(Head-)Normalization of Typeable Rewrite Systems

(Proceedings of RTA'95, LNCS 914, pages 279–293, 1995)

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

[1] Dipartimento di Informatica, Università degli Studi di Torino,
Corso Svizzera 185, 10149 Torino, Italy.
[2] DMI - LIENS (CNRS URA 1327), Ecole Normale Supérieure,
45, rue d'Ulm, 75005 Paris, France.
bakel@di.unito.it maribel@ens.ens.fr

Abstract

In this paper we study normalization properties of rewrite systems that are typeable using intersection types with ω and with sorts. We prove two normalization properties of typeable systems. On one hand, for all systems that satisfy a variant of the Jouannaud-Okada Recursion Scheme, every term typeable with a type that is not ω is head normalizable. On the other hand, non-Curryfied terms that are typeable with a type that does not contain ω, are normalizable.

Introduction

In the study of termination of reduction systems, the notion of types has played an important role. A well explored area in this aspect is that of the Lambda Calculus (LC). For LC, there exists a well understood notion of type assignment, known as the Curry Type Assignment System [8], which expresses abstraction and application, and introduces $\rightarrow$-types. A well-known result for this system is that all typeable terms are strongly normalizable. Another notion of type assignment for LC for which the relation between typeability and normalization has been studied profoundly, is the Intersection Type Discipline (the BCD-system), as presented in [7], that is an extension of Curry's system. The extension consists of allowing more than one type for term-variables and adding a type constant 'ω', and, next to the type constructor '$\rightarrow$', the type constructor '$\cap$'. The set of lambda terms having a head normal form, the set of lambda terms having a normal form, and the set of strongly normalizable lambda terms can all be characterized by the set of their assignable types.

In this paper, instead of studying the problem of termination using types in the setting of LC, the approach taken will be to study the desired property directly on the level of a programming language with patterns, i.e. in the world of term rewriting systems (TRS). For this purpose, in this paper we define a notion of type assignment on Curryfied TRS (CTRS) that uses intersection types (the system without the type constructors $\cap$ and ω is a particular case; more rules and terms are typeable in the intersection system). CTRS are defined as a slight extension of the TRS defined in [9, 12]. The language of the CTRS is first order (i.e. every function symbol has a fixed arity) but CTRS are assumed to contain a notion of application Ap, that allows partial application (Curryfication) of function symbols in the setting of a first order language.

Unlike typeable terms in LC, typeable terms in CTRS need not be normalizable (consider a typeable term t and a rule $t \rightarrow t$). In order to ensure head normalization of typeable terms in CTRS we will impose some syntactical restrictions on recursive rules, inspired by the recursive scheme defined by Jouannaud and Okada in [11]. This kind of recursive definitions

was presented for the incremental definition of higher order functionals based on first order definitions, such that the whole system is terminating. The general scheme of Jouannaud and Okada was also used in [5] and [6] for defining higher order functions compatible with different lambda calculi, as well as in [4] to obtain a strong normalization result for CTRS that are typeable using ω-free intersection types. The main difference between the recursive scheme for CTRS defined in [4] and the one defined in this paper, is that here the patterns of recursive rules are constructor terms, which have sorts as types. It is worth noticing that without this condition, the recursive scheme of [4] does not ensure head normalization of typeable terms in intersection systems with ω (see examples in Section 3.1).

We will prove (using the well-known method of Computability Predicates [13, 10]) that for all typeable CTRS satisfying the variant of the scheme that is defined in this paper, every typeable term has a head normal form. We will also show that if Curryfication is not allowed, then terms whose type does not contain ω are normalizable. These results, together with the strong normalization result of [4], complete the study of the normalization properties of typeable CTRS in intersection type systems.

In [3] and [2] two partial intersection type assignment systems for TRS are presented. Apart from differences in syntax, the system we present here is the first one extended with type constants (called sorts). The system of [2] is a *decidable* restriction of the one presented here (the restriction lies in the structure of types). In [5] a partial type assignment system for higher order rewrite systems that uses intersection types and sorts (but not ω) is defined. It differs from ours in that function symbols are strongly-typed with sorts only, whereas we allow for types to contain type-variables as well, and in this way we can model polymorphism.

We define CTRS in Section 1, and the intersection type assignment system in Section 2. In Section 3 we study head normalization and normalization of typeable CTRS.

1 Curryfied Term Rewriting Systems

We will define Curryfied Term Rewriting Systems (CTRS) (an extension of the TRS defined in [9, 12]), as first order TRS that allow partial application of function symbols. They were first defined in [4].

Definition 1.1 An *alphabet* or *signature* Σ consists of:

i) A countable infinite set $\mathcal{X}$ of variables $x_1, x_2, x_3, \ldots$ (or $x, y, z, x', y', \ldots$).

ii) A non-empty set $\mathcal{F}$ of *function symbols* $F, G, \ldots$, each equipped with an 'arity'.

iii) A special binary operator, called *application* (Ap).

Definition 1.2 The set $T(\mathcal{F},\mathcal{X})$ of *terms* (or *expressions*) is defined inductively:

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 ($i = 1, \ldots, n$) are the arguments of the last term.

Definition 1.3 A *replacement* R is a map 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 function symbol F (here $n \geq 0$). So, R is determined by its restriction to the set of variables, and sometimes we will use the notation $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ to denote a replacement. We also write t^{R} instead of $\mathrm{R}(t)$.

