Termination of constructor systems using semantic unification

Thomas Arts and Hans Zantema
Utrecht University
Department of Computer Science
P.O. Box 80.089
3508 TB Utrecht
The Netherlands
E-mail: thomas@cs.ruu.nl

UU-CS-1995-17
May 1995

Utrecht University
Department of Computer Science
Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : + 31 - 30 - 531454

Termination of constructor systems using semantic unification

Thomas Arts and Hans Zantema
Utrecht University
Department of Computer Science
P.O. Box 80.089
3508 TB Utrecht
The Netherlands
E-mail: thomas@cs.ruu.nl

Technical Report UU-CS-1995-17
May 1995

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924–3275

Termination of constructor systems using semantic unification

Thomas Arts and Hans Zantema
Utrecht University
Department of Computer Science
P.O. Box 80.089
3508 TB Utrecht
The Netherlands
E-mail: thomas@cs.ruu.nl

Abstract

We present a new technique to prove termination of term rewrite systems, or more precise, constructor systems (CSs). In earlier work we introduced a transformation of well-moded logic programs into CSs, such that termination of the logic program follows from termination of the obtained CS. The technique to prove termination of CSs described in this paper is in particular suitable for, but not limited to, CSs that are obtained by this transformation of logic programs. Surprisingly, we need semantic unification in the technique. Thus, semantic unification can be used for giving termination proofs for logic programs. Parts of the technique can be automated very easily. Other parts can be automated for subclasses of CSs. An implementation is in progress that is able to prove termination of some CSs that are not simply terminating.

1. Introduction

There are several approaches to prove termination of logic programs, for a survey see [dSD93]. One of the approaches, introduced by M.R.K. Krishna Rao *et al.* [KKS91], is to transform the logic program into a term rewrite system (TRS) such that the termination property is preserved. More precisely, if the TRS terminates, then the original well-moded logic program is left-terminating. Other authors followed this approach and came up with transformations [GW92, CR93, AM93, AZ94, Mar94] suitable for proving termination of more logic programs. Most transformation algorithms transform the logic programs into constructor systems (CSs), a subclass of the TRSs. This paper describes a technique that is able to prove termination of CSs. The technique is in particular suitable for, but not limited to, those CSs that are obtained from the transformation algorithm described in [AZ94]. Although the technique is mainly developed for logic programs, it is not necessary to know anything about logic programs or the transformations of logic programs into CSs, to understand the technique presented in this paper.

As a typical example, let the constructor system $\mathcal{R}_1$ be

$$\begin{array}{lcl} f(x) & \to & g(h(x)) \\ g(h'(x)) & \to & f(x) \end{array}$$

and let $\mathcal{R}_0$ be a constructor system such that for some term t, $h(t)$ can be rewritten with $\mathcal{R}_0$ to a term $h'(s)$ for some term s. We are interested in the following typical kind of reductions

$$f(t) \to g(h(t)) \to^* g(h'(s)) \to f(s)$$

Reductions with this kind of cyclic behaviour only terminate whenever we can derive in some way that t decreases, *i.e.*, $t > s$ for some well-founded order $>$. Of course, termination of this kind of reductions and of the rewrite system $\mathcal{R}_1 \cup \mathcal{R}_0$ depends on the behaviour of $\mathcal{R}_0$. With our technique we formalize this observation. We give a systematical approach to describe 'in some way t decreases' by introducing an interpretation for the rewrite system $\mathcal{R}_0$, and to collect all the reductions with this kind of cyclic behaviour (in $\mathcal{R}_1$ as well as in $\mathcal{R}_0$). From this collection we can infer a number of conditions that have to be fulfilled by a suitable well-founded order on terms. Thus we translate the termination problem into finding a suitable order fulfilling some conditions. We prove soundness of this transformation, *i.e.*, termination of the CS follows from the existence of the required order.

The following logic program is a typical example of a well-moded logic program for which the technique is applicable.

1.1. EXAMPLE.

$$\begin{array}{lcl} append(nil, xs, xs) & & \\ append(cons(x, xs_1), xs_2, cons(x, ys)) & \leftarrow & append(xs_1, xs_2, ys) \\ & & \\ split(xs, nil, xs) & & \\ split(cons(x, xs), cons(x, ys_1), ys_2) & \leftarrow & split(xs, ys_1, ys_2) \\ & & \\ perm(nil, nil) & & \\ perm(xs, cons(y, ys)) & \leftarrow & split(xs, ys_1, cons(y, ys_2)), \\ & & append(ys_1, ys_2, zs), \\ & & perm(zs, ys) \end{array}$$

By representing the lists by their length, a standard approach, termination of this logic program can be proved completely automatically by the new technique. In [AZ94] we described an algorithm to transform well-moded logic programs into constructor systems and proved that termination of the logic program follows from termination of the obtained constructor system. Therefore, we concentrate on techniques to prove termination of constructor systems.

After the preliminaries in Section 2 we introduce in Section 3 an example to illustrate the technique. In Section 3.5 is proved that this technique is sound. In Section 4 we shortly discuss the problems that arise by an implementation of the technique.

2. Preliminaries

2.1. Term rewrite systems

In this section we summarize some preliminaries from term rewriting that we need in this paper.

2.1. DEFINITION. A *signature* is a set $\mathcal{F}$ of *function symbols*. Associated with every $f \in \mathcal{F}$ is a natural number denoting its arity, *i.e.*, the number of arguments it is supposed to have. The function symbols of arity 0 are called *constants*.

Let $\mathcal{F}$ be a signature and $\mathcal{V}$ a set of *variables* disjoint from $\mathcal{F}$. The set $\mathcal{T}(\mathcal{F},\mathcal{V})$ of *terms* built from $\mathcal{F}$ and $\mathcal{V}$ is the smallest set with the following two properties:

(i) every variable is a term,

(ii) if $f \in \mathcal{F}$ is an n-ary function symbol and $t_1,\ldots,t_n$ are terms then $f(t_1,\ldots,t_n)$ is a term.

