On the Equivalence of Persistent Term Rewriting Systems and Recursive Program Schemes

Zurab Khasidashvili
INRIA-Rocquencourt
78153 Le Chesnay Cedex, France
Zurab.Khasidashvili@inria.fr

Abstract

We introduce Persistent Term Rewriting Systems (PTRSs) by restricting redex-creation during reductions in Orthogonal Term Rewriting Systems (OTRSs [6, 10]). In particular, Recursive (Applicative) Program Schemes (RPSs [4]), considered as TRSs, are persistent. Two PTRSs R and R' are syntactically equivalent when any term t has an R-normal form iff it has an R'-normal form and they coincide. We prove that syntactic equivalence is decidable for PTRSs. Further, we show that the equivalence problem (over all continuous interpretations) is decidable for RPSs with unary basic functions by reducing the question to a decidable number-theory problem. Finally, we show that weak and strong normalization and the reducibility problem also are decidable in PTRSs.

1 Introduction

The behaviour of a rewrite system strongly depends on the character of redex-creation and redex-destruction. In Orthogonal (that is, left-linear and non-ambiguous) Term Rewriting Systems (OTRSs) no destruction of redexes is possible because there are no critical pairs. Thus OTRSs can be classified depending only the possible kinds of redex-creation. The simplest OTRSs are *Persistent* TRSs, where redex-creation depends only on a rewrite rule; actual arguments and the context in which reduction takes place does not matter — pattern of a created redex is in the pattern of the contracted redex. Obviously, Recursive (Applicative) Program Schemes (RPSs [4]), considered as TRSs, are persistent, since the pattern of any redex consists of one symbol only — an unknown function.

Here we study Persistent TRSs with a view toward their applications to RPSs. Recall that an RPS is a pair (R, t), where R is a system of definitions of unknown functions, and t is a term called a main program. In particular, we study two kinds of equivalences of PTRSs and RPSs. Syntactic equivalence is an equivalence that a programmer may be interested in: two PTRSs R and R' are called syntactically equivalent when any term t has R-normal form iff it has an R'-normal form and they coincide. The notion of equivalence of RPSs takes into account also infinite computations in all interpretations, which (conventionally) are continuous algebras: two RPSs (R, t) and (R', t') are called equivalent if the values of t and t' coincide in any interpretation of R and R'.

The main feature of PTRSs is that any *free* subterm, that is, any subterm that is not a pattern-subterm of a redex, remains free under any reduction. Thus free subterms never take part in redex creation. This implies that the question of whether an argument of a redex is *essential*, that is, has a *descendant* under any reduction and hence contributes to the finite or infinite result of a computation, depends only on the rewrite rule. Thus, each rule has an *essentiality indicator*, indicating which arguments are essential. A subterm is inessential iff it is in an inessential argument of a redex, that is, iff there is a redex that makes the subterm inessential. Using these properties, we establish criteria for strong normalization (termination of all reduction sequences) and for (weak) normalization (existence of a normal form). For example, a term is normalizable iff all of its essential free subterms are normalizable. This holds iff all of its essential redexes are *semi-finite*, meaning that left-hand-sides of corresponding rules are normalizable. All the necessary information can be extracted from *essential rule graphs*, which are minimal oriented graphs labeled with rules such that redexes corresponding to sons have essential occurrences in left-hand sides of ancestors. For example, a redex is semi-finite iff the corresponding essential rule graph does not contain an infinite path (and hence is acyclic when the system contains a finite number of rewrite rules).

We present an algorithm that, given a term, decides whether it is normalizable and constructs its normal form in an efficient way. Thus, for any PTRS R one can construct a corresponding *irreducible* PTRS $IR(R)$, that is, one with right-hand-sides of its semi-finite rules in normal form. $IR(R)$ is syntactically equivalent to R. Further, we show that syntactically equivalent irreducible PTRSs are identical except for the right-hand-sides of *non-semi-finite rules*. This criterion leads to a decision algorithm for syntactic equivalence of PTRSs.

Note that decidability of normalization for PTRSs implies immediately that "persistent languages", for example RPSs, do not have the full computational power. Indeed, the existence of an interpretor for, say λ-calculus, in a PTRS would imply decidability of weak normalization for λ-terms, which is not valid [2].

Decidability of equivalence between RPSs is a famous open problem. Several decidable subclasses are found in [1, 5, 15]. Here we show decidability of equivalence between RPSs with unary basic functions. We first define *strongly irreducible* PTRSs, where right-hand sides of semi-finite rules are in normal form and the right-hand sides of non-semi-finite rules contain only non-semi-finite redexes. We further show decidability of equivalence for strongly irreducible RPSs with monadic basic functions by reducing this question to a decidable number-theory problem. The decidability result then follows from the equivalence of any RPS with its corresponding strongly irreducible RPS.

We also give some simple decidable criteria for weak and strong normalization of PTRSs (that is, for weak and strong normalization of all terms in PTRSs). For example, a PTRS R is weakly normalizing iff it is *essentially hierarchical*, meaning that there is an enumeration of rules in R such that only redexes (corresponding to rules) with smaller numbers have essential occurrences in right-hand sides of rules.

The rest of this paper is organized as follows. In section 2 we sketch some properties of PTRSs and establish a criterion for syntactic equivalence of PTRSs. Complete proofs can be found in [9]. In section 3 we give a decision algorithm for syntactic equivalence. Section 4 is devoted to the decidability of equivalence of RPSs with monadic basic functions. Finally, in section 5 we show that strong normalization and the reducibility problem also are decidable in PTRSs.

2 A criterion for syntactic equivalence

We recall some basic notion of TRS theory; one can find comprehensive introductions to the subject in [6] and [10]. A *TRS* is a pair (Σ, R), where the alphabet Σ consists of variables and function symbols and R is a set of rewrite rules r of the form $t \to s$. The left-hand side t is any term different from a variable, and the term s may only contain variables that occur in t. An *r-redex* u is obtained from t by substituting arbitrary terms for the variables in t, and the corresponding instance of s is the *contractum* of t. *Arguments* of u are subterms of u that correspond to variables of t, and the rest is the *pattern* of u. Subterms of u rooted at the pattern are called the *pattern-subterms* of u. The arguments, pattern, and pattern-subterms are defined analogously in the contractum of u. A TRS is *orthogonal* (OTRS) if it is *left-linear* and *non-ambiguous*, that is, patterns of redexes can not overlap.

