Normalization Results for Typeable Rewrite Systems

(Information and Computation, 133(2):73-116, 1997)

Steffen van Bakel[1], Maribel Fernández[2]

[1]) Department of Computing, Imperial College of Science, Technology and Medicine,
180 Queen's Gate, London SW7 2BZ, U.K.,
[2]) DMI - LIENS (CNRS URA 1327), École Normale Supérieure,
45 rue d'Ulm, 75005 Paris, France
svb@doc.ic.ac.uk,maribel@ens.fr

Abstract

In this paper we introduce Curryfied Term Rewriting Systems, and a notion of partial type assignment on terms and rewrite rules that uses intersection types with sorts and ω. Three operations on types – substitution, expansion, and lifting – are used to define type assignment, and are proved to be sound. With this result the system is proved closed for reduction.
Using a more liberal approach to recursion, we define a general scheme for recursive definitions and prove that, for all systems that satisfy this scheme, every term typeable without using the type-constant ω is strongly normalizable. We also show that, under certain restrictions, all typeable terms have a (weak) head-normal form, and that terms whose type does not contain ω are normalizable.

1 Introduction

There are essentially three paradigms in common use for the design of functional programming languages: the λ-calculus (LC for short), Term Rewriting Systems (TRS), and Term Graph Rewriting Systems (TGRS). The LC, or rather combinator systems, forms the underlying model for the functional programming language Miranda[1] [36], TRS are used in the language OBJ [24], and TGRS form the base model for the language Clean [17].

For LC, there exists a well understood notion of type assignment known as the *Curry type assignment system* [20], that expresses abstraction and application. The *intersection type discipline* [18, 14] is an extension of Curry's system that consists of allowing more than one type for term-variables and terms, adding a type constant 'ω', and considering the type constructor '$\cap$' in addition to the type constructor '$\rightarrow$'. One of the most appealing features of intersection type assignment in LC is the fact that normalization of terms can be studied through assignable types (see e.g. [14] and [3]):

- M has a head-normal form iff $B \vdash M{:}\sigma$ and $\sigma \neq \omega$.
- M has a normal form iff $B \vdash M{:}\sigma$ and ω does not occur in B and σ.
- M is strongly normalizable iff $B \vdash M{:}\sigma$ and ω is not used at all.

The *essential intersection system for LC* defined by Bakel-TCS'95 is a restriction of the intersection type discipline that satisfies all the properties above. Its main advantage is that the set of types assignable to a term is significantly smaller than in the full intersection system.

Though many functional programming languages allow programmers to specify an algorithm (function) as a set of rewrite rules, type assignment for TRS has not attracted much

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

attention until now. This is remarkable, since TRS and LC are essentially different: although both formalisms are Turing-complete, there exists no direct translation of TRS to LC. For example, adding the definition of surjective pairing,

$$\begin{array}{ll} \mathsf{Fst}\,(\mathsf{Pair}\,(\mathsf{x},\mathsf{y})) & \to \mathsf{x} \\ \mathsf{Snd}\,(\mathsf{Pair}\,(\mathsf{x},\mathsf{y})) & \to \mathsf{y} \\ \mathsf{Pair}\,(\mathsf{Fst}\,(\mathsf{x}),\mathsf{Snd}\,(\mathsf{x})) & \to \mathsf{x} \end{array}$$

to LC gives a system in which the Church-Rosser property no longer holds [30]; this implies that the above TRS cannot be expressed in LC.

Although it seems straightforward to extend type assignment systems for LC to TRS, it is not evident that those borrowed systems will still have, for general TRS, all the properties they possessed in the setting of LC. For example, some restrictions have to be imposed in the assignment of types to rewrite rules in order to ensure the subject reduction property (i.e. preservation of types under rewriting), as illustrated in [10].

The aim of this paper is to define a notion of (essential) intersection type assignment *directly for TRS* and to study normalization properties in that setting. We use intersection types because more meaningful terms can be typed in this way. Also, the notion of type assignment presented in this paper applies to TGRS and in that framework intersection types are the natural tool to type nodes that are shared (another notion of type assignment on TGRS was defined by Barendsen-Smetsers-FSTTCS'93, to study safeness of destructive updates). Intersection types are also promising for use in functional languages, since they provide a good formalism to express overloading, see [34].

We consider *Curryfied* TRS ($\mathcal{A}$TRS), a slight extension of the TRS defined by Klop'92, and Dershowitz-Jouannaud'90. $\mathcal{A}$TRS contain a special binary operator Ap, that models application and allows for partial application of function symbols (Curryfication). $\mathcal{A}$TRS are also extensions of the constructor systems used in most functional programming languages in that they do not discriminate against the varieties of function symbols that can be used in patterns. However, we will in some cases make this distinction when we will study normalization properties of $\mathcal{A}$TRS.

Recently, some results have been obtained in the field of *typed TRS* [21] and the combination of those with intersection type assignment systems for LC [11]. The idea behind those systems is that rewrite rules aim to describe manipulations of objects of an algebraic data-type and, therefore, concepts like polymorphism are not introduced within TRS. In contrast, in this paper we present a *type assignment system for* $\mathcal{A}$TRS that is closer to the approach of intersection type assignment in LC; in particular, rewrite rules can be polymorphic.

The type assignment system on $\mathcal{A}$TRS that we define is based on a combination of the essential intersection system for LC and the type assignment system of ML [32], both extensions of Curry's type assignment system. Type assignment will be defined through a natural deduction system, assuming that every function symbol has a predefined type, given in an *environment*. This approach is similar to the one taken by Hindley'69 to define the principal Curry type of an object in Combinatory Logic.

The polymorphic aspect of our type assignment system becomes apparent in the derivation rule that deals with the assignment of a type to a term like $F(t_1,\ldots,t_n)$. There the type predefined for F in the environment can be 'instantiated' by applying operations of substitution, expansion, and lifting (see [5]). The operation of substitution deals with the replacement of type-variables by types, the operation of expansion replaces types by the intersection of a number of copies of that type and coincides with the one given by Coppo-Dezani-Venneri'80,

and the operation of lifting deals with both taking more specific types in bases and assigning a more general type to terms. We use these three operations, instead of just substitution, not only because more terms are typeable in this way, but also to obtain a natural embedding of LC in TRS that preserves assignable types (with just substitution, this would not be possible).

The type assignment system presented in this paper can be seen as a generalization of the systems of Bakel-Smetsers-Brock-CAAP'92 and Bakel-FI; the main difference is the set of types used: Curry types in [10], intersection types of Rank 2 in [7], and strict intersection types in this paper. Type assignment in those systems is decidable, whereas in the one presented here it is not. However, the normalization results we will prove hold also for free in these decidable restrictions of the system.

In contrast with LC, typeable terms in $\mathcal{G}$TRS need not even be head-normalizable; for example, consider a typeable term t and a rule $t \to t$. That is why we need to control the use of recursion by imposing some syntactical conditions on the rewrite rules (a generalization of primitive recursion). We will define a recursive scheme for rewrite rules that is inspired by the *general scheme* of Jouannaud-Okada'91. The general scheme was devised for the incremental definition of higher order functionals based on first order definitions, such that their combination with polymorphic LC is terminating. It was also used for defining higher order functions compatible with other lambda calculi by Barbanera-Fernandez-TLCA'93 and Barbanera-Fernandez-Geuvers-LICS'94.

It is worthwhile to notice that, even with the severe restrictions imposed on rewrite rules by the general scheme, the class of $\mathcal{G}$TRS that satisfies the scheme is Turing-complete, a property that systems without Ap would not possess.

For a type assignment system in which the type ω is not used, we will prove (adapting the method of Computability Predicates of Girard-Lafont-Taylor'89 and Tait'67) that for all typeable $\mathcal{G}$TRS satisfying the general scheme, typeable terms are strongly normalizable.

Perhaps surprisingly, in the type system with ω, the general scheme is not enough to ensure head-normalization of typeable terms. Therefore, to study head-normalization of typeable $\mathcal{G}$TRS we will define a suitable restriction of the general scheme, called the *HNF-scheme*, where the patterns of rewrite rules are constructor terms that have sorts as types. We should remark here that our notion of head-normal form for $\mathcal{G}$TRS is similar in spirit to the notion of weak head-normal form in LC; the latter is used in most functional programming languages based on LC, see [1]. We will again use the method of Computability Predicates to prove that for all typeable $\mathcal{G}$TRS satisfying the HNF-scheme, every typeable term has a head-normal form. We will also show that if Curryfication is not allowed, under certain restrictions, terms typeable with a type that does not contain ω are normalizable.

These results apply in particular to Combinator Systems, a class of $\mathcal{G}$TRS that satisfies the required conditions. For Combinator Systems that are combinatory complete, a type assignment system was defined by Dezani-Hindley'92. Our system can be seen as a generalization of that one.

The lay-out of this paper is as follows: We present $\mathcal{G}$TRS in Section 2. In Section 3 we briefly recall the essential intersection system for LC. In Section 4 we introduce the essential intersection system for $\mathcal{G}$TRS, and compare it with the one for LC. In Section 5 we present the general scheme and prove the strong normalization theorem for the type assignment system without ω. We then show that in the system with ω, and considering the restrictions formulated in the HNF-scheme, all typeable terms have a head-normal form. Finally we prove the normalization result. Section 6 contains the conclusions and directions for further work.

The results presented in this paper were first published, in a much condensed form, as [4, 8], and [9].

2 Curryfied Term Rewriting Systems

In this section we present Curryfied Term Rewriting Systems ($\mathcal{C}$TRS), an extension of the TRS defined by Klop'92, and Dershowitz-Jouannaud'90. $\mathcal{C}$TRS are first-order TRS extended with a binary function symbol which models partial application of functions. This feature allows us to make a straightforward translation of LC to $\mathcal{C}$TRS (as we will show in Definition 4.16 below), i.e. to a first-order rewrite system.

$\mathcal{C}$TRS are also an extension of the constructor systems used in most functional programming languages in that not only constructor symbols can be used in the operand space of the left-hand side of rewrite rules, but all function symbols.

Definition 2.1 An *alphabet* or *signature* Σ consists of a countable infinite set $\mathcal{X}$ of variables x, y, z, x', y', ...; a non-empty set $\mathcal{F}$ of *function symbols* F, G, ..., each equipped with an 'arity' (a natural number); and a special binary operator, called *application* (Ap).

Definition 2.2 The set $T(\mathcal{F},\mathcal{X})$ of *terms* is defined inductively by:

i) $\mathcal{X} \subseteq T(\mathcal{F},\mathcal{X})$.

ii) If $F \in \mathcal{F} \cup \{Ap\}$ is an n-ary symbol ($n \geq 0$) and $t_1, \ldots, t_n \in T(\mathcal{F},\mathcal{X})$, then $F(t_1,\ldots,t_n) \in T(\mathcal{F},\mathcal{X})$. The t_i ($1 \leq i \leq n$) are the *arguments* of the last term. We will omit the brackets when $n = 0$.

We will write $Var\,(t)$ for the set $\{x \in \mathcal{X} \mid x \text{ occurs in } t\}$.

We will introduce in some parts a notation different from the one commonly used in term rewriting, because some of the symbols were also used in papers about type assignment. For example, we will call 'term-substitution' the operation that replaces variables by terms, instead of just 'substitution' which will be used for operations that replace type-variables by types. To denote a term-substitution, we will use capital characters like 'R' instead of Greek characters like 'σ', which will be used to denote types.

Definition 2.3 A *term-substitution* R is a mapping from $T(\mathcal{F},\mathcal{X})$ to $T(\mathcal{F},\mathcal{X})$ satisfying

$$\mathrm{R}(F(t_1,\ldots,t_n)) = F(\mathrm{R}(t_1), \ldots, \mathrm{R}(t_n))$$

for every n-ary ($n \geq 0$) function symbol F, and is determined by its restriction to a finite set of variables. Sometimes we will use the notation $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ for term-substitutions. We will also write t^{R} instead of $\mathrm{R}(t)$.

Definition 2.4 *i*) Given a signature Σ with a set $\mathcal{X}$ of variables and a set $\mathcal{F}$ of function symbols, a *rewrite rule* in Σ is a pair (l,r) of terms in $T(\mathcal{F},\mathcal{X})$, such that l is not a variable, and the variables occurring in r appear in l. Often a rewrite rule will get a name, e.g. $\mathbf{r}$, and we will write $\mathbf{r} : l \to r$.

If $F(t_1,\ldots,t_n)$ is the left-hand side of a rule $\mathbf{r}$, and, for $1 \leq i \leq n$, either t_i is not a variable, or t_i is a variable and there is a $1 \leq i \neq j \leq n$ such that $t_i = t_j$, then t_i is called a *pattern* of $\mathbf{r}$.

ii) A *Curryfied Term Rewriting System* ($\mathcal{C}$TRS) is a pair $(\Sigma,\mathbf{R})$ of a signature $\Sigma = (\mathcal{F}, \mathcal{X})$ and a set $\mathbf{R}$ of rewrite rules in Σ, such that, for every $F \in \mathcal{F}$ of arity $n > 0$, there exist n additional function symbols $F_{n-1}, \ldots, F_1, F_0$ in $\mathcal{F}$, the *Curryfied-versions of* F, and $\mathbf{R}$ contains the n rewrite rules:

$$Ap(F_{n-1}(x_1,\ldots,x_{n-1}),x_n) \;\to\; F(x_1,\ldots,x_n)$$

$$\vdots$$

$$\begin{array}{ll} Ap(F_1(x_1),x_2) & \to F_2(x_1,x_2) \\ Ap(F_0,x_1) & \to F_1(x_1) \end{array}$$

If F_i is a Curryfied version of a function symbol F, then its Curryfied versions coincide with the corresponding Curryfied versions of F, being $F_{i-1},\ldots,F_0$. Moreover, we will assume that for any rule $\mathbf{r} : l \to r$ in $\mathbf{R}$, if Ap occurs in l, then $\mathbf{r}$ is of the shape:

$$Ap(F_{i-1}(x_1,\ldots,x_{i-1}),x_i) \quad \to F_i(x_1,\ldots,x_i)$$

for some Curryfied version F_{i-1}, and that Curryfied versions do not appear in the root of any left-hand side.

iii) Terms that do not contain Curryfied versions of symbols are called *non-Curryfied terms*.

iv) A rewrite rule $\mathbf{r} : l \to r$ determines a set of *reductions* $l^{\mathrm{R}} \to r^{\mathrm{R}}$ for all term-substitutions R. The term l^{R} is called a *redex*; it may be replaced by its *contractum* r^{R} inside any context C[ ]; this gives rise to *rewrite steps*:

$$\mathrm{C}[\,l^{\mathrm{R}}\,] \to_{\mathbf{r}} \mathrm{C}[\,r^{\mathrm{R}}\,].$$

We will write $t \to_{\mathbf{R}} t'$ if there is a $\mathbf{r} \in \mathbf{R}$ such that $t \to_{\mathbf{r}} t'$.

Concatenating rewrite steps we have (possibly infinite) *rewrite sequences* $t_0 \to t_1 \to \cdots$. If $t_0 \to \cdots \to t_n$ ($n \geq 0$) we will also write $t_0 \to^* t_n$.

Because of the extra rules for $F_{n-1}, \ldots, F_1, F_0$, etc., the rewrite systems are called *Curry-closed*. When presenting a rewrite system we will sometimes omit the rules that define the Curryfied versions.

Example 2.5 *Curryfied Combinatory Logic* (CCL) is defined as a $\mathcal{A}$TRS with function symbols $\mathcal{F}$ = $\{S, S_2, S_1, S_0, K, K_1, K_0, I, I_0\}$ and rewrite rules:

$$\begin{array}{ll} S(x,y,z) & \to Ap(Ap(x,z),Ap(y,z)) \\ Ap(S_2(x,y),z) & \to S(x,y,z) \\ Ap(S_1(x),y) & \to S_2(x,y) \\ Ap(S_0,x) & \to S_1(x) \\ K(x,y) & \to x \\ Ap(K_1(x),y) & \to K(x,y) \\ Ap(K_0,x) & \to K_1(x) \\ I(x) & \to x \\ Ap(I_0,x) & \to I(x). \end{array}$$

Because CCL is Curry-closed, it inherits combinatory completeness from Combinatory Logic: every λ-term can be translated into a term in CCL (see Definition 4.16).

Recently, Kahrs'96 and Kennaway-et.al-JSC'96 studied the class of rewrite systems obtained by Currying standard first-order term rewrite systems. The Currying of a system R, denoted by *PP(R)*, is defined as R extended with Ap, Curryfied versions of all function symbols, and the additional rules defining Curryfied versions. It is easy to see that *PP(R)* is a $\mathcal{A}$TRS, but note that $\mathcal{A}$TRS are more general; in particular, Ap can be freely used in right-hand sides, which allows us to code Combinatory Logic as a $\mathcal{A}$TRS.

Also the class of applicative systems (i.e. rewrite systems where the signature contains only Ap and 0-ary function symbols) can be seen as a particular case of $\mathcal{A}$TRS if a standard "consistency" restriction is satisfied. Roughly, the left-hand side of every rule has to be the Curryfication of a non-applicative term; otherwise, applicative systems are not $\mathcal{A}$TRS: they could have a left-hand side of an arbitrary form. Note that this condition is usually present in the definition of applicative system (see e.g. [29]).

Definition 2.6 A rewrite rule $\mathbf{r} : l \to r$ *defines* F if F is the leftmost, outermost symbol in l that is not an Ap; we call F *the defined symbol of* $\mathbf{r}$.

We say that $F \in \mathcal{F}$ is a *defined symbol* if there is a rewrite rule that defines F. Otherwise, it is a *constructor*.

We can draw the dependency-graph of the defined function symbols, i.e. we can construct a graph whose nodes are filled with the defined symbols of the rewrite rules, and draw an edge going from F to G if G occurs in the right-hand side of one of the rules that define F. We call a defined symbol F *recursive* if F occurs on a cycle in the dependency-graph, and call every rewrite rule that defines F *recursive*. All function symbols that occur on one cycle in the dependency-graph depend on each other and are, therefore, defined *simultaneously* and are called *mutually recursive*. Since it is always possible to introduce tuples into the language and solve the problem of mutual recursion using only recursive rules, we will assume that rules are *not* mutually recursive.

Definition 2.7 A TRS whose dependency-graph is an ordered a-cyclic graph is called a *hierarchical* TRS. The rewrite rules of a hierarchical TRS can be regrouped in such a way that they are *incremental* definitions of the defined symbols $F^1, \ldots, F^k$, so that the rules defining F^i only depend on $F^1, \ldots, F^i$.

Incremental definitions arise naturally in programming practice. We will see in Section 5 that hierarchical systems play an important role in the study of the normalization properties of $\mathcal{U}$TRS.

Example 2.8 Our definition of recursive symbols, using the notion of defined symbols, is different from the one normally considered. Since Ap is never a defined symbol, the following $\mathcal{U}$TRS

$$\begin{array}{ll} D(x) & \to Ap(x,x) \\ Ap(D_0,x) & \to D(x) \end{array}$$

is *not* considered a recursive system. Notice that, for example, the term $D(D_0)$ has no normal form (this term plays the role of $(\lambda x.xx)(\lambda x.xx)$ in LC). This means that, in the formalism of this paper, there exist non-recursive first-order rewrite systems that are not normalizing.

