : \tau \vdash_R^{\wedge} x : \tau$. □

Property 1 (C1). *Any computable term is strongly normalizable.*

Proof. By induction on τ we prove that $Red(u, \tau)$ implies that u is strongly normalizable.

1. If τ is a base type, we get immediately the thesis by definition.
2. If $\tau = \tau_1 \wedge \tau_2$ then, by definition, $Red(u, \tau)$ implies $Red(u, \tau_1)$. The strong normalizability of u now follows from the induction hypothesis.
3. If $\tau = \sigma \to \rho$
   (a) If u is not neutral then, by definition, for all w such that $Red(w, \sigma)$, $Red(uw, \rho)$ holds, in particular, by Lemma 12, we have $Red(ux, \rho)$. Then, by the induction hypothesis, ux is strongly normalizable. Hence so is u.
   (b) If u is neutral then $Red(u, \tau)$ implies, by definition, strong normalizability of u. □

Property 2 (C2). *For any two terms u, v and type τ:*

$$Red(u, \tau), u \to_{R\beta}^{*} v \Rightarrow Red(v, \tau).$$

Proof. It suffices to prove the property for a one-step derivation. By induction on τ:

1. If τ is a base type then the thesis follows easily by definition and Subject Reduction.
2. If $\tau = \tau_1 \wedge \tau_2$ then, by definition, $Red(u, \tau)$ implies $Red(u, \tau_1)$ and $Red(u, \tau_2)$, and hence, by the induction hypothesis, $Red(v, \tau_1)$ and $Red(v, \tau_2)$ hold. Therefore, by definition, $Red(v, \tau)$.
3. If $\tau = \sigma \to \rho$ then
   (a) If u is not neutral, $Red(u, \tau)$ implies (by definition) that for all w such that $Red(w, \sigma)$, $Red(uw, \rho)$ holds. By the induction we get that $Red(w, \sigma)$ implies that for all a such that $uw \to_{R\beta} a$, $Red(a, \rho)$ holds true. In particular, $uw \to_{R\beta} vw$. Then, for all w such that $Red(w, \sigma)$, we have $Red(vw, \rho)$. Hence, $Red(v, \tau)$ holds by definition.

(b) If u is neutral then $Red(u,\tau)$ implies (by definition) that u is strongly normalizable, and, for all non-neutral q, $u \to^+_{R\beta} q$ implies $Red(q,\tau)$. Now we distinguish two cases. If v is neutral then also v is strongly normalizable and for all non-neutral q, $v \to^+_{R\beta} q$ implies $Red(q,\tau)$. If v is non-neutral then the thesis follows from the definition. □

Property 3 (C3). *If u is neutral, $B \vdash^\wedge_R u : \tau$ for some B and τ, and for all v such that $u \to_{R\beta} v$, $Red(v,\tau)$ holds, then also $Red(u,\tau)$ holds.*

Proof. By induction on τ.

1. If τ is a base type then from the hypothesis and the definition of *Red* it follows that for all v such that $u \to_{R\beta} v$, v is strongly normalizable. Hence, also u is so and then we get $Red(u,\tau)$ by definition.

2. If $\tau = \sigma \to \rho$ then, since u is neutral, to prove $Red(u,\tau)$ we have to show that u is strongly normalizable and that for all non-neutral v, $u \to^+_{R\beta} v$ implies $Red(v,\tau)$. The latter requirement is trivially implied by the hypothesis and Property C2, while the former one is a consequence of Property C1.

3. If $\tau = \tau_1 \wedge \tau_2$ then, from the fact that $B \vdash^\wedge_R u : \tau$ for some B and rule $(\wedge E)$ we get $B \vdash^\wedge_R u : \tau_1$ and $B \vdash^\wedge_R u : \tau_2$. Now, since by definition of *Red* we have that for all v such that $u \to_{R\beta} v$, $Red(v,\tau_1)$ and $Red(v,\tau_2)$ hold, we can use the induction hypothesis to get $Red(u,\tau_1)$ and $Red(u,\tau_2)$ and hence $Red(u,\tau)$ by definition. □

Given a term in $\Lambda_{\wedge R}$, we can "isolate" in it its outermost first-order (quasi-)algebraic part. This notion is made precise in the following definition.

Definition 17 (*Cap and aliens*). Let $u \in \Lambda_{\wedge R}$. We define the cap of u and the multiset of its alien subterms (denoted by $cap(u)$ and $alien(u)$, respectively), as follows. An *alien subterm* of u is any maximal non-variable subterm of u which is not of the form $ft_1 \cdots t_m$ for some first-order symbol f of arity m. Let $alien(u) = \{u|_{p_1}, \ldots, u|_{p_n}\}$, then $cap(u)$ is the first-order $\wedge$-algebraic term $u[x_1]_{p_1} \cdots [x_n]_{p_n}$, where $x_1, \ldots, x_n$ are new variables such that $x_i = x_j$ if $u|_{p_i} \equiv u|_{p_j}$.

In other words, the cap of u is the quasi-$\wedge$-algebraic term obtained by replacing alien subterms by variables (the same variable for equal alien subterms). For example, the cap of $u \equiv (\lambda x.M)N$ is z and $alien(u) = \{(\lambda x.M)N\}$, while the cap of $t \equiv f(\lambda y.y)(\lambda y.y)$ is fzz if f is a binary first-order function symbol and $alien(t) = \{\lambda y.y, \lambda y.y\}$.

The following lemma, which will be needed later on in the proof of the Strong Normalization Theorem, shows that, in a sense, the cap and the alien subterms of a term do not interfere with each other.

Lemma 13 (Principal case). *If R is safe then any $\Lambda_{\wedge R}$-term of the form $ft_1 \cdots t_n$ where f is an n-ary function symbol, is $\to_{R\beta}$-strongly normalizable whenever its alien subterms are so.*

Proof. We interpret a term $t \in \Lambda_{\wedge R}$ by the pair $\langle alien(t), cap(t)\rangle$. Multisets of alien subterms are compared in the multiset extension of $\to_{R\beta} \cup \triangleright$ (recall that $\triangleleft$ is the strict subterm relation), and caps are compared in the algebraic reduction ordering $\to_{FOR}$. This is trivially a well-founded ordering if the hypotheses of the lemma are satisfied. Now, assume $t \to_{R\beta} t'$. We have two cases:

1. If $t \to_{R\beta} t'$ in a position inside an alien subterm then $alien(t)$ $(\to_{R\beta} \cup \triangleright)_{mul}$ $alien(t')$. Note that we need to consider $\triangleright$ since the result of reducing an alien subterm of t can be a term whose root belongs to Σ and so only its strict subterms can be alien subterms of t'.

2. If $t \to_{R\beta} t'$ in a cap-position then it is necessarily a first-order algebraic reduction, and hence $cap(t) \to_{FOR} cap(t')$ or $cap(t)$ reduces to a variable. In the first case, since FOR is non-duplicating, it is easy to check that $alien(t') \subseteq alien(t)$, whereas in the second case $alien(t) \rhd_{mul} alien(t')$.

In both cases the interpretation decreases (strictly), hence t is strongly normalizable by induction. □

Definition 18. A substitution $\{x_1 \mapsto u_1, \ldots, x_m \mapsto u_m\}$ is *computable in a basis* $x_1 : \sigma_1, \ldots, x_n : \sigma_n$, $n \leqslant m$, if $Red(u_1, \sigma_1), \ldots, Red(u_n, \sigma_n)$ hold.

Now, the main result of this section is

Theorem 4 (Strong normalization). *R is safe $\Rightarrow$ $\to_{R\beta}$ is strongly normalizing on $\Lambda_{\wedge R}$.*

Proof. In order to prove the theorem we shall prove a stronger property:

(∗) Let u be a term such that $FV(u) = \{x_1, \ldots, x_n\}$ and $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash^{\wedge}_{R} u : \sigma$. If $\gamma \supseteq \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$ is computable in $B = x_1 : \sigma_1, \ldots, x_n : \sigma_n$, then $Red(u\gamma, \sigma)$ holds.

The theorem follows by (C1), taking γ such that $u_i \equiv x_i$.

To prove the property we apply noetherian induction. Let $F_1, \ldots, F_n$ be the higher-order function symbols of $\mathscr{F}$. We interpret a term $u\gamma$, seen as a term-substitution pair, by the triple

$$\mathscr{I}(u, \gamma) = \langle (i, j), u, \{\gamma\}\rangle$$

where

1. i is the maximal index of the higher-order function symbols in u.
2. $j = min\{arity(F_i) - n \mid F_i t_1 \cdots t_n \triangleleft u, n \leqslant arity(F_i)\}$, and
3. $\{\gamma\}$ is the multiset $\{x\gamma \mid x$ is an occurrence of a free variable in $u\}$.