If c is a constant then we write c to denote the term $c()$.

2.2. DEFINITION. A *rewrite rule* or *reduction rule* is a pair (l,r) of terms satisfying the following two constrains:

(i) the left-hand side l is not a variable,

(ii) the variables that occur in the right-hand side r also occur in l.

Rewrite rules (l,r) will henceforth be written as $l \to r$.

2.3. DEFINITION. A *term rewrite system* (TRS) is a pair $(\mathcal{F},\mathcal{R})$ consisting of a signature $\mathcal{F}$ and a set $\mathcal{R}$ of rewrite rules between terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$. A TRS is called *finite* if both $\mathcal{F}$ and $\mathcal{R}$ are finite.

Constructor systems are a subclass of TRSs.

2.4. DEFINITION (cf. [MT91], [Gra93]). A *constructor system* (CS for short) is a TRS $(\mathcal{F},\mathcal{R})$ with the property that $\mathcal{F}$ can be partitioned into disjoint sets $\mathcal{D}$ and $\mathcal{C}$ such that every left-hand side $f(t_1,\ldots,t_n)$ of a rewrite rule of $\mathcal{R}$ satisfies $f \in \mathcal{D}$ and $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{C},\mathcal{V})$. Function symbols in $\mathcal{D}$ are called *defined symbols* and these in $\mathcal{C}$ *constructor symbols* or *constructors*.

2.5. DEFINITION. The rewrite rules of a TRS $(\mathcal{F},\mathcal{R})$ inductively define a *rewrite relation* $\to_R$ on $\mathcal{T}(\mathcal{F},\mathcal{V})$ by

(i) If $l \to r$ is a rewrite rule, then $l^\sigma \to_R r^\sigma$ for every substitution σ,

(ii) If $f \in \mathcal{F}$ is a function symbol with arity n and $t_1,\ldots,t_n$ and t_k' are terms such that $t_k \to_R t_k'$, then $f(t_1,\ldots,t_k,\ldots,t_n) \to_R f(t_1,\ldots,t_k',\ldots,t_n)$.

A TRS R is called *terminating* if there exists no infinite reduction of the rewrite relation $\to_R$.

2.6. DEFINITION. For a set $\mathcal{F}$ of operation symbols $Emb(\mathcal{F})$ is defined to be the TRS consisting of all the rules $f(x_1,\ldots,x_n) \to x_i$ with $f \in \mathcal{F}$ and $i \in \{1,\ldots,n\}$. These rules are called the *embedding rules*.

A stronger notion of termination, called *simple termination*, can be defined. This definition is motivated by [Zan94].

2.7. DEFINITION. A TRS R over a set $\mathcal{F}$ of function symbols is called *simply terminating* if $R \cup Emb(\mathcal{F})$ is terminating.

A standard technique to prove termination of TRSs, of which several implementations exist, is called RPO (recursive path order). This technique is not applicable to all TRSs. For example it is not applicable to terminating TRSs that are not simply terminating. A direct consequence of the recursive path order (among others in [Der87, FZ94]) is the following theorem

2.8. PROPOSITION. *Let $\triangleright$ be a well-founded order on the signature of a TRS R. If for every rule $l \to r$ in R we have that $head(l) \triangleright f$ for all function symbols f that occur in r, then R is terminating.*

2.2. Semantic unification

Syntactic unification theory is concerned with the problem whether for two given terms t_1 and t_2 the equation $t_1 = t_2$ can be solved 'syntactically', *i.e.*, find a unifier σ such that $t_1^\sigma = t_2^\sigma$; this is a particular case of the problem to solve equations 'semantically', *i.e.*, modulo some equational theory E (for this reason semantical unification is also called E-unification). More precisely, in the presence of an equational theory E, and given an equation $t_1 = t_2$, we want to find unifiers σ such that $t_1^\sigma =_E t_2^\sigma$. So syntactical unification is semantical unification with empty E. Narrowing is a technique to solve equations $t_1 = t_2$ in the presence of an equational theory E. We will not discuss the technique, but refer to [Hul80, Klo92] for the basic principles of narrowing, and to [Sie89, HÖ9, Han94] for surveys in the area.

Let E be an equational theory and $t_1 = t_2$ an equation in the presence of this theory. A substitution σ is called an E-unifier if $t_1^\sigma =_E t_2^\sigma$. Just as for syntactic unification, there is also the notion of a most general unifier for semantical unification. However, there is no unique most general unifier in general. Normally a set of most general unifiers can be obtained.

What is important for this paper is that we need narrowing to find all possible unifiers σ that solve the equation, not just one solution, which is another important question. The set of all unifiers, or the complete set of E-unifiers, is recursively enumerable for any decidable theory E: just enumerate all substitutions and check if each one unifies the given terms, which is possible as E is decidable.

3. Descending chains, a technique to prove termination

In this section we present a new technique to prove termination of constructor systems. First we give a sketch of the technique and following a leading example, we give the formal definitions.
Consider the following rewrite system

$$\begin{array}{lcl} f(s(x)) & \to & k(x) \\ k(x) & \to & f(x) \end{array}$$

Analysing the reason why there is no infinite reduction starting with a term of the form $f(s(t))$, with t in normal form, we encounter that this term is reduced by repeatedly using the two rewrite rules. Every time both rules have been used, the argument has decreased. Eventually this has to come to an end. This very informal observation is made formal in this section. As a leading example we consider the constructor system $\mathcal{R}_{perm}$,

$$\begin{array}{lcl}
append(nil, xs) & \to & appout(xs) \\
append(cons(x, xs_1), xs_2) & \to & k_1(x, append(xs_1, xs_2)) \\
k_1(x, appout(ys)) & \to & appout(cons(x, ys)) \\
split(xs) & \to & spout(nil, xs) \\
split(cons(x, xs)) & \to & k_2(x, split(xs)) \\
k_2(x, spout(ys_1, ys_2)) & \to & spout(cons(x, ys_1), ys_2)
\end{array}$$

$$\begin{array}{lcl}
perm(nil) & \to & permout(nil) \\
perm(xs) & \to & k_3(split(xs)) \\
k_3(spout(ys_1, cons(y, ys_2))) & \to & k_4(y, append(ys_1, ys_2)) \\
k_4(y, appout(zs)) & \to & k_5(y, perm(zs)) \\
k_5(y, permout(ys)) & \to & permout(cons(y, ys))
\end{array}$$

which is obtained by applying the transformation as described in [AZ94] on the logic program of Example 1.1. The set of defined symbols of this constructor system is $\{append, k_1, split, k_2, perm, k_3, k_4, k_5\}$; all other function symbols are constructor symbols.