Definition 2.9 *i*) A term is *neutral* if it is not of the form $F_i(t_1,\ldots,t_i)$, where F_i is a Curryfied version of a function symbol F.

ii) A term is in *normal form* if it is irreducible.

iii) A term t is in *head-normal form* if for all t' such that $t \to^* t'$:

- *a*) t' is not itself a redex and
- *b*) if $t' = Ap(v,u)$, then v is in head-normal form.

iv) A term t is in *constructor-hat normal form* if either

- *a*) $t = \mathrm{C}[u_1,\ldots,u_n]$ where C is a context (possibly empty) that contains only constructor symbols and, for $1 \le i \le n$, u_i cannot be reduced to a term of the form $Q(s_1,\ldots,s_i)$, where Q is a constructor, or
- *b*) $t = Ap(t_1,t_2)$ and t_1 is in constructor-hat normal form.

v) A term is *(head/constructor-hat) normalizable* if it can be reduced to a term in (head/constructor-hat) normal form.

vi) A rewrite system is *strongly normalizing* (or terminating) if all the rewrite sequences are finite; it is *(head/constructor-hat) normalizing* if every term is (head/constructor-hat) normalizable.

Example 2.10 Take the $\mathcal{U}$TRS

$$\begin{array}{ll} F(x,x) & \to A(x) \\ B(H) & \to H \\ H & \to H \end{array}$$

where F, B, and H are defined symbols, and A is a constructor (notice the use of a defined symbol as a pattern in the second rule). The term $F(B(H),H)$ is not a redex, but it is not a head-normal form either, since it reduces to $F(H,H)$ which is a redex. This term reduces to $A(H)$, which is a head-normal form and a constructor-hat normal form. $A(F(x,x))$ is also a head-normal form, but it is not a constructor-hat normal form since it reduces to $A(A(x))$. The latter is in constructor-hat normal form.

Our definition of head-normal form is an extension to rewrite systems with Ap of the notion of root stable form defined by Ariola-et.al-TACS'94. Note that the head of a term of the form $Ap(v,u)$ is in v, since we can think of Ap as an invisible symbol.

This notion of head-normal form corresponds to the notion of weak head-normal form in LC. For instance, if F is a function symbol of arity n, $F_i(t_1,\ldots,t_i)$ is a head-normal form according to the previous definition. Clearly it corresponds to the λ-term

$$\lambda x_{i+1}\ldots x_n.F(t_1,\ldots,t_i,x_{i+1},\ldots,x_n),$$

which is in weak head-normal form.

The notion of constructor-hat normal form is introduced for technical reasons only (constructor-hat normal forms are used in the proof of the Head Normalization Theorem in Section 5.2).

The notations *In-Chnf*(t), *In-Hnf*(t), and *In-Nf*(t) will indicate that t is in constructor-hat normal form, in head-normal form, and in normal form, respectively. The notations $CHN(t)$, $HN(t)$, and $N(t)$ will indicate that t is constructor-hat normalizable, head-normalizable, and normalizable, respectively.

Lemma 2.11 *i)* $HN(Ap(t,u)) \Rightarrow HN(t)$.
ii) $CHN(Ap(t,u)) \Rightarrow CHN(t)$.
iii) *t is neutral & In-Hnf*$(t) \Rightarrow \forall u\ [\textit{In-Hnf}(Ap(t,u))]$.
iv) *t is neutral & In-Chnf*$(t) \Rightarrow \forall u\ [\textit{In-Chnf}(Ap(t,u))]$.
v) *t is neutral* $\Rightarrow \forall u\ [Ap(t,u)$ *is neutral*$]$.

Proof: This is a direct consequence of the definition of neutral term, head-normal form, and constructor-hat normal form (Definition 2.9). ■

3 Essential Intersection System for LC

In this section we present a variant of the essential intersection type assignment system for LC [6] which will serve as a mould to define our type assignment system for $\mathcal{U}$TRS in the following section.

The essential system uses a set of *strict intersection types* (built from a set of *type variables*) which are the representatives of the equivalence classes of types considered in the system of Barendregt-Coppo-Dezani'83. The variant we present in this section uses also a set of *sorts* (names of domains; the constant types of our system). Sorts will play an important role in the proof of the Head Normalization Theorem for $\mathcal{G}$TRS.

Definition 3.1 *i*) $\mathcal{T}_s$, the set of *strict types*, and $\mathcal{T}_S$, the set of *strict intersection types*, are defined through mutual induction by:

a) All type-variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}_s$.

b) All sorts $s_1, s_2, \ldots \in \mathcal{T}_s$.

c) If $\tau \in \mathcal{T}_s$ and $\sigma \in \mathcal{T}_S$, then $\sigma \to \tau \in \mathcal{T}_s$.

d) If $\sigma_1, \ldots, \sigma_n \in \mathcal{T}_s$ $(n \geq 0)$, then $\sigma_1 \cap \cdots \cap \sigma_n \in \mathcal{T}_S$.

ii) The type ω is defined as an intersection of zero types: if $n = 0$, then $\sigma_1 \cap \cdots \cap \sigma_n = \omega$.

iii) On $\mathcal{T}_S$, the relation $\leq$ is defined by:

a) $\forall\, n \geq 1,\ 1 \leq i \leq n\ [\sigma_1 \cap \cdots \cap \sigma_n \leq \sigma_i]$.

b) $\forall\, n \geq 0\ [\forall\, 1 \leq i \leq n\ [\sigma \leq \sigma_i] \ \Rightarrow\ \sigma \leq \sigma_1 \cap \cdots \cap \sigma_n]$.

c) $\sigma \leq \tau \leq \rho \ \Rightarrow\ \sigma \leq \rho$.

d) $\rho \leq \sigma \ \&\ \tau \leq \mu \ \Rightarrow\ \sigma \to \tau \leq \rho \to \mu$.

iv) The relation $\sim$ is defined by: $\sigma \sim \tau \iff \sigma \leq \tau \leq \sigma$.

The motivation for defining ω as an intersection of zero types lies in the semantics of types (see [14]), where $[\![\sigma]\!]$ is the set of terms that can be assigned the type σ. Then, for all $\sigma_1, \ldots, \sigma_n$,

$$[\![\sigma_1 \cap \ldots \cap \sigma_n]\!] \ \subseteq [\![\sigma_1 \cap \ldots \cap \sigma_{n-1}]\!] \ \subseteq \ldots \subseteq [\![\sigma_1 \cap \sigma_2]\!] \ \subseteq [\![\sigma_1]\!],$$

so it is natural to extend this sequence with $[\![\sigma_1]\!] \subseteq [\![\ \ ]\!]$, and therefore to define that the semantics of the empty intersection is the whole set of λ-terms, which is exactly $[\![\omega]\!]$.

Notice that intersection types (so also ω) occur in strict types only as subtypes at the left-hand side of an arrow type. According to the previous definition, if $\sigma_1 \cap \cdots \cap \sigma_n$ is used to denote a type, then all $\sigma_1, \ldots, \sigma_n$ are strict, therefore they cannot be ω. Notice also that $\mathcal{T}_s$ is a proper subset of $\mathcal{T}_S$.

To obtain readable types, instead of φ_i we often write only the number i.

Definition 3.2 *i*) A *statement* is an expression of the form $M{:}\sigma$, where M is a λ-term and $\sigma \in \mathcal{T}_S$. M is the *subject* and σ the *predicate* of $M{:}\sigma$.

ii) A *basis* is a set of statements with only distinct variables as subjects. If $\sigma_1 \cap \cdots \cap \sigma_n$ is a predicate in a basis, then $n \geq 1$.

The relations $\leq$ and $\sim$ extend to bases in the natural way: $B \leq B' \iff \forall\, x{:}\sigma' \in B'\ \exists\, x{:}\sigma \in B\ [\sigma \leq \sigma']$, and $B \sim B' \iff B \leq B' \leq B$.

iii) If B is a basis and $\sigma \in \mathcal{T}_S$, then $\mathcal{T}_{\langle B,\sigma\rangle}$ is the set of all strict subtypes occurring in the pair $\langle B, \sigma\rangle$.

iv) If $B_1, \ldots, B_n$ are bases, then $\Pi\{B_1, \ldots, B_n\}$ is the basis defined as follows: $x{:}\sigma_1 \cap \cdots \cap \sigma_m \in \Pi\{B_1, \ldots, B_n\}$ if and only if $\{x{:}\sigma_1, \ldots, x{:}\sigma_m\}$ is the (non-empty) set of all statements whose subject is x that occur in $B_1 \cup \ldots \cup B_n$.

Notice that if $n = 0$, then $\Pi\{B_1, \ldots, B_n\} = \emptyset$.

Often $B \cup \{x{:}\sigma\}$ (or $B, x{:}\sigma$) will be written for the basis $\Pi\{B, \{x{:}\sigma\}\}$, when x does not occur in B.

Definition 3.3 *i*) *Type assignment* and *derivations* in the essential system for LC are defined by the following natural deduction system (where all types displayed belong to $\mathcal{T}_s$, except σ in the derivation rules ($\rightarrow$I), ($\rightarrow$E), and ($\leq$)):

$$(\rightarrow\text{I}):\quad \frac{\begin{array}{c}[x{:}\sigma]\\ \vdots\\ M{:}\tau\end{array}}{\lambda x.M{:}\sigma\rightarrow\tau}\ (a) \qquad\qquad (\rightarrow\text{E}):\quad \frac{M{:}\sigma\rightarrow\tau \quad N{:}\sigma}{MN{:}\tau}$$

$$(\leq):\quad \frac{x{:}\sigma \quad \sigma\leq\tau}{x{:}\tau}$$

$$(\cap\text{I}):\quad \frac{M{:}\sigma_1 \quad \ldots \quad M{:}\sigma_n}{M{:}\sigma_1\cap\cdots\cap\sigma_n}\ (n\geq 0)$$

(a): If $x{:}\sigma$ is the only statement about x on which $M{:}\tau$ depends.

ii) We write $B \vdash M{:}\sigma$ if there exists a derivation built using the rules given above that has the statement $M{:}\sigma$ as its conclusion, and B contains at least all statements for the free variables of M that occur in the leaves of this derivation.

4 Essential Intersection System for $\mathcal{G}$TRS

In this section we present an intersection type assignment on $\mathcal{G}$TRS that is inspired by the essential system for LC. It is a *partial* system in the sense of Pfenning'88: not only will we define how terms and rewrite rules can be typed, but we will also assume that every function symbol in the signature *has* a type, provided by an *environment* (i.e. a mapping from function symbols to types). There are several reasons to do so.

First of all, a term rewrite system can contain constructors, i.e. symbols that are not defined by the rewrite rules. If the environment provides a type for every constructor, it is possible to (partially) check the consistency of the system, by checking that the types used for a constructor are related to the provided type.

Moreover, for every defined symbol there must be some way of determining what type can be used for an occurrence. If no type is given for a function symbol, the rewrite rules that define that symbol have to be investigated, and from analyzing the structure of those rules the 'most general type' for that symbol is constructed. Instead of investigating all the defining rules for a defined symbol every time the symbol is encountered, we can store the type of the symbol in an environment. Of course it makes no difference to assume the existence from the start of such a mapping from symbols to types, and to define type assignment using it.

In fact, the approach we take here is very much the same as the one taken by Hindley'69 to define the principal Curry type of an object in Combinatory Logic. Even that notion of type assignment could be regarded as a partial one. Also the notion of type assignment on (combinatory complete) combinator systems of Dezani-Hindley'92 uses this approach. An important difference is, however, that there the environment can contain only 'principal types' for combinators, i.e. the principal type of the λ-term that can naturally be associated to that combinator (for a presentation of principal intersection types for λ-terms, see [5]).

4.1 Bases, Types, and Operations

Types and bases are defined as for LC (Section 3). To assign types to terms in the $\mathcal{G}$TRS framework, we are going to use three *operations* on types (that extend to bases and to pairs of

$\langle basis, type \rangle$), namely *substitution*, *expansion*, and *lifting*. These will be taken from [5], where also the here cited properties are proved.

Roughly, substitution is the operation that instantiates a type (replacing type-variables by types), expansion replaces types by the intersection of a number of copies of that type, and the operation of lifting replaces basis and type by a smaller basis and a larger type, in the sense of $\leq$.

These three operations are of use in Definition 4.10, when we want to specify how, for a specific function symbol, a type required by the context can be obtained from the type provided for that symbol by the environment. It is possible to define sound type assignment with fewer or less powerful operations on types, but in order to obtain enough expressive power to prove Property 4.18, all operations specified here are needed.

In the essential type assignment system, substitution has to be defined carefully to avoid going out of the set of strict intersection types. For example, the replacement of φ by ω would transform $\sigma \rightarrow \varphi$ into $\sigma \rightarrow \omega$, which is not in $\mathcal{T}_S$. The following definition takes this fact into account.

Definition 4.1 The *substitution* $(\varphi \mapsto \alpha) : \mathcal{T}_S \rightarrow \mathcal{T}_S$, where φ is a type-variable and $\alpha \in \mathcal{T}_s \cup \{\omega\}$, is defined by:

i) $(\varphi \mapsto \alpha)(\varphi) = \alpha$,

ii) $(\varphi \mapsto \alpha)(\varphi') = \varphi'$, if $\varphi \neq \varphi'$,

iii) $(\varphi \mapsto \alpha)(s) = s$.

iv) $(\varphi \mapsto \alpha)(\sigma \rightarrow \tau) = \omega$, if $(\varphi \mapsto \alpha)(\tau) = \omega$,

v) $(\varphi \mapsto \alpha)(\sigma \rightarrow \tau) = (\varphi \mapsto \alpha)(\sigma) \rightarrow (\varphi \mapsto \alpha)(\tau)$,
if $(\varphi \mapsto \alpha)(\tau) \neq \omega$,

vi) $(\varphi \mapsto \alpha)(\sigma_1 \cap \cdots \cap \sigma_n) = (\varphi \mapsto \alpha)(\tau_1) \cap \cdots$
$\cap (\varphi \mapsto \alpha)(\tau_m)$, where $\{\tau_1, \ldots, \tau_m\} = \{\sigma_i \in \{\sigma_1, \ldots, \sigma_n\} \mid (\varphi \mapsto \alpha)(\sigma_i) \neq \omega\}$.

We will use S to denote a generic substitution. Substitutions extend to bases and to pairs of basis and type in the natural way:

i) $S(B) = \{x{:}S(\alpha) \mid x{:}\alpha \in B \ \& \ S(\alpha) \neq \omega\}$.

ii) $S(\langle B, \sigma \rangle) = \langle S(B), S(\sigma) \rangle$.

Notice that, in part (*vi*), if for $1 \leq i \leq n$, $(\varphi \mapsto \alpha)(\sigma_i) = \omega$, then $(\varphi \mapsto \alpha)(\sigma_1 \cap \cdots \cap \sigma_n) = \omega$.

For substitutions, the following properties hold:

Property 4.2 Let S be a substitution.

i) If $\sigma \leq \tau$, then $S(\sigma) \leq S(\tau)$.

ii) If $B \leq B'$, then $S(B) \leq S(B')$. ■

The operation of expansion deals with the replacement of (sub)types of a type by an intersection of a number of copies of that subtype. In this process, new variables are generated (the notion of *last type variable* in a type, i.e. the rightmost variable occurring in a type, plays an important role in this operation). For more details on the complexity of expansion, see [5].

Definition 4.3 The *last type-variable* of a strict type is defined by:

i) The last type-variable of φ is φ.

ii) s has no last type-variable.

iii) The last type-variable of $\sigma \rightarrow \tau$ is the last type-variable of τ.

An expansion indicates not only the type to be expanded, but also the number of copies that has to be generated.

Definition 4.4 For every $\mu \in \mathcal{T}_s$, $n \geq 2$, basis B and $\sigma \in \mathcal{T}_S$, the quadruple $\langle \mu, n, B, \sigma \rangle$ determines an *expansion* $Exp_{\langle \mu,n,B,\sigma \rangle} : \mathcal{T}_S \rightarrow \mathcal{T}_S$, that is constructed as follows.

i) The set of type-variables $\mathcal{V}_\mu(\langle B,\sigma \rangle)$ is constructed by:
  a) If φ occurs in μ, then $\varphi \in \mathcal{V}_\mu(\langle B,\sigma \rangle)$.
  b) Let τ be a strict (sub)type occurring in $\langle B,\sigma \rangle$, with last type-variable φ_0. If $\varphi_0 \in \mathcal{V}_\mu(\langle B,\sigma \rangle)$, then for all type-variables φ that occur in τ, $\varphi \in \mathcal{V}_\mu(\langle B,\sigma \rangle)$.

ii) Suppose $\mathcal{V}_\mu(\langle B,\sigma \rangle) = \{\varphi_1, \ldots, \varphi_m\}$. Choose $m \times n$ different type-variables $\varphi_1^1, \ldots, \varphi_1^n, \ldots, \varphi_m^1, \ldots, \varphi_m^n$, such that each φ_j^i does not occur in $\langle B,\sigma \rangle$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Let S_i be the substitution that replaces every φ_j by φ_j^i.

iii) $Exp_{\langle \mu,n,B,\sigma \rangle}(\alpha)$ is obtained by traversing α top-down and replacing, in α, a subtype β, with last type-variable in $\mathcal{V}_\mu(\langle B,\sigma \rangle)$, by $S_1(\beta) \cap \cdots \cap S_n(\beta)$.

The operation of expansion extends to bases and to pairs of basis and type in the natural way:

i) $Exp_{\langle \mu,n,B,\sigma \rangle}(B') = \{x{:}Exp_{\langle \mu,n,B,\sigma \rangle}(\rho) \mid x{:}\rho \in B'\}$.

ii) $Exp_{\langle \mu,n,B,\sigma \rangle}(\langle B',\sigma' \rangle) = \langle Exp_{\langle \mu,n,B,\sigma \rangle}(B'), Exp_{\langle \mu,n,B,\sigma \rangle}(\sigma') \rangle$.

Instead of $Exp_{\langle \mu,n,B,\sigma \rangle}$, the notation $\langle \mu,n,B,\sigma \rangle$ will be used.

Example 4.5 Let γ be $(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_3 \rightarrow \varphi_1) \rightarrow \varphi_3 \rightarrow \varphi_2$, and E the expansion $\langle \varphi_1, 2, \emptyset, \gamma \rangle$. Then $\mathcal{V}_{\varphi_1}(\langle \emptyset, \gamma \rangle) = \{\varphi_1, \varphi_3\}$, and $E(\gamma) = ((\varphi_4 \cap \varphi_5) \rightarrow \varphi_2) \rightarrow ((\varphi_6 \rightarrow \varphi_4) \cap (\varphi_7 \rightarrow \varphi_5)) \rightarrow (\varphi_6 \cap \varphi_7) \rightarrow \varphi_2$.

For an operation of expansion the following properties hold:

Property 4.6 Let $E = \langle \mu,n,B,\sigma \rangle$ be an expansion.

i) If $\tau \in \mathcal{T}_{\langle B,\sigma \rangle}$, then either:
  a) $E(\tau) = \tau_1 \cap \cdots \cap \tau_n$, with for every $1 \leq i \leq n$, τ_i is a trivial variant of τ, or:
  b) $E(\tau) \in \mathcal{T}_s$.

ii) $E(\Pi\{B_1, \ldots, B_n\}) = \Pi\{E(B_1), \ldots, E(B_n)\}$.

iii) If $\sigma \leq \tau$, then $E(\sigma) \leq E(\tau)$.

iv) If $B \leq B'$, then $E(B) \leq E(B')$. ■

The last operation defined in this subsection is the operation of lifting.