Definition 1.4 *i*) A *rewrite rule* is a pair (l, r) of terms in $T(\mathcal{F},\mathcal{X})$. Often, a rewrite rule will get a name, e.g. **r**, and we write $\mathbf{r} : l \to r$. Three conditions will be imposed: l is not a

variable, the variables occurring in r are contained in l, and if Ap occurs in l, then $\mathbf{r}$ is of the shape:

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

ii) The systems we consider are Curry-closed, i.e. for every rewrite rule with left hand-side $F\,(t_1,\ldots,t_n)$ there are n additional rewrite rules:

$$\begin{aligned} Ap\,(F_{n-1}\,(x_1,\ldots,x_{n-1}),x_n) &\rightarrow\; F\,(x_1,\ldots,x_n)\\ &\;\;\vdots\\ Ap\,(F_0,x_1) &\rightarrow\; F_1\,(x_1)\end{aligned}$$

$F_{n-1},\ldots,F_0$ are the *Curryfied versions of* F.

iii) A rewrite rule $\mathbf{r} : l \rightarrow r$ determines a set of *reductions* $l^{\mathrm{R}} \rightarrow r^{\mathrm{R}}$ for all replacements R. The left hand side l^{R} is called a *redex*; it may be replaced by its '*contractum*' r^{R} inside a context C[ ]; this gives rise to *rewrite steps*: $\mathrm{C}[\,l^{\mathrm{R}}\,] \rightarrow_{\mathbf{r}} \mathrm{C}[\,r^{\mathrm{R}}\,]$. Concatenating rewrite steps we have *rewrite sequences* $t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \cdots$. If $t_0 \rightarrow \cdots \rightarrow t_n$ we also write $t_0 \rightarrow^* t_n$, and t_n is a *rewrite* or *reduct* of t_0.

iv) A *Curryfied Term Rewriting System* (CTRS) is a pair (Σ, $\mathbf{R}$) of an alphabet Σ and a set $\mathbf{R}$ of rewrite rules. We write $t \rightarrow_{\mathbf{R}} t'$, if there is a $\mathbf{r} \in \mathbf{R}$ such that $t \rightarrow_{\mathbf{r}} t'$. The symbol $\rightarrow^*_{\mathbf{R}}$ denotes the reflexive and transitive closure of $\rightarrow_{\mathbf{R}}$. Terms that contain neither Curryfied versions of symbols, nor Ap, will be called *non-Curryfied terms*.

In this paper we will restrict ourselves to CTRS that satisfy the Church-Rosser property (also known as *confluence*), that is formulated by:

If $t \rightarrow^* u$, and $t \rightarrow^* v$, then there exists w such that $u \rightarrow^* w$, and $v \rightarrow^* w$.

There are several syntactic restrictions that can be posed in rules and reduction strategies so that this property is obtained (see e.g. [12]).

We take the view that in a rewrite rule a certain symbol is defined; it is this symbol to which the structure of the rewrite rule gives a type.

Definition 1.5 *i*) In a rewrite rule, the leftmost, outermost symbol in the left hand side that is not an Ap, is called *the defined symbol* of that rule.

ii) If the symbol F is the defined symbol of the rewrite rule $\mathbf{r}$, then $\mathbf{r}$ *defines* F.

iii) F is *a defined symbol*, if there is a rewrite rule that defines F.

iv) $Q \in \mathcal{F}$ is called a *constructor* or *constant symbol* if Q is not a defined symbol.

Definition 1.6 We assume that rewrite rules are *not* mutually recursive. A TRS whose dependency-graph is an ordered cycle-free graph, is called a *hierarchical* TRS. The rewrite rules of a such a TRS can be regrouped such that they are *incremental* definitions of the defined symbols $F^1, \ldots, F^k$, so that the rules defining F^i only depend on $F^1, \ldots, F^{i-1}$.

Example 1.7 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 rewrite system

$$\begin{aligned} D\,(x) &\rightarrow\; Ap\,(x,x)\\ Ap\,(D_0,x) &\rightarrow\; D\,(x)\end{aligned}$$

is *not* considered a recursive system. Notice that, for example, the term $D\,(D_0)$ has no normal form (these terms play the role of $(\lambda x.xx)(\lambda x.xx)$ in the LC). This means that, in the formalism

of this paper, there exist non-recursive first-order rewrite systems that are not normalizing.

Definition 1.8 *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 is in *constructor-hat normal form* if either

a) it has the form $Ap(t_1,t_2)$ and t_1 is in constructor-hat normal form, or

b) it has the form $C[u_1,\ldots,u_n]$ 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 $C'(s_1,\ldots,s_i)$ where C' is a constructor.

v) A term is *(head, respectively constructor-hat) normalizable* if it can be reduced to a term in (head, respectively constructor-hat) normal form. A rewrite system is *strongly normalizing* (or terminating) if all the rewrite sequences are finite; it is *(head, respectively constructor-hat) normalizing* if every term is (head, respectively constructor-hat) normalizable.

Example 1.9 Take the rules $F(G,H) \to A$, and $B(C) \to G$, then $F(B(C),H)$ is not a redex. But it is neither a head-normal form nor a constructor-hat normal form, since it reduces to $F(G,H)$, which is a redex and reduces to A which is a constructor.

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 1.10 *i)* *HN*$(Ap(t,x)) \Rightarrow$ *HN*(t), *and* *CHN*$(Ap(t,x)) \Rightarrow$ *CHN*(t).

ii) *t is neutral & In-Hnf*$(t) \Rightarrow \forall u$ [*In-Hnf*$(Ap(t,u))$].
t is neutral & In-Chnf$(t) \Rightarrow \forall u$ [*In-Chnf*$(Ap(t,u))$]

iii) *t is neutral* $\Rightarrow \forall u$ [$Ap(t,u)$ *is neutral*].

2 Intersection types and type assignment

Strict types are the types that are strictly needed to assign a type to a term in the system presented in [7] (see also [1]). In the set of strict types, intersection type schemes and the type constant ω play a limited role. We will assume that ω is the same as an intersection over zero elements: if $n = 0$, then $\sigma_1 \cap \cdots \cap \sigma_n \equiv \omega$, so ω does not occur in an intersection subtype. Moreover, intersection type schemes (so also ω) occur in strict types only as subtypes of the left hand side of an arrow type scheme. In this paper we will consider strict types over a set S of sorts (constant types).