A one step *reduction* in which a redex u in a term o is contracted is written $o \xrightarrow{u} e$ or $o \to e$. We write $P : o \twoheadrightarrow e$ if P is a *reduction* of o to e comprising 0 or more steps. We use t, s, e, o for terms, u, v, w for redexes, and P, Q for reductions. $|P|$ denotes the length of P. $!t$ denotes a normal form of t. We write $s \in t$ if s is a subterm of t.

Definition 2.1 Let $t \xrightarrow{u} s$ and let e be the contractum of u in s. For each argument o of u there are 0 or more arguments of e. We call them (u-)*descendants* of o. Correspondingly, subterms of o have 0 or more *descendants*. By definition, the *descendant* of each pattern-subterm of u is e. Descendants of all redexes of t except u are also called *residuals*. By definition, u does not have residuals in s. A redex of s is said to be *created* by contracting u or to be an (u)-*new* redex if it is not a residual of a redex of t'. It is clear what is to be meant under *descendants* of subterms that are not in u. The notion of *descendant* extends naturally to arbitrary reductions.

Definition 2.2 We call a subterm s in t *essential* (written $ES(s,t)$) if s has at least one descendant under any reduction starting from t and *inessential* (written $IE(s,t)$) otherwise.

We call two co-initial reductions $P : t \twoheadrightarrow s$ and $Q : t \twoheadrightarrow e$ *strictly equivalent* if $s = e$ and the P-descendants and Q-descendants of any subterm of t are the same occurrences in s and e. It can be shown [8, 9] that orthogonal TRSs are *strictly Church-Rosser*, i.e., for any co-initial reductions P and Q there are reductions P' and Q' such that the concatenation of P and P' is strictly equivalent to the concatenation of Q and Q'.

The following lemma follows from strict Church-Rosser theorem and Definition 2.1.

Lemma 2.1 **(1)** Let $P : t \twoheadrightarrow t'$ and $s \in t$. Then $IE(s,t)$ iff all P-descendants of s are inessential in t'. In particular, if t' is a normal form, then $ES(s,t)$ iff s has a P-descendant.

(2) Let $e \in s \in t$ and $ES(e,t)$. Then $ES(s,t)$.

(3) Let s be a pattern-subterm of a redex $u \in t$. Then $ES(u,t)$ iff $ES(s,t)$.

The main property of essential subterms is given by the following theorem.

Theorem 2.1 ([7, 8, 12]) Let t be a term in an orthogonal TRS. Contraction of essential redexes in t leads to its normal form whenever t is normalizable.

Definition 2.3 Let $t \xrightarrow{u} s$ in an OTRS R, and let $v \in s$ be a new redex. We call v *generated* if its pattern is in the pattern of the contractum of u. We call an OTRS R *persistent* (PTRS) if, for each reduction step in R, any created redex is generated.

Definition 2.4 We call a subterm s in t *free* if s is not a proper pattern-subterm of a redex in t.

Below we consider only PTRSs. It is easy to see that free subterms remain free under any reduction in PTRSs. This is not the case for OTRSs in general. All the properties of PTRSs established below crucially rely on this persistency property of free subterms.

We write $t = (t_1//e_1, ..., t_k//e_k)e$ if $e_1, ..., e_n$ are non-overlapping proper subterms in e and t is obtained from e by their replacement with $t_1, ..., t_n$ respectively.

Definition 2.5 Let $t = (t_1//s_1, \ldots, t_k//s_n)s$, where subterms s_i and t_i are free in s and t, respectively, and let $P : s = e_0 \xrightarrow{v_0} e_1 \xrightarrow{v_1} \ldots$. Construct the reduction $P\|(t) : t = o_0 \xrightarrow{u_0} o_1 \xrightarrow{u_1} \ldots$ as follows. If v_0 is outside $s_1, \ldots, s_n$, then we take as u_0 its corresponding redex in t_0. Otherwise $u_0 = \emptyset$. Since o_1 is obtained from e_1 by replacing the descendants of $s_1, \ldots, s_n$ with $t_1, \ldots, t_n$ (the replaced subterms are free), in o_1, we can choose the redex u_1 analogously, and so on.

Remark 2.1 The reduction $P\|(t)$ depends not only on P and t, but also on the choice of $s_1, \ldots, s_k$, but the notation does not give rise to ambiguity.

Lemma 2.2 Let $t = (t_1//s_1, \ldots, t_n//s_n)s$, the subterms $s_1, \ldots, s_k$ be free and essential in s, and $s_{k+1}, \ldots, s_n$ be free and inessential in s. Further, let $P : s \twoheadrightarrow s'$ and $Q = P\|(t) : t \twoheadrightarrow t'$. Then

(1) The term t' can be obtained from s' by replacing the descendants of $s_1, \ldots s_n$ with $t_1, \ldots, t_n$ respectively. The replaced occurrences of $t_1, \ldots, t_n$ in t' are the only descendants of $t_1, \ldots, t_n \in t$.

(2) If e and o are corresponding subterms in s and t, then e has P-descendants iff o has Q-descendants and they are corresponding subterms of s' and t'.

(3) If P is normalizing, then, for all $j = 1, \ldots, k$, s_j has a descendant in s', which is a normal form of s_j.

Proof sketch By induction on $|P|$.

Corollary 2.1 Let $t = (t_1//s_1, \ldots, t_n//s_n)s$, where $s_1, \ldots, s_k$ are essential and free in s, subterms $s_{k+1}, \ldots, s_n$ are inessential and free in s, and $t_1, \ldots, t_n$ are free in t.

