Deciding Innermost Loops*

René Thiemann[1], Jürgen Giesl[2], and Peter Schneider-Kamp[2]

[1] Institute of Computer Science, University of Innsbruck, Austria
rene.thiemann@uibk.ac.at
[2] LuFG Informatik 2, RWTH Aachen University, Germany
{giesl,psk}@informatik.rwth-aachen.de

Abstract. We present the first method to disprove *innermost* termination of term rewrite systems automatically. To this end, we first develop a suitable notion of an innermost loop. Second, we show how to detect innermost loops: One can start with any technique amenable to find loops. Then our novel procedure can be applied to decide whether a given loop is an innermost loop. We implemented and successfully evaluated our method in the termination prover AProVE.

1 Introduction

Termination is an important property of term rewrite systems (TRSs). Therefore, much effort has been spent on developing and automating powerful techniques for showing (innermost) termination of TRSs. An important application area for these techniques is termination analysis of functional programs. Since the evaluation mechanism of functional languages is mainly term rewriting, one can transform functional programs into TRSs and prove termination of the resulting TRSs to conclude termination of the functional programs [9]. Although "full" rewriting does not impose any evaluation strategy, this approach is sound even if the underlying programming language has an innermost evaluation strategy.

But in order to detect bugs in programs, it is at least as important to prove *non-termination* of programs or of the corresponding TRSs. Here, the evaluation strategy cannot be ignored, because a non-terminating TRS may still be innermost terminating. Thus, in order to disprove termination of programming languages with an innermost strategy, it is important to develop techniques to disprove *innermost* termination of TRSs automatically.

Only a few techniques for showing non-termination of TRSs have been introduced so far [7,12,17,18,20]. Nevertheless, there already exist several tools that are able to prove non-termination of TRSs automatically by finding loops (e.g., AProVE [8], Jambox [5], Matchbox [23], NTI [20], TORPA [24], TTT [14]). But up to now, all of these techniques and tools only disprove full and not innermost termination. So they can only be applied to disprove innermost termination if the TRS belongs to a known class where termination and innermost termination coincide [11]. In this paper, we demonstrate how to extend all of these techniques such that they can be directly used for disproving innermost termination for any

* Supported by the Deutsche Forschungsgemeinschaft (DFG) under grant GI 274/5-2.

kind of TRS. For instance, this is needed for the following program where the resulting TRS is not confluent and hence, does not fall into a known class where innermost and full termination are the same.

Example 1 (Factorial function). The following ACL2 program [15] computes the factorial function where x is increased from 0 to $y-1$ and in every iteration the result is multiplied by $1+x$.

```
(defun factorial (y) (fact 0 y))
(defun fact (x y)
  (if (== x y)
      1
      (× (+ 1 x) (fact (+ 1 x) y))))
```

Using a translation to TRSs suggested by [22], we obtain the following TRS $\mathcal{R}$ where the rules (5) – (12) are needed to handle the built-in functions of ACL2.

$$\mathsf{factorial}(y) \to \mathsf{fact}(0, y) \quad (1)$$
$$\mathsf{fact}(x, y) \to \mathsf{if}(x == y, x, y) \quad (2)$$
$$\mathsf{if}(\mathsf{true}, x, y) \to \mathsf{suc}(0) \quad (3)$$
$$\mathsf{if}(\mathsf{false}, x, y) \to \mathsf{suc}(x) \times \mathsf{fact}(\mathsf{suc}(x), y) \quad (4)$$
$$0 + y \to y \quad (5)$$
$$\mathsf{suc}(x) + y \to \mathsf{suc}(x + y) \quad (6)$$
$$0 \times y \to 0 \quad (7)$$
$$\mathsf{suc}(x) \times y \to y + (x \times y) \quad (8)$$
$$x == y \to \mathsf{chk}(\mathsf{eq}(x, y)) \quad (9)$$
$$\mathsf{eq}(x, x) \to \mathsf{true} \quad (10)$$
$$\mathsf{chk}(\mathsf{true}) \to \mathsf{true} \quad (11)$$
$$\mathsf{chk}(\mathsf{eq}(x, y)) \to \mathsf{false} \quad (12)$$

Here, it is crucial to use *innermost* instead of full rewriting. Otherwise, it would always be possible to rewrite $s == t \to_{\mathcal{R}} \mathsf{chk}(\mathsf{eq}(s,t)) \to_{\mathcal{R}} \mathsf{false}$, i.e., terms like $0 == 0$ could then be evaluated to both true and false. In contrast, for innermost rewriting one has to apply rule (10) first if s and t are equal.

Note that in this TRS, $s == t$ is indeed evaluated to false whenever s and t are *any* terms that are syntactically different. This is essential to model the semantics of ACL2 correctly, since here there are – like in term rewriting – no types. At the same time, all functions in ACL2 must be "completely defined".

So to perform non-termination proofs for languages like ACL2, we need a way to disprove innermost termination. This problem is harder than disproving termination since one has to take care of the evaluation strategy.

In this paper we investigate *looping* reductions. These are specific kinds of infinite reductions which can be represented in a finite way. To disprove innermost termination of TRSs, we develop an automatic method which in case of success, presents the innermost loop to the user as a counterexample.

For the TRS of Ex. 1, there is indeed an innermost loop. It corresponds to the non-terminating reduction of the ACL2 program when calling $\mathsf{fact}(n, m)$ for natural numbers $n > m$. The reason is that the first argument is increased over and over again, and it will never become equal to m.

The paper is organized as follows. In Sect. 2, we extend the notion of a loop to innermost rewriting. Then as the main contribution of the paper, we describe a

novel decision procedure in Sect. 3 which detects whether a loop for full rewriting is still a loop in the innermost case. How to combine our work with dependency pairs is discussed in Sect. 4. Finally, Sect. 5 summarizes our results and describes their empirical evaluation with the termination prover AProVE.

2 Loops

We only regard finite signatures and TRSs and refer to [2] for the basics of rewriting. An obvious approach to find infinite reductions is to search for a term s which rewrites to a term t containing an instance of s, i.e., $s \to_{\mathcal{R}}^+ t = C[s\mu]$ for some context C and substitution μ. The corresponding infinite reduction is

$$s \to_{\mathcal{R}}^+ C[s\mu] \to_{\mathcal{R}}^+ C[C\mu[s\mu^2]] \to_{\mathcal{R}}^+ C[C\mu[C\mu^2[s\mu^3]]] \to_{\mathcal{R}}^+ \dots$$

Equivalently, one can also represent it as an infinite reduction w.r.t. $\to_{\mathcal{R}}^+ \circ \trianglerighteq$, where $\trianglerighteq$ is the weak subterm relation:

$$s \to_{\mathcal{R}}^+ \circ \trianglerighteq s\mu \to_{\mathcal{R}}^+ \circ \trianglerighteq s\mu^2 \to_{\mathcal{R}}^+ \circ \trianglerighteq s\mu^3 \to_{\mathcal{R}}^+ \circ \trianglerighteq \dots \qquad (\star)$$

Here, for every $s\mu^n$ the same rules are applied at the same positions to obtain $s\mu^{n+1}$. A reduction of the form $s \to_{\mathcal{R}}^+ t \trianglerighteq s\mu$ is called a *loop* and a TRS which admits a loop is called *looping*.