These triples are compared in the ordering

$$((>_N, >_N)_{\text{lex}}, \triangleright, (\to_{R\beta} \cup \triangleright)_{\text{mul}})_{\text{lex}}$$

denoted by $\gg$ in the following, and by $\gg_n$ when we want to indicate that the nth element of the triple has decreased but not the $(n-1)$ first ones. Here, $>_N$ denotes the

standard ordering on natural numbers and $\triangleright$ stands for the well-founded encompassment ordering, that is $u \triangleright v$ if $u \neq v$ modulo renaming of variables and $u|_p = v\delta$ for some position $p \in u$ and substitution δ. lex, mul denote, respectively, the lexicographic and multiset extension of an ordering. Since

1. $\rightarrow_{R\beta}$ is well-founded on the image of γ (by the hypothesis of computability of γ in property $(*)$, and by (C1)); and

2. the union of the relation $\triangleright$ with a terminating rewrite relation is well-founded [16]; it follows that the relation $(\rightarrow_{R\beta} \cup \triangleright)_{\text{mul}}$ is well-founded on $\{\gamma\}$. Hence, $\gg$ is a well-founded ordering on the triples.

We proceed by case analysis, depending on the form of $u\gamma$.

1. Let $u = \lambda x.v$, hence $u\gamma = \lambda x.(v\gamma)$. We know that $B \vdash_R^{\wedge} \lambda x.v : \sigma$, then by Lemmas 1 and 7, there exist τ_i, μ_i such that $B, x : \tau_i \vdash_R^{\wedge} v : \mu_i$ and $\sigma = \bigwedge_{i \in I} \tau_i \rightarrow \mu_i$.

In order to prove $Red(u\gamma, \sigma)$, by definition of *Red* we have to prove that for all $i \in I$, if $Red(t, \tau_i)$ then $Red((u\gamma)t, \mu_i)$. Since $(u\gamma)t$ is a neutral term, by (C3), it is sufficient to prove that all its immediate reducts are computable. We will do this by induction on the sum of the lengths of the derivations from $u\gamma$ and t to their normal forms. This is possible since $u\gamma$ and t are strongly normalizable. In fact, since $\mathscr{I}(u, \gamma) \gg_2 \mathscr{I}(v, \gamma)$, we get $Red(v\gamma, \mu_i)$ by induction. By (C1) it follows that $v\gamma$, and hence $u\gamma$, is strongly normalizable. By (C1) again we obtain that t is strongly normalizable, since we assumed $Red(t, \tau_i)$ to hold.

Let $(u\gamma)t \rightarrow s$. If the reduction takes place inside $u\gamma$ or inside t, then s is computable by induction. If the reduction takes place at the root position we are in the base case of the induction. Then we consider $\gamma' = \gamma \cup \{x \mapsto t\}$, hence γ' is computable in $B, x : \tau_i$. Now, $\mathscr{I}(u, \gamma) \gg_2 \mathscr{I}(v, \gamma')$, hence $Red(v\gamma', \mu_i)(s = v\gamma')$ holds by the induction hypothesis for the property $(*)$.

Therefore, by definition of *Red*, for all $i \in I$ $Red(u\gamma, \tau_i \rightarrow \mu_i)$ holds, and then $Red(u\gamma, \sigma)$ holds.

2. Let $u = ft_1 \cdots t_n$, where $n < arity(f) = m$. If at least one of the t_i's is not a variable, say t_j, then let z be a new variable and consider $\gamma' = \gamma \cup \{z \mapsto t_j\gamma\}$. The substitution γ' is computable because $\mathscr{I}(u, \gamma) \gg_2 \mathscr{I}(t_j, \gamma)$, i.e. $t_j\gamma$ is computable by induction. Again, since $\mathscr{I}(u, \gamma) \gg_2 \mathscr{I}(u[z]_j, \gamma')$, then $u[z]_j\gamma'$ is computable by induction. And $u[z]_j\gamma' = u\gamma$.

We consider now the case $u = fz_1 \cdots z_n$, where $n < arity(f) = m$, and $z_1, \ldots, z_n$ are variables. Then $B \vdash_R^{\wedge} u : \sigma = \sigma_{n+1} \rightarrow \cdots \rightarrow \sigma_m \rightarrow \beta$, and $B \vdash_R^{\wedge} z_i : \sigma_i$. Since $n < m$, by definition of *Red*, in order to prove $Red(u\gamma, \sigma)$, we have to prove that for all $t_{n+1}, \ldots, t_m$ such that $Red(t_i, \sigma_i)$ $(n + 1 \leqslant i \leqslant m)$, $Red((u\gamma)t_{n+1} \cdots t_m, \beta)$ holds.

(a) If f is a first-order symbol, then $Red((u\gamma)t_{n+1} \cdots t_m, \beta)$ follows from Lemma 13 (Principal case) and definition of *Red*, since each t_i is strongly normalizable by (C1).

(b) If f is a higher-order symbol, let $z_{n+1}, \ldots, z_m$ be new variables, and let γ' be the substitution $\gamma \cup \{z_{n+1} \mapsto t_{n+1}, \ldots, z_m \mapsto t_m\}$, which is computable by assumption. Then $\mathscr{I}(u, \gamma) \gg_1 \mathscr{I}((fz_1 \cdots z_m), \gamma')$, hence $(fz_1 \cdots z_m)\gamma'$ is computable by the induction hypothesis, and $(fz_1 \cdots z_m)\gamma' = (u\gamma)t_{n+1} \cdots t_m$.

3. Let u be a neutral term. Two cases are possible:

If u has the form $xt_1 \cdots t_n$ where $n \geqslant 0$ and x is variable, then $x\gamma$ is computable because γ is computable by assumption. If $n = 0$ we are done, since $u\gamma \equiv x\gamma$. Otherwise, since for $1 \leqslant i \leqslant n$, $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}(t_i,\gamma)$, $t_i\gamma$ is computable, we have that, by an analysis of the derivation of $B \vdash_R^{\wedge} u : \sigma$ and by definition of *Red*, $Red(u\gamma,\sigma)$ holds.

We assume in the following that u is neutral and it is not a term of the form $xt_1 \cdots t_n$ where $n \geqslant 0$ and x is a variable. So, $u\gamma$ is a neutral term too.

If $u\gamma$ is irreducible, then $Red(u\gamma,\sigma)$ holds by (C3). Otherwise, let $u\gamma \to_{R\beta} w'$ at position p. We must show either $Red(u\gamma,\sigma)$ itself or (by property (C3)) $Red(w',\sigma)$.

(a) Assume that the position p of $u\gamma$ where the reduction takes place corresponds to a subterm introduced by the substitution γ (that is, p is either a position of a variable in u or it is not a position in u).

Let $p = qp'$, $u|_q = x_i \in \mathscr{X}$ and $u' = u[z]_q$ where z is new variable. If x_i occurs only once in u then we consider $\gamma' = \gamma|_{dom(\gamma)-\{x_i\}} \cup \{z \mapsto w'|_q\}$, otherwise $\gamma' = \gamma \cup \{z \mapsto w'|_q\}$. Then $(u\gamma)|_q \to_{R\beta} w'|_q$ at position p'. Since $u\gamma|_q$ is a term in $\{\gamma\}$, $Red(u\gamma|_q,\sigma_i)$ holds by assumption, and $Red(w'|_q,\sigma_i)$ holds by property (C2), hence γ' is computable in B, $z_i : \sigma_i$. Now, $u = u'$ modulo renaming of variables if the variable $u|_q$ has exactly one occurrence in u, otherwise $u \triangleright u'$. In the first case $\mathscr{I}(u,\gamma) \gg_3 \mathscr{I}(u',\gamma')$, and $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}(u',\gamma')$ in the second case. In both cases $Red(u'\gamma',\sigma)$ holds by induction, and $u'\gamma' \equiv w'$.

(b) Assume that the subterm of $u\gamma$ where the reduction takes place is not introduced by γ, that is, p is a position in u (and it is not a variable position). There are two subcases:

(i) Assume that p is not the root position ($p \neq \varepsilon$). Let $\varepsilon < q \leqslant p$ be a position in u such that for all $\varepsilon < p' \leqslant q$, there is no abstraction at $u|_{p'}$. Then $u \triangleright u|_q$, hence $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}((u|_q),\gamma)$, and $(u|_q)\gamma = (u\gamma)|_q$. Let τ_i be the types assigned to $u|_q$ in the derivation of $B \vdash_R^{\wedge} u : \sigma$, then $Red((u\gamma)|_q,\tau_i)$ holds by induction and hence by definition of *Red*, $Red((u\gamma)|_q,\bigwedge \tau_i)$. Let $u' = u[z]_q$ for a new variable z, and $\gamma' = \gamma \cup \{z \mapsto (u\gamma)|_q\}$, then γ' is computable in B, $z : \bigwedge \tau_i$, and B, $z : \bigwedge \tau_i \vdash_R^{\wedge} u' : \sigma$. Now $u \triangleright u'$, hence $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}(u',\gamma')$, hence $Red(u'\gamma',\sigma)$. But $u\gamma$ and $u'\gamma'$ are the same term, then we get $Red(u\gamma,\sigma)$.

