Detecting Non-Termination of Term Rewriting Systems Using an Unfolding Operator

Etienne Payet

Detecting Non-Termination of Term Rewriting Systems Using an Unfolding Operator

Étienne Payet

IREMIA - Université de la Réunion, France
email: epayet@univ-reunion.fr

Abstract. In this paper, we present an approach to non-termination of term rewriting systems inspired by a technique that was designed in the context of logic programming. Our method is based on a classical unfolding operation together with semi-unification and is independent of a particular reduction strategy. We also describe a technique to reduce the explosion of rules caused by the unfolding process. The analyser that we have implemented is able to solve most of the non-terminating examples in the Termination Problem Data Base.

1 Introduction

Proving termination of a term rewriting system (TRS) $\mathcal{R}$ consists in proving that *every* term only has finite rewritings with respect to $\mathcal{R}$ (a particular reduction strategy may be used). Termination of TRS's has been subject to an intensive research (see *e.g.* [10, 23] for surveys) that has given rise to several automatic proof methods. One of the most powerful is the dependency pair approach [5], recently extended to the dependency pair framework [14, 15], implemented in the termination prover AProVE [16]. In comparison, the dual problem, *i.e.* non-termination, has hardly been studied. It consists in proving that *there exists* a term that *loops*, *i.e.* that leads to an infinite rewriting. Notice that designing non-termination provers is an important issue as this kind of tools can be used to *disprove* termination, *i.e.* to complement any termination prover. In [15], the authors use the dependency pair framework to combine termination and non-termination analyses. In order to detect non-terminating TRS's, they apply forward or backward narrowing to dependency pairs until they find two terms that semi-unify. Some heuristics are used to select forward or backward narrowing and to get a finite search space.

Termination has also been widely studied in the context of logic programming. One of the approaches that have been introduced so far consists in inferring terminating classes of queries, *i.e.* classes where *every* element only has finite left-derivations with respect to a given logic program. Several automatic tools performing termination inference have been designed, *e.g.* TerminWeb [13] or cTI [19]. But as for term rewriting, there are only a few papers about the dual problem, *i.e.* inference of non-terminating classes of queries (classes where *there exists* an element that loops, *i.e.* that has an infinite left-derivation). In [21,

20], the authors introduce the unfold & infer approach to infer non-terminating classes of queries. First, they unfold the logic program P of interest to a binary logic program BP using the unfolding operator of [12]. By the results in [9], a query loops with respect to BP if and only if it loops with respect to P. Then, to infer looping queries, they consider every rule $A \leftarrow B$ in BP; if the body B is more general (up to some computed neutral argument positions) than the head A, they conclude that A loops with respect to BP, hence with respect to P.

On the *theoretical* level, it can be noticed that the unfold & infer approach also works with TRS's. Indeed, there exists some techniques to unfold a TRS $\mathcal{R}$ to a TRS $\mathcal{U}$ such that if a term loops with respect to $\mathcal{U}$ then it also loops with respect to $\mathcal{R}$ (see for instance [7, 22, 3]). Moreover, semi-unification is a powerful tool for detecting looping terms: if there is a rule $l \to r$ in $\mathcal{U}$ where $l\theta_1\theta_2 = r'\theta_1$ for some substitutions θ_1 and θ_2 and some subterm r' of r, then we can deduce that $l\theta_1$ loops with respect to $\mathcal{U}$, hence with respect to $\mathcal{R}$. Notice that the subsumption order is different from that used in logic programming, where the body has to be more general than the head, while here, $l\theta_1$ has to be more general than $r'\theta_1$. This is due to definition of the operational semantics of both paradigms.

On the *practical* level, however, it is not known how the unfold & infer approach behaves in the context of term rewriting. In this paper, we present our experiments on using the narrowing-based unfolding operation described in [3] together with semi-unification to prove non-termination of TRS's. The first analysis that we describe is very simple but leads to an explosion of the number of generated rules. Hence, we refine it into a second one by providing a mechanism that allows us to eliminate some useless rules produced by the unfolding process. The simple and refined analyses are powerful enough to solve most of the non-terminating examples in the Termination Problem Data Base (TPDB) [25], but the refined one runs much faster. We insist that the results we present herein are independent of any particular reduction strategy. This does not mean that our method is parametric in a reduction strategy but that we always consider the whole rewrite relation and not subsets of it.

Our motivations are the following. We want to design a simple formalism for proving non-termination of TRS's (the unfold & infer theory is very simple and clear, as presented above). We do not want any heutistics as in [15]. Moreover, we want another illustration of the unfold & infer technique which was introduced in the context of logic programming. Such an illustration would provide a connection between the paradigm of logic programming and that of term rewriting, by a transfer of a logic programming technique to term rewriting.

The paper is organized as follows. First, in Sect. 2, we give the basic definitions and fix the notations. Then, in Sect. 3 and Sect. 4, we present a non-termination analysis based on an existing unfolding operation together with semi-unification. In Sect. 5, we refine this analysis and in Sect. 6, we present an implementation and some experiments using TRS's from the TPDB. Finally, Sect. 7 discusses related works and concludes the paper.

2 Preliminaries

We briefly present the basic concepts of term rewriting (details can be found *e.g.* in [6]) and the notations that we use in the paper.

We let $\mathbb{N}$ denote the set of non-negative integers and, for any $n \in \mathbb{N}$, $[1, n]$ denotes the set of all the integers i such that $1 \leq i \leq n$ (if $n = 0$, then $[1, n] = \varnothing$).

From now on, we fix a finite *signature* $\mathcal{F}$, *i.e.* a finite set of *function symbols* where every $\mathsf{f} \in \mathcal{F}$ has a unique *arity*, which is the number of its arguments. We write $\mathsf{f}/n \in \mathcal{F}$ to denote that f is an element of $\mathcal{F}$ whose arity is $n \geq 0$. We also fix an infinite countable set $\mathcal{V}$ of *variables* with $\mathcal{F} \cap \mathcal{V} = \varnothing$. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined as the smallest set such that:

- $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$,
- if $\mathsf{f}/n \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ then $\mathsf{f}(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

For $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $root(t)$ denotes the *root symbol* of t and is defined by:

$$root(t) = \begin{cases} \bot & \text{if } t \in \mathcal{V}, \\ \mathsf{f} & \text{if } t = \mathsf{f}(t_1, \ldots, t_n) \end{cases}$$

where $\bot$ is a special symbol not occurring in $\mathcal{F} \cup \mathcal{V}$. We let $Var(t)$ denote the set of variables occurring in t. The *set of positions* in t, denoted by $Pos(t)$, is defined as:

$$Pos(t) = \begin{cases} \{\varepsilon\} & \text{if } t \in \mathcal{V}, \\ \{\varepsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in Pos(t_i)\} & \text{if } t = \mathsf{f}(t_1, \ldots, t_n) \ . \end{cases}$$

When $p \in Pos(t)$, we write $t|_p$ to denote the subterm of t at position p, with $t|_\varepsilon = t$. We write $t[p \leftarrow s]$ to denote the term obtained from t by replacing $t|_p$ with a term s. We say that p is a *non-variable position of* t if $t|_p$ is not a variable. The set of non-variable positions of t is denoted by $NPos(t)$.

