A technique for automatically proving termination of constructor systems

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

UU-CS-1995-32
October 1995

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

A technique for automatically proving termination of constructor systems

Thomas Arts
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-1995-32
October 1995

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

ISSN: 0924–3275

A technique for automatically proving termination of constructor systems

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

Abstract

A technique is described to prove termination of constructor systems (CSs) automatically. The technique consists of three major steps. First, determine the *dependency pairs* of a constructor system. Second, find an equational theory in which the constructor system is contained, and third, prove that no infinite chain w.r.t. the equational theory of these dependency pairs exists. The first step is easy and can be performed completely automatically. Here we first concentrate on the last step. We assume the equational theory given in the form of a complete TRS and present several general criteria on the syntax of the dependency pairs to prove that no infinite chain can exist with respect to the given equational theory. For these criteria no semantic unification is needed and they can be performed completely automatically. Second we demonstrate a technique to find a complete TRS automatically in case the CS that has to be proved terminating is of a special form. We combine all techniques to show how a CS introduced by R. Kennaway of almost 400 lines can be proved terminating completely automatically.

1. Introduction

Termination of term rewrite systems (TRSs), and also of constructor systems (CSs), is undecidable [HL78]. This means that there is no algorithm able to prove termination of an arbitrary CS. We can, however, construct algorithms, like implementations based on the recursive path order (rpo) or Knuth Bendix order (kbo), that are able to either confirm termination or end up with the message that the used technique is not suitable to prove termination. The technique presented in this paper can automatically prove termination of a wide class of CSs among CSs that are not simply terminating. In this sense the technique differs from all methods based on simplification orders, like rpo and kbo, which are not able to prove termination of CSs that are not simply terminating. The technique has been introduced in [AZ95a] as a general applicable technique with

a few remarks added to point out which parts can be implemented and which parts are undecidable. Now we focus on the parts that can be implemented.

Roughly speaking the technique exists of three steps. First, determine the dependency pairs of a constructor system. This can be performed completely automatically. Second, find an equational theory in which the constructor system is contained. Finding such an equational theory is undecidable in general. Heuristics can be given to find equational theories for specific kinds of CSs. These heuristics result in complete TRSs instead of a set of equations, such that checking whether the constructor system is contained in the equational theory can be performed by semantic unification, or better, reduction together with syntactic unification. And third, prove that no infinite chain (w.r.t. the equational theory) of these dependency pairs exists.

At first we concentrate on this last step. We assume the equational theory given in the form of a complete TRS. We present several general criteria on the dependency pairs to prove that no infinite chain can exist w.r.t. the given equational theory. In the general case, semantic unification is needed to find substitutions for which a sequence of dependency pairs is a chain. Semantic unification can be used since the equational theory is given in the form of a complete CS. Unfortunately, semantic unification is not always a terminating process itself. To be sure that the chains of dependency pairs technique is terminating, we developed criteria on the dependency pairs for which we do not need the semantic unification. These criteria do not always apply for a given CS, but when they are applicable, the CS can be proved terminating automatically. Thus, only for a subset of CSs for which the chains of dependency pairs technique is applicable, the technique is applicable automatically.

Second we focus on a technique to derive an equational theory automatically such that the criteria that we presented before can be fulfilled. As an example of the power of this method, we show how termination of a CS of almost 400 lines, introduced by R. Kennaway [Ken95], can be proved completely automatically with the presented techniques.

2. Chains of dependency pairs

To illustrate the definitions that we present in this section, we use the following constructor system as a leading example.

2.1. Example. The constructor system is a transformation (see [AZ95b]) of the logic program

$$
\begin{array}{lcl}
leq(0,0) & & \\
leq(0,s(y)) & & \\
leq(s(x),s(y)) & \leftarrow & leq(x,y).
\end{array}
$$

$$
\begin{array}{lcl}
split(x,nil,nil,nil) & & \\
split(x,cons(y,ys),cons(y,ys_1),ys_2) & \leftarrow & leq(y,x), split(x,ys,ys_1,ys_2) \\
split(x,cons(y,ys),ys_1,cons(y,ys_2)) & \leftarrow & leq(s(x),y), split(x,ys,ys_1,ys_2)
\end{array}
$$

that split a list in two lists, one list with elements greater than some given value and an other list with elements less than or equal to this value.

$$
\begin{array}{lcl}
leq(0,0) & \to & true \\
leq(0,s(y)) & \to & true \\
leq(s(x),s(y)) & \to & k_1(leq(x,y)) \\
k_1(true) & \to & true
\end{array}
$$

$$
\begin{array}{lcl}
split(x,nil) & \to & splitout(nil,nil) \\
split(x,cons(y,ys)) & \to & k_4(x,y,ys,leq(y,x)) \\
k_4(x,y,ys,true) & \to & k_5(y,split(x,ys)) \\
k_5(y,splitout(ys_1,ys_2)) & \to & splitout(cons(y,ys_1),ys_2) \\
split(x,cons(y,ys)) & \to & k_6(x,y,ys,leq(s(x),y)) \\
k_6(x,y,ys,true) & \to & k_7(y,split(x,ys)) \\
k_7(y,splitout(ys_1,ys_2)) & \to & splitout(ys_1,cons(y,ys_2))
\end{array}
$$

2.1. Dependency pairs

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

2.2. 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.

2.3. Example. For the constructor system as given in Example 2.1 this results in the following ten dependency pairs.

$$
\begin{array}{ll}
(1) & \langle leq(s(x),s(y)), k_1(leq(x,y))\rangle \\
(2) & \langle leq(s(x),s(y)), leq(x,y)\rangle \\
(3) & \langle split(x,cons(y,ys)), k_4(x,y,ys,leq(y,x))\rangle \\
(4) & \langle split(x,cons(y,ys)), leq(y,x)\rangle \\
(5) & \langle k_4(x,y,ys,true), k_5(y,split(x,ys))\rangle \\
(6) & \langle k_4(x,y,ys,true), split(x,ys)\rangle \\
(7) & \langle split(x,cons(y,ys)), k_6(x,y,ys,leq(s(x),y))\rangle \\
(8) & \langle split(x,cons(y,ys)), leq(s(x),y)\rangle \\
(9) & \langle k_6(x,y,ys,true), k_7(y,split(x,ys))\rangle \\
(10) & \langle k_6(x,y,ys,true), split(x,ys)\rangle
\end{array}
$$

Further on we define a chain to be a special sequence of dependency pairs in which we compare the arguments of the terms semantically. Checking whether two terms are semantically equivalent is called semantic unification. More precisely, in the presence of an equational theory $\mathcal{E}$, and given an equation $t_1 = t_2$, we want to find unifiers σ such that $t_1^\sigma =_\mathcal{E} t_2^\sigma$. If the equational theory $\mathcal{E}$ is given in the form of a complete TRS, then narrowing can be used as a technique to solve equations $t_1 = t_2$ in the presence of this equational theory. Although this paper aims to proof termination of TRSs by chains of dependency pairs technique *without* the use of semantic unification, we still have to choose our interpretations to be complete TRSs.

2.4. Definition. A TRS $\mathcal{R}$ is contained in a complete TRS $\mathcal{E}$ if $l{\downarrow_\mathcal{E}} = r{\downarrow_\mathcal{E}}$ for every rule $l \to r$ of $\mathcal{R}$.

2.2. Chains of dependency pairs

2.5. Definition. Let $\mathcal{E}$ be a complete TRS, such that $\mathcal{R}$ is contained in $\mathcal{E}$. A sequence of dependency pairs is called a *chain* w.r.t. $\mathcal{E}$ if there exists a $\mathcal{E}$-unifier σ such that for every two consecutive pairs $\langle s_i, t_i \rangle$ and $\langle s_{i+1}, t_{i+1} \rangle$

1. the root symbol of t_i equals the root symbol of s_{i+1}, and
2. 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 $u_1^\sigma =_\mathcal{E} v_1^\sigma, \ldots, u_k^\sigma =_\mathcal{E} v_k^\sigma$

We do not really need the $\mathcal{E}$-unifiers itself, but are satisfied with a substitutions $\tau_1, \tau_2, \ldots$ such that any $\mathcal{E}$-unifier σ is an instance of a τ_i. The main theorem in [AZ95a] states

2.6. Theorem. *Let $\mathcal{R}$ be a constructor system. If there exists a complete TRS $\mathcal{E}$ such that $\mathcal{R}$ is contained in $\mathcal{E}$, and no infinite chain of dependency pairs w.r.t. $\mathcal{E}$ exists, then $\mathcal{R}$ is terminating.*

Hence for proving termination of a constructor system $\mathcal{R}$ it remains to show that, w.r.t. a given equational theory $\mathcal{E}$, no infinite chain of dependency pairs exists.