Definition 2.1 *i*) $\mathcal{T}_s$, the set of *strict types*, is inductively defined by:

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

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

c) If $\tau, \sigma_1,\ldots,\sigma_n \in \mathcal{T}_s$ $(n \ge 0)$, then $\sigma_1 \cap \cdots \cap \sigma_n \to \tau \in \mathcal{T}_s$.

ii) $\mathcal{T}_S$ is defined by: If $\sigma_1,\ldots,\sigma_n \in \mathcal{T}_s$ $(n \ge 0)$, then $\sigma_1 \cap \cdots \cap \sigma_n \in \mathcal{T}_S$.

iii) On $\mathcal{T}_S$, the relation $\leq_S$ is defined by:
  a) $\forall\, 1 \leq i \leq n\ (n \geq 1)\ [\ \sigma_1 \cap \cdots \cap \sigma_n \leq_S \sigma_i\ ]$.
  b) $\forall\, 1 \leq i \leq n\ (n \geq 0)\ [\ \sigma \leq_S \sigma_i\ ] \Rightarrow \sigma \leq_S \sigma_1 \cap \cdots \cap \sigma_n$.
  c) $\sigma \leq_S \tau \leq_S \rho \Rightarrow \sigma \leq_S \rho$.

iv) We define $\leq$ on $\mathcal{T}_S$ like $\leq_S$, by adding an extra alternative.
  d) $\rho \leq \sigma\ \&\ \tau \leq \mu \Rightarrow \sigma \to \tau \leq \rho \to \mu$.

v) We define the relation $\sim$ by: $\sigma \sim \tau \iff \sigma \leq \tau \leq \sigma$.

$\mathcal{T}_S$ may be considered modulo $\sim$. Then $\leq$ becomes a partial order, and in this paper we consider types modulo $\sim$.

Throughout this paper, the symbol φ (often indexed) will be a type-variable. Greek symbols like α, β, γ, μ, ν, η, ρ, σ and τ will range over types. Unless stated otherwise, if $\sigma_1 \cap \cdots \cap \sigma_n$ is used to denote a type, then all $\sigma_1, \ldots, \sigma_n$ are assumed to be strict. Notice that $\mathcal{T}_s$ is a proper subset of $\mathcal{T}_S$.

Definition 2.2 A *basis* is a set of statements of the shape '$x{:}\sigma$', where x is a term-variable, and $\sigma \in \mathcal{T}_S \backslash \omega$. 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 set of all statements about x that occur in $B_1 \cup \cdots \cup B_n$.

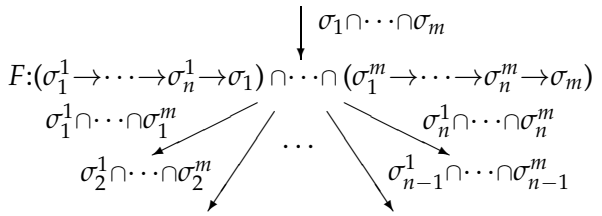
We will often write $B \cup \{x{:}\sigma\}$ for the basis $\Pi\{B, \{x{:}\sigma\}\}$, when x does not occur in B.

Partial intersection type assignment on a CTRS $\mathcal{U}$TRS will be defined as the labelling of nodes and edges in the tree-representation of terms and rewrite rules with types in $\mathcal{T}_S$. All function symbols are assumed to have a type, that is produced by an *environment*.

Our notion of type assignment is based on the definition of (chains of) three operations on types (pairs of basis and type), called substitution, expansion and lifting. Substitution is the operation that instantiates a type (i.e. that replaces type variables by types). The operation of expansion replaces types by the intersection of a number of copies of that type. The operation of lifting replaces basis and type by a smaller basis and a larger type, in the sense of $\leq$. See [3] for formal definitions.

Definition 2.3 Let $\mathcal{U}$TRS be a CTRS, and $\mathcal{E}$ an environment.

i) We say that $t \in T(\mathcal{F}, \mathcal{X})$ *is typeable by* $\sigma \in \mathcal{T}_S$ *with respect to* $\mathcal{E}$, if there exists an assignment of types to edges and nodes that satisfies the following constraints:
  a) The root edge of t is typed with σ; if $\sigma = \omega$, then the root edge is the only thing in the term-tree that is typed.
  b) The type assigned to a function node containing $F \in \mathcal{F} \cup \{Ap\}$ (where F has arity $n \geq 0$) is $\tau_1 \cap \cdots \cap \tau_m$, if and only if for every $1 \leq i \leq m$ there are $\sigma_1^i, \ldots, \sigma_n^i \in \mathcal{T}_S$, and $\sigma_i \in \mathcal{T}_s$, such that $\tau_i = \sigma_1^i \to \cdots \to \sigma_n^i \to \sigma_i$, the type assigned to the j-th $(1 \leq j \leq n)$ outgoing edge is $\sigma_j^1 \cap \cdots \cap \sigma_j^m$, and the type assigned to the incoming edge is $\sigma_1 \cap \cdots \cap \sigma_m$.



c) If the type assigned to a function node containing $F \in \mathcal{F} \cup \{Ap\}$ is τ, then there is a chain C, such that $C(\mathcal{E}(F)) = \tau$.

ii) Let $t \in T(\mathcal{F},\mathcal{X})$ be typeable by σ with respect to $\mathcal{E}$. If B is a basis such that for every statement $x{:}\tau$ occurring in the typed term-tree there is a $x{:}\tau' \in B$ such that $\tau' \leq \tau$, we write $B \vdash_{\mathcal{E}} t{:}\sigma$.

