Guided Unfoldings for Finding Loops in Standard Term Rewriting

Etienne Payet

Guided Unfoldings for Finding Loops in Standard Term Rewriting [Extended Abstract]

Étienne Payet

LIM, Université de La Réunion, France
etienne.payet@univ-reunion.fr

Abstract. In this paper, we reconsider the unfolding-based technique that we have introduced previously for detecting loops in standard term rewriting. We improve it by *guiding* the unfolding process, using distinguished positions in the rewrite rules. This results in a depth-first computation of the unfoldings, whereas the original technique was breadth-first. We have implemented this new approach in our tool NTI and compared it to the previous one on a bunch of rewrite systems. The results we get are promising (better times, more successful proofs).

Keywords: term rewrite systems, dependency pairs, non-termination, loop, unfolding

1 Introduction

In [8], we have introduced a technique for finding *loops* (a periodic, special form of non-termination) in standard term rewriting. It consists of unfolding the term rewrite system (TRS) $\mathcal{R}$ under analysis and of performing a semi-unification [7] test on the unfolded rules for detecting loops. The unfolding operator $U_{\mathcal{R}}$ which is applied processes both forwards and backwards and considers *every* subterm of the rules to unfold, including variable subterms.

Example 1. Let $\mathcal{R}$ be the TRS consisting of the following rules (x is a variable):

$$R_1 = \underbrace{\mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), x)}_{l} \rightarrow \underbrace{\mathsf{f}(x, x, x)}_{r} \qquad R_2 = \mathsf{h} \rightarrow \mathsf{0} \qquad R_3 = \mathsf{h} \rightarrow \mathsf{1} \; .$$

Note that $\mathcal{R}$ is a variation of a well-known example by Toyama [11]. Unfolding the subterm $\mathsf{0}$ of l backwards with the rule R_2, we get the unfolded rule $U_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{1}), x) \rightarrow \mathsf{f}(x, x, x)$. Unfolding the subterm x (a variable) of l backwards with R_2, we get $U_2 = \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), \mathsf{h}) \rightarrow \mathsf{f}(\mathsf{0}, \mathsf{0}, \mathsf{0})$. Unfolding the first (from the left) occurrence of x in r forwards with R_2, we get $U_3 = \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), \mathsf{h}) \rightarrow \mathsf{f}(\mathsf{0}, \mathsf{h}, \mathsf{h})$. We have $\{U_1, U_2, U_3\} \subseteq U_{\mathcal{R}}(\mathcal{R})$. Now, if we unfold the subterm $\mathsf{1}$ of U_1 backwards with R_3, we get $\mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), x) \rightarrow \mathsf{f}(x, x, x)$, which is an element of $U_{\mathcal{R}}(U_{\mathcal{R}}(\mathcal{R}))$.

The left-hand side l_1 of this rule semi-unifies with its right-hand side r_1 *i.e.*, $l_1\theta_1\theta_2 = r_1\theta_1$ for the substitutions $\theta_1 = \{x/\mathsf{s}(\mathsf{h})\}$ and $\theta_2 = \{\}$. Therefore, $l\theta_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))$ loops with respect to $\mathcal{R}$ because it can be rewritten to itself using the rules of $\mathcal{R}$:

$$\mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \underset{R_2}{\rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \underset{R_3}{\rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), \mathsf{s}(\mathsf{h})) \underset{R_1}{\rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \ .$$

Iterative applications of the operator $U_{\mathcal{R}}$ result in a combinatorial explosion which significatively limits the approach. In order to reduce it, a mechanism is introduced in [8] for eliminating the unfolded rules which are estimated as *useless* for detecting loops. Moreover, in practice, three analyses are run in parallel (in different threads): one with forward unfoldings only, one with backward unfoldings only and one with forward and backward unfoldings together.

So, the technique of [8] roughly consists in computing *all* the rules of $U_{\mathcal{R}}(\mathcal{R})$, $U_{\mathcal{R}}(U_{\mathcal{R}}(\mathcal{R}))$, ... and removing the useless ones, until the semi-unification test succeeds on an unfolded rule or a time limit is reached. Therefore, this approach corresponds to a *breadth-first* search for a loop, as the successive iterations of $U_{\mathcal{R}}$ are computed thoroughly, one after the other. However, it is not always necessary to compute all the elements of each iteration of $U_{\mathcal{R}}$. For instance, in Ex. 1 above, U_2 and U_3 do not lead to an unfolded rule satisfying the semi-unification criterion. This is detected by the eliminating mechanism of [8], but only *after* these two rules are generated. In order to *avoid* the generation of these useless rules, one can notice that $\langle \mathsf{s}(\mathsf{0}), x\rangle$ is the leftmost *disagreement pair* of l and r *i.e.*, intuitively, it is the first pair of different subterms that occur when reading both l and r from left to right. Hence, one can first concentrate on resolving this disagreement, unfolding this pair only, and then, once this is resolved, apply the same process to the next disagreement pair.

Example 2 (Ex. 1 continued). $\langle \mathsf{s}(\mathsf{0}), x\rangle$ is the leftmost disagreement pair of l and r. There are two ways to resolve it (*i.e.*, make it disappear).