In this paper we assume some complete TRS $\mathcal{E}$ to be given. There are several heuristics to find these TRSs automatically, but for arbitrary $\mathcal{R}$ it can be very hard to find a complete TRS in which $\mathcal{R}$ is contained.

3. Techniques to prove absence of infinite chains

Given the dependency pairs, we can draw a directed graph of which the nodes are labelled with the dependency pairs and there is an arrow between two dependency pairs whenever the right projection of one pair has the same root symbol as the left projection of the other pair.

3.1. DEFINITION. The *dependency graph* of a set of nodes N of dependency pairs is a directed graph $G = (N, A)$ where $[\langle s_1, t_1\rangle, \langle s_2, t_2\rangle] \in A$ iff the root symbol of t_1 and s_2 are equal.

An infinite chain of dependency pairs has to correspond with an infinite path in the dependency graph. A finite constructor system always results in finitely many dependency pairs, hence the dependency graph is finite. An infinite path in a finite graph implies existence of a cycle in this graph. Thus, for studying infinite chains, we can safely remove all nodes and arcs in the dependency graph that are not on a cycle.

3.2. EXAMPLE. The dependency graph obtained from the dependency pairs of Example 2.3 can be seen in Figure 1. Note that dependency pair (1), (5), and

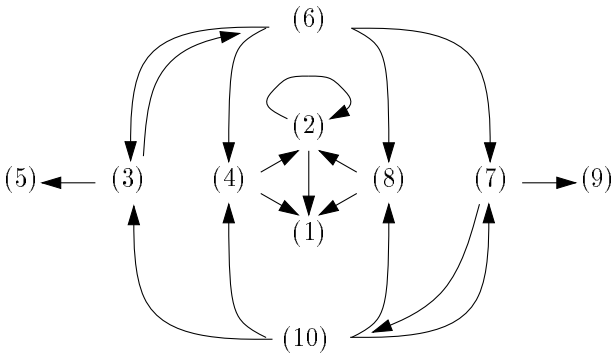

Figure 1: Dependency graph

(9) can never occur in an infinite chain, since none of the dependency pairs may be its direct adjacent. Also (4) and (8) can be removed from the set of dependency pairs. The dependency pairs that are on a cycle (Figure 2) are

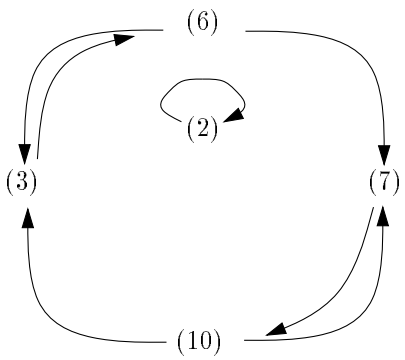

Figure 2: The cycles of the dependency graph

(2) $\langle leq(s(x), s(y)), leq(x, y)\rangle$
(3) $\langle split(x, cons(y, ys)), k_4(x, y, ys, leq(y, x))\rangle$
(6) $\langle k_4(x, y, ys, true), split(x, ys)\rangle$
(7) $\langle split(x, cons(y, ys)), k_6(x, y, ys, leq(s(x), y))\rangle$
(10) $\langle k_6(x, y, ys, true), split(x, ys)\rangle$

By removing nodes from the dependency graph, we might obtain several disconnected cycles. For every cycle, another technique to prove absence of infinite chains with the dependency pairs given by the nodes, may be used.

So far we only considered the root symbols of the terms in a dependency pair. Since a chain of dependency pairs is formed by considering both the root symbols and the arguments, we present in Section 3.2 a generalised form of the dependency graph that also takes the comparison of arguments into account. But, first we present a criterion for which the argument comparison is not necessary.

3.1. Without an equational theory

In some spare cases one can already see on the syntax of the dependency pairs that no infinite chain exists. In these cases, a complete TRS in which the constructor system is contained is not necessary. For example if there is no recursive call in the rewrite system, then every chain of dependency pairs will be finite, or, in other words, there are no cycles in the dependency graph. In this case termination of the rewrite system can also be proved by RPO.

A different criterion, for which we do not need a complete TRS in which the system is contained, is a reformulation of a theorem due to Richard Kennaway [Ken95]. The criterion does not prove that no infinite chain exists, but directly proves termination of the constructor system. The reformulation of this theorem is

3.3. Theorem. *Let $\mathcal{R}$ be a constructor system. If every dependency pair on a cycle in the dependency graph has a right projection that*

- *is linear,*
- *does not contain a defined symbol other than the root symbol, and*
- *is smaller than the left projection,*

then the constructor system is terminating.

In this lemma the left projection is compared to the right projection of a dependency pair w.r.t. the *size*.

In Section 4 we show that in some cases it is also possible to derive an equational theory automatically. This results in a completely automatic method for proving termination of CSs.

3.2. A complete TRS is given

From now on we assume that for the constructor system of which termination has to be proved a complete TRS $\mathcal{E}$, in which the constructor system is contained, is given. This complete TRS might be produced by some algorithm, or can be supplied by a user.

3.4. Example. The constructor system of Example 2.1 is contained in the following complete TRS $\mathcal{E}$.

$$\begin{array}{lcl} leq(x,y) & \to & C \\ k_1(x) & \to & C \\ true & \to & C \\ split(x,y) & \to & C \\ splitout(x,y) & \to & C \\ k_4(x,y,z,w) & \to & C \\ k_5(x,y) & \to & C \\ k_6(x,y,z,w) & \to & C \\ k_7(x,y) & \to & C \end{array}$$

This constructor system is terminating and confluent.

Whenever the given complete TRS is a constructor system itself, one can try to compare arguments of the left and right projection that are constructor terms with respect to this complete CS. Note that constructors of the system that has to be proved terminating need not be constructors in the given complete CS. In the example above, the symbol $true$ is a constructor of the CS of Example 2.1, but a defined symbol in $\mathcal{E}$.

Before we precisely define how we can compare arguments of the left and right projection, we first state an easy lemma.

3.5. Lemma. *Let $\mathcal{E}$ be a complete constructor system. If s and t are terms in which no defined symbol occurs and σ is a normalised substitution such that $s^\sigma =_\mathcal{E} t^\sigma$, then $s^\sigma = t^\sigma$.*

Proof. Since $\mathcal{E}$ is a complete CS, $s^\sigma =_\mathcal{E} t^\sigma$ implies $s^\sigma{\downarrow_\mathcal{E}} = t^\sigma{\downarrow_\mathcal{E}}$. Note that σ is a substitution of normal forms and no defined symbols occur in s or t. Thus, no redex occurs in σ and no redex occurs in t^σ or s^σ. In other words, s^σ and t^σ are normal forms. Thus, $s^\sigma = s^\sigma{\downarrow_\mathcal{E}} = t^\sigma{\downarrow_\mathcal{E}} = t^\sigma$. □

In Section 3 was remarked that dependency pairs that are not on a cycle of the dependency graph could safely be removed. In constructing the dependency graph, we only considered the root symbols of the two projections of the pair. Since a chain of dependency pairs is formed by considering both the root symbols and the arguments, we present a criteria, which takes the comparison of arguments into account, to safely remove arcs from the graph. There is a special function to indicate which arguments are important in this comparison.

3.6. DEFINITION. An *argument selector* arg is a function from a signature $\mathcal{F}$ to the natural numbers such that $1 \leq arg(f) \leq arity(f)$ for every function symbol $f \in \mathcal{F}$. Note that the argument selector is not defined for constants. The argument selector is naturally extended to non-variable terms in such a way that $arg(f(t_1, \ldots, t_n)) = t_{arg(f)}$. Thus, selecting an argument of the function f.

3.7. LEMMA. *Let $\mathcal{E}$ be a complete constructor system, arg an argument selector on the defined function symbols of $\mathcal{E}$. If $[\langle s_1, t_1\rangle, \langle s_2, t_2\rangle]$ is an arc of the dependency graph. such that*

- *in $arg(t_1)\downarrow_{\mathcal{E}}$ and $arg(s_2)\downarrow_{\mathcal{E}}$ no defined symbols occur, and*
- *$arg(t_1)\downarrow_{\mathcal{E}}$ does not unify with $arg(s_2)\downarrow_{\mathcal{E}}$,*

then removing this arc from the graph has no influence on the existence of an infinite chain of dependency pairs on a path of the graph.