Example 2.4 For every occurrence of Ap in a term-tree, there are $\sigma_1,\ldots,\sigma_n$ and $\tau_1,\ldots,\tau_n$ such that the following is part of the term-tree.

$$\begin{array}{c} \downarrow\ \tau_1\cap\cdots\cap\tau_n \\ Ap\colon ((\sigma_1\to\tau_1)\to\sigma_1\to\tau_1) \cap \cdots \cap ((\sigma_n\to\tau_n)\to\sigma_n\to\tau_n) \\ (\sigma_1\to\tau_1) \cap \cdots \cap (\sigma_n\to\tau_n) \swarrow \quad \searrow\ \sigma_1 \cap \cdots \cap \sigma_n \end{array}$$

As in [3] we can prove:

Lemma 2.5 *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 $B \vdash_{\mathcal{E}} Ap(t,u){:}\sigma$, $\sigma \in \mathcal{T}_s$, then there exist τ, such that $B \vdash_{\mathcal{E}} t{:}\tau\to\sigma$, and $B \vdash_{\mathcal{E}} u{:}\tau$.

iv) $B \vdash_{\mathcal{E}} t{:}\sigma_1 \cap \cdots \cap \sigma_n \iff \forall\, 1 \leq i \leq n\ [B \vdash_{\mathcal{E}} t{:}\sigma_i]$.

v) $B \vdash_{\mathcal{E}} F_n(t_1,\ldots,t_n){:}\sigma\ \&\ \sigma \in \mathcal{T}_s \Rightarrow \exists\, \alpha \in \mathcal{T}_S, \beta \in \mathcal{T}_s\ [\sigma = \alpha\to\beta]$.

In [3] a sufficient condition was formulated in order to ensure the subject reduction property: type assignment on rewrite rules was there defined using the notion of principal pair for a typeable term.

Definition 2.6 Let $t \in T(\mathcal{F},\mathcal{X})$. A pair $\langle \mathrm{P}, \pi\rangle$ is called *a principal pair for t with respect to $\mathcal{E}$*, 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 $C(\langle \mathrm{P}, \pi\rangle) = \langle \mathrm{B}, \sigma\rangle$.

Definition 2.7 Let $(\Sigma, \mathbf{R})$ be a CTRS, and $\mathcal{E}$ an environment.

i) We say that $l \to r \in \mathbf{R}$ 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 nodes and edges such that:

a) $\langle \mathrm{P}, \pi\rangle$ 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 $P \vdash_{\mathcal{E}} r{:}\pi$, all nodes containing F are typed with $\mathcal{E}(F)$.

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

To guarantee the subject reduction property, in this paper we accept only those rewrite rules $l \to r$, that are typeable according to the above definition. As in [3], it is then possible to prove:

Theorem 2.8 (Subject Reduction) *For all replacements* R*, bases B and types σ: if $B \vdash_{\mathcal{E}} l^{\mathrm{R}}{:}\sigma$, then $B \vdash_{\mathcal{E}} r^{\mathrm{R}}{:}\sigma$, so: if $B \vdash_{\mathcal{E}} t{:}\sigma$, and $t \to^*_{\mathbf{R}} t'$, then $B \vdash_{\mathcal{E}} t'{:}\sigma$.*

3 Normal Forms and Head Normal Forms

In [4] we introduced a class of CTRS that are strongly normalizing on terms that are typeable without using the constant ω. In this section we will study normalization properties of CTRS in a type assignment system with ω and sorts.

Since typeability alone is not sufficient to ensure any normalization property, we will impose syntactic restrictions on the rules. As a consequence of the results of this section, the class of typeable *non-recursive* CTRS is head-normalizing on terms whose type is not ω, and normaliz-

ing on non-Curryfied terms whose type does not contain ω. To appreciate the non-triviality of this statement, remember Example 1.7: a non-recursive CTRS may be not head-normalizing. In fact, the main result of this section (every term whose type is not ω is head-normalizable) shows that the term $D\,(D_0)$ is only typeable with ω.

But we will actually prove a stronger result: we will characterize a class of *recursive* definitions for which the same normalization properties hold. These results, together with the previous strong normalization result presented in [4], complete the study of the normalization properties of typeable rewrite systems in the intersection type assignment system.

The converse of our result does not hold: not every (head-)normalizable term is typeable. Take for example the strongly normalizing rewrite system

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

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.

3.1 Head-normalization

In [4] we studied systems that define recursive functions satisfying the *general scheme* below. This scheme was inspired by [11] where generalized primitive recursive definitions were shown strongly normalizing when combined with typed LC. The same results were shown in [5] in the context of type assignment systems for LC and in [6] for the Calculus of Constructions.

Definition 3.1 (General scheme) Let $\mathcal{F}_n = \mathcal{Q} \cup \{F^1, \ldots, F^n\}$, where $F^1, \ldots, F^n$ are the defined symbols of the signature that are not Curryfied-versions, and assume that $F^1, \ldots, F^n$ are defined in an incremental way. Suppose, moreover, that the rules defining $F^1, \ldots, F^n$ satisfy the *general scheme*:

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

where $\vec{x}$, $\vec{y}$ are sequences of variables, and $\vec{x} \subseteq \vec{y}$. Also, $\vec{\mathrm{C}}[\ ]$, $\mathrm{C}'[\ ]$, $\vec{\mathrm{C}}_1[\ ]$, and $\vec{\mathrm{C}}_m[\ ]$ are sequences of contexts in $T(\mathcal{F}_{i-1},\mathcal{X})$, and $\vec{\mathrm{C}}[\vec{x}\,] \rhd_{mul} \vec{\mathrm{C}}_j[\vec{x}\,]$ $(1 \le j \le m)$, where $\lhd$ is the strict subterm ordering and *mul* denotes multiset extension.

The rules defining $F^1, \ldots, F^n$ and their Curry-closure together form a *safe recursive system*.

This general scheme imposes some restrictions on the definition of functions: the terms in every $\vec{\mathrm{C}}[\vec{x}\,]$ are subterms of terms in $\vec{\mathrm{C}}[\vec{x}\,]$ (this is the 'primitive recursive' aspect of the scheme), and the variables $\vec{x}$ must also appear as arguments in the left-hand side of the rule.

It is worthwhile noting that the rewrite rules of Combinatory Logic are *not* recursive, so, in particular, satisfy the scheme. Therefore, although the severe restriction imposed on rewrite rules, the systems satisfying the scheme still have full Turing-machine computational power, a property that first-order systems without Ap would not possess.