We write substitutions as sets of the form $\{x_1/t_1, \ldots, x_n/t_n\}$ denoting that for each $i \in [1, n]$, variable x_i is mapped to term t_i (note that x_i may occur in t_i). Applying a substitution θ to an object O is denoted by $O\theta$. The *composition* of substitutions θ and η is denoted by $\theta\eta$ and is the substitution that maps any variable x to $(x\theta)\eta$. A term t is *more general than* a term t' when there exists a substitution θ such that $t' = t\theta$. A substitution θ is *more general than* a substitution η when $\eta = \theta\tau$ for some substitution τ. A *renaming* is a substitution that is a 1-1 and onto mapping from its domain to itself. We say that a term t is a *variant* of a term t' if there exists a renaming γ such that $t' = t\gamma$.

Two terms t and t' *unify* when there exists a substitution θ such that $t\theta = t'\theta$. Then we say that θ is a *unifier* of t and t'. A *most general unifier* of t and t' is a unifier of t and t' that is more general than all unifiers of t and t'. We let $mgu(t, t')$ denote the set of most general unifiers of t and t'. We say that t *semi-unifies* with t' if there exists some substitutions θ and θ' such that $t\theta\theta' = t'\theta$.

A *term rewriting system* (TRS) over $\mathcal{F}$ is a set $\mathcal{R} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ of *rewrite rules*, every element $l \to r$ of which is such that $l \notin \mathcal{V}$ and $Var(r) \subseteq$

$Var(l)$. For every s and t in $\mathcal{T}(\mathcal{F},\mathcal{V})$, we write $s \underset{\mathcal{R}}{\rightarrow} t$ if there is a rewrite rule $l \rightarrow r$ in $\mathcal{R}$, a substitution θ and a position p in $Pos(s)$ such that $s|_p = l\theta$ and $t = s[p \leftarrow r\theta]$. We let $\overset{+}{\underset{\mathcal{R}}{\rightarrow}}$ (resp. $\overset{*}{\underset{\mathcal{R}}{\rightarrow}}$) denote the transitive (resp. reflexive and transitive) closure of $\underset{\mathcal{R}}{\rightarrow}$. In this paper, we only consider finite TRS's. We say that a term t *loops* with respect to (w.r.t.) $\mathcal{R}$ when there exists infinitely many terms $t_1, t_2, \ldots$ such that $t \underset{\mathcal{R}}{\rightarrow} t_1 \underset{\mathcal{R}}{\rightarrow} t_2 \underset{\mathcal{R}}{\rightarrow} \cdots$. We say that $\mathcal{R}$ is *non-terminating* when there exists a term that loops with respect to $\mathcal{R}$.

3 Unfolding a TRS

Usually, unfolding a rule of a term rewriting system consists in performing two elementary transformations: instantiation and unfolding (see e.g. [7, 22]). These transformations can be combined into a single one using narrowing:

Definition 1 (Unfolding [3]). *Let $\mathcal{R}$ be a TRS and $l \rightarrow r \in \mathcal{R}$. If for some $l' \rightarrow r' \in \mathcal{R}$ renamed with fresh variables and for some non-variable position p of r we have $\theta \in mgu(r|_p, l')$, then $(l \rightarrow r[p \leftarrow r'])\theta$ is an unfolding of $l \rightarrow r$.*

The non-termination analysis presented in this paper proceeds by iteratively unfolding sets of rules using a fixed TRS. This is why we rephrase Definition 1 above in the form of an unfolding operator that takes two sets of rules as input: the rules X to be unfolded and the rules $\mathcal{R}$ that are used to unfold:

Definition 2 (Unfolding operator). *For every TRS $\mathcal{R}$, the* unfolding operator *$T_\mathcal{R}$ is defined as: for any set X of rewrite rules,*

$$T_\mathcal{R}(X) = \left\{ (l \rightarrow r[p \leftarrow r'])\theta \;\middle|\; \begin{array}{l} l \rightarrow r \in X \\ p \in NPos(r) \\ l' \rightarrow r' \in \mathcal{R} \text{ renamed with fresh variables} \\ \theta \in mgu(r|_p, l') \end{array} \right\} .$$

Notice that this operator is not monotone. As in [2], the *unfolding sequence starting from* $\mathcal{R}$ is

$$\begin{array}{lcll} T_\mathcal{R} \uparrow 0 & = & \mathcal{R} & \\ T_\mathcal{R} \uparrow (n+1) & = & T_\mathcal{R}(T_\mathcal{R} \uparrow n) & \forall n \in \mathbb{N} . \end{array}$$

Example 1 (Giesl, Thiemann and Schneider-Kamp [15], Example 28). Consider

$$\mathcal{R} = \left\{ \mathsf{f}(x,y,z) \rightarrow \mathsf{g}(x,y,z),\ \mathsf{g}(\mathsf{s}(x),y,z) \rightarrow \mathsf{f}(z,\mathsf{s}(y),z) \right\} .$$

We have:

- $T_\mathcal{R} \uparrow 0 = \mathcal{R}$.
- $T_\mathcal{R} \uparrow 1 = T_\mathcal{R}(T_\mathcal{R} \uparrow 0)$. If we take $\mathsf{f}(x,y,z) \rightarrow \mathsf{g}(x,y,z)$ in $T_\mathcal{R} \uparrow 0$, $p = \varepsilon$, $l' \rightarrow r'$ as $\mathsf{g}(\mathsf{s}(x_1), y_1, z_1) \rightarrow \mathsf{f}(z_1, \mathsf{s}(y_1), z_1)$ in $\mathcal{R}$, $\theta = \{x/\mathsf{s}(x_1), y/y_1, z/z_1\}$, we get the rule
$$\mathsf{f}(\mathsf{s}(x_1), y_1, z_1) \rightarrow \mathsf{f}(z_1, \mathsf{s}(y_1), z_1)$$
as an element of $T_\mathcal{R} \uparrow 1$. □

Example 2 (Toyama [24]). Consider:

$$\mathcal{R} = \left\{ \mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x),\ \mathsf{g}(x,y) \to x,\ \mathsf{g}(x,y) \to y \right\} .$$

We have:

- $T_{\mathcal{R}} \uparrow 0 = \mathcal{R}$.
- $T_{\mathcal{R}} \uparrow 1 = T_{\mathcal{R}}(T_{\mathcal{R}} \uparrow 0)$. Notice that the rules $\mathsf{g}(x,y) \to x$ and $\mathsf{g}(x,y) \to y$ in $T_{\mathcal{R}} \uparrow 0$ cannot be unfolded because there are no non-variable positions in the right-hand side. The rule $\mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x)$ cannot be unfolded too because $\mathsf{f}(x,x,x)$, the only non-variable subterm in the right-hand side, cannot be unified with any variant of a left-hand side. So, $T_{\mathcal{R}} \uparrow 1 = \varnothing$. □

In Sect. 4 below, in order to prove non-termination, we consider the rules $l \to r$ in the unfolding sequence. If l semi-unifies with a subterm of r, then we deduce that $\mathcal{R}$ is non-terminating. Notice that using this mechanism directly, one gets a very limited tool that is unable to solve the smallest examples.