PROOF. Assume $arg(t_1)^\sigma =_{\mathcal{E}} arg(s_2)^\sigma$ for some substitution σ (of normal forms). Note that $arg(t_1)^\sigma \rightarrow^* (arg(t_1)\downarrow_{\mathcal{E}})^\sigma$ and hence,

$$(arg(t_1)\downarrow_{\mathcal{E}})^\sigma =_{\mathcal{E}} (arg(s_2)\downarrow_{\mathcal{E}})^\sigma.$$

By Lemma 3.5 $(arg(t_1)\downarrow_{\mathcal{E}})^\sigma$ and $(arg(s_2)\downarrow_{\mathcal{E}})^\sigma$ are syntactically equivalent. Thus, σ is a unifier for $arg(t_1)\downarrow_{\mathcal{E}}$ and $arg(s_2)\downarrow_{\mathcal{E}}$, which contradicts the assumption that these two terms do not unify. □

3.8. EXAMPLE. Let $\mathcal{R} = f(s(x)) \rightarrow f(p(s(x)))$ be a CS of which termination has to be proved. Note that this CS is not simply terminating, thus we will not be able to prove termination with a simplification order like rpo. There is only one dependency, *viz.* $\langle f(s(x)), f(p(s(x)))\rangle$. A complete CS in which $\mathcal{R}$ is contained is $\mathcal{E} = f(x) \rightarrow C$. Thus, the symbols s and p are constructor symbols. Since $p(s(x))$ will never unify with $s(x)$, this dependency pair can not occur in an infinite chain. Since there are no other dependency pairs, no infinite chain exists and $\mathcal{R}$ is terminating.

If no unification is possible in a cycle of the dependency graph, then we may conclude that no infinite chain can be formed with the dependency pairs on this cycle. An infinite chain of the dependency pairs on a cycle can also be excluded if one of the arguments of a function decreases every time the whole cycle has been visited.

3.9. LEMMA. *Let $\mathcal{E}$ be a complete constructor system, arg an argument selector on the defined function symbols of $\mathcal{E}$, and $>$ a well-founded order on terms closed under substitution. If for every dependency pair $\langle s, t\rangle$ on a cycle of the dependency graph*

- *no defined symbols occur in $arg(s)\downarrow_{\mathcal{E}}$ or in $arg(t)\downarrow_{\mathcal{E}}$, and*
- *$arg(s)\downarrow_{\mathcal{E}} \geq arg(t)\downarrow_{\mathcal{E}}$.*

and there is a dependency pair $\langle s,t\rangle$ *in this cycle with* $arg(s)\downarrow_{\mathcal{E}} > arg(t)\downarrow_{\mathcal{E}}$*, then no infinite chain of dependency pairs on this cycle exists.*

PROOF. Assume there is an infinite chain of dependency pairs

$$\langle s_1,t_1\rangle\langle s_2,t_2\rangle\langle s_3,t_3\rangle\ldots$$

with (normalised) substitution σ. Thus, $arg(t_i)^\sigma =_{\mathcal{E}} arg(s_{i+1})^\sigma$. Note that $arg(t)^\sigma \to^* (arg(t)\downarrow_{\mathcal{E}})^\sigma$ and hence,

$$(arg(t_i)\downarrow_{\mathcal{E}})^\sigma =_{\mathcal{E}} (arg(s_{i+1})\downarrow_{\mathcal{E}})^\sigma.$$

By Lemma 3.5 $(arg(t_i)\downarrow_{\mathcal{E}})^\sigma$ and $(arg(s_{i+1})\downarrow_{\mathcal{E}})^\sigma$ are syntactically equivalent. Remark that for every i we have $arg(s_i)\downarrow_{\mathcal{E}} \geq arg(t_i)\downarrow_{\mathcal{E}}$ and since $>$ is closed under substitution,

$$(arg(s_i)\downarrow_{\mathcal{E}})^\sigma \geq (arg(t_i)\downarrow_{\mathcal{E}})^\sigma.$$

Hence, there exists an infinite sequence

$$(arg(s_1)\downarrow_{\mathcal{E}})^\sigma \geq (arg(s_2)\downarrow_{\mathcal{E}})^\sigma \geq (arg(s_3)\downarrow_{\mathcal{E}})^\sigma \geq \ldots$$

Add to this that there is a dependency pair $\langle s,t\rangle$ with $arg(s)\downarrow_{\mathcal{E}} > arg(t)\downarrow_{\mathcal{E}}$, and we conclude that if there is an infinite chain, then such a dependency pair will occur infinitely many times in that chain, in other words, $\geq$ will be infinitely many times strict in the above sequence. This contradicts the well-foundedness of $>$, hence, no infinite chain of dependency pairs on this cycle exists. □

In stead of an argument selector that selects one argument we can also use argument selectors that pick sequences of arguments and lift the order to an order on sequences of terms.

3.10. EXAMPLE. The following constructor system

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

can not be proved terminating by RPO, but can be proved terminating extremely easy with the chains of dependency pairs technique. The dependency pairs are

$$\begin{array}{ll} (1) & \langle f(s(x)),h(x)\rangle \\ (2) & \langle h(x),f(x)\rangle \end{array}$$

We like to have s as a constructor in the complete CS in which the system is contained. By assigning $arg(f) = arg(h) = 1$ and $>$ the embedding order, Lemma 3.9 proofs termination of the TRS. The complete CS is given by

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

3.11. EXAMPLE. Two separate cycles (Figure 2) occur in the dependency graph of the constructor system of Example 2.1. With Lemma 3.9 we can prove that no infinite chain exists consisting of dependency pairs on the cycles by choosing $arg(leq) = 1$, $arg(split) = 2$, and $arg(k_4) = arg(k_6) = 3$.

In the last two examples the arguments already were in normal form w.r.t. the given constructor system. In the following example, we essentially use that the arguments are reduced before they are compared.

3.12. EXAMPLE. The following logic program reverses a given list

$$\begin{array}{lcl}
rev1(x, nil, x) & & \\
rev1(x, cons(y,z), w) & \leftarrow & rev1(y,z,w) \\
rev(nil, nil) & & \\
rev(cons(x,y), cons(z_1,z_2)) & \leftarrow & rev1(x,y,z_1), rev2(x,y,z_2) \\
rev2(x, nil, nil) & & \\
rev2(x, cons(y,z), cons(z_1,z_2)) & \leftarrow & rev2(y,z,w_1), rev(w_1,w_2), \\
 & & rev1(x,w_2,z_1), rev2(x,w_2,z_2)
\end{array}$$

The transformation of this logic program into a constructor system results in the system $\mathcal{R}$:

$$\begin{array}{lcl}
rev1(x, nil) & \to & rev1out(x) \\
rev1(x, cons(y,z)) & \to & k_1(rev1(y,z)) \\
k_1(rev1out(w)) & \to & rev1out(w) \\
rev(nil) & \to & revout(nil) \\
rev(cons(x,y)) & \to & k_2(x,y,rev1(x,y)) \\
k_2(x,y,rev1out(z_1)) & \to & k_3(z_1, rev2(x,y)) \\
k_3(z_1, rev2out(z_2)) & \to & revout(cons(z_1,z_2)) \\
rev2(x, nil) & \to & rev2out(nil) \\
rev2(x, cons(y,z)) & \to & k_4(x, rev2(y,z)) \\
k_4(x, rev2out(w_1)) & \to & k_5(x, rev(w_1)) \\
k_5(x, revout(w_2)) & \to & k_6(x, w_2, rev1(x,w_2)) \\
k_6(x, w_2, rev1out(z_1)) & \to & k_7(z_1, rev2(x,w_2)) \\
k_7(z_1, rev2out(z_2)) & \to & rev2out(cons(z_1,z_2))
\end{array}$$

After determining the dependency pairs and drawing the dependency graph (Figure 3), we encounter several cyclic subgraphs consisting of the following dependency pairs

$$\begin{array}{ll}
(1) & \langle rev1(x, cons(y,z)), rev1(y,z) \rangle \\
(2) & \langle rev(cons(x,y)), k_2(x,y,rev1(x,y)) \rangle \\
(3) & \langle k_2(x,y,rev1out(z_1)), rev2(x,y) \rangle \\
(4) & \langle rev2(x, cons(y,z)), rev2(y,z) \rangle \\
(5) & \langle rev2(x, cons(y,z)), k_4(x, rev2(y,z)) \rangle \\
(6) & \langle k_4(x, rev2out(w_1)), rev(w_1) \rangle \\
(7) & \langle k_4(x, rev2out(w_1)), k_5(x, rev(w_1)) \rangle \\
(8) & \langle k_5(x, revout(w_2)), k_6(x, w_2, rev1(x,w_2)) \rangle \\
(9) & \langle k_6(x, w_2, rev1out(z_1)), rev2(x,w_2) \rangle
\end{array}$$

