Loop detection in term rewriting using the eliminating unfoldings

Étienne Payet

IREMIA - Université de la Réunion, France

Abstract

In this paper, we present a fully automatizable approach to detecting loops in standard term rewriting. Our method is based on semi-unification and an unfolding operation which processes both forwards and backwards and considers variable subterms. We also describe a technique to reduce the explosion of rules caused by the unfolding process. The idea is to eliminate from the set of unfoldings some rules that are estimated as useless for detecting loops. This is done by an approximation which consists in pruning the left-hand or right-hand side of the rules used to unfold. The analyser that we have implemented is able to solve most of the examples from the Termination Competition'07 that do not terminate due to a loop.

Key words: term rewriting system, non-termination, loop, unfoldings, useless rule

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 $\mathcal{R}$. Termination of TRSs has been subject to an intensive research (see *e.g.* [15,41] for surveys) that has given rise to several automatic proof methods. One of the most powerful is the dependency pair *approach* [7], recently extended to the dependency pair *framework* [23,24], implemented in the termination prover AProVE [25,22,6]. In comparison, the dual problem, *i.e.* non-termination, has hardly been studied. It consists in proving that *there exists* a term 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 [24] non-termination checks consist in applying forward or backward narrowing to dependency pairs until the

Email address: epayet@univ-reunion.fr (Étienne Payet).

left-hand side of a narrowed pair semi-unifies with the corresponding right-hand side. In [47] non-termination checks consist in encoding string rewrite sequences as propositional formulæ which are satisfiable whenever the corresponding sequence includes a looping reduction. In [48] a fragment of the ancestor graph is constructed and non-termination proofs consist in checking whether this fragment is cyclic; cyclicity implies that an infinite reduction exists. In [45] non-termination checks are based on match-bounds and use a Boolean combination of match-height properties of a given string rewrite system.

Termination has also been widely studied in 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 [19] or cTI [35]. 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 has an infinite left-derivation). The unfold & infer approach introduced in [38,37] consists in first unfolding the logic program P of interest to a binary program BP using the operator of [18]. By the results in [11], a query is non-terminating with respect to BP if and only if it is non-terminating with respect to P. Then, every rule $A \leftarrow B$ in BP is examined; if the body B matches (up to some computed neutral argument positions) the head A, one can conclude that A is non-terminating with respect to BP, hence with respect to P.

In theory, the unfold & infer approach also works with TRSs as there exist several techniques (see for instance [9,14,26,39,4]) to unfold a TRS $\mathcal{R}$ to a TRS $\mathcal{U}$ such that if $l \to r$ is a rule of $\mathcal{U}$ then l rewrites to r using the rules of $\mathcal{R}$ (written $l \xrightarrow[\mathcal{R}]{+} r$). Suppose that $l \to r$ is a rule of $\mathcal{U}$ and l *semi-unifies* with a subterm r' of r, *i.e.* $l\theta_1\theta_2 = r'\theta_1$ for some substitutions θ_1 and θ_2. Then, as $l \xrightarrow[\mathcal{R}]{+} r$, we have $l\theta_1 \xrightarrow[\mathcal{R}]{+} r\theta_1$ *i.e.* $l\theta_1 \xrightarrow[\mathcal{R}]{+} C[r'\theta_1]$ for some context C. Therefore, $l\theta_1 \xrightarrow[\mathcal{R}]{+} C[l\theta_1\theta_2]$, *i.e.* $l\theta_1$ *loops* with respect to $\mathcal{R}$, which implies that $l\theta_1$ is non-terminating with respect to $\mathcal{R}$. Consequently, unfoldings+semi-unification provide a simple technique to detect loops, a special form of non-termination. Notice that the subsumption order is different from that used in logic programming, where the body has to match the head, while here $l\theta_1$ has to match $r'\theta_1$; this is due to the definition of the operational semantics of both paradigms. Semi-unification encompasses both matching and unification; for instance, suppose that $\mathsf{f}(\mathsf{s}(x), y, z) \to \mathsf{f}(z, \mathsf{s}(y), z)$ is a rule of $\mathcal{U}$; $\mathsf{f}(\mathsf{s}(x), y, z)$ does not unify with $\mathsf{f}(z, \mathsf{s}(y), z)$ and $\mathsf{f}(\mathsf{s}(x), y, z)$ does not match $\mathsf{f}(z, \mathsf{s}(y), z)$; however, $\mathsf{f}(\mathsf{s}(x), y, z)\theta_1\theta_2 = \mathsf{f}(z, \mathsf{s}(y), z)\theta_1$ for $\theta_1 = \{z/\mathsf{s}(x)\}$ and $\theta_2 = \{y/\mathsf{s}(y)\}$ so $\mathsf{f}(\mathsf{s}(x), y, z)\theta_1 = \mathsf{f}(\mathsf{s}(x), y, \mathsf{s}(x))$ loops with respect to $\mathcal{R}$.

In [36], we initiated the design of an automatic loop detection technique for TRSs using the unfold & infer approach. We continue this work in the present paper and we strictly extend the results of [36]. As in [36], we only consider standard rewriting and disregard the issue of evaluation strategies. In Section 2 we introduce the notations. In Section 3 we consider an unfolding operator that processes both forwards and backwards and considers variable positions (hence it generates a superset of the overlap closure [26]). As expected, this unfolding operator leads to an explosion of the number of generated rules, which directly affects the naive loop detection analysis that we present in Section 4 and that consists in a "brute force" enumeration of the unfoldings. Hence in Section 5 and Section 6, we refine this analysis by providing a mechanism that allows us to eliminate some rules estimated as useless produced by the unfolding process. The idea is to approximate the unfoldings by pruning the left-hand or right-hand side of the rules used to unfold. In practice, the refined analysis is much more efficient than the naive one and is able to detect most of the TRSs of the Termination Competition'07 [33] that admit a loop. This is shown in Section 7 where we present the experimental evaluation we conducted using rewriting systems from the competition. In Section 8 we discuss related works and we conclude in Section 9 with further improvements.

2 Preliminaries

We briefly present the basic concepts of term rewriting (details can be found *e.g.* in [8]) 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 $\mathsf{f}/m \in \mathcal{F}$ to denote that f is an element of $\mathcal{F}$ whose arity is $m \geq 0$. We also fix an infinite countable set $\mathcal{V}$ of *variables* with $\mathcal{F} \cap \mathcal{V} = \varnothing$. Elements of $\mathcal{F}$ are denoted by $\mathsf{f}, \mathsf{g}, \mathsf{h}, \mathsf{0}, \mathsf{1}, \ldots$ and elements of $\mathcal{V}$ by $x, y, z, \ldots$ The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. For any $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we let $\mathit{Var}(t)$ denote the set of variables occurring in t and $\mathit{Pos}(t)$ denote the set of positions in t. We let ε denote the root position. When $p \in \mathit{Pos}(t)$, we write $t|_p$ to denote the subterm of t at position p. 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.

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). The set of variables $\{x_1, \ldots, x_n\}$ is the *domain* of the substitution. We let $Dom(\theta)$ denote the domain of substitution θ. The application of θ to a syntactic object o is denoted by $o\theta$. The set of instances of o is denoted by $instances(o)$; this notation is naturally extended to sets of syntactic objects. For any syntactic objects o and o', we let $mgu(o, o')$ denote the (up to variable renaming) most general unifier of o and o'. We say that o *semi-unifies* with o' when there exist some substitutions θ and θ' such that $o\theta\theta' = o'\theta$.

The elements (l, r) of $\mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ are rather written as $l \to r$. We say that l is the *left-hand side* and r is the *right-hand side* of $l \to r$. We say that $l \to r$ is a *rewrite rule* (or a *rule*) over $\mathcal{F} \cup \mathcal{V}$ when $l \notin \mathcal{V}$ and $Var(r) \subseteq Var(l)$. A *term rewriting system* (TRS) over $\mathcal{F} \cup \mathcal{V}$ is a finite set of rewrite rules over $\mathcal{F} \cup \mathcal{V}$. In this paper, we consider rules modulo variable renaming. Any new occurrence of a rule is always renamed apart (*i.e.* contains fresh variables not previously met).

Given a TRS $\mathcal{R}$ and some terms s and t, we write $s \underset{\mathcal{R}}{\to} t$ if there is a rewrite rule $l \to r$ 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 $\underset{\mathcal{R}}{\overset{+}{\to}}$ (resp. $\underset{\mathcal{R}}{\overset{*}{\to}}$) denote the transitive (resp. reflexive and transitive) closure of $\underset{\mathcal{R}}{\to}$. We say that a term t is *non-terminating* with respect to (*w.r.t.*) $\mathcal{R}$ when there exist infinitely many terms $t_1, t_2, \ldots$ such that $t \underset{\mathcal{R}}{\to} t_1 \underset{\mathcal{R}}{\to} t_2 \underset{\mathcal{R}}{\to} \cdots$. We say that $\mathcal{R}$ is *non-terminating* if there exists a non-terminating term *w.r.t.* $\mathcal{R}$. A term t *loops w.r.t.* $\mathcal{R}$ when $t \underset{\mathcal{R}}{\overset{+}{\to}} C[t\theta]$ for some context C and substitution θ. $\mathcal{R}$ is *looping* or *admits a loop* when there exists a term that loops *w.r.t.* $\mathcal{R}$. It is well-known that if a term loops *w.r.t.* $\mathcal{R}$ then it is non-terminating *w.r.t.* $\mathcal{R}$.

If Y is an operator from a set E to itself, then for any $e \in E$ we let

$$\begin{array}{lcl} (Y \uparrow 0)(e) & = & e \\ (Y \uparrow n+1)(e) & = & Y\Big((Y \uparrow n)(e)\Big) \quad \forall n \in \mathbb{N}\,. \end{array}$$

A total function f is denoted by $\mapsto$. If $f(x) = x$ then x is a *fixpoint* of f. A *poset* $\langle S, \leq \rangle$ is a set S with a reflexive, transitive and antisymmetric relation $\leq$. An *upper (respectively, lower) bound* of $S' \subseteq S$ is an element $u \in S$ such that $u' \leq u$ (respectively, $u' \geq u$) for every $u' \in S'$. A *complete lattice* is a tuple $\langle S, \leq, \sqcup, \sqcap, m, M \rangle$ where $\langle S, \leq \rangle$ is a poset where *least upper bounds* (denoted by $\sqcup$) and *greatest lower bounds* (denoted by $\sqcap$) always exist and $m = \sqcup\varnothing = \sqcap S$ is the *least element* and $M = \sqcap\varnothing = \sqcup S$ is the *greatest element*. If $\langle C, \leq \rangle$ and $\langle A, \preceq \rangle$ are posets, then $f : C \mapsto A$ is *(co-)additive* if it preserves the least upper bounds (respectively, the greatest lower bounds).

We recall now the basics of abstract interpretation [12]. Let $\langle C, \leq \rangle$ and $\langle A, \preceq \rangle$ be two posets (the concrete and the abstract domains). A *Galois connection* is a pair of monotonic maps $\alpha : C \mapsto A$ and $\gamma : A \mapsto C$ such that $\gamma\alpha$ is extensive and $\alpha\gamma$ is reductive. It is a *Galois insertion* when $\alpha\gamma$ is the identity map *i.e.* when the abstract domain does not contain *useless* elements. This is equivalent to α being onto, or γ one-to-one. Note that a Galois insertion can always be derived from a Galois connection by identifying, in the same equivalence class, all abstract elements having the same concretisation under γ. If C and A are complete lattices and α is additive (respectively, γ is co-additive), then α is the abstraction map (respectively, γ is the concretisation map) of a Galois connection. In a Galois connection, γ induces α and vice versa. Namely, given γ we can define $\alpha(c) = \sqcap\{a \mid c \leq \gamma(a)\}$. Hence it is enough to provide γ in order to specify a Galois connection. An abstract operator $\hat{f} : A^n \mapsto A$ is *correct w.r.t.* $f : C^n \to C$ if $\alpha f\gamma \preceq \hat{f}$. The composition of correct operators is correct. If $f\gamma \leq \gamma\hat{f}$ then, by monotonicity, we have $\alpha f\gamma \preceq \alpha\gamma\hat{f}$. If the abstraction is a Galois insertion, $\alpha\gamma$ is the identity map, so that we have $\alpha f\gamma \preceq \hat{f}$. It follows that, in a Galois insertion, $\hat{f}$ is correct *w.r.t.* f when $f\gamma \leq \gamma\hat{f}$. The advantage of this second formulation of correctness is that it does not require the use of the abstraction map α. By Tarski's fixpoint theorem [42], if f and $\hat{f}$ are continuous (hence, in particular, additive) on the complete lattices $\langle C, \leq, \sqcup, \sqcap, m, M \rangle$ and $\langle A, \preceq, \vee, \wedge, \hat{m}, \hat{M} \rangle$ respectively, then $\sqcup_{0 \leq n}(f \uparrow n)(m)$ is the least fixpoint (*lfp*) of f and $\vee_{0 \leq n}(\hat{f} \uparrow n)(\hat{m})$ is that of $\hat{f}$. Moreover, if $\hat{f}$ is correct *w.r.t.* f, then $\mathit{lfp}(\hat{f})$ is correct *w.r.t.* $\mathit{lfp}(f)$ [13], *i.e.* $\mathit{lfp}(f) \leq \gamma(\mathit{lfp}(\hat{f}))$ in a Galois insertion.

3 Unfolding a TRS

Usually, unfolding a set X of rules using a TRS $\mathcal{R}$ consists in performing two elementary transformations (see e.g. [9,39]):

(Instantiation) if $l \to r \in X$, one can add the new rule $(l \to r)\theta$ for any substitution θ.

(Forward unfolding) if $l \to r \in X$, $p \in \mathit{Pos}(r)$ and $r|_p = l'\theta$ for some rule $l' \to r' \in \mathcal{R}$ and some substitution θ, then one can add the new (unfolded) rule $l \to r[p \leftarrow r'\theta]$.

This can be reformulated using an unfolding operator:

Definition 3.1 (Forward unfoldings using instantiation)

$$I_{\mathcal{R}}(X) = \left\{ l \to r[p \leftarrow r'\theta] \;\middle|\; \begin{array}{l} l \to r \in \mathit{instances}(X), \quad p \in \mathit{Pos}(r) \\ l' \to r' \in \mathcal{R},\ r|_p = l'\theta \text{ for a substitution } \theta \end{array} \right\} .$$

Example 3.2 (Toyama [43]) *Consider:*

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

- *If we take* $l \to r$ *as* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(g(x_1,y_1),g(x_1,y_1),g(x_1,y_1))$ *(which is an instance of* $\mathsf{f}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x) \in \mathcal{R}$*),* p *as* 1*,* $l' \to r'$ *as* $\mathsf{g}(x,y) \to x$ *and* θ *as* $\{x/x_1, y/y_1\}$*, we get* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(x_1,g(x_1,y_1),g(x_1,y_1))$ *as an element of* $I_{\mathcal{R}}(\mathcal{R})$*.*
- *If we take* $l \to r$ *as* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(x_1,g(x_1,y_1),g(x_1,y_1)) \in I_{\mathcal{R}}(\mathcal{R})$*,* p *as* 2*,* $l' \to r'$ *as* $\mathsf{g}(x_2,y_2) \to y_2$ *and* θ *as* $\{x_2/x_1, y_2/y_1\}$*, then we get the rule* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(x_1,y_1,g(x_1,y_1))$ *as an element of* $(I_{\mathcal{R}} \uparrow 2)(\mathcal{R})$*.*

In [4], these transformations are combined into a single one using narrowing [27]: if $R = l \to r \in X$, p is a *non-variable position* of r, $l' \to r' \in \mathcal{R}$ and $r|_p$ and l' unify, then one can add the new rule $(l \to r[p \leftarrow r'])\theta$ where $\theta = mgu(r|_p, l')$.