Example 3 (Example 2 continued). $\mathcal{R}$ is known to be non-terminating (for instance, $\mathsf{f}(0,1,\mathsf{g}(0,1))$ loops). Note that $T_{\mathcal{R}} \uparrow 0 = \mathcal{R}$ and for each $n \in \mathbb{N} \setminus \{0\}$, $T_{\mathcal{R}} \uparrow n = \varnothing$. As no left-hand side in $\mathcal{R}$ semi-unifies with a subterm of the corresponding right-hand side, we cannot conclude. □

In order to get a practical analyser, a solution consists in pre-processing the TRS $\mathcal{R}$ of interest by replacing every variable with the left-hand side of each rule of $\mathcal{R}$. The intuition is that as a variable represents any term, it stands in particular for a term that can be rewritten.

Definition 3 (Augmented TRS). *Let $\mathcal{R}$ be a TRS. The* augmented TRS $\mathcal{R}^+$ *is defined modulo renaming as follows: $\mathcal{R}^+$ consists of all the rules $(l \to r)\theta$ where $l \to r$ is an element of $\mathcal{R}$ and θ is a substitution of the form $\{x_1/t_1, \ldots, x_n/t_n\}$ (with $n \in \mathbb{N}$) such that $\{x_1, \ldots, x_n\} \subseteq Var(l)$ and for each $i \in [1,n]$, t_i is a variant of a left-hand side in $\mathcal{R}$ and is variable disjoint from $l \to r$ and from every t_j, $j \in [1,n] \setminus \{i\}$. Note that θ can be empty (take $n = 0$).*

Example 4 (Example 2 continued). The rule $\mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x)$ only contains variable x. Hence, we consider the substitutions

$$\theta_0 = \varnothing,\ \theta_1 = \{x/\mathsf{f}(0,1,x_1)\} \text{ and } \theta_2 = \{x/\mathsf{g}(x_1,y_1)\}$$

and apply them to $\mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x)$. This leads, respectively, to:

$$\begin{array}{c} \mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x) \\ \mathsf{f}(0,1,\mathsf{f}(0,1,x_1)) \to \mathsf{f}(\mathsf{f}(0,1,x_1),\mathsf{f}(0,1,x_1),\mathsf{f}(0,1,x_1)) \\ \mathsf{f}(0,1,\mathsf{g}(x_1,y_1)) \to \mathsf{f}(\mathsf{g}(x_1,y_1),\mathsf{g}(x_1,y_1),\mathsf{g}(x_1,y_1)) \ . \end{array}$$

The variables in rules $\mathsf{g}(x,y) \to x$ and $\mathsf{g}(x,y) \to y$ are x and y. So, we consider the above substitutions θ_0, θ_1, θ_2 together with

$$\begin{array}{ll} \theta_3 = \{y/\mathsf{f}(0,1,x_1)\} & \theta_4 = \{y/\mathsf{g}(x_1,y_1)\} \\ \theta_5 = \{x/\mathsf{f}(0,1,x_1), y/\mathsf{f}(0,1,x_2)\} & \theta_6 = \{x/\mathsf{f}(0,1,x_1), y/\mathsf{g}(x_2,y_2)\} \\ \theta_7 = \{x/\mathsf{g}(x_1,y_1), y/\mathsf{f}(0,1,x_2)\} & \theta_8 = \{x/\mathsf{g}(x_1,y_1), y/\mathsf{g}(x_2,y_2)\} \end{array}$$

that lead to (the rules on the left are obtained from $\mathsf{g}(x,y) \to x$ and those on the right from $\mathsf{g}(x,y) \to y$):

$$\begin{array}{lcc}
\theta_0: & \mathsf{g}(x,y) \to x & \mathsf{g}(x,y) \to y \\
\theta_1: & \mathsf{g}(\mathsf{f}(0,1,x_1),y) \to \mathsf{f}(0,1,x_1) & \mathsf{g}(\mathsf{f}(0,1,x_1),y) \to y \\
\theta_2: & \mathsf{g}(\mathsf{g}(x_1,y_1),y) \to \mathsf{g}(x_1,y_1) & \mathsf{g}(\mathsf{g}(x_1,y_1),y) \to y \\
\vdots & \vdots & \vdots
\end{array}$$

Now, we can compute the unfolding sequence starting from $\mathcal{R}^+$ instead of $\mathcal{R}$. From the rule

$$\mathsf{f}(0,1,\mathsf{g}(x_1,y_1)) \to \mathsf{f}(\mathsf{g}(x_1,y_1),\mathsf{g}(x_1,y_1),\mathsf{g}(x_1,y_1))$$

computed above, using position 1 of the right-hand side and $\mathsf{g}(x_2,y_2) \to x_2$ in $\mathcal{R}$, we get $\mathsf{f}(0,1,\mathsf{g}(x_1,y_1)) \to \mathsf{f}(x_1,\mathsf{g}(x_1,y_1),\mathsf{g}(x_1,y_1))$ as an element of $T_\mathcal{R} \uparrow 1$. Then, from this new rule, using position 2 of the right-hand side together with $\mathsf{g}(x_3,y_3) \to y_3$ in $\mathcal{R}$, we get $\mathsf{f}(0,1,\mathsf{g}(x_1,y_1)) \to \mathsf{f}(x_1,y_1,\mathsf{g}(x_1,y_1))$ as an element of $T_\mathcal{R} \uparrow 2$. As $\mathsf{f}(0,1,\mathsf{g}(x_1,y_1))\theta_1\theta_2 = \mathsf{f}(x_1,y_1,\mathsf{g}(x_1,y_1))\theta_1$ for $\theta_1 = \{x_1/0, y_1/1\}$ and $\theta_2 = \varnothing$, we conclude that $\mathcal{R}$ is non-terminating. □

Following the intuitions of the preceding example, we give these new definitions:

Definition 4 (Unfolding semantics). *The* augmented unfolding sequence of $\mathcal{R}$ *is*

$$\begin{array}{lcl}
T_\mathcal{R} \uparrow 0 & = & \mathcal{R}^+ \\
T_\mathcal{R} \uparrow (n+1) & = & T_\mathcal{R}(T_\mathcal{R} \uparrow n) \quad \forall n \in \mathbb{N}\,.
\end{array}$$