Example 2. The TRS of Ex. 1 admits the following loop where $s = \mathsf{fact}(x, y)$ and $\mu = \{x/\mathsf{suc}(x)\}$.

$$\begin{aligned} s &\to_{\mathcal{R}} \mathsf{if}(x == y, x, y) \\ &\to_{\mathcal{R}} \mathsf{if}(\mathsf{chk}(\mathsf{eq}(x, y)), x, y) \\ &\to_{\mathcal{R}} \mathsf{if}(\mathsf{false}, x, y) \\ &\to_{\mathcal{R}} \mathsf{suc}(x) \times \mathsf{fact}(\mathsf{suc}(x), y) \\ &\trianglerighteq \mathsf{fact}(\mathsf{suc}(x), y) \\ &= s\mu \end{aligned}$$

Clearly, a naive search for looping terms is very costly. Therefore, in current non-termination provers the techniques of forward closures [3,18], unfoldings [20], ancestor graphs [17], forward- or backward-narrowing [7], and overlap closures [12] are used, where all these techniques are special forms of overlap closures. As all mentioned techniques essentially perform narrowing steps, one can modify them by also allowing narrowings into variables. This is proposed in [7] and [20]. For example, the loop of the TRS $\{\mathsf{f}(x, y, x, y, z) \to \mathsf{f}(0, 1, z, z, z), \mathsf{a} \to 0, \mathsf{a} \to 1\}$ of [25] cannot be detected by overlap closures if one does not permit narrowings into variables. Nevertheless, most of these techniques are able to detect the loop of Ex. 2. Another alternative to detect loops (at least for string rewriting) could be based on specialized unification procedures, cf. [4].

However, if one does not consider full rewriting but innermost rewriting, then loopingness does not imply non-termination,[3] since the innermost rewrite relation $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ is not stable under substitutions. More precisely, one should not define

[3] As usual, a TRS is *innermost non-terminating* iff there is a (possibly non-ground) term starting an infinite innermost reduction.

any TRS with a reduction $s \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu$ to be "innermost looping", because then an "innermost looping" TRS could still be innermost terminating as shown by Ex. 3. The reason is that $s \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu$ does not imply $s\mu \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu^2$. And even if $s\mu \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu^2$ is true, then it could be that later on for some larger n there is no reduction $s\mu^n \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu^{n+1}$.

Example 3. Consider the TRS $\mathcal{R}$ consisting of the following rules.

$$\begin{aligned} \mathsf{f}(\mathsf{g}(x)) &\to \mathsf{f}(\mathsf{g}(\mathsf{g}(x))) \\ \mathsf{g}(\mathsf{g}(\mathsf{g}(x))) &\to \mathsf{a} \end{aligned}$$

This TRS would be "innermost looping" according to the definition discussed above, e.g., $\mathsf{f}(\mathsf{g}(x)) \to_{\mathcal{R}} \mathsf{f}(\mathsf{g}(\mathsf{g}(x))) = \mathsf{f}(\mathsf{g}(x))\{x/\mathsf{g}(x)\}$, but it is innermost terminating. The reason is that the first rule is applicable at most twice. Afterwards, one has to use the second rule and no reduction is possible afterwards.

To solve this problem, one might define that a TRS $\mathcal{R}$ is "innermost looping" iff there are a term s and a substitution μ such that $s\mu^n \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu^{n+1}$ *for every natural number* n. A similar definition was already used in [7, Footnote 6]. Then indeed, innermost loopingness implies innermost non-termination. However, the following example shows that this definition does not correspond to a loop in the intuitive way where the reduction $s\mu^n \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}} \circ \trianglerighteq s\mu^{n+1}$ always has the same form and length. Consequently, it would be undecidable whether a known loop is also an innermost loop.

Example 4. Consider the TRS $\mathcal{R}$ with the following rules.

$$\begin{aligned} \mathsf{f}(x, y) &\to \mathsf{f}(\mathsf{suc}(x), \mathsf{g}(\mathsf{h}(x, 0))) \\ \mathsf{h}(\mathsf{suc}(x), y) &\to \mathsf{h}(x, \mathsf{suc}(y)) \\ \mathsf{g}(\mathsf{h}(x, y)) &\to \mathsf{j}(y) \end{aligned}$$

$\mathcal{R}$ is "innermost looping", as for $s = \mathsf{f}(\mathsf{suc}(x), \mathsf{g}(\mathsf{h}(x, 0)))$ and $\mu = \{x/\mathsf{suc}(x)\}$ there is the following reduction for every $n \in \mathbb{N}$.

$$\begin{aligned} s\mu^n &= \mathsf{f}(\mathsf{suc}^{n+1}(x), \mathsf{g}(\mathsf{h}(\mathsf{suc}^n(x), 0))) \\ &\xrightarrow{\mathsf{i}}{}^{n}_{\mathcal{R}} \mathsf{f}(\mathsf{suc}^{n+1}(x), \mathsf{g}(\mathsf{h}(x, \mathsf{suc}^n(0)))) \\ &\xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{f}(\mathsf{suc}^{n+1}(x), \mathsf{j}(\mathsf{suc}^n(0))) \\ &\xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{f}(\mathsf{suc}^{n+2}(x), \mathsf{g}(\mathsf{h}(\mathsf{suc}^{n+1}(x), 0))) \\ &= s\mu^{n+1} \end{aligned}$$

The problem is that the form and the *length* of the reduction from $s\mu^n$ to $s\mu^{n+1}$ depend on n. Therefore, with this definition of "innermost looping", it is not even semi-decidable whether a known loop is an innermost loop.

To see this, recall that it is not semi-decidable whether a (computable) function j over the naturals is total. Since term rewriting is Turing-complete, we can assume that there are confluent rules which compute j by innermost rewriting. But then we can add the three rules of $\mathcal{R}$ and totality of j is equivalent to the question whether the reduction above is an innermost loop, since we obtain an innermost loop iff all terms $\mathsf{j}(\mathsf{suc}^n(0))$ are innermost terminating.

So the problem with the requirement $s\mu^n \xrightarrow{\mathsf{i}}{}^+_{\mathcal{R}} \circ \trianglerighteq s\mu^{n+1}$ is that for every n, the reduction from $s\mu^n$ to $s\mu^{n+1}$ may be completely different. In contrast, in the infinite reduction ($\star$) that corresponds to a loop for full rewriting, the reductions from $s\mu^n$ to $s\mu^{n+1}$ always have the same form. For every n, one can apply exactly the same rules in exactly the same order at exactly the same positions. Hence, one only has to give the reduction $s \to^+_{\mathcal{R}} t \trianglerighteq s\mu$. Then one immediately knows how to continue for $s\mu, s\mu^2, \ldots$. This gives rise to our final definition of "innermost looping".