3.1. An equational theory

Consider the reduction of any term containing a subterm of the form $perm(xs^\sigma)$. This subterm allows the following reduction:

$$\begin{array}{l}
perm(xs^\sigma) \to k_3(split(xs^\sigma)) \\
\qquad\qquad \downarrow_* \\
k_3(spout(ys_1^\sigma, cons(y^\sigma, ys_2^\sigma))) \to k_4(y^\sigma, append(ys_1^\sigma, ys_2^\sigma)) \\
\qquad\qquad\qquad\qquad\qquad\qquad \downarrow_* \\
k_4(y^\sigma, appout(zs^\sigma)) \to k_5(y^\sigma, perm(zs^\sigma)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \| \\
perm(ys^\sigma) \to k_3(split(ys^\sigma))
\end{array}$$

By the knowledge of the behaviour of the logic program, we expect ys^σ to be less than xs^σ, in the sense that the list ys^σ is a list with one element less than the list xs^σ. In other words, we observe that after every following application of the rewrite rule $perm(xs) \to k_3(split(xs))$, the length of the list-argument has decreased.

The correctness of this observation depends on another implicit observation: we assume *append* and *split* to behave as they ought to do, *i.e.*, we assume that splitting a list results in two lists of which the sum of the lengths is the length of the original list. Thus, if $split(s)$ reduces to $spout(s_1, s_2)$, then $|s| = |s_1| + |s_2|$. There is an easy way to find out whether *append* and *split* do behave in the

appropriate manner, just define an equational theory that strokes with the expected behaviour. In particular we would like to have

$$\begin{array}{lcl} nil & = & 0 \\ cons(x, xs) & = & s(xs) \\ append(xs_1, xs_2) & = & xs_1 + xs_2 \\ appout(xs) & = & xs \\ split(ys) & = & ys \\ spout(ys_1, ys_2) & = & ys_1 + ys_2 \end{array}$$

Thereafter we have to check that *the rewrite system is contained in this equational theory*, more precise, for every rewrite rule of $\mathcal{R}_{perm}$ the left-hand side and the right-hand side have to be equal in the equational theory. By adding two more equations, *viz.* $k_1(x, xs) = s(xs)$ and $k_2(x, xs) = s(xs)$, this demand is fulfilled. Now terms equal their reduct in the theory. In this particular case, the length of the list is not changed by applying the rewrite rules. Thus, by rewriting a term of the form $append(t_1, t_2)$ we obtain a term t_3 such that $append(t_1, t_2)$ and t_3 are equal in the theory.

Since the eventual aim is to automate the technique, it should be stressed that finding these equational theories can not be done automatically in general. Later on we show that for a small subclass of CSs a kind of standard theories can be given.

In order to check whether $\mathcal{R}$ is contained in an equational theory E, one can perform E-unification. To perform as much as possible automatically, we want to have an effective method for this E-unification. Therefore, we demand that the equational theory can be described by a complete TRS $\mathcal{M}$, such that narrowing [Sie89, Klo92] suffices to check whether $\mathcal{R}$ is contained in the equational theory. Note that, although many efficient narrowing strategies exist, finding all E-unifiers of a given equation is in general undecidable. In Section 4.3 we shortly discuss this. We like to stress that we are only interested in normal E-unifiers (thus all terms are in normal form w.r.t. a rewrite system $\mathcal{M}$ that represents the theory); in the following we will always assume the E-unifiers to be normal. In practice the rewrite system $\mathcal{M}$ will be very uncomplicated and termination and confluence of this rewrite system can therefore be checked automatically. From now on we identify complete rewrite systems and the equational theory that is obtained by replacing the rewrite relations in the rewrite system by equalities.

The TRS $\mathcal{R}_{perm}$ is contained in the following complete TRS $\mathcal{M}$. Note that we are not interested in the *perm* related function symbols and therefore just require that these function symbols equal a constant.

$$\begin{array}{lcllcl} nil & \to & 0 & perm(xs) & \to & 0 \\ cons(x, xs) & \to & s(xs) & permout(xs) & \to & 0 \\ append(xs_1, xs_2) & \to & xs_1 + xs_2 & k_3(xs) & \to & 0 \\ appout(xs) & \to & xs & k_4(x, xs) & \to & 0 \\ split(xs) & \to & xs & k_5(x, xs) & \to & 0 \\ spout(xs_1, xs_2) & \to & xs_1 + xs_2 & 0 + y & \to & y \\ k_1(x, xs) & \to & s(xs) & s(x) + y & \to & s(x + y) \\ k_2(x, xs) & \to & s(xs) & & & \end{array}$$

3.2. Dependency pairs

We now abstract from the rewriting itself and concentrate on the possible rewrite rules that are concerned in the reduction of a term.

3.1. DEFINITION. Let $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ be a constructor system. If $f(t_1, \ldots, t_m) \to C[g(s_1, \ldots, s_n)]$ is a rewrite rule of $\mathcal{R}$ and $f, g \in \mathcal{D}$, then

$$\langle f(t_1, \ldots, t_m), g(s_1, \ldots, s_n) \rangle$$

is called a *dependency pair* (of $\mathcal{R}$).