The unfolding semantics $unf(\mathcal{R})$ *of* $\mathcal{R}$ *is the limit of the unfolding process described in Definition 2, starting from* $\mathcal{R}^+$*:*

$$unf(\mathcal{R}) = \bigcup_{n \in \mathbb{N}} T_\mathcal{R} \uparrow n\,.$$

Notice that the least fixpoint of $T_\mathcal{R}$ is the empty set. Moreover, $unf(\mathcal{R})$ is not a fixpoint of $T_\mathcal{R}$. This is because $\mathcal{R}^+ \subseteq unf(\mathcal{R})$ (because $T_\mathcal{R} \uparrow 0 = \mathcal{R}^+$) but we do not necessarily have $\mathcal{R}^+ \subseteq T_\mathcal{R}(unf(\mathcal{R}))$ because

$$T_\mathcal{R}(unf(\mathcal{R})) = T_\mathcal{R}(\bigcup_{n \in \mathbb{N}} T_\mathcal{R} \uparrow n) = \bigcup_{n \in \mathbb{N}} T_\mathcal{R}(T_\mathcal{R} \uparrow n) = \bigcup_{n \in \mathbb{N}\setminus\{0\}} T_\mathcal{R} \uparrow n\,.$$

In the logic programming framework, every clause $H \leftarrow B$ of the binary unfoldings specifies that a call to H necessarily leads to a call to B. In the context of term rewriting, we get the following counterpart:

Proposition 1. *Let* $\mathcal{R}$ *be a TRS. If* $l \to r \in unf(\mathcal{R})$ *then* $l \xrightarrow[\mathcal{R}]{+} r$*.*

This result allows us to prove that the unfoldings exhibit the termination properties of a term rewriting system:

Theorem 1. *Let* $\mathcal{R}$ *be a TRS and* t *be a term. Then,* t *loops w.r.t.* $\mathcal{R}$ *if and only if* t *loops w.r.t.* $unf(\mathcal{R})$*.*

4 Inferring Looping Terms

The unfoldings of a TRS can be used to infer terms that loop, hence to prove non-termination. It suffices to add semi-unification [18] to Proposition 1. Notice that semi-unification encompasses both matching and unification. A polynomial-time algorithm for semi-unification can be found in [17].

Theorem 2. *Let $\mathcal{R}$ be a TRS. Suppose that for $l \to r \in unf(\mathcal{R})$ there is a subterm r' of r such that $l\theta_1\theta_2 = r'\theta_1$ for some substitutions θ_1 and θ_2. Then, $l\theta_1$ loops w.r.t. $\mathcal{R}$.*

In order to use Theorem 2 as a practical tool, one can for instance fix a maximum number of iterations of the unfolding operator.

Example 5 (Example 4 continued). $\mathsf{f}(0,1,\mathsf{g}(x_1,y_1)) \to \mathsf{f}(x_1,y_1,\mathsf{g}(x_1,y_1))$ is an element of $T_{\mathcal{R}} \uparrow 2$ with $\mathsf{f}(0,1,\mathsf{g}(x_1,y_1))\theta_1\theta_2 = \mathsf{f}(x_1,y_1,\mathsf{g}(x_1,y_1))\theta_1$ for $\theta_1 = \{x_1/0, y_1/1\}$ and $\theta_2 = \varnothing$. Hence, $\mathsf{f}(0,1,\mathsf{g}(x_1,y_1))\theta_1 = \mathsf{f}(0,1,\mathsf{g}(0,1))$ loops with respect to $\mathcal{R}$. □

Example 6 (Example 1 continued). $\mathsf{f}(\mathsf{s}(x_1),y_1,z_1) \to \mathsf{f}(z_1,\mathsf{s}(y_1),z_1)$ is an element of $T_{\mathcal{R}} \uparrow 1$ with $\mathsf{f}(\mathsf{s}(x_1),y_1,z_1)\theta_1\theta_2 = \mathsf{f}(z_1,\mathsf{s}(y_1),z_1)\theta_1$ for $\theta_1 = \{z_1/\mathsf{s}(x_1)\}$ and $\theta_2 = \{y_1/\mathsf{s}(y_1)\}$. Hence, $\mathsf{f}(\mathsf{s}(x_1),y_1,z_1)\theta_1 = \mathsf{f}(\mathsf{s}(x_1),y_1,\mathsf{s}(x_1))$ loops with respect to $\mathcal{R}$. □

Example 7 (file `Rubio-inn/test76.trs` *in the TPDB).* Consider

$$\mathcal{R} = \left\{ \begin{array}{l} \mathsf{f}(0,\mathsf{s}(0),x) \to \mathsf{f}(x,+(x,x),x), \quad +(x,\mathsf{s}(y)) \to \mathsf{s}(+(x,y)), \\ +(x,0) \to x, \quad \mathsf{g}(x,y) \to x, \quad \mathsf{g}(x,y) \to y \end{array} \right\} .$$

The augmented TRS $\mathcal{R}^+$ contains the rule

$$R_0 = \mathsf{f}(0,\mathsf{s}(0),\mathsf{g}(x_0,y_0)) \to \mathsf{f}(\mathsf{g}(x_0,y_0),+(\mathsf{g}(x_0,y_0),\mathsf{g}(x_0,y_0)),\mathsf{g}(x_0,y_0))$$

obtained from $\mathsf{f}(0,\mathsf{s}(0),x) \to \mathsf{f}(x,+(x,x),x)$ and substitution $\{x/\mathsf{g}(x_0,y_0)\}$.

- If we take position $p = 2.2$ in the right-hand side of R_0, $\mathsf{g}(x_1,y_1) \to x_1$ in $\mathcal{R}$ and $\theta = \{x_1/x_0, y_1/y_0\}$, we get the rule

  $$R_1 = \mathsf{f}(0,\mathsf{s}(0),\mathsf{g}(x_0,y_0)) \to \mathsf{f}(\mathsf{g}(x_0,y_0),+(\mathsf{g}(x_0,y_0),x_0),\mathsf{g}(x_0,y_0))$$

  as an element of $T_{\mathcal{R}} \uparrow 1$.
- If we take position $p = 2$ in the right-hand side of R_1, $+(x_2,0) \to x_2$ in $\mathcal{R}$ and $\theta = \{x_0/0, x_2/\mathsf{g}(0,y_0)\}$, we get the rule

  $$R_2 = \mathsf{f}(0,\mathsf{s}(0),\mathsf{g}(0,y_0)) \to \mathsf{f}(\mathsf{g}(0,y_0),\mathsf{g}(0,y_0),\mathsf{g}(0,y_0))$$

  as an element of $T_{\mathcal{R}} \uparrow 2$.