Definition 4.7 An operation of *lifting* is denoted by a pair $L = <\langle B_0, \tau_0 \rangle, \langle B_1, \tau_1 \rangle>$ such that $\tau_0 \leq \tau_1$ and $B_1 \leq B_0$. L can be applied to a type, a basis, or a pair of basis and type:

i) $L(\sigma) = \tau_1$ if $\sigma = \tau_0$; $L(\sigma) = \sigma$ otherwise.

ii) $L(B) = B_1$ if $B = B_0$; $L(B) = B$ otherwise.

iii) $L(\langle B,\sigma \rangle) = \langle L(B), L(\sigma) \rangle$.

For liftings, the following properties hold:

Property 4.8 *i)* $<\langle B\cup\{x{:}\sigma\},\tau\rangle, \langle B\cup\{x{:}\sigma'\},\tau'\rangle>$ *is a lifting if and only if* $<\langle B,\sigma\to\tau\rangle, \langle B',\sigma'\to\tau'\rangle>$ *is a lifting (where* τ, $\tau' \in \mathcal{T}_s$*).*

ii) $<\langle B_i,\sigma_i\rangle, \langle B'_i,\sigma'_i\rangle>$ *is a lifting for every* $1\le i\le n$*, if and only if*

$$\langle\langle \Pi\{B_1,\ldots,B_n,\{x{:}\sigma_1\to\cdots\to\sigma_n\to\varphi\}\},\varphi\rangle, \langle \Pi\{B'_1,\ldots,B'_n,\{x{:}\sigma'_1\to\cdots\to\sigma'_n\to\varphi\}\},\varphi\rangle\rangle$$

is a lifting. ■

The operations of substitution, expansion, and lifting can be composed to form *chains* of operations. The *set* $\mathcal{C}h$ *of chains* is defined as the smallest set containing expansions, substitutions, and liftings, that is closed under composition $\circ$.

4.2 Type Assignment in $\mathcal{G}u$TRS

The three operations on types defined above will be used in this subsection to define type assignment on $\mathcal{G}u$TRS: the types assigned to occurrences of function symbols will be obtained from the type provided by the environment by making a chain of operations.

We will start by defining an environment, which is a mapping from function symbols to strict types. Since we want to associate the Curryfied versions of a function symbol not only through their rewrite rules, but also through their assignable types, we will require that the environment maps a function F and all its Curryfied versions F_i to the same type.

Definition 4.9 Let $(\Sigma,\mathbf{R})$ be a $\mathcal{G}u$TRS, and $\mathcal{F}$ the set of function symbols in Σ.

i) A mapping $\mathcal{E} : \mathcal{F}\cup\{Ap\} \to \mathcal{T}_s$ is called an *environment* if $\mathcal{E}(Ap) = (\varphi_1\to\varphi_2)\to\varphi_1\to\varphi_2$ and, for every $F\in\mathcal{F}$ with arity n, $\mathcal{E}(F) = \mathcal{E}(F_{n-1}) = \cdots = \mathcal{E}(F_0)$.

ii) For $F\in\mathcal{F}$ with arity $n\ge 0$, $\sigma\in\mathcal{T}_s$, and $\mathcal{E}$ an environment, the environment $\mathcal{E}[F{:}\sigma]$ is defined by:

a) $\mathcal{E}[F{:}\sigma](G) = \sigma$, if $G\in\{F, F_{n-1}, \ldots, F_0\}$.

b) $\mathcal{E}[F{:}\sigma](G) = \mathcal{E}(G)$, otherwise.

Since $\mathcal{E}$ maps all $F\in\mathcal{F}$ to types in $\mathcal{T}_s$, in particular no function symbol is mapped to ω.

In the following we will assume that $\mathcal{E}$ is a given environment for a $\mathcal{G}u$TRS $(\Sigma,\mathbf{R})$.

Definition 4.10 *i)* *Type assignment* and *derivations* are defined by the following natural deduction system (where all types displayed are in $\mathcal{T}_s$, except for σ in rule ($\le$) and $\sigma_1,\ldots,\sigma_n$ in rule ($\to$E)):

$$(\le):\quad \frac{x{:}\sigma \quad \sigma\le\tau}{x{:}\tau} \qquad\qquad (\cap\mathrm{I}):\quad \frac{t{:}\sigma_1 \quad \ldots \quad t{:}\sigma_n}{t{:}\sigma_1\cap\cdots\cap\sigma_n}\ (n\ge 0)$$

$$(\to\mathrm{E}):\quad \frac{t_1{:}\sigma_1 \quad \ldots \quad t_n{:}\sigma_n}{F(t_1,\ldots,t_n){:}\sigma}\ (a)$$

$((a))$: If there exists $C\in\mathcal{C}h$ such that $C(\mathcal{E}(F)) = \sigma_1\to\cdots\to\sigma_n\to\sigma$.

ii) We write $B\vdash_{\mathcal{E}} t{:}\sigma$ if and only if $t{:}\sigma$ is derivable from B by using the natural deduction system above. We will say that *t is typeable with respect to* $\mathcal{E}$ (or simply that t is typeable, if the environment is clear from the context) if there exists a basis B and a type $\sigma\ne\omega$ such that $B\vdash_{\mathcal{E}} t{:}\sigma$.

Notice that if $B\vdash_{\mathcal{E}} t{:}\sigma$, then B can contain more statements than needed to obtain $t{:}\sigma$. Moreover, by ($\cap$I), for every B and t, $B\vdash_{\mathcal{E}} t{:}\omega$.

The use of an environment in derivation rule ($\to$E) introduces a notion of polymorphism into our type assignment system. The environment returns the 'principal type' for a function symbol; this symbol can be used with types that are 'instances' of its principal type, obtained by applying chains of operations.

Example 4.11 *i*) Let $\mathcal{E}_{\mathrm{CL}}$ be the environment (remember that instead of φ_i we just write the number i)

$\mathcal{E}_{\mathrm{CL}}(S) = (1\to 2\to 3)\to(4\to 2)\to 1\cap 4\to 3$,
$\mathcal{E}_{\mathrm{CL}}(K) = 5\to\omega\to 5$,
$\mathcal{E}_{\mathrm{CL}}(I) = 6\to 6$.

(See also Example 4.15). With respect to this environment, the term $S(K_0, S_0, I_0)$ can be assigned the type $7\to 7$.

$$\dfrac{\dfrac{}{K_0{:}(7\to 7)\to\omega\to 7\to 7}\,(\to\mathrm{E}) \quad \dfrac{}{S_0{:}\omega}\,(\cap\mathrm{I}) \quad \dfrac{}{I_0{:}7\to 7}}{S(K_0, S_0, I_0){:}7\to 7}\,(\to\mathrm{E})\,(\to\mathrm{E})$$

Notice that, for example, to obtain

$$((7\to 7)\to\omega\to 7\to 7)\to\omega\to(7\to 7)\to 7\to 7$$

for S, we have used the chain

$$(1\mapsto 7\to 7)\circ(2\mapsto\omega)\circ(3\mapsto 7\to 7)\circ(4\mapsto\omega).$$

ii) If we define $\mathcal{E}_{\mathrm{CL}}(D) = (1\to 2)\cap 1\to 2$, then we can even check that for example $D(S(K_0, S_0, I_0))$ and $D(I_0)$ are both typeable by $8\to 8$, as shown by the following derivations, where $\sigma = 8\to 8$, and $\tau = (8\to 8)\to 8\to 8$:

$$\dfrac{\dfrac{\dfrac{}{I_0{:}\tau} \quad \dfrac{}{I_0{:}\sigma}}{I_0{:}\tau\cap\sigma}}{D(I_0){:}\sigma} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{K_0{:}\tau\to\omega\to\tau} \quad \dfrac{}{S_0{:}\omega} \quad \dfrac{}{I_0{:}\tau}}{S(K_0, S_0, I_0){:}\tau} \quad \dfrac{\dfrac{}{K_0{:}\sigma\to\omega\to\sigma} \quad \dfrac{}{S_0{:}\omega} \quad \dfrac{}{I_0{:}\sigma}}{S(K_0, S_0, I_0){:}\sigma}}{S(K_0, S_0, I_0){:}\tau\cap\sigma}}{D(S(K_0, S_0, I_0)){:}\sigma}$$

iii) Take the environment

$\mathcal{E}(F) = (1\to 2)\to(3\to 1)\to 3\to 2$,
$\mathcal{E}(D) = (4\to 5)\cap 4\to 5$,
$\mathcal{E}(I) = 6\to 6$.

Then the term $F(D_0, I_0, I_0)$ is typeable by $7\to 7$ ($= \alpha$).

$$\dfrac{\dfrac{}{D_0{:}(\alpha\to\alpha)\cap\alpha\to\alpha} \quad \dfrac{\dfrac{}{I_0{:}(\alpha\to\alpha)\to\alpha\to\alpha} \quad \dfrac{}{I_0{:}\alpha\to\alpha}}{I_0{:}((\alpha\to\alpha)\to\alpha\to\alpha)\cap(\alpha\to\alpha)} \quad \dfrac{\dfrac{}{I_0{:}\alpha\to\alpha} \quad \dfrac{}{I_0{:}\alpha}}{I_0{:}(\alpha\to\alpha)\cap\alpha}}{F(D_0, I_0, I_0){:}7\to 7}$$

To obtain the type used for F in this derivation from $\mathcal{E}(F)$, the first step is to perform the expansion $\langle\varphi_1, 2, \emptyset, \mathcal{E}(F)\rangle$.

The following properties are needed further on:

Lemma 4.12 *i) If $B \vdash_{\mathcal{E}} t{:}\sigma$ and $B' \leq B$, then $B' \vdash_{\mathcal{E}} t{:}\sigma$.*
ii) If $B \vdash_{\mathcal{E}} t{:}\sigma$ and $\sigma \leq \tau$, then $B \vdash_{\mathcal{E}} t{:}\tau$.
iii) If $B, x{:}\alpha \vdash_{\mathcal{E}} Ap(t, x){:}\beta$, $\beta \in \mathcal{T}_s$, and x does not occur in t, then $B \vdash_{\mathcal{E}} t{:}\alpha\to\beta$.
iv) $B \vdash_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n$, if and only if, $B \vdash_{\mathcal{E}} t{:}\sigma_i$, for all $1 \leq i \leq n$.

v) For any Curryfied version F_n of a function symbol F, if $B \vdash_{\mathcal{E}} F_n(t_1,\ldots,t_n){:}\sigma$ and $\sigma \in \mathcal{T}_s$, then there are $\alpha \in \mathcal{T}_S$, $\beta \in \mathcal{T}_s$ such that $\sigma = \alpha \rightarrow \beta$.

Proof: By induction on derivations. ■

We will now define the type assignment for rewrite rules. This will be done in a careful way to ensure that the subject reduction property (i.e. preservation of types under rewriting) holds.

In [10] and [7] two restrictions of the type assignment system defined above are discussed, for which there is a decidable and sufficient condition on rewrite rules that ensures subject reduction. The condition a rewrite rule should satisfy is that the *principal pair* for the left-hand side is also a correct pair for the right-hand side of the rule. The notion of principal pair is in those papers defined in a constructive way, by defining a unification algorithm for types and defining principal pairs using that algorithm.

Since at this moment there is no general unification algorithm for types in $\mathcal{T}_S$ that works well on all types, we cannot take this approach here. Therefore, for the notion of type assignment defined in this paper we will show (in Subsection 4.4 below) that *if* a left-hand side of a rewrite rule has *a* principal pair and using that pair the rewrite rule can be typed, then rewriting using this rule preserves types.

Definition 4.13 A pair $\langle P,\pi\rangle$ is called *a principal pair for a term t with respect to an environment $\mathcal{E}$* if $P \vdash_{\mathcal{E}} t{:}\pi$ and, for every B, σ such that $B \vdash_{\mathcal{E}} t{:}\sigma$, there is a chain C such that $C(\langle P,\pi\rangle) = \langle B,\sigma\rangle$.

Definition 4.14 Let $(\Sigma,\mathbf{R})$ be a $\mathcal{G}$TRS, and $\mathcal{E}$ an environment.

i) We will say that $l \rightarrow r \in \mathbf{R}$ with defined symbol F *is typeable with respect to $\mathcal{E}$*, if there are basis P, type $\pi \in \mathcal{T}_s$, and an assignment of types to l and r such that:

a) $\langle P,\pi\rangle$ is a principal pair for l with respect to $\mathcal{E}$, and $P \vdash_{\mathcal{E}} r{:}\pi$.

b) In $P \vdash_{\mathcal{E}} l{:}\pi$ and $P \vdash_{\mathcal{E}} r{:}\pi$, the type actually used for each occurrence of F (or Curryfied versions of F) is $\mathcal{E}(F)$.

ii) We will say that $(\Sigma,\mathbf{R})$ *is typeable with respect to $\mathcal{E}$*, if every $\mathbf{r} \in \mathrm{R}$ is typeable with respect to $\mathcal{E}$.

Condition (*i.b*) of Definition 4.14 is in fact added to make sure that the type provided by the environment for a function symbol F is not in conflict with the rewrite rules that define F. Since by part (*i.b*) of Definition 4.14, all occurrences of the defined symbol in a rewrite rule are typed with the same type, type assignment of rewrite rules is actually defined using Milner's way of dealing with recursion [32].

It is easy to check that if F is a function symbol with arity n and all rewrite rules that define F are typeable, then there are $\gamma_1,\ldots,\gamma_n$, γ such that $\mathcal{E}(F) = \gamma_1 \rightarrow \cdots \rightarrow \gamma_n \rightarrow \gamma$.

Example 4.15 Type assignment for some of the rewrite rules given in Example 2.5, under the assumption that:

$\mathcal{E}(S) = (1\rightarrow 2\rightarrow 3)\rightarrow(4\rightarrow 2)\rightarrow 1\cap 4\rightarrow 3$,
$\mathcal{E}(K) = 5\rightarrow\omega\rightarrow 5$,
$\mathcal{E}(I) = 6\rightarrow 6$.

$$\frac{x{:}1\to2\to3 \qquad y{:}4\to2 \qquad z{:}1\cap4}{S\,(x,y,z){:}3} \quad\to\quad \frac{\dfrac{x{:}1\to2\to3 \qquad \dfrac{z{:}1\cap4}{z{:}1}}{Ap\,(x,z){:}2\to3} \qquad \dfrac{y{:}4\to2 \qquad \dfrac{z{:}1\cap4}{z{:}4}}{Ap\,(y,z){:}2}}{Ap\,(Ap\,(x,z),Ap\,(y,z)){:}3}$$

$$\frac{x{:}5 \qquad \overline{y{:}\omega}}{K(x,y){:}5} \quad\to\quad x{:}5 \qquad\qquad \frac{x{:}6}{I(x){:}6} \quad\to\quad x{:}6$$

4.3 Relating the essential system for $\mathcal{G}u$TRS with the essential system for LC

As remarked after Example 2.5, every λ-term can be translated into a term in CCL. In the setting of $\mathcal{G}u$TRS, the translation is specified by:

Definition 4.16 ([13]) Let Λ be the set of λ-terms and $T_{\mathrm{CCL}} = T(\{S,S_2,S_1,S_0,K,K_1,K_0,I,I_0\},\mathcal{X})$.

i) The mapping $\langle\ \rangle : \Lambda \to T_{\mathrm{CCL}}$ is defined by:

$$\begin{array}{rcl} \langle x\rangle & = & x, \\ \langle \lambda x.M\rangle & = & \lambda^* x.\langle M\rangle, \\ \langle MN\rangle & = & Ap\,(\langle M\rangle,\langle N\rangle). \end{array}$$

where $\lambda^* x.t$, with $t \in T_{\mathrm{CCL}}$, is defined by induction on the structure of M:

$$\begin{array}{rcl} \lambda^* x.x & = & I_0, \\ \lambda^* x.t & = & Ap\,(K_0,t), \text{ if } x \text{ not in } t, \\ \lambda^* x.Ap\,(t,u) & = & Ap\,(Ap\,(S_0,\lambda^* x.t),\lambda^* x.u). \end{array}$$

ii) The mapping $[\![\ ]\!]_{\mathrm{CL}} : T_{\mathrm{CCL}} \to \Lambda$ is defined by:

$$\begin{array}{rcl} [\![x]\!]_{\mathrm{CL}} & = & x, \\ [\![Ap\,(t_1,t_2)]\!]_{\mathrm{CL}} & = & [\![t_1]\!]_{\mathrm{CL}}[\![t_2]\!]_{\mathrm{CL}}, \\ [\![S_0]\!]_{\mathrm{CL}} & = & \lambda xyz.xz(yz), \\ [\![K_0]\!]_{\mathrm{CL}} & = & \lambda xy.x, \\ [\![I_0]\!]_{\mathrm{CL}} & = & \lambda x.x, \\ [\![F_n\,(t_1,\ldots,t_n)]\!]_{\mathrm{CL}} & = & [\![Ap\,(F_{n-1}(t_1,\ldots,t_{n-1}),t_n)]\!]_{\mathrm{CL}}, \\ & & \text{for } F_n \in \{S,S_2,S_1,K,K_1,I\}. \end{array}$$

Notice that the auxiliary function λ^*, that takes a variable and a term in T_{CCL} and returns a term in T_{CCL}, is only evaluated in the definition of $\langle\ \rangle$ with a variable or an application as second argument.

Let $\to_\beta$ denote β-reduction and $\twoheadrightarrow_\beta$ its reflexive and transitive closure. For the interpretations defined above the following property holds:

Property 4.17 ([13]) *i)* $[\![\langle M\rangle]\!]_{\mathrm{CL}} \twoheadrightarrow_\beta M$.

ii) If $t \to_{\mathbf{r}} u$ *in CCL, then* $[\![t]\!]_{\mathrm{CL}} \twoheadrightarrow_\beta [\![u]\!]_{\mathrm{CL}}$. ■

For example,

$$\langle \lambda xy.x\rangle = \lambda^* x.\langle \lambda y.x\rangle = \lambda^* x.\lambda^* y.x = \lambda^* x.Ap\,(K_0,x) =$$
$$Ap\,(Ap\,(S_0,\lambda^* x.K_0),\lambda^* x.x) = Ap\,(Ap\,(S_0,Ap\,(K_0,K_0)),I_0),$$

and

$$[\![Ap\,(Ap\,(S_0,Ap\,(K_0,K_0)),I_0)]\!]_{\mathrm{CL}} = (\lambda xyz.xz(yz))((\lambda xy.x)\lambda xy.x)\lambda x.x \twoheadrightarrow_\beta \lambda xy.x.$$

Notice also that

$$Ap(Ap(S_0, Ap(K_0, K_0)), I_0) \to_{\mathbf{r}} Ap(S_1(Ap(K_0, K_0)), I_0) \to_{\mathbf{r}} S_2(Ap(K_0, K_0), I_0) \to_{\mathbf{r}} S_2(K_1(K_0), I_0),$$

and

$$[\![S_2(K_1(K_0), I_0)]\!]_{\mathrm{CL}} = [\![Ap(Ap(S_0, Ap(K_0, K_0)), I_0)]\!]_{\mathrm{CL}}.$$

The relation between essential type assignment in LC and that in $\mathcal{G}$TRS (restricted to CCL with the environment $\mathcal{E}_{\mathrm{CL}}$ of Example 4.11) is very strong, as the following theorem shows.