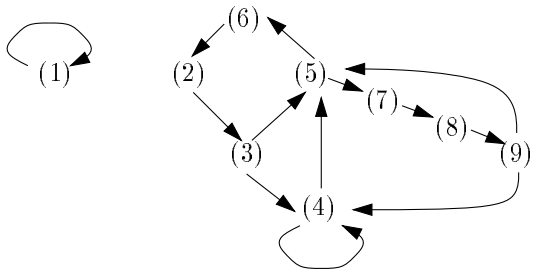

Figure 3: The cycles of the *rev* dependency graph

The cycles of the dependency graph are given in Figure 3. The following complete CS $\mathcal{E}$, in which $\mathcal{R}$ is contained, is assumed to be given

$$\begin{array}{lcl}
rev1(x) & \to & C \\
rev1out(x) & \to & C \\
k_1(x) & \to & x \\
rev(x) & \to & x \\
revout(x) & \to & x \\
rev2(x,y) & \to & y \\
rev2out(x) & \to & x \\
k_2(x,y,z) & \to & s(y) \\
k_3(x,y) & \to & s(y) \\
k_4(x,y) & \to & s(y) \\
k_5(x,y) & \to & s(y) \\
k_6(x,y,z) & \to & s(y) \\
k_7(x,y) & \to & s(y) \\
cons(x,y) & \to & s(y)
\end{array}$$

Note that *cons* is a defined symbol in $\mathcal{E}$, whereas it is a constructor symbol in $\mathcal{R}$. By choosing $arg(rev1) = 2$ dependency pair (1) can not occur in an infinite chain. Choose $arg(rev) = 1$, $arg(rev2) = 2$, $arg(k_2) = 2$, $arg(k_3) = 2$ and $arg(k_4) = arg(k_5) = arg(k_6) = 2$. For every dependency pair $\langle s, t\rangle$ we easily check that $arg(s)\!\downarrow_{\mathcal{E}} \geq arg(t)\!\downarrow_{\mathcal{E}}$. Moreover, in every minimal cycle of the dependency graph occurs either dependency pair (4) or dependency pair (5). Both (4) and (5) are strict dependency pairs, hence by Lemma 3.9 there exists no infinite chain with this set of dependency pairs. Thus, $\mathcal{R}$ is a terminating TRS.

3.3. Notes on an implementation

With the chains of dependency pairs technique we aim on a technique that can be used automatically. The lemmas given in Section 3 determine criteria that exclude existence of infinite chains. To perform things automatically, we like to check automatically whether one of the criteria hold.

Let $\mathcal{R}$ be a given constructor system of which termination has to be proven. For all criteria we first have to determine all dependency pairs and we have to

construct the dependency graph. All vertices in this dependency graph that are not on a cycle may be removed. Thus, we start with a dependency graph consisting of zero or more minimal cycles. First we check the following two criteria

1. If there are no cycles, then $\mathcal{R}$ is terminating.
2. If the conditions of Theorem 3.3 hold, then $\mathcal{R}$ is terminating. These conditions are easily checked automatically.

If these two criteria are not fulfilled, then we need a complete constructor system $\mathcal{E}$ in which $\mathcal{R}$ is contained. We assume $\mathcal{E}$ to be given and, moreover, we assume that $\mathcal{R}$ is contained in $\mathcal{E}$. We normalise the arguments of the left and right projection of every dependency pair on a cycle with respect to the constructor system $\mathcal{E}$. We construct all argument selectors, such that in the normalised selected arguments no defined symbol w.r.t. $\mathcal{E}$ occur. For every argument selector we check that all minimal cycles in the dependency graph whether meet the criterion of Lemma 3.7 or the criterion of Lemma 3.9. The first criterion is easily checked automatically by a syntactic unification algorithm, the second criterion is a little more involved. One could consider one fixed order $>$ on terms, like the embedding order. A more powerful approach is to combine all cycles that do not meet the first criterion, thus, all minimal cycles that have to meet the second criterion. This results in a list of conditions of the form $arg(s)\downarrow_{\mathcal{E}} \geq arg(t)\downarrow_{\mathcal{E}}$. We can easily remove all conditions from this list for which $arg(s)\downarrow_{\mathcal{E}} = arg(t)\downarrow_{\mathcal{E}}$, but we have to check that for every minimal cycle at least on condition has left. For the set of conditions left, we have to find a well-founded order that satisfies these conditions. We can try to find this order by replacing the relation $>$ by a rewrite relation $\rightarrow$ and prove termination of the so obtained rewrite system. Termination of $\mathcal{R}$ then follows from termination of the new obtained rewrite system. However, the so found order $>$ is a rewrite order, thus, closed under context and substitution. We only need an order closed under substitution.

4. Deriving a complete TRS

In the previous section we described a technique to conclude on syntactic criteria that no infinite chain of dependency pairs could exist. Thus, by this technique termination of a CS can be proved. To perform the technique on a CS we assumed a complete TRS in which this CS is contained to be given. Finding an arbitrary complete TRS in which a CS is contained, is for most CSs easy. Just map every function symbol to the same constant. However, with such a complete TRS, the technique to prove absence of infinite chains will not apply. Thus, we need to derive a complete TRS in which the CS is contained that bears enough information to apply the technique with. In this section we describe a technique to derive such a TRS. The main idea is to derive a complete TRS that is a constructor system itself and constructors of the original CS are also constructors in the derived complete CS. In spare cases it might even be enough to know the existence of a complete CS $\mathcal{E}$ such that the CS $\mathcal{R}$ that has

to be proved terminating is contained in $\mathcal{E}$ and all constructors of $\mathcal{R}$ are also constructors of $\mathcal{E}$, without knowing how $\mathcal{E}$ looks like.

4.1. EXAMPLE. In Example 3.11 we prove absence of an infinite chain of dependency pairs by choosing $arg(leq) = 1$, $arg(split) = 2$, and $arg(k_4) = arg(k_6) = 3$ and applying Lemma 3.9. If we had only given that there existed a CS $\mathcal{E}$ in which the CS of Example 2.1 is contained such that the constructors of both systems coincide, then we could still use the above reasoning to prove absence of an infinite chain of dependency pairs, since in the arguments that are compared, no defined symbols occur and hence these arguments are normal forms.

4.1. Mapping to constants

Let $\mathcal{R}$ be a CS. A complete TRS in which $\mathcal{R}$ is contained can always be obtained if $\mathcal{R}$ is non-collapsing.

4.2. LEMMA. *If $\mathcal{R}$ is a non-collapsing CS, then $\{f(x_1, \ldots, x_n) \to C | f \in S\}$, where S is the set of all root symbols of any left-hand side or right-hand side of a rule of $\mathcal{R}$ and C is a constant, is a complete TRS in which $\mathcal{R}$ is contained.*

In Example 2.1, 3.8 and 3.10, the complete TRS was obtained in this way. Thus, the CSs of Example 2.1, 3.8 and 3.10 can be proved terminating completely automatically by the chains of dependency pair technique, whereas the recursive path order does not suffice for this purpose.

Most constructor systems need a more complicated complete TRS in which they are contained. For example the constructor system $\mathcal{R} = \mathcal{K}_1 \cup \mathcal{K}_2$ in Appendix A results in only two dependency pairs, *viz.*

$$\langle inc(9 : x), inc(x)\rangle$$
$$\langle dec(0 : x), dec(x)\rangle.$$

Choosing the complete TRS as proposed by Lemma 4.2 with constant C say, then also $9 : x$ and $0 : x$ are reduced to this constant. It is not hard to see that with this complete TRS an infinite chain of dependency pairs *does* exist. Therefore, a more sophisticated complete TRS has to be constructed. The main idea is to construct a complete TRS in which the CS is contained such that all constructor symbols of the CS are also constructor symbols of the complete TRS, which implies that the complete TRS has to be a constructor system itself. If such a complete TRS can be constructed, then, for this particular CS, by Lemma 3.9 the CS is terminating.

4.2. Unattached sets of function symbols

Assume that for a given CS $\mathcal{R}$ the dependency pairs are such that there exists an argument selector arg on the defined symbols of $\mathcal{R}$ with for every dependency pair $\langle s, t\rangle$ on a cycle of the dependency graph

- no defined symbols occurs in $arg(s)$ or $arg(t)$,
- $arg(s) \geq arg(t)$

and there is a dependency pair $\langle s, t\rangle$ in every cycle with $arg(s) > arg(t)$. In this case Lemma 3.9 is applicable if there exists a complete TRS $\mathcal{E}$ in which the CS is contained, such that all constructor symbols of the CS are also constructor symbols of $\mathcal{E}$, which therefore also has to be a CS.