The first way consists in unifying $\mathsf{s}(\mathsf{0})$ and x, *i.e.*, in computing $R_1\theta$ where θ is the substitution $\{x/\mathsf{s}(\mathsf{0})\}$, which gives $U_0 = \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), \mathsf{s}(\mathsf{0})) \rightarrow \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))$.

The other way is to unfold $\mathsf{s}(\mathsf{0})$ or x. We decide not to unfold variable subterms, hence we select $\mathsf{s}(\mathsf{0})$. As it occurs in the left-hand side of R_1, we unfold it backwards. The only possibility is to use R_2, which results in

$$U_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{1}), x) \rightarrow \mathsf{f}(x, x, x) \ .$$

Note that this approach only generates two rules (U_0 and U_1) at the first iteration of the unfolding operator. In comparison, the approach of [8] produces 14 rules, as all the subterms of R_1 are considered for unfolding.

Hence, the disagreement pair $\langle \mathsf{s}(\mathsf{0}), x\rangle$ has been replaced with the disagreement pair $\langle \mathsf{s}(\mathsf{h}), x\rangle$. Unifying $\mathsf{s}(\mathsf{h})$ and x *i.e.*, computing $U_1\theta'$ where θ' is the substitution $\{x/\mathsf{s}(\mathsf{h})\}$, we get $U_1' = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{1}), \mathsf{s}(\mathsf{h})) \rightarrow \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))$. So, the disagreement $\langle \mathsf{s}(\mathsf{0}), x\rangle$ is solved: it has been replaced with $\langle \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})\rangle$. Now, the leftmost disagreement pair in U_1' is $\langle \mathsf{1}, \mathsf{h}\rangle$ (here we mean the second occurrence of h in the right-hand side of U_1'). Unfolding $\mathsf{1}$ backwards with R_3, we get $V_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \to \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))$ and unfolding h forwards with R_3, we get $V_1' = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(1), \mathsf{s}(\mathsf{h})) \to \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(1), \mathsf{s}(\mathsf{h}))$. The semi-unification test succeeds on both rules: V_1 yields the looping term $\mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))$ and V_1' yields $\mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(1), \mathsf{s}(\mathsf{h}))$.

The approach which is sketched in Ex. 2 corresponds to a *depth-first* search for a loop. The iterations of $U_{\mathcal{R}}$ are not thoroughly computed. Only a selected disagreement pair is considered and once it is resolved we backtrack to the next one. Hence, the unfoldings are *guided* by disagreement pairs. In this paper, we formally describe the intuitions presented above (Sect. 3 and Sect. 4) and we report some experiments on a bunch of rewrite systems from the TPBD [10] (Sect 5). The results we get are promising and we do not need to perform several analyses in parallel, nor to unfold variable subterms, unlike with the approach of [8].

2 Preliminaries

We refer to [4] for the basics of rewriting. From now on, we fix a finite *signature* $\mathcal{F}$ together with an infinite countable set $\mathcal{V}$ of *variables* with $\mathcal{F} \cap \mathcal{V} = \emptyset$. Elements of $\mathcal{F}$ are denoted by $\mathsf{f}, \mathsf{g}, \mathsf{h}, 0, 1, \dots$ and elements of $\mathcal{V}$ by $x, y, z, \dots$ The set of terms over $\mathcal{F} \cup \mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. For any $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we let $root(t)$ denote the root symbol of t: $root(t) = \mathsf{f}$ if $t = \mathsf{f}(t_1, \dots, t_m)$ and $root(t) = \bot$ if $t \in \mathcal{V}$. Moreover, we let $Var(t)$ denote the set of variables occurring in t and $Pos(t)$ denote the set of positions of t. For any $p \in Pos(t)$, we write $t|_p$ to denote the subterm of t at position p and we write $t[p \leftarrow s]$ to denote the term obtained from t by replacing $t|_p$ with a term s. For any $p, q \in Pos(t)$, we write $p \le q$ if and only if p is a prefix of q; we write $p < q$ if and only if $p \le q$ and $p \ne q$. We also define the set of non-variable positions which either are a prefix of p or include p as a prefix:

$$NPos(t, p) = \{q \in Pos(t) \mid q \le p \vee p \le q,\ t|_q \notin \mathcal{V}\} \ .$$

For any non-empty set of positions S, we let $\min S$ denote the position in S which is leftmost and downmost (for instance, $\min\{1, 2, 1.2, 1.3, 2.1\} = 1.2$). We let $\min \emptyset$ be undefined.

We write substitutions as sets of the form $\{x_1/t_1, \dots, x_n/t_n\}$ denoting that for each $1 \le i \le n$, variable x_i is mapped to term t_i (note that x_i may occur in t_i). The empty substitution (identity) is denoted by id. The application of a substitution θ to a syntactic object o is denoted by $o\theta$. We let $mgu(s, t)$ denote the set of most general unifiers of terms s and t. A *disagreement pair* of s and t is an ordered pair $\langle s|_p, t|_p \rangle$ where $p \in Pos(s) \cap Pos(t)$, $root(s|_p) \ne root(t|_p)$ and, for every $q < p$, $root(s|_q) = root(t|_q)$.