(ii) Now let $p = \varepsilon$. We distinguish the cases:

(A) $u = (\lambda x.v)t$ and $u\gamma = (\lambda x.v\gamma)t\gamma$. Hence $u \triangleright t$, and $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}(t,\gamma)$. By induction we get $Red(t\gamma,\tau_i)$, where τ_i are the types assigned to t in $B \vdash_R^{\wedge} u : \sigma$, and hence $Red(t\gamma,\bigwedge \tau_i)$. It follows that if $\gamma' = \gamma \cup \{x \mapsto t\gamma\}$, γ' is computable in B, $x : \bigwedge \tau_i$. Now $u \triangleright v$, hence $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}(v,\gamma')$, and since $v\gamma' = w'$, $Red(w',\sigma)$ holds.

(B) $u\gamma \to^r w'$, where $r \in R$. Again we have to consider different cases:

- Let F be the (first or higher-order) algebraic function symbol in the root of u, and let $z_1,\ldots,z_q$ be new variables. Assume that $u \triangleright Fz_1 \cdots z_q$ (the following cases deal with $u = Fz_1 \cdots z_q$ modulo renaming of variables). Let $u'_i = u|_i$ and $\gamma' = \gamma \cup \{z_i \mapsto u'_i\gamma\}$. $u \triangleright u'_i$, hence $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}(u'_i,\gamma)$. Then, if $B \vdash_R^{\wedge} u'_i : \sigma'_i$, $Red(u'_i\gamma,\sigma'_i)$ holds. Hence, γ' is computable in B, $z_1 : \sigma'_1,\ldots,z_q : \sigma'_q$. But $\mathscr{I}(u,\gamma) \gg_2 \mathscr{I}((Fz_1 \cdots z_q),\gamma')$ and B,

$z_1 : \sigma'_1, \ldots, z_q : \sigma'_q \vdash^{\wedge}_R Fz_1 \cdots z_q : \sigma$. Hence, $Red((Fz_1 \cdots z_q)\gamma', \sigma)$ holds, and since $(Fz_1 \cdots z_q)\gamma'$ and $u\gamma$ are the same term, we also get $Red(u\gamma, \sigma)$.

- $u = F_k z_1 \cdots z_q$ where $z_1, \ldots, z_q$ are variables, and F_k is a higher-order symbol of arity q. Then $(F_k z_1 \cdots z_q)\gamma$ must be an instance of a left-hand side of a rule in *HOR*:

$$u\gamma = F_k \vec{l}[\vec{M}, \vec{t}]\vec{N} \to^r v[(F_k \vec{r_1}[\vec{M}, \vec{t}]\vec{N}), \ldots, (F_k \vec{r_m}[\vec{M}, \vec{t}]\vec{N})] = w'.$$

In this case the first component of the interpretation of $u\gamma$ is $(k, 0)$.

We will prove $Red(w', \sigma)$ in 3 steps:

(I) Note that $\vec{l}[\vec{M}, \vec{t}]$ and $\vec{N}$ are all terms in $\{\gamma\}$, hence computable by hypothesis. Also the terms in $\vec{M}$ are computable because $\vec{M} \subseteq \vec{N}$ by definition of the general scheme. Then terms in $\vec{l}[\vec{M}, \vec{t}]$ are strongly normalizable by (C1). Terms in $\vec{t}$ are of base type and strongly normalizable as subterms of strongly normalizable terms, hence they are computable by definition. Let γ' be the substitution such that $\vec{X}\gamma' = \vec{M}$, $\vec{Y}\gamma' = \vec{N}$, and $\vec{x}\gamma' = \vec{t}$. Since $\vec{M}$, $\vec{N}$ and $\vec{t}$ are computable, γ' is computable. Now, $\forall j \in [1..m]$, $F_k \notin \vec{r_j}$ (by definition of *HOR*), hence $\mathscr{I}((F_k z_1 \cdots z_q), \gamma) \gg_1 \mathscr{I}(r_j, \gamma')$, hence $r_j\gamma'$ is computable.

(II) We define now the substitutions $\gamma''_j (1 \leqslant j \leqslant m)$ such that $u\gamma''_j = (F_k \vec{r_j}[\vec{X}, \vec{x}]\vec{Y})\gamma'$. These substitutions are computable because we proved in (I) the terms $r_j\gamma'$ to be computable and because $\vec{Y}\gamma'$ are computable. Since $\vec{l} \rhd_{\text{mul}} \vec{r_j}$ (by definition of the general scheme), and $\rhd$ is closed under substitution (then $\vec{l}\gamma' \rhd_{\text{mul}} \vec{r_j}\gamma'$), we obtain that $\mathscr{I}((F_k z_1 \cdots z_q), \gamma) \gg_3 \mathscr{I}((F_k z_1 \cdots z_q), \gamma''_j)$, and hence $(F_k z_1 \cdots z_q)\gamma''_j$ is computable.

(III) Let now v' be the term obtained from v (the right-hand side the higher-order rule we are applying) by replacing the "recursive calls" by new variables $z'_1, \ldots, z'_m$, and let γ''' be the substitution such that $u\gamma \to^r v'\gamma'''$ and that we have proved in (II) to be computable. Since $F_j \in v$ implies $j < k$ (by definition of the general scheme), then $\mathscr{I}(u, \gamma) \gg_1 \mathscr{I}(v', \gamma''')$. Hence w', which is the same as $v'\gamma'''$, is computable, and we at last obtain $Red(w', \sigma)$.

- $u = f z_1 \cdots z_q$, where f is a first-order symbol of arity q. Since $u\gamma$ is of base type, it is computable iff it is strongly normalizable. The latter property follows from Lemma 13 (Principal case). □

4.2. *Complete safeness implies completeness of* $\to_{R\beta}$

In this section we shall prove that, for completely safe sets of rewrite rules, $\to_{R\beta}$ is a complete reduction relation on $\Lambda_{\wedge R}$. We have already proved that, for R safe, $\to_{R\beta}$ is strongly normalizing. Now, since for strongly normalizing reduction relations, local confluence is equivalent to confluence (Newman's Lemma [29]), it will be enough to prove the local confluence property of $\to_{R\beta}$ in case R is completely safe. Let us recall the definition of local confluence.

A reduction relation $\to$ is *locally confluent* if for any t, v_1 and v_2 such that $t \to v_1$ and $t \to v_2$, there exists v_3 such that $v_1 \to^* v_3$ and $v_2 \to^* v_3$.

We begin by proving that the local confluence of $\rightarrow_{FOR}$ on first-order $\wedge$-algebraic terms transfers to $\Lambda_{\wedge R}$.

Lemma 14 (Local confluence of $\rightarrow_{FOR}$ on $\Lambda_{\wedge R}$). *If $\rightarrow_{FOR}$ is locally confluent on $\Lambda_{\wedge R}^{\mathrm{Alg}}$ then it is so also on $\Lambda_{\wedge R}$.*

Proof. Let $u \in \Lambda_{\wedge R}$. We proceed by structural induction. If $u \in \Lambda_{\wedge R}^{\mathrm{Alg}}$ then it is locally confluent by assumption, otherwise the following cases are possible:

1. If $u \equiv xt_1 \cdots t_n$ $(n \geqslant 0)$, or $u \equiv (\lambda x.t_0)t_1 \cdots t_n$ $(n \geqslant 0)$, or $u \equiv Ft_1 \cdots t_n$ where F is a higher-order function symbol, or $u \equiv ft_1 \cdots t_n$ where f is a first-order function symbol of arity $m \neq n$, then the thesis follows from the induction hypothesis since all the redexes are strictly inside these terms.

2. If $u \equiv ft_1 \cdots t_n$ and f is a first-order symbol of arity n, then the cap of u is not a variable, and $\rightarrow_{FOR}$ is locally confluent on $alien(u)$ by the induction hypothesis. Then, we only have to consider the case where $u \rightarrow_{FOR} v_1$ and $u \rightarrow_{FOR} v_2$ in (non-variable) cap positions. We distinguish three cases:

(a) If the above rewrite steps are collapsing at the root (a rewrite step is collapsing if the right-hand side of the rule applied is a variable) then $v_1 = v_2$ by local confluence of FOR.

(b) If both rewrite steps are not collapsing at the root, then $cap(u) \rightarrow_{FOR} cap(v_1)$ and $cap(u) \rightarrow_{FOR} cap(v_2)$, and since FOR is confluent on $cap(u)$ by assumption, there exists v' such that $cap(v_1) \rightarrow_{FOR}^{*} v'$ and $cap(v_2) \rightarrow_{FOR}^{*} v'$. Each variable z_i of v' appears also in $cap(u)$. Let A_i be the subterm of u replaced by z_i to obtain $cap(u)$. Then, $v_1 \rightarrow_{FOR}^{*} v'\{z_i \mapsto A_i\}$ and $v_2 \rightarrow_{FOR}^{*} v'\{z_i \mapsto A_i\}$.

(c) If one of the steps, say $u \rightarrow_{FOR} v_1$, is collapsing at the root and the other is not, then $v_2 \rightarrow_{FOR}^{*} v_1$ by local confluence of FOR and definition of cap. □

For terminating first-order rewrite systems it is well-known that the absence of critical pairs implies confluence.[8] The following lemma shows that the same property, restricted to local confluence, holds for our notion of higher-order rewriting.