(1) If s' and t' are any corresponding subterms in s and t, then $ES(s', s)$ iff $ES(t', t)$.

(2) If s is weakly normalizable, then s_i is weakly normalizable for all $i = 1, \ldots, k$.

(3) If s is weakly normalizable and t_i is weakly normalizable for all $i = 1, \ldots, k$, then t is weakly normalizable and the normal form $!t$ of t can be obtained from $!s$ by replacing the descendants of $s_1, \ldots, s_k$, which are occurrences of $!s_1, \ldots, !s_k$, with $!t_1, \ldots, !t_k$ respectively.

We call redexes *similar* if they are instances of the same rule. Since arguments of redexes are free, it follows from Corollary 2.1.(1) that corresponding arguments of similar redexes are either both essential or both inessential. Hence, to each rule r one can assign an *essentiality indicator* $IE(r)$, which is a list of numbers of essential arguments of any r-redex. By definition, all r-redexes have the same essentiality indicator.

The following lemma reflects some structural properties of PTRSs.

Lemma 2.3 Let t be a term in a PTRS. Then

(1) Each outermost redex in t is essential.

(2) If a subterm e is inessential in t, then e is in an inessential argument of an essential redex.

(3) If s is a free essential subterm in t and e is essential in s, then e is essential in t as well.

Proof sketch (1) From Definition 2.2.

(2) The inessential subterm e is in an inessential argument of the innermost essential redex containing e.

(3) A redex that makes e inessential can neither occur in s nor contain s in its inessential argument. Thus, such a redex does not exist.

Definition 2.6 We call a redex in a PTRS R *trivial* if it is the left-hand side of a rewrite rule in R. We call a redex u *semi-finite* if its similar trivial redex is normalizable and call *strictly infinite* otherwise. We call a rule $r \in R$ *semi-finite* if the trivial r-redex is semi-finite and *strictly infinite* otherwise.

Lemma 2.4 Let t be a term in a PTRS. Then t is normalizable iff each essential redex in t is semi-finite.

Proof ($\Rightarrow$) By Corollary 2.1.(2), any essential redex u in t and all essential free subterms of u are normalizable. So by Corollary 2.1.(3), the trivial redex similar to u is normalizable as well. ($\Leftarrow$) By induction on the number n of essential redexes in t. If $n = 0$, then, by Lemma 2.3.(1), t is a normal form. So suppose that $n > 0$. Let u be an innermost essential redex in t and let v be the trivial redex similar to u. By the assumption, v is normalizable. It follows from Lemma 2.3.(1),(3) that essential arguments of u are in normal form. Thus, by Corollary 2.1.(2), u is normalizable. Let t' be obtained from t by replacing u with its normal form $!u$. By Corollary 2.1.(1), t' contains $n-1$ essential redexes. By the induction assumption, t' is normalizable. Hence t is normalizable.

Definition 2.7 Let R be a PTRS and let $r, r' \in R$. We call r' an *essential successor* of r if an r'-redex has an essential occurrence in the right-hand side of r. The *essential r-graph* is the minimal oriented graph labeled with R-rules that contains r and in which the sons of each node are (exactly) its essential successors.

Lemma 2.5 Let r' be an essential successor of r. If u is an essential r-redex in t and $t \xrightarrow{u} s$, then s contains an essential r'-redex created by u.

Proof By Definition 2.7, the contractum e of u contains an essential r'-redex v created by u. By Lemma 2.1, $ES(e, s)$. Thus, by Lemma 2.3.(3), $ES(v, s)$.

Lemma 2.6 Let R be a PTRS, let $r \in R$, and let u be an r-redex. Then u is a semi-finite redex iff the essential r-graph does not contain an infinite path .

Proof ($\Rightarrow$) If the essential r-graph has an infinite path $r_0, r_1, ...$, then, by Lemma 2.5, we can construct an infinite essential reduction of trivial r-redex $v \xrightarrow{v} t_1 \xrightarrow{v_1} t_2 \xrightarrow{v_2} ...$, where v_1 is an essential r_1-redex created by v, v_2 is an essential r_2-redex created by v_1, and so on. ($\Leftarrow$) By induction on the height n of the essential r-graph (which is a finite tree). If $n = 0$, then the contractum s of the trivial r-redex v is a normal form. If $n > 1$, then by the induction assumption, each essential redex in s is semi-finite. Hence, by Lemma 2.4, s is normalizable. So v is normalizable, that is, u is semi-finite.

Definition 2.8 ([10]) A reduction P is called *outermost-fair* if P does not contain a term with an outermost redex that forever stays outermost and is never contracted.

We shall use the following theorem of Klop [10].

Theorem 2.2 ([10]) Outermost-fair reductions are normalizing in OTRSs.

Definition 2.9 Let R and R' be the OTRSs with the same alphabet. We call R and R' *syntactically equivalent* (written $R \approx R'$) when a term t has an R-normal form iff it has an R'-normal form and they coincide.

Lemma 2.7 Let $R = \{r_i : t_i \to s_i \mid i \in I\}$ be an OTRS, let $s_i \twoheadrightarrow e_i$ in R, and let $R' = \{r_i' : t \to e_i \mid i \in I\}$. Further, let u be an R-redex and let $u \xrightarrow{u} o$ in R'. Then $u \twoheadrightarrow o$ in R.

Proof Let u be an r_j-redex for some $j \in I$. Then the reduction obtained from the R-reduction $t_j \to s_j \twoheadrightarrow e_j$ by replacing variables with corresponding arguments of u is a reduction $u \twoheadrightarrow o$ in R.

Lemma 2.8 Let $R = \{r_i : t_i \to s_i \mid i \in I\}$ be an OTRS, let $s_i \twoheadrightarrow e_i$ in R, and let $R' = \{r_i' : t \to e_i \mid i \in I\}$. Further, let $P : t = t_0 \to t_1 \to \ldots$ be an outermost-fair reduction in R'. Then there is an outermost-fair reduction $Q : t_0 \twoheadrightarrow t_1 \twoheadrightarrow \ldots$ in R.