Example 3.3 *Consider* $\mathcal{R}$ *in Example 3.2. The set of unfoldings of* $\mathcal{R}$ *using narrowing as described above is empty. Indeed, in the right-hand side of* $\mathsf{g}(x,y) \to x$ *and* $\mathsf{g}(x,y) \to y$ *there are no non-variable subterms. Moreover, in the right-hand side of* $\mathsf{f}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x)$ *the only non-variable subterm is* $\mathsf{f}(x,x,x)$ *and no left-hand side of a renamed rule of* $\mathcal{R}$ *unifies with* $\mathsf{f}(x,x,x)$*.*

In this paper, we consider the rules $l \to r$ in the unfoldings of $\mathcal{R}$ in order to prove that $\mathcal{R}$ admits a loop. If l and r satisfy a criterion that we will precise later, then we can conclude existence of a loop. Example 3.3 above shows that if we use the unfolding technique of [4], we get a rather limited approach that is unable to solve the smallest problems. Indeed, $\mathcal{R}$ in Example 3.2 is known to admit a loop (for instance $\mathsf{f}(\mathsf{0},\mathsf{1},\mathsf{g}(\mathsf{0},\mathsf{1}))$ loops), but as the set of unfoldings of $\mathcal{R}$ is empty, we cannot prove anything. Moreover, we want to design a completely automatic tool; this does not seem possible from the unfolding technique of Definition 3.1 which is based on the infinite set of instances of X. A solution to meet our goals consists in also considering variable subterms in the technique of [4].

Definition 3.4 (Forward unfoldings) *We let*

$$F_{\mathcal{R}}(X) = \left\{ (l \to r[p \leftarrow r'])\theta \;\middle|\; \begin{array}{l} l \to r \in X,\ p \in Pos(r) \\ l' \to r' \in \mathcal{R},\ \theta = mgu(r|_p, l') \end{array} \right\}.$$

Example 3.5 *Consider* $\mathcal{R}$ *in Example 3.2.*

- *If we take* $l \to r$ *as* $\mathsf{f}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x) \in \mathcal{R}$*,* p *as* 1*,* $l' \to r'$ *as* $\mathsf{g}(x_1,y_1) \to x_1$ *and* θ *as* $\{x/\mathsf{g}(x_1,y_1)\}$*, we get* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(x_1,g(x_1,y_1),g(x_1,y_1))$ *as an element of* $F_{\mathcal{R}}(\mathcal{R})$*. We also obtained this rule in Example 3.2.*

- *If we take* $l \to r$ *as* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(x_1,g(x_1,y_1),g(x_1,y_1)) \in F_{\mathcal{R}}(\mathcal{R})$, p *as* 2, $l' \to r'$ *as* $\mathsf{g}(x_2,y_2) \to y_2$ *and* θ *as* $\{x_2/x_1, y_2/y_1\}$, *we get the rule* $\mathsf{f}(\mathsf{0},\mathsf{1},g(x_1,y_1)) \to \mathsf{f}(x_1,y_1,g(x_1,y_1))$ *as an element of* $(F_{\mathcal{R}} \uparrow 2)(\mathcal{R})$. *We also obtained this rule in Example 3.2.*

Notice that an approach based on Definition 3.4 is theoretically less powerful than one based on Definition 3.1 because of the following result. However, Definition 3.4 is usable in practice, unlike Definition 3.1.

Lemma 3.6 $F_{\mathcal{R}}(X) \subseteq I_{\mathcal{R}}(X)$ *always holds, but* $I_{\mathcal{R}}(X) \subseteq F_{\mathcal{R}}(X)$ *does not.*

The fact that $I_{\mathcal{R}}(X)$ is not generally included in $F_{\mathcal{R}}(X)$ is not caused by the use of *most general* unifiers: the same result holds if one replaces $\theta = mgu(r|_p, l')$ in Definition 3.4 with θ *is a unifier of* $r|_p$ *and* l'. Consider for instance $\mathcal{R} = \{\mathsf{f}(\mathsf{g}(x)) \to x,\ \mathsf{0} \to \mathsf{1}\}$; as $\mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{0}))) \to \mathsf{g}(\mathsf{0})$ is an instance of $\mathsf{f}(\mathsf{g}(x)) \to x$, the rule $R = \mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{0}))) \to \mathsf{g}(\mathsf{1})$ is an element of $I_{\mathcal{R}}(\{\mathsf{f}(\mathsf{g}(x)) \to x\})$; however, we have $R \notin F_{\mathcal{R}}(\{\mathsf{f}(\mathsf{g}(x)) \to x\})$ even if we use unifiers instead of most general unifiers in Definition 3.4.

Definition 3.4 consists in rewriting the right-hand side of the rules of X using the rules of $\mathcal{R}$ forwards. A variant of this technique consists in proceeding backwards, *i.e.* in rewriting the left-hand side of the rules of X using the rules of $\mathcal{R}$ backwards.

Definition 3.7 (Backward unfoldings) *We let*

$$B_{\mathcal{R}}(X) = \left\{ (l[p \leftarrow l'] \to r)\theta \;\middle|\; \begin{array}{l} l \to r \in X,\ p \in Pos(l) \\ l' \to r' \in \mathcal{R},\ \theta = mgu(l|_p, r') \end{array} \right\}.$$

Clearly, Definition 3.1 can be modified to proceed backwards. This leads to an operator $I'_{\mathcal{R}}$ which is such that $B_{\mathcal{R}}(X) \subseteq I'_{\mathcal{R}}(X)$ always holds but the converse does not.

Example 3.8 *Consider* $\mathcal{R}$ *in Example 3.2.*

- *If we take* $l \to r$ *as* $\mathsf{f}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x) \in \mathcal{R}$, p *as* 1, $l' \to r'$ *as* $\mathsf{g}(x_1,y_1) \to x_1$ *and* θ *as* $\{x_1/\mathsf{0}\}$, *we get* $\mathsf{f}(\mathsf{g}(\mathsf{0},y_1),\mathsf{1},x) \to \mathsf{f}(x,x,x)$ *as an element of* $B_{\mathcal{R}}(\mathcal{R})$.
- *If we take* $l \to r$ *as* $\mathsf{f}(\mathsf{g}(\mathsf{0},y_1),\mathsf{1},x) \to \mathsf{f}(x,x,x) \in B_{\mathcal{R}}(\mathcal{R})$, p *as* 2, $l' \to r'$ *as* $\mathsf{g}(x_2,y_2) \to y_2$ *and* θ *as* $\{y_2/\mathsf{1}\}$, *we get* $\mathsf{f}(\mathsf{g}(\mathsf{0},y_1),\mathsf{g}(x_2,\mathsf{1}),x) \to \mathsf{f}(x,x,x)$ *as an element of* $(B_{\mathcal{R}} \uparrow 2)(\mathcal{R})$.

In the sequel of this paper, we decide to unfold both forwards and backwards. As we also consider variable subterms, we get an unfolding operator whose underlying principle relies on the paramodulation rule [29]. Using forward unfoldings only or backward unfoldings only is not sufficient as we have:

Lemma 3.9 *$F_{\mathcal{R}}(X) \subseteq B_{\mathcal{R}}(X)$ and $B_{\mathcal{R}}(X) \subseteq F_{\mathcal{R}}(X)$ do not always hold.*

Section 4 provides examples of TRSs that cannot be proved to admit a loop with forward (backward) unfoldings only.

Definition 3.10 (Unfoldings) *We consider the following set of unfoldings of X w.r.t. $\mathcal{R}$:*

$$U_{\mathcal{R}}(X) = F_{\mathcal{R}}(X) \cup B_{\mathcal{R}}(X) \ .$$

The unfolding semantics is defined as follows, in the style of [3].

Definition 3.11 (Unfolding semantics) *The unfolding semantics $unf(\mathcal{R})$ of $\mathcal{R}$ is the limit of the unfolding process described in Definition 3.10, starting from $\mathcal{R}$:*

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

The only differences between $unf(\mathcal{R})$ and the *overlap closure* [26] of $\mathcal{R}$ (denoted by $OC(\mathcal{R})$) are the following. We consider variable subterms while [26] does not, hence $unf(\mathcal{R})$ is a superset of $OC(\mathcal{R})$. Moreover, in order to compute $unf(\mathcal{R})$, one overlaps unfoldings with the rules of $\mathcal{R}$ whereas in order to compute $OC(\mathcal{R})$, one overlaps closures with closures. The next result is well-known for overlap closures and straightforwardly extends to $unf(\mathcal{R})$.

Proposition 3.12 ([26]) *If $l \to r \in unf(\mathcal{R})$ then $l \xrightarrow[\mathcal{R}]{+} r$.*

4 Inferring terms that loop

The unfoldings of a TRS $\mathcal{R}$ can be used to infer terms that loop *w.r.t.* $\mathcal{R}$. It suffices to add semi-unification [32] to Proposition 3.12. Semi-unification encompasses both matching and unification. A polynomial-time algorithm for semi-unification can be found in [30].

Theorem 4.1 *If 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. $\mathcal{R}$.*

The set $unf(\mathcal{R})$ is possibly infinite, but for any $n \in \mathbb{N}$, $(U_{\mathcal{R}} \uparrow n)(\mathcal{R})$ is finite (modulo renaming of variables). So, in order to use Theorem 4.1 as a practical tool, one can for instance fix a maximum number of iterations of $U_{\mathcal{R}}$. Another alternative consists in fixing a time limit. Notice that Theorem 4.1 can only detect terms that loop, hence TRSs which are non-terminating but not looping cannot be handled.

Example 4.2 *Again, consider $\mathcal{R}$ in Example 3.2.*

- *As the rule* $\mathsf{f}(\mathsf{0},\mathsf{1},\mathsf{g}(x_1,y_1)) \to \mathsf{f}(x_1,y_1,\mathsf{g}(x_1,y_1))$ *is an element of* $(F_{\mathcal{R}} \uparrow 2)(\mathcal{R})$ *(see Example 3.5), it belongs to* $unf(\mathcal{R})$. *As* $\mathsf{f}(\mathsf{0},\mathsf{1},\mathsf{g}(x_1,y_1))\theta_1\theta_2 = \mathsf{f}(x_1,y_1,\mathsf{g}(x_1,y_1))\theta_1$ *for* $\theta_1 = \{x_1/\mathsf{0}, y_1/\mathsf{1}\}$ *and* $\theta_2 = \varnothing$, *we can conclude that* $\mathsf{f}(\mathsf{0},\mathsf{1},\mathsf{g}(x_1,y_1))\theta_1 = \mathsf{f}(\mathsf{0},\mathsf{1},\mathsf{g}(\mathsf{0},\mathsf{1}))$ *loops w.r.t.* $\mathcal{R}$. *Hence, loopingness of* $\mathcal{R}$ *can be proved using forward unfoldings only.*
- *As the rule* $\mathsf{f}(\mathsf{g}(\mathsf{0},y_1),\mathsf{g}(x_2,\mathsf{1}),x) \to \mathsf{f}(x,x,x)$ *is in* $(B_{\mathcal{R}} \uparrow 2)(\mathcal{R})$ *(see Example 3.8), it belongs to* $unf(\mathcal{R})$. *As* $\mathsf{f}(\mathsf{g}(\mathsf{0},y_1),\mathsf{g}(x_2,\mathsf{1}),x)\theta_1\theta_2 = \mathsf{f}(x,x,x)\theta_1$ *for* $\theta_1 = \{y_1/\mathsf{1}, x_2/\mathsf{0}, x/\mathsf{g}(\mathsf{0},\mathsf{1})\}$ *and* $\theta_2 = \varnothing$, *we can conclude that the term* $\mathsf{f}(\mathsf{g}(\mathsf{0},y_1),\mathsf{g}(x_2,\mathsf{1}),x)\theta_1 = \mathsf{f}(\mathsf{g}(\mathsf{0},\mathsf{1}),\mathsf{g}(\mathsf{0},\mathsf{1}),\mathsf{g}(\mathsf{0},\mathsf{1}))$ *loops w.r.t.* $\mathcal{R}$. *Hence, loopingness of* $\mathcal{R}$ *can also be proved using backward unfoldings only.*

Example 4.3 (Communicated to the author by René Thiemann)

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

The rule $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x) \to \mathsf{f}(x,x,x)$ *is an element of* $(B_{\mathcal{R}} \uparrow 2)(\mathcal{R})$, *so it belongs to* $unf(\mathcal{R})$. *As* $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x)\theta_1\theta_2 = \mathsf{f}(x,x,x)\theta_1$ *for* $\theta_1 = \{x/\mathsf{s}(\mathsf{h})\}$ *and* $\theta_2 = \varnothing$, *we conclude that* $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x)\theta_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}))$ *loops w.r.t.* $\mathcal{R}$. *So, loopingness of* $\mathcal{R}$ *can be proved using backward unfoldings only. On the other hand, the rules* $\mathsf{h} \to \mathsf{0}$ *and* $\mathsf{h} \to \mathsf{1}$ *cannot be unfolded forwards. The rule* $\mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{1}),x) \to \mathsf{f}(x,x,x)$ *can be unfolded forwards, but no resulting rule satisfies the semi-unification criterion of Theorem 4.1. So, loopingness of* $\mathcal{R}$ *cannot be proved using Theorem 4.1 with forward unfoldings only.*

Example 4.4 *Consider the reversed version of the TRS in Example 4.3:*

$$\mathcal{R}^{-1} = \left\{\mathsf{f}(x,x,x) \to \mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{1}),x),\ \mathsf{0} \to \mathsf{h},\ \mathsf{1} \to \mathsf{h}\right\}.$$

The rule $\mathsf{f}(x,x,x) \to \mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x)$ *is an element of* $(F_{\mathcal{R}^{-1}} \uparrow 2)(\mathcal{R}^{-1})$, *so it belongs to* $unf(\mathcal{R}^{-1})$ *and we get that* $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}))$ *loops w.r.t.* $\mathcal{R}^{-1}$. *Hence, loopingness of* $\mathcal{R}^{-1}$ *can be proved using forward unfoldings only. On the other hand, loopingness of* $\mathcal{R}^{-1}$ *cannot be proved using backward unfoldings only.*

Example 4.5 *Consider*

$$\mathcal{R} = \left\{\mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{1}),x,x) \to \mathsf{f}(x,x,\mathsf{s}(\mathsf{2}),\mathsf{s}(\mathsf{3})),\ \mathsf{h} \to \mathsf{0},\ \mathsf{h} \to \mathsf{1},\ \mathsf{2} \to \mathsf{h},\ \mathsf{3} \to \mathsf{h}\right\}.$$

The rule $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x,x) \to \mathsf{f}(x,x,\mathsf{s}(\mathsf{2}),\mathsf{s}(\mathsf{3}))$ *is an element of* $(B_{\mathcal{R}} \uparrow 2)(\mathcal{R})$, *so* $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x,x) \to \mathsf{f}(x,x,\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}))$ *belongs to* $(F_{\mathcal{R}} \uparrow 2)((B_{\mathcal{R}} \uparrow 2)(\mathcal{R}))$. *Hence, this rule is in* $unf(\mathcal{R})$. *As* $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x,x)\theta_1\theta_2 = \mathsf{f}(x,x,\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}))\theta_1$ *for* $\theta_1 = \{x/\mathsf{s}(\mathsf{h})\}$ *and* $\theta_2 = \varnothing$, *we can conclude that the term* $\mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),x,x)\theta_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}),\mathsf{s}(\mathsf{h}))$ *loops w.r.t.* $\mathcal{R}$. *Hence, loopingness of* $\mathcal{R}$ *can be proved using forward and backward unfoldings together. On the other hand, loopingness of* $\mathcal{R}$ *cannot be proved using forward unfoldings only or backward unfoldings only.*

The next example illustrates the use of semi-unification (in the preceding examples, unification is sufficient as θ_2 is always empty).