In this section we present a criteria that ensure existence of such a complete CS. Only a restricted class of all CSs will meet the criteria, for example the CS has to be non-overlapping. However, the criteria can easily be checked automatically. Moreover, a CS of almost 400 lines introduced by R. Kennaway in [Ken95], can be proved terminating completely automatically by using the criteria and the chains of dependency pair technique.

In fact it is a little more complicated. We present a transformation that transforms a CS $\mathcal{R}$ in a CS $\overline{\mathcal{R}}$ such that

- if $\mathcal{R}$ is non-overlapping, then $\overline{\mathcal{R}}$ is non-overlapping,
- the dependency pairs that occur on a cycle in the dependency graph are the same for both $\mathcal{R}$ and $\overline{\mathcal{R}}$,
- the constructors of $\mathcal{R}$ are constructors of $\overline{\mathcal{R}}$.
- $\mathcal{R}$ is innermost normalising, if $\overline{\mathcal{R}}$ is innermost normalising.

Thereafter, we show that the transformation is very suitable for constructing a complete CS $\mathcal{E}$ such that $\overline{\mathcal{R}}$ is contained in $\mathcal{E}$ and all constructors of $\overline{\mathcal{R}}$ are constructors of $\mathcal{E}$.

4.3. Definition. Let $\mathcal{R}$ be a CS. A subset F of the defined function symbols is called an *unattached* set for $\mathcal{R}$ if no $f \in F$ is a constant, and for all rewrite rules $l \to r$ of $\mathcal{R}$

- whenever a symbol f of F occurs in r, then the root symbol of l is an element of F.
- symbols of F do not occur nested in r, i.e., if $f(s_1, \ldots, s_k)$ occurs in r and $f \in F$, then no $g \in F$ occurs in $s_1, \ldots, s_k$.

We have chosen to demand that unattached symbol may not be a constant to prevent more case distinctions in the transformation.

4.4. Example. An unattached set of function symbols for the constructor system $\mathcal{K}_1 \cup \mathcal{K}_2$ in Appendix A is $\{dec, inc\}$.

4.5. Definition. We denote sequences of terms by $\langle t_1, \ldots, t_n\rangle$ and introduce the operation $\circ$ to concatenate sequences of terms, *viz.*

$$\langle t_1, \ldots, t_m\rangle \circ \langle s_1, \ldots, s_n\rangle = \langle t_1, \ldots, t_m, s_1, \ldots, s_n\rangle$$

4.6. Definition. Let F be a set of function symbols. A redex is called an F-redex if the root symbol is an element of F.

For the definitions below, $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ is a constructor system and $F \subseteq \mathcal{D}$ is an unattached set for $(\mathcal{D}, \mathcal{C}, \mathcal{R})$. We define a transformation on constructor systems that preserves the properties mentioned before. The functions INN, CAP, NR and NEWY are only defined on terms with non nested unattached symbols.

4.7. DEFINITION. The function INN applied on a term results in a sequence of all subterms in which either no element of F occurs, or the root symbol of the subterm is an element of F.

$$\begin{array}{lcl} \text{INN}(x) & = & \langle\rangle \\ \text{INN}(g(s_1,\ldots,s_k)) & = & \begin{cases} \langle g(s_1,\ldots,s_k)\rangle & \text{if no } f \in F \text{ in } s_1,\ldots,s_k \\ \text{INN}(s_1) \circ \ldots \circ \text{INN}(s_k) & \text{if some } f \in F \text{ in} \\ & s_1,\ldots,s_k \text{ and } g \notin F \end{cases} \end{array}$$

Note that if $g \in F$ then no $f \in F$ in $s_1,\ldots,s_k$ since F is an unattached set.

4.8. DEFINITION. The function NR applied on a term, results in the number of terms that are obtained by INN. Thus, if $\text{INN}(t) = \langle t_1,\ldots,t_n\rangle$, then $\text{NR}(t) = n$.

$$\begin{array}{lcl} \text{NR}(x) & = & 0 \\ \text{NR}(g(s_1,\ldots,s_k)) & = & \begin{cases} 1 & \text{if no } f \in F \text{ in } s_1,\ldots,s_k \\ \text{NR}(s_1) + \ldots + \text{NR}(s_k) & \text{if some } f \in F \text{ in} \\ & s_1,\ldots,s_k \text{ and } g \notin F \end{cases} \end{array}$$

4.9. DEFINITION. The function CAP applied on a term and a natural number n, replaces all subterms in the term that are extracted by the function INN by a fresh variable. The function NEWY applied on a term and the same natural number n, results in the list of all fresh variables that are substituted in the term by the function CAP, in case the substituted subterm had f as root symbol, then NEWY results for this subterm in $fout(y_n)$, where $fout$ is a new function symbol.

$$\begin{array}{lcl} \text{CAP}(x,n) & = & x \\ \text{CAP}(g(s_1,\ldots,s_k),n) & = & \begin{cases} y_n & \text{if no } f \in F \text{ in } s_1,\ldots,s_k \\ g(\text{CAP}(s_1,n_1),\ldots, & \text{if some } f \in F \text{ in} \\ \quad \text{CAP}(s_k,n_k)) & s_1,\ldots,s_k \text{ and } g \notin F \end{cases} \end{array}$$

$$\begin{array}{lcl} \text{NEWY}(x,n) & = & \langle\rangle \\ \text{NEWY}(g(s_1,\ldots,s_k),n) & = & \begin{cases} \langle y_n\rangle & \text{if no } f \in F \text{ in} \\ & s_1,\ldots,s_k \text{ and } g \notin F \\ \langle fout(y_n)\rangle & \text{if no } f \in F \text{ in} \\ & s_1,\ldots,s_k \text{ and } g \in F \\ \text{NEWY}(s_1,n_1) \circ \ldots & \text{if some } f \in F \text{ in} \\ \quad \circ \text{NEWY}(s_k,n_k) & s_1,\ldots,s_k \text{ and } g \notin F \end{cases} \end{array}$$

where all y_n are fresh variables and $n_i = n + \sum_{j=1}^{i-1} \text{NR}(s_j)$ for $1 \le i \le k$.

4.10. DEFINITION. The function TRANS applied on a rewrite rule results in a set of rewrite rules. $\text{TRANS}(l \to r) =$

$\{l \to r\}$	if no $f \in F$ in l
$\{l \to fout(r)\}$	if $f \in F$ root of l and no $g \in F$ in r
$\{l \to k_i(vars(r), \text{INN}(r)),$ $k_i(vars(r), \text{NEWY}(r,1)) \to fout(\text{CAP}(r,1))\}$	if $f \in F$ root of l and some $s \in F$ in r

where all k_i are new, fresh defined symbols. Define $\overline{\mathcal{R}} = \bigcup_{l \to r \in \mathcal{R}} \text{TRANS}(l \to r)$.

4.11. EXAMPLE. The transformation applied on the CS of $\mathcal{K}_1 \cup \mathcal{K}_2$ of Appendix A with unattached set $\{inc, dec\}$ results in $\mathcal{K}_1$ together with

$$\begin{array}{lcl} inc(END) & \to & incout(1:END) \\ inc(0:x) & \to & incout(1:x) \\ & \dots & \\ inc(9:x) & \to & k_1(x, inc(x)) \\ k_1(x, incout(y)) & \to & incout(0:y) \\ & \dots & \end{array}$$

From the transformation and the fact that F is an unattached set we see that all dependency pairs that are on a cycle in the dependency graph of $\mathcal{R}$ are on a cycle in the dependency graph of $\overline{\mathcal{R}}$ and vice versa. Now we prove that this transformation is innermost normalisation preserving.

4.12. LEMMA. *Let t, s be terms such that $t \to_{\mathcal{R}} s$. If s contains a term $f(s_1, \dots, s_k)$ that is not in normal form and $f \in F$, then in t occurs a term $g(t_1, \dots, t_n)$ that is not in normal form and $g \in F$.*

PROOF. The proof is by induction on the structure of t. Note that t can not be a variable, since a variable is a normal form. Thus, $t = h(u_1, \dots, u_m)$ and the induction hypothesis holds for all $u_1, \dots, u_m$. If $h \in F$ we are done. Assume $h \notin F$, then there are two possibilities

1. $u_i \to \tilde{u}_i$ for some $1 \leq i \leq m$ and $h(u_1, \dots, \tilde{u}_i, \dots, u_m) = s$. If the subterm $f(s_1, \dots, s_k)$ occurs in $\tilde{u}_i$, then by induction, a subterm $g(t_1, \dots, t_n)$ with $g \in F$ that is not in normal form occurs in u_i and hence in t. Otherwise the subterm $f(s_1, \dots, s_k)$ occurs in u_j for some $1 \leq j \leq m$ with $i \neq j$, since $h \notin F$.