In a type assignment system without ω, the conditions imposed by the general scheme are sufficient to ensure strong normalization of typeable terms [4]. Unfortunately, the general scheme is not enough to ensure head normalization of typeable terms in a type system with ω: take the rewrite system

$$\begin{aligned} F(C(x)) &\to F(x),\\ A(x,y) &\to Ap(y, Ap(Ap(x,x),y)) \end{aligned}$$

that is typeable with respect to the environment

$$\begin{aligned} &\mathcal{E}(F) = \omega\to\sigma,\\ &\mathcal{E}(C) = \omega\to\sigma,\\ &\mathcal{E}(A) = ((\alpha\to\mu\to\beta)\cap\alpha)\to((\beta\to\rho)\cap\mu)\to\rho, \end{aligned}$$

Then $B \vdash_{\mathcal{E}} F(A(A_0,C_0)):\sigma$, but $F(A(A_0,C_0)) \to^*_{\mathbf{R}} F(C(A(A_0,C_0))) \to_{\mathbf{R}} F(A(A_0,C_0))$. The underlying problem is that, using full intersection types, there are two kinds of typeable recursion in CTRS: the one explicitly present in the syntax, as well as the one obtained by the so-called *fixed-point combinators*; for every G that has type $\omega\to\sigma$, the term $A(A_0,G_0)$ has type σ, and $A(A_0,G_0) \to^*_{\mathbf{R}} G(A(A_0,G_0))$.

So, we need to impose stronger conditions on the scheme. We will consider those CTRS having an alphabet with a set $\mathcal{C}$ of constructors, such that constructors are given arrow-ground types in all the environments, i.e. for all environment $\mathcal{E}$, if $C \in \mathcal{C}$ then $\mathcal{E}(C) = s_1\to\ldots\to s_n\to s$ where $s_1,\ldots,s_n,s$ are sorts.

Definition 3.2 (Safety-scheme) A rewrite rule

$$F^i(\vec{C}[\vec{x}],\vec{y}) \to C'[F^i(\vec{C_1}[\vec{x}],\vec{y}),\ldots,F^i(\vec{C_m}[\vec{x}],\vec{y}),\vec{x},\vec{y}]$$

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

'for $1\le j\le m$, $\vec{C}[\vec{x}] \rhd_{mul} \vec{C_j}[\vec{x}]$'

by the condition:

'for $1\le j\le m$, $\vec{C}[\vec{x}] \rhd_{mul} \vec{C_j}[\vec{x}]$, $\vec{C}[\vec{x}]$, $\vec{C_j}[\vec{x}] \in T(\mathcal{C},\mathcal{X})$, and the "patterns" $\vec{C}[\vec{x}]$ appear at positions where $\mathcal{E}(F^i)$ requires arguments of sort type'

From now on, we will call the systems that satisfy the Safety-scheme *safe*.

The rest of this section will be devoted to the proof of the head normalization theorem. We will use the well-known method of Computability Predicates [13, 10]. We will prove simultaneously that every typeable term is head normalizable and constructor-hat normalizable. The proof has two parts; in the first one we give the definition of a 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 $\sigma\neq\omega$ such that $Comp(B,t,\sigma)$ holds, then $HN(t)$ and $CHN(t)$. In the second part *Comp* is shown to hold for each typeable term.

In the following we assume that $\mathcal{G}$TRS is a typeable and safe CTRS in the environment $\mathcal{E}$.

Definition 3.3 *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) $\sigma = \varphi$, or $\sigma = s$ (sort).
$Comp(B,t,\sigma) \iff HN(t)$ & $CHN(t)$.

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

c) $\sigma = \sigma_1\cap\cdots\cap\sigma_n$ $(n \ge 0)$.
$Comp(B,t,\sigma_1\cap\cdots\cap\sigma_n) \iff \forall\, 1\le i\le n$ [$Comp(B,t,\sigma_i)$].

ii) We say that a replacement 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 $Comp\,(B,t,\omega)$ holds as special case of part (*i.c*). Also, since we use intersection types, and because of Definition 2.2, in part (*ii*) we need not consider the existence of different bases for each $x{:}\sigma \in B$.

Property 3.4 *C1)* $Comp\,(B,t,\sigma)\ \&\ \sigma \neq \omega\ \Rightarrow\ HN(t)\ \&\ CHN\,(t)$.

C2) If $Comp\,(B,t,\sigma)$, *and* $t \to^*_{\mathbf{R}} t'$, *then* $Comp\,(B,t',\sigma)$.

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* $Comp\,(B,t,\sigma)$.

C4) 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 3.3. (*i.a*), using Theorem 2.8 and the Church-Rosser property for *C2*, and Theorem 2.8 for *C3* and *C4*.

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

C1) $Comp\,(B,t,\alpha{\to}\beta)$ & x does not occur in $B\ \Rightarrow$ (*IH.C4*)
$Comp\,(B,t,\alpha{\to}\beta)$ & $Comp\,(\{x{:}\alpha\},x,\alpha)\ \Rightarrow$ (*3.3. (i.b)*)
$Comp\,(B \cup \{x{:}\alpha\}, Ap\,(t,x),\beta)\ \Rightarrow$ (*IH.C1*)
$HN\,(Ap\,(t,x))$ & $CHN\,(Ap\,(t,x))\ \Rightarrow$ (*1.10.(i)*)
$HN\,(t)$ & $CHN\,(t)$.

C2) $Comp\,(B,t,\alpha{\to}\beta)\ \Rightarrow$ (*3.3. (i.b)*)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\}, Ap\,(t,u),\beta)\ )\ \Rightarrow$ (*IH.C2*)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\}, Ap\,(t',u),\beta)\ )\ \Rightarrow$ (*3.3. (i.b)*)
$Comp\,(B,t',\alpha{\to}\beta)$.

C3) t is neutral & $B \vdash_{\mathcal{E}} t{:}\alpha{\to}\beta\ \&\ \exists\, v\ [t \to^*_{\mathbf{R}} v\ \&\ Comp\,(B,v,\alpha{\to}\beta)]\ \Rightarrow$
(*3.3. (i.b)*)
$(Comp\,(B',u,\alpha)\ \Rightarrow\ \exists\, v[t \to^*_{\mathbf{R}} v\ \&\ Comp\,(\Pi\{B,B'\}, Ap(v,u),\beta)])\ \Rightarrow$
(*1.10.(iii)*)
$(Comp\,(B',u,\alpha)\ \Rightarrow\ \exists\, v[Ap(t,u) \to^*_{\mathbf{R}} v\ \&\ Comp\,(\Pi\{B,B'\}, v,\beta)])\ \Rightarrow$
(*IH.C3*)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\}, Ap\,(t,u),\beta)\ )\ \Rightarrow$ (*3.3. (i.b)*)
$Comp\,(B,t,\alpha{\to}\beta)$.