Example 4.6 ([24]) *Consider*

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

The rule $\mathsf{f}(\mathsf{s}(x_1),y_1,z_1) \to \mathsf{f}(z_1,\mathsf{s}(y_1),z_1)$ *is an element of* $F_{\mathcal{R}}(\mathcal{R})$*, so it belongs to* $unf(\mathcal{R})$*. As* $\mathsf{f}(\mathsf{s}(x_1),y_1,z_1)\theta_1\theta_2 = \mathsf{f}(z_1,\mathsf{s}(y_1),z_1)\theta_1$ *for* $\theta_1 = \{z_1/\mathsf{s}(x_1)\}$ *and* $\theta_2 = \{y_1/\mathsf{s}(y_1)\}$*, we conclude that* $\mathsf{f}(\mathsf{s}(x_1),y_1,z_1)\theta_1 = \mathsf{f}(\mathsf{s}(x_1),y_1,\mathsf{s}(x_1))$ *loops w.r.t.* $\mathcal{R}$*.*

5 Eliminating useless rules

The analysis described in the preceding sections leads to an explosion of the number of unfolded rules. In order to prove that the TRS of Example 3.2 admits a loop, one has to compute $(U_{\mathcal{R}} \uparrow 2)(\mathcal{R})$; the set $\cup_{0\leq n\leq 2}(U_{\mathcal{R}} \uparrow n)(\mathcal{R})$ consists of 450 rules and can be easily generated by any modern personal computer. However, 450 is quite a big number for such a small TRS. In order to prove that the TRS of Example 4.5 admits a loop, one has to compute $(U_{\mathcal{R}} \uparrow 4)(\mathcal{R})$; the set $\cup_{0\leq n\leq 4}(U_{\mathcal{R}} \uparrow n)(\mathcal{R})$ consists of 204 867 rules, which took the computer of the author (a 2.33GHz Intel Core 2 Duo) 7 minutes to compute!

A solution to reduce this explosion consists in designing a mechanism that eliminates the unfolded rules that are *useless* for proving loopingness. On the basis of Theorem 4.1, we say that a rule is useless for a TRS $\mathcal{R}$ when it cannot be unfolded with $\mathcal{R}$ to a rule $l \to r$ where l semi-unifies with a subterm of r.

Example 5.1 *Consider* $\mathcal{R}$ *in Example 4.3 again:*

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

$U_{\mathcal{R}}(\mathcal{R})$ *contains the rule*

$$R = \mathsf{f}(\mathsf{s}(0),\mathsf{s}(1),\mathsf{h}) \to \mathsf{f}(0,\mathsf{h},\mathsf{h})$$

obtained from unfolding $\mathsf{f}(\mathsf{s}(0),\mathsf{s}(1),x) \to \mathsf{f}(x,x,x)$ *forwards using* $\mathsf{h} \to 0$*. The unfoldings of* R *w.r.t.* $\mathcal{R}$ *have the form*

$$\mathsf{f}(\mathsf{s}(\cdots),\mathsf{s}(\cdots),\mathsf{h}) \to \mathsf{f}(0,t_1,t_2)$$

where $t_1, t_2 \in \{\mathsf{h},0,1\}$*. Hence, the left-hand side of every unfolding of* R *does not semi-unify with a subterm of the corresponding right-hand side. Therefore,* R *is useless and can be safely eliminated.*

An idea to detect useless rules is to concentrate first on the full right-hand side (not on its inner subterms); if the full right-hand side is useless, then the rule is replaced with a set of rules obtained from the left-hand side and inner subterms of the right-hand side. The intuition is to eliminate as many subterms as possible from the right-hand side.

Example 5.2 *$U_{\mathcal{R}}(\mathcal{R})$ in Example 5.1 also contains the rule*

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

obtained from unfolding $\mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{1}),x) \to \mathsf{f}(x,x,x)$ forwards using itself. Let l be the left-hand side of this unfolded rule and r be its right-hand side. The unfoldings of $l \to r$ w.r.t. $\mathcal{R}$ have the form

$$\mathsf{f}(\mathsf{s}(\cdots),\mathsf{s}(\cdots),\mathsf{f}(\cdots)) \to \mathsf{f}(\mathsf{f}(\cdots),\mathsf{f}(\cdots),\mathsf{f}(\cdots)) \ .$$

The left-hand side of every unfolding does not semi-unify with the corresponding right-hand side. Of course, this is not sufficient to completely *eliminate $l \to r$ because, by Theorem 4.1, we have to consider every subterm of the right-hand sides. However, an idea is to replace $l \to r$ with the rules $l \to r|_1$, $l \to r|_2$ and $l \to r|_3$ i.e. $l \to \mathsf{f}(x_1,x_1,x_1)$ and $l \to \mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{1}),x_1)$ and to test the uselessness of these new rules with a similar process.*

So, we say that a rule is *root-useless* for $\mathcal{R}$ when it cannot be unfolded with $\mathcal{R}$ to a rule $l \to r$ such that l semi-unifies with r (*root* in *root-useless* comes from the fact that we concentrate on the full right-hand sides, *i.e.* the subterms at root position). Notice that every useless rule is also root-useless but not vice versa; hence, using root-uselessness as an approximation of uselessness, one may remove some rules which are actually useful for proving non-termination.

Example 5.3 *Let $\mathcal{R} = \left\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1}),\ \mathsf{g}(\mathsf{1}) \to \mathsf{g}(\mathsf{f}(\mathsf{0}))\right\}$. The unfoldings of $R = \mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1}) \in (U_{\mathcal{R}} \uparrow 0)(\mathcal{R})$ are the elements of*

$$\mathcal{U} = \left\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1}),\ \mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{f}(\mathsf{0})),\ \mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{g}(\mathsf{1})),\ \mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{g}(\mathsf{f}(\mathsf{0}))),\ \ldots\right\} .$$

As the left-hand side of every rule in $\mathcal{U}$ does not semi-unify with the corresponding right-hand side, R is root-useless and the idea of Example 5.2 consists in replacing it in $(U_{\mathcal{R}} \uparrow 0)(\mathcal{R})$ with the rule $\mathsf{f}(\mathsf{0}) \to \mathsf{1}$, the unfoldings of which do not allow to conclude non-termination. On the other hand, as the left-hand side of $\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{f}(\mathsf{0})) \in \mathcal{U}$ semi-unifies with the subterm $\mathsf{f}(\mathsf{0})$ of its right-hand side, R is not useless for $\mathcal{R}$ and, by Theorem 4.1, $\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{f}(\mathsf{0}))$ establishes non-termination of $\mathcal{R}$.

The eliminating mechanism that we present from now is an extension of that of [36] *i.e.* it provides a better approximation of the useless rules (see Section 7 and Section 8.2). We also use a formalisation that is different from that of [36].

The technique is based on the detection of root-useless rules; although it may remove some useful rules, it gives good results in practice (see Section 7).

5.1 Root-useless rules

In Section 6, an underapproximation of the set of root-useless rules is computed using an abstract fixpoint. The corresponding concrete setting is described below.

We denote by $\mathbb{R}(\mathcal{F},\mathcal{V})$ the powerset of $\mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V})$. Then $\mathbb{R}(\mathcal{F},\mathcal{V})$ is partially ordered by the relation $\leq$ defined as:

$$\forall X, X' \in \mathbb{R}(\mathcal{F},\mathcal{V}),\ X \leq X' \text{ iff } \mathit{instances}(X) \subseteq \mathit{instances}(X') \ .$$

The poset $\langle \mathbb{R}(\mathcal{F},\mathcal{V}), \leq \rangle$ can be extended to a complete lattice. It suffices to consider the standard set union as least upper bound $\sqcup$ and the intersection of the sets of instances as greatest lower bound $\sqcap$.

Definition 5.4 ($\sqcup$,$\sqcap$) *For any $I \subseteq \mathbb{N}$ and $\{X_i\}_{i\in I} \subseteq \mathbb{R}(\mathcal{F},\mathcal{V})$, we let*

$$\bigsqcup_{i\in I} X_i = \bigcup_{i\in I} X_i \quad \textit{and} \quad \bigsqcap_{i\in I} X_i = \bigcap_{i\in I} \mathit{instances}(X_i) \ .$$

For instance, $\{\mathsf{f}(x,1) \to \mathsf{f}(x,1)\} \sqcap \{\mathsf{f}(0,x) \to \mathsf{f}(0,x)\} = \{\mathsf{f}(0,1) \to \mathsf{f}(0,1)\}$.

Proposition 5.5 *$\langle \mathbb{R}(\mathcal{F},\mathcal{V}), \leq, \sqcup, \sqcap, \varnothing, \mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V}) \rangle$ is a complete lattice.*

Notice that the least (*w.r.t.* $\leq$) fixpoint of $U_{\mathcal{R}}$ is the empty set. So, in order to capture the unfoldings of a rule with a least fixpoint, we introduce the following operator.

Definition 5.6 (Unfoldings of a rule) *Let R be a rewrite rule. For any set $X \in \mathbb{R}(\mathcal{F},\mathcal{V})$, we let*

$$U_{\mathcal{R},R}(X) = U_{\mathcal{R}}(X) \cup \{R\} \ .$$

The next proposition implies that for any rewrite rule R, the least fixpoint (*lfp*) of $U_{\mathcal{R},R}$ always exists and $\mathit{lfp}(U_{\mathcal{R},R}) = \sqcup_{n\in\mathbb{N}}(U_{\mathcal{R},R} \uparrow n)(\varnothing)$.

Proposition 5.7 *For any rewrite rule R, the operator $U_{\mathcal{R},R}$ is continuous on $\langle \mathbb{R}(\mathcal{F},\mathcal{V}), \leq, \sqcup, \sqcap, \varnothing, \mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V}) \rangle$.*

We have:

Lemma 5.8 *For any rewrite rule R, $\mathit{lfp}(U_{\mathcal{R},R}) = \cup_{n\in\mathbb{N}}(U_{\mathcal{R}} \uparrow n)(\{R\})$.*

So, we will consider the following fixpoint definition of root-useless rules.

Definition 5.9 (Root-useless rule) *A rule R is root-useless for $\mathcal{R}$ when no element $l \to r$ of $lfp(U_{\mathcal{R},R})$ is such that l semi-unifies with r.*

5.2 Eliminating unfoldings

Our elimination function transforms a root-useless rule R to a (possibly empty) set of rules that are not root-useless by considering the subterms of the right-hand side of R.

Definition 5.10 (Elimination function) *Let $l \to r$ be a rule. We define*

$$elim_{\mathcal{R}}(l \to r) = \begin{cases} \text{if } l \to r \text{ is not root-useless for } \mathcal{R} \text{ then } \{l \to r\} \\ \text{else if } r = \mathsf{f}(t_1, \ldots, t_m) \text{ then } \bigcup_{i \in [1,m]} elim_{\mathcal{R}}(l \to t_i) \\ \text{else } \varnothing \end{cases}$$

For any set X of rewrite rules, we let $elim_{\mathcal{R}}(X) = \bigcup_{R \in X} elim_{\mathcal{R}}(R)$.

Now we can define a new unfolding operator that eliminates root-useless rules from the result provided by $U_{\mathcal{R}}$.

Definition 5.11 (Eliminating unfoldings) *Let X be a set of rewrite rules. The eliminating unfoldings of X w.r.t. $\mathcal{R}$ are defined as*

$$EU_{\mathcal{R}}(X) = elim_{\mathcal{R}}(U_{\mathcal{R}}(X)) \ .$$

This operator allows us to define the eliminating counterpart of the unfolding semantics.

Definition 5.12 (Eliminating unfolding semantics) *We define the eliminating unfolding semantics of $\mathcal{R}$ as the limit of the unfolding process described in Definition 5.11, starting from $elim_{\mathcal{R}}(\mathcal{R})$:*

$$eunf(\mathcal{R}) = \bigcup_{n \in \mathbb{N}} (EU_{\mathcal{R}} \uparrow n)(elim_{\mathcal{R}}(\mathcal{R})) \ .$$

The relevance of a loop detection analysis based on these notions is clarified by the following correctness result. In contrast to Theorem 4.1, we do not consider the subterms of the right-hand sides as $eunf(\mathcal{R})$ splits the right-hand sides by means of the elimination function $elim_{\mathcal{R}}(\mathcal{R})$:

Theorem 5.13 (Correctness) *If for $l \to r \in eunf(\mathcal{R})$ there are some substitutions θ_1 and θ_2 such that $l\theta_1\theta_2 = r\theta_1$ then $l\theta_1$ loops w.r.t. $\mathcal{R}$.*

6 Approximating root-useless rules

In this section, we approximate root-useless rules by means of an abstraction that consists in pruning the left-hand or right-hand side of the rules used to unfold.

6.1 Our abstract domain

From now on, we consider an infinite countable set $\hat{\mathcal{V}}$ of new variables, disjoint from $\mathcal{V}$, that are used in the pruning process. Terms are cut by replacing each subterm that is ignored with a *new* variable from $\hat{\mathcal{V}}$. Therefore, we consider the subset of $\mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}) \times \mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, every element of which does not contain multiple occurrences of a variable from $\hat{\mathcal{V}}$. Let $\mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ denote this subset. Each variable of $\hat{\mathcal{V}}$ occurring in an element of $\mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ corresponds to any term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. This is formalised by the following concretisation function.

Definition 6.1 (Concretisation) *The concretisation of $R \in \mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ is denoted by $\gamma(R)$ and is defined as:*

$$\gamma(R) = \left\{ R\hat{\sigma} \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V}) \mid \hat{\sigma} \text{ a substitution, } Dom(\hat{\sigma}) \subseteq \hat{\mathcal{V}} \right\} .$$

The concretisation of $X \subseteq \mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ is $\gamma(X) = \bigcup_{R \in X} \gamma(R)$.

We denote by $\mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ the powerset of $\mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ where we identify the elements having the same (modulo $\leq$) concretisation. Then $\mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ is partially ordered by the relation $\preceq$ defined as:

$$\forall X, X' \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}),\ X \preceq X' \text{ iff } \gamma(X) \leq \gamma(X') .$$

The poset $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq \rangle$ can be extended to a complete lattice. It suffices to consider the standard set union as least upper bound $\vee$ and the intersection of the instances of the concretisations as greatest lower bound $\wedge$. Notice that by *instances* we still mean *instances over* $\mathcal{F} \cup \mathcal{V}$ (not over $\mathcal{F} \cup \mathcal{V} \cup \hat{\mathcal{V}}$), as in the concrete setting.

Definition 6.2 ($\vee$,$\wedge$) *For any $I \subseteq \mathbb{N}$ and $\{X_i\}_{i \in I} \subseteq \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, we let*

$$\bigvee_{i \in I} X_i = \bigcup_{i \in I} X_i \quad \text{and} \quad \bigwedge_{i \in I} X_i = \bigcap_{i \in I} instances(\gamma(X_i)) .$$

Proposition 6.3 *$\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq, \vee, \wedge, \varnothing, \mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}) \rangle$ is a complete lattice.*

We have:

Proposition 6.4 *For any $I \subseteq \mathbb{N}$ and $\{X_i\}_{i \in I} \subseteq \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$,*

$$\bigsqcap_{i \in I} \gamma(X_i) = \gamma\Big(\bigwedge_{i \in I} X_i \Big) \, .$$

As a consequence of Proposition 6.4, γ is co-additive and hence it is the concretisation map of a Galois connection (Section 2). Moreover, as we identify all elements of $\mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ having the same concretisation, γ is the concretisation map of a Galois insertion.