2. there is a rewrite rule $h(v_1, \dots, v_m) \to r$ in $\mathcal{R}$ and a substitution σ, such that
$$h(u_1, \dots, u_m) = h(v_1^\sigma, \dots, v_m^\sigma) \to r^\sigma = s$$
The subterm $f(s_1, \dots, s_k)$ occurs in r^σ, but f does not occur in r by definition of an unattached set. Thus, there is an x in r with $\sigma(x) = f(s_1, \dots, s_k)$. Since, $var(r) \subseteq var(g(v_1, \dots, v_m))$, the variable x occurs in a v_i for some $1 \leq i \leq m$. Thus, $f(s_1, \dots, s_k)$ is a subterm of $u_i = v_i^\sigma$. $\square$

4.13. LEMMA. *If there exists an infinite innermost reduction starting with a term t in which infinitely many F-redexes are contracted, then there exists such a reduction starting with a subterm $f(t_1, \ldots, t_n)$ of t with $f \in F$ and $t_1, \ldots, t_n$ innermost normalising terms.*

PROOF. Assume there exists such a reduction starting with a term t. Let $g(t_1, \ldots, t_n)$ be the subterm of minimal depth starting such an infinite reduction, thus $t_1, \ldots, t_n$ are innermost normalising. Note that $g(t_1, \ldots, t_n) \stackrel{i}{\rightarrow} {}^+g(u_1, \ldots, u_n)$ with $u_1, \ldots, u_n$ normal forms and $g(u_1, \ldots, u_n)$ starts such an infinite reduction. In this reduction infinitely many F-redexes are contracted, hence in this reduction a term s occurs with a subterm $f(s_1, \ldots, s_k)$ with $f \in F$ that is not in normal form. By Lemma 4.12, in $g(u_1, \ldots, u_n)$ occurs a subterm $f(w_1, \ldots, w_k)$ with $f \in F$ that is not in normal form. Since $u_1, \ldots, u_n$ are in normal form, this subterm is the term $g(u_1, \ldots, u_n)$ and hence $g \in F$. □

4.14. LEMMA. *Let f and g be symbols of F and $u_1, \ldots, u_n$ terms in normal form. If $f(u_1, \ldots, u_n) \stackrel{i}{\rightarrow}_{\mathcal{R}} C[g(s_1, \ldots, s_k)]$ and $g(s_1, \ldots, s_k)$ starts an infinite innermost reduction in which infinitely many F-redexes are contracted, then $f(u_1, \ldots, u_n) \stackrel{i}{\rightarrow}_{\overline{\mathcal{R}}} \tilde{C}[g(s_1, \ldots, s_k)]$ and $s_1, \ldots, s_k$ are of the form $v_1^\sigma, \ldots, v_k^\sigma$ with no $f \in F$ occurs in $v_1, \ldots, v_k$ and σ is a substitution of normal forms.*

PROOF. Note that there is a rewrite rule $f(t_1, \ldots, t_n) \rightarrow r$ in $\mathcal{R}$ and a substitution σ such that $f(u_1, \ldots, u_n) = f(t_1^\sigma, \ldots, t_n^\sigma) \stackrel{i}{\rightarrow}_{\mathcal{R}} r^\sigma = C[g(s_1, \ldots, s_k)]$ Since $u_1, \ldots, u_n$ are normal forms, σ is a substitution of normal forms. The redex $g(s_1, \ldots, s_k)$ occurs in r^σ and no redex occurs in σ, thus a term $g(v_1, \ldots, v_k)$ occurs in r, such that $g(v_1, \ldots, v_k)^\sigma = g(s_1, \ldots, s_k)$. By definition of an unattached set, no $f \in F$ occurs in $v_1, \ldots, v_k$. By definition of the transformation, the rewrite rule $f(t_1, \ldots, t_n) \rightarrow k_j(\ldots, g(v_1, \ldots, v_k), \ldots)$ occurs in $\overline{\mathcal{R}}$. Thus,

$$\begin{array}{ll} f(u_1, \ldots, u_n) = & \\ f(t_1^\sigma, \ldots, t_n^\sigma) & \stackrel{i}{\rightarrow}_{\overline{\mathcal{R}}} \quad k_j(\ldots, g(v_1, \ldots, v_k), \ldots)^\sigma \\ & \qquad = \tilde{C}[g(s_1, \ldots, s_k)] \end{array}$$

and $s_1, \ldots, s_k$ are of the form $v_1^\sigma, \ldots, v_k^\sigma$. □

4.15. LEMMA. *Let s be a term in which no $f \in F$ occurs and let σ be a substitution of normal forms. If $s^\sigma \stackrel{i}{\rightarrow}_{\mathcal{R}} t$, then $s^\sigma \stackrel{i}{\rightarrow}_{\overline{\mathcal{R}}} t$ and $t = u^\tau$ for some term u in which no $f \in F$ occurs and substitution σ of normal forms such that $\sigma(x) = \tau(x)$ for all x in $Dom(\sigma) \cap Dom(\tau)$.*

PROOF. The proof is by induction on the structure of s. Assume $s = g(s_1, \ldots, s_k)$ and for $s_1, \ldots, s_k$ the induction hypothesis holds. Since no $f \in F$ occurs in s, $g \notin F$. Distinguish two possibilities

1. $s_i^\sigma \stackrel{i}{\rightarrow}_{\mathcal{R}} v$ for some $1 \leq i \leq k$ and $g(s_1^\sigma, \ldots, v, \ldots, s_k^\sigma) = t$.
2. All terms $s_1, \ldots, s_k$ are in normal form and a rewrite rule $g(t_1, \ldots, t_n) \rightarrow u$ in $\mathcal{R}$ and a substitution τ exist such that $g(s_1^\sigma, \ldots, s_k^\sigma) = g(s_1^\tau, \ldots, s_k^\tau) \rightarrow_{\mathcal{R}} u^\tau = t$. □

4.16. THEOREM. *Let $\mathcal{R}$ be a CS and F be an unattached set of function symbols in $\mathcal{R}$. If the transformation $\overline{\mathcal{R}}$ is innermost normalising, then $\mathcal{R}$ is innermost normalising.*

PROOF. Assume there is an infinite innermost reduction in $\mathcal{R}$. If only finitely many times an F-redex is contracted in this reduction, then the tail of the reduction is a reduction in $\overline{\mathcal{R}}$. Assume in the infinite innermost reduction infinitely many times an F-redex is contracted. By Lemma 4.13 there exists such a reduction starting with a term $f(t_1,\ldots,t_n)$ with $f \in F$ and $t_1,\ldots,t_n$ innermost normalising terms. Thus, $f(t_1,\ldots,t_n) \stackrel{i}{\rightarrow}^+_{\mathcal{R}} f(u_1,\ldots,u_n)$ with $u_1,\ldots,u_n$ in normal form and $f(u_1,\ldots,u_n)$ starts such a reduction.

By Lemma 4.13, $f(u_1,\ldots,u_n) \stackrel{i}{\rightarrow}_{\mathcal{R}} C[g(s_1,\ldots,s_k)]$ with $g \in F$, $s_1,\ldots,s_k$ innermost normalising and $g(s_1,\ldots,s_k)$ starts an infinite innermost reduction in which infinitely many F-redexes are contracted. By Lemma 4.14, $f(u_1,\ldots,u_n) \stackrel{i}{\rightarrow}_{\overline{\mathcal{R}}} \tilde{C}[g(s_1,\ldots,s_k)]$ and $s_1 = v_1^\sigma,\ldots,s_k = v_k^\sigma$ for $s_1,\ldots,s_k$ without $f \in F$ and σ a substitution of normal forms. Thus, $g(s_1,\ldots,s_k) = g(v_1^\sigma,\ldots,v_k^\sigma) \stackrel{i}{\rightarrow}^+_{\mathcal{R}} g(\tilde{u}_1,\ldots,\tilde{u}_k)$ with $\tilde{u}_1,\ldots,\tilde{u}_k$ normal forms and $g(\tilde{u}_1,\ldots,\tilde{u}_k)$ starts such a reduction. By Lemma 4.15, $g(s_1,\ldots,s_k) \stackrel{i}{\rightarrow}^+_{\overline{\mathcal{R}}} g(\tilde{u}_1,\ldots,\tilde{u}_k)$. Thus, by repeating this argument, we obtain an infinite innermost reduction in $\overline{\mathcal{R}}$, which contradicts that $\overline{\mathcal{R}}$ is innermost normalising. □

The main advantage of translating $\mathcal{R}$ into $\overline{\mathcal{R}}$ is the construction of the complete CS that has to be found to apply the chains of dependency pairs technique. Finding a such a complete CS is easier for $\overline{\mathcal{R}}$ then it is for $\mathcal{R}$ itself. Just choose one constant for all symbols of F, $Fout$ and the k-symbols. The CS $\overline{\mathcal{R}}$ is split in two parts, $\mathcal{R}_0 = \{l \rightarrow r | \text{root symbol of l in } F\}$ and $\overline{\mathcal{R}} \setminus \mathcal{R}_0$. The problem of finding a complete CS in which $\overline{\mathcal{R}}$ is contained is now reduced to finding a complete CS in which $\mathcal{R}_0$ is contained, which is also a part of $\mathcal{R}$. Thus, we eliminated a part of the CS that can perform problems in finding a complete CS in which the CS is contained.