Example 3. Let $s = \mathsf{f}(\mathsf{s}(0), \mathsf{s}(1), y)$, $t = \mathsf{f}(x, x, x)$, $p_1 = 1$, $p_2 = 2$ and $p_3 = 3$. Then, $\langle s|_{p_1}, t|_{p_1} \rangle = \langle \mathsf{s}(0), x \rangle$ and $\langle s|_{p_2}, t|_{p_2} \rangle = \langle \mathsf{s}(1), x \rangle$ are disagreement pairs of s and t. However, $\langle s|_{p_3}, t|_{p_3} \rangle = \langle y, x \rangle$ is not a disagreement pair of s and t because $root(y) = root(x) = \bot$.

A *rewrite rule* (or *rule*) over $\mathcal{F} \cup \mathcal{V}$ has the form $l \to r$ with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $l \notin \mathcal{V}$ 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 $\mathcal{F} \cup \mathcal{V}$. Given a TRS $\mathcal{R}$ and some terms s and t, we write $s \xrightarrow[\mathcal{R}]{} t$ if there is a rewrite rule $l \to r$ in $\mathcal{R}$, a substitution θ and $p \in Pos(s)$ such that $s|_p = l\theta$ and $t = s[p \leftarrow r\theta]$. We let $\xrightarrow[\mathcal{R}]{+}$ (resp. $\xrightarrow[\mathcal{R}]{*}$) denote the transitive (resp. reflexive and transitive) closure of $\xrightarrow[\mathcal{R}]{}$. 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 $t \xrightarrow[\mathcal{R}]{} t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \cdots$. We say that $\mathcal{R}$ is *non-terminating* if there exists a non-terminating term *w.r.t.* it. A term t *loops w.r.t.* $\mathcal{R}$ when $t \xrightarrow[\mathcal{R}]{+} C[t\theta]$ for some context C and substitution θ. Then $t \xrightarrow[\mathcal{R}]{+} C[t\theta]$ is called a *loop* for $\mathcal{R}$. We say that $\mathcal{R}$ is *looping* when it admits a loop. If a term loops *w.r.t.* $\mathcal{R}$ then it is non-terminating *w.r.t.* $\mathcal{R}$.

We refer to [3] for details on dependency pairs. The *defined symbols* of a TRS $\mathcal{R}$ over $\mathcal{F} \cup \mathcal{V}$ are $\mathcal{D}_{\mathcal{R}} = \{root(l) \mid l \to r \in \mathcal{R}\}$. For every $\mathsf{f} \in \mathcal{F}$ we let $\mathsf{f}^\#$ be a fresh *tuple symbol* with the same arity as f. The set of tuple symbols is denoted as $\mathcal{F}^\#$. The notations and definitions above with terms over $\mathcal{F} \cup \mathcal{V}$ are naturally extended to terms over $(\mathcal{F} \cup \mathcal{F}^\#) \cup \mathcal{V}$. Elements of $\mathcal{F} \cup \mathcal{F}^\#$ are denoted as $f, g, \ldots$ If $t = \mathsf{f}(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we let $t^\#$ denote the term $\mathsf{f}^\#(t_1, \ldots, t_m)$, and we call $t^\#$ an $\mathcal{F}^\#$*-term*. An $\mathcal{F}^\#$*-rule* is a rule whose left-hand and right-hand sides are $\mathcal{F}^\#$-terms. The set of *dependency pairs* of $\mathcal{R}$ is

$$\{l^\# \to t^\# \mid l \to r \in \mathcal{R},\ t \text{ is a subterm of } r,\ root(t) \in \mathcal{D}_{\mathcal{R}}\}\ .$$

A sequence $s_1 \to t_1, \ldots, s_n \to t_n$ of dependency pairs of $\mathcal{R}$ is an $\mathcal{R}$*-chain* if there exists a substitution σ such that $t_i\sigma \xrightarrow[\mathcal{R}]{*} s_{i+1}\sigma$ holds for every two consecutive pairs $s_i \to t_i$ and $s_{i+1} \to t_{i+1}$ in the sequence.

Theorem 1 ([3]). *$\mathcal{R}$ is non-terminating if and only if there exists an infinite $\mathcal{R}$-chain.*

The *dependency graph* of $\mathcal{R}$ is the graph whose nodes are the dependency pairs of $\mathcal{R}$ and there is an arc from $s \to t$ to $u \to v$ if and only if $s \to t, u \to v$ is an $\mathcal{R}$-chain. This graph is not computable in general since it is undecidable whether two dependency pairs of $\mathcal{R}$ form an $\mathcal{R}$-chain. Hence, for automation, one constructs an estimated graph containing all the arcs of the real graph. This is done by computing *connectable terms*, which form a superset of those terms s, t where $s\sigma \xrightarrow[\mathcal{R}]{*} t\sigma$ holds for some substitution σ. The approximation uses the transformations CAP and REN where, for any $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}^\#, \mathcal{V})$, CAP($t$) (resp. REN($t$)) results from replacing all subterms of t with defined root symbol (resp.