Definition 6.5 *The complete lattice $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq, \vee, \wedge, \varnothing, \mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}) \rangle$ is our abstract domain. Its elements stand silently for their equivalence class.*

6.2 Abstract unfoldings of a rule

The pruning function that we consider only keeps the root of the terms.

Definition 6.6 (Pruning) *For any $x \in \mathcal{V}$ and any $\mathsf{f}(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$,*

$$\mathit{prune}(x) = \hat{x}_1 \quad \text{and} \quad \mathit{prune}(\mathsf{f}(t_1, \ldots, t_m)) = \mathsf{f}(\hat{x}_1, \ldots, \hat{x}_m)$$

where $\hat{x}_1, \ldots, \hat{x}_m$ are distinct variables from $\hat{\mathcal{V}}$ not previously met.

The definitions and results below are parametric in a TRS $\mathcal{R}$ and a rewrite rule R, both over $\mathcal{F} \cup \mathcal{V}$.

The abstract counterpart of the unfolding operator $U_{\mathcal{R},R}$ (Section 5.1) is defined hereafter. The intuition consists in computing a finite and easily obtainable set that approximates the structure of the unfoldings of R.

Example 6.7 *Consider the following TRS:*

$$\mathcal{R} = \Big\{ \mathsf{f}(\mathsf{h}(x), \mathsf{s}(\mathsf{0})) \to \mathsf{f}(\mathsf{h}(x), \mathsf{g}(x)),\ \mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{g}(x)),\ \mathsf{g}(\mathsf{0}) \to \mathsf{0} \Big\}$$

and $R = \mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{g}(x)) \in (U_{\mathcal{R}} \uparrow 0)(\mathcal{R})$. The unfoldings of R all have the form $\mathsf{g}(\cdots) \to \mathsf{s}(\cdots)$. The left-hand side of these unfoldings does not semi-unify with the corresponding right-hand side, hence R is root-useless. So, R is removed from $(U_{\mathcal{R}} \uparrow 0)(\mathcal{R})$ and function $\mathit{elim}_{\mathcal{R}}$ replaces it with $\mathsf{g}(\mathsf{s}(x)) \to \mathsf{g}(x)$, whose root-uselessness is then checked.

Now consider $R' = \mathsf{f}(\mathsf{h}(x), \mathsf{s}(\mathsf{0})) \to \mathsf{f}(\mathsf{h}(x), \mathsf{g}(x)) \in (U_{\mathcal{R}} \uparrow 0)(\mathcal{R})$. The left-hand side of R' does not semi-unify with the right-hand side. But the right-hand

side can be unfolded to a term t of the form $\mathsf{f}(\mathsf{h}(x), \mathsf{s}(\cdots))$, as the second rule of $\mathcal{R}$ has the form $\mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(\cdots)$. The left-hand side of R' may semi-unify with t. Hence, we consider that R is not root-useless for $\mathcal{R}$.

The idea is to approximate the structure of the terms obtained when rewriting the subterms of R (in the second part of Example 6.7, $\mathsf{s}(\cdots)$ is an approximation of some rewritings of subterm $\mathsf{g}(x)$). This is done by pruning the left-hand or right-hand side of the rules of $\mathcal{R}$ during the unfolding process. The pruning function we consider (Definition 6.6) only keeps the roots of the terms; another alternative is to use a *depth*(k) cut, where k is a parameter of the analysis, but our technique still can detect many root-useless rules (see Section 7). In the definition below, we again consider variable positions but not for the elements of $\hat{\mathcal{V}}$. Moreover, we do not apply the most general unifiers (in contrast to Definition 3.4 and Definition 3.7) as we want to get a finite least fixpoint (see Theorem 6.15 below).

Definition 6.8 (Abstract unfoldings of a rule) *Let $X \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$. Then,*

$$\hat{U}_{\mathcal{R},R}(X) = \hat{F}_{\mathcal{R}}(X) \cup \hat{B}_{\mathcal{R}}(X) \cup X \cup \{R\}$$

where

$$\hat{F}_{\mathcal{R}}(X) = \left\{ \hat{l} \to \hat{r}[p \leftarrow \mathit{prune}(r')] \;\middle|\; \begin{array}{l} \hat{R} = \hat{l} \to \hat{r} \in X,\ p \in \mathit{Pos}(\hat{r}),\ \hat{r}|_p \notin \hat{\mathcal{V}} \\ l' \to r' \in \mathcal{R},\ \hat{r}|_p \text{ unifies with } l' \end{array} \right\}$$

and

$$\hat{B}_{\mathcal{R}}(X) = \left\{ \hat{l}[p \leftarrow \mathit{prune}(l')] \to \hat{r} \;\middle|\; \begin{array}{l} \hat{R} = \hat{l} \to \hat{r} \in X,\ p \in \mathit{Pos}(\hat{l}),\ \hat{l}|_p \notin \hat{\mathcal{V}} \\ l' \to r' \in \mathcal{R},\ \hat{l}|_p \text{ unifies with } r' \end{array} \right\} .$$

The next proposition implies that the least fixpoint of $\hat{U}_{\mathcal{R},R}$ is $\mathit{lfp}(\hat{U}_{\mathcal{R},R}) = \bigvee_{n \in \mathbb{N}} (\hat{U}_{\mathcal{R},R} \uparrow n)(\varnothing)$.

Proposition 6.9 *The abstract unfolding operator $\hat{U}_{\mathcal{R},R}$ is continuous on the complete lattice $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq, \vee, \wedge, \varnothing, \mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}) \rangle$.*

Notice that unlike $U_{\mathcal{R},R}(X)$ (Definition 5.6), the set $\hat{U}_{\mathcal{R},R}(X)$ includes X. This is an essential point to ensure that the following correctness result holds.

Proposition 6.10 (Correctness) *For any $X \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ we have*

$$U_{\mathcal{R},R}(\gamma(X)) \leq \gamma(\hat{U}_{\mathcal{R},R}(X)) .$$

Example 6.11 *Let $\mathcal{R} = \{\mathsf{f}(\mathsf{0}, x) \to \mathsf{h}(x), \mathsf{1} \to \mathsf{2}\}$, $X = \{\mathsf{f}(\mathsf{0}, x) \to \mathsf{h}(\mathsf{h}(\hat{x}))\}$ and $R = \mathsf{f}(\mathsf{0}, x) \to \mathsf{h}(x)$. Then, $\mathsf{f}(\mathsf{0}, x) \to \mathsf{h}(\mathsf{h}(\mathsf{1})) \in \gamma(X)$ (it is obtained by in-*

stantiating the rule of X with $\{\hat{x}/1\}$). Unfolding this rule forwards, we get $\mathsf{f}(0,x) \to \mathsf{h}(\mathsf{h}(2))$. This rule is in $\gamma(X)$ but not in $\gamma(\hat{F}_{\mathcal{R}}(X) \cup \hat{B}_{\mathcal{R}}(X) \cup \{R\})$. The point is that we concretise X and then unfold a rule at a position that is not a permitted position of the corresponding abstract rule. Such an unfolding cannot be captured by $\hat{F}_{\mathcal{R}}$ and $\hat{B}_{\mathcal{R}}$.

From Proposition 6.10 and the framework of abstract interpretation (Section 2), we directly get the following correctness result.

Theorem 6.12 (Correctness) *$lfp(U_{\mathcal{R},R}) \leq \gamma(lfp(\hat{U}_{\mathcal{R},R}))$.*

Therefore, $lfp(\hat{U}_{\mathcal{R},R})$ provides a presentation of a superset of $lfp(U_{\mathcal{R},R})$. This presentation can be used to approximate root-useless rules as we have:

Theorem 6.13 (Root-Useless rule) *Suppose that for all $l \to r \in lfp(\hat{U}_{\mathcal{R},R})$, l does not semi-unify with r. Then, R is root-useless for $\mathcal{R}$.*

The operator $\hat{U}_{\mathcal{R},R}$ can be used to eliminate root-useless rules in a fully automatic loop detection analysis as $lfp(\hat{U}_{\mathcal{R},R})$ is finite. We prove this below. As we did in the case of $\mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}) \times \mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, we suppose that the elements of $\mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ do not contain multiple occurrences of a variable from $\hat{\mathcal{V}}$ and we identify the subsets of $\mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ having the same concretisation (γ is straightforwardly extended to such subsets). Since the definition of $\hat{F}_{\mathcal{R}}(X)$ neither changes nor considers the left-hand sides of rules in X, we can mimic $\hat{F}_{\mathcal{R}}(X)$ by just applying it on the right-hand sides (see $D_{\mathcal{R}}$ below). The same is true for $\hat{B}_{\mathcal{R}}$ and $A_{\mathcal{R}}$ below.

Definition 6.14 (Descendants, ascendants) *For any $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$,*

$$D_{\mathcal{R}}(T) = \left\{ t[p \leftarrow prune(r')] \;\middle|\; \begin{array}{l} t \in T,\ p \in Pos(t),\ t|_p \notin \hat{\mathcal{V}} \\ l' \to r' \in \mathcal{R},\ t|_p \text{ unifies with } l' \end{array} \right\}$$

and

$$A_{\mathcal{R}}(T) = \left\{ t[p \leftarrow prune(l')] \;\middle|\; \begin{array}{l} t \in T,\ p \in Pos(t),\ t|_p \notin \hat{\mathcal{V}} \\ l' \to r' \in \mathcal{R},\ t|_p \text{ unifies with } r' \end{array} \right\}.$$

The descendants and the ascendants of $t \in \mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ w.r.t. $\mathcal{R}$ are respectively $\Delta_{\mathcal{R}}(t) = \cup_{n \in \mathbb{N}}(D_{\mathcal{R}} \uparrow n)(\{t\})$ and $\nabla_{\mathcal{R}}(t) = \cup_{n \in \mathbb{N}}(A_{\mathcal{R}} \uparrow n)(\{t\})$.

For any $t \in \mathcal{T}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, as there are finitely many subterms in t and as $\mathcal{R}$ is finite, then $\Delta_{\mathcal{R}}(t)$ and $\nabla_{\mathcal{R}}(t)$ are finite and there exist $n_d, n_a \in \mathbb{N}$ such that $\Delta_{\mathcal{R}}(t) = \cup_{n \leq n_d}(D_{\mathcal{R}} \uparrow n)(\{t\})$ and $\nabla_{\mathcal{R}}(t) = \cup_{n \leq n_a}(A_{\mathcal{R}} \uparrow n)(\{t\})$. The next theorem establishes that $lfp(\hat{U}_{\mathcal{R},R})$ is finite and can be obtained by finitely

iterating $A_{\mathcal{R}}$ starting from the left-hand side of R and $D_{\mathcal{R}}$ starting from the right-hand side of R.

Theorem 6.15 (Finiteness) *Let l and r be the left-hand and right-hand side of R, respectively. Then, $lfp(\hat{U}_{\mathcal{R},R}) = \nabla_{\mathcal{R}}(l) \times \Delta_{\mathcal{R}}(r)$.*

Example 6.16 (Example 6.7 continued)

$$lfp(\hat{U}_{\mathcal{R},R}) = \nabla_{\mathcal{R}}(\mathsf{g}(\mathsf{s}(x))) \times \Delta_{\mathcal{R}}(\mathsf{s}(\mathsf{g}(x)))$$

where

$$\nabla_{\mathcal{R}}(\mathsf{g}(\mathsf{s}(x))) = \left\{\mathsf{g}(\mathsf{s}(x)), \mathsf{g}(\mathsf{g}(\hat{x}_1)), \mathsf{g}(\mathsf{s}(\mathsf{f}(\hat{x}_2, \hat{x}_3))), \mathsf{g}(\mathsf{s}(\mathsf{g}(\hat{x}_4)))\right\}$$
$$\Delta_{\mathcal{R}}(\mathsf{s}(\mathsf{g}(x))) = \left\{\mathsf{s}(\mathsf{g}(x)), \mathsf{s}(\mathsf{s}(\hat{x}_5)), \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{g}(\mathsf{f}(\hat{x}_6, \hat{x}_7))), \mathsf{s}(\mathsf{g}(\mathsf{s}(\hat{x}_8))), \mathsf{s}(\mathsf{g}(\mathsf{0}))\right\}.$$

Every element $l \to r \in lfp(\hat{U}_{\mathcal{R},R})$ is such that l does not semi-unify with r. So, R is root-useless for $\mathcal{R}$.

The set of descendants of a term t provides an over-approximation of the set of forward narrowings of t. Similarly, the set of ascendants of t provides an over-approximation of the set of backward narrowings of t. This resembles the dependency-pair analysis described in [24] where loop detection is performed using forward or backward narrowing, considering variable positions in some particular cases. One difference with our approach is that we compute the descendants and the ascendants in order to approximate root-useless rules (not to detect loops directly) and that, in Theorem 6.15, we mix the ascendants of l with the descendants of r. In contrast, [24] either considers the backward narrowings of l or the forward narrowings of r.

7 Experimental evaluation

We have implemented the techniques of this paper in our analyser NTI (Non-Termination Inference). The current version NTI'08 is available from

`http://personnel.univ-reunion.fr/epayet`

Elimination of root-useless rules really decreases the number of generated rules but the analysis is still expensive as our unfolding technique processes both forwards and backwards and considers variable positions. Notice that mixing forward and backward unfoldings is not always necessary for detecting loops (Section 4 provides some examples). Hence, by default our tool runs three analyses in parallel: one with forward unfoldings only, one with backward unfoldings only and one with forward and backward unfoldings together. The

process that terminates first "kills" the others. This default setting can be easily overridden by the user who can select one unfolding direction only.

7.1 Term rewriting

We have run [1] NTI'08 on the 129 non-terminating TRSs of the sub-category *standard rewriting* of the Termination Competition'07 [33]. We fixed a 2 minute time limit for each TRS. We get the results in Figure 1 where we also report the performance of the three best [2] tools of the competition that we have run in the same conditions as NTI'08. For each tool, we indicate

AProVE'07	NTI'08 on	NTI'08 off	NTI'07	TTT2'07
128/129	128/129	122/129	117/129	69/129
1 `maybe`	1 `time out`	7 `time out`	7 `time out` 5 `don't know`	60 `maybe`
	5min 33s	16min 08s	14min 52s	
	161 869 rules	1 163 354 rules	205 632 rules	

Fig. 1. Results regarding the TRSs of the Termination Competition'07

the number of successful non-termination proofs and the answers when failure occurs. If relevant, we also report the total time elapsed and the number of unfolded rules. AProVE'07, NTI'07 and TTT2'07 respectively refer to the version of AProVE [25,22,6], NTI and TTT2 [28,44] used in the Termination Competition'07. NTI'07 [3] implements the techniques of [36], *i.e.* forward unfoldings and non-variable positions only. In the column NTI'08 on (resp. NTI'08 off), we present the results of our current analyser with elimination of root-useless rules on (resp. off). We do not indicate the timing of AProVE'07 as this tool is specialised to solve termination problems *i.e.* it first performs a termination analysis and starts the non-termination proof only when this analysis fails. TTT2'07 performs its non-termination test before any termination analysis but, as this test is very simple, nearly 100% of the time is spent for a termination proof attempt; hence, we do not indicate the total time of TTT2'07 as well. As NTI'08 runs three processes in parallel, for each TRS we only count the rules generated by the successful process that terminates first; if no process is successful within the time limit, then we count the rules generated by

[1] using a 2.33GHz Intel Core 2 Duo with 2Gb DDR2 SDRAM

[2] in the category *TRS*, sub-category *standard rewriting*, regarding non-termination only

[3] Actually, two versions of NTI'07, denoted by NTI and NTI2 in [33], are used in the Termination Competition'07. NTI2 is a corrected version of NTI, the parser of which does not work properly. In this paper, NTI'07 refers to NTI2.

the process computing forward and backward unfoldings together.