The following corollary links the power of all methods described in this paper. Note that all conditions of this corollary, except for innermost normalisation of $\mathcal{R}_0$, can be checked automatically.

4.17. COROLLARY. *Let $\mathcal{R}$ be a CS and F an unattached set for $\mathcal{R}$. Define $\mathcal{R}_0 = \{l \rightarrow r | \text{no } f \in F \text{ occurs in } l, r\}$. The CS $\mathcal{R}$ is complete if*

- *$\mathcal{R}$ is non-overlapping,*
- *$\mathcal{R}_0$ is innermost normalising,*
- *there exists an argument selector arg on the defined symbols of $\mathcal{R}$ with for every dependency pair $\langle s,t\rangle$ on a cycle of the dependency graph*
  - *no defined symbols occurs in $arg(s)$ or $arg(t)$,*
  - *$arg(s) \geq arg(t)$*

  and there is a dependency pair $\langle s,t\rangle$ in every cycle with $arg(s) > arg(t)$.

PROOF. Assume the demands of the corollary are fulfilled. Define $\overline{\mathcal{R}}$ as above. Note that dependency pairs on a cycle of the dependency graph of $\mathcal{R}$ coincide with the pairs on a cycle of the dependency graph of $\overline{\mathcal{R}}$. By Lemma 3.9, termination of $\overline{\mathcal{R}}$ follows from the prove that there exists a complete CS $\mathcal{E}$ such that $\overline{\mathcal{R}}$ is contained in $\mathcal{E}$ and all constructors of $\overline{\mathcal{R}}$ are also constructors of $\mathcal{E}$. Since $\mathcal{R}_0$ is innermost normalising and non-overlapping, it is also complete ([Gra95]). Thus,

$$\mathcal{E} = \mathcal{R}_0 \cup \{f(x_1, \ldots, x_k) \to c | f \in F \cup Fout \cup k\text{-symbols}\}$$

is a complete CS in which $\overline{\mathcal{R}}$ is contained. By the transformation, all constructor symbols of $\overline{\mathcal{R}}$ are also constructor symbols of $\mathcal{E}$, hence $\overline{\mathcal{R}}$ is terminating. The transformation is innermost normalisation preserving (Lemma 4.16), thus $\mathcal{R}$ is innermost normalising. By using [Gra95] and the assumption that $\mathcal{R}$ is non-overlapping, we derive that $\mathcal{R}$ is complete. □

4.18. EXAMPLE. The CS $\mathcal{K}_1$ is proved terminating automatically by absence of cycles in the dependency graph.

For the CS $\mathcal{K}_1 \cup \mathcal{K}_2$ the unattached set $\{inc, dec\}$ is found. The unattached set defines $\mathcal{R}_0$ to be $\mathcal{K}_1$. Note that the CS is non-overlapping and by the previous observation, terminating and hence innermost normalising. The dependency pairs are

$$\langle inc(9 : x), inc(x)\rangle$$
$$\langle dec(0 : x), dec(x)\rangle.$$

It is easily seen that these two dependency pairs meet the last condition of the corollary. Hence, $\mathcal{K}_1 \cup \mathcal{K}_2$ is terminating.

For the CS $\mathcal{K}_1 \cup \mathcal{K}_2 \cup \mathcal{K}_3$ the unattached set $\{normadd, normsub\}$ is found. The unattached set defines $\mathcal{R}_0$ to be $\mathcal{K}_1 \cup \mathcal{K}_2$. Again the CS is non-overlapping and by the previous observation innermost normalising. No new dependency pairs on a cycle are introduced, thus the dependency pairs are the same as for $\mathcal{K}_1 \cup \mathcal{K}_2$ and hence the CS $\mathcal{K}_1 \cup \mathcal{K}_2 \cup \mathcal{K}_3$ is terminating.

For the CS $\mathcal{K}_1 \cup \mathcal{K}_2 \cup \mathcal{K}_3 \cup \mathcal{K}_4$ the unattached set $\{+, -\}$ is found. The unattached set defines $\mathcal{R}_0$ to be the CS that is proved terminating in the previous observation. The new dependency pairs on a cycle are

$$\langle (w : x) + (y : z), x + z\rangle$$
$$\langle (w : x) + MINUS(z), (w : x) - z\rangle$$
$$\langle MINUS(x) + (y : z), (y : z) - x\rangle$$
$$\langle MINUS(x) + MINUS(z), x + z\rangle$$
$$\langle (w : x) - (y : z), x - z\rangle$$
$$\langle (w : x) - MINUS(z), (w : x) + z\rangle$$
$$\langle MINUS(x) - (y : z), x + (y : z)\rangle$$
$$\langle MINUS(x) - MINUS(z), z - x\rangle$$

An argument selector that selects both arguments suffices to meet the last condition of the corollary, hence, the CS $\mathcal{K}_1 \cup \mathcal{K}_2 \cup \mathcal{K}_3 \cup \mathcal{K}_4$ is terminating.

The CS $\mathcal{K}_1 \cup \ldots \cup \mathcal{K}_5$ is proven termination in the same way with unattached symbol *timesten*.

For the CS $\mathcal{K}_1 \cup \ldots \cup \mathcal{K}_6$ the unattached symbol *times* is found. The CS $\mathcal{R}_0$ is defined to be $\mathcal{K}_1 \cup \ldots \cup \mathcal{K}_5$ which is proved terminating in the previous observation. The new dependency pairs on a cycle are

$$\begin{array}{l}\langle (w:x) \times (y:z), (w:END) \times z \rangle \\ \langle (w:x) \times (y:z), x \times (y:END) \rangle \\ \langle (w:x) \times (y:z), x \times z \rangle \\ \langle (w:x) \times MINUS(z), (w:x) \times z \rangle \\ \langle MINUS(x) \times (y:z), x \times (y:z) \rangle \\ \langle MINUS(x) \times MINUS(z), x \times z \rangle \end{array}$$

An argument selector that selects both arguments and the Knuth Bendix order (performed automatically as described in [DKM90]) suffice to meet the last condition of the corollary. Hence, the CS of Appendix A is terminating and moreover, can be proved terminating completely automatically.

5. Conclusions

The technique, proving absence of infinite chains of dependency pairs, can be performed automatically for a subclass of CSs. Even CSs that are not simply terminating can automatically be proved terminating. Therefore, the technique essentially differs from automatic techniques that are based on simplification orders, and thus unable to prove termination of CSs that are not simply terminating. Examples like Example 3.8 occur in literature as examples of non-simply terminating, and hence difficult, TRSs. However, intuitively proving termination of these TRSs is not difficult. The technique of infinite chains of dependency pairs justifies the intuition by giving an easy argumentation, that can be found automatically, for termination of these TRSs.

We showed that the described techniques are powerful enough to prove termination of the CS of Appendix A automatically. With Theorem 3.3, which is the reformulating of the theorem R. Kennaway developed to prove termination of this CS, termination can be proved directly. However, the described techniques are more general applicable. The techniques do not force linearity of any subterm, thus the techniques still apply on a slight modification of the CS of Appendix A with a non-linear subterm on a critical place. In the definition of an unattached set we only demand that there are no defined symbols of this same set nested. In Kennaway's approach no defined symbol may occur in a term starting with a function symbol of the unattached set (if this term occurs in a dependency pair that occurs on a cycle). Thus, replacing the rewrite rule for $(w:x) \times (y:z)$ of $\mathcal{K}_6$ by

$$\begin{array}{lll}(w:x) \times (y:z) & \rightarrow & (w \times_d y)+ \\ & & (((timesten(w:END) \times z) + (x \times timesten(y:END)))+ \\ & & timesten(timesten(x \times z))) \end{array}$$

results in a modified CS that can still be proved terminating automatically by the described techniques, whereas Theorem 3.3 fails on the dependency pairs

$$\begin{array}{l}\langle (w:x) \times (y:z), timesten(w:END) \times z \rangle \\ \langle (w:x) \times (y:z), x \times timesten(y:END) \rangle \end{array}$$

since *timesten* occurs in the right projection. Finally, Kennaway compares subterms with respect to the *size* order, whereas the presented techniques may use any well-founded order.