Definition 5 (Innermost Looping TRS). *A TRS $\mathcal{R}$ is* innermost looping *iff there are a substitution μ, a number $m \geq 1$, terms $s_1, \ldots, s_m, t$, rules $\ell_1 \to r_1, \ldots, \ell_m \to r_m \in \mathcal{R}$, and positions $p_1, \ldots, p_m$ such that for all $n \in \mathbb{N}$ all steps in the following looping reduction[4] are innermost steps.*

$$s_1\mu^n \to_{\ell_1 \to r_1, p_1} s_2\mu^n \to_{\ell_2 \to r_2, p_2} \ldots s_m\mu^n \to_{\ell_m \to r_m, p_m} t\mu^n \trianglerighteq s_1\mu^{n+1} \qquad (\star\star)$$

Note that ($\star\star$) is the same as the looping reduction in ($\star$), which is just written down in a more detailed way. Hence, one can represent an innermost loop in the same way as a loop for termination: by just giving the reduction $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \ldots s_m \to_{\mathcal{R}} t \trianglerighteq s_1\mu$, i.e., $s_1 \to^+_{\mathcal{R}} t \trianglerighteq s_1\mu$.

3 Detecting Innermost Loops

It is clear that with Def. 5, every innermost looping TRS is innermost non-terminating. Moreover, there exist several techniques and tools to find ordinary loops (for full rewriting). Such loops are good starting points when searching for innermost loops because an innermost loop is a loop which satisfies the additional requirements of Def. 5. The only remaining problem is to check whether such an ordinary loop is also an innermost loop.

Example 6. Consider the looping reduction of Ex. 2. To check whether this is an innermost loop we have to check for $\mu = \{x/\mathsf{suc}(x)\}$ and for all $n \in \mathbb{N}$ whether the corresponding steps are innermost steps when instantiating the terms with μ^n. The problem in this example is the reduction $\mathsf{if}(\mathsf{chk}(\mathsf{eq}(x,y)), x, y)\mu^n \to_{\mathcal{R}} \mathsf{if}(\mathsf{false}, x, y)\mu^n$ at position 1 since the redex contains the subterm $\mathsf{eq}(x,y)\mu^n$ which might not be a normal form for some n due to rule $\mathsf{eq}(x,x) \to \mathsf{true}$.

In the remainder of this section we will show the main result that it is decidable whether a given loop is an innermost loop. For example, it will turn out that the loop in Ex. 2 is an innermost loop whereas the one of Ex. 3 is not. We show this result in 4 steps, corresponding to the sections 3.1 – 3.4.

3.1 From Innermost Loops to Redex Problems

Note that ($\star\star$) is an innermost loop iff every direct subterm of every redex $s_i\mu^n|_{p_i}$ is in normal form. Since a term t is in normal form iff t does not contain a redex w.r.t. $\mathcal{R}$, we can reformulate the question about innermost loopingness in terms

[4] Here, $\to_{\ell \to r, p}$ denotes a rewrite step with the rule $\ell \to r$ at position p.

of so-called *redex problems*.

Definition 7 (Redex, Matching, and Identity Problems). *Let s and ℓ be terms, let μ be a substitution (with finite domain). Then a* redex problem *is a triple $(s \mid\!\rhd \ell, \mu)$, a* matching problem *is a triple $(s \rhd \ell, \mu)$, and an* identity problem *is a triple $(s \approx \ell, \mu)$.*

A redex problem $(s \mid\!\rhd \ell, \mu)$ is solvable *iff there are a position p, a substitution σ, and an $n \in \mathbb{N}$ such that $s\mu^n|_p = \ell\sigma$. A matching problem is* solvable *iff there are a substitution σ and an $n \in \mathbb{N}$ such that $s\mu^n = \ell\sigma$. An identity problem is* solvable *iff there is an $n \in \mathbb{N}$ such that $s\mu^n = \ell\mu^n$.*

Theorem 8 (Setting up Redex Problems). *In the reduction ($\star\star$) all steps are innermost steps iff for all direct subterms s of the $s_i|_{p_i}$ and all left-hand sides ℓ of rules from $\mathcal{R}$, the redex problem $(s \mid\!\rhd \ell, \mu)$ is not solvable.*

Proof. Some reduction $s_i\mu^n \to_{\ell_i \to r_i, p_i} u$ is not an innermost step iff for some direct subterm s of $s_i|_{p_i}$, the term $s\mu^n$ is not in normal form, since $s_i\mu^n|_{p_i} = s_i|_{p_i}\mu^n$. (Note that even for $n = 0$ we have a reduction at position p_i in ($\star\star$). Hence, p_i is a position of s_i and moreover, $s_i|_{p_i}$ cannot be a variable. Thus, the "direct subterms of $s_i|_{p_i}$" are indeed properly defined.) Equivalently, there are some rule $\ell \to r$ and position p such that $s\mu^n|_p = \ell\sigma$. But then the redex problem $(s \mid\!\rhd \ell, \mu)$ is solvable. □

Example 9. The loop of Ex. 2 is an innermost loop iff for $\mu = \{x/\mathsf{suc}(x)\}$ all redex problems $(s \mid\!\rhd \ell, \mu)$ are not solvable where s is from the set $\{x, y, \mathsf{eq}(x, y), \mathsf{false}\}$ of direct subterms of redexes in the loop and ℓ is a left-hand side of $\mathcal{R}$.

The loop of Ex. 3 is an innermost loop iff both $(\mathsf{g}(x) \mid\!\rhd \mathsf{f}(\mathsf{g}(x)), \mu')$ and $(\mathsf{g}(x) \mid\!\rhd \mathsf{g}(\mathsf{g}(\mathsf{g}(x))), \mu')$ are not solvable where $\mu' = \{x/\mathsf{g}(x)\}$.

To find out whether a redex problem $(s \mid\!\rhd \ell, \mu)$ is solvable, we search for three unknowns: the position p, the substitution σ, and the number n. We will now eliminate these unknowns one by one and start with the position p. This will result in matching problems. Then in a second step we will further transform matching problems into identity problems where only the number n is unknown. Finally, we will present an algorithm to decide identity problems. Therefore, at the end of this section we will have a decision procedure for redex problems, and thus also for the question whether a given loop is an innermost loop.

3.2 From Redex Problems to Matching Problems

To start with simplifying a redex problem $(s \mid\!\rhd \ell, \mu)$ into a finite disjunction of matching problems, note that since the position p can be chosen freely within any of the terms $s, s\mu, s\mu^2, \ldots$, it is not feasible to just try out all possibilities. But the following theorem shows that it is indeed possible to reduce redex problems to finitely many matching problems. Essentially, it states that it suffices to consider all subterms of s and all subterms of terms that are introduced by μ. Here, $\mathcal{V}$ is the set of all variables and for any term t, $\mathcal{V}(t)$ is the set of its variables and $Pos(t)$ is the set of its positions.

Theorem 10 (Solving Redex Problems). *Let $(s \mid\trianglerighteqslant \ell, \mu)$ be a redex problem. Let $\mathcal{W} = \bigcup_{i\in\mathbb{N}} \mathcal{V}(s\mu^i)$. Then $(s \mid\trianglerighteqslant \ell, \mu)$ is solvable iff ℓ is a variable or if one of the matching problems $(u \trianglerighteqslant \ell, \mu)$ is solvable for some non-variable subterm u of a term in $\{s\} \cup \{x\mu \mid x \in \mathcal{W}\}$.*

Proof. If ℓ is a variable then the redex problem is obviously solvable, so let $\ell \notin \mathcal{V}$. We consider both directions separately.