Proof By Lemma 2.7, there is a reduction $Q : t = s_0 \twoheadrightarrow s_{k_1} \twoheadrightarrow \ldots \twoheadrightarrow s_{k_i} \twoheadrightarrow \ldots$ in R, where $s_{k_i} = t_i$ ($i = 1, 2, \ldots$) and $Q_i : s_{k_i} \twoheadrightarrow s_{k_{i+1}}$ is constructed as in Lemma 2.7. Let $k_i \leq m < k_{i+1}$ and let u_m be an outermost redex in s_m. Since P is outermost-fair, no outermost residual of u_m in $s_{k_{i+1}}$ (if any) stays outermost in Q, i.e., Q is outermost-fair as well.

Definition 2.10 We call a semi-finite rule $r : t \to s$ in a PTRS R *irreducible* if s is in R-normal form. $IR(r) : t \to !s$ is the *irreducible rule corresponding to* r. We call R *irreducible* if all semi-finite rules in R are irreducible. The irreducible PTRS obtained from R by replacing all semi-finite rules with the corresponding irreducible rules is denoted by $IR(R)$.

Lemma 2.9 Let $R = \{r_i : t_i \to s_i \mid i \in I\}$ be an OTRS, let $s_i \twoheadrightarrow e_i$ in R, and let $R' = \{r'_i : t \to e_i \mid i \in I\}$. Then $R \approx R'$. In particular, $R \approx IR(R)$.
Proof From Lemma 2.8 and Theorem 2.2.

It is easy to see that, in the notation of Lemma 2.9, the equation relation $='$ induced by R' is a subset of the equation $=$ induced by R. The converse $=\subseteq='$ does not hold in general. Indeed, if $R = \{r_1 : a \to b, r_2 : b \to a\}$ and $R' = \{r'_1 : a \to a, r'_2 : b \to b\}$, then obviously $a = b$ but $a \neq' b$.

Theorem 2.3 Let R and R' be irreducible PTRSs with the same alphabet. Then $R \approx R'$ iff R and R' are identical except for the right-hand sides of strictly infinite rules.
Proof ($\Rightarrow$) Let $r : t \to s$ be a rule in R with s in R-normal form. Since $R \approx R'$, t has the same normal form s in R'. By Lemma 2.3.(1), t contains an essential R'-redex u. By Lemma 2.4, u is semi-finite in R'. Since $R \approx R'$, u is not in R-normal form and it contains an R-redex v. Thus the trivial R-redex t contains an R-redex v. By orthogonality of R, v must coincide with t, and hence $t = u = v$. So t is also an R'-redex. Let $t = C[e_1, \ldots, e_k]$, where $C[\ ]$ is the pattern of t in R', and let $o = C[x_1, \ldots, x_k]$. Since o is an R'-redex, we can prove by analogy that o is also an R-redex. Since t is a trivial R-redex, it follows from the orthogonality of R that $o = C[x_1, \ldots, x_k] = C[e_1, \ldots, e_k] = t$, i.e., t is a trivial R'-redex. Hence, $r \in R'$. Now consider a strictly infinite R-rule $r' : t' \to s'$. One can prove by analogy that t' is the left-hand side of an R'-rule and s' does not have an R'-normal form, since it does not have an R-normal form.

($\Leftarrow$) By Theorem 2.1, if a term t has an R-normal form, then an essential R-reduction $P : t \xrightarrow{u_0} t_1 \xrightarrow{u_1} \ldots \xrightarrow{u_{n-1}} t_n$ leads to its R-normal form t_n. By Lemma 2.4, u_i is semi-finite in R. Since the sets of semi-finite rules in R and R' coincide and the sets of R-redexes and R'-redexes are the same, P is also a normalizing R'-reduction. Thus if t has an R-normal form, then it has the same R'-normal form. Since the converse is also valid, we have $R \approx R'$.

3 Decidability of syntactic equivalence

In this section, we present an efficient decision procedure for syntactic equivalence of PTRSs with a finite number of rules. To make the algorithm efficient, we need an optimal normalizing strategy. Optimality of normalization in RPSs has been studied in [3, 13, 16]. To avoid duplication of redexes, Vuillemin [16] and Montangero et al. [13] introduced a sharing machinism, which leads to a graph implementation of RPSs. Berry and Lévy [3] formalized the sharing machinism by a family relation and suggested contracting all redexes in a family in parallel in one step. We show in [9] that contraction of innermost essential redexes leads to a normal form in the smallest number of rewrite steps. (This result also holds for orthogonal TRSs [8].) We use the innermost essential strategy for construction of irreducible rules.

Proposition 3.1 Let t be a term in a PTRS, and let corresponding rules of essential redexes of t be semi-finite and their essentiality indicators and corresponding irreducible rules be known. Then we can

(a) find all essential redexes in t, and

(b) find the normal form $!t$ of t using the innermost essential strategy

as follows.

Algorithm 3.1 **(a)** Mark all outermost redexes in t; in essential arguments of marked redexes mark all outermost redexes; and so on, as long as possible. Marked redexes are all essential redexes in t.

(b) Choose an innermost marked redex in t and reduce it using appropriate irreducible rule. Reduce the resulting term in the same way, and so on, as long as possible.

Proof sketch (a) By Lemma 2.3. (b) By Lemmas 2.1, 2.4, and 2.6.

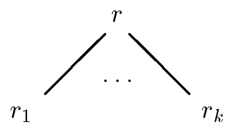
Proposition 3.2 Let $r : t \to s$ be a rule in a PTRS, let $r_1 \ldots r_k$ be all its essential successors, and let $IR(r_1), \ldots, IR(r_k)$ and $EI(r_1), ..., EI(r_k)$ be known. Then $IR(r)$, $EI(r)$, and $EI(IR(r))$ can be found as follows.

Algorithm 3.2 Using Algorithm 3.1, find $!s$. $IR(r) : t \to !s$ is the irreducible r-rule. If x_i is the i-th argument of t, then $i \in EI(r) = EI(IR(r))$ iff x_i occurs in $!s$.