We say that two dependency pairs $\langle s_1, t_1 \rangle$ and $\langle s_2, t_2 \rangle$ are equivalent, notation $\langle s_1, t_1 \rangle \sim \langle s_2, t_2 \rangle$, if there exists a renaming τ such that $s_1^\tau \equiv s_2$ and $t_1^\tau \equiv t_2$. We are interested in dependency pairs up to equivalence and when useful, we may assume, without loss of generality, that two dependency pairs have disjoint sets of variables. For $\mathcal{R}_{perm}$, we have the following dependency pairs:

$$\begin{array}{ll}
(1) & \langle append(cons(x, xs_1), xs_2), \quad k_1(x, append(xs_1, xs_2)) \rangle \\
(2) & \langle append(cons(x, xs_1), xs_2), \quad append(xs_1, xs_2) \rangle \\
(3) & \langle split(cons(x, xs)), \quad k_2(x, split(xs)) \rangle \\
(4) & \langle split(cons(x, xs)), \quad split(xs) \rangle \\
(5) & \langle perm(xs), \quad k_3(split(xs)) \rangle \\
(6) & \langle perm(xs), \quad split(xs) \rangle \\
(7) & \langle k_3(spout(ys_1, cons(y, ys_2))), \quad k_4(y, append(ys_1, ys_2)) \rangle \\
(8) & \langle k_3(spout(ys_1, cons(y, ys_2))), \quad append(ys_1, ys_2) \rangle \\
(9) & \langle k_4(y, appout(zs)), \quad k_5(y, perm(zs)) \rangle \\
(10) & \langle k_4(y, appout(zs)), \quad perm(zs) \rangle
\end{array}$$

3.3. Descending chains

3.2. DEFINITION. Let E be an equational theory, such that $\mathcal{R}$ is contained in E. A sequence $\langle s_1, t_1 \rangle \langle s_2, t_2 \rangle \ldots \langle s_n, t_n \rangle$ $(n > 1)$ of dependency pairs is called a *chain* w.r.t. E if

1. $\langle s_1, t_1 \rangle \sim \langle s_n, t_n \rangle$, *i.e.*, first and last dependency pair are equivalent, and
2. the root symbol of t_i equals the root symbol of s_{i+1} for all $1 \leq i < n$
3. there exists a E-unifier σ such that for all $1 \leq i < n$ the arguments of t_i^σ equal in the equational theory the arguments of s_{i+1}^σ; thus if $t_i = f_i(u_1, \ldots, u_k)$ and $s_{i+1} = f_i(v_1, \ldots, v_k)$, then $u_1^\sigma =_E v_1^\sigma, \ldots, u_k^\sigma =_E v_k^\sigma$

3.3. DEFINITION. Let $\langle s_1, t_1 \rangle \langle s_2, t_2 \rangle \ldots \langle s_n, t_n \rangle$ be a chain with s_n a renaming of s_1 and $s_1 = f(u_1, \ldots, u_m)$ and $s_n = f(\overline{u}_1, \ldots, \overline{u}_m)$. The chain is called *descending* with respect to an order $\sqsupset$ if for all E-unifiers σ of this chain

$$u_1^\sigma \downarrow_{\mathcal{M}} \sqsupset \overline{u}_1^\sigma \downarrow_{\mathcal{M}}, \ldots, u_m^\sigma \downarrow_{\mathcal{M}} \sqsupset \overline{u}_m^\sigma \downarrow_{\mathcal{M}}$$

Note that if the order $\sqsupset$ is closed under substitution, then it suffices to check the property for all most general E-unifiers of this chain. Moreover, we do not really need the E-unifiers itself, but are satisfied with a substitutions $\tau_1, \tau_2, \ldots$ such that any E-unifier σ is an instance of a τ_i and for all τ_i we have $s_1^{\tau_i} \sqsupset s_n^{\tau_i}$. Most comfortable is ofcourse a finit set of substitutions $\tau_1, \ldots, \tau_m$; checking is than easily automated.

With Theorem 3.6 we prove that if there exists an equational theory E such that $\mathcal{R}$ is contained in E, and a well-founded order $\sqsupset$, such that all chains w.r.t. E are descending w.r.t. $\sqsupset$, then termination of the rewrite system is guaranteed. Hence for proving termination of $\mathcal{R}_{perm}$ it remains to show that all chains w.r.t. $\mathcal{M}$, as given above, are descending. Referring to the above numbering of dependency pairs, we obtain the following chains

1. $(2)(2)$ is a chain,
2. $(4)(4)$ is a chain,
3. $(5)(7)(10)(5)$, $(7)(10)(5)(7)$ and $(10)(5)(7)(10)$ are chains.
4. any other chain is formed by taking a chain $(n_1)(n_2)\ldots(n_k)(n_1)$ and substitute the last dependency pair (n_1) by one of the chains above starting with (n_1).

Note that the dependency pairs (1), (3), (6), (8) and (9) can not occur in a chain. It is not hard to see that all chains are obtained as described above. Neither is it hard to see that if the chains mentioned in 1, 2 and 3 are descending, then all chains are descending. Thus, we try to prove that these five chains are descending. If we are looking for a well-founded order $\sqsupset$ closed under substitution, then not all, but only all most general $\mathcal{M}$-unifiers of a chain have to be checked. The chain

$$\begin{array}{rcl}(2)(2) & = & \langle append(cons(x, xs_1), xs_2), append(xs_1, xs_2)\rangle \\ & & \langle append(cons(y, ys_1), ys_2), append(ys_1, ys_2)\rangle\end{array}$$

has only $\mathcal{M}$-unifier that are instances of $\tau = \{xs_1 = s(ys_1), xs_2 = ys_2\}$. (Recall that we only consider normal substitutions w.r.t. $\mathcal{M}$). Thus, if there is an order $\sqsupset$ closed under substitution, that satisfies