Theorem 4.18 *i) $B \vdash M{:}\sigma$ implies $B \vdash_{\mathcal{E}_{\mathrm{CL}}} \langle M\rangle{:}\sigma$.*
ii) $B \vdash_{\mathcal{E}_{\mathrm{CL}}} t{:}\sigma$ implies $B \vdash [\![t]\!]_{\mathrm{CL}}{:}\sigma$.

Proof: By induction on the definition of $\langle\ \rangle$ and $[\![\ ]\!]_{\mathrm{CL}}$. ■

As a corollary, we obtain the undecidability of type assignment in our system (or, more precisely, the undecidability of type assignment in CCL with respect to $\mathcal{E}_{\mathrm{CL}}$).

4.4 Subject Reduction

In Subsection 4.2 we defined type assignment on rules in such a way that a rewrite rule is typeable only if it can be typed using the principal pair of the left-hand side. We will show that this condition is sufficient for subject reduction. First we need to prove that the three operations on pairs (substitution, expansion, and lifting), defined in Subsection 4.1, are sound on typeable terms. We will also show that the operations of substitution and expansion are sound on rewrite rules. It is not possible to prove such a property for the operation of lifting.

The following theorem shows the soundness of substitution.

Theorem 4.19 *Let S be a substitution and $\mathcal{E}$ an environment.*

i) If $B \vdash_{\mathcal{E}} t{:}\sigma$, then $S(B) \vdash_{\mathcal{E}} t{:}S(\sigma)$.

ii) Let $\mathbf{r}$: $l \to r$ be a rewrite rule typeable with respect to the environment $\mathcal{E}$, and let F be the defined symbol of $\mathbf{r}$. Then $\mathbf{r}$ is typeable with respect to $\mathcal{E}[F{:}S(\mathcal{E}(F))]$.

Proof: *i*) By induction on the structure of derivations.

$((\leq))$: Then $t \equiv x$ and $B \leq \{x{:}\sigma\}$. By Lemma 4.2(*ii*), $S(B) \leq S(\{x{:}\sigma\}) = \{x{:}S(\sigma)\}$, so $S(B) \vdash_{\mathcal{E}} x{:}S(\sigma)$.

$((\to E))$: Then $t \equiv F(t_1, \ldots, t_n)$, there are $\sigma_1, \ldots, \sigma_n \in \mathcal{T}_{\mathrm{S}}$ and a chain C such that, for every $1 \leq i \leq n$, $B \vdash_{\mathcal{E}} t_i{:}\sigma_i$ and $C(\mathcal{E}(F)) = \sigma_1 \to \cdots \to \sigma_n \to \sigma$. Then by induction, for every $1 \leq i \leq n$, $S(B) \vdash_{\mathcal{E}} t_i{:}S(\sigma_i)$; since $S \circ C$ is a chain and

$$S \circ C(\mathcal{E}(F)) = S(\sigma_1) \to \cdots \to S(\sigma_n) \to S(\sigma),$$

by ($\to$E) also $S(B) \vdash_{\mathcal{E}} t{:}S(\sigma)$.

$((\cap I))$: Then $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ and $B \vdash_{\mathcal{E}} t{:}\sigma_i$, for all $1 \leq i \leq n$. By induction, $S(B) \vdash_{\mathcal{E}} t{:}S(\sigma_i)$, for every $1 \leq i \leq n$. Then, by ($\cap$I), $S(B) \vdash_{\mathcal{E}} t{:}S(\sigma_1) \cap \cdots \cap S(\sigma_n)$, so also $S(B) \vdash_{\mathcal{E}} t{:}S(\sigma)$.

ii) If $\mathbf{r}$ is a rewrite rule typeable with respect to $\mathcal{E}$, then by Definition 4.14(*i*) there is a basis P, and $\pi \in \mathcal{T}_{\mathrm{s}}$, such that

a) $\langle P, \pi\rangle$ is a principal pair for l with respect to $\mathcal{E}$,

b) $P \vdash_{\mathcal{E}} r{:}\pi$,

c) In $P \vdash_{\mathcal{E}} l{:}\pi$ and $P \vdash_{\mathcal{E}} r{:}\pi$, all occurrences of F are typed with $\mathcal{E}(F)$.

Let $\mathcal{E}' = \mathcal{E}[F{:}S(\mathcal{E}(F))]$. From $P \vdash_{\mathcal{E}} l{:}\pi$ and part (*i*) we obtain $S(P) \vdash_{\mathcal{E}} l{:}S(\pi)$. Since $\mathcal{E}$ and $\mathcal{E}'$ only differ in the type for F, it is easy to see that $S(P) \vdash_{\mathcal{E}'} l{:}S(\pi)$. Likewise, we

can conclude that $S(P) \vdash_{\mathcal{E}'} r{:}S(\pi)$. Notice that in these type assignments, all Fs are typed with $\mathcal{E}'(F)$.

We will now prove that $\langle S(P), S(\pi)\rangle$ is the principal pair for l with respect to $\mathcal{E}'$. Suppose $B \vdash_{\mathcal{E}'} l{:}\sigma$, then also $B \vdash_{\mathcal{E}} l{:}\sigma$. Since $\langle P,\pi\rangle$ is principal for l with respect to $\mathcal{E}$, there is a chain C such that $C(\langle P,\pi\rangle) = \langle B,\sigma\rangle$. Suppose F occurs n times in l; from $B \vdash_{\mathcal{E}'} l{:}\sigma$, by Definition 4.10 for the j-th occurrence ($1 \leq j \leq n$) there is a chain C_j such that the type used for that F is $C_j(\mathcal{E}(F))$. Assume, without loss of generality, that these chains C_j and C do not interfere. Take $C' = C \circ C_1 \circ \cdots \circ C_n$, then $C'(\langle S(P), S(\pi)\rangle) = \langle B,\sigma\rangle$. ■

The following theorem shows the soundness of expansion, for which we need the next lemma.

Lemma 4.20 Let $B' \vdash_{\mathcal{E}} t{:}\tau$, where $\tau \in \mathcal{T}_s$, $E = \langle \mu, n, B, \sigma\rangle$ be an expansion such that $\mathcal{T}_{<B',\tau>} \subseteq \mathcal{T}_{\langle B,\sigma\rangle}$, and $\rho \in \mathcal{T}_{\langle B,\sigma\rangle}$. Then

i) a) For $1 \leq i \leq n$, there are ρ_i and a substitution S_i such that $S_i(\rho) = \rho_i$ and $E(\rho) = \rho_1 \cap \cdots \cap \rho_n$, or,

b) $E(\rho) \in \mathcal{T}_s$.

ii) a) For $1 \leq i \leq n$, there are B_i, τ_i, and substitution S_i such that $E(\langle B',\tau\rangle) = \langle \Pi\{B_1,\ldots,B_n\}, \tau_1 \cap \cdots \cap \tau_n\rangle$, and $S_i(\langle B',\tau\rangle) = \langle B_i, \tau_i\rangle$, for every $1 \leq i \leq n$, or,

b) $E(\langle B',\tau\rangle) = \langle B'', \tau'\rangle$, with $\tau' \in \mathcal{T}_s$.

Proof: By Definition 4.4 and Lemma 4.6 (*i*). ■

Theorem 4.21 *Let E be an expansion such that $E(\langle B,\sigma\rangle) = \langle B',\sigma'\rangle$.*

i) If $B \vdash_{\mathcal{E}} t{:}\sigma$, then $B' \vdash_{\mathcal{E}} t{:}\sigma'$.

ii) Let **r**: *$l \to r$ be a rewrite rule typeable with respect to the environment $\mathcal{E}$ and let F be the defined symbol of* **r**. *If $E(\mathcal{E}(F)) = \sigma_1 \cap \cdots \cap \sigma_n$, then for every $1 \leq i \leq n$,* **r** *is typeable with respect to $\mathcal{E}[F{:}\sigma_i]$.*

Proof: *i*) By induction on $\mathcal{T}_S$, of which we will only show the part $\sigma \in \mathcal{T}_s$. Let $E(\langle B,\sigma\rangle) = \langle B',\sigma'\rangle$. Then by Lemma 4.20 either:

a) $\sigma' = \tau_1 \cap \cdots \cap \tau_m$, for every $1 \leq j \leq m$ there are a substitution S_j and basis B_j such that $S_j(\sigma) = \tau_j$, $S_j(B) = B_j$, and $B' = \Pi\{B_1,\ldots,B_m\}$. Then by Theorem 4.19(*i*), for every $1 \leq j \leq m$, $B_j \vdash_{\mathcal{E}} t{:}\tau_j$, so, by ($\cap$I) and 4.2(*ii*), $B' \vdash_{\mathcal{E}} t{:}\sigma'$.

or,

b) $\sigma' \in \mathcal{T}_s$. This part is proved by induction on the structure of derivations.

$((\leq))$: Then $t \equiv x$ and $B \leq \{x{:}\sigma\}$. By Lemma 4.6(*iv*), $B' \leq \{x{:}\sigma'\}$, so $B' \vdash_{\mathcal{E}} x{:}\sigma'$.

$((\to E))$: Then $t \equiv F(t_1,\ldots,t_n)$, and there are σ_i, $\sigma'_i \in \mathcal{T}_S$, for $1 \leq i \leq n$, and a chain C such that $C(\mathcal{E}(F)) = \sigma_1 \to \cdots \to \sigma_n \to \sigma$ and, for every $1 \leq i \leq n$, $B \vdash_{\mathcal{E}} t_i{:}\sigma_i$ and $E(\sigma_i) = \sigma'_i$. By induction, for every $1 \leq i \leq n$, $B' \vdash_{\mathcal{E}} t_i{:}\sigma'_i$; since $E \circ C$ is a chain and $E \circ C(\mathcal{E}(F)) = \sigma'_1 \to \cdots \to \sigma'_n \to \sigma$, also $B' \vdash_{\mathcal{E}} t{:}\sigma$.

ii) Since $\mathcal{E}(F) \in \mathcal{T}_s$, by Lemma 4.6(*i*) either:

a) $\tau = \tau_1 \cap \cdots \cap \tau_n$. Then, for every $1 \leq i \leq n$, there is a substitution S such that $S(\mathcal{E}(F)) = \tau_i$. The proof is completed by Theorem 4.19(*ii*).

or,

b) $\tau \in \mathcal{T}_s$. The proof for this part is very similar to part (*ii*) of the proof of Theorem 4.19. ■

The following theorem shows the soundness of lifting.

Theorem 4.22 *If $B \vdash_{\mathcal{E}} t{:}\sigma$ and L is a lifting, then also $L(B) \vdash_{\mathcal{E}} t{:}L(\sigma)$.*

Proof: By Lemma 4.12(*i*) and (*ii*). ■

Since a lifting can introduce non-relevant types into bases, obviously not every lifting performed on a pair $\langle B,\sigma\rangle$ such that $\langle B,\sigma\rangle$ is a principal pair for t produces a pair with this same property. Since type assignment of rewrite rules is defined using the notion of principal pairs, it is clear that lifting cannot be a sound operation on rewrite rules. This can also be illustrated by the following:

Take the rewrite system

$$\begin{array}{ll} I(x) & \to x \\ F(I_0) & \to I_0 \end{array}$$

that is typeable with respect to the environment $\mathcal{E}_1(I) = 1{\to}1$, $\mathcal{E}_1(F) = (2{\to}2){\to}3{\to}3$. Since $(2{\to}2){\to}3{\to}3 \leq (2{\to}2){\cap}4{\to}3{\to}3$, $<\langle\emptyset,(2{\to}2){\to}3{\to}3\rangle, \langle\emptyset,(2{\to}2){\cap}4{\to}3{\to}3\rangle>$ is a lifting. It is not possible to show that the rewrite rule that defines F is typeable with respect to $\mathcal{E}[F{:}(2{\to}2){\cap}4{\to}3{\to}3]$, since all types in $(2{\to}2){\cap}4$ should be types for I.

Combining the above results for the different operations, we have:

Theorem 4.23 *i) If $B \vdash_{\mathcal{E}} t{:}\sigma$ then for every chain C such that $C(\langle B,\sigma\rangle) = \langle B',\sigma'\rangle$, $B' \vdash_{\mathcal{E}} t{:}\sigma'$.*

ii) Let $\mathbf{r}$: $l \to r$ be a rewrite rule typeable with respect to the environment $\mathcal{E}$ and let F be the defined symbol of $\mathbf{r}$. If C is a chain that contains no lifting, then: if $C(\mathcal{E}(F)) = \sigma_1{\cap}\cdots{\cap}\sigma_n$, then for every $1 \leq i \leq n$, $\mathbf{r}$ is typeable with respect to $\mathcal{E}[F{:}\sigma_i]$.

Proof: By Theorems 4.19, 4.21, and 4.22. ■

In the proof of Subject Reduction we will use one more lemma:

Lemma 4.24 Let $\mathcal{E}$ be an environment, $t \in T(\mathcal{F},\mathcal{X})$, and R *a term-substitution.*

i) If $B \vdash_{\mathcal{E}} t{:}\sigma$ and B' is a basis such that $B' \vdash_{\mathcal{E}} x^{\mathrm{R}}{:}\rho$ for every statement $x{:}\rho \in B$, then $B' \vdash_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma$.

ii) If there are B and σ such that $B \vdash_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma$, then for every x occurring in t there is a type ρ_x such that

$$\{x{:}\rho_x \mid x \text{ occurs in } t\} \vdash_{\mathcal{E}} t{:}\sigma, \text{ and } B \vdash_{\mathcal{E}} x^{\mathrm{R}}{:}\rho_x.$$

Proof: By induction on the structure of derivations. ■

Theorem 4.25 (Subject Reduction Theorem) *Let $(\Sigma,\mathbf{R})$ be a typeable $\mathcal{G}$TRS with respect to an environment $\mathcal{E}$. If $B \vdash_{\mathcal{E}} t{:}\sigma$ and $t \to_{\mathbf{R}} t'$, then $B \vdash_{\mathcal{E}} t'{:}\sigma$.*

Proof: Let $\mathbf{r} : l \to r$ be the typeable rewrite rule applied in the rewrite step $t \to_{\mathbf{R}} t'$. We will prove that for every term-substitution R and type μ, if $B \vdash_{\mathcal{E}} l^{\mathrm{R}}{:}\mu$, then $B \vdash_{\mathcal{E}} r^{\mathrm{R}}{:}\mu$, which proves the theorem.

Since $\mathbf{r}$ is typeable, there are P, π such that $\langle P,\pi\rangle$ is a principal pair for l with respect to $\mathcal{E}$, and $P \vdash_{\mathcal{E}} r{:}\pi$. Suppose R is a term-substitution such that $B \vdash_{\mathcal{E}} l^{\mathrm{R}}{:}\mu$. By Lemma 4.24(*ii*) there is a B' such that for every $x{:}\rho \in B'$, $B \vdash_{\mathcal{E}} x^{\mathrm{R}}{:}\rho$, and $B' \vdash_{\mathcal{E}} l{:}\mu$. Since $\langle P,\pi\rangle$ is a principal pair for l with respect to $\mathcal{E}$, by Definition 4.13 there is a chain C such that $C(\langle P,\pi\rangle) = \langle B',\mu\rangle$. Since $P \vdash_{\mathcal{E}} r{:}\pi$, by Theorem 4.23(*i*) also $B' \vdash_{\mathcal{E}} r{:}\mu$. Then by Lemma 4.24(*i*) $B \vdash_{\mathcal{E}} r^{\mathrm{R}}{:}\mu$. ■

It is important to note that the condition '$\langle P,\pi\rangle$ is a principal pair for l with respect to $\mathcal{E}$' in Definition 4.14 is crucial. Just saying naively:

$l \to r \in \mathbf{R}$ is typeable with respect to $\mathcal{E}$ if there are basis B and type $\sigma \in \mathcal{T}_s$ such that: $B \vdash_{\mathcal{E}} l{:}\sigma$ and $B \vdash_{\mathcal{E}} r{:}\sigma$,

would give a notion of type assignment that is not closed under rewriting and is not a natural extension of the essential intersection system for LC. The following is an example of the loss of subject reduction (see [10] for more details).

Example 4.26 Consider the rewrite system

$$\begin{array}{lcl} H(S_2(x,y)) & \to & S_2(I_0,y) \\ S(x,y,z) & \to & Ap(Ap(x,z),Ap(y,z)) \\ K(x,y) & \to & x \\ I(x) & \to & x \end{array}$$

and the environment

$$\begin{array}{l} \mathcal{E}_0(H) = ((1\to2)\to3)\to(1\to2)\to2, \\ \mathcal{E}_0(S) = (1\to2\to3)\to(1\to2)\to1\to3, \\ \mathcal{E}_0(K) = 1\to2\to1, \\ \mathcal{E}_0(I) = 1\to1. \end{array}$$

The first rule is naively typeable with respect to $\mathcal{E}_0$:

$$\dfrac{\dfrac{x{:}(1\to2)\to1\to3 \qquad y{:}(1\to2)\to1}{S_2(x,y){:}(1\to2)\to3}}{H(S_2(x,y)){:}(1\to2)\to2} \qquad \to \qquad \dfrac{\dfrac{}{I_0{:}(1\to2)\to1\to2} \qquad y{:}(1\to2)\to1}{S_2(I_0,y){:}(1\to2)\to2}$$

Take the term $H(S_2(K_0,I_0))$, which reduces to $S_2(I_0,I_0)$. Although the first term is typeable with respect to $\mathcal{E}_0$:

$$\dfrac{\dfrac{\dfrac{}{K_0{:}(4\to5)\to(4\to5)\to4\to5} \qquad \dfrac{}{I_0{:}(4\to5)\to4\to5}}{S_2(K_0,I_0){:}(4\to5)\to4\to5}}{H(S_2(K_0,I_0)){:}(4\to5)\to5}$$

the term $S_2(I_0,I_0)$ is not typeable with respect to $\mathcal{E}_0$ with the type $(4\to5)\to5$. In our system the rule is not typeable in this way, because the type assignment used for $H(S_2(x,y))$ is not a principal one. To illustrate this, consider the following derivation:

$$\dfrac{\dfrac{x{:}(1\to2)\to4\to3 \qquad y{:}(1\to2)\to4}{S_2(x,y){:}(1\to2)\to3}}{H(S_2(x,y)){:}(1\to2)\to2}$$

The pair $\langle\{x{:}(1\to2)\to4\to3, y{:}(1\to2)\to4\},(1\to2)\to2\rangle$ cannot be obtained from $\langle\{x{:}(1\to2)\to1\to3, y{:}(1\to2)\to1\},(1\to2)\to2\rangle$ by a chain of operations; in the opposite direction, the operation needed is that of $(4\mapsto1)$.

We should emphasize that, when defining type assignment in a naive way, the loss of the subject reduction property is not caused by the fact that intersection types are used. The environment $\mathcal{E}_0$ maps function symbols to Curry-types, so even for a notion of type assignment based on Curry-types, types are not preserved under rewriting.