Proof sketch $EI(r) = EI(IR(r))$ follows from Lemma 2.1. The rest follows from Proposition 3.1.

Definition 3.1 **(1)** Let t be a term in a PTRS R, let $F_R = \{r', ..., r^*\}$ be a set of R-rules, and let $EI(r'), \ldots, EI(r^*)$ be known. Define the F_R-*essential redexes* in t as follows. Mark all outermost redexes in t; in essential arguments of marked r'-$, \ldots, r^*$-redexes mark all outermost redexes; and so on, as long as possible. The marked redexes are all F_R-essential redexes in t.

(2) Let $r : t \to s \in R$. *A local r-figure* is a tree of the form



(1)

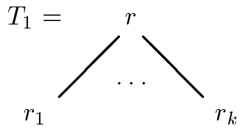
where $r_1, \ldots, r_k$ are (not necessarily all the) successors of r. (By "r' is a successors of r" we mean that an r'-redex has an occurrence in the right-hand side of r.) We call (1) the *initial local r-figure* if $r_1, \ldots, r_k$ are all rules such that r_1-,$\ldots, r_k$-redexes have outermost occurrences in s. We call (1) the *essential local r-figure* if $r_1, \ldots, r_k$ are all the essential successors of r. If F_R is a set of R-rules such that F_R-essential redexes in s are exactly the occurrences of r_1-,$\ldots, r_k$-redexes, then we call (1) the *local r-figure corresponding to F_R* or just the *local F_R-r-figure.*

Remark 3.1 Obviously, the initial local r-figure is the $\emptyset$-r-figure.

Lemma 3.1 Let r be a rule in a PTRS, F_R be a set of its successors, (1) be the local F_R-r-figure, and $\{r_1, ..., r_k\} \subset F_R$. Then (1) is the essential local r-figure.
Proof By Proposition 3.1.(a).

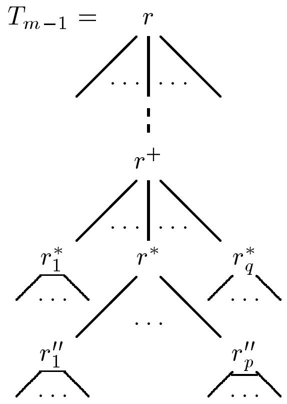
Proposition 3.3 Let R be a PTRS containing n rules and let $r \in R$. We can find a cycle in the essential r-graph if r is strictly infinite and otherwise we can construct the essential r-graph (which is a tree with the height less then n) and find irreducible rules and essentiality indicators for rules in the essential r-graph as follows.

Algorithm 3.3 First step: Construct the initial r-figure T_1:



If $k = 0$, then find $IR(r)$ (it coincides with r) and $EI(r)$ using Algorithm 3.2. The essential r-graph is constructed: it is just r and the algorithm stops. If $k \neq 0$ and some r_i coincides with r, then the algorithm stops by finding the cycle r, r_i. Otherwise proceed by performing the second step.

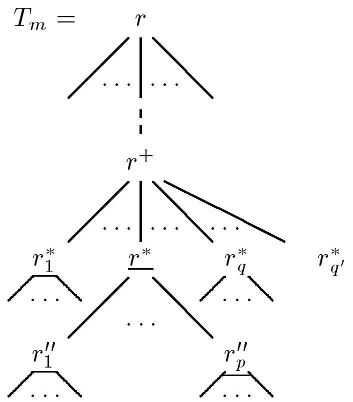
m^{th} step ($m > 1$): Suppose that the figure T_{m-1}, constructed in $(m-1)^{th}$ step, has the form:



Choose in T_{m-1} either any unmarked leaf r^* or any unmarked node r^* all successors of which are marked. If such a node does not exist, i.e., if each node in T_{m-1} is marked, then the algorithm stops; T_{m-1} (without marks) is the essential r-graph and essentiality indicators and irreducible rules for all rules in the essential r-graph are found. Otherwise:

(a) If r^* is a leaf and is not irreducible, then construct T_m from T_{m-1} by substituting the initial local r^*-figure for r^*.

(b) If r^* is not a leaf or is an irreducible leaf, then mark r^*, using Algorithm 3.2 find $IR(r^*)$ and $EI(r^*)$, and construct T_m from T_{m-1} by adding new sons $r^*_{q+1}, ..., r^*_{q'}$ of r^+



where the local r^+-figure in T_m corresponds to the set of all marked successors of r^+ in T_m.

If in T_m there is a cycle, then the algorithm stops. Otherwise proceed by performing the next step.

Proof It follows from Lemma 3.1 that, in the case (b) of the m^{th} step, $r''_1, \ldots r''_p$ are all essential successors of r^*. Hence Algorithm 3.2 is applicable and Algorithm 3.3 works. Since in each step of Algorithm 3.3

either new nodes are created or an unmarked node is marked, it terminates eventually by finding a cycle or by constructing the essential r-graph. Now the lemma follows from Lemma 2.6.

Proposition 3.4 Let t be a term in a PTRS with a finite number of rules. We can establish whether t is normalizable or not, and in the case of positive answer find its normal form $!t$ in the least number of steps, using the following algorithm:

Algorithm 3.4 Using Algorithm 3.3, for each rule for which a corresponding redex occurs in t establish whether it is semi-finite or not, and, if so, find corresponding irreducible rules and essentiality indicators. If at least one strictly infinite redex is not in an inessential argument of a semi-finite redex, then t is not normalizable and the algorithm stops. Otherwise replace in t all strictly infinite redexes by extra variables and, using Algorithm 3.1, find its normal form. It coincides with $!t$.

Proof sketch By Propositions 3.1 and 3.3, Corollary 2.1, and Lemmas 2.4 and 2.9.

Theorem 3.1 Let R and R' be PTRSs with the same alphabet containing a finite number of rules. Then it is decidable whether $R \approx R'$. A decision procedure is as follows.