– If we take position $p = 1$ in the right-hand side of R_2, $\mathsf{g}(x_3, y_3) \to x_3$ in $\mathcal{R}$ and $\theta = \{x_3/0, y_3/y_0\}$, we get the rule

$$R_3 = \mathsf{f}(0, \mathsf{s}(0), \mathsf{g}(0, y_0)) \to \mathsf{f}(0, \mathsf{g}(0, y_0), \mathsf{g}(0, y_0))$$

as an element of $T_{\mathcal{R}} \uparrow 3$.
– If we take position $p = 2$ in the right-hand side of R_3, $\mathsf{g}(x_4, y_4) \to y_4$ in $\mathcal{R}$ and $\theta = \{x_4/0, y_4/y_0\}$, we get the rule

$$R_4 = \mathsf{f}(0, \mathsf{s}(0), \mathsf{g}(0, y_0)) \to \mathsf{f}(0, y_0, \mathsf{g}(0, y_0))$$

as an element of $T_{\mathcal{R}} \uparrow 4$.

Notice that the left-hand side $\mathsf{f}(0, \mathsf{s}(0), \mathsf{g}(0, y_0))$ of R_4 semi-unifies with the right-hand side $\mathsf{f}(0, y_0, \mathsf{g}(0, y_0))$ for $\theta_1 = \{y_0/\mathsf{s}(0)\}$ and $\theta_2 = \varnothing$. Consequently, $\mathsf{f}(0, \mathsf{s}(0), \mathsf{g}(0, y_0))\theta_1 = \mathsf{f}(0, \mathsf{s}(0), \mathsf{g}(0, \mathsf{s}(0)))$ loops with respect to $\mathcal{R}$. □

5 Eliminating Useless Rules

The operator of Definition 2 produces many useless rules, *i.e.* rules that cannot be unfolded to $l \to r$ where l semi-unifies with a subterm of r.

Example 8 (Example 4 continued). The augmented TRS $\mathcal{R}^+$ contains the rule

$$\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(0, 1, x_1), \mathsf{f}(0, 1, x_1), \mathsf{f}(0, 1, x_1)) \ .$$

The left-hand side does not semi-unify with any subterm of the right-hand side. Applying $T_{\mathcal{R}}$ to this rule, one gets:

$$\begin{aligned}
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(x_1, x_1, x_1), \mathsf{f}(0, 1, x_1), \mathsf{f}(0, 1, x_1)) \\
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(0, 1, x_1), \mathsf{f}(x_1, x_1, x_1), \mathsf{f}(0, 1, x_1)) \\
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(0, 1, x_1), \mathsf{f}(0, 1, x_1), \mathsf{f}(x_1, x_1, x_1)) \ .
\end{aligned}$$

None of these new rules satisfies the semi-unification criterion. Applying $T_{\mathcal{R}}$ again, one gets:

$$\begin{aligned}
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(x_1, x_1, x_1), \mathsf{f}(x_1, x_1, x_1), \mathsf{f}(0, 1, x_1)) \\
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(x_1, x_1, x_1), \mathsf{f}(0, 1, x_1), \mathsf{f}(x_1, x_1, x_1)) \\
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(0, 1, x_1), \mathsf{f}(x_1, x_1, x_1), \mathsf{f}(x_1, x_1, x_1)) \\
&\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(x_1, x_1, x_1), \mathsf{f}(0, 1, x_1), \mathsf{f}(x_1, x_1, x_1)) \ .
\end{aligned}$$

None of these rules satisfies the semi-unification criterion. Finally, unfolding one more time leads to:

$$\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1)) \to \mathsf{f}(\mathsf{f}(x_1, x_1, x_1), \mathsf{f}(x_1, x_1, x_1), \mathsf{f}(x_1, x_1, x_1)),$$

a rule that does not satisfy the semi-unification criterion and cannot be unfolded. □

5.1 Abstraction

The analysis described in the preceding sections leads to an explosion of the number of generated rules (this is illustrated by the results of Sect. 6). A solution to reduce this explosion consists in designing a mechanism that detects, as soon as possible, rules that are useless for proving non-termination. We can also notice that the semi-unification criterion we introduced before consists in checking, for each subterm of a right-hand side, that the corresponding left-hand side semi-unifies. One disadvantage of this technique is that a same semi-unification test may be performed several times.

Example 9 (Example 8 continued). The left-hand side of each rule computed in Example 8 is $\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1))$. Moreover, each rule, except the last one, has $\mathsf{f}(0, 1, x_1)$ as a subterm of the right-hand side. Consequently, semi-unification of $\mathsf{f}(0, 1, \mathsf{f}(0, 1, x_1))$ with $\mathsf{f}(0, 1, x_1)$ is checked several times. □

In order to avoid any repetition of the same semi-unification test, one solution consists in making those tests explicit by "flattening" each rule $l \to r$ into pairs of terms (l, r') where r' is a subterm of r. Then, semi-unification test on a pair (l, r') is only performed at the root position of r'.

Following these intuitions, we introduce a new domain.

Definition 5 (Abstract domain). *An* abstract TRS *is a finite set, each element of which is either a pair of terms or* true *or* false. *The* abstract domain $P^{\#}$ *is the set of all abstract TRS's.*

The special element true denotes any pair of terms (l, r) such that l semi-unifies with r. The special element false corresponds to any non-useful pair of terms:

Definition 6 (Useful pair). *Let $\mathcal{R}$ be a TRS. A pair (l, r) of terms is* useful *for $\mathcal{R}$ when it can be unfolded, using the rules of $\mathcal{R}$, to a pair (l_1, r_1) where l_1 semi-unifies with r_1.*

The set $P^{\#}$ is a sort of *abstract domain*, the corresponding *concrete domain* of which is the set $P^{\flat}$ of TRS's as defined in Sect. 2. The *abstraction function* that transforms a concrete TRS to an abstract one is defined as follows.

Definition 7 (Abstraction function). *The* abstraction function α *maps every element $\mathcal{R}$ of $P^{\flat}$ to an element of $P^{\#}$ as follows:*

$$\alpha(\mathcal{R}) = \bigcup_{l \to r \in \mathcal{R}} \left\{ \alpha_{\mathcal{R}}(l, r|_p) \mid p \in Pos(r) \right\}$$

where, for any pair (l, r) of terms,

$$\alpha_{\mathcal{R}}(l, r) = \begin{cases} \textit{if } l \textit{ semi-unifies with } r \textit{ then } \mathsf{true} \\ \textit{else if } (l, r) \textit{ is useful for } \mathcal{R} \textit{ then } (l, r) \\ \textit{else } \mathsf{false} \end{cases}$$

The operator that we use to unfold abstract TRS's is defined as follows.

Definition 8 (Abstract unfolding operator). *Let $\mathcal{R}$ be a concrete TRS. For any abstract TRS $X^\#$, if* $\mathsf{true} \in X^\#$ *then* $T^\#_\mathcal{R}(X^\#) = \{\mathsf{true}\}$*, otherwise*

$$T^\#_\mathcal{R}(X^\#) = \left\{ \alpha_\mathcal{R}(l\theta, r[p \leftarrow r']\theta) \;\middle|\; \begin{array}{l} (l,r) \in X^\# \\ p \in NPos(r) \\ l' \to r' \in \mathcal{R} \text{ renamed with fresh variables} \\ \theta \in mgu(r|_p, l') \end{array} \right\}$$

This operator allows us to define an abstract semantics.