To illustrate the fact that, when assigning types in a naive way, also the relation with the

essential intersection type assignment system for LC is lost, we give another example.

Example 4.27 Take the rewrite rule

$$E(x,y) \to Ap(x,y).$$

Let $\mathcal{E}(E) = 3{\to}1{\to}4$. Take $B = \{x{:}3{\cap}(1{\to}4), y{:}1\}$, then we can derive $B \vdash_{\mathcal{E}} E(x,y){:}4$ and $B \vdash_{\mathcal{E}} Ap(x,y){:}4$. This rewrite rule for E corresponds to the λ-term $\lambda xy.xy$, but $3{\to}1{\to}4$ is not a correct type for this term in the type assignment system of Section 3.

Therefore, a minimal requirement for subject reduction is to demand that all types used for variables in the derivation for the right-hand side of a rewrite rule are those actually needed in the derivation for the left-hand side. This is accomplished by restricting the possible bases to those that are principal for the left-hand side.

5 Normalization Properties of Typeable $\mathcal{G}$TRS

In this section we study normalization properties of $\mathcal{G}$TRS, using the type assignment system defined above. As in LC, the type ω plays an important role in this study.

As mentioned in the introduction, in the rewriting framework typeability alone does not ensure any normalization property (for example, consider a typeable term t and a one-rule recursive $\mathcal{G}$TRS of the form $t \to t$). This means that the characterization of normalizability of terms in $\mathcal{G}$TRS cannot be based on type conditions only, as is possible for LC, but that also syntactic restrictions on the rules have to be imposed. For this reason, we will introduce a *general scheme of recursion*, inspired by [27], that restricts the use of recursion to ensure strong normalization of all terms typeable without using ω. Moreover, by restricting the scheme a little further, we will show that when ω is used, all typeable terms have a head-normal form. More precisely, we will prove the following three results:

Theorem Let $(\Sigma, \mathbf{R})$ be a $\mathcal{G}$TRS that satisfies certain conditions to be formulated below, and let $t \in T(\mathcal{F}, \mathcal{X})$. Then:

- If $B \vdash_{\mathcal{E}} t{:}\sigma$ and $\sigma \neq \omega$, then t has a head-normal form.
- If $B \vdash_{\mathcal{E}} t{:}\sigma$, t is a non-Curryfied term, and ω does not occur in B and σ, then t has a normal form.
- If $B \vdash_{\mathcal{E}} t{:}\sigma$ and ω is not used at all, then t is strongly normalizable.

To prove this theorem we will use the method of Computability Predicates of Tait'67, adapted to the rewriting framework. In contrast with LC, the structure of the rewrite rules in $\mathcal{G}$TRS is not fixed, and hence the general scheme plays a crucial role in the proof.

As a consequence of the previous theorem, we can deduce that in the intersection system without ω the class of typeable *non-recursive* $\mathcal{G}$TRS is strongly normalizing. To appreciate the non-triviality of this condition, remember Example 2.8: a non-recursive $\mathcal{G}$TRS may be non-terminating, or worse, not even head-normalizing. In fact, the main result of Subsection 5.2 (every typeable term is head-normalizable) shows that the only type that can be assigned to the term $D(D_0)$ is ω.

The converse of these results does not hold, due to the fact that arbitrary patterns are allowed, as shown below:

Example 5.1 Take the strongly normalizing rewrite system

$$\begin{array}{ll} I(x) & \to x \\ K(x,y) & \to x \\ F(I_0) & \to I_0 \\ F(K_0) & \to K_0. \end{array}$$

It is not possible to give an environment such that these rules can be typed, since there is no type σ that is a type for both I and K.

5.1 Strong Normalization

In the following we will define the general scheme and the class of safe recursive systems, and prove that, using the type assignment system without ω, typeable safe systems are strongly normalizing.

Definition 5.1 Let Σ be a signature with a set of function symbols $\mathcal{F}_n = \mathcal{C} \cup \{F^1, \ldots, F^n\}$, where $F^1, \ldots, F^n$ will be the defined symbols that are not Curryfied-versions, and $\mathcal{C}$ the set of constructors and Curryfied versions of symbols. Assume that $F^1, \ldots, F^n$ are defined incrementally, by rules that satisfy the *general scheme*:

$$F^i(\bar{\mathrm{C}}[\bar{x}],\bar{y}) \to \mathrm{C}'[F^i(\overline{\mathrm{C}_1}[\bar{x}],\bar{y}),\ldots,F^i(\overline{\mathrm{C}_m}[\bar{x}],\bar{y}),\bar{y}],$$

where $\bar{x}$, $\bar{y}$are sequences of variables such that $\bar{x} \subseteq \bar{y}$; $\bar{\mathrm{C}}[\,]$, $\mathrm{C}'[\,]$, $\overline{\mathrm{C}_1}[\,]$, $\ldots$, $\overline{\mathrm{C}_m}[\,]$ are sequences of contexts in $T(\mathcal{F}_{i-1},\mathcal{X})$; and for every $1 \le j \le m$, $\bar{\mathrm{C}}[\bar{x}] \rhd_{mul} \overline{\mathrm{C}_j}[\bar{x}]$, where $\lhd$ is the strict subterm ordering (i.e. $\rhd$ denotes strict superterm) and *mul* denotes multiset extension.

Then the hierarchical $\mathcal{U}$TRS that contains the rules defining $F^1, \ldots, F^n$ is a *safe recursive system*.

This general scheme is a generalization of primitive recursion. It imposes two main restrictions on the definition of functions: the terms in the multisets $\overline{\mathrm{C}_j}[\bar{x}]$ are subterms of terms in $\bar{\mathrm{C}}$(this is the 'primitive recursive' aspect of the scheme), and the variables $\bar{x}$must also appear as arguments in the left-hand side of the rule. Both restrictions are essential to prove the Strong Normalization Theorem (Theorem 5.12 below). The last one can be replaced by a typing condition, requiring that the variables in $\bar{x}$ that are not included in $\bar{y}$ can only be assigned base types. Also, instead of the multiset extension of the subterm ordering, a lexicographic extension can be used, or even a combination of lexicographic and multiset (see [23] for details about these variants of the scheme).

Example 5.2 The following rewrite system on natural numbers and lists of natural numbers is safe: it is a hierarchical system, the variables that do not appear as arguments in the left-hand sides can only have base types, and the recursive calls in the right-hand sides satisfy the required subterm condition. The signature contains the constructors *Succ*, *Zero*, *Nil*, and *Cons*, and the defined symbols *Add*, *Mul*, *Con*, *Len*, and *Rev*.

$$\begin{array}{lll} Add(Zero,y) & \to & y \\ Add(Succ(x),y) & \to & Succ(Add(x,y)) \\ Mul(Zero,y) & \to & Zero \\ Mul(Succ(x),y) & \to & Add(Mul(x,y),y) \\ Con(Nil,l) & \to & l \\ Con(Cons(a,b),l) & \to & Cons(a,Con(b,l)) \end{array}$$

$$
\begin{array}{ll}
Len\,(Nil) & \rightarrow\ Zero \\
Len\,(Cons\,(x,y)) & \rightarrow\ Succ\,(Len\,(y)) \\
Len\,(Con\,(x,y)) & \rightarrow\ Add\,(Len\,(x),Len\,(y)) \\
Rev\,(Nil) & \rightarrow\ Nil \\
Rev\,(Cons\,(a,b)) & \rightarrow\ Con\,(Rev\,(b),Cons\,(a,Nil))
\end{array}
$$

If we extend the definition of *Add* with the rule that expresses associativity,

$$Add\,(Add\,(x,y),z) \ \rightarrow Add\,(x,Add\,(y,z))$$

the rewrite system is no longer safe.

Note that although the general scheme has a primitive recursive aspect, it allows the definition of non-primitive functions thanks to the higher-order features available in $\mathcal{G}$TRS: for example, Ackermann's function can be represented. Moreover, the rewrite rules of CCL (Example 2.5) are *not* recursive, so, in particular, satisfy the scheme. Therefore, even with the severe restrictions imposed on rewrite rules by the general scheme, the class of safe $\mathcal{G}$TRS is Turing-complete, a property that systems without Ap would not possess.

5.1.1 Type Assignment without ω

We will consider environments that map function symbols to types without ω. Such environments will be called *ω-free*, and in general we will use the expression *ω-free* as prefix to indicate that ω does not appear in an object.

The sets $\mathcal{T}_s^{\omega}$, $\mathcal{T}_S^{\omega}$ of *ω-free strict types* and *ω-free strict intersection types*, that will be used in this subsection, are subsets of the ones used in Definition 3.1: we will not consider types containing the type constant ω. These types are, in fact, the types used by Coppo-Dezani'80.

Notice that $\mathcal{T}_s^{\omega}$ is a proper subset of $\mathcal{T}_S^{\omega}$, and that now in $\sigma_1 \cap \cdots \cap \sigma_n$, n cannot be zero.

The ω-free type assignment system is defined as above, i.e. using three operations on pairs of $\langle basis, type \rangle$, namely ω-free substitution, ω-free expansion, and ω-free lifting, that are ω-free variants of similar definitions given in Subsection 4.1. *ω-free chains* are obtained by composing operations of ω-free substitution, ω-free expansion, or ω-free lifting.

Type assignment on terms is defined in a way similar to that of Subsection 4.2, using a natural deduction system with ($\leq$), ($\cap$I), and ($\rightarrow$E) rules. Apart from the fact that in those rules now only types in $\mathcal{T}_S^{\omega}$ are considered, the only difference is that in rule ($\cap$I), $n \geq 1$.

Definition 5.3 *ω-free type assignment* and *ω-free derivations* are defined by the following natural deduction system:

$$(\leq):\ \frac{x{:}\sigma \qquad \sigma \leq \tau}{x{:}\tau} \qquad (\cap\mathrm{I}):\ \frac{t{:}\sigma_1 \quad \ldots \quad t{:}\sigma_n}{t{:}\sigma_1 \cap \cdots \cap \sigma_n}\ (n \geq 1)$$

$$(\rightarrow\mathrm{E}):\ \frac{t_1{:}\sigma_1 \quad \ldots \quad t_n{:}\sigma_n}{F(t_1,\ldots,t_n){:}\sigma}\ (a)$$

(a): If there exists $C \in \mathcal{C}h$ such that $C\,(\mathcal{E}(F)) = \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow \sigma$.

We will use the symbol $\vdash_{\mathcal{E}}^{\omega}$ for this type assignment system.

Example 5.4 As for the system $\vdash_{\mathcal{E}}$, also for $\vdash_{\mathcal{E}}^{\omega}$ the term $S\,(K_0, S_0, I_0)$ can be typed with the

type $\sigma\to\sigma$, under the assumption that:

$$\begin{aligned}
\mathcal{E}\,(S) &= (1\to2\to3)\to(4\to2)\to1\cap4\to3,\\
\mathcal{E}\,(K) &= 5\to6\to5,\\
\mathcal{E}\,(I) &= 7\to7.
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{K_0{:}(\sigma\to\sigma)\to((\tau\to\rho)\to(\rho\to\mu)\cap\tau\to\mu)\to\sigma\to\sigma}}{\vdots} \quad \dfrac{\dfrac{}{S_0{:}((\rho\to\mu)\to\rho\to\mu)\to(\tau\to\rho)\to(\rho\to\mu)\cap\tau\to\mu}}{\vdots} \quad \dfrac{}{I_0{:}(\sigma\to\sigma)\cap((\rho\to\mu)\to\rho\to\mu)}}{S\,(K_0, S_0, I_0){:}\sigma\to\sigma}$$

Notice that, to obtain the type

$$((\sigma\to\sigma)\to((\tau\to\rho)\to(\rho\to\mu)\cap\tau\to\mu)\to\sigma\to\sigma) \to (((\rho\to\mu)\to\rho\to\mu)\to(\tau\to\rho)\to(\rho\to\mu)\cap\tau\to\mu) \to (\sigma\to\sigma)\cap((\rho\to\mu)\to\rho\to\mu) \to \sigma\to\sigma,$$

for S, we have used the chain

$$(1\mapsto\sigma\to\sigma)\circ(2\mapsto((\tau\to\rho)\to(\rho\to\mu)\cap\tau\to\mu))\circ(3\mapsto\sigma\to\sigma)\circ(4\mapsto(\rho\to\mu)\to\rho\to\mu).$$

The following properties, where bases and types are ω-free, are needed further on:

Lemma 5.5 *i) If $B\vdash^{\omega}_{\mathcal{E}} t{:}\sigma$ and $B'\leq B$, then $B'\vdash^{\omega}_{\mathcal{E}} t{:}\sigma$.*
ii) If $B\vdash^{\omega}_{\mathcal{E}} t{:}\sigma$ and $\sigma\leq\tau$, then $B\vdash^{\omega}_{\mathcal{E}} t{:}\tau$.
iii) If $B, x{:}\alpha\vdash^{\omega}_{\mathcal{E}} Ap\,(t,x){:}\beta$, $\beta\in\mathcal{T}_s$, and x does not occur in t, then $B\vdash^{\omega}_{\mathcal{E}} t{:}\alpha\to\beta$.
iv) $B\vdash^{\omega}_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n \iff \forall\, 1\leq i\leq n\ [B\vdash^{\omega}_{\mathcal{E}} t{:}\sigma_i]$.
v) $B\vdash^{\omega}_{\mathcal{E}} F_n\,(t_1,\ldots,t_n){:}\sigma\ \&\ \sigma\in\mathcal{T}_s \Rightarrow \exists\,\alpha\in\mathcal{T}_S,\ \beta\in\mathcal{T}_s\ [\sigma=\alpha\to\beta]$.

Proof: By induction on the length of the derivations. ■

This lemma is just an ω-free variant of Lemma 4.12; however, none of the results formulated there imply those mentioned here.

To ensure the subject reduction property, as in Subsection 4.4, type assignment on rewrite rules is defined using the notion of principal pair for a typeable term.

Let $\mathcal{E}$ be an ω-free environment. As in Subsection 4.4, it is possible to show that the operations on pairs are sound on typed terms and are sound on rewrite rules in the following sense: if there is an operation O – either a substitution or an expansion – such that $O\,(\mathcal{E}\,(F)) = \sigma_1\cap\cdots\cap\sigma_n$, then for every $1\leq i\leq n$, the rewrite rules that define F are typeable with respect to the changed ω-free environment $\mathcal{E}\,[F{:}\sigma_i]$.

Then, using the same technique as in Subsection 4.4, it is possible to show that subject reduction holds.

Theorem 5.6 *If $B\vdash^{\omega}_{\mathcal{E}} t{:}\sigma$ and $t\to_{\mathbf{R}} t'$, then $B\vdash^{\omega}_{\mathcal{E}} t'{:}\sigma$.* ■

5.1.2 The Strong Normalization Theorem

We will prove now that the $\mathcal{G}$TRS that are typeable in the ω-free system and satisfy the general scheme are strongly normalizing. The proof has two parts; in the first one we define a computability predicate *Comp* on bases, terms, and types, and prove some properties of *Comp*. The most important one states that if for a term t there are a basis B and type σ such that

$Comp\,(B,t,\sigma)$, then t is strongly normalizable. In the second part *Comp* is shown to hold for each typeable term.

From now on, we will abbreviate 't is strongly normalizable' by $SN\,(t)$ and we will assume that $\mathcal{E}$ is a given ω-free environment.

Definition 5.7 *i*) Let B be a basis, t a term, and σ a type, such that $B \vdash^{\omega}_{\mathcal{E}} t{:}\sigma$. We define the Computability Predicate $Comp\,(B,t,\sigma)$ recursively on σ by:

a) $Comp\,(B,t,\varphi) \iff SN\,(t)$.

b) $Comp\,(B,t,s) \iff SN\,(t)$.

c) $Comp\,(B,t,\sigma\to\tau) \iff \forall\, u \in T(\mathcal{F},\mathcal{X})$ [$Comp\,(B',u,\sigma) \Rightarrow Comp\,(\Pi\{B,B'\},Ap\,(t,u),\tau)$].

d) $Comp\,(B,t,\sigma_1\cap\cdots\cap\sigma_n)$ $(n \geq 1) \iff \forall\, 1 \leq i \leq n$ [$Comp\,(B,t,\sigma_i)$].

ii) We say that a term t is *computable in type* σ if there exists B such that $Comp\,(B,t,\sigma)$.

iii) We say that a term-substitution R is *computable in a basis* B if there is a basis B' such that for every $x{:}\sigma \in B$, $Comp\,(B',x^{\mathrm{R}},\sigma)$ holds.

Notice that because we use intersection types and because of Definition 3.2(*iv*), in part (*iii*) we need not consider the existence of different bases for each $x{:}\sigma \in B$.

Comp is closed under $\leq$:

Lemma 5.8 $Comp\,(B,t,\sigma)\ \&\ \sigma \leq \tau \Rightarrow Comp\,(B,t,\tau)$.

Proof: By induction on the definition of $\leq$. ■

Comp satisfies the standard properties of computability predicates (see [25]). C1 and C3 are actually proved by mutual induction:

Property 5.9 (*C1*): *If* $Comp\,(B,t,\sigma)$, *then* $SN\,(t)$.

(*C2*): *If* $Comp\,(B,t,\sigma)$ *and* $t \to t'$, *then* $Comp\,(B,t',\sigma)$.

(*C3*): *If* t *is neutral,* $B \vdash^{\omega}_{\mathcal{E}} t{:}\sigma$ *for some* B, σ, *and, for all* v *such that* $t \to_{\mathbf{R}} v$, $Comp\,(B,v,\sigma)$, *then also* $Comp\,(B,t,\sigma)$.

Proof: By simultaneous induction on the structure of types.

i) $\sigma = \varphi$, or $\sigma = s \in S$.

(*C1*): By 5.7 (*i.a*) and (*i.b*).

(*C2*): By 5.7 (*i.a*) and (*i.b*), $B \vdash^{\omega}_{\mathcal{E}} t{:}\sigma$, and $SN\,(t)$. Certainly $SN\,(t')$, and by Theorem 5.6 also $B \vdash^{\omega}_{\mathcal{E}} t'{:}\sigma$, so by 5.7 (*i.a*) and (*i.b*) we obtain $Comp\,(B,t',\sigma)$.

(*C3*): By 5.7 (*i.a*) and (*i.b*), $SN\,(v)$. Obviously, also $SN\,(t)$ and, again by 5.7 (*i.a*) and (*i.b*), $Comp\,(B,t,\sigma)$ holds.

ii) $\sigma = \alpha\to\beta$.

(*C1*): Let $u \equiv x$ (a new variable). By Property C3, since x is a neutral term in normal form, and trivially $\{x{:}\alpha\} \vdash^{\omega}_{\mathcal{E}} x{:}\alpha$, also $Comp\,(\{x{:}\alpha\},x,\alpha)$. Then $Comp\,(B\cup\{x{:}\alpha\},Ap\,(t,x),\beta)$ by 5.7 (*i.c*). Then, by induction, $SN\,(Ap\,(t,x))$, which implies $SN\,(t)$.

