Observing Loopingness

Etienne Payet

Observing Loopingness

Étienne Payet
LIM - Université de la Réunion, France

Abstract

In this paper, we consider non-termination in logic programming and in term rewriting and we recall some well-known results for observing it. Then, we instantiate these results to loopingness, a simple form of non-termination. We provide a bunch of examples that seem to indicate that the instantiations are correct as well as partial proofs.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Rewrite systems; Theory of computation → Program analysis

Keywords and phrases Logic Programming, Term Rewriting Systems, Non-Termination, Loop

1 Introduction

Proving non-termination is an important topic in logic programming and term rewriting. It is also important to determine classes of non-termination and compare them, *e.g.*, in terms of complexity and decidability, for a better understanding of the underlying mechanisms. Loopingness is the simplest form of non-termination and the vast majority of automated techniques for proving non-termination are designed for finding loops. In [10], the more general concept of *inner-loopingness* in term rewriting is introduced and proved undecidable.

Our aim in this paper is to contribute to the understanding of loopingness. We consider some well-known results for observing non-termination in the *unfoldings* and the *chains* and we instantiate them to loopingness. We provide several examples that seem to indicate that the instantiations are correct as well as partial proofs. Observing loopingness (instead of just non-termination) provides clarifications on the non-termination hardness of the program. On the other hand, an observed non-looping non-termination cannot be detected by an automated technique designed for finding loops, hence it is useless to run such a technique for proving this non-termination.

2 Preliminaries

We assume the reader is familiar with the standard definitions of logic programming [1] and term rewriting [3]. We let $\mathbb{N}$ denote the set of non-negative integers. For any set E, we let $\wp(E)$ denote its power set. We let $\xrightarrow{+}$ (resp. $\xrightarrow{*}$) denote the transitive (resp. reflexive and transitive) closure of a binary relation $\rightarrow$. We fix a finite *signature* $\mathcal{F}$ (the *function symbols*) together with an infinite countable set $\mathcal{V}$ of *variables* with $\mathcal{F} \cap \mathcal{V} = \emptyset$. Constant symbols are denoted by $\mathtt{0}, \mathtt{1}, \dots$, function symbols of positive arity by $\mathtt{f}, \mathtt{g}, \mathtt{s}, \dots$, variables by $x, y, z, \dots$ and terms by $l, r, s, t, \dots$ For any term t, we let $Var(t)$ denote the set of variables occurring in t and $root(t)$ denote the root symbol of t. The set of positions of t is denoted by $Pos(t)$. For any $p \in Pos(t)$, we write $t|_p$ to denote the subterm of t at position p and $t[p \leftarrow s]$ to denote the term obtained from t by replacing $t|_p$ with a term s. A substitution is a finite mapping from variables to terms written as $\{x_1/t_1, \dots, x_n/t_n\}$. A *(variable) renaming* is a substitution that is a bijection on $\mathcal{V}$. The application of a substitution θ to a syntactic object o (*i.e.*, a construct consisting of terms) is denoted by $o\theta$, and $o\theta$ is called an *instance* of o. When θ is a renaming, $o\theta$ is also called a *variant* of o. The substitution θ is a *unifier* of the syntactic objects o and o' if $o\theta = o'\theta$. We let $mgu(o, o')$ denote the (up to variable renaming) most general unifier of o and o'. If O is a set of syntactic objects, we write $o \ll O$ to denote that o is a new occurrence of an element of O whose variables are new (not previously met).