We observe in Figure 1 that compared to NTI'08 off, NTI'08 on globally achieves a 85% reduction (approximately) of the number of unfolded rules and runs 3 times faster. Moreover, the detection of root-useless rules does not cause any major slowdown; the NTI'08 on average time regarding the successful proofs is 1.67 second per TRS whereas that of NTI'08 off is 0.92 second per TRS. Although it computes forward and backward unfoldings together and considers variable positions, NTI'08 on generates fewer rules and globally runs faster than NTI'07. This is because NTI'07 reaches the time limit for 7 TRSs; non-termination of 6 of these TRSs is solved by NTI'08 on (in less than 1 second for three TRSs, in 6 seconds for one TRS and in 70 seconds for two TRSs). The sets of successfully solved TRSs corresponding to each tool are related as follows:

$$\mathsf{TTT2'07} \subset [\mathsf{NTI'07}, \mathsf{NTI'08\ off}] \subset [\mathsf{NTI'08\ on}, \mathsf{AProVE'07}] \ .$$

AProVE'07 successfully solves `TRCSR/Ex1_GM99_iGM` while NTI'08 on does not and NTI'08 on successfully solves `nontermin/AG01/#4.19` while AProVE'07 does not. Apart from these TRSs, AProVE'07 and NTI'08 on handle the same systems. The non-terminating term computed by AProVE'07 in the case of `TRCSR/Ex1_GM99_iGM` can be inferred from iteration 7 of the unfolding operator $U_{\mathcal{R}}$. We let NTI'08 on run for 3 hours on `TRCSR/Ex1_GM99_iGM`; it computed iteration 0 (13 rules), iteration 1 (277 rules), iteration 2 (4 393 rules), iteration 3 (49 387 rules), iteration 4 (422 275 rules) and "only" the first 158 164 rules of iteration 5. We then run NTI'08 on with unfolding of variable positions disabled; we got the complete non-termination proof in less than 6 seconds with 6 298 unfolded rules generated!

NTI'07 answers `don't know` for 5 TRSs. `don't know` happens when an iteration of the eliminating unfolding operator is empty. Emptiness can be caused by the eliminating technique of NTI'07 that removes some rules which are actually useful; it also can be due to the unfolding operator which only processes forwards without considering variable positions (hence some useful rules may not be computed). For each failure of NTI'07, the table in Figure 2 indicates whether NTI'08 on can solve the problem (2 minute time limit). When NTI'08 on is successful, we report the unfolding direction and the use of variable positions (no indication means that considering variable positions is not necessary). NTI'08 on is not able to solve `cime4` with forward unfoldings only, `#4.13` without considering variable positions and `OvCons`* with forward unfoldings only or backward unfoldings only. This possibly explains why these problems cannot be solved by NTI'07 as it does not unfold backwards nor consider variable positions. NTI'07 fails on `jwno1-6`, `#4.19`, `2.05`, `Hamming` and `Ex4_DLMMU04_FR` because its eliminating mechanism removes some useful rules (which are not removed by the mechanism of NTI'08 on).

problem	NTI'07	NTI'08 on
secret05/cime4	don't know	backwards
Waldmann/jwno1	don't know	forwards
Waldmann/jwno4	don't know	forwards
Waldmann/jwno6	don't know	forwards
nontermin/AG01/#4.13	don't know	forwards variable positions
nontermin/AG01/#4.19	time out	forwards
SK90/2.05	time out	forwards
higher-order/Bird/Hamming	time out	forwards
TRCSR/OvConsOS_nosorts-noand_FR	time out	forwards+backwards
TRCSR/OvConsOS_nosorts_noand_GM	time out	forwards+backwards
TRCSR/Ex4_DLMMU04_FR	time out	forwards
TRCSR/Ex1_GM99_iGM	time out	time out

Fig. 2. Failures of NTI'07

iterations	0	1	2	3	4	5	6	8
TRSs	71	13	21	7	8	5	2	1
time (s)	0.02	0.04	0.04	0.96	9.70	2.31	35.74	43.23
rules	1	9	28	2 595	4 525	1 659	15 207	17 061

Fig. 3. Iterations of the eliminating unfolding operator

In Figure 1, the number of iterations of the eliminating unfolding operator that is computed by NTI'08 on for each successful proof ranges from 0 to 8; the number of TRSs, average time (in seconds) and average number of unfolded rules corresponding to each number of iterations are given in Figure 3. For 71 TRSs, NTI'08 on stops the proof at iteration 0 *i.e.* it directly gets a looping term from the rules of the system under analysis; these TRSs are trivially non-terminating. TTT2'07 was able to prove non-termination of TRSs in this set only as it just checks whether the left-hand side of a rule is contained in its right-hand side; as this simple check can be performed very fastly, the average proof time of TTT2'07 regarding the successful proofs is 0.06 second.

We have also run NTI'08 on the 66 non-terminating string rewriting systems [4] (SRSs) of the Termination Competition'07 [33] (sub-category standard rewriting). We used the same machine as for term rewriting (see footnote 1) and fixed a 2 minute time limit for each SRS. Notice that:

Theorem 7.1 ([21]) *A SRS admits a loop if and only if its forward closure contains a rule of the form $t \to utv$ (where t, u and v are strings).*

The forward closures of a SRS correspond to its forward unfoldings where variable positions are disregarded. Hence, we have run NTI'08 with forward unfoldings only and variable positions disabled. We get the results in Figure 4 where we also report the performance of the three best [5] tools of the competition that we have run in the same conditions as NTI'08. Matchbox'07 refers

Matchbox'07	NTI'08 on	NTI'08 off	NTI'07	AProVE'07
65/66	46/66	22/66	15/66	6/66
1 `maybe`	20 `time out`	44 `time out`	41 `time out` 10 `don't know`	37 `maybe` 23 `time out`
	50min 39s	01h 32min 54s	01h 23min 30s	
	861 597 rules	2 486 121 rules	756 658 rules	

Fig. 4. Results regarding the SRSs of the Termination Competition'07

to the version of Matchbox [45,34] used in the Termination Competition'07 and NTI'08 on (resp. NTI'08 off) corresponds to our current analyser with elimination of root-useless rules on (resp. off), forward unfoldings only and no unfolding of variable subterms. We do not report the timings of Matchbox'07 and AProVE'07 as these tools first perform a termination analysis and start the non-termination proof only when this analysis fails. More precisely, Matchbox'07 first applies a cheap termination method (*e.g.* simplex method for additive weights), then it performs loop detection and finally it applies an expensive termination method (matrices).

We observe in Figure 4 that compared to NTI'08 off, NTI'08 on globally achieves a 65% reduction (approximately) of the number of unfolded rules and is 42 minutes faster. The NTI'08 on average time regarding the successful proofs is 13.89 seconds per SRS whereas that of NTI'08 off is 10.63 seconds

[4] NTI'08 classically converts a SRS into a TRS and proves non-termination of the resulting TRS

[5] in the category *SRS*, sub-category *standard rewriting*, regarding non-termination only

problem	NTI'07	NTI'08 on
HofWald/1	don't know	✓
secret2007/matchbox/num-514	don't know	✓
secret2007/matchbox/num-530	don't know	✓
Waldmann07a/size-11-alpha-3-num-15	don't know	✓
Waldmann07a/size-12-alpha-2-num-11	don't know	time out
Waldmann07a/size-12-alpha-2-num-18	don't know	✓
Waldmann07b/size-12-alpha-3-num-233	don't know	✓
Waldmann07b/size-12-alpha-3-num-475	don't know	time out
Waldmann07b/size-12-alpha-3-num-540	don't know	time out
Waldmann07b/size-12-alpha-3-num-90	don't know	✓

Fig. 5. Failures of NTI'07

per SRS. Non-termination of Waldmann07b/size-12-alpha-3-num-20 is successfully proved by NTI'08 on, NTI'08 off and NTI'07 but not by Matchbox'07. If we disregard this SRS, the sets of successfully solved SRSs corresponding to each tool are related as follows:

$$\mathsf{AProVE'07} \subset \mathsf{NTI'07} \subset [\mathsf{NTI'08\ off}, \mathsf{NTI'08\ on}] \subset \mathsf{Matchbox'07}\ .$$

Every SRS that is successfully handled by NTI'08 off is also successfully handled by NTI'08 on, except Waldmann07b/num-243 where NTI'08 on removes some rules that are actually useful. This SRS is not solved by NTI'07 either. NTI'07 reaches the time limit for 41 SRSs and answers don't know for 10 others. The table in Figure 5 indicates for each don't know of NTI'07 whether NTI'08 on can solve the problem (2 minute time limit). These results may look surprising at first glance as NTI'08 on unfolds SRSs forwards only without considering variable positions, exactly as NTI'07. Hence, both tools should provide the same results. However, NTI'08 on and NTI'07 estimate useless rules differently and NTI'08 on's estimation is better (*i.e.* removes fewer useful rules) than that of NTI'07 (see Section 8.2). NTI'08 on globally generates more rules than NTI'07 (see Figure 4) but is able to conclude non-termination in more cases.

Although Theorem 7.1 holds, as the useless rule estimation and the time limit introduce a loss of precision, one may wonder if using forward and backward unfoldings with variable positions (as in Section 7.1) would provide more positive answers (regarding non-termination) and accelerate the proofs. We also run NTI'08 with elimination of root-useless rules and variable positions enabled, 3 proofs in parallel (forward only, backward only and forward and backward mixed) and a 2 minute time limit. We only got 36 successful proofs

iterations	0	1	3	5	6	7	8	9	10
SRSs	1	2	3	5	2	1	1	5	2
time (s)	0.37	0.06	0.06	0.07	0.78	0.14	0.14	20.49	2.80
rules	1	23	6	37	290	62	104	1 636	1 587

iterations	11	12	13	14	15	17	18	19	20
SRSs	2	1	1	5	1	3	2	1	1
time (s)	1.76	2.51	25.88	6.99	7.97	12.72	46.92	11.12	10.70
rules	1 536	1 378	9 588	2 482	6 205	10 164	12 764	6 106	6 012

iterations	23	24	25	26	27	29
SRSs	1	2	1	1	1	1
time (s)	97.81	11.14	51.25	36.28	92.74	2.20
rules	33 032	6 713	32 141	13 197	14 325	1 153

Fig. 6. Iterations of the eliminating unfolding operator

with a total amount of 698 673 unfolded rules and a global proof time of 1h 07min 51s.

In Figure 4, the number of iterations of the eliminating unfolding operator that is computed by NTI'08 on for each successful proof ranges from 0 to 29; the number of SRSs, average time (in seconds) and average number of unfolded rules corresponding to each number of iterations are given in Figure 6.

8 Related work

8.1 Overlap Closure

[26] introduces the overlap closure of a TRS, an unfolding operation that processes both forwards and backwards. The only differences with Definition 3.11 herein are that the overlap closure does not consider variable positions (hence Definition 3.11 provides a superset of the overlap closure) and in order to compute $unf(\mathcal{R})$, one has to overlap the unfoldings with the rules of $\mathcal{R}$ whereas in order to compute the overlap closure of $\mathcal{R}$ one has to overlap closures with closures. [26] proposes a method for proving *uniform termination* of TRSs. First the authors prove that the rewriting relation of a finite set of rules is uniformly terminating *if and only if* it is both globally finite and acyclic; this result is not related to term rewriting only. Then they provide a sufficient condition to global finiteness that can be syntactically checked. Finally they establish that if a TRS $\mathcal{R}$ is right-linear or left-linear and $\underset{\mathcal{R}}{\rightarrow}$ is globally finite, then $\underset{\mathcal{R}}{\rightarrow}$ is uniformly terminating *if and only if* the overlap closure of $\mathcal{R}$ contains no

rule of the form $t \to t$. The technique we use herein does not consider global finiteness and is able to detect loops only. On the other hand, we unfold variable positions and use semi-unification, which encompasses equality between the left-hand and right-hand side of rules.

In [21] the authors use the forward closure [14], a restricted form of the overlap closure with forward reductions only, to characterize the existence of loops in string rewriting. They prove that a SRS $\mathcal{R}$ admits a loop *if and only if* the forward closure of $\mathcal{R}$ contains a rule of the form $t \to utv$ (where t, u and v are strings). We considered this result in our experiments, see Section 7.2.

8.2 Comparison with [36]

This paper continues the work initiated in [36] where we used the unfolding technique of [4] which only processes forwards. Hence, the analysis presented in [36] is strictly less powerful than that of this paper as it is unable to prove loopingness of TRSs requiring backward unfoldings (*e.g.* Example 4.3 and Example 4.5). Moreover, the unfolding technique of [4] does not consider variable subterms, which led us to introduce the *augmented* version of a TRS in [36]. Roughly speaking, a TRS is augmented by replacing every variable in its rules with the left-hand sides of its original rules. Unfolding from an augmented TRS is not as powerful as unfolding from the original TRS using variable positions. For instance, loopingness of

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

cannot be proved from the augmented version of $\mathcal{R}$ whereas it can be with variable positions. This TRS corresponds to problem `#4.13.trs` in the Termination Problem Data Base [6] . It was given by Drosten in [16] and is successfully solved by NTI'08 (on and off). The estimation of useless rules described in Section 5 and Section 6 is better than that of [36] in the sense that it removes fewer useful rules. More precisely, both estimations behave similarly except for the unfolded rules $l \to r$ where l and r have different roots. In this case, [36] only considers the roots of the rewritings of r; if the root of l is one of these, then $l \to r$ is kept, otherwise it is removed. Of course, this is not sufficient as we also have to consider the strict subterms of the rewritings of r, as l may semi-unify with one of them. We do so in this paper by defining an eliminating unfolding operator that splits r into its (strict and non-strict) subterms. In [36], splitting is only applied to iteration 0 of the unfolding operator (*i.e.* to the rules of the augmented form of the TRS under analysis).

[6] available from `http://www.lri.fr/~marche/termination-competition/`

8.3 AProVE 1.2

AProVE [25,22,6] is a very powerful tool for automatically proving termination and non-termination of TRSs (it also handles several other formalisms as logic programs, functional programs, ...) Version 1.2 implements the general concept of *dependency pair framework* for combining termination techniques in a modular way [23]. Termination problems are solved by repeatedly decomposing them into smaller sub-problems and non-termination proofs are only performed on those sub-problems that are detected as possibly non-terminating. A major advantage is that the non-termination proofs only have to regard a *subset* of the rules. Considering subsets of rules resembles what we do herein as we use an eliminating mechanism to remove rules that do not contribute to a non-termination proof. One difference is that our technique may remove some rules which are actually useful for proving non-termination (see Example 5.3). In contrast, non-termination proofs in AProVE only regard the sub-problems that the tool could not prove terminating, hence a useful sub-problem for non-termination is never unconsidered.