First, let $(u \trianglerighteqslant \ell, \mu)$ be solvable, i.e., there are σ and n such that $u\mu^n = \ell\sigma$. If u is a subterm of s, i.e., $u = s|_p$ for some p, then $s\mu^n|_p = s|_p\mu^n = u\mu^n = \ell\sigma$ proves that $(s \mid\trianglerighteqslant \ell, \mu)$ is solvable. Otherwise, if u is a subterm of some $x\mu$ with $x \in \mathcal{W}$ then there is some i such that $x \in \mathcal{V}(s\mu^i)$. Hence, there is a position p such that $s\mu^{i+1}|_p = u$. Again, $s\mu^{i+1+n}|_p = u\mu^n = \ell\sigma$ proves that $(s \mid\trianglerighteqslant \ell, \mu)$ is solvable.

For the other direction of the equivalence we assume that $(s \mid\trianglerighteqslant \ell, \mu)$ is solvable, so let $s\mu^n|_p = \ell\sigma$ for some p, σ, and n. If $p \in Pos(s)$ and $s|_p \notin \mathcal{V}$, then we are done as the matching problem $(u \trianglerighteqslant \ell, \mu)$ for the corresponding subterm $u = s|_p$ is obviously solvable.

Otherwise, there must be an $0 \leq i < n$ such that $p \in Pos(s\mu^{i+1})$ with $s\mu^{i+1}|_p \notin \mathcal{V}$ (as $\ell \notin \mathcal{V}$) and either $s\mu^i|_p \in \mathcal{V}$ or $p \notin Pos(s\mu^i)$. In both cases there must be a variable x and a position p' such that $x \in \mathcal{V}(s\mu^i) \subseteq \mathcal{W}$ and $x\mu|_{p'} = s\mu^{i+1}|_p$. We choose the non-variable subterm $u = x\mu|_{p'}$ of $x\mu$. Then indeed the matching problem $(u \trianglerighteqslant \ell, \mu)$ is solvable since

$$u\mu^{n-(i+1)} = x\mu|_{p'}\mu^{n-(i+1)} = s\mu^{i+1}|_p\mu^{n-(i+1)} = s\mu^n|_p = \ell\sigma. \qquad \square$$

Note that the set $\mathcal{W}$ is a subset of the finite set $\mathcal{V}(s) \cup \bigcup_{x\in Dom(\mu)} \mathcal{V}(x\mu)$. Thus, one can compute $\mathcal{W}$ by adding $\mathcal{V}(s\mu^i)$ for larger and larger i until one reaches an i where the set does not increase anymore. Hence, Thm. 10 can easily be automated.

Example 11. We use Thm. 10 for the redex problems of Ex. 9. We first consider the redex problems resulting from the loop of Ex. 2. Since there are no new variables occurring when applying μ we obtain $\mathcal{W} = \{x, y\}$. Thus, the loop is an innermost loop iff none of the matching problems $(s \trianglerighteqslant \ell, \mu)$ is solvable where s is now chosen from $\{\mathsf{suc}(x), \mathsf{eq}(x, y), \mathsf{false}\}$. (So the variables x, y do not have to be regarded anymore, but now one has to consider the new term $\mathsf{suc}(x)$ from the substitution.)

In the same way, the loop of Ex. 3 is an innermost loop iff none of the matching problems $(\mathsf{g}(x) \trianglerighteqslant \mathsf{f}(\mathsf{g}(x)), \mu')$ and $(\mathsf{g}(x) \trianglerighteqslant \mathsf{g}(\mathsf{g}(\mathsf{g}(x))), \mu')$ is solvable.

3.3 From Matching Problems to Identity Problems

Now the question remains whether a given matching problem is solvable. This amounts to detecting the matcher σ and the unknown number n. Our next aim is to reduce this problem to a conjunction of identity problems, i.e., to eliminate the need to search for matchers σ. However, we first have to generalize the notion of matching problems $(s \trianglerighteqslant \ell, \mu)$ which contain one pair of terms $s \trianglerighteqslant \ell$ to matching problems which allow a set of pairs of terms.

Definition 12 (General Matching Problem). *A* general matching problem $(\mathcal{M}, \mu)$ *consists of a set* $\mathcal{M}$ *of pairs* $\{s_1 \gtrdot \ell_1, \ldots, s_k \gtrdot \ell_k\}$ *together with a substitution* μ*. A general matching problem* $(\mathcal{M}, \mu)$ *is* solvable *iff there are a substitution* σ *and an* $n \in \mathbb{N}$ *such that for all* $1 \leq j \leq k$ *the equality* $s_j\mu^n = \ell_j\sigma$ *is valid.*

If $\mathcal{M}$ *only contains one pair* $s \gtrdot \ell$ *then we identify* $(\mathcal{M}, \mu)$ *with* $(s \gtrdot \ell, \mu)$*, and if* μ *is clear from the context we write* $\mathcal{M}$ *as an abbreviation for* $(\mathcal{M}, \mu)$*.*

We now give a set of four transformation rules which either detect that a matching problem is not solvable (indicated by $\bot$), or which transform a matching problem into solved form. Here, a (general) matching problem $(\{s_1 \gtrdot \ell_1, \ldots, s_k \gtrdot \ell_k\}, \mu)$ is in *solved form* iff all $\ell_1, \ldots, \ell_k$ are variables. Once we have reached a matching problem in solved form, it is easily possible to translate it into identity problems.

Definition 13 (Transformation of Matching Problems). *We define the following transformation* $\Rightarrow$ *on general matching problems. If* $(\mathcal{M}, \mu)$ *is a general matching problem with* $\mathcal{M} = \mathcal{M}' \uplus \{s \gtrdot \ell\}$ *where* $\ell \notin \mathcal{V}$*, and if* $\mathcal{V}_{incr} = \{x \in \mathcal{V} \mid$ *there is some* $n \in \mathbb{N}$ *with* $x\mu^n \notin \mathcal{V}\}$ *is the set of* increasing variables*, then*

(i) $\mathcal{M} \Rightarrow \{s'\mu \gtrdot \ell' \mid s' \gtrdot \ell' \in \mathcal{M}\}$*, if* $s \in \mathcal{V}_{incr}$
(ii) $\mathcal{M} \Rightarrow \bot$*, if* $s \in \mathcal{V} \setminus \mathcal{V}_{incr}$
(iii) $\mathcal{M} \Rightarrow \bot$*, if* $s = f(\ldots)$*,* $\ell = g(\ldots)$*, and* $f \neq g$
(iv) $\mathcal{M} \Rightarrow \mathcal{M}' \cup \{s_1 \gtrdot \ell_1, \ldots, s_k \gtrdot \ell_k\}$*, if* $s = f(s_1, \ldots, s_k)$*,* $\ell = f(\ell_1, \ldots, \ell_k)$

Rule (iv) just decomposes terms and Rule (iii) handles a symbol-clash. These rules are standard for classical matching algorithms. However, if the left-hand side is a variable x and the right-hand side is not, then a matching problem may still be solvable. If x is increasing then we just have to apply μ until a new symbol is produced on the left-hand side. This is done by Rule (i) and will be illustrated in more detail when solving the matching problems of the loop in Ex. 3. However, if x is not increasing then the matching problem is not solvable since $x\mu^n$ will always remain a variable. Hence, $\bot$ is obtained by Rule (ii). The following theorem shows that every matching problem $(s \gtrdot \ell, \mu)$ can be automatically reduced to a finite conjunction of identity problems.