Algorithm 3.5 For each rule $r : t \to s$ in R, establish, using Algorithm 3.4, whether s is normalizable or not and find the R-normal form of s if it exists. Replace all semi-finite rules in R by corresponding irreducible rules, constructing in this way $IR(R)$. Do the same for R'. Then $R \approx R'$ iff $IR(R)$ and $IR(R')$ are identical except for the right-hand sides of strictly infinite rules.

Proof By Theorem 2.3, Lemma 2.9, and Proposition 3.4.

4 Equivalence of RPSs with monadic basic functions

Definition 4.1 *A Recursive Program Scheme* (RPS) is a pair (R, t), where R is a TRS

$$R = \{f_i(x_1, \ldots, x_{n_i}) \to t_i \mid i = 1, \ldots m\}$$

and t is a term in the alphabet Σ of R. Σ consists of *unknown functions* $f_1, \ldots, f_m$, a set of *basic functions* (which may appear in right-hand sides of rules), and variables.

Recursive Program Schemes are usually defined as sets of equations

$$R = \{f_i(x_1, \ldots, x_{n_i}) = t_i \mid i = 1, \ldots m\}.$$

We chose the above definition since we are interested in the operational semantics of RPSs. We refer to [4] for an introduction to the subject. We only recall that the interpretations of RPSs are *continuous algebras*. A continuous algebra is a *partially ordered set* (having a least element and containing a limit of each increasing sequence) with interpretations of basic functions as *monotone* and *continuous* functions. The *value* $t_{R,D}$ of t in the continuous algebra D of finite and infinite trees (constructed from variables, basic function symbols, and a fresh constant ω) can be computed using the parallel-outermost strategy. The tree $t_{R,D}$ coincides with the normal form of t if t is normalizable. If t is not normalizable, then the parallel-outermost strategy constructs an infinite reduction $t \twoheadrightarrow t_1 \twoheadrightarrow \ldots$ and $t_{R,D}$ is the limit of the sequence of finite trees $t_\omega, t_{1\omega}, \ldots$ obtained from $t, t_1, \ldots$ by replacing all subterms headed by unknown functions with ω [4, 14]. It is easy to see that it is sufficient (and necessary) to take any outermost-fair reduction instead of parallel-outermost reductions. The *value* of t is defined analogously in any interpretation.

Two RPSs (R, t) and (R', s) (with the same alphabet) are called *equivalent* (written $R \equiv R'$) if the values of t and s coincide in every interpretation. It is well known [4] that (R, t) and (R', s) are equivalent iff the values $t_{R,D}$ and $s_{R',D}$ of t and s in the algebra D of finite and infinite trees coincide. Using this result, we now prove the decidability of equivalence between RPS with monadic basic functions.

Definition 4.2 We call a strictly infinite rule $r' : t' \to s'$ *irreducible* if s' contains only strictly infinite redexes. We call a PTRS R *strongly irreducible* if all of its rules are irreducible. The strongly irreducible PTRS obtained from R by replacing all rules with the corresponding irreducible rules is denoted by $SIR(R)$.

Lemma 4.1 It is decidable whether an equation

$$nx - my = l \tag{2}$$

with non-negative integers n, m, and l has a non-negative integer solution.

Proof sketch It is well known from Number Theory [11] that, for any equation

$$ax + by = c \tag{3}$$

with integers a, b, and c, if d is the greatest common divisor for a and b (written $d = gcd(a, b)$), then

(a) if c is not divisible by d, then (3) is not solvable in integers. Hence (2) is not solvable in non-negative integers.

(b) if c is divisible by d, then (3) can be rewritten in the form

$$a_1dx + b_1dy = c_1d$$

and then in the form

$$a_1x + b_1y = c_1$$

with $gcd(a_1, b_1) = 1$, and its integer solutions are the pairs

$$(x = (-1)^{l-1}c_1m_1 + b_1n,\ y = (-1)^l c_1m_2 - a_1n)$$

for some integers l, m_1, and m_2, and an arbitrary integer n. Since, in (2), $a_1 \geq 0$ and $b_1 \leq 0$, it follows that for sufficiently small negative n both x and y are positive, i.e., (2) has a non-negative integer solution.

Thus it is decidable whether (2) has a non-negative integer solution.

Lemma 4.2 Let $S = (R, t)$ and $S' = (R', s)$ be strongly irreducible RPSs with monadic basic functions. Then $S \equiv S'$ is decidable.

Proof By Proposition 3.4, it is decidable whether t and s are normalizable. If t and s have the same normal form, then $S \equiv S'$. If either t or s does not have a normal form, then $S \not\equiv S'$. So assume that t and s are not normalizable, and let $P : t = t_0 \xrightarrow{u_0} t_1 \xrightarrow{u_1} \ldots$ and $Q : t = s_0 \xrightarrow{v_0} s_1 \xrightarrow{v_1} \ldots$ be the outermost reductions. (Since the basic functions are monadic, there is exactly one outermost redex in any term t_i and s_j.) It follows from strong irreducibility of R and R' that all redexes executed in P or Q except u_0 and v_0 are generated by previous steps. Since R and R' have finite numbers of rules, there are leftmost pairs (u_p, u_q) and (v_k, v_l) of similar redexes in P and Q. Hence there are contexts $C[\,]$ and $C'[\,]$ such that $u_p \twoheadrightarrow C[u_q]$ and $v_k \twoheadrightarrow C'[v_l]$, where $C[\,]$ and $C'[\,]$ do not depend on arguments of u_q and v_k. Thus if $t_p = C_1[u_p]$ and $s_k = C_2[v_k]$, then $t_{R,D} = C_1[C[C[\ldots]]]$ and $s_{R',D} = C_2[C'[C'[\ldots]]]$. Let $C[\,] = g_1(g_2(\ldots(g_n[\,])\ldots))$ and $C'[\,] = g'_1(g'_2(\ldots(g'_m[\,])\ldots))$, where g_i and g'_j are basic functions. If the top-parts of $t_{R,D}$ and $s_{R',D}$ containing $C_1[\,]$ and $C_2[\,]$ do not coincide, then $t_{R,D} \neq s_{R',D}$. Otherwise, $t_{R,D} \neq s_{R',D}$ iff there are different functions $g_{n'}$ and $g'_{m'}$ that have occurrences in $t_{R,D}$ and $s_{R',D}$ at the same depth. This holds iff the equation $h(C_1) + nx + n' = h(C_2) + my + m'$ (where $h(C_1)$ and $h(C_2)$ are heights of $C_1[\,]$ and $C_2[\,]$ respectively) has a solution in non-negative integers. Hence, the lemma follows from Lemma 4.1.