$$append(s(s(ys_1)), ys_2) \sqsupset append(s(ys_1), ys_2),$$

then for all $\mathcal{M}$-unifiers the chain is descending. The $\mathcal{M}$-unifiers of the chain

$$\begin{array}{rcl}(5)(7)(10)(5) = & \langle perm(xs), k_3(split(xs))\rangle \\ & \langle k_3(spout(ys_1, cons(y, ys_2))), k_4(y, append(ys_1, ys_2))\rangle \\ & \langle k_4(y, appout(zs)), perm(zs)\rangle \\ & \langle perm(zs), k_3(split(zs))\rangle\end{array}$$

are all instances of $\sigma = \{xs = s(zs)\}$. Thus, $\sqsupset$ has to fulfil

$$perm(s(zs)) \sqsupset perm(zs).$$

In the same way, the other three chains result in the following three demands on the order $\sqsupset$

$$split(s(s(ys))) \sqsupset split(s(ys))$$
$$k_3(ys_1 + s(s(ys_2))) \sqsupset k_3(ys_1 + s(ys_2))$$
$$k_4(ys_1 + s(ys_2)) \sqsupset k_4(ys_1 + ys_2).$$

All demands can be fulfilled by choosing $\sqsupset$ as the the embedding order. Therefore, all the chains are descending and by Theorem 3.6, the TRS $\mathcal{R}_{perm}$ is terminating.

The technique as described above, following the example rewrite system $\mathcal{R}_{perm}$, can be summarized as follows

1. Find a complete term rewrite system $\mathcal{M}$ such that $\mathcal{R}$ is contained in $\mathcal{M}$.
2. Calculate all dependency pairs.
3. Form all chains, find all most general $\mathcal{M}$-unifiers of these chains.
4. Every most general $\mathcal{M}$-unifier σ for a chain $\langle s_1, t_1\rangle \ldots \langle \overline{s}_1, \overline{t}_1\rangle$ w.r.t. $\mathcal{M}$ determines a requirement $s_1^\sigma \sqsupset \overline{s}_1{}^\sigma$. Form a list of these requirements on the order $\sqsupset$.
5. Find a well-founded order that is closed under substitution that satisfies these requirements.

Termination of $\mathcal{R}$ then follows from Theorem 3.6. Before we present this theorem, we first introduce *semantic labelling*, as it is needed in the proof of the theorem.

3.4. Semantic labelling

Semantic labelling [Zan93] is a technique to transform a TRS, of which termination has to be proved, into a TRS that might be easier to prove terminating. The transformation is sound and complete with respect to termination, such that termination of the TRS may be concluded from termination of the transformed TRS.

A standard technique to prove termination of TRSs, of which several implementations exist, is called RPO (recursive path order). This technique is only applicable to a restricted class of TRSs. For example it is not applicable to terminating TRSs that are not simply terminating. Semantic labelling is able to overcome this deficiency, by transforming a TRS that can not be proved terminating by RPO into a TRS on which RPO is applicable. Since we are mainly interested in constructor systems, we describe semantic labelling in this section restricted to constructor systems. For a complete and more detailed description of the technique we refer to [Zan93]. For the reader who is already familiar with the technique, we can remark that we perform a self-labelling on all defined symbols.

Let $\mathcal{R} = (\mathcal{D}, \mathcal{C}, R)$ be a CS, over a signature $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$ and a set $\mathcal{V}$ of variable symbols, of which termination has to be proved. Let $\mathcal{M}$ be an $\mathcal{F}$-algebra consisting of a carrier set M and for every function symbol $f \in \mathcal{F}$ of arity n a function symbol $f_{\mathcal{M}} : M^n \to M$.

3.4. DEFINITION. For a valuation $\rho : \mathcal{V} \to M$ the term interpretation $[\![]\!]_\rho : \mathcal{T}(\mathcal{F},\mathcal{V}) \to M$ is defined inductively by

$$\begin{aligned} [\![x]\!]_\rho &= \rho(x), \\ [\![f(t_1,\ldots,t_n)]\!]_\rho &= f_{\mathcal{M}}([\![t_1]\!]_\rho,\ldots,[\![t_n]\!]_\rho) \end{aligned}$$

for $x \in \mathcal{V}, f \in \mathcal{F}, t_1,\ldots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{V})$.

Semantic labelling can be applied to transform the CS $\mathcal{R}$ into a CS $\overline{\mathcal{R}} = (\overline{\mathcal{D}},\mathcal{C},\overline{R})$, whenever the following demand is fulfilled (recall that $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$):

> There exists an $\mathcal{F}$-algebra $\mathcal{M}$ consisting of a carrier set M and for every function symbol $f \in \mathcal{F}$ of arity n a function symbol $f_{\mathcal{M}} : M^n \to M$, such that $\mathcal{M}$ is a model for $\mathcal{R}$, *i.e.*, $[\![l]\!]_\rho = [\![r]\!]_\rho$ for every rewrite rule $l \to r$ of R and for all valuations ρ.

One of the main difficulties in the technique is to find such a model. If this model $\mathcal{M}$ is given, then the semantic labelling transformation [1] is fixed by

- For every defined symbol $f \in \mathcal{D}$ of arity n we introduce a set of label symbols S_f consisting of all terms of the form $f(t_1,\ldots,t_n)$, where $t_1,\ldots,t_n$ are elements of M.
- A new set of defined symbols is defined by

$$\overline{\mathcal{D}} = \{f_s | f \in \mathcal{D}, s \in S_f\}$$

- A new signature $\overline{\mathcal{F}}$ is defined by $\overline{\mathcal{D}} \cup \mathcal{C}$.