all variables in t) by different new variables not previously met. More formally:

$$\begin{aligned}
\text{CAP}(x) &= x \text{ if } x \in \mathcal{V} \\
\text{CAP}(f(t_1,\ldots,t_m)) &= \begin{cases} \text{a new variable} & \text{if } f \in \mathcal{D}_{\mathcal{R}} \\ f(\text{CAP}(t_1),\ldots,\text{CAP}(t_m)) & \text{if } f \notin \mathcal{D}_{\mathcal{R}} \end{cases} \\
\text{REN}(x) &= \text{a new variable if } x \in \mathcal{V} \\
\text{REN}(f(t_1,\ldots,t_m)) &= f(\text{REN}(t_1),\ldots,\text{REN}(t_m))
\end{aligned}$$

A term s is *connectable* to a term t if $\text{REN}(\text{CAP}(s))$ unifies with t. An $\mathcal{F}^{\#}$-rule $l \to r$ is connectable to an $\mathcal{F}^{\#}$-rule $s \to t$ if r is connectable to s. The *estimated dependency graph* of $\mathcal{R}$ is denoted as $DG(\mathcal{R})$. Its nodes are the dependency pairs of $\mathcal{R}$ and there is an arc from N to N' iff N is connectable to N'.

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

$$\begin{aligned}
(Y \uparrow 0)(e) &= e \\
(Y \uparrow n+1)(e) &= Y\big((Y \uparrow n)(e)\big) \quad \forall n \in \mathbb{N} .
\end{aligned}$$

Finite sequences are written as $[e_1,\ldots,e_n]$. We let :: denote the concatenation operator over finite sequences. A *path* in $DG(\mathcal{R})$ is a finite sequence $[N_1, N_2, \ldots, N_n]$ of nodes where, for each $1 \le i < n$, there is an arc from N_i to N_{i+1}. When there is also an arc from N_n to N_1, the path is called a *cycle*. It is called a *simple cycle* if, moreover, there is no repetition of nodes (modulo variable renaming). We let $SCC(\mathcal{R})$ denote the set of strongly connected components of $DG(\mathcal{R})$ that contain at least one arc. Hence, a strongly connected component consisting of a unique node is in $SCC(\mathcal{R})$ only if there is an arc from the node to itself.

Example 4. Let $\mathcal{R}$ be the TRS of Ex. 1. We have $SCC(\mathcal{R}) = \{\mathcal{C}\}$ where $\mathcal{C}$ consists of the node $N = \mathsf{f}^{\#}(\mathsf{s}(0), \mathsf{s}(1), x) \to \mathsf{f}^{\#}(x, x, x)$ and of the arc (N, N).

Example 5. Let $\mathcal{R}' = \{\mathsf{f}(0) \to \mathsf{f}(1),\ \mathsf{f}(2) \to \mathsf{f}(0),\ 1 \to 0\}$. We have $SCC(\mathcal{R}') = \{\mathcal{C}'\}$ where $\mathcal{C}'$ consists of the nodes $N_1 = \mathsf{f}^{\#}(0) \to \mathsf{f}^{\#}(1)$ and $N_2 = \mathsf{f}^{\#}(2) \to \mathsf{f}^{\#}(0)$ and of the arcs $\{N_1, N_2\} \times \{N_1, N_2\} \setminus \{(N_2, N_2)\}$. The strongly connected component of $DG(\mathcal{R}')$ which consists of the unique node $\mathsf{f}^{\#}(0) \to 1^{\#}$ does not belong to $SCC(\mathcal{R}')$ because it has no arc.

3 Guided unfoldings

In the sequel of this paper, we let $\mathcal{R}$ denote a TRS over $\mathcal{F} \cup \mathcal{V}$.

While the method sketched in Ex. 2 can be applied directly to the TRS $\mathcal{R}$ under analysis, we use a refinement based on the estimated dependency graph of $\mathcal{R}$. The cycles in $DG(\mathcal{R})$ are over-approximations of the infinite $\mathcal{R}$-chains *i.e.*, any infinite $\mathcal{R}$-chain corresponds to a cycle in the graph but some cycles in the graph may not correspond to any $\mathcal{R}$-chain. Moreover, by Theorem 1, if we find an infinite $\mathcal{R}$-chain then we have proved that $\mathcal{R}$ is non-terminating. Hence, we concentrate on the cycles in $DG(\mathcal{R})$. We try to *solve* them *i.e.*, to find out if they correspond to any infinite $\mathcal{R}$-chain. This is done by iteratively unfolding the $\mathcal{F}^{\#}$-rules of the cycles. If the semi-unification test succeeds on one of the generated unfolded rules, then we have found a loop.

Definition 1 (Syntactic loop). *A* syntactic loop *in $\mathcal{R}$ is a finite sequence $[N_1, \dots, N_n]$ of distinct (modulo variable renaming) $\mathcal{F}^{\#}$-rules where, for each $1 \le i < n$, N_i is connectable to N_{i+1} and N_n is connectable to N_1. We identify syntactic loops consisting of the same (modulo variable renaming) elements, not necessarily in the same order.*

Note that the simple cycles in $DG(\mathcal{R})$ are syntactic loops. For any $\mathcal{C} \in SCC(\mathcal{R})$, we let $s\text{-}cycles(\mathcal{C})$ denote the set of simple cycles in $\mathcal{C}$. We also let

$$s\text{-}cycles(\mathcal{R}) = \cup_{\mathcal{C} \in SCC(\mathcal{R})}\, s\text{-}cycles(\mathcal{C})$$

be the set of simple cycles in $\mathcal{R}$. The rules of any simple cycle in $\mathcal{R}$ are assumed to be pairwise variable disjoint.