Definition 9 (Abstract unfolding semantics). *Let $\mathcal{R}$ be a concrete TRS. The* abstract unfolding sequence of $\mathcal{R}$ *is*

$$\begin{array}{lcl} T^\#_\mathcal{R} \uparrow 0 & = & \alpha(\mathcal{R}^+) \\ T^\#_\mathcal{R} \uparrow (n+1) & = & T^\#_\mathcal{R}(T^\#_\mathcal{R} \uparrow n) \quad \forall n \in \mathbb{N}\,. \end{array}$$

The abstract unfolding semantics *$unf^\#(\mathcal{R})$ of $\mathcal{R}$ is the limit of the unfolding process described in Definition 8:*

$$unf^\#(\mathcal{R}) = \bigcup_{n \in \mathbb{N}} T^\#_\mathcal{R} \uparrow n\,.$$

The relevance of a non-termination analysis based on these notions is clarified by the following correctness result.

Proposition 2 (Correctness). *Let $\mathcal{R}$ be a concrete TRS. If* $\mathsf{true} \in unf^\#(\mathcal{R})$*, then $\mathcal{R}$ is non-terminating.*

5.2 Detecting Useful Pairs

The intuitions and results of this section rely on the following observation.

Lemma 1. *If (l,r) is a useful pair of terms where l is not a variable, then (l,r) can be unfolded to a pair (l_1,r_1) such that $root(l_1) = root(l)$ and $root(r_1) \in \{root(l), \bot\}$.*

Consider a useful pair of terms (l,r). Then, l semi-unifies with r or (l,r) can be unfolded, in at least one step, to (l_1,r_1) such that l_1 semi-unifies with r_1. By Definition 2, the latter case corresponds to narrowing r to r_1 in at least one step and then in applying to l the computed substitution θ to get l_1. As there is at least one step of narrowing, r cannot be a variable. Hence, r has the form $\mathsf{f}(t_1,\ldots,t_n)$. Let us consider the possible forms of the narrowing from r to r_1.

1. There does not exist a step of the narrowing that is performed at the root position, *i.e.* r_1 has the form $\mathsf{f}(t'_1,\ldots,t'_n)$ and, roughly, each t_i is narrowed to t'_i, in 0 or more steps.
2. There exists a step of the narrowing that is performed at the root position, *i.e.* (roughly) first each t_i is narrowed (in 0 or more steps) to a term t'_i then $\mathsf{f}(t'_1,\ldots,t'_n)$ is narrowed at root position using a rule $\mathsf{f}(s_1,\ldots,s_n) \to \cdots$ whose right-hand side further leads to r_1.

Consider the first case above when l is not a variable. As $root(r_1) \neq \bot$, by Lemma 1 $root(r_1) = root(l)$ so l has the form $\mathsf{f}(s_1, \ldots, s_n)$. Hence, by Lemma 1 again, l_1 has the form $\mathsf{f}(s'_1, \ldots, s'_n)$. Notice that for each $i \in [1, n]$, t_i is narrowed to t'_i and s'_i semi-unifies with t'_i. Consequently, (s_i, t_i) is a useful pair.

Now, consider the second case above, again when l is not a variable. We note that the following result holds.

Lemma 2. *Let $\mathsf{f}(t_1, \ldots, t_n)$ be a term where each t_i can be narrowed to t'_i, in 0 or more steps. Suppose that for a term $\mathsf{f}(s_1, \ldots, s_n)$, we have*

$$mgu(\mathsf{f}(t'_1, \ldots, t'_n), \mathsf{f}(s_1, \ldots, s_n) \text{ renamed with fresh variables}) \neq \varnothing \ .$$

Then, each t_i unifies with any variable disjoint variant of s_i or can be narrowed in at least one step to a term whose root symbol is that of s_i or $\bot$.

Moreover, the right-hand side of the rule $\mathsf{f}(s_1, \ldots, s_n) \to \cdots$ has to lead to r_1, *i.e.*, by Lemma 1, to a term whose root symbol is that of l or $\bot$. This corresponds to a path in the graph of functional dependencies that we define as follows, in the style of [4, 1].

Definition 10 (Graph of functional dependencies). *The* graph of functional dependencies *induced by a concrete TRS $\mathcal{R}$ is denoted by $\mathcal{G}_\mathcal{R}$. The following transformation rules define the edges E and the initial vertices I of $\mathcal{G}_\mathcal{R}$:*

$$\frac{l \to r \in \mathcal{R}}{\langle \mathcal{R}, E, I \rangle \mapsto \langle \mathcal{R} \setminus \{l \to r\}, E \cup \{l \to root(r)\}, I \cup \{l\} \rangle}$$

$$\frac{l \to \mathsf{f} \in E \wedge l' \to \mathsf{g} \in E \wedge l \in I \wedge l' \in I \wedge \mathsf{f} \notin I \wedge \mathsf{g} \notin I \wedge (root(l') = \mathsf{f} \vee \mathsf{f} = \bot)}{\langle \mathcal{R}, E, I \rangle \mapsto \langle \mathcal{R}, E \cup \{\mathsf{f} \to l'\}, I \rangle}$$

To build $\mathcal{G}_\mathcal{R}$, the algorithm starts with $\langle \mathcal{R}, \varnothing, \varnothing \rangle$ and applies the transformation rules as long as they add new arrows.

Example 10. Consider Toyama's example again:

$$\mathcal{R} = \left\{ \mathsf{f}(0, 1, x) \to \mathsf{f}(x, x, x),\ \mathsf{g}(x, y) \to x,\ \mathsf{g}(x, y) \to y \right\} .$$

The graph $\mathcal{G}_\mathcal{R}$ can be depicted as follows:

$$\boxed{\mathsf{g}(x, y)} \longleftarrow \bot \longrightarrow \boxed{\mathsf{f}(0, 1, x)} \longleftarrow \mathsf{f}$$

where the boxes correspond to the initial vertices. □

Notice that the initial vertices of $\mathcal{G}_\mathcal{R}$ are the left-hand sides of the rules of $\mathcal{R}$. Hence, a path in $\mathcal{G}_\mathcal{R}$ from an initial vertex s to a symbol f indicates that any term s' such that $mgu(s, s'$ renamed with fresh variables$) \neq \varnothing$ *may* be narrowed (using the rules of $\mathcal{R}$) to a term t with $root(t) = \mathsf{f}$. The first step of such a narrowing is performed at the root position of s'. We synthesize case 2 above by the following definition.