Theorem 14 (Solving Matching Problems). *Let* $(\mathcal{M}, \mu)$ *be a general matching problem.*

(i) The transformation rules of Def. 13 are confluent and terminating.
(ii) If $\mathcal{M} \Rightarrow \bot$ *then* $\mathcal{M}$ *is not solvable.*
(iii) If $\mathcal{M} \Rightarrow \mathcal{M}'$ *with* $\mathcal{M}' \neq \bot$*, then* $\mathcal{M}$ *is solvable iff* $\mathcal{M}'$ *is solvable.*
(iv) $\mathcal{M}$ *is solvable iff* $\mathcal{M} \Rightarrow^* \mathcal{M}'$ *for some matching problem* $\mathcal{M}' = \{s_1 \gtrdot x_1, \ldots, s_k \gtrdot x_k\}$ *in solved form, such that for all* $i \neq j$ *with* $x_i = x_j$ *the identity problem* $(s_i \cong s_j, \mu)$ *is solvable.*

Proof. (i) To prove confluence one can show that $\Rightarrow$ is strongly confluent by a simple case analysis.

To show termination of $\Rightarrow$ first note that no transformation rule increases the size of the terms in the right-hand sides of a matching problem. Thus, Rule (iv) can only be applied finitely often. But since every sequence of transformations with Rule (i) eventually triggers an application of Rule (iii) or (iv), Rule (i) cannot be used infinitely often either.

(ii) If $\mathcal{M} \Rightarrow \bot$ due to Rule (iii) then $s \trianglerighteq \ell \in \mathcal{M}$ with $s = f(\dots)$ and $\ell = g(\dots)$ where $f \neq g$. But then for every $n \in \mathbb{N}$ the terms $s\mu^n = f(\dots)$ and $\ell\sigma = g(\dots)$ are different. Hence, $\mathcal{M}$ is not solvable.
If $\mathcal{M} \Rightarrow \bot$ due to Rule (ii) then $x \trianglerighteq \ell \in \mathcal{M}$ with $x \in \mathcal{V} \setminus \mathcal{V}_{incr}$ and $\ell = f(\dots)$. But since x is not an increasing variable we know that $x\mu^n \in \mathcal{V}$ for all $n \in \mathbb{N}$. Thus, the terms $x\mu^n$ and $\ell\sigma = f(\dots)$ are different for all n. Hence, $\mathcal{M}$ is not solvable.

(iii) We first consider Rule (i) for $\mathcal{M} = \{s_1 \trianglerighteq \ell_1, \dots, s_k \trianglerighteq \ell_k\}$.

$$\begin{array}{ll} & \mathcal{M} \text{ is solvable} \\ \text{iff} & \exists \sigma, n : s_1\mu^n = \ell_1\sigma \wedge \dots \wedge s_k\mu^n = \ell_k\sigma \\ \text{iff} & \exists \sigma', n : s_1\mu^{n+1} = \ell_1\sigma' \wedge \dots \wedge s_k\mu^{n+1} = \ell_k\sigma' \quad (\text{as } s_i \in \mathcal{V} \text{ for some } i) \\ \text{iff} & \mathcal{M}' = \{s_1\mu \trianglerighteq \ell_1, \dots, s_k\mu \trianglerighteq \ell_k\} \text{ is solvable} \end{array}$$

For Rule (iv) the result follows from the fact that $f(s_1, \dots, s_k)\mu^n = f(\ell_1, \dots, \ell_k)\sigma$ iff $s_i\mu^n = \ell_i\sigma$ for all $1 \leq i \leq k$.

(iv) If $\mathcal{M}$ is solvable then due to (ii) and (iii), $\mathcal{M}$ cannot be transformed to $\bot$. So let $\mathcal{M}'$ be a normal form of $\mathcal{M}$ w.r.t. $\Rightarrow$. Then, obviously $\mathcal{M}'$ has the form $\{s_1 \trianglerighteq x_1, \dots, s_k \trianglerighteq x_k\}$ and $\mathcal{M}'$ is solvable due to (iii). Thus, there are a substitution σ and a number n such that for all $1 \leq i \leq k$ the equality $s_i\mu^n = x_i\sigma$ is valid. Hence, for all $i \neq j$ with $x_i = x_j$ the identity problem $(s_i \cong s_j, \mu)$ is solvable.
For the other direction let $\mathcal{M} \Rightarrow^* \mathcal{M}' = \{s_1 \trianglerighteq x_1, \dots, s_k \trianglerighteq x_k\}$ where for every $i \neq j$ with $x_i = x_j$ there is some n_{ij} with $s_i\mu^{n_{ij}} = s_j\mu^{n_{ij}}$. Let n be the maximum of all n_{ij}. Then, obviously $s_i\mu^n = s_j\mu^n$ for all these i and j. We define $\sigma = \{x_1/s_1\mu^n, \dots, x_k/s_k\mu^n\}$. First note that σ is well defined by construction. But as then $s_i\mu^n = x_i\sigma$ is valid for all $1 \leq i \leq k$ we know that $\mathcal{M}'$ is solvable. Using (iii) we finally conclude that $\mathcal{M}$ is solvable. □

Example 15. We illustrate the transformation rules by continuing Ex. 11.

For the loop of Ex. 2 we can reduce all but one matching problem to $\bot$ by Rule (iii). Only the matching problem $(\mathsf{eq}(x, y) \trianglerighteq \mathsf{eq}(x, x), \mu)$ is transformed by Rule (iv) into its solved form $\{x \trianglerighteq x, y \trianglerighteq x\}$. Hence, by Thm. 14 the loop is an innermost loop iff the identity problem $(x \cong y, \mu)$ is not solvable.

For the loop of Ex. 3, we had to find out whether $(\mathsf{g}(x) \trianglerighteq \mathsf{g}(\mathsf{g}(\mathsf{g}(x))), \mu')$ is solvable. Applying Rule (iv) yields $(x \trianglerighteq \mathsf{g}(\mathsf{g}(x)), \mu')$. Since x is an increasing variable for μ', we now have to apply Rule (i) and obtain $(\mathsf{g}(x) \trianglerighteq \mathsf{g}(\mathsf{g}(x)), \mu')$ as $x\mu' = \mathsf{g}(x)$. Repeated application of Rules (iv) and (i) results in the solved form $(x \trianglerighteq x, \mu')$. Hence, by Thm. 14 the matching problem $(\mathsf{g}(x) \trianglerighteq \mathsf{g}(\mathsf{g}(\mathsf{g}(x))), \mu')$ is solvable as no identity problems are created. Thus, we have detected that the loop of Ex. 3 is not an innermost loop.

3.4 Deciding Identity Problems

Note that for left-linear TRSs, identity problems are never created, since there the right-hand sides of a general matching problem are always variable disjoint. However, in order to handle also non-left-linear TRSs, it remains to give an algorithm which decides solvability of an identity problem.[5] This algorithm is presented in Fig. 1, and we now explain its steps one by one.