Lemma 15 (Local confluence of $\rightarrow_{HOR}$ on $\Lambda_{\wedge R}$). *If HOR introduces no critical pair then $\rightarrow_{HOR}$ is locally confluent on $\Lambda_{\wedge R}$.*

Proof. To prove local confluence it is sufficient to show the commutation of $\rightarrow_{HOR}$ reductions on overlapped redexes. Let $t \in \Lambda_{\wedge R}$ such that $t \rightarrow_{HOR} v_1$ at position p and $t \rightarrow_{HOR} v_2$ at position $p \cdot q$. Since there are not critical pairs, the subterm $t|_{p \cdot q}$ of t must be covered by a variable z of the rule applied at position p. Let t' be the term obtained out of t by replacing the subterm at position $p \cdot q$ and all other occurrences of $t|_{p \cdot q}$ corresponding to z by a new variable x. Then, t' is still reducible at position $p : t' \rightarrow_{HOR} v'$. If x appears in v' at positions $m_1, \ldots, m_n$ then $t|_{p \cdot q}$ appears in v_1 at the

[8] This is not true for arbitrary notions of higher-order rewriting, as shown in [30].

same positions. Let t'' be the term obtained after reducing v_1 at positions $m_1, \cdots, m_n$. Then $v_2 \to_{HOR} t''$ at position p. Hence, HOR is locally confluent. □

Let us prove now, using an argument similar to that of the proof above, that $\to_{R\beta}$ is locally confluent when $\to_{FOR}$ is locally confluent on $\Lambda_{\wedge R}^{\mathrm{Alg}}$ and HOR does not introduce critical pairs.

Lemma 16 (Local confluence of $\to_{R\beta}$). *If FOR is locally confluent on $\Lambda_{\wedge R}^{\mathrm{Alg}}$, and HOR does not introduce critical pairs (i.e. there is no critical pair between rules of HOR, between FOR and HOR, between β and HOR), then $\to_{R\beta}$ is locally confluent on $\Lambda_{\wedge R}$.*

Proof. It suffices to show the commutation of β-, $\to_{FOR}$- and $\to_{HOR}$-reductions on overlapped redexes. But since on $\Lambda_{\wedge R}$ $\to_\beta$ is confluent, $\to_{HOR}$ is locally confluent (by Lemma 15) and $\to_{FOR}$ is locally confluent (by Lemma 14), it is sufficient to prove that for all t such that $t \to_{R\beta} v_1$ at position p using one of the reduction relations, and $t \to_{R\beta} v_2$ at position $p \cdot q$ using a different reduction relation, there exists v_3 such that $v_1 \to_{R\beta}^* v_3$ and $v_2 \to_{R\beta}^* v_3$.

Since there are no critical pairs, the subterm $t|_{p\cdot q}$ of t must be covered by a variable z of the rule applied at position p. Let t' be the term obtained out of t by replacing the subterm at position $p \cdot q$ and all other occurrences of $t|_{p\cdot q}$ corresponding to z by a new variable x. Then, t' is still reducible at position p : $t' \to_{R\beta} v'$. If x appears in v' at positions $m_1, \ldots, m_n$ then $t|_{p\cdot q}$ appears in v_1 at the same positions. Let t'' be the term obtained after reducing v_1 at positions $m_1, \ldots, m_n$. Then $v_2 \to_{R\beta} t''$ at position p. Hence, $\to_{R\beta}$ is locally confluent. □

As pointed out in [24], the class of higher-order rewriting systems defining higher-order functions by primitive recursion (structured recursion) on first-order data structures, satisfies the hypothesis of the lemma above.

Now, the main result of this subsection is

Theorem 5 (Completeness). *R is completely safe $\Rightarrow$ $\to_{R\beta}$ is complete on $\Lambda_{\wedge R}$.*

Proof. Immediate by Newman's Lemma, using Lemma 16 and Theorem 4. □

5. Conclusions

We showed that the well-known "Divide et Impera" principle can be safely used to prove termination and completeness for extensions of the intersection type assignment system with algebraic rewriting, provided that certain conditions are satisfied. More precisely, we have shown that if the first-order algebraic rewrite rules are non-duplicating and the higher-order definitions satisfy the general scheme, then termination and

completeness for the extended systems follow from the corresponding properties of their algebraic parts.

From a practical point of view the results presented here show that the extensions of intersection type assignments with algebraic rewriting can be used as a basis for the definition of algebraic functional programming languages. Of course, in that case we have to consider a *decidable* restriction of the intersection system (see e.g. [3], where the Rank 2 intersection system, which is a decidable extension of Curry's system with intersection types, is presented). The advantage of the use of intersection types is that more (still meaningful) terms are typeable: for instance, if a function f has type $int \to bool \to int$, the term fxx is typeable in an algebraic functional language based on intersection types, while it is not if Curry types are used. Such a language would be even more interesting if intersection types could also be used in the signature, but this will be the subject of further work.

A question that remains to be investigated (in our setting as well as in that of term rewriting systems alone) is whether the requirements of the general scheme for higher-order definitions could be relaxed. The general scheme guarantees that higher-order rules are terminating by construction. van de Pol [35, 36] presented a method to prove termination of higher-order rules by semantic interpretations. Unfortunately, those results cannot be directly applied to the higher-order definitions used in the present paper, since the rewrite systems he considers differ from ours in that the λ-calculus is part of the meta-language, and the rewrite relation is only defined between terms in β-normal form.

Relaxing the hypothesis of the general scheme is a difficult problem, since there are simple cases of terminating higher-order definitions, such as the one rule system

$$F(Xx)x \to F(Xx)(Xx)$$

that generate infinite derivations when combined with β-reductions:

$$F((\lambda y.\, y)x)x \to F((\lambda y.\, y)x)((\lambda y.\, y)x) \to F((\lambda y.\, y)x)x \ldots$$

Variants of the general scheme that use lexicographic orderings instead of multiset orderings, as defined in [18], could also be used here.

The use of shared rewriting for algebraic reductions as well as for β-reduction is interesting from a practical point of view: most implementations of rewriting use sharing for efficiency reasons. We expect that the same (or stronger) modularity results hold in this framework.

In the present paper the property of confluence has been addressed only as part of the completeness property, i.e. only in presence of strong normalization. The problem of the modularity of confluence without assuming to have termination is then still open and will be the subject of further research.

Acknowledgements

We wish to express our gratitude to Mariangiola Dezani and Jean-Pierre Jouannaud for their scientific support and guidance, and to two anonymous referees for many useful comments. The first author is also grateful to Tiziana Pernarella and Sandro Costantini for their steadfast encouragement.

Appendix: Combining many-sorted rewrite systems with $\vdash^\wedge$

In the paper we defined intersection type assignment systems with algebraic rewriting and showed that safe systems have interesting modularity properties. The problem of "how" to build such systems, however, has not been addressed at all. In fact, it could be the case that we wish to obtain a safe $\vdash^\wedge_R$ system by combining the intersection type system $\vdash^\wedge$ with *many-sorted term rewriting systems* (TRSs). Such a combination, even if quite natural, has to be clearly defined, because of the different perspective on types of TRSs and system $\vdash^\wedge$. Moreover, it would be useful if the safeness and complete safeness properties (defined for $\vdash^\wedge_R$ in the paper, but easily definable also for TRSs) could be preserved by combining a safe TRS with $\vdash^\wedge$.

In this appendix we address such problems. We first recall the definition of TRS (we refer to [16] for more details) and present the notion of (complete) safeness for TRSs. Then, after having shown how to embed a given TRS into system $\vdash^\wedge$, obtaining a system $\vdash^\wedge_R$, we shall prove that the safeness and complete safeness properties transfer from the TRS to $\vdash^\wedge_R$.

The problem of passing from a TRS to a system $\vdash^\wedge_R$ is mainly that of currifying terms. Such a problem has been deeply analyzed in [25] where, differently from our case, the systems considered are untyped. The results of [25] are then of no help here since, for instance, there exist strongly normalizing TRSs that turn out to be non strongly normalizing if types are not considered.

A.1. Many-sorted rewrite systems

Definition A.1. Let $\mathscr{S}$ be a set of *sorts*. The set $\mathsf{T}_\mathscr{S}$ of *types based on* $\mathscr{S}$ is inductively defined by

1. If $s \in \mathscr{S}$ then $s \in \mathsf{T}_\mathscr{S}$.
2. If $\sigma_1,\ldots,\sigma_n$, $\sigma \in \mathsf{T}_\mathscr{S}$ then $\sigma_1,\ldots,\sigma_n \to \sigma \in \mathsf{T}_\mathscr{S}$.

Definition 20. A *signature* F for a TRS is a set of *first* and *higher-order function symbols*: $\mathsf{F} = \bigcup_{\tau\in\mathsf{T}_\mathscr{S}} \mathsf{F}_\tau$, where F_τ denotes the set of function symbols of type τ ($\tau \neq \tau'$ implies $\mathsf{F}_\tau \cap \mathsf{F}_{\tau'} = \emptyset$). In case $\tau \equiv s_1 \ldots s_n \to s$ where $s_1,\ldots,s_n$, $s \in \mathscr{S}$, F_τ is a set of *first-order* function symbols, *higher-order* otherwise. We denote by Σ the set of all first-order function symbols of F.