The technique is also very suitable to prove termination of CS that are obtained by transforming logic programs into CSs [AZ95b], and therefore, by the termination preserving property of the transformation, also to prove termination of logic programs.

In Corollary 4.17 we demand $\mathcal{R}_0$ to be non-overlapping and innermost normalising. By [Gra95], $\mathcal{R}_0$ is complete. One could expect, by considering this and the transformation, that the first two demands in the corollary can be replaced by one demand, viz. $\mathcal{R}_0$ is complete. However, the CS of K. Drosten [Dro89] rejects this possibility.

$$\begin{array}{lcl} 0 & \to & 2 \\ 1 & \to & 2 \\ g(x,y,y) & \to & x \\ g(y,y,x) & \to & x \\ \\ f(0,1,x) & \to & f(x,x,x) \\ f(x,y,z) & \to & 2 \end{array}$$

Choose the set of unattached symbols $\{f\}$, then $\mathcal{R}_0$ is defined to be the upper four rules. Since all critical pairs are joinable, $\mathcal{R}_0$ is complete. The only dependency pair $\langle f(0,1,x), f(x,x,x)\rangle$ can not be on a cycle by Lemma 3.7. But the CS is not terminating, since $f(g(0,1,1), g(0,1,1), g(0,1,1))$ has an infinite reduction.

A. Constructor system for decimal arithmetic

R. Kennaway proposed in [Ken95] a constructor system for decimal arithmetic. The constructor system of approximately 400 rewrite rules can be split into a base system and several systems that extend the base system. The complete system can be proved terminating automatically by the chains of dependency pairs technique together with Corollary 4.17. The base system is denoted by $\mathcal{K}_1$ and $\mathcal{K}_2 \ldots \mathcal{K}_5$ denote the extensions of the system.

$\mathcal{K}_1$

$$\begin{array}{lcl} 0 +_d 0 & \to & 0 \\ 0 +_d 1 & \to & 1 \\ & \vdots & \\ 9 +_d 9 & \to & 18 \\ \\ 0 \times_d 0 & \to & END \\ 1 \times_d 0 & \to & END \\ & \vdots & \\ 9 \times_d 9 & \to & 1 : (8 : END) \end{array} \qquad \begin{array}{lcl} 0 -_d 0 & \to & 0 \\ 0 -_d 1 & \to & MINUS(1) \\ & \vdots & \\ 7 -_d 5 & \to & 2 \\ 5 -_d 7 & \to & MINUS(2) \\ & \vdots & \\ 9 -_d 9 & \to & 0 \end{array}$$

for all digits in the range $0 \ldots 9$.

$\mathcal{K}_2$

$$
\begin{array}{lcl}
inc(END) & \rightarrow & 1 : END \\
inc(0 : x) & \rightarrow & 1 : x \\
inc(1 : x) & \rightarrow & 2 : x \\
 & \vdots & \\
inc(8 : x) & \rightarrow & 9 : x \\
inc(9 : x) & \rightarrow & 0 : inc(x)
\end{array}
\qquad
\begin{array}{lcl}
dec(0 : x) & \rightarrow & 9 : dec(x) \\
dec(1 : x) & \rightarrow & 0 : x \\
 & \vdots & \\
dec(9 : x) & \rightarrow & 8 : x
\end{array}
$$

$\mathcal{K}_3$

$$
\begin{array}{lcl}
normadd(0, x) & \rightarrow & 0 : x \\
 & \vdots & \\
normadd(9, x) & \rightarrow & 9 : x \\
normadd(10, x) & \rightarrow & 0 : inc(x) \\
 & \vdots & \\
normadd(18, x) & \rightarrow & 8 : inc(x)
\end{array}
$$

$$
\begin{array}{lcl}
normsub(0, END) & \rightarrow & END \\
normsub(0, x : y) & \rightarrow & 0 : (x : y) \\
normsub(0, MINUS(x)) & \rightarrow & MINUS(0 : x) \\
normsub(1, END) & \rightarrow & 1 : END \\
normsub(1, x : y) & \rightarrow & 1 : (x : y) \\
normsub(1, MINUS(x)) & \rightarrow & MINUS(9 : dec(x)) \\
 & \vdots & \\
normsub(9, END) & \rightarrow & 9 : END \\
normsub(9, x : y) & \rightarrow & 9 : (x : y) \\
normsub(9, MINUS(x)) & \rightarrow & MINUS(1 : dec(x)) \\
normsub(MINUS(1), END) & \rightarrow & MINUS(1 : END) \\
normsub(MINUS(1), x : y) & \rightarrow & 9 : dec(x : y) \\
normsub(MINUS(1), MINUS(x)) & \rightarrow & MINUS(1 : x) \\
 & \vdots & \\
normsub(MINUS(9), END) & \rightarrow & MINUS(9 : END) \\
normsub(MINUS(9), x : y) & \rightarrow & 1 : dec(x : y) \\
normsub(MINUS(9), MINUS(x)) & \rightarrow & MINUS(9 : x)
\end{array}
$$

$\mathcal{K}_4$

$$
\begin{array}{lcl}
END + END & \rightarrow & END \\
END + (y : z) & \rightarrow & y : z \\
END + MINUS(z) & \rightarrow & MINUS(z) \\
(w : x) + END & \rightarrow & w : x \\
(w : x) + (y : z) & \rightarrow & normadd(w +_d y, x + z) \\
(w : x) + MINUS(z) & \rightarrow & (w : x) - z \\
MINUS(x) + END & \rightarrow & MINUS(x) \\
MINUS(x) + (y : z) & \rightarrow & (y : z) - x \\
MINUS(x) + MINUS(z) & \rightarrow & MINUS(x + z)
\end{array}
$$

$$
\begin{array}{lcl}
END - END & \rightarrow & END \\
END - (y : z) & \rightarrow & MINUS(y : z) \\
END - MINUS(z) & \rightarrow & z \\
(w : x) - END & \rightarrow & w : x \\
(w : x) - (y : z) & \rightarrow & normsub(w -_d y, x - z) \\
(w : x) - MINUS(z) & \rightarrow & (w : x) + z \\
MINUS(x) - END & \rightarrow & MINUS(x) \\
MINUS(x) - (y : z) & \rightarrow & MINUS(x + (y : z)) \\
MINUS(x) - MINUS(z) & \rightarrow & z - x
\end{array}
$$

$\mathcal{K}_5$

$$
\begin{array}{lcl}
timesten(END) & \rightarrow & END \\
timesten(x : y) & \rightarrow & 0 : (x : y) \\
timesten(MINUS(x)) & \rightarrow & MINUS(timesten(x))
\end{array}
$$

$\mathcal{K}_6$

$$
\begin{array}{lcl}
END \times END & \rightarrow & END \\
END \times (y : z) & \rightarrow & END \\
END \times MINUS(z) & \rightarrow & END \\
(w : x) \times END & \rightarrow & END \\
(w : x) \times (y : z) & \rightarrow & (w \times_d y) + \\
 & & (timesten(((w : END) \times z) + (x \times (y : END))) + \\
 & & timesten(timesten(x \times z))) \\
(w : x) \times MINUS(z) & \rightarrow & MINUS((w : x) \times z) \\
MINUS(x) \times END & \rightarrow & END \\
MINUS(x) \times (y : z) & \rightarrow & MINUS(x \times (y : z)) \\
MINUS(x) \times MINUS(z) & \rightarrow & x \times z
\end{array}
$$

References

[AZ95a] Thomas Arts and Hans Zantema. Termination of constructor systems using semantic unification. Technical Report UU-CS-1995-17, Utrecht, May 1995.

[AZ95b] Thomas Arts and Hans Zantema. Termination of logic programs via labelled term rewrite systems. In *Proceedings of CSN 1995*. Sion, November 1995. Full version appeared as technical report UU-CS-1994-20.

[DKM90] Jeremy Dick, John Kalmus, and Ursula Martin. Automating the knuth bendix ordering. *Acta Informatica*, 28:95–119, 1990.

[Dro89] K. Drosten. Termersetzungssysteme : Grundlagen der prototypgenerierung algebraischer spezifikationen. Springer, Berlin, 1989.

[Gra95] Bernhard Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.

[HL78] G. Huet and D. Lankford. On the uniform halting problem for term rewriting systems. Technical Report report 283, INRIA, 1978.

[Ken95] Richard Kennaway. Complete term rewrite systems for decimal arithmetic and other total recursive functions. Presented at the Workshop on Termination, La Bresse, France, May 1995.