Input: An identity problem $(s \approxeq t, \mu)$.
Output: "Yes", if the identity problem is solvable, and "No", if it is not.

(i) While μ contains a cycle of length $n > 1$ do $\mu := \mu^n$.
(ii) $S := \emptyset$
(iii) If $s = t$ then stop with result "Yes".
(iv) If there is a shared position p of s and t such that $s|_p = f(\dots)$ and $t|_p = g(\dots)$ and $f \neq g$ then stop with result "No".
(v) If there is a shared position p of s and t such that $s|_p = x$, $t|_p = g(\dots)$, and x is not an increasing variable then stop with result "No".
Repeat this step with s and t exchanged.
(vi) If there is a shared position p of s and t such that $s|_p = x$, $t|_p = y$, $x \neq y$, and $x, y \notin Dom(\mu)$ then stop with result "No".
(vii) Add the triple $(x, p, t|_p)$ to S for all shared positions p of s and t such that $x = s|_p \neq t|_p$ where x is an increasing variable.
Repeat this step with s and t exchanged.
(viii) If $(x, p_1, u_1) \in S$ and $(x, p_2, u_2) \in S$ where
(a) u_1 and u_2 are not unifiable or where
(b) $u_1 = u_2$ and $p_1 < p_2$,
then stop with result "No".
(ix) $s := s\mu, t := t\mu$
(x) Continue with Step (iii).

Fig. 1. An algorithm to decide solvability of identity problems

First we replace the substitution μ by μ^n such that μ^n does not contain cycles. Here, a substitution δ contains a *cycle* of length n iff $\delta = \{x_1/x_2, x_2/x_3, \dots, x_n/x_1, \dots\}$ where the x_i are pairwise different variables. Obviously, if δ contains a cycle of length n then in δ^n all variables $x_1, \dots, x_n$ do not belong to the domain any more. Thus, Step (i) terminates and afterwards, μ does not contain cycles of length 2 or more.

Note that the identity problem $(s \approxeq t, \mu)$ is solvable iff $(s \approxeq t, \mu^n)$ is solvable. Hence, after Step (i) we still have to decide solvability of $(s \approxeq t, \mu)$ for the modified μ. The advantage is that now μ has a special structure. For all $x \in Dom(\mu)$, either x is an increasing variable or for some n the term $x\mu^n$ is a variable

[5] It could also be possible to express identity problems as primal unification problems and to use an algorithm for primal unification [13] instead. But then one would have to extend the results of [13] to allow arbitrary dependencies of function symbols. Moreover, our algorithm has the advantage of being very easy to implement.

which is not in $Dom(\mu)$. For such substitutions μ, the terms $s, s\mu, s\mu^2, \ldots$ finally become *stationary* at each position, i.e., for every position p there is some n such that either all terms $s\mu^n|_p, s\mu^{n+1}|_p, s\mu^{n+2}|_p, \ldots$ are of the form $f(\ldots)$, or all these terms are the same variable $x \notin Dom(\mu)$. Therefore, it is possible to define $s\mu^\infty$ as the (possibly infinite) term where $\mathrm{root}(s\mu^\infty|_p) = f$ iff $\mathrm{root}(s\mu^n|_p) = f$ for some n, and $s\mu^\infty|_p = x$ iff there is some n such that $s\mu^m|_p = x$ for all $m \geq n$.

If the identity problem is solvable then there is some n such that $s\mu^n = t\mu^n$ which will be detected in Step (iii). The reason is that with Steps (ix) and (x) one iterates over all pairs (s,t), $(s\mu, t\mu)$, $(s\mu^2, t\mu^2)$, ...

If the identity problem is not solvable, then this could be due to a *stationary conflict*, i.e., $s\mu^\infty \neq t\mu^\infty$. Then the identity problem is unsolvable since $s\mu^n = t\mu^n$ would imply $s\mu^\infty = t\mu^\infty$. If the terms $s\mu^\infty$ and $t\mu^\infty$ differ, then there is some position p such that the symbols at position p in $s\mu^\infty$ and $t\mu^\infty$ differ, or $s\mu^\infty|_p$ is a variable and $t\mu^\infty|_p$ is not a variable (or vice versa), or both $s\mu^\infty|_p$ and $t\mu^\infty|_p$ are different variables. Recall that the terms $s, s\mu, s\mu^2, \ldots$ and the terms $t, t\mu, t\mu^2, \ldots$ finally become stationary. Hence, if we choose n high enough, then the conflict at position p can already be detected by inspecting $s\mu^n|_p$ and $t\mu^n|_p$. Thus, then one of three cases in Steps (iv)–(vi) will hold.

With the steps described up to now, we can detect all solvable identity problems and all identity problems which are not solvable due to a stationary conflict. However, there remain other identity problems which are not solvable, but which do not have a stationary conflict. As an example consider $(x \mathrel{\tilde{\approx}} y, \{x/\mathsf{f}(x), y/\mathsf{f}(y)\})$. Then $s\mu^\infty = \mathsf{f}(\mathsf{f}(\ldots))) = t\mu^\infty$ but this identity problem is not solvable since $x\mu^n = \mathsf{f}^n(x) \neq \mathsf{f}^n(y) = y\mu^n$ for all $n \in \mathbb{N}$. We call such identity problems *infinite*.

The remaining steps (ii), (vii), and (viii) are used to detect infinite identity problems. In the set S we store sub-problems (x, p, u) such that whenever the identity problem is solvable, then $x\mu^m = u\mu^m$ must hold for some m to make the terms $s\mu^n$ and $t\mu^n$ equal at position p.

We give some intuition why the two abortion criteria in Step (viii) are correct. For (viii–a), note that if u_1 and u_2 are not unifiable then $x\mu^m$ cannot be both $u_1\mu^m$ and $u_2\mu^m$, which means that the sub-problems (x, p_1, u_1) and (x, p_2, u_2) (resp. $(x \mathrel{\tilde{\approx}} u_1, \mu)$ and $(x \mathrel{\tilde{\approx}} u_2, \mu)$) are not solvable. For (viii–b), in order to make $x\mu^m$ equal to $u_1\mu^m$, we again produced the same problem at a lower position. Then the original identity problem is again not solvable, since this repeated generation of the same sub-problem would continue forever. As usual, $p_1 < p_2$ denotes that position p_1 is strictly above p_2.

The following theorem shows that all answers of the algorithm are indeed correct and it also shows that it always returns an answer. The termination proof is quite involved since we have to show that the criteria in Step (viii) suffice to detect all infinite identity problems.