Example 6 (Ex. 4 and 5 continued). We have

$$s\text{-}cycles(\mathcal{R}) = \{[N]\} \quad \text{and} \quad s\text{-}cycles(\mathcal{R}') = \{[N_1], [N_1, N_2]\}$$

with, in $s\text{-}cycles(\mathcal{R}')$, $[N_1, N_2] = [N_2, N_1]$.

The operators we use for unfolding an $\mathcal{F}^{\#}$-rule are defined as follows. They only unfold non-variable subterms. Moreover, they use narrowing, see (2) in Def. 2-3: there, $l' \to r' \ll \mathcal{R}$ means that $l' \to r'$ is a new occurrence of a rule of $\mathcal{R}$ that is renamed apart *i.e.*, contains new variables not previously met.

Definition 2 (Forward guided unfoldings). *Let $l \to r$ be an $\mathcal{F}^{\#}$-rule, s be an $\mathcal{F}^{\#}$-term and p be the position of a disagreement pair of r and s. The forward unfoldings of $l \to r$ at position p, guided by s and w.r.t. $\mathcal{R}$ are*

$$F_{\mathcal{R}}(l \to r, s, p) = \left\{ U \;\middle|\; \begin{array}{l} q \in NPos(r,p),\ q \le p \\ \theta \in mgu(r|_q, s|_q),\ U = (l \to r)\theta \end{array} \right\}^{(1)} \cup \left\{ U \;\middle|\; \begin{array}{l} q \in NPos(r,p),\ l' \to r' \ll \mathcal{R} \\ \theta \in mgu(r|_q, l'),\ U = (l \to r[q \leftarrow r'])\theta \end{array} \right\}^{(2)} .$$

Definition 3 (Backward guided unfoldings). *Let $s \to t$ be an $\mathcal{F}^{\#}$-rule, r be an $\mathcal{F}^{\#}$-term and p be the position of a disagreement pair of r and s. The backward unfoldings of $s \to t$ at position p, guided by r and w.r.t. $\mathcal{R}$ are*

$$B_{\mathcal{R}}(s \to t, r, p) = \left\{ U \;\middle|\; \begin{array}{l} q \in NPos(s,p),\ q \le p \\ \theta \in mgu(r|_q, s|_q),\ U = (s \to t)\theta \end{array} \right\}^{(1)} \cup \left\{ U \;\middle|\; \begin{array}{l} q \in NPos(s,p),\ l' \to r' \ll \mathcal{R} \\ \theta \in mgu(s|_q, r'),\ U = (s[q \leftarrow l'] \to t)\theta \end{array} \right\}^{(2)} .$$

Example 7 (Ex. 4 and 6 continued). $[N]$ is a simple cycle in $\mathcal{R}$ with

$$N = \underbrace{\mathsf{f}^{\#}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), x)}_{s} \rightarrow \underbrace{\mathsf{f}^{\#}(x, x, x)}_{t} \ .$$

Let $r = t$. Then $p = 1$ is a disagreement pair position of r and s. Moreover, $q = 1.1 \in NPos(s, p)$ because $p \leq q$ and $s|_q = \mathsf{0}$ is not a variable. Let $l' \rightarrow r' = \mathsf{h} \rightarrow \mathsf{0} \in \mathcal{R}$. We have $id \in mgu(s|_q, r')$. Hence, by (2) in Def. 3, we have

$$U_1 = \underbrace{\mathsf{f}^{\#}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{1}), x)}_{s_1} \rightarrow \underbrace{\mathsf{f}^{\#}(x, x, x)}_{t_1} \in B_{\mathcal{R}}(N, r, p) \ .$$

Let $r_1 = t_1$. Then, $p = 1$ is a disagreement pair position of r_1 and s_1. Moreover, $p \in NPos(s_1, p)$ with $s_1|_p = \mathsf{s}(\mathsf{h})$, $p \leq p$ and $r_1|_p = x$. As $\{x/\mathsf{s}(\mathsf{h})\} \in mgu(r_1|_p, s_1|_p)$, by (1) in Def. 3 we have

$$U_1' = \underbrace{\mathsf{f}^{\#}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{1}), \mathsf{s}(\mathsf{h}))}_{s_1'} \rightarrow \underbrace{\mathsf{f}^{\#}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))}_{t_1'} \in B_{\mathcal{R}}(U_1, r_1, p) \ .$$

Let $r_1' = t_1'$. Then, $p' = 2.1$ is a disagreement pair position of r_1' and s_1' with $p' \in NPos(s_1', p')$. Let $l'' \rightarrow r'' = \mathsf{h} \rightarrow \mathsf{1} \in \mathcal{R}$. We have $id \in mgu(s_1'|_{p'}, r'')$. Hence, by (2) in Def. 3, we have

$$U_1^{''} = \mathsf{f}^{\#}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \rightarrow \mathsf{f}^{\#}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \in B_{\mathcal{R}}(U_1', r_1', p') \ .$$

We choose to guide the unfoldings using the leftmost disagreement pair of the left-hand and right-hand sides of rules.