Note that $\overline{\mathcal{F}}$ can be infinite, even if $\mathcal{F}$ is finite. We define a labelling of terms $\mathsf{lab} : \mathcal{T}(\mathcal{F},\mathcal{V}) \times M^{\mathcal{V}} \to \mathcal{T}(\overline{\mathcal{F}},\mathcal{V})$ inductively by

$$\begin{aligned} \mathsf{lab}(x,\rho) &= x, \\ \mathsf{lab}(f(t_1,\ldots,t_n),\rho) &= f_{f([\![t_1]\!]_\rho,\ldots,[\![t_n]\!]_\rho)}(\mathsf{lab}(t_1,\rho),\ldots,\mathsf{lab}(t_n,\rho)) \end{aligned}$$

for $x \in \mathcal{V}, \rho : \mathcal{V} \to M, f \in \mathcal{F}, t_1,\ldots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{V})$. We call $f([\![t_1]\!]_\rho,\ldots,[\![t_n]\!]_\rho)$ the label of the term $f(t_1,\ldots,t_n)$. Now $\overline{R}$ is defined to be the TRS over $\overline{\mathcal{F}}$ consisting of the rules $\mathsf{lab}(l,\rho) \to \mathsf{lab}(r,\rho)$ for all $\rho : \mathcal{V} \to M$ and all rules $l \to r$ of R. It is not hard to see that $\overline{R}$ is indeed a constructor system with $\overline{\mathcal{D}}$ as set of defined symbols and $\mathcal{C}$ as set of constructor symbols. The following proposition directly follows from the main result of semantic labelling (for a proof we refer to [Zan93]).

3.5. PROPOSITION. *Let $\mathcal{M}$ be a model for a CS $\mathcal{R}$ and let $\overline{\mathcal{R}}$ be defined as above. Then $\mathcal{R}$ is terminating if and only if $\overline{\mathcal{R}}$ is terminating.*

[1] We consider a special kind of semantic labelling, the general transformation is more complex and not fixed by a given model.

3.5. Soundness of the technique

Now we are ready to present the main result of this paper. We prove that the introduced technique is sound, *i.e.*, whenever this technique enables us to prove that all chains are descending, we may conclude that the constructor system is terminating.

3.6. THEOREM. *Let $(\mathcal{D},\mathcal{C},\mathcal{R})$ be a constructor system, $\mathcal{M}$ a complete TRS such that $\mathcal{R}$ is contained in $\mathcal{M}$, and $\sqsupset$ some well-founded order. If all chains w.r.t. $\mathcal{M}$ are descending w.r.t. $\sqsupset$, then $\mathcal{R}$ is terminating.*

The proof is based on semantic labelling. We construct a labelling on $\mathcal{R}$ such that the elements in the dependency pairs correspond to the labels of the defined symbols of R. Thus, if $append(cons(x,xs_1),xs_2) \to k_1(x,append(xs_1,xs_2))$ is a rewrite rule of $\mathcal{R}$ and E as given in 3.1 defines the model, then this rewrite rule is transformed by semantic labelling in all rewrite rules of the form

$$\begin{aligned} &append_{append(s(m_1),m_2)}(cons(x,xs_1),xs_2) \to \\ &\quad k_{1,k_1(m,m_1+m_2)}(x,append_{append(m_1,m_2)}(xs_1,xs_2)) \end{aligned}$$

with $m,m_1,m_2 \in M$. The dependency pairs corresponding to the labels of this rule

$$\begin{array}{ll} (1) & \langle append(s(xs_1),xs_2), k_1(x,xs_1+xs_2)\rangle \\ (2) & \langle append(s(xs_1),xs_2), append(xs_1,xs_2)\rangle \end{array}$$

The main observation is that if a term of the form $append_l(\ldots)$, where l is a label, has to be reduced, then it has to be reduced by a rewrite rule which exactly matches this label, *i.e.*, a rewrite rule of the form $append_l(t) \to \ldots$. If we can derive that the labels are descending, then the reduction can only be finite.

PROOF. We define an algebra $\mathcal{A}$ to consist of a carrier set $\mathcal{T}(\mathcal{F})$ and for every f in $\mathcal{D}\cup\mathcal{C}$ the interpretation $f_{\mathcal{A}}(x_1,\ldots,x_n) = f(x_1,\ldots,x_n)\downarrow_{\mathcal{M}}$. Since $\mathcal{M}$ is complete, this is well defined. By the semantic labelling, the transformed CS $\overline{\mathcal{R}}$ is hereby fixed. We will prove that the labelled constructor system $\overline{\mathcal{R}}$ is terminating. Hence by Proposition 3.5, $\mathcal{R}$ is terminating.

Define a relation $\rightsquigarrow$ on defined symbols, *i.e.*, elements of $\overline{\mathcal{D}}$, of the labelled CS $\overline{\mathcal{R}}$ as follows: $f_l \rightsquigarrow g_m$ if $f_l(t_1,\ldots,t_n) \to C[g_m(s_1,\ldots,s_k)]$ is a rewrite rule of the labelled CS. With this relation a precedence $\triangleright$ is defined as the transitive closure of $\rightsquigarrow$ together with $\overline{\mathcal{D}} \triangleright \mathcal{C}$ (*i.e.*, all constructor symbols are smaller than a defined symbol). If the precedence $\triangleright$ is well-founded, then termination of $\overline{\mathcal{R}}$ is proved by Proposition 2.8, since in that case the root symbol of the left-hand side of a rule is always larger than all symbols in the right-hand side of that rule. Hence, it suffices to prove well-foundedness of $\triangleright$. Assume $\triangleright$ is not well-founded, then there is an infinite sequence

$$d_{1,l_1} \rightsquigarrow d_{2,l_2} \rightsquigarrow d_{3,l_3} \rightsquigarrow \ldots$$

of labelled defined symbols. Note that $d_{1,l_1} \rightsquigarrow d_{2,l_2}$ means that there is a rewrite rule in $\overline{\mathcal{R}}$ of the form $d_{1,l_1}(\vec{t_1}) \to C[d_{2,l_2}(\vec{s_1})]$ and a ground substitution ρ such that $l_1 = d_1([\![\vec{t_1}]\!]_\rho)$ and $l_2 = d_2([\![\vec{s_1}]\!]_\rho)$. Thus,

- for every $d_{i,l_i} \rightsquigarrow d_{i+1,l_{i+1}}$ there is a rewrite rule in $\mathcal{R}$ of the form $d_i(\vec{t_i}) \to C[d_{i+1}(\vec{s_i})]$ that defines a dependency pair $\langle d_i(\vec{t_i}), d_{i+1}(\vec{s_i})\rangle$, and

- there exist ground substitutions ρ_i such that $l_i = d_i([\![\vec{t_i}]\!]_{\rho_i})$ and $l_{i+1} = d_{i+1}([\![\vec{s_i}]\!]_{\rho_i})$.