We say that f has *arity* n, if $f \in \mathsf{F}_{\sigma_1\ldots\sigma_n\to\sigma}$.

Let $\mathsf{X} = \bigcup_{\tau \in \mathsf{T}_{\mathscr{S}}} \mathsf{X}_\tau$, where X_τ denotes a denumerable set of variables of type $\tau (\tau \neq \tau'$ implies $\mathsf{X}_\tau \cap \mathsf{X}_{\tau'} = \emptyset)$. X contains first-order variables, i.e. variables whose type is a sort, and higher-order variables. The set of first-order variables $\bigcup_{s \in \mathscr{S}} \mathsf{X}_s$ will be called $\mathsf{X}_{\mathscr{S}}$. We will use $x, y, \ldots$ to denote variables and $X, Y, \ldots$ when we want to emphasize that they are higher-order variables.

Definition 21. The set $T(\mathsf{F}, \mathsf{X})_\sigma$ of higher-order algebraic terms of type σ is inductively defined by

1. If $f \in \mathsf{F}_{\sigma_1 \ldots \sigma_n \to \sigma}$, $t_1 \in T(\mathsf{F}, \mathsf{X})_{\sigma_1}, \ldots, t_n \in T(\mathsf{F}, \mathsf{X})_{\sigma_n} (n \geqslant 0)$, then $f t_1 \ldots t_n \in T(\mathsf{F}, \mathsf{X})_\sigma$ (f is either a first-order or a higher-order function symbol).
2. If $X \in \mathsf{X}_{\sigma_1 \ldots \sigma_n \to \sigma}$, $t_1 \in T(\mathsf{F}, \mathsf{X})_{\sigma_1}, \ldots, t_n \in T(\mathsf{F}, \mathsf{X})_{\sigma_n} (n \geqslant 0)$, then $X t_1 \ldots t_n \in T(\mathsf{F}, \mathsf{X})_\sigma$. The set of higher-order algebraic terms is $T(\mathsf{F}, \mathsf{X}) = \bigcup_\tau T(\mathsf{F}, \mathsf{X})_\tau$. It includes the set $T(\Sigma, \mathsf{X}_{\mathscr{S}})$ of first-order algebraic terms: $T(\Sigma, \mathsf{X}_{\mathscr{S}}) = \bigcup_{s \in \mathscr{S}} T(\Sigma, \mathsf{X}_{\mathscr{S}})_s$.

Definition A.2. A *many-sorted rewrite rule* (denoted by $\mathsf{r} : t \to t'$) is a pair $\mathsf{r} = \langle t, t' \rangle$, such that $t, t' \in T(\mathsf{F}, \mathsf{X})_\tau$ for some $\tau \in \mathsf{T}_{\mathscr{S}}$, i.e. the rewriting rules are sound with respect to types. Besides, in $\mathsf{r} : t \to t'$, t cannot be a variable and the set of variables in t' must be a subset of the variables in t.

A rewrite rule is *first-order* if t and t' are so, *higher-order* otherwise. As in the paper, given a set R of rewrite rules we denote by FOR the subset of its first-order rules and by HOR that of higher-order rules. We will denote by $\to_{\mathsf{R}}$ ($\to_{\mathsf{FOR}}, \to_{\mathsf{HOR}}$) the reduction relation induced by R ($\mathsf{FOR}, \mathsf{HOR}$) in $T(\mathsf{F}, \mathsf{X})$.

A many-sorted higher-order algebraic rewriting system is then completely specified by a set $\mathscr{S}$ of sorts, a set X of (typed) variables, an extended signature F and a set $\mathsf{R} = \mathsf{FOR} \cup \mathsf{HOR}$ of rewriting rules.

We are going to study a particular class of TRSs which are "well behaved" when interacting with system $\vdash^\wedge$. We will call "safe" such TRSs since, when combined with system $\vdash^\wedge$, they produce safe systems (according to Definition 12).

Definition A.3. A many-sorted higher-order algebraic rewriting system $\langle \mathscr{S}, \mathsf{X}, \mathsf{F}, \mathsf{R} \rangle$ is *safe* if

1. The rules in FOR are *non-duplicating*, and *terminating* on the set $T(\Sigma, \mathsf{X})$ of first-order algebraic terms.
2. The rules in HOR satisfy the *general schema*, i.e. are of the forms

$$F\vec{l}[\vec{X}, \vec{x}]\vec{Y} \to v[(F\vec{r}_1[\vec{X}, \vec{x}]\vec{Y}), \ldots, (F\vec{r}_m[\vec{X}, \vec{x}]\vec{Y})]$$

where

(a) $\vec{X}$ is a sequence of higher-order variables such that $\vec{X} \subseteq \vec{Y}$, and $\vec{x}$ is a sequence of first-order variables such that for any $x_i \in \vec{x}$ there exists a subterm $f t_1 \ldots t_n$ of the left-hand side such that $t_j = x_i$ for some $1 \leqslant j \leqslant n$.

(b) $\vec{l}, \vec{r}_1, \ldots, \vec{r}_m$ are terms with variables in $\vec{X}, \vec{x}$, and $\forall i \in [1..m]$, $\vec{l} \rhd_{\text{mul}} \vec{r}_i$ (where $\lhd$ is the strict subterm ordering and mul denotes multiset extension),

(c) F is a higher-order function symbol not appearing in $\vec{l}, \vec{r}_1, \ldots, \vec{r}_m$; the only occurrences of F on the right-hand side v of the rule are the ones explicitly indicated.

Note the similarity between condition 2(a) in the previous definition, and the definition of rewritable $\wedge$-algebraic term. In the many-sorted framework, this restriction is needed for Subject Reduction to hold.

As said before, the non-duplicating condition for FOR is necessary to ensure that $\mathsf{FOR} \cup \mathsf{HOR}$ is terminating on algebraic terms when FOR is (i.e. modularity of termination).

The notion of *complete safeness* is immediate to give for TRSs by rephrasing that for $\vdash_R^{\wedge}$.

A.2. Embedding many-sorted rewrite systems in $\vdash^{\wedge}$

In this section we describe how to obtain a system $\vdash_R^{\wedge}$ by embedding a TRS into system $\vdash^{\wedge}$. Besides, we prove that in such an embedding the property on safeness is preserved.

We first define a currifying function $cf : \mathsf{T}_{\mathscr{S}} \to \mathscr{T}_{\mathscr{S}}$ as follows.

$$cf(s) = s \quad \text{if } s \in \mathscr{S}$$

$$cf(\sigma_1, \ldots, \sigma_n \to \sigma) = cf(\sigma_1) \to \cdots \to cf(\sigma_n) \to cf(\sigma)$$

Definition A.4. Given a TRS

$$\mathsf{R} = \langle \mathscr{S}, \mathsf{X}, \mathsf{F}, \mathsf{R} \rangle$$

we embed it into system $\vdash^{\wedge}$, by defining

$$\vdash_R^{\wedge} = \langle \mathscr{S}, \mathscr{F}, \mathscr{AX}, R \rangle,$$

where

1. $\mathscr{F}$ has exactly the function symbols of F as elements, but now considered *untyped*, i.e. we disregard the types of the elements of F.[9] We retain their arities: if $f \in \mathsf{F}_{\sigma_1,\ldots,\sigma_n \to \sigma}$ then the arity of f as element of $\mathscr{F}$ is n.

2. $\mathscr{AX}_{\mathscr{S}\mathscr{F}} = \{f_1 : cf(\sigma_1), \ldots, f_n : cf(\sigma_n)\}$ where $\mathsf{F} = \{f_1, \ldots, f_n\}$ and, for all $1 \leqslant i \leqslant n$, $f_i : cf(\sigma_i) \in \mathscr{AX}_{\mathscr{S}\mathscr{F}} \Leftrightarrow f_i \in \mathsf{F}_{\sigma_i}$.

3. R is induced by R as shown in Definition A.5 below.

To avoid ambiguities we will call "terms" the terms of $\vdash_R^{\wedge}$ just defined, algebraic terms the terms in $T(\mathsf{F}, \mathsf{X})$.

Before giving the formal definition of how a rewrite rule in a TRS induces a rewrite rule in $\vdash_R^{\wedge}$, we show that there is a correspondence between $\wedge$-algebraic terms in $\vdash_R^{\wedge}$ and algebraic terms.

[9] This means that even if $\mathscr{S} = \{int, bool\}$ and $f \in \mathscr{F}_{int \to int}$, a term such as $\lambda x.f$ True is well-formed, although it will not be typeable.

Each algebraic term can be seen as a $\wedge$-algebraic term, but, strictly speaking, it is not the same since in the definition of $\wedge$-algebraic terms the variables are considered untyped. Actually, there is a trivial mapping (type erasure) from $T(\mathsf{F},\mathsf{X})$ to the set of $\wedge$-algebraic terms (indeed to its restriction to $\Lambda^{\mathrm{Alg}}_{\wedge R}$): terms in $T(\mathsf{F},\mathsf{X})$ can be seen as $\wedge$-algebraic if types are not taken into consideration. The term in $\Lambda^{\mathrm{Alg}}_{\wedge R}$ corresponding to $t \in T(\mathsf{F},\mathsf{X})$ will be denoted by t.