Definition 4 (Disagreement). *The* minimal disagreement position *of terms s and t is denoted as $minpos(s, t)$. It is defined as*

$$minpos(s, t) = \min \left\{ p \;\middle|\; \begin{array}{l} p \in Pos(s) \cap Pos(t) \\ \langle s|_p, t|_p \rangle \text{ is a disagreement pair of } s \text{ and } t \end{array} \right\} \ .$$

So, $minpos(s, t)$ is undefined if there is no disagreement pair of s and t.

Example 8. We have $minpos(\mathsf{f}^{\#}(x, x, x), \mathsf{f}^{\#}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), x)) = 1$ because

$$\langle \mathsf{f}^{\#}(x, x, x)|_1, \ \mathsf{f}^{\#}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), x)|_1 \rangle = \langle x, \mathsf{s}(\mathsf{0}) \rangle$$

is the leftmost disagreement pair of the terms $\mathsf{f}^{\#}(x, x, x)$ and $\mathsf{f}^{\#}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{1}), x)$.

Our approach consists of iteratively unfolding syntactic loops using the following operator.

Definition 5 (Guided unfoldings). *Let X be a set of syntactic loops of $\mathcal{R}$. The* guided unfoldings *of X w.r.t. $\mathcal{R}$ are defined as*

$$GU_{\mathcal{R}}(X) = \left\{ [U] :: L \;\middle|\; \begin{array}{l} [l \to r, s \to t] :: L \in X,\ \theta \in mgu(r,s) \\ U = (l \to t)\theta,\ [U] :: L \textit{ is a syntactic loop} \end{array} \right\}^{(1)} \cup$$

$$\left\{ [U, s \to t] :: L \;\middle|\; \begin{array}{l} [l \to r, s \to t] :: L \in X,\ mgu(r,s) = \emptyset \\ p = minpos(r,s),\ U \in F_{\mathcal{R}}(l \to r, s, p) \\ [U, s \to t] :: L \textit{ is a syntactic loop} \end{array} \right\}^{(2)} \cup$$

$$\left\{ [l \to r, U] :: L \;\middle|\; \begin{array}{l} [l \to r, s \to t] :: L \in X,\ mgu(r,s) = \emptyset \\ p = minpos(r,s),\ U \in B_{\mathcal{R}}(s \to t, r, p) \\ [l \to r, U] :: L \textit{ is a syntactic loop} \end{array} \right\}^{(3)} \cup$$

$$\left\{ [U] \;\middle|\; \begin{array}{l} [l \to r] \in X,\ p = minpos(r,l) \\ U \in F_{\mathcal{R}}(l \to r, l, p) \cup B_{\mathcal{R}}(l \to r, r, p) \\ [U] \textit{ is a syntactic loop} \end{array} \right\}^{(4)} .$$

So, the idea is to walk through the syntactic loops, from the first rule on the left to the last rule on the right[1]. Whenever the right-hand side of the first rule unifies with the left-hand side of the second rule, then the first and second rules are *merged* (case (1) in Def. 5), meaning that we succeeded in passing the first rule and in reaching (connecting to) the second one. When the right-hand side of the first rule does not unify with the left-hand side of the second rule, then we cannot connect the first rule to the second one yet. We use the operators $F_{\mathcal{R}}$ and $B_{\mathcal{R}}$ to try to connect to the second rule (cases (2) and (3) in Def. 5). Once we have reached the last rule of a syntactic loop, then we have computed a *compressed* form of the loop. We keep on unfolding this compressed form (case (4) in Def. 5), which corresponds to a walk through the entire loop, forwards or backwards, in one go. Note that after unfolding a rule, we might get a sequence which is not a syntactic loop: the newly generated rule might be identical to another rule in the sequence or it might not be connectable to its predecessor or successor in the sequence. Therefore, (1)–(4) in Def. 5 require that the generated sequence is a syntactic loop.

The guided unfolding semantics is defined as follows, in the style of [1,8].

Definition 6 (Guided unfolding semantics). *The* guided unfolding semantics *of $\mathcal{R}$ is the limit of the unfolding process described in Def. 5, starting from the simple cycles in $\mathcal{R}$:*

$$gunf(\mathcal{R}) = \bigcup_{n \in \mathbb{N}} (GU_{\mathcal{R}} \uparrow n)(s\text{-}cycles(\mathcal{R}))\ .$$

Example 9. By Ex. 7 and (4) in Def. 5, we have $[U_1''] \in (GU_{\mathcal{R}} \uparrow 3)(s\text{-}cycles(\mathcal{R}))$ hence $[U_1''] \in gunf(\mathcal{R})$.

[1] Going from left to right is an arbitrary (although natural) choice here. There might exist clever strategies, guided for instance by the form of the disagreement pairs, but we have not investigated this for the moment.