The idea of the dependency pair framework is to treat a set of dependency pairs $\mathcal{P}$ together with a TRS $\mathcal{R}$ (initially, $\mathcal{P}$ is the set of dependency pairs of $\mathcal{R}$) and to prove absence/presence of infinite $(\mathcal{P}, \mathcal{R})$-chains instead of examining $\underset{\mathcal{R}}{\rightarrow}$. Intuitively, a dependency pair corresponds to a function call and a chain represents a possible sequence of calls. In comparison, our approach directly works with the rules (not the dependency pairs) but our eliminating process splits the right-hand sides of the unfolded rules (Definition 5.10). If a rule $l \rightarrow r$ is split into $l \rightarrow t_i$ where the root of t_i is not a defined symbol, then $l \rightarrow t_i$ is necessarily detected as root-useless because the roots of the ascendants of l are defined symbols and the descendants of t_i all have the same root as t_i. Hence, our approach resembles the dependency pair method from that point of view.

The techniques implemented in AProVE for proving non-termination are described in [24]. They can detect loops only, exactly as our approach. Given $(\mathcal{P}, \mathcal{R})$ as above, the idea consists in narrowing the dependency pairs in $\mathcal{P}$ until the left-hand side of a narrowed pair semi-unifies with the corresponding right-hand side. Narrowing operations are performed either directly with the rules of $\mathcal{R}$ (forward narrowing) or with the reversed rules (backward narrowing). To select forward or backward narrowing, heuristics are introduced: if $\mathcal{P} \cup \mathcal{R}$ is right- and not left-linear, then forward narrowing is performed; otherwise backward narrowing is used and if $\mathcal{P} \cup \mathcal{R}$ is not left-linear then narrowing is also permitted on variable subterms. Notice that the set of descendants (resp. ascendants) of a term t (end of Section 6) provides an over-approximation of the set of forward (resp. backward) narrowings of t but we compute the descendants and the ascendants in order to approximate root-useless rules (not

to detect loops directly). Moreover, in Theorem 6.15, for any rule $l \to r$ we *mix* the ascendants of l with the descendants of r. We also mix forward and backward unfoldings in Definition 3.11. In contrast, [24] either considers the backward narrowings of the left-hand sides or the forward narrowings of the right-hand sides of the dependency pairs. Our tool NTI'08 does not implement any heuristic to select forward/backward narrowing and to permit/forbid unfolding of variables. Instead, by default NTI'08 always unfold variables and runs 3 proofs in parallel: one with forward unfoldings only, one with backward unfoldings only and one with forward and backward unfoldings together. In practice, the results are comparable to those of AProVE on TRSs (see Section 7.1).

The elimination technique presented in this paper can be easily integrated into a non-termination analysis which works on dependency pairs (as in [24]). Given a set of dependency pairs $\mathcal{P}$ together with a TRS $\mathcal{R}$, one can use Theorem 5.13 where $\mathit{eunf}(\mathcal{R})$ is replaced with

$$\bigcup_{n \in \mathbb{N}} (EU_{\mathcal{R}} \uparrow n)(\mathit{elim}_{\mathcal{R}}(\mathcal{P}))$$

and root-uselessness of a rule R is detected with $\mathit{lfp}(\hat{U}_{\mathcal{R},R})$ and Theorem 6.13.

8.4 Matchbox

Matchbox [45,34] is a tool that implements powerful techniques for detecting loops in string rewriting. The 2007 version handles SRSs and their reverse concurrently (as a SRS is non-terminating if and only if its reverse is non-terminating). It also enumerates the forward closures using a priority queue of closures initialised with the SRS of interest; smallest closures are extracted first and for each successor Matchbox'07 checks for loop and inserts into the queue. In parallel with forward closures, Matchbox'07 also extracts *transport systems* [46] and checks whether they are looping (as if a rewriting system admits a looping transport system then it admits a loop). The combination of these techniques is very powerful (see Figure 4) and the transport system approach is able to find long loops very efficiently. NTI'08 also enumerates closures but it does not use priority queues and does not implement the transport system technique.

8.5 TORPA

TORPA (*Termination Of Rewriting Proved Automatically*) [48] is a powerful tool for proving and disproving termination of SRSs. It implements a com-

bination of different techniques (for instance the dependency pair method) and starts the non-termination proof only when the termination analysis has failed. The non-termination proof technique consists in generating a directed graph with labelled nodes which is a fragment of the ancestor graph described in [31,20]. The graph $\mathcal{G}_{\mathcal{R}}$ corresponding to a SRS $\mathcal{R}$ is such that if there is a path from a node labelled by u to a node labelled by u' then $u \xrightarrow[\mathcal{R}]{+} u_1 u' u_2$ for some strings u_1 and u_2. Therefore, if $\mathcal{G}_{\mathcal{R}}$ admits a cycle, then an infinite reduction (corresponding to a loop) exists for $\mathcal{R}$.

8.6 TTT2

TTT2 (*Tyrolean Termination Tool 2*) [28,44] is a powerful tool for automatically proving and disproving termination of rewrite systems. It implements the dependency pair method and computes the strongly connected components (SCC) of an over-approximation of the dependency graph; each SCC is processed separately by a recursive algorithm that uses a bunch of different methods. Hence, several comments we made in Section 8.3 are also applicable to TTT2. Unlike our approach, the non-termination proof technique of TTT2 does not enumerate closures until a looping one is found. During the Termination Competition'07, TTT2 employed only a simple non-termination check (it checked whether the left-hand side of a rule is contained in its right-hand side) performed before any termination analysis. Notice however that a more elegant and powerful method is being implemented for SRSs. In order to get smaller SCCs it tries to remove rules which cannot contribute to a non-terminating sequence. Then, it encodes rewrite sequences as propositional formulæ; if a formula is satisfiable then the corresponding sequence includes a looping reduction [47]. The formulæ are handled by the SAT solver MiniSat [17].

8.7 Other related works

In [5,2], 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 [5,2] 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, [5,2] consider unification instead of semi-unification because the aim of the authors is to detect non-termination of narrowing. We are also aware of the work described in [10] 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 [5,2] consider a static approach.

In [1], the authors present a generic scheme for debugging functional programs modeled as TRSs. The debugging methodology is based on abstract interpretation and proceeds by approximating a continuous immediate consequence operator by means of a $depth(k)$ cut. This is related to our work as in Section 6 we also approximate a continuous operator by pruning the left-hand or right-hand side of rewrite rules.

9 Conclusion

Although it removes many root-useless rules, the loop detection analysis implemented in NTI'08 still suffers from the explosion of the search space. A solution to extend the applicability and reduce the cost of our approach consists in *underapproximating* the unfoldings. The current version of our analyser increments a depth k, starting from zero, and proceeds as follows: during the computation of the unfoldings, a generated rule $l \to r$ is discarded if the depth of l or r is more than k (notice that for a given value of k, the set of the unfoldings whose depth is not more than k is finite); if no rule satisfying the semi-unification criterion of Theorem 5.13 is found, then k is incremented by one. Hence, for each value of k, the analyser computes a *subset* of the unfoldings. Another candidate to underapproximation is the method described in [40] which does not consider subsets of the set C being approximated but some set C' such that $C' \cap C \neq \varnothing$ instead.

Another possibility to reduce the search space consists in integrating our approach into the dependency pair method, *i.e.* to apply it to the strongly connected components of the dependency graph that are detected as possibly non-terminating. This would reduce the number of unfolded rules as the unfolding process would only consider a subset of the dependency pairs. Moreover, the heuristics implemented in AProVE to permit/forbid unfolding of variables would possibly lead to more successful proofs (for instance, `TRCSR/Ex1_GM99_iGM` is not solved by NTI'08 unless variable positions are not unfolded, see Section 7.1).

Acknowledgments

The author thanks the anonymous referees for their numerous helpful comments on this work. He also thanks Johannes Waldmann and Harald Zankl for explanations regarding the non-termination proof techniques implemented in Matchbox and TTT2 respectively.

References

[1] María Alpuente, Marco Comini, Santiago Escobar, Moreno Falaschi, and Salvador Lucas. Abstract diagnosis of functional programs. In Michael Leuschel, editor, *Proc. of the 12th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'02)*, volume 2664 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2003.

[2] María Alpuente, Moreno Falaschi, and Ferdinando Manzo. Analyses of unsatisfiability for equational logic programming. *Journal of Logic Programming*, 311(1–3):479–525, 1995.

[3] María Alpuente, Moreno Falaschi, Gines Moreno, and Germán Vidal. Safe folding/unfolding with conditional narrowing. In Michael Hanus, Jan Heering, and Karl Meinke, editors, *Proc. of Algebraic and Logic Programming, 6th International Joint Conference (ALP/HOA 97)*, volume 1298 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1997.

[4] María Alpuente, Moreno Falaschi, Gines Moreno, and Germán Vidal. Rules + strategies for transforming lazy functional logic programs. *Theoretical Computer Science*, 311(1–3):479–525, 2004.

[5] María Alpuente, Moreno Falaschi, María José Ramis, and Germán Vidal. Narrowing approximations as an optimization for equational logic programs. In Maurice Bruynooghe and Jaan Penjam, editors, *Proc. of the 5th International Symposium on Programming Language Implementation and Logic Programming (PLILP'93)*, volume 714 of *Lecture Notes in Computer Science*, pages 391–409. Springer-Verlag, 1993.

[6] AProVE's web site. `http://aprove.informatik.rwth-aachen.de/`.

[7] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[8] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge, 1998.

[9] Rod M. Burstall and John Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

[10] Jacques Chabin and Pierre Réty. Narrowing directed by a graph of terms. In G. Goos and J. Hartmanis, editors, *Proc. of the 4th International Conference on Rewriting Techniques and Applications (RTA'91)*, volume 488 of *Lecture Notes in Computer Science*, pages 112–123. Springer-Verlag, 1991.

[11] Michael Codish and Cohavit Taboch. A semantics basis for termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.

[12] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unifed lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of the 4th Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM, 1977.

[13] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. of the 5th Symposium on Principles of Programming Languages (POPL'78)*, pages 84–96. ACM, 1978.

[14] Nachum Dershowitz. Termination of linear rewriting systems. In *Proc. of the 8th International Colloquium on Automata, Languages and Programming (ICALP'81)*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458. Springer, 1981.

[15] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.

[16] Klaus Drosten. *Termersetzungssysteme: Grundlagen der Prototyp-Generierung algebraischer Spezifikationen*. Springer Verlag, Berlin, 1989.

[17] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proc. of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer-Verlag, 2004.

[18] Maurizio Gabbrielli and Roberto Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In *Proc. of the ACM Symposium on Applied Computing (SAC'94)*, pages 394–399. ACM Press, 1994.

[19] Samir Genaim and Michael Codish. Inferring termination condition for logic programs using backwards analysis. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Proc. of Logic for Programming, Artificial intelligence and Reasoning (LPAR'01)*, volume 2250 of *Lecture Notes in Computer Science*, pages 685–694. Springer-Verlag, 2001.

[20] Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Deciding termination for ancestor match-bounded string rewriting systems. Technical report, National Institute of Aerospace, Hampton, VA, 2004.

[21] Alfons Geser and Hans Zantema. Non-looping string rewriting. *RAIRO Theoretical Informatics and Applications*, 33:279–301, 1999.

[22] Jürgen Giesl, Peter Schneider-Kamp, and René Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In Ulrich Furbach and Natarajan Shankar, editors, *Proc. of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 281–286. Springer-Verlag, 2006.

[23] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. The dependency pair framework: combining techniques for automated termination proofs. In Franz Baader and Andrei Voronkov, editors, *Proc. of the 11th International Conference on Logic for Programming, Artificial intelligence and Reasoning (LPAR'04)*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 210–220. Springer-Verlag, 2004.

[24] Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. Proving and disproving termination of higher-order functions. In Bernhard Gramlich, editor, *Proc. of the 5th International Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *Lecture Notes in Artificial Intelligence*, pages 216–231. Springer-Verlag, 2005.

[25] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Automated termination proofs with AProVE. In Vincent van Oostrom, editor, *Proc. of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220. Springer-Verlag, 2004.

[26] John V. Guttag, Deepak Kapur, and David R. Musser. On proving uniform termination and restricted termination of rewriting systems. *SIAM Journal of Computing*, 12(1):189–214, 1983.

[27] Michael Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19,20:583–628, 1994.

[28] Nao Hirokawa and Aart Middeldorp. Tyrolean Termination Tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.

[29] Steffen Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, 1989.

[30] Deepak Kapur, David R. Musser, Paliath Narendran, and Jonathan Stillman. Semi-unification. *Theoretical Computer Science*, 81:169–187, 1991.

[31] Winfried Kurth. *Termination und konfluenz von semi-Thue-systemen mit nur einer regel*. PhD thesis, Technische Universität Clausthal, Germany, 1990.

[32] D.S. Lankford and David R. Musser. A finite termination criterion. Unpublished Draft, USC Information Sciences Institute, Marina Del Rey, CA, 1978.

[33] Claude Marché and Hans Zantema. The termination competition. In Franz Baader, editor, *Proc. of Term Rewriting and Applications, 18th International Conference (RTA'07)*, volume 4533 of *Lecture Notes in Computer Science*, pages 303–313. Springer-Verlag, 2007.

[34] Matchbox's web site. `http://dfa.imn.htwk-leipzig.de/matchbox/`.

[35] Fred Mesnard and Roberto Bagnara. cTI: a constraint-based termination inference tool for iso-prolog. *Theory and Practice of Logic Programming*, 5(1–2):243–257, 2005.

[36] Étienne Payet. Detecting non-termination of term rewriting systems using an unfolding operator. In Germán Puebla, editor, *Proc. of the 16th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'06)*, volume 4407 of *Lecture Notes in Computer Science*, pages 194–209. Springer-Verlag, 2007.

[37] Étienne Payet and Fred Mesnard. Non-termination inference for constraint logic programs. In Roberto Giacobazzi, editor, *Proc. of the 11th International Symposium on Static Analysis (SAS'04)*, volume 3148 of *Lecture Notes in Computer Science*, pages 377–392. Springer-Verlag, 2004.

[38] Étienne Payet and Fred Mesnard. Non-termination inference of logic programs. *ACM Transactions on Programming Languages and Systems*, 28, Issue 2:256–289, March 2006.

[39] Alberto Pettorossi and Maurizio Proietti. Rules and strategies for transforming functional and logic programs. *ACM Computing Surveys*, 28(2):360–414, 1996.

[40] David A. Schmidt. A calculus of logical relations for over- and underapproximating static analyses. *Science of Computer Programming*, 64(1):29–53, 2007.

[41] Joachim Steinbach. Simplification orderings: history of results. *Fundamenta Informaticae*, 24:47–87, 1995.

[42] Alfred Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[43] Yoshihito Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.

[44] TTT2's web site. `http://colo6-c703.uibk.ac.at/ttt2`.

[45] Johannes Waldmann. Matchbox: A tool for match-bounded string rewriting. In Vincent van Oostrom, editor, *Proc. of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94. Springer-Verlag, 2004.

[46] Johannes Waldmann. Compressed loops (draft). Available at `http://dfa.imn.htwk-leipzig.de/matchbox/methods/`, 2007.

[47] Harald Zankl and Aart Middeldorp. Nontermination of string rewriting using SAT. In *Proc. of the 9th International Workshop on Termination (WST'07)*, pages 52–55, 2007.

[48] Hans Zantema. Termination of string rewriting proved automatically. *Journal of Automated Reasoning*, 34(2):105–139, 2005.