Definition A.5. We define the set R of rewrite rules for $\vdash^{\wedge}_{R}$ out of R, as follows:

$$r : \mathsf{t} \to \mathsf{t}' \in R \;\Leftrightarrow\; \mathsf{r} : t \to t' \in \mathsf{R}.$$

To show that the above definition is sound we have to prove that the conditions of Definition 7 are satisfied by the elements of R.

For first-order rules it is easy to check that this is the case. However, it is not so if we consider unrestricted higher-order algebraic rewrite rules (conditions 2 and 3 of Definition 6 fail to hold.) However, if we consider only higher-order algebraic rules satisfying the general schema (condition 2 of Definition A.3), then Definition A.5 turns out to be sound also for higher-order rules.

Note that this is not a severe restriction, since what we wish to prove is that a safe TRS induce a safe $\vdash^{\wedge}_{R}$, and satisfying the general schema is one of the required conditions for safeness. It is immediate to see that if a higher-order rewrite rule $\mathsf{r} \in \mathsf{R}$ satisfies the general schema for a TRS, then induced rule $r \in R$ satisfies the general schema for $\vdash^{\wedge}_{R}$.

Since it is always clear from the context, we often use the same notation for reductions on algebraic terms and on terms of $\Lambda_{\wedge R}$.

The main result of the Appendix will be that safeness and complete safeness are preserved by the embedding of a TRS into $\vdash^{\wedge}$.

Theorem A.1 (Preservation of (complete) safeness).

(i) R *is a safe TRS* $\Rightarrow$ R *is safe for* $\vdash^{\wedge}_{R}$.

(ii) R *is a completely safe TRS* $\Rightarrow$ R *is completely safe for* $\vdash^{\wedge}_{R}$.

A.3 From $\Lambda^{\mathrm{Alg}}_{\wedge R}$ *to* $T(\mathsf{F},\mathsf{X})$

We assume to be given a TRS. Let $\vdash^{\wedge}_{R}$ be the assignment system obtained by embedding the given TRS into system $\vdash^{\wedge}$. We have seen above that, by means of type-erasing, any element in $T(\mathsf{F},\mathsf{X})$ has an immediate counterpart in $\Lambda^{\mathrm{Alg}}_{\wedge R}$. Because of the presence of intersection types, there exist, instead, elements of $\Lambda^{\mathrm{Alg}}_{\wedge R}$, like fxx (with $f \in \mathsf{F}_{int,bool\to int}$ and $x : int \wedge bool$ in the basis), that have no immediate counterpart in $T(\mathsf{F},\mathsf{X})$.

In this subsection we shall define a natural mapping from $\Lambda^{\mathrm{Alg}}_{\wedge R}$ to $T(\mathsf{F},\mathsf{X})$. It will be needed later on in order to prove that (complete) safeness transfers from TRS to $\vdash^{\wedge}_{R}$. The mapping we are to define, if applied to the term fxx above, should return the algebraic term $f^{int,bool\to int}x_1^{int}x_2^{bool}$. In fact, while in $\vdash^{\wedge}_{R}$ x can have type $int\wedge bool$ in a basis,

we cannot have intersection types in algebraic terms. We have then to rename variable occurrences. It is easy to see, however, that in order to formally define this mapping one needs to consider more information than the mere $\wedge$-algebraic terms: consider, for instance, the $\wedge$-algebraic term $y(fxx)zz$. It could be mapped to $y^{i,i,i\to i}(f^{i,b\to i}x_1^i x_2^b)z_1^i z_1^i$ or $y^{i,i,b\to i}(f^{i,b\to i}x_1^i x_2^b)z_1^i z_2^b$. The extra information needed to make the definition of the mapping uniquely determined is contained in the derivations in $\vdash_R^{\wedge}$ for our $\wedge$-algebraic term. We will then define a mapping not from $\wedge$-algebraic terms, but from derivations for them, to algebraic terms. The definition, even if quite natural, will require some cumbersome technicalities in order to be formally stated.

Given a derivation $\mathscr{D}$ for a term M, we denote by $\mathscr{D}|_p$ its subderivation for the subterm $M|_p$.

Definition A.6. We define a derivation $\mathscr{D}$ in $\wedge R$ to be:
- $\wedge$-var if all rules $(\wedge I)$ and $(\wedge E)$ contained in it have only variable statements as premises.
- $\wedge$-free if no symbol $\wedge$ appears in it.

It is easy to check that there is a one–one correspondence between $\wedge$-free derivations for $\wedge$-algebraic terms and terms in $T(\mathsf{F},\mathsf{X})$.

Lemma A.1. *Any typeable $\wedge$-algebraic term has a $\wedge$-var derivation for it.*

Proof. By induction on the structure of the term. We need, however, to strengthen our thesis and prove that any $\wedge$-albebraic term M typeable with a type, say α, is typeable with α, but from a possibly different basis, also by means of a $\wedge$-var derivation.

- $M \equiv xM_1 \ldots M_n$, $(n \geqslant 0)$. By hypothesis, $M_1,\ldots,M_n$ are typeable, so, let $\beta_1,\ldots,\beta_n$ be their types. By induction $M_1,\ldots,M_n$ are typeable with $\beta_1,\ldots,\beta_n$ by means of $\wedge$-var derivations. If $\gamma_1,\ldots,\gamma_m$ are the types for x used in such derivations, then, from a basis where x has type $\gamma_1 \wedge \cdots \wedge \gamma_m \wedge (\beta_1 \to \cdots \to \beta_n \to \beta)$, M is typeable with β by means of a $\wedge$-var derivation.
- $M \equiv fM_1 \ldots M_n, (n \geqslant 0)$. Easy by the induction hypothesis and the fact that in any derivation for M the types for $M_1,\ldots,M_n$ depend necessarily on the (fixed) type of f. □

Because of the lemma above we can always assume a derivation for a $\wedge$-algebraic term to be $\wedge$-var. We will implicitly make such an assumption in the rest of the present section.

Definition A.7. The function $\kappa : \mathscr{T}_{\wedge S} \to \mathscr{T}_S$ is inductively defined as follows: For $s \in \mathscr{S}$, $\kappa(s) = s$. For any type variable $\varphi, \kappa(\varphi) = s_0$ for a certain fixed $s_0 \in \mathscr{S}$.
$\kappa(\sigma \wedge T) = \kappa(\sigma)$ if σ precedes τ in some chosen lexicographical order between types.[10]
$\kappa(\sigma \to \tau) = \kappa(\sigma) \to \kappa(\tau)$.

[10] We need such an order since types are considered modulo commutativity.

We say that an occurrence of a statement in a derivation for a $\wedge$-algebraic term t is $\wedge$-last if in the path from this statement to the conclusion there is no application of a $(\wedge I)$ or $(\wedge E)$ rule. It is worth noticing the following fact, to be used below: there is a one-to-one correspondence between one occurrence of variable in a typeable $\wedge$-algebraic term M (i.e. a variable position p of M) and a $\wedge$-last occurrence of a variable statement in a ($\wedge$-var) derivation for M. A derivation for a $\wedge$-algebraic term is called $\wedge$-*linear* if there are no two $\wedge$-last occurrences of a variable statement for the same variable and with types, say σ and τ, such that $\kappa(\sigma) \not\equiv \kappa(\tau)$. A *renaming* ρ for a $\wedge$-algebraic term M is any function from occurrences of variables in M to variables. A $\wedge$-*renaming* for a derivation $\mathscr{D}$ for a $\wedge$-algebraic term M is any renaming ρ for M such that, given two variable occurrences p_1 and p_2:

$$\rho(p_1) \equiv \rho(p_2) \text{ iff } \wedge\text{-}l_{\mathscr{D}}(p_1) \equiv \wedge\text{-}l_{\mathscr{D}}(p_2),$$

where $\wedge$-$l_{\mathscr{D}}(p)$ is defined as the $\wedge$-last statement in $\mathscr{D}$ corresponding to the variable occurrence p. It is easy to check that, since M is $\wedge$-algebraic, the above definition is sound. Given a $\wedge$-algebraic term M and a renaming ρ for a derivation of it, we denote by M_ρ the application of the renaming to M.

Given a ($\wedge$-var) derivation $\mathscr{D}$ for a $\wedge$-algebraic term M and a $\wedge$-renaming for it, it is possible to get a $\wedge$-linear derivation (still $\wedge$-var). We can proceed as follows: replace each axiom occurrence $B, x : \tau \vdash x : \tau$ in $\mathscr{D}$ by $B, \rho(p) : \tau \vdash \rho(p) : \tau$, where p is the variable occurrence in M whose corresponding $\wedge$-last variable statement in $\mathscr{D}$ has the axiom occurrence considered as premise, and modify the rest of the deduction accordingly. We call $\wedge$-*linearization* such a procedure and denote with $\mathscr{D}^l_\rho$ the $\wedge$-linearization of a derivation $\mathscr{D}$ for a $\wedge$-algebraic term, given a $\wedge$-renaming ρ for it.