Example 10. Let $\mathcal{R} = \{\mathsf{f}(0) \to \mathsf{g}(1),\ \mathsf{g}(1) \to \mathsf{f}(0)\}$. Then, $SCC(\mathcal{R}) = \{\mathcal{C}\}$ where $\mathcal{C}$ consists of the nodes $N_1 = \mathsf{f}^\#(0) \to \mathsf{g}^\#(1)$ and $N_2 = \mathsf{g}^\#(1) \to \mathsf{f}^\#(0)$ and of the arcs (N_1, N_2) and (N_2, N_1). Moreover, $s\text{-}cycles(\mathcal{R}) = \{[N_1, N_2]\}$. As $id \in mgu(\mathsf{g}^\#(1), \mathsf{g}^\#(1))$ and $(\mathsf{f}^\#(0) \to \mathsf{f}^\#(0))id = \mathsf{f}^\#(0) \to \mathsf{f}^\#(0)$, by (1) in Def. 5 we have $[\mathsf{f}^\#(0) \to \mathsf{f}^\#(0)] \in (GU_{\mathcal{R}} \uparrow 1)(s\text{-}cycles(\mathcal{R}))$ so $[\mathsf{f}^\#(0) \to \mathsf{f}^\#(0)] \in gunf(\mathcal{R})$.

Proposition 1. *For any $n \in \mathbb{N}$ and $[s^\# \to t^\#] \in \bigcup_{k \leq n} (GU_{\mathcal{R}} \uparrow k)(s\text{-}cycles(\mathcal{R}))$ there exists some context C such that $s \xrightarrow[\mathcal{R}]{+} C[t]$.*

Proof. For some context C, we have $s \to C[t] \in \bigcup_{k \leq n} (U_{\mathcal{R}} \uparrow k)(\mathcal{R})$ where $U_{\mathcal{R}}$ is the unfolding operator defined in [8]. Hence, by Prop. 3.12 of [8], we have $s \xrightarrow[\mathcal{R}]{+} C[t]$.

4 Inferring terms that loop

As in [8], we use semi-unification [7] for detecting loops. A polynomial-time algorithm for semi-unification can be found in [6].

Theorem 2. *If for $[s^\# \to t^\#] \in gunf(\mathcal{R})$ there exist some substitutions θ_1 and θ_2 such that $s\theta_1\theta_2 = t\theta_1$, then the term $s\theta_1$ loops w.r.t. $\mathcal{R}$.*

Proof. By Prop. 1, $s \xrightarrow[\mathcal{R}]{+} C[t]$ for some context C. Since $\xrightarrow[\mathcal{R}]{}$ is stable, we have

$$s\theta_1 \xrightarrow[\mathcal{R}]{+} C[t]\theta_1 \quad i.e., \quad s\theta_1 \xrightarrow[\mathcal{R}]{+} C\theta_1[t\theta_1] \quad i.e., \quad s\theta_1 \xrightarrow[\mathcal{R}]{+} C\theta_1[s\theta_1\theta_2]\ .$$

Hence, $s\theta_1$ loops *w.r.t.* $\mathcal{R}$.

Example 11 (Ex. 9 continued). We have

$$\underbrace{[\mathsf{f}^\#(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h})) \to \mathsf{f}^\#(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))]}_{U_1''} \in gunf(\mathcal{R})$$

with $\mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))\theta_1\theta_2 = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))\theta_1$ for $\theta_1 = \theta_2 = id$. Consequently, $\mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))\theta_1 = \mathsf{f}(\mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}), \mathsf{s}(\mathsf{h}))$ loops *w.r.t.* $\mathcal{R}$.

Example 12 (Ex. 10 continued). $[\mathsf{f}^\#(0) \to \mathsf{f}^\#(0)] \in gunf(\mathcal{R})$ with $\mathsf{f}(0)\theta_1\theta_2 = \mathsf{f}(0)\theta_1$ for $\theta_1 = \theta_2 = id$. Hence, $\mathsf{f}(0)\theta_1 = \mathsf{f}(0)$ loops *w.r.t.* $\mathcal{R}$.

5 Experiments

We have implemented the technique of this paper in our analyser NTI[2] (Non-Termination Inference) and we have run it on a set of selected rewrite systems built as follows. We have extracted from the directory `TRS_Standard` of the

[2] `http://lim.univ-reunion.fr/staff/epayet/Research/NTI/NTI.html`

TPBD [10] all the valid rewrite systems[3] that are proved looping by AProVE [2,5]. We ended up with a set of 171 rewrite systems, some characteristics of which are reported in Table 1. Note that the complete set of simple cycles of a TRS may be really huge, hence NTI only computes a subset of it. The simple cycle characteristics reported in Table 1 relate to the subsets computed by NTI.

	Min	Max	Average
TRS size	1 [17]	104 [1]	10.98
Number of SCCs	1 [100]	12 [1]	1.94
SCC size	1 [95]	192 [1]	4.47
Number of simple cycles	1 [47]	185 [1]	8.54
Simple cycle size	1 [156]	9 [2]	2.25
Number of function symbols	1 [4]	66 [1]	9.01
Function symbol arity	0 [151]	5 [2]	1.07
Number of defined function symbols	1 [28]	58 [1]	5.16
Defined function symbol arity	0 [73]	5 [2]	1.38

Table 1. Some characteristics of the 171 analysed TRSs. Sizes are in number of rules. In square brackets, we report the number of TRSs with the corresponding min or max.