Lemma 4.3 Let (R, t) be an RPS. Then $(R, t) \equiv (SIR(R), t)$.

Proof By Lemma 2.8, for any outermost-fair reduction $P : t \twoheadrightarrow t_1 \twoheadrightarrow t_2 \ldots$ in $SIR(R)$ there is an outermost-fair reduction $Q : t \twoheadrightarrow t_1 \twoheadrightarrow t_2 \ldots$ in R. Hence either P and Q end at the same term, or they are both infinite and $t_{R,D}$ and $t_{SIR(R),D}$ coincide with the limit of $t_\omega, t_{1\omega}, t_{2\omega}, \ldots$ Thus, $R \equiv SIR(R)$.

Theorem 4.1 Equivalence of RPSs with monadic basic functions is decidable.

Proof From Lemmas 4.2 and 4.3.

5 Other decidable properties of PTRSs

In this section, we give several criteria for weak and strong normalization in PTRSs that imply decidability of these properties both for particular terms and for rewriting systems. We also show that the reducibility problem is decidable in PTRSs: for any terms t and s, it is decidable whether $t \twoheadrightarrow s$ or not.

Lemma 5.1 Let $t = (t_1//s_1, \ldots, t_n//s_n)s$, where the subterms s_i and t_i are free in s and t, respectively ($i = 1, \ldots, n$). If P is a reduction starting from s in which infinitely many redexes are executed outside the descendants of $s_1, \ldots s_n$, then $P\|(t)$ is infinite.

Proof sketch From Definition 2.5.

Corollary 5.1 Let $t = (t_1//s_1, \ldots, t_n//s_n)s$, where $s_1, \ldots, s_n$ are free in s and $t_1, \ldots, t_n$ are free in t. If s is strongly normalizable and t_i is strongly normalizable for all $i = 1 \ldots n$, then t is strongly normalizable.

Definition 5.1 We call a redex u *finite* if its similar trivial redex is strongly normalizable and call *infinite* otherwise. We call a rule $r \in R$ *finite* if the trivial r-redex is finite and *infinite* otherwise.

Lemma 5.2 Let t be a term in a PTRS. Then t is strongly normalizable iff each redex in t is finite.

Proof ($\Rightarrow$) Obviously, any redex u in t is strongly normalizable. Thus, by Corollary 5.1, the trivial redex similar to u is strongly normalizable as well. ($\Leftarrow$) By induction on the number n of redexes in t. The case $n = 0$ is trivial. So let $n > 0$, let u be an innermost redex in t, and let $s = (x//u)t$. By the induction assumption, s is strongly normalizable. Since arguments of u are in normal form and u is a finite redex, Corollary 5.1 implies that u is strongly normalizable. Since $t = (u//x)s$, we have again by Corollary 5.1 that t is strongly normalizable.

Definition 5.2 Let R be a PTRS and $r \in R$.

(1) We call the *r-graph* the minimal oriented graph labeled with R-rules that contains r and in which the sons of each node are (exactly) its successors.

(2) We call R *hierarchical* (resp. *essentially hierarchical*) if there exists a function $f : R \to N$ such that if $r' \in R$ is a successor (resp. an essential successor) of some $r'' \in R$, then $f(r'') > f(r')$. We call such an f a *hierarchical* (resp. an *essentially hierarchical*) enumeration of R.

(3) A TRS is called *weakly* (resp. *strongly*) *normalizing* if each term in it is weakly (resp. strongly) normalizable.

Lemma 5.3 Let R be a PTRS, let $r \in R$, and let u be an r-redex. Then u is a finite redex iff the r-graph does not contain an infinite path.

Proof ($\Rightarrow$) If the r-graph has an infinite path $r_0, r_1, \ldots$, then one can construct an infinite reduction of the trivial r-redex $v \xrightarrow{v} t_1 \xrightarrow{v_1} t_2 \xrightarrow{v_2} \ldots$, where v_1 is an r_1-redex created by v, v_2 is an r_2-redex created by v_1, and so on. ($\Leftarrow$) By induction on the height n of the r-graph (which is a finite tree). If $n = 0$, then the contractum s of the trivial r-redex v is a normal form. If $n > 1$, then, by the induction assumption, each redex in s is finite. Hence, by Lemma 5.2, s is strongly normalizable. Thus, v is also strongly normalizable, i.e., u is finite.

Proposition 5.1 Strong normalization is a decidable property in PTRSs with a finite number of rules.

Proof From Lemmas 5.2 and 5.3.

Lemma 5.4 Let R be a PTRS with a finite number of rules. Then the following are equivalent:

(1) R is strongly normalizing;

(2) all graphs in R are acyclic;

(3) R is hierarchical;

(4) all redexes in R are finite.

Proof ((1)$\Rightarrow$(2)) As we have seen in the proof of Lemma 5.3, the existence of an infinite path would imply the existence of an infinite reduction.

((2)$\Rightarrow$(3)) For any $r \in R$, let $f(r)$ be the height of the r-graph. Then f is a hierarchical enumeration of R.

((3)$\Rightarrow$(4)) Let f be a hierarchical enumeration of R and let $r \in R$. We show by induction on $n = f(r)$ that the trivial r-redex v is strongly normalizable. If $n = 0$, then the contractum s of v is a normal form. If $n > 1$, then, by the induction assumption, each redex in s is finite. Hence, by Lemma 5.2, s is strongly normalizable. Thus v is strongly normalizable, i.e., any r-redex is finite.