(*C2*): Take u such that $Comp\,(B',u,\alpha)$, then, by 5.7 (*i.c*), $Comp\,(\Pi\{B,B'\},Ap\,(t,u),\beta)$. Since $Ap\,(t,u) \to^{*} Ap\,(t',u)$, by induction we get $Comp\,(\Pi\{B,B'\},Ap\,(t',u),\beta)$. Then, by 5.7 (*i.c*), $Comp\,(B,t',\alpha\to\beta)$.

$(C3)$: We have to prove that

$$\forall\, u \in T(\mathcal{F},\mathcal{X})\ [\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\},Ap\,(t,u),\beta)\ ].$$

Since $Ap\,(t,u)$ is neutral of type β, by induction, it is sufficient to prove that $Comp\,(\Pi\{B,B'\},v',\beta)$ holds for all v' such that $Ap\,(t,u) \to_{\mathbf{R}} v'$. For this we apply induction on the length of the maximal derivation from u to its normal form (by Property C1 we know that $SN\,(u)$).

$(Base)$: If u is a normal form, then $Ap\,(t,u)$ reduces only inside t (because t is neutral), so $Ap\,(t,u) \to_{\mathbf{R}} Ap\,(v,u)$ and $Comp\,(B,v,\sigma)$ holds by assumption. So, by 5.7 (*i.c*), we have $Comp\,(\Pi\{B,B'\},Ap\,(v,u),\beta)$.

$(Induction\ step)$: Consider all possible one-step reductions from $Ap\,(t,u)$: For $Ap\,(t,u) \to_{\mathbf{R}} Ap\,(v,u)$ we proceed as before. For the case $Ap\,(t,u) \to_{\mathbf{R}} Ap\,(t,u')$, by induction the result $Comp\,(\Pi\{B,B'\},Ap\,(t,u'),\beta)$ follows. And these are all possible cases, because $Ap\,(t,u)$ cannot be a redex itself since t is neutral and the rewrite system is safe.

iii) $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$.

$(C1)$: By 5.7 (*i.d*), $Comp\,(B,t,\sigma)$ implies $Comp\,(B,t,\sigma_i)$, for $1 \le i \le n$, and by induction $SN\,(t)$.

$(C2)$: By 5.7 (*i.d*), $Comp\,(B,t,\sigma)$ implies $Comp\,(B,t,\sigma_i)$, for $1 \le i \le n$. By induction, for $1 \le i \le n$, $Comp\,(B,t',\sigma_i)$, so, by 5.7 (*i.d*), also $Comp\,(B,t',\sigma)$.

$(C3)$: By Lemma 5.5(*iv*), $B \vdash^{\omega}_{\mathcal{E}} t{:}\sigma_i$ for $1 \le i \le n$, and by Theorem 5.6, $B \vdash^{\omega}_{\mathcal{E}} v{:}\sigma_i$. Moreover, $Comp\,(B,v,\sigma)$ implies $Comp\,(B,v,\sigma_i)$, for every $1 \le i \le n$. Then, by induction, for $1 \le i \le n$, $Comp\,(B,t,\sigma_i)$ and by 5.7 (*i.d*), $Comp\,(B,t,\sigma)$. ■

Terms can, as usual, be seen as trees; the subterm of t at position p will be denoted by $t|_p$, and $t[u]_p$ will denote the result of replacing in t the subterm at position p by u.

In order to prove that every typeable term is computable we shall prove a stronger property, for which we will need the following ordering.

Definition 5.10 *i*) Let $>_{\mathbb{N}}$ denote the standard ordering on natural numbers, and *lex*, *mul* denote respectively the *lexicographic* and *multiset* extension of an ordering. Let $\rhd$ stand for the well-founded encompassment ordering, i.e. $u \rhd v$ if $u \neq v$ modulo renaming of variables, and $u|_p = v^{\mathrm{R}}$ for some position $p \in u$ and term-substitution R. Note that encompassment contains strict superterm ($\rhd$).

ii) We define the ordering $\gg^{\mathrm{SN}}$ on triples – consisting of a pair of natural numbers, a term, and a multiset of terms – as the object

$$((>_{\mathbb{N}},>_{\mathbb{N}})_{lex},\rhd\ ,(\to_{\mathbf{R}} \cup \rhd)_{mul})_{lex}.$$

iii) We will interpret the term u^{R} by the triple $\langle (i,j),u,\{\mathrm{R}\}\rangle = \mathcal{I}(u^{\mathrm{R}})$, where

a) i is the maximal super-index of the function symbols belonging to u,

b) j is the minimum of the differences $arity\,(F^i)$ - $arity\,(F^i_k)$ such that F^i_k occurs in u, and

c) $\{\mathrm{R}\}$ is the multiset $\{x^{\mathrm{R}} \mid x \in Var\,(u)\}$.

These triples are compared in the ordering $\gg^{\mathrm{SN}}$.

When R is computable, then by Property C1 every t in the image of R is strongly normalizable, so $\to_{\mathbf{R}}$ is well-founded on the image of R. Also, because the union of the relation $\rhd$ with a terminating rewrite relation is well-founded [21], the relation $(\to_{\mathbf{R}} \cup \rhd)_{mul}$ is well-founded on $\{\mathrm{R}\}$. Hence, when restricted to computable term-substitutions, $\gg^{\mathrm{SN}}$ is a well-founded ordering.

We will use $\gg^{\mathrm{SN}}_n$ when we want to indicate that the nth element of the triple has decreased

and the n-1 first ones have not increased.

We would like to stress that we do not just interpret terms, but pairs of terms and term-substitutions. This implies that although it can be that the terms t^{R} and $t'^{\mathrm{R}'}$ are equal, their interpretations need not be equal as well.

We now come to the main theorem of this section, in which we will show that for any typeable term and computable term-substitution R also the term t^{R} is computable. The strong normalization result then follows, using Property C1, for any typeable term t, taking for R the identity.

In the proof, the main idea is to write a term t^{R} like $t'^{\mathrm{R}'}$ (so they are equal as terms), where $t \rhd t'$, and R′ is a computable extension of R. This is accomplished by taking a computable subterm v of t, to put a new variable z in its place and to define $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto v\}$.

Property 5.11 Let t be such that $B \vdash^{\omega}_{\mathcal{E}} t{:}\sigma$, and R *be computable in B. Then there is a B' such that $Comp\,(B', t^{\mathrm{R}}, \sigma)$.*

Proof: By noetherian induction on $\gg^{\mathrm{SN}}$.

If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, then, by Definition 5.7 (*i.d*), we have to prove that for every $1 \leq i \leq n$, $Comp\,(B', t^{\mathrm{R}}, \sigma_i)$. So, without loss of generality, we can consider $\sigma \in \mathcal{T}^{\omega}_{\mathrm{s}}$.

Let $B = \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\}$, and $\mathrm{R} = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$. We distinguish the cases:

i) t is a neutral term. If t is a variable x, then, by derivation rule ($\leq$), there is a τ such that $x{:}\tau \in B$, and $\tau \leq \sigma$. Since x^{R} is computable of type τ, by Lemma 5.8 it is also computable of type σ. So, without loss of generality we assume that t is not a variable and then also t^{R} is neutral. If t^{R} is irreducible, then $Comp\,(B', t^{\mathrm{R}}, \sigma)$ holds by Property C3. Otherwise, let $t^{\mathrm{R}} \to_{\mathbf{R}} w$ at position p. We will prove either $Comp\,(B', t^{\mathrm{R}}, \sigma)$ itself, or prove $Comp\,(B', w, \sigma)$ and apply Property C3.

a) $p = qp'$, $t|_q = x_i \in \mathcal{X}$, so the rewriting takes place in a subterm of t^{R} that is introduced by the term-substitution Let z be a new variable.
Take $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto w|_q\}$ and note that $t^{\mathrm{R}}|_q \to_{\mathbf{R}} w|_q$ at position p'. Since $t^{\mathrm{R}}|_q \in \{\mathrm{R}\}$ and R is assumed to be computable, also $Comp\,(B, t^{\mathrm{R}}|_q, \sigma_i)$ holds. So $Comp\,(B, w|_q, \sigma_i)$ holds by Property C2, hence R′ is computable in $B \cup \{z_i{:}\sigma_i\}$.
Now, if the variable x_i ($= t|_q$) has exactly *one* occurrence in t, then $t = t[z]_q$ modulo renaming of variables, and otherwise $t \rhd t[z]_q$. In the first case (since R contains a term that is rewritten to get R′) we have $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}}_3 \mathcal{I}([z]_q{}^{\mathrm{R}'})$, and $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}}_2 \mathcal{I}(t[z]_q{}^{\mathrm{R}'})$ in the second case. Both cases yield, by induction, $Comp\,(B', t[z]_q{}^{\mathrm{R}'}, \sigma)$. Note that $t[z]_q{}^{\mathrm{R}'} \equiv w$.

b) Now assume that p is a non-variable position in t. We analyze separately the cases:

1) p is not the root position. Then $t \rhd t|_p$, hence $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}}_2 \mathcal{I}(t|_p{}^{\mathrm{R}})$, and note that $t|_p{}^{\mathrm{R}} = t^{\mathrm{R}}|_p$. Let τ be the type assigned to $t|_p$ in the derivation of $B \vdash^{\omega}_{\mathcal{E}} t{:}\sigma$, then $Comp\,(B, t^{\mathrm{R}}|_p, \tau)$ holds by induction.
Let z be a new variable, and $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto t^{\mathrm{R}}|_p\}$, then R′ is computable in $B \cup \{z{:}\tau\}$, and $B \cup \{z{:}\tau\} \vdash^{\omega}_{\mathcal{E}} t[z]_p{:}\sigma$. Now $t \rhd t[z]_p$, hence $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}}_2 \mathcal{I}(t[z]_p{}^{\mathrm{R}'})$, hence $Comp\,(B, t^{\mathrm{R}}, \sigma)$.

2) p is the root position. Then the possible cases for t are:

A) $t \equiv F(t_1, \ldots, t_n)$, where at least one of the t_i is not a variable, and F is either a defined symbol of arity n, or $F \equiv Ap$ and $n = 2$. Take $\mathrm{R}' = \{z_1 \mapsto t_1{}^{\mathrm{R}}, \ldots, z_n \mapsto t_n{}^{\mathrm{R}}\}$. Since $t \rhd t_i$, $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}}_2 \mathcal{I}(t_i{}^{\mathrm{R}})$. Then if $B \vdash^{\omega}_{\mathcal{E}} t_i{:}\sigma_i$, $Comp\,(B, t_i{}^{\mathrm{R}}, \sigma_i)$ holds by induction. Hence, R′ is computable in $B \cup \{z_1{:}\sigma_1, \ldots, z_n{:}\sigma_n\}$. But $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}}_2 \mathcal{I}(F(z_1, \ldots, z_n)^{\mathrm{R}'})$, and note that $F(z_1, \ldots, z_n)^{\mathrm{R}'} = t^{\mathrm{R}}$ and

$$B \cup \{z_1{:}\sigma_1, \ldots, z_n{:}\sigma_n\} \vdash^{\omega}_{\mathcal{E}} F(z_1, \ldots, z_n){:}\sigma.$$

Hence $Comp\,(B, t^{\mathrm{R}}, \sigma)$.

B) $t \equiv F^k(z_1,\ldots,z_n)$ where $z_1,\ldots,z_n$ are different variables. (If $z_i = z_j$ for some $i \neq j$, we can reason as in part (*i.a*).) Then t^{R} must be an instance of the left-hand side of a rule defining F^k:

$$t^{\mathrm{R}} = F^k(z_1,\ldots,z_n)^{\mathrm{R}} = F^k(\bar{\mathrm{C}}[\overline{M}],\overline{N}) \to_{\mathbf{R}}$$
$$\mathrm{C}'[F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}),\overline{N}] = w,$$

where $\bar{\mathrm{C}}[\overline{M}]$, $\overline{N}$are all terms in $\{\mathrm{R}\}$, so are computable by hypothesis.
Now we will deduce $Comp(B,w,\sigma)$ in three steps:

(*Step I*): Let R' be the term-substitution that maps the left-hand side of the rewrite rule into t^{R}, so $\overline{x^{\mathrm{R}'}} = \overline{M}$. Since $\overline{M} \subseteq \overline{N}$and all $\overline{N}$are computable, also R' is computable[2]. For every $1 \leq j \leq m$, F^k does not occur in $\overline{\mathrm{C}_j}$(by definition of the general scheme), hence $\mathcal{I}(F^k(z_1,\ldots,z_n)^{\mathrm{R}}) \gg_1^{\mathrm{SN}} \mathcal{I}(\mathrm{C}_j{}^{\mathrm{R}'})$, so also the term $\mathrm{C}_j{}^{\mathrm{R}'}$ is computable.

(*Step II*): Let, for $1 \leq j \leq m$, R_j be the computable term-substitution such that $t^{\mathrm{R}_j} = F^k(\overline{\mathrm{C}_j}[\overline{M}],\overline{N}) = F^k(\overline{\mathrm{C}_j}[\bar{x}],\bar{y})^{\mathrm{R}'}$. Since $\bar{\mathrm{C}} \rhd_{mul} \overline{\mathrm{C}_j}$, and $\rhd$ is closed under term-substitution also $\bar{\mathrm{C}}^{\mathrm{R}'} \rhd_{mul} \overline{\mathrm{C}_j}^{\mathrm{R}'}$, so $\mathcal{I}(F^k(z_1,\ldots,z_n)^{\mathrm{R}}) \gg_3^{\mathrm{SN}} \mathcal{I}(F^k(z_1,\ldots,z_n)^{\mathrm{R}_j})$, and therefore $F^k(z_1,\ldots,z_n)^{\mathrm{R}_j}$ is computable.

(*Step III*): Let v be the term obtained by replacing, in the right-hand side of the rule, the terms $F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}),\overline{N}$by fresh variables. Let R'' be the term-substitution such that

$$v^{\mathrm{R}''} = \mathrm{C}'[F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}),\overline{N}],$$

then $t^{\mathrm{R}} \to_{\mathbf{R}} v^{\mathrm{R}''}$. Notice that above we have shown that R'' is computable. When an F^j occurs in v, then by definition of the general scheme $j < k$ and therefore $\mathcal{I}(t^{\mathrm{R}}) \gg_1^{\mathrm{SN}} \mathcal{I}(v^{\mathrm{R}''})$, hence $v^{\mathrm{R}''}$ is computable. Since $w = v^{\mathrm{R}''}$, we get $Comp(B,w,\sigma)$.

C) $t = Ap(z_1,z_2)$ where $z_1,z_2 \in \mathcal{X}$. By assumption, $z_1{}^{\mathrm{R}}$ and $z_2{}^{\mathrm{R}}$ are computable and since t is well-typed, z_1 must have an arrow type. Then, by 5.7, $Ap(z_1{}^{\mathrm{R}},z_2{}^{\mathrm{R}})$ is computable. But $Ap(z_1{}^{\mathrm{R}},z_2{}^{\mathrm{R}})$ is the same as $Ap(z_1,z_2)^{\mathrm{R}}$.

ii) t is not neutral. Let $t \equiv F_n^i(t_1,\ldots,t_n)$, where F_n^i is some Curryfied version of the symbol F^i. Again we distinguish two cases:

a) Assume that at least one of the t_i is not a variable. Since $t \rhd t_i$ for $1 \leq i \leq n$, by induction there exist B', σ_i such that $Comp(B',t_i,\sigma_i)$. Therefore, also the term-substitution $\mathrm{R}' = \{z_1 \mapsto t_1,\ldots,z_n \mapsto t_n\}$ is computable. We have $\mathcal{I}(t^{\mathrm{R}}) \gg^{\mathrm{SN}} \mathcal{I}(t^{\mathrm{R}'})$ since $t \rhd F_n^i(z_1,\ldots,z_n)$, and computability of $t^{\mathrm{R}'}$ by induction. Note that $t^{\mathrm{R}'} = t^{\mathrm{R}}$.

b) All t_i are variables. Since $B \vdash_{\mathcal{E}}^{\omega} t{:}\sigma$, by Lemma 5.5(*v*) there exist $\alpha \in \mathcal{T}_{\mathrm{S}}^{\omega}$, $\beta \in \mathcal{T}_{\mathrm{s}}^{\omega}$ such that $\sigma = \alpha \to \beta$. We have to prove $Comp(B',t^{\mathrm{R}},\alpha\to\beta)$, that is, for all u such that $Comp(B'',u,\alpha)$, we have to prove $Comp(\Pi\{B',B''\},Ap(t^{\mathrm{R}},u),\beta)$. Since the term $Ap(t^{\mathrm{R}},u)$ is neutral, by Property 5.9, it is sufficient to prove $Comp(\Pi\{B',B''\},t',\beta)$ for all t' such that $Ap(t^{\mathrm{R}},u) \to_{\mathbf{R}} t'$. This will be proved by induction on the sum of the length of the rewrite sequences out of u and out of R. Note that since u and R are computable, by Property C1, $SN(u)$ and $SN(\mathrm{R})$.

(*Base*): If u and R are in normal form, the only reduction step out of $Ap(t^{\mathrm{R}},u)$ could be:

$$Ap(F_n^i(z_1,\ldots,z_n)^{\mathrm{R}},u) \to_{\mathbf{R}} t' \equiv F_{n+1}^i(z_1{}^{\mathrm{R}},\ldots,z_n{}^{\mathrm{R}},u);$$

then $\mathcal{I}(t^{\mathrm{R}}) \gg_1^{\mathrm{SN}} \mathcal{I}(F_{n+1}^i(z_1{}^{\mathrm{R}},\ldots,z_n{}^{\mathrm{R}},u))$, so t' is computable.

(*Induction step*): If the reduction step out of $Ap(t^{\mathrm{R}},u)$ takes place inside u or t^{R} (in the

[2] With the version of the scheme that does not require $\overline{M} \subseteq \overline{N}$but assumes that the terms in $\overline{M}$that do not appear in $\overline{N}$are assigned base types, we can deduce that R' is computable because $SN(\overline{M})$.

last case it must be inside R since the rewrite system is safe), then t' is computable by induction. If $Ap(t^{\mathrm{R}},u) \to_{\mathbf{R}} F^i_{n+1}(z_1{}^{\mathrm{R}},\ldots,z_n{}^{\mathrm{R}},u)$, so $F^i_n(z_1,\ldots,z_n)^{\mathrm{R}} \equiv t$, then we proceed as in the base case. ■

Theorem 5.12 (Strong Normalization Theorem) *If* $(\Sigma,\mathbf{R})$ *is typeable in* $\vdash^{\omega}_{\mathcal{E}}$ *and safe, then any typeable term is strongly normalizable with respect to* $\mathbf{R}$.