C4) t is neutral & $B \vdash_{\mathcal{E}} t{:}\alpha{\to}\beta$ & $In\text{-}Hnf\,(t)$ & $In\text{-}Chnf\,(t)\ \Rightarrow$ (*1.10.(ii)*)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Ap\,(t,u)$ neutral & $\Pi\{B,B'\} \vdash_{\mathcal{E}} Ap\,(t,u){:}\beta$ &
$In\text{-}Hnf\,(Ap\,(t,u))$ & $In\text{-}Chnf\,(Ap\,(t,u))\ )\ \Rightarrow$ (*IH.C4*)
$(\ Comp\,(B',u,\alpha)\ \Rightarrow\ Comp\,(\Pi\{B,B'\}, Ap\,(t,u),\beta)\ )\ \Rightarrow$ (*3.3. (i.b)*)
$Comp\,(B,t,\alpha{\to}\beta)$.

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

C1) $Comp\,(B,t,\sigma_1 \cap \cdots \cap \sigma_n)\ \Rightarrow$ (*3.3. (i.c)*)
$\forall\, 1 \le i \le n\ [Comp\,(B,t,\sigma_i)]\ \Rightarrow$ (*IH.C1 & $n \neq 0$*)
$HN\,(t)$ & $CHN\,(t)$.

C2) $Comp\,(B,t,\sigma_1 \cap \cdots \cap \sigma_n)\ \&\ t \to^* t'\ \Rightarrow$ (*3.3. (i.c)*)
$\forall\, 1 \le i \le n\ [Comp\,(B,t,\sigma_i)]\ \&\ t \to^* t'\ \Rightarrow$ (*IH.C2*)
$\forall\, 1 \le i \le n\ [Comp\,(B,t',\sigma_i)]\ \Rightarrow$ (*3.3. (i.c)*)
$Comp\,(B,t',\sigma_1 \cap \cdots \cap \sigma_n)$.

C3) t is neutral & $B \vdash_{\mathcal{E}} t{:}\sigma_1 \cap \cdots \cap \sigma_n\ \&\ \exists\, v\ [t \to_{\mathbf{R}} v\ \&$
$Comp\,(B,v,\sigma_1 \cap \cdots \cap \sigma_n)]\ \Rightarrow$ (*2.5.(iv) & 3.3. (i.c)*)
$\exists\, v\ [t \to_{\mathbf{R}} v\ \&\ \forall\, 1 \le i \le n\ [Comp\,(B,v,\sigma_i)\ \&\ B \vdash_{\mathcal{E}} t{:}\sigma_i]]\ \Rightarrow$ (*IH.C3*)

$\forall\, 1 \le i \le n\ [Comp(B,t,\sigma_i)] \Rightarrow$ (*3.3. (i.c)*)
$Comp(B,t,\sigma_1 \cap \cdots \cap \sigma_n)$.

C4) t is neutral & $B \vdash_{\mathcal{E}} t{:}\sigma_1 \cap \cdots \cap \sigma_n$ & $In\text{-}Hnf(t)$ & $In\text{-}Chnf(t) \Rightarrow$ (*2.5.(iv)*)
t is neutral & $\forall\, 1 \le i \le n\ [B \vdash_{\mathcal{E}} t{:}\sigma_i$ & $In\text{-}Hnf(t)$ & $In\text{-}Chnf(t)] \Rightarrow$ (*IH.C4*)
$\forall\, 1 \le i \le n\ [Comp(B,t,\sigma_i)] \Rightarrow$ (*3.3. (i.c)*)
$Comp(B,t,\sigma_1 \cap \cdots \cap \sigma_n)$.

In order to prove the head normalization theorem we shall prove a stronger property, for which we will need the following ordering and lemma.

Definition 3.5 Let $\mathcal{G}$iTRS be a CTRS.

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

ii) We define the ordering $\gg$ on triples – consisting of a pair of natural numbers, a term, and a multiset of terms – as the object $((>_{\mathbb{N}},>_{\mathbb{N}})_{lex},\rhd\ ,(\ \rightarrow_{Chnf} \cup \rhd_{Chnf})_{mul})_{lex}$, where
   a) $t \rightarrow_{Chnf} t'$ if $t \rightarrow^*_{\mathbf{R}} t'$, $\neg In-Chnf(t)$ and $In\text{-}Chnf(t')$,
   b) $t \rhd_{Chnf} t'$ if $t \rhd t'$, $In\text{-}Chnf(t)$, $In\text{-}Chnf(t')$.

iii) For computable R, we interpret a term t^{R} by the triple $\mathcal{I}(t^{\mathrm{R}}) = [\mathrm{(i,j)},\mathrm{t},\{R\}]$, where
   a) i is the maximal super-index of the function symbols belonging to t,
   b) j is the minimum of the differences $arity(F^i) - arity(F^i_k)$ such that F^i_k occurs in t,
   c) {R} is the multiset $\{x^{\mathrm{R}} \mid x \in Var\,(\mathrm{t})$ & *the type of x in t is not* $\omega\}$.
   These triples are compared in the ordering $\gg$, which is well-founded because R is computable, and we are taking the elements of its image that have a type different from ω.

Lemma 3.6 $Comp(B,t,\sigma)$, $\sigma \le \rho \Rightarrow Comp(B,t,\rho)$.

We now come to the main theorem of this section, in which we show that for any typeable term and computable replacement R such that the term t^{R} is typeable, t^{R} is computable.