((4)$\Rightarrow$(1)) By Lemma 5.2.

Corollary 5.2 It is decidable whether a PTRS R with a finite number of rules is strongly normalizing.

Lemma 5.5 Let R be a PTRS with a finite number of rules. Then the following are equivalent:

(1) R is weakly normalizing;

(2) all essential graphs in R are acyclic;

(3) R is essentially hierarchical;

(4) all redexes in R are semi-finite.

Proof ((1)$\Rightarrow$(2)) As we have seen in the proof of Lemma 2.6, the existence of an infinite essential path would imply the existence of an infinite essential reduction.

((2)$\Rightarrow$(3)) For any $r \in R$, let $f(r)$ be the height of the essential r-graph. Then f is an essentially hierarchical enumeration of R.

((3)$\Rightarrow$(4)) Let f be an essentially hierarchical enumeration of R and let $r \in R$. We show by induction on $n = f(r)$ that the trivial r-redex v is weakly normalizable. If $n = 0$, then by Lemma 2.3.(1) the contractum s of v is a normal form. If $n > 1$, then, by the induction assumption, each essential redex in s is semi-finite. Hence, by Lemma 2.4, s is weakly normalizable. Thus v is weakly normalizable, i.e., any r-redex is semi-finite.

((4)$\Rightarrow$(1)) By Lemma 2.4.

Corollary 5.3 It is decidable whether a PTRS R with a finite number of rules is weakly normalizing.

Theorem 5.1 Let t and s be any terms in a PTRS R with a finite number of rewrite rules. Then it is decidable whether $t \twoheadrightarrow s$ or not.

Proof First observe that $t \twoheadrightarrow s$ iff there is a term t' obtained from t by an outermost reduction such that the top-symbol of t' coincides with the top-symbol of s and $t' \twoheadrightarrow s$. Now we prove the theorem by induction on the length of s.

(1) Suppose first that t and s are not redexes. If the top-symbols of t and s coincide, then the decidability of $t \twoheadrightarrow s$ follows from the induction assumption. Otherwise t cannot be reduced to s.

(2) Let t be a redex and s a non-redex. Further, let $P : t \to t_1 \to \ldots \to t_n$ be an outermost reduction, where n is the number of rewrite rules in R. If at least one term t_i is not a redex and t_j is such a term with the minimum j, then $t \twoheadrightarrow s$ iff $t_j \twoheadrightarrow s$ and the decidability of $t_j \twoheadrightarrow s$ is established as in case (1). Otherwise, contraction of outermost redexes gives a "cyclic" reduction containing only the terms that are already in P. Hence t cannot be reduced to a non-redex, in particular to s.

(3) Let both t and s be redexes. If, among the terms in P, there is a redex similar to s, then we can use the induction assumption. If, among the terms in P, there is no redex similar to s but there is a non-redex t_k, then t cannot be reduced to s. Otherwise, as in the last case in (2), all terms in the infinite outermost reduction starting from t are redexes that are not similar to s. Hence t cannot be reduced to s.

Acknowledgements

I enjoyed discussions with H. Barendregt, P.-L. Curien, J. W. Klop, J.-J. Lévy, A. Middeldorp, G. Mints, Sh. Pkhakadze, Kh. Rukhaia, V. Sazonov, and T. Vepkhvadze. I also would like to thank G. Gonthier, B. Pierce, G. Tagviashvili, and K. Urbaitis for their help in the preparation of this paper.

References

[1] Ashcroft E., Manna Z., Pnueli A. Decidable properties of monadic functional schemes. In: Z. Kohavi, A. Paz, eds. Theory of Machines and Computations, Academic Press, New York, 1971, p. 3-18.

[2] Barendregt H.P. The Lambda Calculus, its Syntax and Semantics. North-Holland, 1984.

[3] Berry G., Lévy J.-J. Minimal and optimal computations of recursive programs. JACM 26, 1979, p.148-175.

[4] Courcelle B. Recursive Applicative Program Schemes. In: J. van Leeuwen ed. Handbook of Theoretical Computer Science, Chapter 9, vol.B, 1990, p. 459-492.

[5] Courcelle B., Gallier J. Decidable subclasses of the equivalence problem for recursive program schemes. Theoretical Informatics and Applications 21, 1987, p. 245-286.

[6] Dershowitz N., Jouannaud J.-P. Rewrite Systems. In: J. van Leeuwen ed. Handbook of Theoretical Computer Science, Chapter 6, vol. B, 1990, p. 243-320.

[7] Huet G., Lévy J.-J. Computations in Orthogonal Rewriting Systems. In: J.-L. Lassez and G. Plotkin, eds. Computational Logic, Essays in Honor of Alan Robinson, MIT Press, 1991.

[8] Khasidashvili Z. Minimal Normalizing Strategy for Orthogonal and Persistent Term Rewriting Systems (short version). Proceedings of I.Vekua Institute of Applied Mathematics of Tbilisi State University, vol. 36, 1990, p. 189-199.

[9] Khasidashvili Z. Decidable properties and optimal normalization in Persistent Term Rewriting Systems. Report 1837, INRIA, 1992.

[10] Klop J.W. Term Rewriting Systems. In: S. Abramsky, D. Gabbay, and T. Maibaum eds. Handbook of Logic in Computer Science, Oxford University Press, 1990.

[11] Knuth D. E. The Art of Computer Programming. vol. 2, Addison-Weslay, 1981.

[12] Maranget L. La stratégie paresseuse, These de l'Universite' de Paris VII, 1992.

[13] Montangero C., Pacini G., Turini F. Graph representation and computation rules for typeless recursive program schemes. Springer LNCS, 1974.

[14] Nivat M. On the interpretations of recursive programs schemes. In: Symposia Matematica, vol. 15, Rome, 1975, p. 225-281.

[15] Rosen B. Program equivalence and context-free grammars. J. Computer and System Sciences 11, 1975, 358-374.

[16] Vuillemin J. Proof techniques for recursive programs. Ph.D. Thesis, Stanford University, Stanford, 1973.