10 Proofs

10.1 Additional notations and results

For any substitution θ and any set of variables V, we let $\theta|V$ denote the substitution obtained from θ by restricting its domain to V. For any renaming γ, we denote by γ^{-1} the unique substitution such that $\gamma\gamma^{-1} = \gamma^{-1}\gamma = \varnothing$. The substitution γ^{-1} is also a renaming.

10.2 Proof of Lemma 3.6

Proof.

- Let $l_1 \to r_1 \in F_{\mathcal{R}}(X)$. Then by Definition 3.4, there exist $R = l \to r \in X$, $p \in Pos(r)$, $l' \to r' \in \mathcal{R}$ and $\theta = mgu(r|_p, l')$ such that

$$l_1 \to r_1 = (l \to r[p \leftarrow r'])\theta \ .$$

  Therefore, $l\theta \to r\theta$ is an instance of a rule of X, $p \in Pos(r\theta)$, $l' \to r' \in \mathcal{R}$ and $r\theta|_p = r|_p\theta$ (because $p \in Pos(r)$) with $r|_p\theta = l'\theta$ (because $\theta = mgu(r|_p, l')$). So, by Definition 3.1, $l\theta \to r\theta[p \leftarrow r'\theta] \in I_{\mathcal{R}}(X)$. As $r\theta[p \leftarrow r'\theta] = r[p \leftarrow r']\theta$ (because $p \in Pos(r)$), we have $l_1 \to r_1 \in I_{\mathcal{R}}(X)$.
- Consider $\mathcal{R} = \{\mathsf{f}(\mathsf{g}(x)) \to x,\ \mathsf{0} \to \mathsf{1}\}$. Then,

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

  Moreover, $\mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{0}))) \to \mathsf{g}(\mathsf{0})$ is an instance of $\mathsf{f}(\mathsf{g}(x)) \to x$ from which we get $\mathsf{f}(\mathsf{g}(\mathsf{g}(\mathsf{0}))) \to \mathsf{g}(\mathsf{1}) \in I_{\mathcal{R}}(\{\mathsf{f}(\mathsf{g}(x)) \to x\})$. Therefore,

$$I_{\mathcal{R}}(\{\mathsf{f}(\mathsf{g}(x)) \to x\}) \not\subseteq F_{\mathcal{R}}(\{\mathsf{f}(\mathsf{g}(x)) \to x\}) \ .$$

□

10.3 Proof of Lemma 3.9

Proof.

- Let $\mathcal{R} = \{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1}),\ \mathsf{1} \to \mathsf{2}\}$. Then, $\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{2}) \in F_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\})$ and $B_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\}) = \varnothing$. Hence, $F_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\}) \not\subseteq B_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\})$.
- Let $\mathcal{R} = \{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1}),\ \mathsf{2} \to \mathsf{0}\}$. Then, $\mathsf{f}(\mathsf{2}) \to \mathsf{g}(\mathsf{1}) \in B_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\})$ and $F_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\}) = \varnothing$. Hence, $B_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\}) \not\subseteq F_{\mathcal{R}}(\{\mathsf{f}(\mathsf{0}) \to \mathsf{g}(\mathsf{1})\})$.

□

10.4 Proof of Theorem 4.1

Proof. Suppose that there is $l \to r$ in $unf(\mathcal{R})$ and a subterm r' of r such that $l\theta_1\theta_2 = r'\theta_1$ for some substitutions θ_1 and θ_2. By Proposition 3.12, $l \xrightarrow{+} r$. Since $\underset{\mathcal{R}}{\to}$ is stable, we have $l\theta_1 \underset{\mathcal{R}}{\overset{+}{\to}} r\theta_1$ *i.e.* $l\theta_1 \underset{\mathcal{R}}{\overset{+}{\to}} C[r'\theta_1]$ for some context C *i.e.* $l\theta_1 \underset{\mathcal{R}}{\overset{+}{\to}} C[l\theta_1\theta_2]$. Hence, $l\theta_1$ loops *w.r.t.* $\mathcal{R}$. □

10.5 Proof of Proposition 5.5

The proof of Proposition 5.5 follows from the lemmas below.

Lemma 10.1 *$\leq$ is a partial order on $\mathbb{R}(\mathcal{F}, \mathcal{V})$.*

Proof. Reflexivity and transitivity follow from those of $\subseteq$. Let us check antisymmetry. Let $X, X_1 \in \mathbb{R}(\mathcal{F}, \mathcal{V})$. Suppose that $X \leq X_1$ and $X_1 \leq X$. Then, we have $instances(X) \subseteq instances(X_1)$ and $instances(X_1) \subseteq instances(X)$. So, $instances(X) = instances(X_1)$, *i.e.* X and X_1 are equal up to variable renaming, hence they denote the same element of $\mathbb{R}(\mathcal{F}, \mathcal{V})$. $\square$

Lemma 10.2 *For any $I \subseteq \mathbb{N}$ and any $\{X_i\}_{i \in I} \subseteq \mathbb{R}(\mathcal{F}, \mathcal{V})$, the least upper bound of $\{X_i\}_{i \in I}$ is $\sqcup_{i \in I} X_i$.*

Proof. For all $j \in I$, $X_j \subseteq \cup_{i \in I} X_i$ so $instances(X_j) \subseteq instances(\cup_{i \in I} X_i)$ *i.e.* $X_j \leq \cup_{i \in I} X_i$ *i.e.* $X_j \leq \sqcup_{i \in I} X_i$. Hence, $\sqcup_{i \in I} X_i$ is an upper bound of $\{X_i\}_{i \in I}$.

Let B be another upper bound of $\{X_i\}_{i \in I}$. Then for all $i \in I$, we have $X_i \leq B$ *i.e.* $instances(X_i) \subseteq instances(B)$. Therefore, $\cup_{i \in I} instances(X_i) \subseteq instances(B)$ *i.e.* $instances(\cup_{i \in I} X_i) \subseteq instances(B)$. Hence, $\cup_{i \in I} X_i \leq B$ *i.e.* $\sqcup_{i \in I} X_i \leq B$.

So, $\sqcup_{i \in I} X_i$ is a least upper bound of $\{X_i\}_{i \in I}$. As $\leq$ is antisymmetric, there is only one least upper bound. $\square$

Lemma 10.3 *For any $I \subseteq \mathbb{N}$ and any $\{X_i\}_{i \in I} \subseteq \mathbb{R}(\mathcal{F}, \mathcal{V})$, the greatest lower bound of $\{X_i\}_{i \in I}$ is $\sqcap_{i \in I} X_i$.*

Proof. Let $j \in I$. We have $\cap_{i \in I} instances(X_i) \subseteq instances(X_j)$ so $\sqcap_{i \in I} X_i \subseteq instances(X_j)$. Therefore, $instances(\sqcap_{i \in I} X_i) \subseteq instances(instances(X_j))$ *i.e.* $instances(\sqcap_{i \in I} X_i) \subseteq instances(X_j)$ *i.e.* $\sqcap_{i \in I} X_i \leq X_j$. Consequently, $\sqcap_{i \in I} X_i$ is a lower bound of $\{X_i\}_{i \in I}$.

Let B be another lower bound of $\{X_i\}_{i \in I}$. Then for all $i \in I$, we have $B \leq X_i$ *i.e.* $instances(B) \subseteq instances(X_i)$. So, $instances(B) \subseteq \cap_{i \in I} instances(X_i)$ *i.e.* $instances(B) \subseteq \sqcap_{i \in I} X_i$. Hence, $instances(instances(B)) \subseteq instances(\sqcap_{i \in I} X_i)$ *i.e.* $instances(B) \subseteq instances(\sqcap_{i \in I} X_i)$ *i.e.* $B \leq \sqcap_{i \in I} X_i$.

So, $\sqcap_{i \in I} X_i$ is a greatest lower bound of $\{X_i\}_{i \in I}$. As $\leq$ is antisymmetric, there is only one greatest lower bound. $\square$

Lemma 10.4 *$\varnothing$ is the least element of $\langle \mathbb{R}(\mathcal{F}, \mathcal{V}), \leq \rangle$.*

Proof. $\varnothing$ is a least element of $\langle \mathbb{R}(\mathcal{F}, \mathcal{V}), \leq \rangle$. Moreover, as $\leq$ is antisymmetric, there is only one least element. $\square$

Lemma 10.5 *$\mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V})$ is the greatest element of $\langle \mathbb{R}(\mathcal{F},\mathcal{V}), \leq \rangle$.*

Proof. $\mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V})$ is a greatest element of $\langle \mathbb{R}(\mathcal{F},\mathcal{V}), \leq \rangle$. Moreover, as $\leq$ is antisymmetric, there is only one greatest element. □

10.6 Proof of Proposition 5.7

As $F_{\mathcal{R}}$ and $B_{\mathcal{R}}$ are additive on $\langle \mathbb{R}(\mathcal{F},\mathcal{V}), \leq, \sqcup, \sqcap, \varnothing, \mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V}) \rangle$, so is $U_{\mathcal{R}}$. Therefore, for any rewrite rule R, $U_{\mathcal{R},R}$ is additive, hence continuous.

10.7 Proof of Lemma 5.8

First, we need a lemma.

Lemma 10.6 *Let R be a rewrite rule. For any $n \in \mathbb{N}$, we have*

$$(U_{\mathcal{R},R} \uparrow n+1)(\varnothing) = \bigcup_{i \leq n} (U_{\mathcal{R}} \uparrow i)(\{R\}) \; .$$

Proof. By induction on n, using the definition of $U_{\mathcal{R},R}$ and the fact that $U_{\mathcal{R}}$ is additive. □

We can now prove Lemma 5.8:

Proof. By Proposition 5.7, $lfp(U_{\mathcal{R},R}) = \bigsqcup_{n \in \mathbb{N}} (U_{\mathcal{R},R} \uparrow n)(\varnothing) = \bigcup_{n \in \mathbb{N}} (U_{\mathcal{R},R} \uparrow n)(\varnothing)$. Moreover,

$$\begin{aligned} \bigcup_{n \in \mathbb{N}} (U_{\mathcal{R},R} \uparrow n)(\varnothing) &= \bigcup_{1 \leq n} (U_{\mathcal{R},R} \uparrow n)(\varnothing) \\ &= \bigcup_{1 \leq n} \bigcup_{i \leq n-1} (U_{\mathcal{R}} \uparrow i)(\{R\}) \quad \text{by Lemma 10.6} \\ &= \bigcup_{0 \leq n} (U_{\mathcal{R}} \uparrow n)(\{R\}) \end{aligned}$$

□

10.8 Proof of Theorem 5.13

First, we need a lemma and a proposition.

Lemma 10.7 *Let $\mathcal{R}$ be a TRS and $l' \to r'$ be a rule. Then, for any rule $l \to r \in elim_{\mathcal{R}}(l' \to r')$, we have $l = l'$ and r is a subterm of r'.*

Proof. If $elim_{\mathcal{R}}(l' \to r')$ is empty, then the result holds vacuously. Otherwise, proceed by structural induction on r'. $\square$

Proposition 10.8 *Let $n \in \mathbb{N}$. For any $l \to r \in (EU_{\mathcal{R}} \uparrow n)(elim_{\mathcal{R}}(\mathcal{R}))$, there exists $l' \to r' \in (U_{\mathcal{R}} \uparrow n)(\mathcal{R})$ such that $l = l'$ and r is a subterm of r'.*

Proof. By induction on n using Lemma 10.7 and the definition of $EU_{\mathcal{R}}$, $U_{\mathcal{R}}$, $F_{\mathcal{R}}$ and $B_{\mathcal{R}}$. $\square$

We can now prove Theorem 5.13:

Proof. As $l \to r \in eunf(\mathcal{R})$, then by Definition 5.12 there exists $n \in \mathbb{N}$ such that $l \to r \in (EU \uparrow n)(elim_{\mathcal{R}}(\mathcal{R}))$. Hence, by Proposition 10.8, there exists $l' \to r' \in (U_{\mathcal{R}} \uparrow n)(\mathcal{R})$, *i.e.* $l' \to r' \in unf(\mathcal{R})$ by Definition 3.11, such that $l = l'$ and r is a subterm of r'. So, as $l\theta_1\theta_2 = r\theta_1$ for some substitutions θ_1 and θ_2, by Theorem 4.1 we have that $l\theta_1$ loops *w.r.t.* $\mathcal{R}$. $\square$

10.9 Proof of Proposition 6.3

The proof of Proposition 6.3 follows from the lemmas below.

Lemma 10.9 *$\preceq$ is a partial order on $\mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$.*

Proof. Reflexivity and transitivity follow from those of $\leq$. Antisymmetry follows from that of $\leq$ and the fact that we identify the elements of $\mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ having the same concretisation modulo $\leq$. $\square$

Lemma 10.10 *For any $I \subseteq \mathbb{N}$ and any $\{X_i\}_{i \in I} \subseteq \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, the least upper bound of $\{X_i\}_{i \in I}$ is $\vee_{i \in I} X_i$.*

Proof. Let $j \in I$. We have $X_j \subseteq \cup_{i \in I} X_i$, hence $\gamma(X_j) \subseteq \gamma(\cup_{i \in I} X_i)$, hence $instances(\gamma(X_j)) \subseteq instances(\gamma(\cup_{i \in I} X_i))$. Therefore, $X_j \preceq \cup_{i \in I} X_i$ *i.e.* $X_j \preceq \vee_{i \in I} X_i$. Consequently, $\vee_{i \in I} X_i$ is an upper bound of $\{X_i\}_{i \in I}$.

Let B be another upper bound of $\{X_i\}_{i \in I}$. Then for all $i \in I$, $X_i \preceq B$ *i.e.* $instances(\gamma(X_i)) \subseteq instances(\gamma(B))$. Consequently, $\cup_{i \in I} instances(\gamma(X_i)) \subseteq instances(\gamma(B))$ which yields $instances(\cup_{i \in I} \gamma(X_i)) \subseteq instances(\gamma(B))$ *i.e.* $\cup_{i \in I} \gamma(X_i) \leq \gamma(B)$. Therefore, we have $\gamma(\cup_{i \in I} X_i) \leq \gamma(B)$ *i.e.* $\cup_{i \in I} X_i \preceq B$ *i.e.* $\vee_{i \in I} X_i \preceq B$.

So, $\vee_{i \in I} X_i$ is a least upper bound of $\{X_i\}_{i \in I}$. As $\preceq$ is antisymmetric, there is only one least upper bound. $\square$

Lemma 10.11 *For any $I \subseteq \mathbb{N}$ and any $\{X_i\}_{i \in I} \subseteq \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, the greatest lower bound of $\{X_i\}_{i \in I}$ is $\wedge_{i \in I} X_i$.*

Proof. Let $j \in I$. We have $\cap_{i\in I} instances(\gamma(X_i)) \subseteq instances(\gamma(X_j))$, so $\wedge_{i\in I} X_i \subseteq instances(\gamma(X_j))$. Hence, $\gamma(\wedge_{i\in I} X_i) \subseteq \gamma(instances(\gamma(X_j)))$. As no element of $\hat{\mathcal{V}}$ occurs in $instances(\gamma(X_j))$, then $\gamma(\wedge_{i\in I} X_i) \subseteq instances(\gamma(X_j))$. So $instances(\gamma(\wedge_{i\in I} X_i)) \subseteq instances(instances(\gamma(X_j))) \subseteq instances(\gamma(X_j))$ *i.e.* $\wedge_{i\in I} X_i \preceq X_j$. Consequently, $\wedge_{i\in I} X_i$ is a lower bound of $\{X_i\}_{i\in I}$.