Proof: Let R be such that $x^{\mathrm{R}} = x$, then by Property C3, R is computable. The result then follows from Properties 5.11 and Property C1. ■

5.2 Head-normalization

As shown in the previous subsection, in a type assignment system without ω the conditions imposed by the general scheme are sufficient to ensure strong normalization of typeable terms. Unfortunately, the general scheme is not enough to ensure head-normalization of typeable terms in a type system with ω: take the rewrite system

$$\begin{array}{ll} F(G(x)) & \to F(x), \\ A(x,y) & \to Ap(y,Ap(Ap(x,x),y)) \end{array}$$

that is typeable with respect to the environment

$\mathcal{E}(F) = \omega\to\sigma$,
$\mathcal{E}(G) = \omega\to\sigma$,
$\mathcal{E}(A) = ((\alpha\to\mu\to\beta)\cap\alpha)\to((\beta\to\rho)\cap\mu)\to\rho$,

then $B \vdash_{\mathcal{E}} F(A(A_0,G_0)){:}\sigma$, but

$$F(A(A_0,G_0)) \to^*_{\mathbf{R}} F(G(A(A_0,G_0))) \to_{\mathbf{R}} F(A(A_0,G_0)).$$

The underlying problem is that, using ω, there are two kinds of typeable recursion in $\mathcal{U}$TRS: the one explicitly present in the syntax, as well as the one obtained by the so-called *fixed-point combinators*; for every H that has type $\omega\to\sigma$, the term $A(A_0,H_0)$ has type σ, and $A(A_0,H_0) \to^*_{\mathbf{R}} H(A(A_0,H_0))$. In fact, the term $A_1(A_0)$ corresponds to $\Theta = (\lambda xy.y(xxy))(\lambda xy.y(xxy))$, Turing's fixed-point combinator for LC.

So, to obtain a head-normalization theorem in a type system with ω, we will have to impose stronger conditions than just those imposed by the general scheme. In this subsection we will only consider environments that map constructors to types without type variables and ω. In other words, we will consider those $\mathcal{U}$TRS having an alphabet with a set $\mathcal{C}$ of constructors, such that, for every environment $\mathcal{E}$, if $H \in \mathcal{C}$ then $\mathcal{E}(H) = s_1\to\ldots\to s_n\to s$, where n is the arity of H and $s_1,\ldots,s_n,s$ are sorts, i.e. type constants. In the following, $\mathcal{E}$ will denote an environment satisfying this condition.

Definition 5.13 (HNF-scheme) A rewrite rule

$$F^i(\bar{\mathrm{C}}[\bar{x}],\bar{y}) \to \mathrm{C}'[F^i(\overline{\mathrm{C}_1}[\bar{x}],\bar{y}),\ldots,F^i(\overline{\mathrm{C}_m}[\bar{x}],\bar{y}),\bar{y}]$$

satisfies the *HNF-scheme* in the environment $\mathcal{E}$, if it satisfies the conditions of the general scheme, where we replace the condition of Definition 5.1:

'for $1 \le j \le m$, $\bar{\mathrm{C}}[\bar{x}] \rhd_{mul} \overline{\mathrm{C}_j}[\bar{x}]$'

by the condition:

'for $1 \leq j \leq m$, $\bar{\mathsf{C}}[\bar{x}] \rhd_{mul} \overline{\mathsf{C}_j}[\bar{x}]$, $\bar{\mathsf{C}}[\bar{x}]$, $\overline{\mathsf{C}_j}[\bar{x}] \in T(\mathcal{C},\mathcal{X})$, and the patterns (see Definition 2.4(*i*)) appear at positions where $\mathcal{E}(F^i)$ requires arguments of sort type'

The systems that satisfy the HNF-scheme will be called *HNF-safe*.

Actually, the condition $\bar{x} \subseteq \bar{y}$is not needed in this case, since the only types that can be assigned to the variables in $\bar{x}$are sorts; but in order to simplify the proofs we will keep it. Also in this case a lexicographic extension of the ordering $\rhd$ can be used.

The rest of this section will be devoted to the proof of the Head Normalization Theorem. We will prove, simultaneously, that every typeable term is head-normalizable and constructor-hat normalizable in a typeable and HNF-safe $\mathcal{U}$TRS. Again we will use the method of Computability Predicates. The proof has two parts; in the first we will define a new predicate *Comp* on bases, terms, and types, and prove some properties of *Comp*. The most important states that, if for a term t there are a basis B and type $\sigma \neq \omega$ such that $Comp\,(B,t,\sigma)$ holds, then $HN\,(t)$ and $CHN\,(t)$ (see notations at the end of Section 2). In the second part *Comp* is shown to hold for each typeable term.

In the following we will assume that $(\Sigma,\mathbf{R})$ is a typeable and HNF-safe $\mathcal{U}$TRS in the environment $\mathcal{E}$.

Definition 5.14 *i*) Let B be a basis, t a term, and σ a type such that $B \vdash_{\mathcal{E}} t{:}\sigma$. We define the Computability Predicate $Comp\,(B,t,\sigma)$ recursively on σ by:

a) $Comp\,(B,t,\varphi) \iff HN\,(t)$ & $CHN\,(t)$.

b) $Comp\,(B,t,s) \iff HN\,(t)$ & $CHN\,(t)$.

c) $Comp\,(B,t,\alpha\to\beta) \iff \forall\, u \in T(\mathcal{F},\mathcal{X})\ [\ Comp\,(B',u,\alpha) \Rightarrow Comp\,(\Pi\{B,B'\},Ap\,(t,u),\beta)\ ]$

d) $Comp\,(B,t,\sigma_1\cap\cdots\cap\sigma_n)\ (n \geq 0) \iff \forall\, 1 \leq i \leq n\ [\ Comp\,(B,t,\sigma_i)\ ]$.

ii) We say that a term t is *computable of type* σ, if there is a basis B such that $Comp\,(B,t,\sigma)$.

iii) We say that a term-substitution R is *computable in basis* B if there is a basis B' such that for every $x{:}\sigma \in B$, $Comp\,(B',x^{\mathrm{R}},\sigma)$ holds.

Notice that $Comp\,(B,t,\omega)$ holds as special case of part (*i.d*). Also, since we use intersection types, and because of Definition 3.2, in part (*iii*) we need not consider the existence of different bases for each $x{:}\sigma \in B$.

We are going to prove that *Comp* satisfies the Properties C1, which now states head-normalization of computable terms, and C3, which will be divided in two parts to gain readability. Notice that with the notion of computability we are using here we do not need to prove Property C2.

Property 5.15 (*C1*): $Comp\,(B,t,\sigma)$ & $\sigma \neq \omega \Rightarrow HN\,(t)$ & $CHN\,(t)$.

(*C3*): *Let t be neutral. If $B \vdash_{\mathcal{E}} t{:}\sigma$, and there is a v such that $Comp\,(B,v,\sigma)$ and $t \to^*_{\mathbf{R}} v$, then $Comp\,(B,t,\sigma)$.*

(*C3'*): *Let t be neutral. If $B \vdash_{\mathcal{E}} t{:}\sigma$, $In\text{-}Hnf\,(t)$, and $In\text{-}Chnf\,(t)$, then $Comp\,(B,t,\sigma)$.*

Proof: By simultaneous induction on the structure of types.

i) $\sigma = \varphi$, or $\sigma = s \in S$. By Definition 5.14 (*i.a*) and (*i.b*), and Theorem 4.25 for Property C3. For Property C3', notice that $In\text{-}Hnf\,(t)$ implies $HN\,(t)$, and $In\text{-}Chnf\,(t)$ implies $CHN\,(t)$.

ii) $\sigma = \alpha\to\beta$.

(*C1*): $Comp\,(B,t,\alpha\to\beta)$ & x not in $B \Rightarrow$ (*IH C3'*)

$Comp\,(B,t,\alpha\to\beta)$ & $Comp\,(\{x{:}\alpha\},x,\alpha) \Rightarrow$ (*5.14 (i.c)*)

$Comp\,(B \cup \{x{:}\alpha\},Ap\,(t,x),\beta) \Rightarrow$ (*IH C1*)

$HN\,(Ap\,(t,x))$ & $CHN\,(Ap\,(t,x))$ $\Rightarrow$ (2.11(i) & (ii))
$HN\,(t)$ & $CHN\,(t)$.

(C3): t is neutral & $B \vdash_{\mathcal{E}} t{:}\alpha\to\beta$ & $\exists\, v\ [t \to^*_{\mathbf{R}} v$ & $Comp\,(B,v,\alpha\to\beta)]$ $\Rightarrow$ (5.14 (i.c))
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ \exists\, v\ [t \to^*_{\mathbf{R}} v$ & $Comp\,(\Pi\{B,B'\},Ap\,(v,u),\beta)]\ )$ $\Rightarrow$ (2.11(v))
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ \exists\, v\ [Ap\,(t,u) \to^*_{\mathbf{R}} v$ & $Comp\,(\Pi\{B,B'\},v,\beta)]\ )$ $\Rightarrow$ (IHC3)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\},Ap\,(t,u),\beta)\ )$ $\Rightarrow$ (5.14 (i.c))
$Comp\,(B,t,\alpha\to\beta)$.

(C3′): t is neutral & $B \vdash_{\mathcal{E}} t{:}\alpha\to\beta$ & $In\text{-}Hnf\,(t)$ & $In\text{-}Chnf\,(t)$ $\Rightarrow$ (2.11(iii))
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Ap\,(t,u)$ neutral & $\Pi\{B,B'\} \vdash_{\mathcal{E}} Ap\,(t,u){:}\beta$ &
$In\text{-}Hnf\,(Ap\,(t,u))$ & $In\text{-}Chnf\,(Ap\,(t,u))\ )$ $\Rightarrow$ (IHC3′)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\},Ap\,(t,u),\beta)\ )$ $\Rightarrow$ (5.14 (i.c))
$Comp\,(B,t,\alpha\to\beta)$.

iii) $\sigma = \sigma_1\cap\cdots\cap\sigma_n$.

(C1): $Comp\,(B,t,\sigma_1\cap\cdots\cap\sigma_n)$ $\Rightarrow$ (5.14 (i.d))
$\forall\, 1\le i\le n\ [Comp\,(B,t,\sigma_i)]$ $\Rightarrow$ (IHC1 & $n \neq 0$)
$HN\,(t)$ & $CHN\,(t)$.

(C3): t is neutral & $B \vdash_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n$ & $\exists\, v\ [t \to_{\mathbf{R}} v$ & $Comp\,(B,v,\sigma_1\cap\cdots\cap\sigma_n)]$ $\Rightarrow$
(4.12(iv) & 5.14 (i.d))
$\exists\, v\ [t \to_{\mathbf{R}} v$ & $\forall\, 1\le i\le n\ [Comp\,(B,v,\sigma_i)$ & $B \vdash_{\mathcal{E}} t{:}\sigma_i]]$ $\Rightarrow$ (IHC3)
$\forall\, 1\le i\le n\ [Comp\,(B,t,\sigma_i)]$ $\Rightarrow$ (5.14 (i.d))
$Comp\,(B,t,\sigma_1\cap\cdots\cap\sigma_n)$.

(C3′): t is neutral & $B \vdash_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n$ & $In\text{-}Hnf\,(t)$ & $In\text{-}Chnf\,(t)$ $\Rightarrow$ (4.12(iv))
t is neutral & $\forall\, 1\le i\le n\ [B \vdash_{\mathcal{E}} t{:}\sigma_i$ & $In\text{-}Hnf\,(t)$ & $In\text{-}Chnf\,(t)]$ $\Rightarrow$ (IHC3′)
$\forall\, 1\le i\le n\ [Comp\,(B,t,\sigma_i)]$ $\Rightarrow$ (5.14 (i.d))
$Comp\,(B,t,\sigma_1\cap\cdots\cap\sigma_n)$. ■

In order to prove that typeable terms are computable we shall prove a stronger property, for which we will need the following ordering and lemma.

Definition 5.16 Let $(\Sigma,\mathbf{R})$ be a $\mathcal{A}$TRS.

i) We define the ordering $\gg^{\mathrm{HNF}}$ on triples – consisting of a pair of a natural number and a multiset of natural numbers, a term, and a multiset of terms – as the object

$$((>_{\mathbb{N}},(>_{\mathbb{N}})_{mul})_{lex},\rhd\ ,(\ \to_{Chnf}\cup\rhd_{Chnf})_{mul})_{lex},$$

where

a) $t \to_{Chnf} t'$ if $t \to^*_{\mathbf{R}} t'$, $\neg In\text{-}Chnf\,(t)$ and $In\text{-}Chnf\,(t')$,

b) $t \rhd_{Chnf} t'$ if $t \rhd t'$, $In\text{-}Chnf\,(t)$, and $In\text{-}Chnf\,(t')$.

ii) Let t be such that $B \vdash_{\mathcal{E}} t{:}\sigma$, and R a term-substitution, computable in B. We interpret the term t^{R} by the triple $\mathcal{I}(t^{\mathrm{R}}) = \langle (i,M),t,\{\mathrm{R}\}\rangle$, where

a) i is the maximal super-index of the function symbols belonging to t,

b) M is the multiset of the differences $arity\,(F^j)$ - $arity\,(F^j_k)$ such that F^j_k occurs in t,

c) $\{\mathrm{R}\}$ is the multiset $\{x^{\mathrm{R}} \mid x \in Var\,(t)$ & $\exists\,\rho\ [x{:}\rho \in B]\}$.

These triples are compared in the ordering $\gg^{\mathrm{HNF}}$, which is well-founded.

Lemma 5.17 $Comp\,(B,t,\sigma)$ & $\sigma \le \rho$ $\Rightarrow$ $Comp\,(B,t,\rho)$.

Proof: By induction on the definition of $\le$. ■

We now come to the main theorem of this section, in which we will show that, for any term typeable with σ and computable term-substitution R such that the term t^{R} is typeable, t^{R} is computable in σ. The technique used in this proof is the same as in the proof of Property 5.11.

Property 5.18 Let t be a term such that $B \vdash_{\mathcal{E}} t{:}\sigma$, and R *a term-substitution computable in B. Then there exists B' such that $Comp\,(B', t^{\mathrm{R}}, \sigma)$.*

Proof: By noetherian induction on $\gg^{\mathrm{HNF}}$ (which is well-founded), using $\mathcal{I}(t^{\mathrm{R}})$.

If $\sigma = \omega$ then t^{R} is trivially computable of type ω. If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, then by Definition 5.14 (*i.d*), we have to prove that, for every $1 \leq i \leq n$, $B \vdash_{\mathcal{E}} t{:}\sigma_i$ and $Comp\,(B, t^{\mathrm{R}}, \sigma_i)$. So, without loss of generality we can assume that $\sigma \in \mathcal{T}_{\mathrm{s}}$.

If t is a variable then, by ($\leq$) there is a τ, such that $x{:}\tau \in B$, and $\tau \leq \sigma$. Since t^{R} is computable of type τ, by Lemma 5.17, it is also computable of type σ. We will now consider the case that t is not a variable (so neither is t^{R}). We will prove that t^{R} is computable of type σ.

We consider separately the cases:

i) t^{R} is neutral.

($In\text{-}Hnf\,(t^{\mathrm{R}})$ & $In\text{-}Chnf\,(t^{\mathrm{R}})$): Then, by Property C3′, t^{R} is computable of type σ.

($In\text{-}Hnf\,(t^{\mathrm{R}})$ & $\neg In\text{-}Chnf\,(t^{\mathrm{R}})$): Then $t^{\mathrm{R}} = Q\,(t_1, \ldots, t_n)$, for Q a constructor or $t^{\mathrm{R}} = Ap\,(t_1, t_2)$ (otherwise $In\text{-}Chnf\,(t^{\mathrm{R}})$). For $1 \leq i \leq n$, t_i is computable, either because it is in R or by induction.

1) In case $t^{\mathrm{R}} = Q\,(t_1, \ldots, t_n)$, because a constructor can only have a 'sort'-type, each term t_i ($1 \leq i \leq n$) is computable of a type different from ω. Then by Property C1 they have a constructor-hat normal form, which implies $CHN\,(t^{\mathrm{R}})$.

2) In case $t^{\mathrm{R}} = Ap\,(t_1, t_2)$, t_1 is computable of type $\alpha \rightarrow \beta$, for some α, β (so in particular, $\alpha \rightarrow \beta \neq \omega$), then, by Property C1, $CHN\,(t_1)$ which implies $CHN\,(t^{\mathrm{R}})$.

In both cases, t^{R} reduces to a neutral term t' such that $In\text{-}Chnf\,(t')$ and $In\text{-}Hnf\,(t')$, and by Theorem 4.25, $B \vdash_{\mathcal{E}} t'{:}\sigma$. By Property C3′, t' is computable of type σ, and t^{R} is computable of type σ by Property C3.

($\neg In\text{-}Hnf\,(t^{\mathrm{R}})$): Then two cases are possible for t^{R}.

1) $t = Ap\,(t_1, t_2)$. In this case, again ${t_1}^{\mathrm{R}}$ and ${t_2}^{\mathrm{R}}$ are computable either because they are in $\{\mathrm{R}\}$, or by induction (since the second component of the interpretation is strictly smaller). By assumption, $B \vdash_{\mathcal{E}} t{:}\sigma$ and $\sigma \neq \omega$, so ${t_1}^{\mathrm{R}'}$ must have an arrow type, and since it is computable, t^{R} is computable by Definition 5.14 (*i.c*).

2) $t = F^k\,(t_1, \ldots, t_n)$, with F^k a defined symbol.

If $t \neq F^k\,(z_1, \ldots, z_n) = u$ modulo renaming of variables, where $z_1, \ldots, z_n$ are different variables, then $t^{\mathrm{R}} = u^{\mathrm{R}'}$, where R′ is the term-substitution that assigns ${t_i}^{\mathrm{R}}$ to each z_i. By induction, R′ is computable since $\mathcal{I}\,({t_i}^{\mathrm{R}}) \gg_2^{\mathrm{HNF}} \mathcal{I}\,(t^{\mathrm{R}})$. And since $\mathcal{I}\,(t^{\mathrm{R}}) \gg_2^{\mathrm{HNF}} \mathcal{I}\,(u^{\mathrm{R}'})$, again by induction we obtain that $u^{\mathrm{R}'}$ (which is the same as t^{R}) is computable.

If $t = F^k\,(z_1, \ldots, z_n)$ modulo renaming of variables, where $z_1, \ldots, z_n$ are different variables (without loss of generality we can assume $t = u$), then we distinguish the cases:

A) t^{R} is not itself a redex. For $1 \leq i \leq n$, ${z_i}^{\mathrm{R}}$ is computable because it is in $\{\mathrm{R}\}$. Moreover, each ${z_i}^{\mathrm{R}}$ that appears in a pattern position of a rule defining F^k must be typed with a sort, since the system is HNF-safe. Then these terms are constructor-hat normalizable by Property C1. For each ${z_i}^{\mathrm{R}}$ in a pattern position we can compute its constructor-hat normal form t'_i (and we know that at least one ${t_i}^{\mathrm{R}}$ can be reduced in this way, because we are assuming $\neg In\text{-}Hnf\,(t^{\mathrm{R}})$).

Let R' be the term-substitution that assigns to each z_i its corresponding t'_i. Then $\mathcal{I}(t^{\mathrm{R}}) \gg_3^{\mathrm{HNF}} \mathcal{I}(t^{\mathrm{R}'})$, and $t^{\mathrm{R}'}$ is computable by induction. Then t^{R} is computable by Property C3.