Property 3.7 Let t be a term such that $B \vdash_{\mathcal{E}} t{:}\sigma$, and R *a computable replacement in B. Then there exists B' such that $Comp(B',t^{\mathrm{R}},\sigma)$.*

Proof: We will prove that t^{R} is computable or reduces to a computable term, by noetherian induction on $\gg$:

If t is a variable then t^{R} is computable by Lemma 3.6, since R is. Also, if σ is ω then t^{R} is trivially computable. Then, without loss of generality we assume that t is not a variable and that $\sigma \neq \omega$. Also, because of part (*i.c*) of Definition 3.3, we can restrict the proof to the case $\sigma \in \mathcal{T}_s$. We consider separatedly the cases:

i) t^{R} is neutral.
   a) If $In\text{-}Hnf(t^{\mathrm{R}})$ and $In\text{-}Chnf(t^{\mathrm{R}})$ then t^{R} is computable by *C4*.
   b) If $In\text{-}Hnf(t^{\mathrm{R}})$ but not $In\text{-}Chnf(t^{\mathrm{R}})$, then $t^{\mathrm{R}} = C(t_1,\ldots,t_n)$ where $C \in \mathcal{C}$ (constructor) or $t^{\mathrm{R}} = Ap(t_1,t_2)$ (in other case we would have $In\text{-}Chnf(t^{\mathrm{R}})$). For $1 \le i \le n$, t_i is computable, either because it is in R or by induction. In case $t^{\mathrm{R}} = C(t_1,\ldots,t_n)$, $t_1,\ldots,t_n$ have a type different from ω, then they have a constructor-hat normal form by *C1*, which implies $CHN(t^{\mathrm{R}})$. In case $t^{\mathrm{R}} = Ap(t_1,t_2)$, t_1 has a type different from ω, then by *C1*, $CHN(t_1)$, which implies $CHN(t^{\mathrm{R}})$. In both cases, t^{R} reduces to a neutral term

t' such that *In-Chnf* (t') and *In-Hnf* (t'). By *C3*, t' is computable, and again by *C3*, t^{R} is computable.

c) If not *In-Hnf* (t^{R}), then there exists v such that $t^{\mathrm{R}} \to^*_{\mathbf{R}} v$ and v is itself a redex or $v = Ap(v_1, v_2)$ and v_1 is a redex. Without loss of generality, we can assume that $t^{\mathrm{R}} \to^*_{\mathbf{R}} v$ is the shortest derivation that satisfies this condition, then v can be decomposed into a non-variable term $u = F(z_1, \ldots, z_n)$ $(F \in \mathcal{F} \cup \{Ap\})$ and a computable replacement R', i.e. $v = u^{\mathrm{R}'}$. Note that R' is computable by induction and *C2*, since none of the reduction steps in the derivation $t^{\mathrm{R}} \to^*_{\mathbf{R}} v$ takes place at the root position (because it has minimal length).

1) If $t \neq u$ modulo renaming of variables, then $\mathcal{I}(t^{\mathrm{R}}) \gg_2 \mathcal{I}(u^{\mathrm{R}'})$, then by induction, $u^{\mathrm{R}'}$ is computable and so is t^{R} by *C3*.

2) If $t = u$ modulo renaming of variables (without loss of generality, we can assume $t = u$); since t^{R} is neutral, v is neutral too, then only two cases are possible for u:

A) $t = u = Ap(z_1, z_2)$.
In this case, $t^{\mathrm{R}} \to^*_{\mathbf{R}} v = Ap(z_1, z_2)^{\mathrm{R}'}$. By the Subject Reduction Theorem, v has a type different from ω, then $z_1{}^{\mathrm{R}'}$ must have an arrow type, and since R' is computable, v is computable by 3.3. (*i.b*). Then t^{R} is computable by *C3*.

B) $t = u = F^k(z_1, \ldots, z_n)$, and $v = t^{\mathrm{R}'}$ is reducible at the root position. Then, there is a rewrite rule
$$F^k(\vec{\mathrm{C}}[\vec{x}], \vec{y}) \to \mathrm{C}'[F^k(\vec{\mathrm{C}}_1[\vec{x}], \vec{y}), \ldots, F^k(\vec{\mathrm{C}}_n[\vec{x}], \vec{y}), \vec{x}, \vec{y}]$$
such that $t^{\mathrm{R}} \to^* v = t^{\mathrm{R}'} = F^k(\vec{\mathrm{C}}[\vec{M}], \vec{N})$. By definition of safe CTRS, the patterns $\vec{\mathrm{C}}[\vec{x}]$ are constructor terms, and the terms $\vec{M}$ have sorts as types. Also, since $\vec{x} \subseteq \vec{y}$, we know that $\vec{M} \subseteq \vec{N}$, and then $\vec{M}$ are computable because $\vec{N}$ are (they are in R'). Then $CHN(\vec{M})$. Let R'' be the computable replacement obtained from R' by putting $\vec{M}$ in CHNF (note that R'' is computable by *C2*). There are two possible cases: either $\mathcal{I}(t^{\mathrm{R}}) \gg_3 \mathcal{I}(t^{R''})$, and then $t^{R''}$ is computable by induction, and so is t^{R} by *C3*, or
$$t^{\mathrm{R}} = t^{R''} = F^k(\vec{\mathrm{C}}[\vec{M}], \vec{N}) \to_{\mathbf{R}}$$
$$\mathrm{C}'[F^k(\vec{\mathrm{C}}_1[\vec{M}], \vec{N}), \ldots, F^k(\vec{\mathrm{C}}_n[\vec{M}], \vec{N}), \vec{M}, \vec{N}].$$
Since $\vec{N}$ is computable (because it is in the image of R) and $\vec{M}$ is computable (because it is a subset of $\vec{N}$), the terms $\vec{\mathrm{C}}_i[\vec{M}]$ are computable by induction (since by definition of the scheme, all function symbols in C_i have a superindex smaller than k). Also, by definition of the scheme, and because *In-Chnf* $(\vec{M})$, $\vec{\mathrm{C}}[\vec{M}]$ $(\rhd_{Chnf})_{mul}$ $\vec{\mathrm{C}}_i[\vec{M}]$, then $F^k(\vec{\mathrm{C}}_i[\vec{M}], \vec{N}], \vec{N})$ is computable by induction. Again, by definition of the scheme and induction, $\mathrm{C}'[F^k(\vec{\mathrm{C}}_1[\vec{M}], \vec{N}), \ldots, F^k(\vec{\mathrm{C}}_n[\vec{M}], \vec{N}), \vec{M}, \vec{N}]$ is computable, since C' does not contain F^k. Then, by *C3*, t^{R} is computable since it is neutral.

ii) t^{R} is not neutral. Then $t = F_i(t_1, \ldots, t_i)$, where F_i is the Curryfied version of some function symbol. In this case, since the type of t is not ω, t must have an arrow type, $\alpha \to \beta$. We have to prove that $Ap(F_i(t_1, \ldots, t_i), z)^{\mathrm{R}'}$ is computable for any replacement $R' = R \cup \{z \mapsto u\}$ such that u is computable of type α. But since $Ap(F_i(t_1, \ldots, t_i), z)^{\mathrm{R}'}$ is a neutral term, it is sufficient to prove that it reduces to a computable term (*C3*). Now, by definition of CTRS, $Ap(F_i(t_1, \ldots, t_i), z)^{\mathrm{R}'} \to_{\mathbf{R}} F_{i+1}(t_1, \ldots, t_i, z)^{\mathrm{R}'}$, and $\mathcal{I}(Ap(F_i(t_1, \ldots, t_i), z)^{\mathrm{R}'})$ $\gg_1 \mathcal{I}(F_{i+1}(t_1, \ldots, t_i, z)^{\mathrm{R}'})$, then $F_{i+1}(t_1, \ldots, t_i, z)^{\mathrm{R}'}$ is computable by induction, and we are finished.

Theorem 3.8 (HEAD NORMALIZATION THEOREM) *If GTRS is typeable in $\vdash_{\mathcal{E}}$, and safe, then every term t such that $B \vdash_{\mathcal{E}} t{:}\sigma$ and $\sigma \neq \omega$ has a head normal form.*

Proof: The theorem follows from Prop.3.7 and *C1*, taking R such that $x^{\mathrm{R}} = x$, which is computable by Prop.*C4*.

3.2 Normalization

In the intersection system for LC, it is well-known that terms that are typeable without ω in base 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:

$$Z(x,y) \rightarrow y,$$
$$D(x) \rightarrow Ap(x,x).$$

The term $Z_1(D(D_0))$ has type $\beta\rightarrow\beta$ in an environment where Z is typed with $\alpha\rightarrow\beta\rightarrow\beta$ and D with $\alpha\cap(\alpha\rightarrow\alpha)\rightarrow\alpha$, but is not normalizable.

We will then only study normalization of non-Curryfied terms in CTRS. Actually, to get a normalization result similar to that of LC we will also need to impose the following condition on CTRS:

Definition 3.9 A CTRS is *complete* if whenever a typeable non-Curryfied term t is reducible at a position p such that $t|_p$ has a type containing ω, t is reducible also at some $q < p$ such that $t|_q$ has a type without ω.

In order to be complete, a rewrite system must have rules that enable a reduction of $F(t_1,\ldots,t_n)$ at the root whenever there is a redex t_i the type of which contains ω. This means that the rules defining F cannot have non-variable patterns that have types with ω, and also that a constructor cannot accept arguments having a type which contains ω.

Constructors and recursive function symbols of safe systems satisfy these conditions. So, a safe recursive system is complete whenever the non-recursive defined symbols do not distinguish patterns that have a type containing ω.

From now on, we will consider only non-Curryfied CTRS that are safe and complete. This section will be devoted to the proof of the normalization theorem. 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 3.10 Let GTRS be a CTRS. 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 $\gg$ on triples composed of a natural number and two terms, as the object $(>_{\mathbb{N}}, \rhd, \succ)_{lex}$.

Theorem 3.11 (NORMALIZATION THEOREM) *Let t be a non-Curryfied term in a typeable, complete, and safe CTRS. If $B \vdash_{\mathcal{E}} t{:}\sigma$ and ω does not appear in σ, then t is normalizable.*

Proof: By noetherian induction on $\gg$. We will interpret the term u by the triple $\mathcal{I}(u)$ = [i,u',u] 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 $\gg$.

Assume t is not in normal form. All its strict subterms that have a type without ω are either already in normal form, or smaller than t with respect to $\gg$ and then normalizable by induction. Let v be the term obtained from t by reducing these subterms to normal form.

If $v \neq t$ then $\mathcal{I}(t) \gg_2 \mathcal{I}(v)$. Then v is normalizable by induction, and so is 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:

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

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

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

that have a type without ω are normalizable by induction. Let $C''[\vec{u}]$ 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, by the induction, it is normalizable.

References

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

[2] S. van Bakel. Partial Intersection Type Assignment of Rank 2 in Applicative Term Rewriting Systems. Technical Report 92-03, Department of Computer Science, University of Nijmegen, 1992.

[3] S. van Bakel. Partial Intersection Type Assignment in Applicative Term Rewriting Systems. In *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications,* Utrecht, the Netherlands, volume 664 of *LNCS*, pages 29–44. Springer-Verlag, 1993.

[4] S. van Bakel and M. Fernández. Strong Normalization of Typeable Rewrite Systems. In *Proceedings of HOA '93. First International Workshop on Higher Order Algebra, Logic and Term Rewriting*, Amsterdam, the Netherlands. *Selected Papers*, volume 816 of *LNCS*, pages 20–39. Springer-Verlag, 1994.

[5] F. Barbanera and M. Fernández. Combining first and higher order rewrite systems with type assignment systems. In *Proceedings of TLCA '93. International Conference on Typed Lambda Calculi and Applications,* Utrecht, the Netherlands, volume 664 of *LNCS*, pages 60–74. Springer-Verlag, 1993.

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

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

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

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

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

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

[12] J.W. Klop. Term Rewriting Systems. In *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–116. Clarendon Press, 1992.

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