It is possible, out of a $\wedge$-linear derivation, to get a $\wedge$-free derivation, and hence an algebraic term. The $\wedge$-*flattening* of a linear derivation for a $\wedge$-algebraic term is obtained by modifying the derivation as follows: for any path from an axiom to a $\wedge$-last variable statement, say $B \vdash x : \tau$, replace all the path by the axiom $B', x : \kappa(\tau) \vdash x : \kappa(\tau)$, where B' is such that $y : \delta \in B'$ if and only if $\delta \equiv \kappa(\sigma)$ and $y : \sigma$ is in a $\wedge$-last variable statement.

We denote by $\mathscr{D}^f$ the $\wedge$-flattening of $\mathscr{D}$. By induction on the derivation it is easy to check that $\mathscr{D}^f$ is a correct derivation for a term M if $\mathscr{D}$ is so. We can now define our mapping from (derivations of) $\wedge$-algebraic terms and renamings for them to algebraic terms.

Definition A.8 (*The mapping* ζ). Let $\mathscr{D}$ be a derivation for a $\wedge$-algebraic term M and ρ a $\wedge$-renaming for it. We define

$$\zeta(\mathscr{D}, \rho) =_{\text{Def}} at((\mathscr{D}^l_\rho)^f),$$

where at($\mathscr{D}$) for $\mathscr{D}$ $\wedge$-free is the algebraic term corresponding to it.

$\zeta(\mathscr{D},\rho)$ will be also denoted, in the following, by $\zeta^{\rho}(\mathscr{D})$. If ρ is a $\wedge$-renaming for a derivation $\mathscr{D}$, we shall denote by $\rho|_p$ its restriction to $\mathscr{D}|_p$.

A.4. Preservation of safeness and complete safeness

In this subsection we show that safeness and complete safeness naturally transfer from TRS to $\vdash_R^{\wedge}$. This fact will allow us to apply the modularity results of the previous sections to embeddings of TRSs into the intersection system for λ-calculus. For safeness, since the embedding preserves the fact that higher-order rules satisfy the general schema, what we need to prove is that, if $\rightarrow_{\mathsf{FOR}}$ is strongly normalizing in $T(\mathsf{F},\mathsf{X})$, so is $\rightarrow_{FOR}$ for terms in $\Lambda_{\wedge R}^{\mathrm{Alg}}$ of $\vdash_R^{\wedge}$. Preservation of complete safeness needs also the local confluence of $\rightarrow_{FOR}$ on $\Lambda_{\wedge R}^{\mathrm{Alg}}$ if $\rightarrow_{\mathsf{FOR}}$ is locally confluent in $T(\mathsf{F},\mathsf{X})$.

We first note that, for safe rewrite rules, it is immediate to extend to derivations the notion of rewriting. If $\mathscr{D}$ is a derivation for the term $M \in \Lambda_{\wedge R}^{\mathrm{Alg}}$ which has a r-redex, for $r \in R$, at position p, we denote by $r_p(\mathscr{D})$ the derivation obtained out of $\mathscr{D}$ by means of that reduction. It is easy to check that if $M \rightarrow^r N$ and $\mathscr{D}$ is a derivation for M, then $r_p(\mathscr{D})$ is a correct derivation for N (moreover, it remains $\wedge$-var).

A reduction on $\wedge$-algebraic terms can be seen also as a map from $\wedge$-renamings to $\wedge$-renamings. Let ρ be a $\wedge$-renaming for the derivation $\mathscr{D}$ for the $\wedge$-algebraic term M and let N be the result of applying the rule r at position p in M, which we denote by $M \xrightarrow{p,r} N$. We define the $\wedge$-renaming ρ^r_p for $r_p(\mathscr{D})$ as follows: given a variable occurrence q in N, $\rho^r_p(q) =_{Def} \rho(q'_p)$, where q'_p is the variable occurrence in M whose residual by $\xrightarrow{p,r}$ is q. It is easy to check that ρ^r_p is really a $\wedge$-renaming for $r_p(\mathscr{D})$.

There is one more point to be discussed before going into the heart of the proof of preservation of (complete) safeness. A key point in proof will be the fact that the mapping $\zeta(-)$ preserves reductions, as shown in the lemma below. This, however, is not true if we consider arbitrary $\wedge$-var derivations for $\wedge$-algebraic terms, as the following example shows. Let us consider the term $g(yz)(yz)$ (where $g \in \mathscr{F}_{int,int\rightarrow bool}$ and assume to have the reduction rule $gx^{int}x^{int} \rightarrow^r a$. Now, it is easy to check that $g(yz)(yz) \rightarrow^r a$. On the contrary, an algebraic term obtained from $g(yz)(yz)$ using $\zeta(-)$ reduces to a only if the $\wedge$-var derivation considered for $g(yz)(yz)$ is such that $\wedge\text{-}l_{\mathscr{D}}(11) \equiv \wedge\text{-}l_{\mathscr{D}}(21)$ and $\wedge\text{-}l_{\mathscr{D}}(12) \equiv \wedge\text{-}l_{\mathscr{D}}(22)$, (otherwise, $g(yz)(yz)$ could be mapped, for instance, into $g(uv)(xv)$, which is not reducible by r). However, because of the "safeness" of the reduction rules, it is easy to check that it is always possible to transform a derivation for a $\wedge$-algebraic term (maintaining the property of being $\wedge$-var) in such a way that two sub-derivations for the same subterm are equal in case the subterm occurs in variable positions of a pattern for a reduction rule. Then, in the rest of this section we will consider only derivations with such a property. Besides, notice that reductions on derivations preserve this property.

There is a correspondence between reductions on terms in $\Lambda_{\wedge R}^{\mathrm{Alg}}$, and on terms in $T(\mathsf{F},\mathsf{X})$.

Property A.1. *Let M and N be $\wedge$-algebraic terms, $\mathscr{D}$ a derivation for $M, r \in R, \rho$ a $\wedge$-renaming for $\mathscr{D}$, and p a position in M :*

$$M \xrightarrow{p,r} N \Rightarrow \zeta^{\rho}(\mathscr{D}) \xrightarrow{p,r} \zeta^{\rho^r_p}(r_p(\mathscr{D})).$$

Proof. Let $r : t \to t'$. Hence, $M|_p \equiv \mathrm{t}\{x_2 \mapsto N_1, \ldots, x_n \mapsto N_n\} \to^r \mathrm{t}'\{x_1 \mapsto N_1, \ldots, x_n \mapsto N_n\} \equiv N|_p$.

It is not difficult to check that, by definition of ρ^r_p, we get $\zeta^{\rho|_p}(\mathscr{D}|_p) \to^r \zeta^{\rho^r_p|_p}(r_\varepsilon(\mathscr{D}|_p))$. Hence $\zeta^{\rho}(\mathscr{D}) \xrightarrow{p,r} \zeta^{\rho^r_p}(r_p(\mathscr{D}))$. □

Property A.2. *Let M be a $\wedge$-algebraic term, $\mathscr{D}$ a derivation for it, N an algebraic term, $r \in R$, and p a position in M. If $\zeta^{\rho}(\mathscr{D}) \xrightarrow{p,r} N$ then $M \xrightarrow{p,r} \mathrm{N}_{(\rho^r_p)^{-1}}$ and $\zeta^{\rho^r_p}(r_p(\mathscr{D})) \equiv N$.*

Proof. Let $r : t \to t'$, then $\zeta^{\rho}(\mathscr{D})|_p \equiv t\phi$ for some ϕ and $N \equiv \zeta^{\rho}(\mathscr{D})[t'\phi]_p$. By definition of the reduction relation and of ρ^r_p, we get $M \to^r \mathrm{N}_{(\rho^r_p)^{-1}}$, where $(\rho^r_p)^{-1}$ is the obvious inverse of the renaming ρ^r_p. Moreover, it is easy to check that $r_p(\mathscr{D})$ is a correct derivation for $\mathrm{N}_{(\rho^r_p)^{-1}}$ and that $\zeta^{\rho^r_p}(r_p(\mathscr{D})) \equiv N$. □

We can now prove the Preservation of (complete) safeness Theorem.

Proof of Theorem A.1. Let M be a $\wedge$-algebraic term, $\mathscr{D}$ a derivation for it and ρ a renaming.

(i) It is enough to prove, using the hypothesis, that $\to_{FOR}$ is strongly normalizing on typeable $\wedge$-algebraic terms.

By contradiction: given an infinite FOR-reduction sequence out of M, using Property A.1 we obtain an infinite FOR-reduction sequence out of $\zeta^{\rho}(\mathscr{D})$.

(ii) It is enough to prove, using the hypothesis, that $\to_{FOR}$ is locally confluent on typeable $\wedge$-algebraic terms.