B) t^{R} is a redex, that is, $t = F^k(z_1,\ldots,z_n)$, and t^{R} is reducible at the root position. Then there is a rewrite rule

$$F^k(\bar{\mathrm{C}}[\bar{x}], \bar{y}) \to \mathrm{C}'[F^k(\overline{\mathrm{C}_1}[\bar{x}], \bar{y}), \ldots, F^k(\overline{\mathrm{C}_m}[\bar{x}], \bar{y}), \bar{y}]$$

such that $t^{\mathrm{R}} = F^k(\bar{\mathrm{C}}[\overline{M}], \overline{N})$. By definition of HNF-safe $\mathcal{A}$TRS, the patterns $\bar{\mathrm{C}}[\bar{x}]$ are constructor terms with sorts as types, hence, since R is computable, $CHN(\bar{\mathrm{C}}[\overline{M}])$. Let R' be the computable term-substitution obtained from R by reducing all $\bar{\mathrm{C}}[\overline{M}]$ to constructor-hat normal form (note that R' is computable by Definition 5.14 (*i.b*) since the $\bar{\mathrm{C}}[\overline{M}]$ are typed with sorts). There are two possible cases: either $\mathcal{I}(t^{\mathrm{R}}) \gg_3^{\mathrm{HNF}} \mathcal{I}(t^{\mathrm{R}'})$, and then $t^{\mathrm{R}'}$ is computable by induction, and so is t^{R} by Property C3, or the terms $\overline{M}$are already in constructor-hat normal form, and

$$t^{\mathrm{R}} = F^k(\bar{\mathrm{C}}[\overline{M}], \overline{N}) \to_{\mathbf{R}} \mathrm{C}'[F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}), \overline{N}].$$

In the latter case, since $\overline{N}$and $\overline{M}$ are computable (because $\overline{M} \subseteq \overline{N}$and $\overline{N}$is in R), the terms $\overline{\mathrm{C}_i}[\overline{M}]$ are computable by induction. Also, by definition of the scheme and because *In-Chnf*$(\overline{M})$, $\bar{\mathrm{C}}[\overline{M}]\ (\rhd_{Chnf})_{mul}\ \overline{\mathrm{C}_i}[\overline{M}]$, then $F^k(\overline{\mathrm{C}_i}[\overline{M}], \overline{N})$ is computable by induction. Again, by definition of the scheme and induction,

$$\mathrm{C}'[F^k(\overline{\mathrm{C}_1}[\overline{M}], \overline{N}), \ldots, F^k(\overline{\mathrm{C}_m}[\overline{M}], \overline{N}), \overline{N}]$$

is computable, since $\mathrm{C}'[\bar{x}]$ does not contain F^k. Then, by Property C3, t^{R} is computable since it is neutral.

ii) t^{R} is not neutral. Then $t = F_i^k(t_1,\ldots,t_i)$, where F_i^k is a Curryfied version of some function symbol F^k. Then, by Lemma 4.12(*v*), t must have an arrow type $\alpha\to\beta$. We have to prove that $Ap(F_i^k(t_1,\ldots,t_i),z)^{\mathrm{R}'}$ is computable for any term-substitution $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto u\}$ such that u is computable of type α. But since $Ap(F_i^k(t_1,\ldots,t_i),z)^{\mathrm{R}'}$ is a neutral term, it is sufficient to prove that it reduces to a computable term and then to use Property C3. Now, by definition of $\mathcal{A}$TRS, $Ap(F_i^k(t_1,\ldots,t_i),z)^{\mathrm{R}'} \to_{\mathbf{R}} F_{i+1}^k(t_1,\ldots,t_i,z)^{\mathrm{R}'}$ and since

$$\mathcal{I}(Ap(F_i^k(t_1,\ldots,t_i),z)^{\mathrm{R}'}) \gg_1^{\mathrm{HNF}} \mathcal{I}(F_{i+1}^k(t_1,\ldots,t_i,z)^{\mathrm{R}'}),$$

we have that $F_{i+1}^k(t_1,\ldots,t_i,z)^{\mathrm{R}'}$ is computable by induction, and we are finished. ■

Theorem 5.19 (Head Normalization Theorem) *If* $(\Sigma,\mathbf{R})$ *is typeable in* $\vdash_{\mathcal{E}}$ *and HNF-safe, then for every term t such that* $B \vdash_{\mathcal{E}} t{:}\sigma$ *and* $\sigma \neq \omega$, $HN(t)$.

Proof: The theorem follows from Properties 5.18 and C1, taking R such that $x^{\mathrm{R}} = x$, which is computable by Property C3′. ■

5.3 Normalization

In the intersection system for LC it is well-known that terms that are typeable without ω in basis and type are normalizable. This is not true in the rewriting framework, even if one considers safe recursive systems only. Take for instance the safe system:

$$\begin{aligned} Z(x,y) &\to y \\ D(x) &\to Ap(x,x). \end{aligned}$$

The term $Z_1(D(D_0))$ has type $\varphi\to\varphi$ in an environment where Z is typed with $\varphi_1\to\varphi_2\to\varphi_2$ and D with $(\varphi_3\to\varphi_4)\cap\varphi_3\to\varphi_4$, but is not normalizable. The characterization of normalization can therefore only be obtained for a restricted class of terms. We will consider only non-Curryfied

terms and $\mathcal{G}$TRS where reduction is closed on non-Curryfied terms (the latter will be called non-Curryfied $\mathcal{G}$TRS). Actually, to get a normalization result similar to that of LC we will also need to impose the following condition on $\mathcal{G}$TRS:

Definition 5.20 A $\mathcal{G}$TRS is *complete* if whenever a typeable non-Curryfied term t of which the type does not contain ω has a reducible subterm $t|_p$ that is typeable with a type containing ω, there exists $q < p$ such that $t|_q$ is typeable with a type without ω and $t|_q[x]_p$ (where x is a fresh variable) is not in head-normal form.

Intuitively, in a complete $\mathcal{G}$TRS, a term $F(t_1,\ldots,t_n)$ that can be assigned a type that does not contain ω, and where a t_i typeable with a type that contains ω is a redex, will be reducible either at the root or in some t_j, typeable with a type that does not contain ω. This means that the rules defining F cannot have patterns that have types with ω, and also that constructors cannot accept arguments having a type that contains ω. Moreover, if a defined function accepts arguments having types with ω, then its definition must be exhaustive.

Constructors and defined functions of HNF-safe systems satisfy the two first conditions. So, a HNF-safe recursive system is complete whenever for all defined function F that accepts arguments with types that contain ω, the patterns of the rules defining F cover all possible cases.

From now on, we will consider only non-Curryfied $\mathcal{G}$TRS that are HNF-safe and complete. We will call this class of systems *NF-safe*. This section will be devoted to the proof of the Normalization Theorem:

Let t be a non-Curryfied term in a typeable, NF-safe $\mathcal{G}$TRS.
If $B \vdash_{\mathcal{E}} t{:}\sigma$ and σ does not contain ω, then t is normalizable.

We could use the method of Computability Predicates, as in the previous section, but since only non-Curryfied terms are considered, a direct proof is simpler. We will prove the theorem by noetherian induction, for which we will use the following ordering:

Definition 5.21 Let $(\Sigma,\mathbf{R})$ be a $\mathcal{G}$TRS. Let $\succ$ denote the following well-founded ordering between terms: $t \succ t'$ if $t \rhd t'$ or t' is obtained from t by replacing the subterm $t|_p = F(t_1,\ldots,t_n)$ by the term $F(s_1,\ldots,s_n)$ where $\{t_1,\ldots,t_n\} \rhd_{mul} \{s_1,\ldots,s_n\}$. We define the ordering $\gg^{\mathrm{NF}}$ on triples composed of a natural number and two terms, as the object $(>_{\mathbb{N}}, \rhd, \succ)_{lex}$.

Theorem 5.22 (Normalization Theorem) *Let t be a non-Curryfied term in a typeable, and NF-safe $\mathcal{G}$TRS. If $B \vdash_{\mathcal{E}} t{:}\sigma$ and ω does not appear in σ, then t is normalizable.*

Proof: By noetherian induction on $\gg^{\mathrm{NF}}$. We will interpret the term u by the triple $\mathcal{I}(u) = \langle i, u', u\rangle$ where i is the maximum of the super-indexes of the function symbols belonging to u that do not appear only in subterms in normal form or having a type with ω, and u' is the term obtained from u by replacing subterms in normal form with fresh variables. These triples are compared using $\gg^{\mathrm{NF}}$.

Assume that t is not in normal form. For every strict subterm u of t that has a type without ω, either *In-Nf*(u), or u is smaller than t with respect to $\gg^{\mathrm{NF}}$ and then $N(u)$ by induction. Let v be the term obtained from t by reducing these subterms to normal form.

If $v \neq t$ then $\mathcal{I}(t) \gg_2^{\mathrm{NF}} \mathcal{I}(v)$, $N(v)$ by induction, and so $N(t)$.

If $v = t$ and it is a normal form, we are done. Otherwise, since the system is complete, t must be reducible at the root, and since it is a non-Curryfied term, the only possible reduction is:

$$t = F^i\,(\overline{\mathrm{C}}[\overline{M}], \overline{N}) \rightarrow_{\mathbf{R}} \mathrm{C}'[F^i\,(\overline{\mathrm{C}_1}[\overline{M}],\, \overline{N}), \ldots, F^i\,(\overline{\mathrm{C}_m}[\overline{M}],\, \overline{N}),\, \overline{N}]$$

Now the subterms of the right-hand side of the form:

$$F^i\,(\overline{\mathrm{C}_1}[\overline{M}],\, \overline{N}), \ldots, F^i\,(\overline{\mathrm{C}_m}[\overline{M}],\, \overline{N}),\, \overline{N}$$

that have a type without ω are normalizable by induction. Let $\mathrm{C}''[\bar{u}]$ be the term obtained after normalizing those subterms, and including in the context the subterms that have a type with ω. By the Subject Reduction Theorem, this term has a type without ω, and by definition of the general scheme, it is smaller than t. Then $N\,(\mathrm{C}''[\bar{u}])$ by induction. ■

6 Conclusions and Final Remarks

Intersection type assignment systems for LC have two nice properties: they are closed under β-equality (whereas Curry's system is only closed under β-reduction) and the sets of head-normalizable, normalizable, and strongly normalizable λ-terms can be characterized by the sets of assignable types.

We have shown that also in the world of term rewriting, intersection type systems are a useful tool to study normalization properties. But typeability alone is not enough in this setting. Our normalization results take also the way in which recursion is used in the rewrite system into account and that is where the general scheme plays an important role. The general scheme of recursion has been used in many different contexts to ensure strong normalization of rewriting. It is interesting to notice that in the context of $\mathcal{A}$TRS, neither the general scheme nor the type system alone can guarantee any normalization property, as the examples given in this paper show. It is their combination that provides the right framework for the study of normalization properties of $\mathcal{A}$TRS. This is in contrast with first-order term rewriting systems (without Ap), where the general scheme alone ensures strong normalization (and since strong normalization is preserved under Curryfication, also the system *PP(R)* obtained by Currying a first-order system R, is strongly normalizing in this case).

Combinator Systems can be seen as a particular case of $\mathcal{A}$TRS (see for instance the $\mathcal{A}$TRS for Combinatory Logic in Example 2.5). Moreover, Combinator Systems are trivially safe (since all left-hand sides of rules have the form $C(x_1,\ldots,x_n)$, where $x_1,\ldots,x_n$ are different variables, and right-hand sides contain only variables and Ap), hence all the results presented in this paper hold in particular for these systems. Dezani and Hindley presented a type assignment system for Combinator Systems that are combinatory complete [22]: in order to assign a type to a term, it is assumed that there is a basic type for each combinator, which coincides with the principal type of the corresponding λ-term. Our system can be seen as an extension of this one, since we do not require the systems to be combinatory complete. The results we showed also apply to the type assignment system of Dezani and Hindley.

The type systems presented in this paper (with and without ω) are undecidable in general, so they cannot be directly included in the interpreter of a programming language. However, if we restrict the set of types by considering only intersection types of rank 2 (as in [7]), then the system becomes decidable, and the same normalization results hold in the restricted system.

The relation between typeability and normalization properties of terms can be studied directly, as was done in this paper, or through the notion of approximant. Approximants have been defined both for LC and TRS, and used mainly to study reduction properties and semantic properties of these systems. Intuitively, approximants can be seen as descriptions of the normal forms of terms, and they are meaningful for terms having at least a head normal form. In the future we will study the relation between approximation, normalization, and typeability in the essential intersection system for $\mathcal{A}$TRS.

References

[1] Abramsky, S. (1990). The Lazy Lambda Calculus. In D. Turner, editor, *Research Topics in Functional Programming*. pages 65–117. Addison Wesley.

[2] Ariola, Z., Kennaway, R., Klop, J.W., Sleep, R., and de Vries, F-J. (1994). Syntactic definitions of undefined: on defining the undefined. In Hagiya, M. and Mitchell, J.C., editors, *Proceedings of TACS '94. International Symposium on Theoretical Aspects of Computer Software,* Sendai, Japan, volume 789 of *Lecture Notes in Computer Science*, pages 543–554. Springer-Verlag.

[3] van Bakel, S. (1992). Complete restrictions of the Intersection Type Discipline. *Theoretical Computer Science*, 102:135–163.

[4] van Bakel, S. (1993a). Partial Intersection Type Assignment in Applicative Term Rewriting Systems. In Bezem, M. and Groote, J.F., editors, *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications,* Utrecht, the Netherlands, volume 664 of *Lecture Notes in Computer Science*, pages 29–44. Springer-Verlag.

[5] van Bakel, S. (1993b). Principal type schemes for the Strict Type Assignment System. *Logic and Computation*, 3(6):643–670.

[6] van Bakel, S. (1995). Intersection Type Assignment Systems. *Theoretical Computer Science*, 151(2):385–435.

[7] van Bakel, S. (1996). Rank 2 Intersection Type Assignment in Term Rewriting Systems. *Fundamenta Informaticae*. To appear.

[8] van Bakel, S. and Fernández, M. (1994). Strong Normalization of Typeable Rewrite Systems. In Heering, J., Meinke, K., Möller, B., and Nipkow, T., editors, *Proceedings of HOA '93. First International Workshop on Higher Order Algebra, Logic and Term Rewriting,* Amsterdam, the Netherlands. *Selected Papers*, volume 816 of *Lecture Notes in Computer Science*, pages 20–39. Springer-Verlag.

[9] van Bakel, S. and Fernández, M. (1995). (Head-)Normalization of Typeable Rewrite Systems. In Hsiang, J, editor, *Proceedings of RTA '95. 6th International Conference on Rewriting Techniques and Applications,* Kaiserslautern, Germany, volume 914 of *Lecture Notes in Computer Science*, pages 279–293. Springer-Verlag.

[10] van Bakel, S., Smetsers, S., and Brock, S. (1992). Partial Type Assignment in Left Linear Applicative Term Rewriting Systems. In Raoult, J.-C., editor, *Proceedings of CAAP '92. 17th Colloquium on Trees in Algebra and Programming,* Rennes, France, volume 581 of *Lecture Notes in Computer Science*, pages 300–321. Springer-Verlag.

[11] Barbanera, F. and Fernández, M. (1993). Combining first and higher order rewrite systems with type assignment systems. In Bezem, M. and Groote, J.F., editors, *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications,* Utrecht, the Netherlands, volume 664 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag.

[12] Barbanera, F., Fernández, M., and Geuvers, H. (1994). Modularity of Strong Normalization and Confluence in the Algebraic λ-cube. In *Proceedings of the ninth Annual IEEE Symposium on Logic in Computer Science,* Paris, France.

[13] Barendregt, H. (1984). *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition.

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

[15] Barendregt, H.P., van Eekelen, M.C.J.D., Glauert, J.R.W., Kennaway, J.R., Plasmeijer, M.J., and Sleep, M.R. (1987). Term graph rewriting. In *Proceedings of PARLE, Parallel Architectures and Languages Europe,* Eindhoven, The Netherlands, volume 259-II of *Lecture Notes in Computer Science*, pages 141–158. Springer-Verlag.

[16] Barendsen, E. and Smetsers, S. (1993). Conventional and Uniqueness Typing in Graph Rewrite Systems. In Shyamasunda, R.K., editor, *Proceedings of FST&TCS '93. 13th Conference on Foundations of Software Technology and Theoretical Computer Science,* Bombay, India, volume 761 of *Lecture Notes in Computer Science*, pages 41–52. Springer-Verlag.

[17] Brus, T., van Eekelen, M.C.J.D., van Leer, M.O., and Plasmeijer, M.J. (1987). Clean - A Language for Functional Graph Rewriting. In *Proceedings of the Third International Conference on Functional Programming Languages and Computer Architecture,* Portland, Oregon, USA, volume 274 of *Lecture Notes in Computer Science*, pages 364–368. Springer-Verlag.

[18] Coppo, M. and Dezani-Ciancaglini, M. (1980). An Extension of the Basic Functionality Theory for the λ-Calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693.

[19] Coppo, M., Dezani-Ciancaglini, M., and Venneri, B. (1980). Principal type schemes and λ-calculus semantics. In Hindley, J.R. and Seldin, J.P., editors, *To H.B. Curry, Essays in combinatory logic, lambda-calculus and formalism*, pages 535–560. Academic press, New York.

[20] Curry, H.B. and Feys, R. (1958). *Combinatory Logic*, volume 1. North-Holland, Amsterdam.

[21] Dershowitz, N. and Jouannaud, J.P. (1990). Rewrite systems. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 245–320. North-Holland.

[22] Dezani-Ciancaglini, M. and Hindley, J.R. (1992). Intersection types for combinatory logic. *Theoretical Computer Science*, 100:303–324.

[23] Fernández, M. and Jouannaud, J.P. (1994). Modular termination of term rewriting systems revisited. In Astesiano, E. and Tarlecki, A., editors, *Recent Trends in Data Type Specification. 10th Workshop on Specification of Abstract Data Types,* S. Margherita, Italy, volume 906 of *Lecture Notes in Computer Science*, pages 255–272. Springer-Verlag.

[24] Futatsugi, K., Goguen, J., Jouannaud, J.P., and Meseguer, J. (1985). Principles of OBJ2. In *Proceedings 12th ACM Symposium on Principles of Programming Languages*, pages 52–66.

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

[26] Hindley, J.R. (1969). The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60.

[27] Jouannaud, J.P. and Okada, M. (1991). Executable higher-order algebraic specification languages. In *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 350–361.

[28] Kahrs, S. (1996). Confluence of Curried Term-Rewriting Systems. *Journal of Symbolic Computation*, 19(6):601–623.

[29] Kennaway, R., Klop, J.W., Sleep, R., and de Vries, F.J. (1996). Comparing curried and uncurried rewriting. *Journal of Symbolic Computation*. To appear.

[30] Klop, J.W. (1987). Term Rewriting Systems: a tutorial. *EATCS Bulletin*, 32:143–182.

[31] Klop, J.W. (1992). Term Rewriting Systems. In Abramsky, S., Gabbay, Dov.M., and Maibaum, T.S.E., editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–116. Clarendon Press.

[32] Milner, R. (1978). A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375.

[33] Pfenning, F. (1988). Partial Polymorphic Type Inference and Higher-Order Unification. In *Proceedings of the 1988 ACM conference on LISP and Functional Programming Languages*, pages 153–163.

[34] Pierce, B.C. (1991). *Programming with Intersection Types and Bounded Polymorphism*. PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh. CMU-CS-91-205.

[35] Tait, W.W. (1967). Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–223.

[36] Turner, D.A. (1985). Miranda: A non-strict functional language with polymorphic types. In *Proceedings of the conference on Functional Programming Languages and Computer Architecture*, volume 201 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag.