Approximation and Normalization Results for Typeable Term Rewriting Systems

(HOA'95, LNCS 1074, pages 17-36, 1996)

Steffen van Bakel and Maribel Fernández

Department of Computing,
Imperial College,
180 Queens Gate, London SW7 2BZ,
svb@doc.ic.ac.uk

DMI - LIENS (CNRS URA 1327)
École Normale Supérieure
45 Rue d'Ulm, 75005 Paris, France
maribel@dmi.ens.fr

Abstract

We consider an intersection type assignment system for term rewriting systems extended with application, and define a notion of (finite) approximation on terms. We then prove that for typeable rewrite systems satisfying a general scheme for recursive definitions, every typeable term has an approximant of the same type. This approximation result, and the proof technique developed to obtain it, allow us to deduce in a direct way a head-normalization, a normalization, and a strong normalization theorem, for different classes of typeable terms.

1 Introduction

Term rewriting systems (TRS) can be seen as a model of computation, as programming or specification languages, or as formulae manipulating systems that can be used in various applications such as program optimization or automated theorem proving. *Confluence* and *termination* (also called *strong normalization*) are the most important properties of these systems: confluence ensures determinacy, whereas termination ensures that all rewrite sequences are finite. For some applications, in particular when TRS are used as a programming language, weaker properties than termination suffice. For instance, during computation *head-normalization* is a useful property: a system is head-normalizing if every term can be reduced to a term which will never reduce at the root.

In this paper we focus on normalization properties of TRS. We consider a class of systems, called *Curryfied Term Rewriting Systems* ($\mathcal{C}$urryTRS), that are first-order TRS with a binary function symbol Ap that allows for *partial application* of function symbols. This function symbol can be used to define Curryfied versions of other function symbols. An advantage of having Ap in the language is that it makes it easy to code higher-order languages, like Combinator Systems and λ-calculus (LC), as $\mathcal{C}$urryTRS (i.e. first-order rewrite systems).

It is well-known that in the study of normalization of reduction systems, the notion of *types* plays an important role. The type system used in this paper, the *intersection type discipline* [10] (see also [8, 9, 2]) is an extension of Curry's type assignment system for LC [11]. The extension consists of that terms and term-variables are (essentially) allowed to have more than one type. Intersection types are constructed by having, in addition to the type constructor '$\rightarrow$', the type constructor '$\cap$' and the type constant 'ω'. Using intersection types a characterization of typeable λ-terms can be given:

- the set of terms typeable without using ω is the set of strongly normalizable terms,
- the set of terms typeable with type σ from a basis B, such that ω does not occur in B and σ, is the set of normalizable terms, and

- the set of terms typeable with type $\sigma \neq \omega$ is the set of terms having a head-normal form.

(see, for instance, [19, 2]). Perhaps less well-known is the fact that the notion of *approximant* can be useful in the study of the relation between typeability and normalization. Intuitively, an approximant of a term is a finite description of its (possibly infinite) normal forms (see, e.g., [23] for a definition of approximants for LC). The aim of this paper is to investigate the relation between normalization and typeability through the use of approximants within the framework of $\mathcal{CA}$TRS.

In [6] and [7] we presented an intersection type assignment system for $\mathcal{CA}$TRS and showed that, provided the rewrite rules are typeable and satisfy a *general scheme of recursion* (inspired by the scheme of Jouannaud and Okada [16]):

- if ω is not used, typeable terms are strongly normalizable,
- non-Curryfied terms (i.e. terms without Curryfied functions and Ap) typeable with type σ from a basis B, such that ω does not occur in B and σ, are normalizable, and
- terms typeable with type $\sigma \neq \omega$ have a head-normal form.

Each of the above properties was proved separately. In this paper we will show that these properties can in fact be derived from one result. To that aim, we first present the formal construction needed to show that any typeable term in a typeable $\mathcal{CA}$TRS has an approximant of the same type (the *approximation theorem*), and then show that all of the above properties can be derived from this result in a straightforward manner. This then confirms our initial claim that the notion of approximant is a useful tool in the study of the relations between typeability and normalization in the rewriting framework.

In order to define approximants of terms in $\mathcal{CA}$TRS, we introduce a special symbol $\perp$ (*bottom*) into our systems, and follow the ideas of Thatte [22], which in turn are based on the definition of Ω-normal forms of Huet and Lévy [15]. As shown by Thatte, it is possible to define a fully abstract model of a term rewriting language by interpreting terms by sets of approximants. Our approximation theorem shows the connections between the intersection type system and the semantics of the language: any typeable term has a "meaning" in the model (i.e. an approximant different from $\perp$). In [1] (see also [17]), Ariola et al. discuss several notions of "undefined" (or "meaningless") terms. It turns out that our meaningless terms (which, as a consequence of the approximation theorem are untypeable) do not have a head-normal form. In the terminology of [1], the set of terms without head-normal form is the set $\mathcal{U}_{rs}$ (terms without a root stable form). Hence, our notion of undefined terms can be seen as an extension to typeable systems of the latter one.

In order to prove the approximation theorem, we define a rewrite relation on type derivations (called *derivation reduction*), and prove that under some restrictions on the use of recursion in the rewrite rules (the general scheme) this relation is strongly normalizing. We use the technique of Computability Predicates for this proof (see, e.g., [14]). We then show that an approximant of t can be obtained simply by replacing untypeable subterms by $\perp$ in the normal form of the derivation for t. The strong normalization of the standard rewrite relation on typeable terms when ω is not used, is an immediate consequence of the strong normalization theorem for derivation reduction, whereas the head-normalization and the normalization theorems are direct consequences of the approximation theorem. These results hold in particular for Combinator Systems, as a particular case of $\mathcal{CA}$TRS that satisfies the required conditions.

This paper is organized as follows: in Section 2 we recall the definition of $\mathcal{CA}$TRS, and in Section 3 the intersection type assignment system for $\mathcal{CA}$TRS. Section 4 deals with derivation reduction and strong normalization of type derivations. In Section 5 we define approximants, prove the approximation theorem, and then the head-normalization, normalization and strong

normalization theorems. Section 6 contains the conclusions and directions for future work.

2 Curryfied Term Rewriting Systems

Roughly speaking, Curryfied Term Rewriting Systems ($\mathcal{C}$TRS) are first-order TRS with a binary function symbol Ap that allows for partial application of function symbols. $\mathcal{C}$TRS extend the function-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. In the following we assume familiarity with the basic notions and notations of term rewriting systems; we refer the reader to the surveys [18, 12] for a detailed account and examples.

We consider a *signature* Σ consisting of a denumerable set $\mathcal{X}$ of *variables*, a finite set $\mathcal{F}$ of *function symbols* (each equipped with an *arity*, which is a natural number), and a binary operator Ap for *application*. The set $T(\mathcal{F},\mathcal{X})$ of *terms* is defined inductively as usual, but from $\mathcal{X}$ and $\mathcal{F}\cup\{Ap\}$. $Var\,(t)$ denotes the set of variables that appear in t.

We will call 'term-substitution' the operation that replaces term-variables by terms (we reserve the word 'substitution' for the operation that replaces 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. The application of the term-substitution R to the term t will be denoted by t^{R}. We will use the notation $\{x_1\mapsto t_1,\ldots,x_n\mapsto t_n\}$ to denote a term-substitution.

Definition 2.1 *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 $T(\mathcal{F},\mathcal{X})$. Often a rewrite rule will get a name, e.g. **r**, and we write $\mathbf{r}: l\to r$. As usual, two conditions are imposed:

a) l is not a variable.

b) The variables occurring in r are contained in l.

If $\mathbf{r}: F(t_1,\ldots,t_n)\to r$ and, for $1\le i\le n$, either t_i is not a variable, or t_i is variable and there is a $1\le i\ne j\le n$ such that $t_i = t_j$, then t_i is called a *pattern* of **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 **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}, ..., F_1, F_0 in $\mathcal{F}$, the *Curryfied-versions of* F, and **R** contains the n rewrite rules:

$$\begin{array}{ll} Ap\,(F_{n-1}\,(x_1,\ldots,x_{n-1}),x_n) & \to\ F\,(x_1,\ldots,x_n)\\ \quad\vdots & \\ Ap\,(F_1\,(x_1),x_2) & \to\ F_2\,(x_1,x_2)\\ Ap\,(F_0,x_1) & \to\ F_1\,(x_1)\end{array}$$

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

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

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

iii) 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 a context C[ ]; this gives rise to *rewrite steps*: $\mathrm{C}[\,l^{\mathrm{R}}\,]\to_{\mathbf{r}}\mathrm{C}[\,r^{\mathrm{R}}\,]$. We 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* $t_0\to$

$t_2 \to \cdots$. If $t_0 \to \cdots \to t_n$ ($n \geq 0$) we also write $t_0 \to^* t_n$, and t_n is a *reduct* of t_0.
Terms that do *not* contain Curryfied versions of function symbols or Ap will be called *non-Curryfied terms*.

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

Example 2.2 *Curryfied Combinatory Logic* (CCL) is the $\mathcal{G}$TRS with function symbols $\mathcal{F} = \{S, S_2, S_1, S_0, K, K_1, K_0, I, I_0\}$, and rewrite rules

$$\begin{array}{ll} S(x,y,z) & \to\ Ap(Ap(x,z),Ap(y,z)) \\ K(x,y) & \to\ x \\ I(x) & \to\ x \end{array}$$

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

Definition 2.3 *i*) A term is in *normal form* if it is irreducible.
ii) 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.
The notions *(head-)normalizable* and *strongly normalizing* are defined as usual.

Our definition of head-normal form is an extension to systems with Ap of the notion of root stable form defined by Ariola et al. [1] (see also [17]). 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.

Example 2.4 Take the $\mathcal{G}$TRS $F(G,H) \to A$, $B(C) \to G$, then $F(B(C),H)$ and $Ap(F(B(C),H))$ are not redexes, but are neither head-normal forms since $F(B(C),H)$ reduces to $F(G,H)$ which is a redex.

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; the set $\mathcal{F}$ of function symbols can be divided into *defined symbols*, and other symbols, the *constructors*.
We will assume that rules are *not* mutually recursive.

Definition 2.5 A symbol F *depends on* G if G occurs in the right-hand side of a rule that defines F. A $\mathcal{G}$TRS whose dependency-graph is an ordered a-cyclic graph, is called *hierarchical*.

The rewrite rules of a hierarchical $\mathcal{G}$TRS can be regrouped in such a way that they are *incremental* definitions of the defined symbols F^1, ..., F^k, so that the rules defining F^i only depend on $F^1, \ldots, F^{i-1}$.

Example 2.6 Since Ap is never a defined symbol, $D(x) \to Ap(x,x)$ 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 $\mathcal{G}$TRS that are not normalizing.

3 Type assignment in $\mathcal{G}\iota$TRS

In this section we recall a variant of the intersection type assignment system for $\mathcal{G}\iota$TRS presented in [3]. This notion of type assignment is partial in the sense of [20]: not only will we define how terms and rewrite rules can be typed, but we will also assume that every function symbol already *has* a type, provided by an *environment* (i.e. a mapping from function symbols to types).

3.1 Bases, Types, and Operations

We consider a set of *sorts* (names of domains), the constant types of our system. When sorts are not taken into account, the *strict intersection types* defined below are the representatives for equivalence classes of the types considered in the system of [8]. We will define the type ω as an intersection of zero types: if $n = 0$, then $\sigma_1 \cap \cdots \cap \sigma_n = \omega$.

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) *1*) All type-variables $\varphi_0, \varphi_1, \ldots \in \mathcal{T}_s$.
2) All sorts $s_1, s_2, \ldots \in \mathcal{T}_s$.
3) If $\tau \in \mathcal{T}_s$ and $\sigma \in \mathcal{T}_S$, then $\sigma \to \tau \in \mathcal{T}_s$.

b) 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) 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) $\sigma \leq_S \tau \leq_S \rho \Rightarrow \sigma \leq_S \rho$.

iii) We define the relation $\leq$ on $\mathcal{T}_S$ like the relation $\leq_S$, but we add an extra alternative:

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

iv) On $\mathcal{T}_S$, the relation $\sim$ is defined by: $\sigma \sim \tau \Leftrightarrow \sigma \leq \tau \leq \sigma$.

$\mathcal{T}_S$ will be considered modulo $\sim$, then $\leq$ becomes a partial order.

Notice that ω does not occur in an intersection subtype. Moreover, intersection types (so also ω) occur in strict types only as subtypes at the left-hand side of an arrow type. In general, 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. Notice also that $\mathcal{T}_s$ is a proper subset of $\mathcal{T}_S$.

A *statement* is an expression of the form $t{:}\sigma$, where $t \in T(\mathcal{F}, \mathcal{X})$ and $\sigma \in \mathcal{T}_S$. t is the *subject* and σ the *predicate* of $t{:}\sigma$. 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 $\sim$ are extended to bases by: $B \leq B' \Leftrightarrow \forall\, x{:}\sigma' \in B'\ \exists\, x{:}\sigma \in B\ [\sigma \leq \sigma']$, and $B \sim B' \Leftrightarrow B \leq B' \leq B$.

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 whose subject is x that occur in $B_1 \cup \ldots \cup B_n$. 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.

We will use three different operations on types (that extend to pairs of $\langle basis, type \rangle$), namely *substitution*, *expansion*, and *lifting*. These were first defined in [4] to show that the strict type assignment system of [2] has the principal type property. 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 3.2 A *chain Ch* is a sequence $[O_1,\ldots,O_n]$, where each O_i is an operation of expansion, substitution, or lifting, and

$$[O_1,\ldots,O_n]\,(\langle B,\sigma\rangle) = O_n\,(\cdots(O_1\,(\langle B,\sigma\rangle))\cdots).$$

3.2 Intersection type assignment in $\mathcal{G}$TRS

Definition 3.3 Let $(\Sigma,\mathbf{R})$ be a $\mathcal{G}$TRS. 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, $\mathcal{E}\,(F) = \mathcal{E}\,(F_{n-1}) = \cdots = \mathcal{E}(F_0)$. Since $\mathcal{E}$ maps all $F\in\mathcal{F}$ to types in $\mathcal{T}_s$, no function symbol is mapped to ω.

The notion of type assignment on $\mathcal{G}$TRS will be defined in two stages. In the next definition we consider type assignment on terms, in Def. 3.6 we will define type assignment on term rewrite rules.

Definition 3.4 *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 ($\leq$), and $\sigma_1,\ldots,\sigma_n$ in rule ($\to$E)):

$$(\leq):\ \frac{x{:}\sigma\in B \quad \sigma\leq\tau}{B\vdash_{\mathcal{E}} x{:}\tau} \qquad (\cap\mathrm{I}):\ \frac{B\vdash_{\mathcal{E}} t{:}\sigma_1 \quad\ldots\quad B\vdash_{\mathcal{E}} t{:}\sigma_n}{B\vdash_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n}\ (n\geq 0)$$

$$(\to\mathrm{E}):\ \frac{B\vdash_{\mathcal{E}} t_1{:}\sigma_1 \quad\ldots\quad B\vdash_{\mathcal{E}} t_n{:}\sigma_n}{B\vdash_{\mathcal{E}} F(t_1,\ldots,t_n){:}\sigma}\ (\exists Ch\ [Ch\,(\mathcal{E}(F)) = \sigma_1\to\cdots\to\sigma_n\to\sigma])$$

ii) We write $B\vdash_{\mathcal{E}} t{:}\sigma$ if and only if there is a derivation that has $B\vdash_{\mathcal{E}} t{:}\sigma$ as conclusion. A derivation D for $B\vdash_{\mathcal{E}} t{:}\sigma$ will be denoted by $\mathrm{D}::B\vdash_{\mathcal{E}} t{:}\sigma$ (or simply D when the conclusion $B\vdash_{\mathcal{E}} t{:}\sigma$ is clear from the context).

Notice that, by ($\cap$I), for every B and t, $B\vdash_{\mathcal{E}} t{:}\omega$. However, we will limit the expression *typeable terms* to those terms that have a type different from ω.

Note that the types that can be assigned to occurrences of function symbols or Ap are obtained from the type provided by the environment by using a chain of operations. The use of an environment in rule ($\to$E) introduces a notion of *polymorphism* into our system.

To guarantee the subject reduction property, as shown in [3], it is sufficient to define type assignment on rewrite rules using a notion of principal pairs.

Definition 3.5 $\langle P,\pi\rangle$ is called *a principal pair for t with respect to* $\mathcal{E}$, if $P\vdash_{\mathcal{E}} t{:}\pi$ and, if $B\vdash_{\mathcal{E}} t{:}\sigma$, there is a chain Ch such that $Ch\,(\langle P,\pi\rangle) = \langle B,\sigma\rangle$.

Definition 3.6 Let $(\Sigma,\mathbf{R})$ be a $\mathcal{G}$TRS, and $\mathcal{E}$ an environment. 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 l and r such that:

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

ii) 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 $\mathcal{E}\,(F)$.

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

Theorem 3.7 Subject Reduction. ([3]) *Let* $(\Sigma, \mathbf{R})$ *be a typeable G*TRS *with respect to an environment* $\mathcal{E}$. *If* $B \vdash_{\mathcal{E}} t{:}\sigma$ *and* $t \to^*_{\mathbf{R}} t'$, *then* $B \vdash_{\mathcal{E}} t'{:}\sigma$.

The type assignment system we have presented is undecidable, but its restriction to intersection types of Rank 2 is decidable, as shown in [5]. The properties we will show in the following sections hold also in the decidable Rank 2 system.

4 Strong normalization of derivation reduction

In this section, we will define the notion of reduction on derivations (here called *derivation reduction*) as a generalization of cut-elimination: reductions on a derivation $\mathrm{D}:: B \vdash'_{\mathcal{E}} t{:}\sigma$ will follow standard reduction, by contracting redexes that have a type different from ω in D, and replacing the derivation for the redex by the derivation for the right-hand side of the rewrite rule. Derivation reduction has an interesting property: it is strongly normalizing when the rewrite rules are typeable and satisfy certain syntactical conditions (a general scheme of recursion). This property will be used in the following section to prove the approximation theorem (for the full system), as well as to deduce the properties of head-normalization, normalization, and strong normalization of terms according to the set of assignable types.

To prove strong normalization of derivation reduction, however, we will restrict the definition of type assignment: for technical reasons, the derivation rule $(\leq)$ of Def. 3.4 is too powerful, and we will limit its use to that of a rule $(\cap\mathrm{E})$, thus defining a notion $\vdash'_{\mathcal{E}}$.

Definition 4.1 *i*) *Strict type assignment* and *strict derivations* are defined by the following natural deduction system (where all types displayed are in $\mathcal{T}_s$, except for $\sigma_1,\ldots,\sigma_n$ in rule $(\to\mathrm{E})$):

$$(\cap\mathrm{E}):\ \frac{x{:}\sigma_1\cap\cdots\cap\sigma_n \in B}{B \vdash'_{\mathcal{E}} x{:}\sigma_i}\ (1\leq i\leq n) \qquad (\cap\mathrm{I}):\ \frac{B \vdash'_{\mathcal{E}} t{:}\sigma_1 \quad \ldots \quad B \vdash'_{\mathcal{E}} t{:}\sigma_n}{B \vdash'_{\mathcal{E}} t{:}\sigma_1\cap\cdots\cap\sigma_n}\ (n \geq 0)$$

$$(\to\mathrm{E}):\ \frac{B \vdash'_{\mathcal{E}} t_1{:}\sigma_1 \quad \ldots \quad B \vdash'_{\mathcal{E}} t_n{:}\sigma_n}{B \vdash'_{\mathcal{E}} F(t_1,\ldots,t_n){:}\sigma}\ (\exists Ch\ [Ch\ (\mathcal{E}(F)) = \sigma_1\to\cdots\to\sigma_n\to\sigma])$$

ii) We write $B \vdash'_{\mathcal{E}} t{:}\sigma$ if and only if there is a derivation that has $B \vdash'_{\mathcal{E}} t{:}\sigma$ as conclusion. A derivation D for $B \vdash'_{\mathcal{E}} t{:}\sigma$ will be denoted by $\mathrm{D}:: B \vdash'_{\mathcal{E}} t{:}\sigma$, or simply D if the conclusion $B \vdash'_{\mathcal{E}} t{:}\sigma$ is clear from the context.

The relation between the two notions of type assigment is formulated by:

Lemma 4.2 i) If $B \vdash_{\mathcal{E}} t{:}\sigma$, *then there is a* B' *such that* $B \leq B'$, *and* $B' \vdash'_{\mathcal{E}} t{:}\sigma$.
ii) If $B \vdash'_{\mathcal{E}} t{:}\sigma$, *and* B' *is such that* $B' \leq B$, *then* $B' \vdash_{\mathcal{E}} t{:}\sigma$.

Rewriting preserves types also in the strict type assignment system:

Theorem 4.3 Strict Subject Reduction. *Let* $(\Sigma, \mathbf{R})$ *be a typeable G*TRS *with respect to an environment* $\mathcal{E}$. *If* $B \vdash'_{\mathcal{E}} t{:}\sigma$, *and* $t \to^*_{\mathbf{R}} t'$, *then* $B \vdash'_{\mathcal{E}} t'{:}\sigma$.

Definition 4.4 We will use a short-hand notation for strict derivations.

i) We write $\langle\cap\mathrm{E}\rangle :: B \vdash'_{\mathcal{E}} x{:}\sigma$ to denote the derivation of $B \vdash'_{\mathcal{E}} x{:}\sigma$ that consists of nothing but an application of rule $(\cap\mathrm{E})$.

ii) We write $\mathrm{D} = \langle \mathrm{D}_1,\ldots,\mathrm{D}_n,\cap\mathrm{I}\rangle$, if there are t, $\sigma_1,\ldots,\sigma_n$ such that $\mathrm{D}_i :: B \vdash'_{\mathcal{E}} t{:}\sigma_i$, for every $1 \le i \le n$, and D is obtained from $\mathrm{D}_1, \ldots, \mathrm{D}_n$ by applying rule ($\cap$I).

iii) We write $\mathrm{D} = \langle \mathrm{D}_1,\ldots,\mathrm{D}_n,\to\mathrm{E}\rangle$, if there are $F \in \mathcal{F} \cup \{Ap\}$, $t_1,\ldots,t_n$, and $\sigma_1,\ldots,\sigma_n$ such that $\mathrm{D}_i :: B \vdash'_{\mathcal{E}} t_i{:}\sigma_i$, for every $1 \le i \le n$, and D is obtained from $\mathrm{D}_1, \ldots, \mathrm{D}_n$ by applying rule ($\to$E).

We can prove the following properties, that are needed further on:

Lemma 4.5 *i) If* $\mathrm{D} :: B \vdash'_{\mathcal{E}} x{:}\tau$, *then there is* $x{:}\sigma \in B$ *such that* $\sigma \le_{\mathrm{S}} \tau$.
ii) $B \vdash'_{\mathcal{E}} F_n(t_1,\ldots,t_n){:}\sigma \ \&\ \sigma \in \mathcal{T}_{\mathrm{s}} \Rightarrow \exists\, \alpha \in \mathcal{T}_{\mathrm{S}}, \beta \in \mathcal{T}_{\mathrm{s}}\, [\sigma = \alpha{\to}\beta]$.

In order to define derivation reduction, we need a notion of substitution on derivations (for lack of space, we do not give a formal definition):

Definition 4.6 Let $B' = \{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\}$, $\mathrm{D} :: B' \vdash'_{\mathcal{E}} t{:}\tau$, and let R, B be such that, for every $1 \le i \le n$, there is a $\mathrm{D}_i :: B \vdash'_{\mathcal{E}} x_i{}^{\mathrm{R}}{:}\sigma_i$. Each leaf of the form $x_i{:}\sigma_i \in B'$ that appears in D is the premise for a ($\cap$E)-rule. More precisely, for each of those leaves there is a subderivation

$$\mathrm{D}_{i,j} :: \quad \frac{x_i{:}\sigma_i \in B'}{B' \vdash'_{\mathcal{E}} x_i{:}\rho^i_j}\ (\cap\mathrm{E})$$

in D, where, by Lem. 4.5-(*i*), $\sigma_i \le_{\mathrm{S}} \rho^i_j$. Now, there are two possibilities: either $\sigma_i = \rho^i_j$, or $\sigma_i = \rho^i_1 \cap \cdots \cap \rho^i_{m_i}$ and hence the last step in the derivation D_i has to be ($\cap$I), i.e. D_i is of the shape:

$$\frac{\begin{array}{cc}\overset{\mathrm{D}^i_1}{\vdots} & \overset{\mathrm{D}^i_{m_i}}{\vdots} \\ B \vdash'_{\mathcal{E}} x_i{}^{\mathrm{R}}{:}\rho^i_1 & \cdots \quad B \vdash'_{\mathcal{E}} x_i{}^{\mathrm{R}}{:}\rho^i_{m_i}\end{array}}{B \vdash'_{\mathcal{E}} x_i{}^{\mathrm{R}}{:}\sigma_i}\ (\cap\mathrm{I})$$

The derivation $\mathrm{D}[\mathrm{D}_i / x_i{:}\sigma_i]$ is defined as the derivation obtained from D by replacing all occurrences of $\mathrm{D}_{i,j}$ such that $\sigma_i = \rho^i_j$ by D_i, and the others by the corresponding D^i_j, and making in t the corresponding replacement of x_i by $x_i{}^{\mathrm{R}}$.

We now give the definition of reduction on derivations.

Definition 4.7 The *derivation reduction* relation, denoted by $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma \ \to_{\mathcal{D}}\ \mathrm{D}' :: B \vdash'_{\mathcal{E}} t'{:}\sigma$, is defined as follows: Suppose there is a rewrite rule $l \to r$ where $Var\,(l) = \{x_1,\ldots,x_n\}$, and a subterm of t at position p (denoted $t|_p$) such that: $t|_p = l^{\mathrm{R}} = F(t_1,\ldots,t_m)$ where $\mathrm{R} = \{x_1 \mapsto u_1,\ldots,x_n \mapsto u_n\}$. Assume moreover that for $t|_p$, D contains at least one subderivation D_0 of the form:

$$\frac{\begin{array}{cc}\overset{\mathrm{D}_1}{\vdots} & \overset{\mathrm{D}_n}{\vdots} \\ B \vdash'_{\mathcal{E}} u_1{:}\sigma_1 & \cdots \quad B \vdash'_{\mathcal{E}} u_n{:}\sigma_n\end{array}}{\begin{array}{c}\overset{\mathrm{D}_0}{\vdots} \\ B \vdash'_{\mathcal{E}} F(t_1,\ldots,t_m){:}\tau\end{array}}$$

such that $\tau \neq \omega$, and the root of D_0 is the first occurrence of the statement $t|_p{:}\tau$ in a path from the root of D to a leaf. Then by the Strict Subject Reduction Theorem (Thm. 4.3), there exists $\mathrm{D}'_0 :: \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\} \vdash'_{\mathcal{E}} r{:}\tau$. Let D' be obtained from D by replacing each subderivation D_0 of $t|_p$ satisfying the previous conditions by the corresponding

$$\mathrm{D}'_0\,[\mathrm{D}_1 / x_1{:}\sigma_1, \ldots, \mathrm{D}_n / x_n{:}\sigma_n] :: B \vdash'_{\mathcal{E}} r^{\mathrm{R}}{:}\tau,$$

and propagating the replacement of $t|_p$ by r^{R} along all the derivation tree. Let t' be obtained from t by replacing $t|_p$ with r^{R}. Then we write $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma \;\to_{\mathcal{D}}\; \mathrm{D}' :: B \vdash'_{\mathcal{E}} t'{:}\sigma$.

The reflexive and transitive closure of $\to_{\mathcal{D}}$ is denoted by $\to^*_{\mathcal{D}}$, and we write $SN(\mathrm{D})$ to indicate that D is strongly normalizable with respect to $\to_{\mathcal{D}}$.

Lemma 4.8 *i) If* $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma \;\to_{\mathcal{D}}\; \mathrm{D}' :: B \vdash'_{\mathcal{E}} t'{:}\sigma$, *then* $t \to t'$.

ii) Let $\mathrm{D} :: B \vdash'_{\mathcal{E}} Ap(t,u){:}\sigma = \langle D_1, D_2, {\to}\mathrm{E}\rangle$, *where* $\mathrm{D}_1 :: B \vdash'_{\mathcal{E}} t{:}\tau{\to}\sigma$ *and* $\mathrm{D}_2 :: B \vdash'_{\mathcal{E}} u{:}\tau$, *then:* $SN(\mathrm{D}) \;\Rightarrow\; SN(\mathrm{D}_1)$ *&* $SN(\mathrm{D}_2)$.

iii) Let $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma_1{\cap}\sigma_2 = \langle D_1, D_2, {\cap}\mathrm{I}\rangle$, *where* $\mathrm{D}_1 :: B \vdash'_{\mathcal{E}} t{:}\sigma_1$ *and* $\mathrm{D}_2 :: B \vdash'_{\mathcal{E}} t{:}\sigma_2$. *Then* $\mathrm{D} \to_{\mathcal{D}} \mathrm{D}' :: B \vdash'_{\mathcal{E}} t'{:}\sigma_1{\cap}\sigma_2$ *if and only if* $\mathrm{D}_1 \to_{\mathcal{D}} \mathrm{D}'_1 :: B \vdash'_{\mathcal{E}} t'{:}\sigma_1$, *or* $\mathrm{D}_2 \to_{\mathcal{D}} \mathrm{D}'_2 :: B \vdash'_{\mathcal{E}} t'{:}\sigma_2$.

iv) If $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma_1{\cap}\sigma_2 = \langle D_1, D_2, {\cap}\mathrm{I}\rangle$, *where* $\mathrm{D}_1 :: B \vdash'_{\mathcal{E}} t{:}\sigma_1$ *and* $\mathrm{D}_2 :: B \vdash'_{\mathcal{E}} t{:}\sigma_2$, *then* $SN(\mathrm{D}_1)$ *and* $SN(\mathrm{D}_2)$, *if and only if* $SN(\mathrm{D})$. ■

Note that in part (*iii*) we have an 'or' because the redex might appear only in one of the derivations with a type different from ω.

It is easy to see that in the presence of recursive rules, $\to_{\mathcal{D}}$ is not strongly normalizing in general. Therefore, as in [6] and [7], we will control the use of recursion in the rewrite rules by imposing syntactical conditions inspired by the general scheme of Jouannaud and Okada [16] (a generalization of primitive recursion). The scheme defined in [6] ensures strong normalization of typeable terms when the constant ω is not included in the type assignment system. The version of the recursive scheme that we will use here takes also the presence of the type constant ω into account.

In a type system with ω, there are two kinds of typeable recursion: the one explicitly present in the syntax, as well as the one obtained by the so-called *fixed-point combinators*. Take, for example, the rewrite system

$$F(C(x)) \to F(x), \;\; A(x,y) \to Ap(y, Ap(Ap(x,x),y)),$$

that satisfies the general recursive scheme of [6], and is typeable with respect to

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

Then we can derive $\vdash'_{\mathcal{E}} F(A(A_0, C_0)){:}\sigma$, but notice that we also have the reduction $F(A(A_0,C_0)) \to^*_{\mathbf{R}} F(C(A(A_0,C_0))) \to_{\mathbf{R}} F(A(A_0,C_0))$. Hence this rewrite system is not strongly normalizing on typeable terms (and not even head-normalizing). The underlying problem is that A is acting as a fixed-point combinator: for every G that has type $\omega{\to}\sigma$, the term $A(A_0, G_0)$ has type σ, and $A(A_0,G_0) \to^*_{\mathbf{R}} G(A(A_0,G_0))$. To avoid this problem we will demand that *patterns are never typed using the type constant* ω. The general scheme defined in [7] ensures that patterns are not typed using ω: it requires constructor patterns, with sorts as types. The variant of the Jouannaud-Okada scheme used in this paper generalizes the one in [7] (still requiring that patterns are such that they cannot be assigned the type ω, but taking into account type derivations). It is defined as follows:

Definition 4.9 *Safety scheme.* 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*:

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

where $\bar{x}$, $\bar{y}$ are sequences of variables, and $\bar{x} \subseteq \bar{y}$. Also, $\overline{C}[\ ]$, $C'[\ ]$, $\overline{C_1}[\ ]$, and $\overline{C_m}[\ ]$ are sequences of contexts in $T(\mathcal{F}_{i-1},\mathcal{X})$, and, for $1 \le j \le m$, $\overline{C}[\bar{x}] \rhd_{mul} \overline{C_j}[\bar{x}]$ where $\lhd$ is the strict subterm ordering ($\rhd$ denotes strict superterm), and *mul* denotes multiset extension.

If, moreover, patterns cannot be typed with ω (i.e. for any typing of $F^i(\overline{C}[\bar{x}],\bar{y})$, no variable typed with ω occurs twice in $F^i(\overline{C}[\bar{x}],\bar{y})$, and no non-variable subterm of $\overline{C}[\bar{x}]$ can be typed with ω), and in any typing the derivations for the arguments $\overline{C_j}[\bar{x}]$ of the recursive calls in the right-hand side are subderivations of those of $\overline{C}[\bar{x}]$, then the rewrite system is called *safe*.

Note that the rewrite system of the example above is not safe: the pattern $C(x)$ in the first rule can be typed with ω in a derivation of $F(C(x)):\sigma$. However, the system containing only the second rule is safe.

Example 4.10 Combinator Systems are safe by definition, 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. In particular, CCL (see Ex. 2.2) is typeable in the environment $\mathcal{E}_{\mathrm{CL}}$:

$$\begin{aligned}
\mathcal{E}_{\mathrm{CL}}(S) &= (\varphi_1\to\varphi_2\to\varphi_3)\to(\varphi_4\to\varphi_2)\to\varphi_1\cap\varphi_4\to\varphi_3,\\
\mathcal{E}_{\mathrm{CL}}(K) &= \varphi_5\to\omega\to\varphi_5,\\
\mathcal{E}_{\mathrm{CL}}(I) &= \varphi_6\to\varphi_6.
\end{aligned}$$

We could also use the standard environment with arrow types only, but more types are typeable using $\mathcal{E}_{\mathrm{CL}}$.

More examples of safe systems can be found in [7]. The definition of safe system given in [7] is a particular case of this one: if patterns are constructor terms, and constructors have ground types (with sorts only), then the conditions of the previous definition are asatisfied. Hence, all the systems that are safe in the sense of [7] are also safe according to the previous definition.

The rest of this section will be devoted to the proof of strong normalization of $\to_{\mathcal{D}}$ for all typeable rewrite systems that are safe. We will use the well-known method of Computability Predicates [21] (see also [14]), adapted to the rewriting framework. The proof will have two parts; in the first one we define the predicate *Comp* on derivations, and prove that if $Comp(D)$ holds, then $SN(D)$. In the second part *Comp* is shown to hold for each derivation (this second part differs from the usual proofs for LC in that the structure of the rewrite rules can vary; here is where the general scheme plays an important role).

Definition 4.11 *i*) Let B be a basis, $t \in T(\mathcal{F},\mathcal{X})$, σ a type, and $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma$ a derivation. We define the Computability Predicate $Comp(\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma)$ recursively on σ by:

a) If $\sigma = \varphi$, or $\sigma = s$, then $Comp(\mathrm{D}) \iff SN(\mathrm{D})$.

b) If $\sigma = \alpha\to\beta$, then $Comp(\mathrm{D}) \iff \forall\, \mathrm{D}' :: B' \vdash'_{\mathcal{E}} u{:}\alpha$ [$Comp(\mathrm{D}') \Rightarrow$ $[Comp(\langle \mathrm{D},\mathrm{D}',{\to}\mathrm{E}\rangle :: \Pi\{B,B'\} \vdash'_{\mathcal{E}} Ap(t,u){:}\beta)]$].

c) If $\sigma = \sigma_1\cap\cdots\cap\sigma_n$, then $\mathrm{D} = \langle \mathrm{D}_1,\ldots,\mathrm{D}_n,{\cap}\mathrm{I}\rangle$, and $Comp(\mathrm{D}) \iff$ $\forall\, 1 \le i \le n$ $[Comp(\mathrm{D}_i :: B \vdash'_{\mathcal{E}} t{:}\sigma_i)]$.

ii) We say that a term-substitution R is *computable in a basis B with respect to a basis B'* if for

every $x{:}\sigma \in B$, there is $\mathrm{D}{::}\, B' \vdash'_{\mathcal{E}}\ x^{\mathrm{R}}{:}\sigma$, such that *Comp* (D).

Note that a derivation $\mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\omega$ is trivially computable, by Def. 4.11- (*i.c*).

The class of *neutral* terms plays a crucial role in strong normalization proofs. A notion of neutrality can be defined for derivations as well:

Definition 4.12 *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 derivation $\mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\sigma$ is *neutral* if t is a neutral term.

We will prove that *Comp* satisfies the standard properties of computability predicates. First we need a lemma:

Lemma 4.13 Let $\mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\sigma$ *be a derivation, and* u *be a subterm of* t *that is only typed with* ω *in* D. *Let* D' *be a derivation obtained from* D *by replacing the subderivations of the form* $B' \vdash'_{\mathcal{E}}\ u{:}\omega$ *by* $B' \vdash'_{\mathcal{E}}\ u'{:}\omega$, *for* u' *an arbitrary term (and propagating the replacements of* u *by* u' *along the derivation). Then Comp* $(\mathrm{D})\ \Rightarrow$ *Comp* (D').

Property 4.14 $(C1)$: *If Comp* (D), *then SN* (D).

$(C2)$: *If Comp* (D) *and* $\mathrm{D} \rightarrow_{\mathcal{D}} \mathrm{D}'$, *then Comp* (D').

$(C3)$: *Let* $\mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\sigma$ *be a neutral derivation. Then for all* D' *such that* $\mathrm{D} \rightarrow_{\mathcal{D}} \mathrm{D}'$, *Comp* (D') *holds, then Comp* (D). ■

In order to prove the Strong-Normalization Theorem we shall prove a stronger property, for which we will need the following ordering.

Definition 4.15 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. The subderivation relation (as well as the subterm ordering) will be denoted by $\lhd$, i.e. we write $\mathrm{D}'{::}\, B' \vdash'_{\mathcal{E}}\ t'{:}\sigma' \lhd \mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\sigma$ if D' is a derivation for a strict subterm t' of t, contained in D. Let $(\Sigma, \mathbf{R})$ be a *Gt*RS. We define the ordering $\gg$ on triples – consisting of a pair of natural numbers, a term, and a multiset of derivations – as the object

$$((>_{\mathbb{N}}, >_{\mathbb{N}})_{lex}, \rhd\ , (\rightarrow_{\mathcal{D}} \cup \rhd)_{mul})_{lex}$$

where $>_{\mathbb{N}}$ denotes the standard ordering on natural numbers, and *lex*, *mul* denote respectively the *lexicographic* and *multiset* extension of an ordering.

We now come to the main theorem of this section, in which we show that for any derivation $\mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\sigma$ and computable term-substitution R in $B = \{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\}$, also the derivation $\mathrm{D}'{::}\, B' \vdash'_{\mathcal{E}}\ t^{\mathrm{R}}{:}\sigma$ obtained by substituting the computable derivations $\mathrm{D}_i{::}\, B' \vdash'_{\mathcal{E}}\ x_i{}^{\mathrm{R}}{:}\sigma_i$ in D is computable. The strong normalization result then follows, using Prop. *C1*, for any derivation for t, taking for R the identity.

Property 4.16 Let R *a computable term-substitution in* $B = \{x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n\}$, *i.e. for* $1 \leq i \leq n$, *Comp* $(\mathrm{D}_i{::}\, B' \vdash'_{\mathcal{E}}\ x_i{}^{\mathrm{R}}{:}\sigma_i)$, *and* $\mathrm{D}{::}\, B \vdash'_{\mathcal{E}}\ t{:}\sigma$. *Let* D' *be the derivation*

$$\mathrm{D}' = \mathrm{D}[\mathrm{D}_1 / x_1{:}\sigma_1,\ldots,\mathrm{D}_n / x_n{:}\sigma_n]{::}\, B' \vdash'_{\mathcal{E}}\ t^{\mathrm{R}}{:}\sigma.$$

Then Comp (D').

Proof: We will interpret D' by the triple $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) = \langle (i,j), t, \{\mathrm{R}\}\rangle$ where i is the maximal super-index of the function symbols belonging to t, j is the minimum of the differences $arity(F^i)$ - $arity(F^i_k)$ such that F^i_k occurs in t, and $\{\mathrm{R}\}$ is the multiset of derivations $\{\mathrm{D}_{ij} :: B' \vdash'_{\mathcal{E}} x_i{}^{\mathrm{R}}{:}\rho_{ij}\}$ that are substituted for the $x_i{:}\sigma_i$ in the leaves of D to obtain D'. These triples are compared in the ordering $\gg$.

When R is computable, then by Def. 4.11-(*i.c*) and Def. 4.6, the derivations in $\{\mathrm{R}\}$ are all computable, and hence by Prop. *C1* they are strongly normalizable, so $\to_{\mathcal{D}}$ is well-founded on $\{\mathrm{R}\}$. Also, it is easy to see that the union of $\rhd$ with the terminating relation $\to_{\mathcal{D}}$ is well-founded. Hence, when restricted to computable term-substitutions, $\gg$ is a well-founded ordering. With the help of this ordering, we can prove the property by noetherian induction.

We will assume that $\sigma \neq \omega$, since otherwise the property is trivial. If $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$, then, by Def. 4.11-(*i.c*), we have to prove $Comp\,(\mathrm{D}'_i :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma_i)$ for very $1 \leq i \leq n$. So, without loss of generality we can consider $\sigma \in \mathcal{T}_{\mathrm{s}}$.

We distinguish the cases:

i) $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma$ is a neutral derivation.

If t is a variable then, by Lem. 4.5-(*i*), there is a τ such that $x{:}\tau \in B$, and $\tau \leq_{\mathrm{S}} \sigma$. $Comp\,(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\tau)$ holds by assumption, and then, by Def. 4.11-(*i.c*), $Comp\,(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma)$.

If t is not a variable, then also t^{R} is neutral, and we can use Prop. *C3*: If $\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma$ is irreducible, then $Comp\,(\mathrm{D}')$ holds by *C3*. Otherwise, let $\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma \to_{\mathcal{D}} \mathrm{D}'' :: B' \vdash'_{\mathcal{E}} w{:}\sigma$ at position p in t^{R}. In the following we will prove either $Comp\,(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma)$ itself, or prove $Comp\,(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} w{:}\sigma)$ and apply *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 $\tau_1, \ldots, \tau_n$ be the types of $t|_q$ in D. Let z be a new term-variable.

Take $\mathrm{R}' = \mathrm{R} \cup \{z \mapsto w|_q\}$, and note that D' has a subderivation D'_j such that $\mathrm{D}'_j :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}|_q{:}\tau_j \to_{\mathcal{D}} \mathrm{D}''_j :: B' \vdash'_{\mathcal{E}} w|_q{:}\tau_j$ at position p'. Since $t|_q = x_i$, and R is assumed to be computable, $Comp\,(\mathrm{D}'_j :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}|_q{:}\tau_j)$ holds for each subderivation of D that has root $t|_q{:}\tau_j$. So $Comp\,(\mathrm{D}''_j :: B' \vdash'_{\mathcal{E}} w|_q{:}\tau_j)$ holds by Prop. *C2*, hence R' is computable in $B \cup \{z{:}\tau_1 \cap \cdots \cap \tau_n\}$.

Now, if the variable x_i (= $t|_q$) has exactly *one* occurrence in t, then $t = t[z]_q$ modulo renaming of term-variables, and otherwise $t \rhd t[z]_q$. In the first case (since $\{\mathrm{R}\}$ contains a derivation that is rewritten to get $\{\mathrm{R}'\}$) we have $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) \gg_3 \mathcal{I}(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} t[z]_q{}^{\mathrm{R}'}{:}\sigma)$, and $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) \gg_2 \mathcal{I}(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} t[z]_q{}^{\mathrm{R}'}{:}\sigma)$ in the second case. Both cases yield, by induction, $Comp\,(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} t[z]_q{}^{\mathrm{R}'}{:}\sigma)$ and 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 $t \rhd t|_p$. Let $\tau_1, \ldots, \tau_n$ be the types assigned to $t|_p$ in the derivation $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma$. Since $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) \gg_2 \mathcal{I}(\mathrm{D}''_j :: B' \vdash'_{\mathcal{E}} t|_p{}^{\mathrm{R}}{:}\tau_j)$ then $Comp\,(\mathrm{D}''_j :: B \vdash'_{\mathcal{E}} t|_p{}^{\mathrm{R}}{:}\tau_j)$ 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 $B \cup \{z{:}\tau_1 \cap \cdots \cap \tau_n\}$, and $B \cup \{z{:}\tau_1 \cap \cdots \cap \tau_n\} \vdash'_{\mathcal{E}} t[z]_p{:}\sigma$. Now $t \rhd t[z]_p$, hence $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) \gg_2 \mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t[z]_p{}^{\mathrm{R}'}{:}\sigma)$, hence $Comp\,(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma)$ by induction.

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 $\mathrm{R}' = \mathrm{R} \cup \{z_1 \mapsto t_1{}^{\mathrm{R}}, \ldots,$

$z_n \mapsto t_n{}^{\mathrm{R}}\}$. Then if $\mathrm{D}_i :: B \vdash'_{\mathcal{E}} t_i{:}\sigma_i$ is a subderivation of D, $Comp\,(\mathrm{D}'_i :: B' \vdash'_{\mathcal{E}} t_i{}^{\mathrm{R}}{:}\sigma_i)$ holds by induction, since $t \rhd t_i$. Hence, R′ is computable in $B \cup \{z_1{:}\sigma_1, \ldots, z_n{:}\sigma_n\}$, and $B \cup \{z_1{:}\sigma_1, \ldots, z_n{:}\sigma_n\} \vdash'_{\mathcal{E}} F(z_1,\ldots,z_n){:}\sigma$. But, since $t \rhd F(z_1,\ldots,z_n)$, $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) \gg_2 \mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} F(z_1,\ldots,z_n)^{\mathrm{R}'}{:}\sigma)$. Hence $Comp\,(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma)$.

B) $t \equiv F^k(z_1,\ldots,z_n)$ where $z_1,\ldots,z_n$ are different term-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, that is, there is a rule

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

such that $t^{\mathrm{R}} = F^k(z_1,\ldots,z_n)^{\mathrm{R}} = F^k(\overline{\mathrm{C}}[\overline{M}],\overline{N}) \to_{\mathbf{R}} \mathrm{C}'[F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}),\overline{M},\overline{N}] = w$, where $\overline{\mathrm{C}}[\overline{M}]$, $\overline{N}$ are all terms in the image of R hence they have computable derivations by assumption.

Now, we will deduce $Comp\,(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} w{:}\sigma)$ 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 $\overline{x^{\mathrm{R}'}} = \overline{M}$, $\overline{y^{\mathrm{R}'}} = \overline{N}$. Since $\bar{x} \subseteq \bar{y}$ and R is computable, by Def. 4.11-(*i.c*), also R′ is computable. For every $1 \leq j \leq m$, F^k does not occur in $\overline{\mathrm{C}_j}$ (by definition of the general scheme), hence $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} F^k(z_1,\ldots,z_n)^{\mathrm{R}}{:}\sigma) \gg_1 \mathcal{I}(\mathrm{D}_j :: B' \vdash'_{\mathcal{E}} \mathrm{C}_j[\bar{x}]^{\mathrm{R}'}{:}\sigma_j)$, so also the derivations $\mathrm{D}_j :: B' \vdash'_{\mathcal{E}} \mathrm{C}_j[\bar{x}]^{\mathrm{R}'}{:}\sigma_j$ are computable.

(*Step II*): Let, for $1 \leq j \leq m$, R_j be the computable term-substitution such that $t^{\mathrm{R}_j} = F^k(\overline{\mathrm{C}_j}[\bar{x}],\bar{y})^{\mathrm{R}'}$. Since $\overline{\mathrm{C}}[\bar{x}] \rhd_{mul} \overline{\mathrm{C}_j}[\bar{x}]$, and by Def. 4.9, the derivation for $\overline{\mathrm{C}_j}^{\mathrm{R}'}[\bar{x}]$ is a subderivation of the one for $\overline{\mathrm{C}}^{\mathrm{R}'}[\bar{x}]$, also $\overline{\mathrm{D}_l} :: B' \vdash'_{\mathcal{E}} \mathrm{C}[\bar{x}]^{\mathrm{R}'}{:}\tau \rhd_{mul} \overline{\mathrm{D}_{r_j}} :: B' \vdash'_{\mathcal{E}} \mathrm{C}_j[\bar{x}]^{\mathrm{R}'}{:}\tau$, hence

$$\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} F^k(z_1,\ldots,z_n)^{\mathrm{R}}{:}\sigma) \gg_3 \mathcal{I}(\mathrm{D}'_j :: B' \vdash'_{\mathcal{E}} F^k(z_1,\ldots,z_n)^{\mathrm{R}_j}{:}\sigma_j),$$

and therefore the derivation $\mathrm{D}'_j :: B' \vdash'_{\mathcal{E}} F^k(z_1,\ldots,z_n)^{\mathrm{R}_j}{:}\sigma_j$ is computable.

(*Step III*): Let v be the term obtained by replacing, in the right-hand side of the rule, the terms $F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}),\overline{M},\overline{N}$ by fresh variables. Let R″ be the term-substitution such that $C'[F^k(\overline{\mathrm{C}_1}[\overline{M}],\overline{N}),\ldots,F^k(\overline{\mathrm{C}_m}[\overline{M}],\overline{N}),\overline{M},\overline{N}] = v^{\mathrm{R}''}$, then $t^{\mathrm{R}} \to_{\mathbf{R}} v^{\mathrm{R}''}$, and since $\sigma \neq \omega$, $\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma \to_{\mathcal{D}} \mathrm{D}'' :: B' \vdash'_{\mathcal{E}} v^{\mathrm{R}''}{:}\sigma$. Notice that above we have shown that R″ is computable. When an F^j occurs in v, then by definition of the general scheme $j \mathbin{¡} k$, and therefore $\mathrm{D}' \gg_1 \mathrm{D}''$, hence D″ is computable, and since $w = v^{\mathrm{R}''}$, we get $Comp\,(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} w{:}\sigma)$.

C) $t = Ap(z_1,z_2)$ where $z_1,z_2 \in \mathcal{X}$. By assumption, the derivations for $z_1{}^{\mathrm{R}}$ and $z_2{}^{\mathrm{R}}$ are computable, and since t is well-typed, z_1 must have an arrow type. Then, by Def. 4.11, $\mathrm{D}' :: B' \vdash'_{\mathcal{E}} Ap(z_1{}^{\mathrm{R}},z_2{}^{\mathrm{R}}){:}\sigma$ is computable. But $Ap(z_1{}^{\mathrm{R}},z_2{}^{\mathrm{R}})$ is the same as $Ap(z_1,z_2)^{\mathrm{R}}$.

ii) $\mathrm{D} :: B \vdash'_{\mathcal{E}} t{:}\sigma$ is not neutral. Let $t \equiv F_n(t_1,\ldots,t_n)$. There are two cases:

a) Assume that at least one of the t_i is not a term-variable. Since $t \rhd t_i$ for $1 \leq i \leq n$, by induction, for each subderivation D_i of D with root $t_i : \tau_i$, we have $Comp\,(\mathrm{D}'_i :: B' \vdash'_{\mathcal{E}} t_i{}^{\mathrm{R}}{:}\tau_i)$. Hence, also the term-substitution R′ $= \{z_1 \mapsto t_1{}^{\mathrm{R}},\ldots,z_n \mapsto t_n{}^{\mathrm{R}}\}$ is computable. Since $t \rhd F_n(z_1,\ldots,z_n)$, we have $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\sigma) \gg_2 \mathcal{I}(\mathrm{D}'' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}'}{:}\sigma)$, and D″ is computable by induction. Note that $t^{\mathrm{R}'} = t^{\mathrm{R}}$, and $\mathrm{D}'' = \mathrm{D}'$.

b) All t_i are variables. Since $B \vdash'_{\mathcal{E}} t{:}\sigma$, by Lem. 4.5-(*ii*) $\sigma = \alpha{\to}\beta$. We have to prove $Comp\,(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}{:}\alpha{\to}\beta)$, that is, if $Comp\,(\mathrm{D}'' :: B'' \vdash'_{\mathcal{E}} u{:}\alpha)$, then $Comp\,(\langle \mathrm{D}',\mathrm{D}'',{\to}\mathrm{E}\rangle :: \Pi\{B',B''\} \vdash'_{\mathcal{E}} Ap(t^{\mathrm{R}},u){:}\beta)$, for every D″. Take D_0 =

$\langle \mathrm{D}', \mathrm{D}'', \to \mathrm{E}\rangle :: \Pi\{B', B''\} \vdash'_{\mathcal{E}} Ap(t^{\mathrm{R}}, u):\beta$. Since $Ap(t^{\mathrm{R}}, u)$ is neutral, by Prop. *C*3, it is sufficient to prove that, if $\mathrm{D}_0 \to_{\mathcal{D}} \mathrm{D}''' :: \Pi\{B', B''\} \vdash'_{\mathcal{E}} v:\beta$, then *Comp* (D''').

This will be proved by induction on the sum of the lengths of the rewrite sequences out of $\mathrm{D}'' :: B'' \vdash'_{\mathcal{E}} u:\alpha$ and out of the substitution. Note that since both are computable, by Prop. *C1*, *SN* $(\mathrm{D}'' :: B'' \vdash'_{\mathcal{E}} u:\alpha)$, and *SN* $(\{\mathrm{R}\})$.

(*Base*): If the type-derivations for u and $\{\mathrm{R}\}$ are in normal form, the only reduction step out of D_0 could be:

$$\langle \mathrm{D}', \mathrm{D}'', \to \mathrm{E}\rangle :: \Pi\{B', B''\} \vdash'_{\mathcal{E}} Ap(F_n(z_1, \ldots, z_n)^{\mathrm{R}}, u):\beta \to_{\mathcal{D}} \mathrm{D}''' :: \Pi\{B', B''\} \vdash'_{\mathcal{E}} F_{n+1}(z_1{}^{\mathrm{R}}, \ldots, z_n{}^{\mathrm{R}}, u):\beta,$$

then $\mathcal{I}(\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t^{\mathrm{R}}:\sigma) \gg_1 \mathcal{I}(\mathrm{D}''')$ is computable.

(*Induction step*): If the reduction out of D_0 takes place inside u or inside t^{R} (in the last case it must be inside $\{\mathrm{R}\}$ since the rewrite system is safe) then D''' is computable by induction. If $\langle \mathrm{D}', \mathrm{D}'', \to \mathrm{E}\rangle \to_{\mathcal{D}} \mathrm{D}''' :: \Pi\{B', B''\} \vdash'_{\mathcal{E}} F_{n+1}(z_1{}^{\mathrm{R}}, \ldots, z_n{}^{\mathrm{R}}, u):\beta$, we proceed as in the base case. ■

Theorem 4.17 Strong Normalization of Derivation Reduction. *If* $(\Sigma, \mathbf{R})$ *is typeable in* $\vdash'_{\mathcal{E}}$ *and safe, then for every* $\mathrm{D} :: B \vdash'_{\mathcal{E}} t:\sigma$, *SN* (D).

Proof: From Prop. 4.16 and *C1*, taking R such that $x^{\mathrm{R}} = x$. ■

Note that for the strong normalization property to hold, it is not enough to define a reduction relation that considers only typeable redexes in a term. It is crucial to preserve the derivations, as the following example shows:

Example 4.18 Consider the $\mathcal{G}$TRS with rules:

$$\begin{array}{ll} E(x,y) & \to Ap(y, Ap(Ap(x,x),y)), \\ C(x,y) & \to y, \end{array}$$

This rewrite system is typeable and safe with respect to

$$\begin{array}{l} \mathcal{E}(E) = \omega\to(\omega\to(\alpha\to\alpha)) \to(\alpha\to\alpha), \\ \mathcal{E}(C) = \omega\to\alpha\to\alpha. \end{array}$$

Notice that the term $E(E_0, C_0)$ is typeable by $\alpha\to\alpha$ in this environment. If instead of reducing derivations we would allow to reduce a redex whenever it has a type different from ω, then there is an infinite reduction sequence out of this term: $E(E_0, C_0) \to_{\mathbf{R}} Ap(C_0, Ap(Ap(E_0, E_0), C_0)) \to_{\mathbf{R}} C_1(Ap(Ap(E_0, E_0), C_0)) \to^*_{\mathbf{R}} C_1(E(E_0, C_0)) \ldots$

However, any derivation for $\vdash'_{\mathcal{E}} E(E_0, C_0) : \alpha\to\alpha$ is strongly normalizable according to $\to_{\mathcal{D}}$. Take for instance the derivation that assigns ω to E_0 and $\omega\to\alpha\to\alpha$ to C_0 (we will annotate function symbols with types instead of writing the type-derivation):

$$\begin{array}{c} \vdash'_{\mathcal{E}} E(E_0{:}\omega, C_0{:}\omega\to\alpha\to\alpha){:}\alpha\to\alpha \to_{\mathcal{D}} \\ \vdash'_{\mathcal{E}} Ap(C_0{:}\omega\to\alpha\to\alpha, Ap(Ap(E_0, E_0), C_0){:}\omega){:}\alpha\to\alpha \to_{\mathcal{D}} \\ \vdash'_{\mathcal{E}} C_1(Ap(Ap(E_0, E_0), C_0){:}\omega){:}\alpha\to\alpha \end{array}$$

and the last derivation is in normal form, since the type assigned in this derivation to $Ap(Ap(E_0, E_0), C_0)$ is ω.

5 Approximation and normalization properties

In this section, the results of the previous section will be used to prove four theorems. The first is an approximation theorem; our definition of approximants is a combination of the one given by Wadsworth for the Lambda Calculus [23], and the approximants for Term Rewriting Systems defined by Thatte [22], based on the notion of Ω-normal forms of Huet and Lévy [15]. The last three theorems will formulate a relation between assignable types and head-normalization, strong normalization and normalization properties, respectively.

In order to define approximants of terms, we start by introducing a special symbol $\bot$ (*bottom*) into the language (so $\bot$ is not in $\mathcal{X}$, and neither in $\mathcal{F}$), that is intended to represent meaningless terms. (The definition of this new set of terms $T(\mathcal{F},\mathcal{X},\bot)$ is straightforward.) To define type assignment on $T(\mathcal{F},\mathcal{X},\bot)$, the type assignment rules given in Def. 3.4 need not be changed, it suffices that terms are allowed to be in $T(\mathcal{F},\mathcal{X},\bot)$. Since $\bot \notin \mathcal{F}\cup\{Ap\}$, this implies that $\bot$ can only be given the type ω, or appear in subterms that are given the type ω.

Terms in $T(\mathcal{F},\mathcal{X},\bot)$ can be ordered using the relation $\sqsubseteq$:

Definition 5.1 *i*) $t \sqsubseteq u$ is inductively defined by:

a) For every $u \in T(\mathcal{F},\mathcal{X},\bot)$, $\bot \sqsubseteq u$.

b) For every $t \in T(\mathcal{F},\mathcal{X},\bot)$, $t \sqsubseteq t$.

c) $F(t_1,\ldots,t_n) \sqsubseteq F(u_1,\ldots,u_n)$, if and only if, for all $1 \le i \le n$, $t_i \sqsubseteq u_i$.

ii) We write $t\uparrow u$ (and say that t and u *are compatible*) if there is a $v \in T(\mathcal{F},\mathcal{X},\bot)$ such that $t \sqsubseteq v$ and $u \sqsubseteq v$. We write $t\uparrow V$ if there is an $l \in V$ such that $t\uparrow l$.

By abuse of notation, we will use the symbol $\bot$ also for the term-substitution that replaces term-variables by $\bot$: $\bot = \{x \mapsto \bot \mid x \in \mathcal{X}\}$. In the following we consider a given $\mathcal{G}$TRS $(\Sigma,\mathbf{R})$, and $Lhs^{\bot} = \{l^{\bot} \mid \exists r\ [l \to r \in \mathbf{R}]\}$.

We will now develop the notion of approximant of a term with respect to a given $\mathcal{G}$TRS. A particular difference with the definition of approximant for lambda terms [23] is that our definition is 'static', whereas the other notion was defined as normal forms with respect to an extended notion of reduction. This approach would not be appropriate for our paper, because, to name just one problem, we would not be able to prove a subject reduction result for such a notion of reduction. Instead, we will recursively replace redexes by $\bot$. While doing this, it can be that a term is created that itself is not a redex, but looks like one, in the sense that is compatible to a left-hand side of a rewrite rule (where variables are replaced by $\bot$). Also such 'possible redexes' will be replaced by $\bot$.

Definition 5.2 $\mathcal{DA}(t)$, the *direct approximant of* t with respect to $(\Sigma,\mathbf{R})$ is defined by:

i) $t = x$. $\mathcal{DA}(x) = x$.

ii) $t = F(t_1,\ldots,t_n)$; let, for $1 \le i \le n$, $a_i = \mathcal{DA}(t_i)$.
$\mathcal{DA}(t) = \bot$, if $F(a_1,\ldots,a_n)\uparrow Lhs^{\bot}$; otherwise, $\mathcal{DA}(t) = F(a_1,\ldots,a_n)$.

iii) $t = Ap(t_1,t_2)$; let $a_1 = \mathcal{DA}(t_1)$, and $a_2 = \mathcal{DA}(t_2)$.
$\mathcal{DA}(t) = \bot$, if $a_1 = \bot$, or $a_1 = F_i(a_1,\ldots,a_i)$; otherwise, $\mathcal{DA}(t) = Ap(a_1,a_2)$.

Approximants of terms are obtained by taking direct approximants of their reducts (and making a downward closure).

Definition 5.3 *i*) $\mathcal{DA}$, the set of *approximate normal forms* is defined as

$$\mathcal{DA} = \{a \in T(\mathcal{F},\mathcal{X},\bot) \mid \mathcal{DA}(a) = a\}.$$

ii) $\mathcal{A}(t)$, the *set of approximants of* t, is defined by:

$$\mathcal{A}(t) = \{a \in \mathcal{DA} \mid \exists\, u\ [t \to^* u\ \&\ a \sqsubseteq \mathcal{DA}(u)]\}.$$

Intuitively, the terms whose only approximant is $\bot$ are undefined (i.e. meaningless). We will see below (Cor. 5.7) that typeable terms cannot be undefined. We will also see that this implies that typeable terms are head-normalizable. We introduce more notation now:

Definition 5.4 For any derivation $\mathrm{D} :: B \vdash_{\mathcal{E}} t{:}\sigma$, we denote by t_{D} the term obtained from t by replacing all its subterms that do not have a type different from ω in D, by $\bot$.

In other words, t_{D} is obtained by replacing any subterm u of t such that in D we only have $B' \vdash_{\mathcal{E}} u{:}\omega$, by $\bot$.

Lemma 5.5 i) Let $\mathrm{D} :: B \vdash_{\mathcal{E}} t{:}\sigma$. *Then there exists a derivation* $\mathrm{D}' :: B \vdash_{\mathcal{E}} t_{\mathrm{D}}{:}\sigma$.
ii) If $\mathrm{D} :: B \vdash_{\mathcal{E}} t{:}\sigma$ *is irreducible with respect to* $\to_{\mathcal{D}}$ *then* $t_{\mathrm{D}} \in \mathcal{A}(t)$.
iii) If $\mathcal{DA}(t) \neq \bot$, *then* t *is in head-normal form.*

With Thm. 4.17 and Lem. 5.5 we are able to prove:

Theorem 5.6 Approximation Theorem. *If* $(\Sigma, \mathbf{R})$ *is typeable in* $\vdash_{\mathcal{E}}$ *and safe, then for every* t *such that* $B \vdash_{\mathcal{E}} t{:}\sigma$ *there is an* $a \in \mathcal{A}(t)$ *such that* $B \vdash_{\mathcal{E}} a{:}\sigma$.

Proof: Let $B \vdash_{\mathcal{E}} t{:}\sigma$, then by Lem. 4.2-(*i*) there is a B' such that $B \leq B'$ and $B' \vdash'_{\mathcal{E}} t{:}\sigma$. Let D be a derivation for this last result, then by Thm. 4.17, $SN(\mathrm{D})$. Let $\mathrm{D}' :: B' \vdash'_{\mathcal{E}} t'{:}\sigma$ be the normal form of D with respect to $\to_{\mathcal{D}}$ and let $a = t'_{\mathrm{D}'}$. Then, by Lem. 5.5-(*ii*), $a \in \mathcal{A}(t')$, and by Lem. 4.8-(*i*), $t \to_{\mathbf{R}} t'$. Hence $a \in \mathcal{A}(t)$. By Lem. 5.5-(*i*), $B' \vdash'_{\mathcal{E}} a{:}\sigma$, and by Lem. 4.2-(*ii*), $B \vdash_{\mathcal{E}} a{:}\sigma$. ■

Corollary 5.7 Let $(\Sigma, \mathbf{R})$ *be typeable with respect to* $\mathcal{E}$ *and safe. If* $B \vdash_{\mathcal{E}} t{:}\sigma$, *and* $\sigma \neq \omega$, *then there exists* $a \in \mathcal{A}(t)$ *such that* $a \neq \bot$.

In other words, all typeable terms are meaningful.

Theorem 5.8 Head Normalization Theorem. *If* $B \vdash_{\mathcal{E}} t{:}\sigma$, *and* $\sigma \neq \omega$, *then* t *has a head-normal form.*

Proof: If $B \vdash_{\mathcal{E}} t{:}\sigma$, then by Cor. 5.7, there exists $a \in \mathcal{A}(t)$ such that $a \neq \bot$. Since $a \in \mathcal{A}(t)$, there is a v such that $t \to^* v$ and $a \sqsubseteq \mathcal{DA}(v)$. Hence $\mathcal{DA}(v) \neq \bot$. Then, by Lem. 5.5-(*iii*), v is in head-normal form, so, in particular, t has a head-normal form. ■

We have seen that typeable terms have an approximant different from $\bot$, and a head-normal form. Ariola et al. [1] define a notion of meaningless terms as terms without a root stable form; the set of undefined terms according to this notion is called U_{rs}. Untypeable terms are the meaningless terms in our system. Let U_t be the set of untypeable terms. By Thm. 5.8, $U_{rs} \subseteq U_t$.

Theorem 5.9 Strong Normalization Theorem. *Let* $(\Sigma, \mathbf{R})$ *be a safe rewrite system, typeable without using* ω *at all. If* $B \vdash_{\mathcal{E}} t{:}\sigma$, *and* ω *is not used to derive this result, then* t *is strongly normalizable.*

Proof: Let $\mathrm{D} :: B \vdash_{\mathcal{E}} t{:}\sigma$. By Thm. 4.17, D is strongly normalizable. Since ω is not used, and the system without ω has the Subject Reduction Property [6], by definition of $\to_{\mathcal{D}}$ and Lem. 4.8-(*i*), we obtain strong normalization of t. ■

Notice that the converses of the two previous theorems do not hold because the environment is given (and fixed).

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 for $\mathcal{G}\iota$TRS, even if one considers safe recursive systems only. Then, as in [7], we will restrict the study of normalization properties of $\mathcal{G}\iota$TRS to non-Curryfied terms. Actually, to get a normalization result similar to that of LC we also need to impose the following condition on the $\mathcal{G}\iota$TRS:

Definition 5.10 A $\mathcal{G}\iota$TRS is *complete* if whenever a non-Curryfied term t that is assigned the ω-free type σ is reducible at a position p such that $t|_p$ can be assigned a type containing ω, there exists $q < p$ such that $t|_q$ can be assigned an ω-free type and $t|_q[x]_p$ is not in head-normal form.

Intuitively, in a complete $\mathcal{G}\iota$TRS a non-Curryfied term $F(t_1,\ldots,t_n)$ that has an ω-free type, and where there is a redex t_i that has a type containing ω, will be reducible either at the root (without taking t_i into account), or in some t_j with an ω-free type. This means that the rules defining F cannot have patterns that can be assigned types with ω, and also that constructors cannot accept arguments having a type which contains ω. Moreover, if a defined function accepts arguments having types with ω then its definition must be exhaustive.

Defined functions of safe systems satisfy the first condition. So, a safe system is complete whenever constructors have ground types without ω, and for all defined function F that accepts arguments with types that contain ω, the patterns of the rules defining F cover all possible cases.

Example 5.11 Combinator Systems are complete, since there is a rule for each combinator.

Theorem 5.12 Normalization Theorem. *Let t be a non-Curryfied term in a typeable, safe, and complete $\mathcal{G}\iota$TRS. If $B \vdash_{\mathcal{E}} t{:}\sigma$ and ω does not appear in B, σ, then t is normalizable.*

Proof: By Thm. 4.17, $SN(\mathrm{D} :: B \vdash_{\mathcal{E}} t{:}\sigma)$. Let D' be a normal form of D, i.e. $\mathrm{D}' :: B \vdash_{\mathcal{E}} t'{:}\sigma$, and B,σ are ω-free. We prove that t' is in normal form by case analysis.

Since t' is a non-Curryfied term, two cases are possible:

i) t' is a variable, hence it is in normal form.

ii) $t' = F(t_1,\ldots,t_n)$ In this case, t' cannot be a redex itself, because the derivation $B \vdash_{\mathcal{E}} t'{:}\sigma$ is in normal form and $\sigma \neq \omega$. Moreover, only subterms that are typed with ω can be reducible. But since the system is complete, the existence of redex of type ω implies the existence of redex of type different from ω, which leads to a contradiction. Then t' is in normal form. ■

6 Conclusions and future work

Combinator Systems are $\mathcal{G}\iota$TRS that are trivially safe and complete (see Ex. 4.10, 5.11), 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 [13]. 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.

Approximants can be used to characterize equality in models of term rewriting languages, as shown by Thatte [22]: a fully abstract model can be built by interpreting terms as sets of

approximants (more precisely, the interpretation of t is $\mathcal{A}(t)$). Our approximation theorem shows the connection between the intersection type assignment system and the semantics of the language: any typeable term has an interpretation different from $\perp$ in the model (i.e. it has a "meaning"). Approximants can be used to characterize equality in lambda models as well [23]. In the future, we will look at these properties in the setting of the combination of TRS and LC.

References

[1] Z. Ariola, R. Kennaway, J.W. Klop, R. Sleep, and F-J. de Vries. Syntactic definitions of undefined: on defining the undefined. In *TACS '94*, *LNCS*, 789, pages 543–554, 1994.

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

[3] S. van Bakel. Partial Intersection Type Assignment in Applicative Term Rewriting Systems. In *TLCA '93*, *LNCS* 664, pages 29–44, 1993.

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

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

[6] S. van Bakel and M. Fernández. Strong Normalization of Typeable Rewrite Systems. In *HOA '93*, *LNCS* 816, pages 20–39, 1994.

[7] S. van Bakel and M. Fernández. (Head-)Normalization of Typeable Rewrite Systems. *RTA '95*, *LNCS* 914, pages 279–293, 1995.

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

[9] F. Cardone and M. Coppo. Two Extensions of Curry's Type Inference System. In *Logic and Computer Science*, pages 19–75, 1990.

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

[11] H.B. Curry and R. Feys. *Combinatory Logic*, volume 1, 1958.

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

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

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

[15] G. Huet and J.J. Lévy. Computations in Orthogonal Rewriting Systems. In *Computational Logic. Essays in Honour of Alan Robinson*, 1991.

[16] J.P. Jouannaud and M. Okada. Executable higher-order algebraic specification languages. In *LiCS '91*, pages 350–361, 1991.

[17] R. Kennaway, V. van Oostrom, and F.J. de Vries. Meaningless terms in rewriting. Submitted for publication. Obtainable as: http://wwwbroy.informatik.tu-muenchen.de/ oostrom, 1996.

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

[19] D. Leivant. Typing and computational properties of lambda expressions. *Theoretical Computer Science*, 44:51–68, 1986.

[20] F. Pfenning. Partial Polymorphic Type Inference and Higher-Order Unification. In *LISP and Functional Programming Languages '88*, pages 153–163, 1988.

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

[22] S.R. Thatte. Full Abstraction and Limiting Completeness in Equational Languages. *Theoretical Computer Science*, 65:85–119, 1989.

[23] C.P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞-models of the lambda-calculus. *SIAM J. Comput.*, 5:488–521, 1976.