Definition 11 (The transition relation $\xrightarrow[\mathcal{G}_\mathcal{R}]{+}$). *Let $\mathcal{G}_\mathcal{R}$ be the graph of functional dependencies of a concrete TRS $\mathcal{R}$, $\mathsf{f}(t_1,\dots,t_n)$ be a term and g be a function symbol or $\bot$. We write $\mathsf{f}(t_1,\dots,t_n) \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \mathsf{g}$ if there exists a non-empty path in $\mathcal{G}_\mathcal{R}$ from an initial vertex of the form $\mathsf{f}(s_1,\dots,s_n)$ to g and, for each $i \in [1,n]$, one of these conditions holds:*

- *$mgu(t_i, s_i$ renamed with fresh variables$) \neq \varnothing$,*
- *$t_i \xrightarrow[\mathcal{G}_\mathcal{R}]{+} root(s_i)$ or $t_i \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \bot$.*

Example 11 (Example 10 continued). $\mathsf{g}(\mathsf{g}(0,0),1) \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \bot$ holds as there is a non-empty path from $\mathsf{g}(x,y)$ to $\bot$ and $\mathsf{g}(0,0) \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \bot$ (because there is a non-empty path from $\mathsf{g}(x,y)$ to $\bot$ and $\mathsf{g}(0,0)$ unifies with $\mathsf{g}(x,y)$) and 1 unifies with y. □

Finally, we synthesize both cases 1 and 2 above as follows:

Definition 12 (The relation $useful_\mathcal{R}$). *For any concrete TRS $\mathcal{R}$ and any terms l and r, we write $useful_\mathcal{R}(l,r)$ if one of these conditions holds:*

- *l semi-unifies with r,*
- *$l = \mathsf{f}(s_1,\dots,s_n)$, $r = \mathsf{f}(t_1,\dots,t_n)$ and, for each $i \in [1,n]$, $useful_\mathcal{R}(s_i,t_i)$,*
- *$l = \mathsf{g}(s_1,\dots,s_m)$, $r = \mathsf{f}(t_1,\dots,t_n)$ and $r \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \mathsf{g}$ or $r \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \bot$.*

Note that in the third condition, we may have $\mathsf{g}/m = \mathsf{f}/n$.

This definition allows us to compute a superset of the set of useful pairs:

Proposition 3 (Completeness). *Let $\mathcal{R}$ be a concrete TRS and (l,r) be a pair of terms. If (l,r) is useful for $\mathcal{R}$, then $useful_\mathcal{R}(l,r)$ holds.*

In order to get a practical tool from the theory of Sect. 5.1, we use the relation $useful_\mathcal{R}$ in function $\alpha_\mathcal{R}$ of Definition 7.

Example 12. Consider Toyama's example. In $\mathcal{R}^+$, one can find the rule:

$$\mathsf{f}(0,1,\mathsf{f}(0,1,x_1)) \to \mathsf{f}(\mathsf{f}(0,1,x_1),\mathsf{f}(0,1,x_1),\mathsf{f}(0,1,x_1))$$

(see Example 4). Let l and r be the left and right-hand side of this rule, respectively. Notice that l does not semi-unify with r, so the first condition of Definition 12 is not satisfied. Let us try the second one. As the root symbols of l and r are identical, we check if each argument of l is in relation with the corresponding argument of r. This test fails for the first argument: we do not have $useful_\mathcal{R}(0,\mathsf{f}(0,1,x_1))$ because 0 does not semi-unify with $\mathsf{f}(0,1,x_1)$ and in $\mathcal{G}_\mathcal{R}$ there is no path from a vertex of the form $\mathsf{f}(\dots)$ to 0 or to $\bot$. Finally, the third condition of Definition 12 is not satisfied as well because neither $r \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \mathsf{f}$ nor $r \xrightarrow[\mathcal{G}_\mathcal{R}]{+} \bot$ holds. Hence, we do not have $useful_\mathcal{R}(l,r)$, so (l,r) is not useful for $\mathcal{R}$ and we get $\alpha_\mathcal{R}(l,r) = \mathsf{false}$. Consequently, this pair will be eliminated. □

6 Experimental Results

We have implemented two analysers, one performing concrete analyses as described in Sect. 4 and the other performing abstract analyses as described in Sect. 5. Both are written in C++ and are available at

`www.univ-reunion.fr/~epayet/Research/TRS/TRSanalyses.html`

Our analysers compute the concrete or abstract unfolding sequence until a user-fixed maximum number of iterations is reached or a looping term is found.

Despite its name, the `nontermin` directory of the TPDB [25] contains subdirectories with terminating TRS's:

- in `AG01`, only `#4.2.trs`, `#4.3.trs`, `#4.4.trs`, `#4.5.trs`, `#4.7.trs` and `#4.12a.trs`, `#4.13.trs`, `#4.14.trs`, `#4.15.trs`, `#4.16.trs`, `#4.17.trs`, `#4.18.trs`, `#4.19.trs` are non-terminating;
- in `cariboo`, all the TRS's are non-terminating except `tricky1.trs`;
- in `CSR`, all the TRS's are non-terminating except `Ex49_GM04.trs`;
- in `Rubio-inn`, all the TRS's are non-terminating except `test830.trs`.

We have run our analysers together with AProVE 1.2 on all the non-terminating TRS's in the `nontermin` directory. We have also run these programs on Example 26, Example 26-2, Example 29, Example 34 and Example 40 of [5] and on Example 28 and footnote 8 of [15]. We fixed a 2 minutes time limit. Using a PowerPC G4, 1.25 GHz, 512 Mo DDR SDRAM, MacOS 10.4.6, we get the results in Table 1. Timings are average over 5 runs. In column "gen" we have reported the number of rules generated by the unfolding process. The abstract analyser runs

directory	concrete analysis			abstract analysis			AProVE 1.2	
	solved	gen	min:sec	solved	gen	min:sec	solved	min:sec
`AG01`	12/13	31218	0:56	12/13	9471	0:41	11/13	3:00
`cariboo`	6/6	833	0:00	6/6	234	0:00	6/6	0:06
`CSR`	36/36	39937	2:03	36/36	128	0:00	36/36	1:26
`HM`	1/1	8	0:00	1/1	6	0:00	1/1	0:00
`Rubio-inn`	8/9	33436	2:04	8/9	19283	2:01	7/9	2:29
`TRCSR`	1/1	13	0:00	1/1	2	0:00	1/1	0:04
[5]	5/5	397	0:00	5/5	73	0:00	5/5	0:01
[15]	2/2	932	0:00	2/2	292	0:00	2/2	0:00
total	71/73	106774	5:03	71/73	29489	2:42	69/73	7:06

Table 1.

much faster than its counterparts. The best total score in column "solved" is achieved by both the concrete and abstract analysers. As expected, the abstract analyser produces much fewer rules than the concrete one.