Theorem 16 (Solving Identity Problems). *The algorithm in Fig. 1 to decide solvability of identity problems is correct and it terminates.*

Proof. One can easily show that in the k-th iteration, S is the following set S_k.

$$S_k = \{(x,p,u) \mid x \in \mathcal{V}_{incr} \wedge x \neq u \wedge \exists m \leq k : \\ (s\mu^m|_p = x \wedge u = t\mu^m|_p) \vee (t\mu^m|_p = x \wedge u = s\mu^m|_p)\}$$

Since the correctness of Steps (i)–(vi) was already illustrated in the explanation of the algorithm, we only prove the correctness of Step (viii) formally. So let (x,p_1,u_1) and (x,p_2,u_2) be elements of some S_k. Hence, there exist $m_1 \leq k$ and $m_2 \leq k$ such that w.l.o.g. for both $i = 1$ and $i = 2$, we have $s\mu^{m_i}|_{p_i} = x$ and $t\mu^{m_i}|_{p_i} = u_i$ where x is a variable with $x \neq u_i$. If the identity problem $(s \approxeq t, \mu)$ is not solvable then there is nothing to show. Otherwise, there is some n with $s\mu^n = t\mu^n$. Since $s\mu^{m_i}|_{p_i} = x \neq u_i = t\mu^{m_i}|_{p_i}$, we know that $n > m_i$ for both i. Hence, we can conclude the following equalities for both $i \in \{1,2\}$:

$$x\mu^{n-m_i} = s\mu^{m_i}|_{p_i}\mu^{n-m_i} = s\mu^n|_{p_i} = t\mu^n|_{p_i} = t\mu^{m_i}|_{p_i}\mu^{n-m_i} = u_i\mu^{n-m_i} \quad (13)$$

Assume that we have applied (viii–a) and the algorithm wrongly returned "No". This directly leads to a contradiction since by (13), $u_1\mu^n = x\mu^n = u_2\mu^n$ proves that u_1 and u_2 are unifiable.

Now assume that we applied (viii–b) and wrongly obtained "No". W.l.o.g. let $p_1 < p_2$. Since $s\mu^{m_1}|_{p_1}$ is the variable x, we must apply μ at least one more time to obtain a term with the position p_2 and thus, $m_1 < m_2$. As $x\mu^{n-m_1} = u_1\mu^{n-m_1}$ by (13), there must be some smallest number $n' \leq n$ such that $x\mu^{n'-m_1} = u_1\mu^{n'-m_1}$ is valid. From $x \neq u_1$ we conclude $n' > m_1$ and from $s\mu^{n'}|_{p_1} = x\mu^{n'-m_1} = u_1\mu^{n'-m_1} = t\mu^{n'}|_{p_1}$ we derive that also the subterms $s\mu^{n'}|_{p_2}$ of $s\mu^{n'}|_{p_1}$ and $t\mu^{n'}|_{p_2}$ of $t\mu^{n'}|_{p_1}$ are identical. Again, $n' > m_2$ must hold and we obtain $x\mu^{n'-m_2} = u_2\mu^{n'-m_2}$. But this is a contradiction to the minimality of n' since $u_1 = u_2$ and $n' - m_2 < n' - m_1$.

To prove termination, we have already argued in the explanation of the algorithm why we can detect all solvable identity problems and all those problems which have a stationary conflict. So it remains to prove that all infinite problems can be detected. To this end, we start with three observations on infinite identity problems, i.e., unsolvable identity problems $(s \approxeq t, \mu)$ where $s\mu^\infty = t\mu^\infty$.

First, if $(s \approxeq t, \mu)$ is infinite then $(s\mu \approxeq t\mu, \mu)$ is infinite as well.

Second, if $(s \approxeq t, \mu)$ is infinite then there is no position p where $(s|_p \approxeq t|_p, \mu)$ has a stationary conflict (i.e., $s|_p\mu^\infty \neq t|_p\mu^\infty$). Otherwise there would also be a stationary conflict for $(s \approxeq t, \mu)$ which contradicts the infinity of $(s \approxeq t, \mu)$.

And third, whenever $(s \approxeq t, \mu)$ is infinite then there is some position p such that $s|_p \neq t|_p$, at least one of the terms $s|_p$ or $t|_p$ is an increasing variable, and $(s|_p\mu \approxeq t|_p\mu, \mu)$ is infinite, too. This can be proved as follows. Let p be one of the longest (i.e., lowest) shared positions of s and t such that $(s|_p \approxeq t|_p, \mu)$ is not solvable. (Such positions must exist since $(s \approxeq t, \mu)$ is not solvable.) Due to the second observation we know that $(s|_p \approxeq t|_p, \mu)$ again is infinite. Moreover, using the maximality of p we conclude that at least one of the terms $s|_p$ or $t|_p$ is a variable. Since $(s|_p \approxeq t|_p, \mu)$ is infinite, this variable must be increasing. Finally, by the first observation, $(s\mu|_p \approxeq t\mu|_p, \mu)$ is infinite as well.

Now we show that if there were an infinite run of the algorithm, we would insert an infinite number of triples into S where the corresponding positions p_0,

$p_0\,p_1$, $p_0\,p_1\,p_2$, ... are getting longer and longer: Since $(s \approx t, \mu)$ is infinite, due to the third observation there is a position p_0 such that a triple $(x_0, p_0, s|_{p_0})$ or $(x_0, p_0, t|_{p_0})$ is added to S. Moreover, $(s\mu|_{p_0} \approx t\mu|_{p_0}, \mu)$ is infinite. Hence, again using the third observation we obtain a position p_1 such that $(s\mu|_{p_0}\mu|_{p_1} \approx t\mu|_{p_0}\mu|_{p_1}, \mu) = (s\mu^2|_{p_0p_1} \approx t\mu^2|_{p_0p_1}, \mu)$ is infinite where $s\mu^2|_{p_0p_1}$ and $t\mu^2|_{p_0p_1}$ are different terms, one of them being an increasing variable x_1. Thus, again the corresponding triple $(x_1, p_0\,p_1, s\mu|_{p_0\,p_1})$ or $(x_1, p_0\,p_1, t\mu|_{p_0\,p_1})$ is added to S. By iterating this reasoning, we obtain the desired infinite sequence of triples in S.

As there exist only finitely many increasing variables, there must be some x which occurs infinitely often in this sequence. Thus, we obtain an infinite subsequence $(x, p_0 \dots p_{i_1}, u_{i_1})$, $(x, p_0 \dots p_{i_2}, u_{i_2})$, ... where $i_1 < i_2 < \dots$ and $p_0 \dots p_{i_1} < p_0 \dots p_{i_2} < \dots$. Due to Kruskal's tree theorem [16], there must be some i_j and i_k such that $i_j < i_k$ and u_{i_j} is embedded in u_{i_k}. If $u_{i_j} = u_{i_k}$ then this is a contradiction to an infinite run of the algorithm since then the criterion in Step (viii–b) would hold and the algorithm would be stopped. Otherwise, u_{i_j} is strictly embedded in u_{i_k}. But then u_{i_j} cannot be unified with u_{i_k} since the embedding relation is stable under substitutions. Hence in that case, the criterion in Step (viii–a) will stop the algorithm. □

Example 17. We illustrate the algorithm with the identity problem $(x \approx y, \mu)$ where $\mu = \{x/\mathsf{f}(y, u_0), y/\mathsf{f}(z, u_0), z/\mathsf{f}(x, u_0), u_0/u_1, u_1/u_0\}$.