Let $N_1 \ {}_{r_1}\!\leftarrow M \to_{r_2} N_2$. By Property A.1, we get $\zeta^{\rho_1}(\mathscr{D}_{N_1})_{r_1} \leftarrow \zeta^{\rho}(\mathscr{D}) \to_{r_2} \zeta^{\rho_2}(\mathscr{D}_{N_2})$, where $\rho_1 \equiv \rho^{r_1}_{p_1}$ and $\rho_2 \equiv \rho^{r_2}_{p_2}$. By the local confluence property of $\to_{\mathsf{FOR}}$ on algebraic terms, there exists a term t such that $\zeta^{\rho_1}(\mathscr{D}_{N_1}) \to^*_{\mathsf{FOR}} t \ {}^*_{\mathsf{FOR}}\!\leftarrow \zeta^{\rho_2}(\mathscr{D}_{N_2})$. Let $\xrightarrow{p'_1} r'_1 \ldots \xrightarrow{p'_{k'}} r'_{k'}$ and $\xrightarrow{p''_1} r''_1 \ldots \xrightarrow{p''_{k''}} r''_{k''}$ be the reduction sequences from $\zeta^{\rho_1}(\mathscr{D}_{N_1})$ to t and from $\zeta^{\rho_2}(\mathscr{D}_{N_2})$ to t, respectively. By several applications of Property A.2 we get that, by defining $\rho'_1 = (\ldots(((\rho_1)^{r'_1}_{p'_1})^{-1} \ldots)^{r'_{k'}}_{p'_{k'}})^{-1}$ and ρ'_2 similarly, $N_1 \to^*_{FOR} \mathrm{t}_{\rho'_1}$ and $\mathrm{t}_{\rho'_2} \ {}^*_{FOR}\!\leftarrow N_2$. By definition of ρ^r_p, it is easy to check that $\rho'_1 = \rho'_2$. Therefore local confluence holds for $\wedge$-algebraic terms. □

As a consequence of Theorems A.1, 4 and 5, we obtain:

Theorem A.2. (i) R *is a safe TRS* $\Rightarrow \vdash^{\wedge}_{R}$ *is strongly normalizable.*

(ii) R *is a completely safe TRS* $\Rightarrow \vdash^{\wedge}_{R}$ *is complete.*

References

[1] S. van Bakel, Intersection type disciplines in lambda calculus and applicative term rewrting systems, Ph.D. Thesis, University of Nijmegen, 1993.

[2] S. van Bakel, The heart of intersection type assignment; normalization proofs revisited. Tech. Report, Dipartimento di Informatica, Universita' di Torino, 1995.

[3] S. van Bakel, Partial intersection type assignment of rank 2 in applicative term rewriting systems, Fundam. Inform., to appear.

[4] S. van Bakel, S. Smetsers and S. Brock, Type assignment in left linear applicative term rewriting systems, in: *Proc. Colloq. on Trees in Algebra and Programming*, Rennes, Lecture Notes in Computer Science, Vol. 581 (Springer, Berlin, 1992).

[5] F. Barbanera, Combining term rewriting and type assignment systems, *Internat. J. Foundations Comput. Sci.* **1** (1990) 165–184.

[6] F. Barbanera and M. Fernández, Combining first and higher-order rewrite systems with type assignment systems, in: *Proc. 1st Internat. Conf. on Typed Lambda Calculus and Applications*, Utrecht, Holland, Lecture Notes in Computer Science, Vol. 664 (Springer, Berlin, 1993).

[7] F. Barbanera, M. Fernández and H. Geuvers, Modularity of strong normalization and confluence in the λ-algebraic-cube, in: *Proc. 4th Ann. IEEE Symp. on Logic in Computer Science*, Paris, 1994.

[8] H.P. Barendregt, *The Lambda Calculus, its Syntax and Semantics* (North-Holland, Amsterdam, 2nd ed., 1984).

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

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

[11] V. Breazu-Tannen and J. Gallier, Polymorphic rewriting conserves algebraic strong normalization, *Theoret. Comput. Sci.* (1990).

[12] F. Cardone and M. Coppo, Two extensions of Curry's type inference system, in: P. Odifreddi, ed., *Logic and Computer Science* (Academic Press, New York, 1990).

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

[14] M. Coppo, M. Dezani, F. Honsell and G. Longo, Extended type structures and filter lambda models, in: *Logic Colloq. 82* (North-Holland, Amsterdam, 1984).

[15] M. Coppo, M. Dezani-Ciancaglini and B. Venneri, Principal type schemes and λ calculus semantics, in: J.P. Seldin and J.R. Hindley, eds., *To H.B. Curry: Essays on Combinatory logic, Lambda Calculus and Formalism* (Academic Press, New York, 1980).

[16] N. Dershowitz and J.-P. Jouannaud, Rewrite systems, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. B (North-Holland, Amsterdam, 1990) 242–309.

[17] D.J. Dougherty, Adding algebraic rewriting to the untyped lambda calculus, in: *Proc. 4th Rewriting Techniques and Applications*, Como, Lecture Notes in Computer Science, Vol 488 (Springer, Berlin, 1991).

[18] M. Fernández and J.-P. Jouannaud, Modular termination of term rewriting systems revisited, in: *Recent Trends in Data Type Specification; Proc. 10th Workshop of Specification of Abstract Data Types (ADT'94)*, Santa Margherita, Italy, Lecture Notes in Computer Science, Vol. 906 (Springer, Berlin, 1995).

[19] K. Futatsugi, J. Goguen, J.P. Jouannaud and J. Meseguer, Principles of OBJ2, in: *Proc. 12th ACM Symp. on Principles of Programming Languages* (1985) 52–66.

[20] J.-Y. Girard, Y. Lafont and P. Taylor, *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science (Cambridge Univ. Press, Cambridge, 1989).

[21] M. Gordon, R. Milner and C. Wadsworth, *Edinburgh LCF*, Lecture Notes in Computer Science, Vol. 78 (Springer, Berlin, 1979).

[22] R. Hindley, Types with intersection, an introduction, *Formal Aspects Comput.* (1990).

[23] R. Hindley and J. Seldin, *Introduction to Combinators and λ-calculus* (Cambridge Univ. Press, Cambridge, 1986).

[24] J.-P. Jouannaud and M. Okada, Executable higher-order algebraic specification languages, in: *Proc. 6th IEEE Symp. Logic in Computer Science*, Amsterdam (1991) 350–361.

[25] R. Kennaway, J.W. Klop, R. Sleep and F.J. de Vries, Comparing curried and uncurried rewriting, *J. Symbolic Comput.* **11** (1995).
[26] J.W. Klop, Term rewriting systems: a tutorial, *EATCS Bull.* **32** (1987) 143–182.
[27] J.W. Klop, V. van Oostrom and F. van Raamsdonk, Combinatory reduction systems, introduction and survey, *Theoret. Comput. Sci.* **121** (1993).
[28] D. Leivant, Typing and computational properties of lambda expressions, *Theoret. Comput. Sci.* **44** (1986) 51–68.
[29] M.H.A. Newman, On theories with a combinatorial definition of 'equivalence', *Ann. Math.* **43** (1942) 223–243.
[30] T. Nipkow, Higher order critical pairs, in: *Proc. IEEE Symp. on Logic in Comp. Science*, Amsterdam, 1991.
[31] M. Okada, Strong normalizability for the combined system of the types lambda calculus and an arbitrary convergent term rewrite system, in: *Proc. 20th Internat. Symp. on Symbolic and Algebraic Computation*, Portland, Oregon, 1989.
[32] V. van Oostrom and F. van Raamsdonk, Comparing combinatory reduction systems and higher-order rewrite systems, in: *Proc. 1st Internat. Workshop on Higher-Order Algebra, Logic and Term Rewriting*, Amsterdam, Lecture Notes in Computer Science, Vol. 816 (Springer, Berlin, 1994).
[33] V. van Oostrom and F. van Raamsdonk, Weak orthogonality implies confluence: the higher order case, in: *Proc. 3rd Internat. Symp. on Logical Foundation of Computer Science*, St. Petersburg, Lecture Notes in Computer Science, Vol. 813 (Springer, Berlin, 1994).
[34] D. Plump, Implementing term rewriting by graph reduction: termination of combined systems, in: *Proc. Conditional and Typed Rewriting Systems*, Lecture Notes in Computer Science, Vol 516 (Springer, Berlin, 1991).
[35] J. van de Pol, Termination proofs for higher-order rewrite systems, in: *Proc. 1st Internat. Workshop on Higher-Order Algebra, Logic and Term Rewriting*, Amsterdam, Lecture Notes in Computer Science, Vol. 816 (Springer, Berlin, 1994).
[36] J. van de Pol and Schwichtenberg, Strinct functionals for termination proofs, in: *Proc. 2nd Internat. Conf. on Typed Lambda Calculi and Applications*, Edinburgh, Lecture Notes in Computer Science, Vol. 902 (Springer, Berlin, 1995).
[37] G. pottinger, A type assignment for the strongly normalizable λ-terms, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism* (Academic Press, New York, 1980) 561–578.
[38] M. Rusinowitch, On termination of the direct sum of term rewriting systems, *Inform. Processing Lett.* **26** (1987) 65–70.
[39] Y. Toyama, Counterexamples to termination for the direct sum of term rewriting systems, *Inform. Processing Lett.* **25** (1987) 141–143.
[40] D.A. Turner, Miranda: a non-strict functional language with polymorphic types, in: *Functional Programming Languages and Computer Architecture*, Nancy, Lecture Notes in Computer Science, Vol. 201 (Springer, Berlin, 1985).