We have compared our new approach to that of [8], which is also implemented in NTI. The results are promising (see Table 2). We get a larger number of successful proofs with better times. However, the results regarding the number of generated unfolded rules are worse. This may come from the fact that in the new approach we did not implement any mechanism for eliminating useless unfolded rules (unlike in the approach of [8]). Another point to note is that the implementation of the new approach does not unfold variable subterms (in compliance with Def. 2 and Def. 3) and does not perform several analyses in parallel, unlike the implementation of [8] which unfolds variable subterms and performs three analyses in parallel (one with forward unfoldings only, one with backward unfoldings only and one with forward and backward unfoldings together).

AProVE is able to prove loopingness of all the 171 rewrite systems of our set. In comparison, our approach succeeds on 152 systems only. Similarly to our approach, AProVE handles the SCCs of the estimated dependency graph independently, but it performs both a termination and a non-termination analysis on each SCC. Hence, when an SCC is proved terminating, then its non-termination analysis is stopped, and vice-versa. On the contrary, NTI is a pure non-termination analyser *i.e.*, it only performs non-termination analyses. If an SCC is terminating, it cannot prove it and keeps on trying a non-termination proof, unnecessarily generating unfolded rules at the expense of the analysis of the other SCCs. Hence, in our opinion, a comparison of our approach with

[3] Surprisingly, the subdirectory `Transformed_CSR_04` contains 60 files where an invalid rule *i.e.*, a pair $l \to r$ with $Var(r) \not\subseteq Var(l)$, occurs.

	NTI'08	NTI'18
Success	150	152
Don't know	0	2
Time out	21	17
Total time	2862.34s	2144.09s
Total number of generated rules	10 845 546	11 219 422
Average time for a success	2.28s	0.51s
Average number of generated rules for a success	7206	8298

Table 2. Analysis results on our selected set of 171 rewrite systems. The time limit fixed for a proof is 120s. **Time out** corresponds to a situation where the computation did not stop within the time limit. **Don't know** corresponds to a situation where the computation stopped within the time limit with no positive answer (typically, this means that no more unfolded rule were generated at some point *i.e.*, $(GU_{\mathcal{R}} \uparrow n)(s\text{-}cycles(\mathcal{R})) = \emptyset$ for some n, and no generated rule led to success). NTI'08 refers to the technique of [8], NTI'18 to the technique presented in this paper. We used an Intel 2-core i5 at 2 GHz with 8 GB of RAM.

AProVE does not make sense (we do not know how to turn off the termination analyser of AProVE in order to only compare its non-termination analyser with ours).

6 Conclusion

We have reconsidered the unfolding-based technique introduced in [8] for detecting loops in standard term rewriting. We have improved it by guiding the unfoldings, using disagreement pairs. This results in a depth-first search for loops, whereas the technique of [8] is breadth-first. Another difference is that the new approach unfolds the dependency pairs, whereas [8] directly works with the rules of the TRS under analysis. Moreover, the new approach is modular, in the sense that it considers the SCCs of the estimated dependency graph independently; in [8], no SCC is computed.

We have implemented the new approach in our tool NTI and compared it to [8] on a set of 171 rewrite systems. The results we get are promising (better times, more successful proofs) but the number of generated rules is still too important (it is larger than with the approach of [8]). We plan to add an elimination mechanism to the new technique, similarly to [8], to address this problem. Another possibility that we are considering is to select the rules which are *usable* for unfolding an element of a syntactic loop; this would *avoid* the generation of useless rules, whereas an elimination mechanism would require to generate the rule first and then to eliminate it *afterwards*. It might also be interesting to investigate the idea of running an existing termination analyser on each SCC of the estimated dependency graph, either in parallel with NTI (once an analyser succeeds, the other is stopped) or sequentially (first run the termination analyser for a portion of the fixed time limit, then run NTI only on those SCCs whose termination could not be proved). Several efficient and powerful termination analysers have been implemented so far [9] and such a technique would avoid the generation of useless unfolded rules. A final idea to improve our approach would be to consider other depth-first strategies. In this paper, we proceed from left to right *i.e.*, we always select the leftmost disagreement pairs. Instead of keeping the strategy so fixed, one could guide it more efficiently using specific properties of the disagreement pairs, other than their position.

Acknowledgements

The author thanks the anonymous reviewers for their many helpful comments and constructive criticisms.

References

1. M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Safe folding/unfolding with conditional narrowing. In M. Hanus, J. Heering, and K. 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, 1997.
2. AProVE Web site. `http://aprove.informatik.rwth-aachen.de/`.
3. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
4. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
5. J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing program termination and complexity automatically with AProVE. *Journal of Automated Reasoning*, 58(1):3–31, 2017.
6. D. Kapur, D. Musser, P. Narendran, and J. Stillman. Semi-unification. *Theoretical Computer Science*, 81(2):169–187, 1991.
7. D.S. Lankford and D. R. Musser. A finite termination criterion. Unpublished Draft, USC Information Sciences Institute, Marina Del Rey, CA, 1978.
8. É. Payet. Loop detection in term rewriting using the eliminating unfoldings. *Theoretical Computer Science*, 403(2-3):307–327, 2008.
9. Termination Portal – An (incomplete) overview of existing tools for termination analysis. `http://termination-portal.org/wiki/Category:Tools`.
10. Termination Problems Data Base. `http://termination-portal.org/wiki/TPDB`.
11. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987.