As μ contains a cycle of length 2 we replace μ by $\mu^2 = \{x/\mathsf{f}(\mathsf{f}(z, u_0), u_1)$, $y/\mathsf{f}(\mathsf{f}(x, u_0), u_1)$, $z/\mathsf{f}(\mathsf{f}(y, u_0), u_1)\}$. Since $x\mu^\infty = \mathsf{f}(\mathsf{f}(\mathsf{f}(\dots, u_1), u_0), u_1) = y\mu^\infty$, we know that the problem is either solvable or infinite. Hence, the criteria in Steps (iv)–(vi) will never apply. We start with $s = x$ and $t = y$. Since the terms are different we add (x, ε, y) and (y, ε, x) to S. In the next iteration we have $s = \mathsf{f}(\mathsf{f}(z, u_0), u_1)$ and $t = \mathsf{f}(\mathsf{f}(x, u_0), u_1)$. Again, the terms are different and we add $(x, 11, z)$ and $(z, 11, x)$ to S. The next iteration yields the new triples $(y, 1111, z)$ and $(z, 1111, y)$, and after having applied μ three times, we obtain the two last triples $(x, 111111, y)$ and $(y, 111111, x)$. Then due to the criterion (viii–b), the algorithm terminates with "No".

By simply combining all theorems of Sect. 3, we finally obtain a decision procedure which solves the question whether a loop is also an innermost loop.

Corollary 18 (Deciding Innermost Loops). *For every loop*

$$s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \dots \to_{\mathcal{R}} s_m \to_{\mathcal{R}} t \trianglerighteq s_1\mu$$

of a TRS $\mathcal{R}$, it is decidable whether that loop is also an innermost loop.

Example 19. In Ex. 15 we observed that the loop of Ex. 2 is an innermost loop iff $(x \approx y, \mu)$ is not solvable where $\mu = \{x/\mathsf{suc}(x)\}$. We apply the algorithm of Fig. 1 to show that this identity problem is not solvable. Hence, we show that the loop is an innermost loop and thus, the TRS of Ex. 1 is not innermost terminating.

Since μ only contains cycles of length 1, we skip Step (i). So, let $s = x$ and $t = y$. Then none of the steps (iii)–(vi) is applicable. Hence, we add (x, ε, y) to S and continue with $s = \mathsf{suc}(x)$ and $t = y$. Then, in Step (v) the algorithm is stopped with the answer "No" due to a stationary conflict.

4 Integration into the Dependency Pair Framework

In [7], we showed that in order to find loops automatically, it is advantageous to use the *dependency pair framework* [1,6,10] because of a reduced search space. There are two main reasons for this: First, one can drop the contexts when looking for loops, i.e., one can drop the $\trianglerighteq$ in "$\to_{\mathcal{R}}^{+} \circ \trianglerighteq$" and will still be able to detect every looping TRS [7, Thm. 23]. Second and more important, by using dependency pairs one can often prove termination of large parts of the TRS, and hence only has to search for loops for a small subsystem of the original TRS.

While the results of this paper have only been presented for TRSs, it is easy to extend our notion of "innermost looping" (Def. 5) to DP problems – the basic data structure within the dependency pair framework. Then the methods of Sect. 3 can again be used to decide whether a looping DP problem is innermost looping. Moreover, one can extend [7, Thm. 23] to the innermost case, i.e., a TRS is innermost looping iff the corresponding DP problem is innermost looping. The details of these extensions can be found in [21, Chapter 8].

5 Conclusion

To prove non-termination of innermost rewriting, we first extended the notion of a loop to the innermost case. An innermost loop is an innermost reduction with a strong regularity which admits the same infinite reduction as an ordinary loop does for full rewriting. Afterwards, we developed a novel procedure to decide whether a given loop is also an innermost loop. Our procedure can be combined with any method to detect loops for full rewriting, regardless whether it directly searches for loops of the TRS or whether it performs this search within the dependency pair framework.

We have implemented our procedure in combination with dependency pairs in our termination prover AProVE [8] which already featured a method to detect loops, cf. [7]. Note that while proving the soundness and the termination of our novel decision procedure is non-trivial, the procedure itself is very easy to implement. To evaluate its usefulness empirically, we tested it on the *termination problem data base* (TPDB). This is the collection of examples used in the annual *International Competition of Termination Tools* [19]. Currently, the TPDB contains 129 TRSs where at least one tool has been able to disprove termination in the competition in 2007. With the results of this paper, AProVE now also disproves *innermost* termination for 93 of these TRSs (where we use a time limit of 1 minute per example). In contrast, we are not aware of any other existing tool for disproving innermost termination. The fact that from the remaining 36 TRSs at least 30 are innermost terminating demonstrates the power of our approach. Moreover, of course AProVE can also disprove innermost termination of Ex. 1. Concerning efficiency, the check whether a loop that was found is also an innermost loop needs less than 8 seconds in total for all TRSs of the TPDB. For further details on our experiments and to run this new version of AProVE via a web interface, we refer to `http://aprove.informatik.rwth-aachen.de/eval/decidingLoops`.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge, 1998.
3. N. Dershowitz. Termination of rewriting. *J. Symb. Comp.*, 3:69–116, 1987.
4. V. Diekert. Makanin's algorithm. In M. Lothaire, editor, *Combinatorics on Words*, pp. 387–442. Cambridge University Press, 2002.
5. J. Endrullis. Jambox. http://joerg.endrullis.de.
6. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR '04*, LNAI 3452, pp. 301–331, 2005.
7. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. *Proc. FroCoS '05*, LNAI 3717, pp. 216–231, 2005.
8. J. Giesl, P. Schneider-Kamp, R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. *Proc. IJCAR '06*, LNAI 4130, pp. 281–286, 2006.
9. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. RTA '06*, LNCS 4098, pp. 297–312, 2006.
10. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
11. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
12. J. Guttag, D. Kapur, and D. Musser. On proving uniform termination and restricted termination of rewriting systems. *SIAM J. Computation*, 12:189–214, 1983.
13. M. Hermann and R. Galbavý. Unification of infinite sets of terms schematized by primal grammars. *Theoretical Computer Science*, 176(1-2):111–158, 1997.
14. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
15. M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer, 2000.
16. J. B. Kruskal. Well-quasi-orderings, the Tree Theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–223, 1960.
17. W. Kurth. *Termination und Konfluenz von Semi-Thue-Systemen mit nur einer Regel*. PhD thesis, Technische Universität Clausthal, Germany, 1990.
18. D. Lankford and D. Musser. A finite termination criterion. Unpublished Draft. USC Information Sciences Institute, 1978.
19. C. Marché and H. Zantema. The termination competition. In *Proc. RTA '07*, LNCS 4533, pp. 303–313, 2007.
20. É. Payet. Detecting non-termination of term rewriting systems using an unfolding operator. In *Proc. LOPSTR '06*, LNCS 4407, pp. 194–209, 2006.
21. R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting*. PhD thesis, RWTH Aachen University, 2007. Available as Technical Report AIB-2007-17, http://aib.informatik.rwth-aachen.de/2007/2007-17.pdf.
22. D. Vroon. Personal communication, 2007.
23. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. RTA '04*, LNCS 3091, pp. 85–94, 2004.
24. H. Zantema. Termination of string rewriting proved automatically. *Journal of Automated Reasoning*, 34:105–139, 2005.
25. X. Zhang. Overlap closures do not suffice for termination of general term rewriting systems. *Information Processing Letters*, 37(1):9–11, 1991.