Hence there exist an infinite sequence of dependency pairs

$$\langle d_1(\vec{t_1}), d_2(\vec{s_1})\rangle\langle d_2(\vec{t_2}), d_3(\vec{s_2})\rangle\langle d_3(\vec{t_3}), d_4(\vec{s_3})\rangle\ldots$$

together with an infinite sequence of ground substitutions $\rho_1, \rho_2, \rho_3, \ldots$, such that

$$\begin{array}{rll} & l_1 & = d_1([\![\vec{t_1}]\!]_{\rho_1}) \\ d_2([\![\vec{s_1}]\!]_{\rho_1}) = & l_2 & = d_2([\![\vec{t_2}]\!]_{\rho_2}) \\ d_3([\![\vec{s_2}]\!]_{\rho_2}) = & l_3 & = d_3([\![\vec{t_3}]\!]_{\rho_3}) \\ & \vdots & \end{array}$$

Note also that all dependency pairs can be chosen in such a way that the variables of each pair are disjoint. Thus, there exists one infinite substitution $\rho = \rho_1 \circ \rho_2 \circ \rho_3 \circ \ldots$, which is a unifier that unifies all connected dependency pairs, hence

$$\begin{array}{lll} \vec{s_1}^{\rho} & =_{\mathcal{M}} & \vec{t_2}^{\rho} \\ \vec{s_2}^{\rho} & =_{\mathcal{M}} & \vec{t_3}^{\rho} \\ & \vdots & \end{array}$$

Since for a CS $\mathcal{R}$ only finitely many dependency pairs are defined, there is a dependency pair that occurs infinitely many times in the sequence. Every sequence of dependency pairs between two pairs that are equivalent is a chain. Since all chains are descending with respect to $\sqsupset$, we obtain an infinite sequence

$$d_i(\vec{u_1}^{\rho}) \sqsupset d_i(\vec{u_2}^{\rho}) \sqsupset d_i(\vec{u_3}^{\rho}) \sqsupset \ldots$$

which contradicts that $\sqsupset$ is well-founded.

4. Performing the technique automatically

There are three problems in performing the described technique automatically. Firstly, the technique is such that normally infinitely many chains are formed, such that checking whether all chains are descending is not easy. Secondly, we would like to construct automatically an equational theory E that can be described by a complete TRS $\mathcal{M}$, such that $\mathcal{R}$ is contained in $\mathcal{M}$. And lastly, we are interested in finding a complete set of E-unifiers of a given equation in some equational theory E. Narrowing is a technique to achieve this. Thus, narrowing has to result in a complete set of E-unifiers. There is, however, no algorithm that provides this task in general.

4.1. Minimal chains

Every constructor system consists of finitely many rules and every rewrite rule corresponds to finitely many dependency pairs. In general, infinitely many chains can be formed with this finite set of dependency pairs. Since we can not check whether infinite chains are descending, we need a finite criterion on which we can decide that all chains are descending. The best thing that we might achieve is a notion of minimal chain, such that there are only finitely many chains and if all minimal chains are descending, then all chains are descending. Unfortunately we did not find such a notion yet. However, we did find a finite criterion that is very useful in practice.

4.1. Definition.

- A *basic chain* is a chain in which every dependency pair occurs at most once (except for the first and last pair in the chain, these are always equivalent).
- We say that a chain is a *connection* of chains if the first dependency pair occurs within the chain (not as first or last dependency pair).

A dependency pair $\langle n \rangle$ is called *premated* if there are two dependency pairs $\langle n_1 \rangle$ and $\langle n_2 \rangle$, not necessarily different, such that if $\langle n \rangle$ occurs in a chain, then $\langle n_1 \rangle$ is the only dependency pair that may occur left of it and $\langle n_2 \rangle$ is the only dependency pair that may occur right of it.

To check whether all dependency pairs are premated, in general complete sets of most general E-unifiers must be obtained, which is an undecidable problem. However, in practice, we are often able to decide by syntactic unification whether a dependency pair may occur next to another. The following lemma is straightforward.

4.2. Lemma. *A connection of descending chains is descending.*

4.3. Lemma. *If all dependency pairs are premated, then all chains are connections of basic chains.*

Proof. Induction on the length of the chain. Assume a chain $\langle 1 \rangle \langle 2 \rangle \ldots \langle n \rangle \langle 1 \rangle$ is not a basic chain. Then one of the dependency pairs occurs more than once. If this is $\langle 1 \rangle$, then it trivially is a connection of two chains. By induction it is a connection of basic chains. If another dependency pair occurs twice, say $\langle k \rangle$, then the chain looks like

$$\langle 1 \rangle \langle 2 \rangle \ldots \langle k \rangle \langle k+1 \rangle \ldots \langle k+m \rangle \langle k \rangle \langle k+m+2 \rangle \ldots \langle 1 \rangle.$$

Without loss of generality we may assume that no $\langle k \rangle$ occurs in the sequence $\langle k+m+2 \rangle \ldots \langle 1 \rangle$. Since all dependency pairs are premated, $\langle k+1 \rangle$ equals $\langle k+m+2 \rangle$ *etc.* Hence in the sequence $\langle k+1 \rangle \ldots \langle k+m \rangle$, there occurs a dependency pair that equals $\langle 1 \rangle$. Thus, the chain is a connection of two shorter chains and therefore by induction a connection of basic chains.