Let B be another lower bound of $\{X_i\}_{i\in I}$. Then for all $i \in I$, we have $B \preceq X_i$ *i.e.* $instances(\gamma(B)) \subseteq instances(\gamma(X_i))$. Consequently, $instances(\gamma(B)) \subseteq \cap_{i\in I} instances(\gamma(X_i))$ *i.e.* $instances(\gamma(B)) \subseteq \wedge_{i\in I} X_i$. Therefore, we have that $\gamma(instances(\gamma(B))) \subseteq \gamma(\wedge_{i\in I} X_i)$. As no element of $\hat{\mathcal{V}}$ occurs in $instances(\gamma(B))$ we have $instances(\gamma(B)) \subseteq \gamma(\wedge_{i\in I} X_i)$. Hence, $instances(instances(\gamma(B))) \subseteq instances(\gamma(\wedge_{i\in I} X_i))$ *i.e.* $instances(\gamma(B)) \subseteq instances(\gamma(\wedge_{i\in I} X_i))$ *i.e.* $B \preceq \wedge_{i\in I} X_i$.

So, $\wedge_{i\in I} X_i$ is a greatest lower bound of $\{X_i\}_{i\in I}$. As $\preceq$ is antisymmetric, there is only one greatest lower bound. □

Lemma 10.12 *$\varnothing$ is the least element of $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq \rangle$.*

Proof. $\varnothing$ is a least element of $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq \rangle$. Moreover, as $\preceq$ is antisymmetric, there is only one least element. □

Lemma 10.13 *$\mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ is the greatest element of $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq \rangle$.*

Proof. $\mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ is a greatest element of $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq \rangle$. Moreover, as $\preceq$ is antisymmetric, there is only one greatest element. □

10.10 Proof of Proposition 6.4

Proof. By Definition 6.2, we have $\gamma(\wedge_{i\in I} X_i) = \gamma(\cap_{i\in I} instances(\gamma(X_i)))$. So, $\gamma(\wedge_{i\in I} X_i) = \cap_{i\in I} instances(\gamma(X_i))$ because $\cap_{i\in I} instances(\gamma(X_i))$ contains no element of $\hat{\mathcal{V}}$. Hence, by Definition 5.4, $\gamma(\wedge_{i\in I} X_i) = \sqcap_{i\in I} \gamma(X_i)$. □

10.11 Proof of Proposition 6.9

Proof. As $\hat{F}_{\mathcal{R}}$ and $\hat{B}_{\mathcal{R}}$ are additive on $\langle \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}), \preceq, \vee, \wedge, \varnothing, \mathcal{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}}) \rangle$, so is $\hat{U}_{\mathcal{R},R}$. Hence, $\hat{U}_{\mathcal{R},R}$ is continuous. □

10.12 Proof of Proposition 6.10

First we need some additional results.

Proposition 10.14 *For all $X \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ we have*

$$F_{\mathcal{R}}(\gamma(X)) \subseteq \mathit{instances}(\gamma(\hat{F}_{\mathcal{R}}(X) \cup X)) \; .$$

Proof. Let $l \to r \in F_{\mathcal{R}}(\gamma(X))$. Then by Definition 3.4, there exist $R_1 = l_1 \to r_1 \in \gamma(X)$, $p \in \mathit{Pos}(r_1)$, $l' \to r' \in \mathcal{R}$ and $\theta = \mathit{mgu}(r_1|_p, l')$ such that

$$l \to r = (l_1 \to r_1[p \leftarrow r'])\theta \; .$$

As $l_1 \to r_1 \in \gamma(X)$, there exists $\hat{l} \to \hat{r} \in X$ such that $l_1 \to r_1 \in \gamma(\hat{l} \to \hat{r})$. Then by Definition 6.1, there is a substitution $\hat{\sigma}$ such that $\mathit{Dom}(\hat{\sigma}) \subseteq \hat{\mathcal{V}}$ and

$$l_1 \to r_1 = (\hat{l} \to \hat{r})\hat{\sigma} \; .$$

- Suppose that $p \notin \mathit{Pos}(\hat{r})$ or that $p \in \mathit{Pos}(\hat{r})$ with $\hat{r}|_p \in \hat{\mathcal{V}}$. Then, there exists $\hat{p} \in \mathit{Pos}(\hat{r})$ such that $\hat{r}|_{\hat{p}} \in \hat{\mathcal{V}}$ and $r_1|_p$ is a subterm of $\hat{\sigma}(\hat{r}|_{\hat{p}})$, *i.e.* $\hat{\sigma}(\hat{r}|_{\hat{p}})|_q = r_1|_p$ for a position q of $\hat{\sigma}(\hat{r}|_{\hat{p}})$. As $\hat{r}|_{\hat{p}}$ is a variable that occurs only once in $\hat{l} \to \hat{r}$, we can define the substitution $\hat{\eta}$ as:

$$\begin{aligned} \hat{\eta}(\hat{x}) &= \hat{\sigma}(\hat{x}) \quad \forall \hat{x} \in \hat{\mathcal{V}} \setminus \{\hat{r}|_{\hat{p}}\} \\ \hat{\eta}(\hat{r}|_{\hat{p}}) &= \hat{\sigma}(\hat{r}|_{\hat{p}})[q \leftarrow r'] \end{aligned}$$

  and we have $\hat{l}\hat{\eta} = \hat{l}\hat{\sigma} = l_1$ and $\hat{r}\hat{\eta} = r_1[p \leftarrow r']$. Then,

$$l \to r = (\hat{l} \to \hat{r})\hat{\eta}\theta \; .$$

  We have $\hat{l} \to \hat{r} \in X$, $(\hat{l} \to \hat{r})\hat{\eta} \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathit{Dom}(\hat{\eta}) \subseteq \hat{\mathcal{V}}$. So, $(\hat{l} \to \hat{r})\hat{\eta} \in \gamma(X)$. Hence, $l \to r \in \mathit{instances}(\gamma(X))$. As $X \subseteq \hat{F}_{\mathcal{R}}(X) \cup X$,

$$l \to r \in \mathit{instances}(\gamma(\hat{F}_{\mathcal{R}}(X) \cup X)) \; .$$

- Suppose that $p \in \mathit{Pos}(\hat{r})$ and $\hat{r}|_p \notin \hat{\mathcal{V}}$. Then, as $r_1 = \hat{r}\hat{\sigma}$, we have $r_1|_p = \hat{r}|_p\hat{\sigma}$. As $\theta = \mathit{mgu}(r_1|_p, l')$, we have $r_1|_p\theta = l'\theta$, so

$$\hat{r}|_p\hat{\sigma}\theta = l'\theta \; .$$

  Let γ be a renaming such that $(l' \to r')\gamma$ is variable disjoint with $\hat{l} \to \hat{r}$. Let

$$\eta = \hat{\sigma}\theta|\,\mathit{Var}(\hat{r}|_p) \quad \text{and} \quad \eta' = \gamma^{-1}\theta|\,\mathit{Var}(l'\gamma) \; .$$

  Then, $\eta \cup \eta'$ is a well-defined substitution because $l'\gamma$ is variable disjoint with $\hat{r}|_p$. We have:

$$\hat{r}|_p(\eta \cup \eta') = \hat{r}|_p\eta = \hat{r}|_p\hat{\sigma}\theta = l'\theta = (l'\gamma)\gamma^{-1}\theta = (l'\gamma)\eta' = (l'\gamma)(\eta \cup \eta') \; .$$

  Hence, $\hat{r}|_p$ and $l'\gamma$ unify. Therefore, we have:

· $\hat{l} \to \hat{r} \in X$,
· $p \in Pos(\hat{r})$ with $\hat{r}|_p \notin \hat{\mathcal{V}}$,
· $(l' \to r')\gamma \in \mathcal{R}$,
· $\hat{r}|_p$ and $l'\gamma$ unify,
· $prune(r'\gamma) = prune(r')$.
Consequently by Definition 6.8,

$$\hat{l} \to \hat{r}[p \leftarrow prune(r')] \in \hat{F}_{\mathcal{R}}(X) \ .$$

As function *prune* introduces new variables from $\hat{\mathcal{V}}$, we can define the substitution $\hat{\eta}$ as:

$$\hat{\eta}(\hat{x}) = \hat{\sigma}(\hat{x}) \quad \forall \hat{x} \in \hat{\mathcal{V}} \setminus Var(prune(r'))$$
$$prune(r')\hat{\eta} = r'$$

and we have $\hat{l}\hat{\eta} = \hat{l}\hat{\sigma} = l_1$ and $\hat{r}[p \leftarrow prune(r')]\hat{\eta} = \hat{r}\hat{\sigma}[p \leftarrow prune(r')\hat{\eta}] = \hat{r}\hat{\sigma}[p \leftarrow r'] = r_1[p \leftarrow r']$. Therefore,

$$l \to r = (\hat{l} \to \hat{r}[p \leftarrow prune(r')])\hat{\eta}\theta$$

with $\hat{l} \to \hat{r}[p \leftarrow prune(r')] \in \hat{F}_{\mathcal{R}}(X)$, $(\hat{l} \to \hat{r}[p \leftarrow prune(r')])\hat{\eta} \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $Dom(\hat{\eta}) \subseteq \hat{\mathcal{V}}$. So, $(\hat{l} \to \hat{r}[p \leftarrow prune(r')])\hat{\eta} \in \gamma(\hat{F}_{\mathcal{R}}(X))$. Hence, $l \to r \in instances(\gamma(\hat{F}_{\mathcal{R}}(X)))$. As $\hat{F}_{\mathcal{R}}(X) \subseteq \hat{F}_{\mathcal{R}}(X) \cup X$, we have

$$l \to r \in instances(\gamma(\hat{F}_{\mathcal{R}}(X) \cup X)) \ .$$

□

Proposition 10.15 *For all* $X \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, $F_{\mathcal{R}}(\gamma(X)) \leq \gamma(\hat{F}_{\mathcal{R}}(X) \cup X)$.

Proof. Let $R \in instances(F_{\mathcal{R}}(\gamma(X)))$. Then, $R = R_1\theta_1$ for $R_1 \in F_{\mathcal{R}}(\gamma(X))$ and a substitution θ_1. By Proposition 10.14, $R_1 = R_2\theta_2$ for $R_2 \in \gamma(\hat{F}_{\mathcal{R}}(X) \cup X)$ and a substitution θ_2. Hence, $R = R_2\theta_2\theta_1$, *i.e.* $R \in instances(\gamma(\hat{F}_{\mathcal{R}}(X) \cup X))$. So, we have proved that $instances(F_{\mathcal{R}}(\gamma(X))) \subseteq instances(\gamma(\hat{F}_{\mathcal{R}}(X) \cup X))$ *i.e.* $F_{\mathcal{R}}(\gamma(X)) \leq \gamma(\hat{F}_{\mathcal{R}}(X) \cup X)$. □

Proposition 10.16 *For all* $X \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$ *we have*

$$B_{\mathcal{R}}(\gamma(X)) \subseteq instances(\gamma(\hat{B}_{\mathcal{R}}(X) \cup X)) \ .$$

Proof. The proof is similar to that of Proposition 10.14. □

Proposition 10.17 *For all* $X \in \mathbb{R}(\mathcal{F}, \mathcal{V} \cup \hat{\mathcal{V}})$, $B_{\mathcal{R}}(\gamma(X)) \leq \gamma(\hat{B}_{\mathcal{R}}(X) \cup X)$.

Proof. The proof is similar to that of Proposition 10.15. □

We can now prove Proposition 6.10:

Proof. By Propositions 10.15 and 10.17,

$$F_{\mathcal{R}}(\gamma(X)) \cup B_{\mathcal{R}}(\gamma(X)) \cup \{R\} \leq \gamma(\hat{F}_{\mathcal{R}}(X) \cup X) \cup \gamma(\hat{B}_{\mathcal{R}}(X) \cup X) \cup \{R\}$$

with $\{R\} = \gamma(\{R\})$ and

$$\gamma(\hat{F}_{\mathcal{R}}(X) \cup X) \cup \gamma(\hat{B}_{\mathcal{R}}(X) \cup X) \cup \gamma(\{R\}) = \gamma(\hat{F}_{\mathcal{R}}(X) \cup \hat{B}_{\mathcal{R}}(X) \cup X \cup \{R\}) \ .$$

So, $U_{\mathcal{R}}(\gamma(X)) \cup \{R\} \leq \gamma(\hat{F}_{\mathcal{R}}(X) \cup \hat{B}_{\mathcal{R}}(X) \cup X \cup \{R\})$ *i.e.*

$$U_{\mathcal{R},R}(\gamma(X)) \leq \gamma(\hat{U}_{\mathcal{R},R}(X)) \ .$$

□

10.13 Proof of Theorem 6.13

Proof. By contraposition. Suppose that R is not root-useless for $\mathcal{R}$ *i.e.*, by Definition 5.9, that there exists $l \to r \in \mathit{lfp}(U_{\mathcal{R},R})$ with $l\theta_1\theta_2 = r\theta_1$ for some substitutions θ_1 and θ_2. As $l \to r \in \mathit{lfp}(U_{\mathcal{R},R})$, by Theorem 6.12 $l \to r \in \mathit{instances}(\gamma(\mathit{lfp}(\hat{U}_{\mathcal{R},R})))$ *i.e.*

$$l \to r = (\hat{l} \to \hat{r})\hat{\sigma}\theta$$

for $\hat{l} \to \hat{r} \in \mathit{lfp}(\hat{U}_{\mathcal{R},R})$ and some substitutions $\hat{\sigma}$ and θ. Then, we have

$$\hat{l}(\hat{\sigma}\theta\theta_1)\theta_2 = (\hat{l}\hat{\sigma}\theta)\theta_1\theta_2 = l\theta_1\theta_2 = r\theta_1 = (\hat{r}\hat{\sigma}\theta)\theta_1 = \hat{r}(\hat{\sigma}\theta\theta_1)$$

i.e. $\hat{l}$ semi-unifies with $\hat{r}$. □

10.14 Proof of Theorem 6.15

The proof follows from the lemmas below.

Lemma 10.18 *For all* $n \in \mathbb{N}$, $(\hat{U}_{\mathcal{R},R} \uparrow n)(\varnothing) \subseteq \nabla_{\mathcal{R}}(l) \times \Delta_{\mathcal{R}}(r)$.

Proof. By induction on n using Definitions 6.8 and Definition 6.14. □

Lemma 10.19 $\nabla_{\mathcal{R}}(l) \times \Delta_{\mathcal{R}}(r) \subseteq \mathit{lfp}(\hat{U}_{\mathcal{R},R})$.

Proof. Let $l' \to r' \in \nabla_{\mathcal{R}}(l) \times \Delta_{\mathcal{R}}(r)$. Then, there exist $n, m \in \mathbb{N}$ such that $l' \in (A_{\mathcal{R}} \uparrow n)(\{l\})$ and $r' \in (D_{\mathcal{R}} \uparrow m)(\{r\})$. In Definition 6.8 and Definition 6.14, we do not use the mgu's and the selected element R' of $\mathcal{R}$ to compute the result

(we just need to know that some terms unify and we only consider the root of the left-hand or right-hand side of R'). Therefore, $l' \to r' \in (\hat{F}_{\mathcal{R}} \uparrow m)((\hat{B}_{\mathcal{R}} \uparrow n)(\{R\}))$. Notice that we have $(\hat{F}_{\mathcal{R}} \uparrow m)((\hat{B}_{\mathcal{R}} \uparrow n)(\{R\})) \subseteq \mathit{lfp}(\hat{U}_{\mathcal{R},R})$. So, $l' \to r' \in \mathit{lfp}(\hat{U}_{\mathcal{R},R})$. $\square$