The TRS `#4.13.trs` in subdirectory `AG01` was given by Drosten [11]:

$$\mathcal{R} = \left\{ \begin{array}{l} \mathtt{f}(0,1,x) \to \mathtt{f}(x,x,x),\ \mathtt{f}(x,y,z) \to 2,\ 0 \to 2,\ 1 \to 2, \\ \mathtt{g}(x,x,y) \to y,\ \mathtt{g}(x,y,y) \to x \end{array} \right\} .$$

AProVE answers "maybe" within the time limit when run on this TRS. Both $unf(\mathcal{R})$ and $unf^{\#}(\mathcal{R})$ are finite (for each $n \geq 11$, $T_{\mathcal{R}} \uparrow n = \varnothing$ and for each $n \geq 5$, $T^{\#}_{\mathcal{R}} \uparrow n = \varnothing$). These sets are computed by our analysers before the time limit is reached. No rule in $unf(\mathcal{R})$ satisfies the semi-unification criterion and $\mathtt{true} \notin unf^{\#}(\mathcal{R})$. So, this TRS is an example of failure of our method that is not caused by the explosion of the unfolding process.

7 Conclusion

We have presented an automatic technique for proving non-termination of TRS's independently of a particular reduction strategy. It is based on the "unfold & infer" mechanism that was designed in the context of logic programming, thus establishing a connection between both paradigms. We have also described a method for eliminating useless rules to reduce the search space. Notice that we did not implement such a method in our logic programming non-termination tool as unfolding in this context is less explosive than with TRS's (because the particular left-to-right selection rule is classically considered). We have also run our analyser on TRS's from the TPDB; the results are very encouraging as our tool is able to solve 71 over 73 non-terminating examples.

In comparison, the AProVE system solves 69 examples and is slower. The technique implemented in AProVE consists in narrowing dependency pairs until two terms that semi-unify are found. Narrowing operations are performed either directly with the rules of the TRS of interest (forward narrowing) or with the reversed rules (backward narrowing). To select forward or backward narrowing, heuristics are introduced: if the TRS is right and not left-linear, then forward narrowing is performed, otherwise backward narrowing is used. To obtain a finite search space, an upper bound is used on the number of times that a rule can be applied for narrowing. An approximation of the graph of dependency pairs is also constructed and AProVE processes the strongly connected components of this graph separately.

Our approach directly works with the rules (not the dependency pairs) and forward narrowing is sufficient as we pre-process the TRS's. We also do not need heuristics and in order to get a finite search space, we introduce a user-fixed maximum number of iterations. The graph that we use is not a graph of dependency pairs and is closely related to that of [4, 1]. In these papers, the authors define a framework for the static analysis of the unsatisfiability of equation sets. This framework uses a loop-checking technique based on a graph of functional dependencies. Notice that in order to eliminate useless rules within our approach, an idea would consist in using the results of [4, 1] as we are also interested in a form of satisfiability: is a pair of terms (l, r) unfoldable to (l', r') such that l' semi-unifies with r'? However, [4, 1] consider unification instead

of semi-unification and both sides of the pairs can be rewritten (whereas the unfolding operation only rewrites the right-hand side). We are also aware of the work described in [8] where the authors consider a graph of terms to detect loops in the search tree. The graph of terms is used within a dynamic approach whereas our paper and [4, 1] consider a static approach. Another future work consists in designing a bottom-up technique for proving non-termination of TRS's. What we describe in this paper is a top-down mechanism, as the unfolding process starts from the rules of the TRS $\mathcal{R}$ of interest and then rewrites the right-hand sides down as much as possible. In [21, 20], the authors use the unfolding operator T_P^β of [12] that leads to a bottom-up computation of the unfoldings of P starting from the emptyset, instead of P. Given a set of rules X, $T_P^\beta(X)$ unfolds P using the elements of X whereas $T_\mathcal{R}(X)$ unfolds X using the rules of $\mathcal{R}$.

Acknowledgements. We greatly thank an anonymous reviewer for many constructive comments. We also thank Fred Mesnard, Germán Puebla and Fausto Spoto for encouraging us to submit the paper.

References

1. M. Alpuente, M. Falaschi, and F. Manzo. Analyses of unsatisfiability for equational logic programming. *Journal of Logic Programming*, 311(1–3):479–525, 1995.
2. M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Safe folding/unfolding with conditional narrowing. In *Proc. of ALP/HOA 97*, pages 1–15, 1997.
3. M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Rules + strategies for transforming lazy functional logic programs. *Theoretical Computer Science*, 311(1–3):479–525, 2004.
4. M. Alpuente, M. Falaschi, M. J. Ramis, and G. Vidal. Narrowing approximations as an optimization for equational logic programs. In *Proc. of PLILP 1993*, pages 391–409, 1993.
5. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
6. F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge, 1998.
7. R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.
8. J. Chabin and P. Réty. Narrowing directed by a graph of terms. In G. Goos and J. Hartmanis, editors, *Proc. of RTA'91*, volume 488 of *LNCS*, pages 112–123. Springer-Verlag, Berlin, 1991.
9. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
10. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.
11. K. Drosten. *Termersetzungssysteme: Grundlagen der Prototyp-Generierung algebraischer Spezifikationen.* Springer Verlag, Berlin, 1989.
12. M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In *Proc. of SAC'94*, pages 394–399. ACM Press, 1994.
13. S. Genaim and M. Codish. Inferring termination conditions for logic programs using backwards analysis. In R. Nieuwenhuis and A. Voronkov, editors, *Proc. of LPAR'01*, volume 2250 of *LNCS*, pages 685–694. Springer-Verlag, Berlin, 2001.

14. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: combining techniques for automated termination proofs. In F. Baader and A. Voronkov, editors, *Proc. of LPAR'04*, volume 3452 of *LNAI*, pages 210–220. Springer-Verlag, 2004.
15. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In B. Gramlich, editor, *Proc. of FroCoS'05*, volume 3717 of *LNAI*, pages 216–231. Springer-Verlag, 2005.
16. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In V. van Oostrom, editor, *Proc. of RTA'04*, volume 3091 of *LNCS*, pages 210–220. Springer-Verlag, 2004.
17. D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. *Theoretical Computer Science*, 81:169–187, 1991.
18. D.S. Lankford and D.R. Musser. A finite termination criterion. Unpublished Draft, USC Information Sciences Institute, Marina Del Rey, CA, 1978.
19. F. Mesnard and R. Bagnara. cTI: a constraint-based termination inference tool for iso-prolog. *Theory and Practice of Logic Programming*, 5(1–2):243–257, 2005.
20. E. Payet and F. Mesnard. Non-termination inference for constraint logic programs. In R. Giacobazzi, editor, *Proc. of SAS'04*, volume 3148 of *LNCS*, pages 377–392. Springer-Verlag, 2004.
21. E. Payet and F. Mesnard. Non-termination inference of logic programs. *ACM Transactions on Programming Languages and Systems*, 28, Issue 2:256–289, 2006.
22. A. Pettorossi and M. Proietti. Rules and strategies for transforming functional and logic programs. *ACM Comput. Surv.*, 28(2):360–414, 1996.
23. J. Steinbach. Simplification orderings: history of results. *Fundamenta Informaticae*, 24:47–87, 1995.
24. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.
25. Termination Problem Data Base. `http://www.lri.fr/~marche/termination-competition/`.