With the above two lemmas we have

4.4. COROLLARY. *If all dependency pairs are premated and all basic chains are descending, then all chains are descending.*

Note that there are only finitely many basic chains. The test whether a dependency pair is premated is in practice carried out by excluding all dependency pairs that can not be adjacent to it. In the leading example, all dependency pairs are premated. Therefore, only the basic chains are taken into account.

4.2. Constructing an equational theory

By constructing an equational theory E, such that the CS $\mathcal{R}_{perm}$ is contained in this theory, we were not interested in the rules that define $perm$. We just assigned a constant to all the root symbols in left- and right-hand side. In particular with CSs that are obtained by translating a logic program, this method is very promising. These CSs can be seen as constructor systems with a special subset of constructor symbols, the *out*-symbols, like $appout$, $spout$ and $permout$. The CSs are such that the root symbol of the right-hand side of a rule is either a defined symbol or an *out*-symbol. If we have a hierarchical combination of two CSs $\mathcal{R}_0$ and $\mathcal{R}_1$ like this such that also the sets of *out*-symbols are disjoint, then constructing an equational theory can be done by assigning constants to the defined symbols and *out*-symbols of $\mathcal{R}_1$ and finding a satisfying theory for $\mathcal{R}_0$. This latter theory can for example be obtained by $\mathcal{R}_0$ itself, if termination and confluence for that smaller CS can be proved. One can also try to complete $\mathcal{R}_0$ by some completion algorithm and use the obtained TRS.

4.5. EXAMPLE. The CS

$$\begin{array}{lcl} h(x) & \to & s(x) \\ f(x,p(x)) & \to & g(h(x)) \\ g(s(s(x))) & \to & f(x,x) \\ f(x,x) & \to & f(x,p(x)) \end{array}$$

is a hierarchical combination of a CS $\mathcal{R}_0$ consisting of the first rule and a CS $\mathcal{R}_1$ consisting of the other rules. Since $\mathcal{R}_0$ is confluent and terminating, $\mathcal{R}_0 \cup \mathcal{R}_1$ is contained in the following TRS

$$\begin{array}{lcl} h(x) & \to & s(x) \\ f(x,y) & \to & C \\ g(x) & \to & C \end{array}$$

Finding the dependency pairs is easy. All dependency pairs are premated. With the embedding order we can show that all basic chains, and with Corollary 4.4 all chains, are descending. Hence, the constructor system is terminating.

4.3. Narrowing

Efficient methods based on narrowing strategies to solve systems of equations have been devised [Klo92, Sie89]. For the purpose of our technique we may

use any narrowing strategy. Although the equational theories that we consider are of a very specific form, we know no strategy that will always succeed in giving all most general E-unifiers for any equational theory E. Although this is a drawback, we still obtain that with this new technique some constructor systems can be proved terminating automatically, which could not be done automatically before.

5. Conclusion and further research

We presented a new technique for proving termination of constructor systems, in particular for those constructor systems that are obtained from the translation of logic programs. Since many parts of the technique can be automated, an implementation is in progress. The three problems of Section 4 have to be studied in more detail and better solutions have to be found. Although many improvements can be carried out on the implementation part, we are now able to prove termination automatically of constructor systems that can not be proved terminating automatically with existing standard methods.

References

[AM93] G. Aguzzi and U. Modigliani. Proving termination of logic programs by transforming them into equivalent term rewriting systems. *Proceedings of FST&TCS 13*, Lecture Notes in Computer Science(761):114–124, December 1993.

[AZ94] Thomas Arts and Hans Zantema. Termination of logic programs via labelled term rewrite systems. Technical Report UU-CS-1994-20, Utrecht University, PO box 80.089, 3508 TB Utrecht, May 1994.

[CR93] Maher Chtourou and Michaël Rusinowitch. Méthode transformationnelle pour la preuve de terminaison des programmes logiques. Unpublished manuscript in French, 1993.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1):69–116, 1987.

[dSD93] Danny de Schreye and Stefaan Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 12:1–199, 1993.

[FZ94] Maria Ferreira and Hans Zantema. Syntactical analysis of total termination. *Proceedings of ALP'94*, Lecture Notes in Computer Science(850):204–222, September 1994.

[Gra93] Bernhard Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. Technical Report SR-93-09, Universität Kaiserslautern, June 1993.

[GW92] Harald Ganzinger and Uwe Waldmann. Termination proofs of well-moded logic programs via conditional rewrite systems. *Proceedings of CTRS*, Lecture Notes in Computer Science(656):430–437, July 1992.

[Ḧ89] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 1989. Subseries of Lecture Notes in Computer Science.

[Han94] M. Hanus. The intergration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19-20:583–628, 1994.

[Hul80] J.M. Hullot. Canonical forms and unification. *5th International Conference on Automated Deduction*, Lecture Notes in Computer Science(87):318–334, 1980.

[KKS91] M.R.K. Krishna Rao, Deepak Kapur, and R.K. Shyamasundar. A transformational methodology for proving termination of logic programs. *Proceedings of CSL*, Lecture Notes in Computer Science(626):213–226, 1991.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, New York, 1992.

[Mar94] Massimo Marchiori. Logic programs as term rewrite systems. *Proceedings of ALP'94*, Lecture Notes in Computer Science(850):223–241, September 1994.

[MT91] Aart Middeldorp and Yoshihito Toyama. Completeness of combinations of constructor systems. *Proceedings of RTA-91*, Lecture Notes in Computer Science(488):188–199, April 1991.

[Sie89] J.H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7:207–274, 1989.

[Zan93] Hans Zantema. Termination of term rewriting by semantic labelling. Technical Report RUU-CS-93-24, Utrecht University, July 1993. Accepted for special issue on term rewriting of Fundamenta Informaticae.

[Zan94] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.