2.1 Logic programming

We also fix a finite set of *predicate symbols* disjoint from $\mathcal{F}$ and $\mathcal{V}$ that is used for constructing atoms. Predicate symbols are denoted by $\mathsf{p}, \mathsf{q}, \ldots$, atoms by $H, A, B, \ldots$ and queries (*i.e.*, sequences of atoms) by bold uppercase letters. Consider a non-empty query $\langle A, \mathbf{A}\rangle$ and a clause c. Let $H \leftarrow \mathbf{B}$ be a variant of c variable disjoint with $\langle A, \mathbf{A}\rangle$ and assume that $\theta = mgu(A, H)$. Then $\langle A, \mathbf{A}\rangle \underset{c}{\overset{\theta}{\Rightarrow}} \langle \mathbf{B}, \mathbf{A}\rangle\theta$ is a *derivation step* with $H \leftarrow \mathbf{B}$ as its *input clause*. If the substitution θ or the clause c is irrelevant, we drop a reference to it. For any logic program (LP) P and queries $\mathbf{Q}, \mathbf{Q}'$, we write $\mathbf{Q} \underset{P}{\Rightarrow} \mathbf{Q}'$ if $\mathbf{Q} \underset{c}{\overset{\theta}{\Rightarrow}} \mathbf{Q}'$ holds for some clause $c \in P$ and some substitution θ. A maximal sequence $\mathbf{Q}_0 \underset{P}{\Rightarrow} \mathbf{Q}_1 \underset{P}{\Rightarrow} \cdots$ of derivation steps is called a *derivation of* $P \cup \{\mathbf{Q}_0\}$ if the *standardization apart* condition holds, *i.e.*, each input clause used is variable disjoint from the initial query $\mathbf{Q}_0$ and from the mgu's and input clauses used at earlier steps. We say that a query $\mathbf{Q}$ is *non-terminating* w.r.t. P if there exists an infinite derivation of $P \cup \{\mathbf{Q}\}$. We say that P is *non-terminating* if there exists a query which is non-terminating w.r.t. it.

2.2 Term rewriting

For any terms s and t and any rewrite rule $R = l \to r$, we write $s \underset{R}{\Rightarrow} t$ if there is a substitution θ and a position $p \in Pos(s)$ such that $s|_p = l\theta$ and $t = s[p \leftarrow r\theta]$. Then $s \underset{R}{\Rightarrow} t$ is called a *rewrite step*. For any term rewriting system (TRS) $\mathcal{R}$, we write $s \underset{\mathcal{R}}{\Rightarrow} t$ if $s \underset{R}{\Rightarrow} t$ holds for some $R \in \mathcal{R}$ (then, we also call $s \underset{\mathcal{R}}{\Rightarrow} t$ a rewrite step). A maximal sequence $s_0 \underset{\mathcal{R}}{\Rightarrow} s_1 \underset{\mathcal{R}}{\Rightarrow} \cdots$ of rewrite steps is called a *rewrite of* $\mathcal{R} \cup \{s_0\}$. We say that a term s is *non-terminating* w.r.t. $\mathcal{R}$ if there exists an infinite rewrite of $\mathcal{R} \cup \{s\}$ and we say that $\mathcal{R}$ is *non-terminating* if there exists a term which is non-terminating w.r.t. it.

3 Observing non-termination in logic programming

The *binary unfoldings* [4, 5] transform a LP P into a possibly infinite set of binary clauses. Intuitively, each generated *binary clause* $H \leftarrow B$ (where B is an atom or the empty query true) specifies that, w.r.t. P, a call to H (or any of its instances) necessarily leads to a call to B (or its corresponding instance). A generated clause of the form $H \leftarrow \mathsf{true}$ indicates a success pattern. In the definition below, $\Im$ denotes the domain of binary clauses (viewed modulo renaming) and id denotes the set of all binary clauses of the form $\mathsf{true} \leftarrow \mathsf{true}$ or $\mathsf{p}(x_1, \ldots, x_n) \leftarrow \mathsf{p}(x_1, \ldots, x_n)$, where p is a predicate symbol of arity n and $x_1, \ldots, x_n$ are distinct variables. Given any set X of binary clauses, $T_P^\beta(X)$ is constructed by unfolding prefixes of clause bodies of P, using elements of $X \cup id$, to obtain new binary clauses.

▶ **Definition 1** (Binary unfoldings).

$$T_P^\beta : \wp(\Im) \to \wp(\Im)$$

$$X \mapsto \left\{ (H \leftarrow B)\theta \;\middle|\; \begin{array}{l} c = H \leftarrow B_1, \ldots, B_m \in P,\ i \in \{1, \ldots, m\} \\ \langle H_j \leftarrow \mathsf{true}\rangle_{j=1}^{i-1} \ll X \\ H_i \leftarrow B \ll X \cup id,\ i < m \Rightarrow B \neq \mathsf{true} \\ \theta = mgu(\langle B_1, \ldots, B_i\rangle, \langle H_1, \ldots, H_i\rangle) \end{array} \right\}$$

and $unf(P) = \bigcup_{n\in\mathbb{N}}(T_P^\beta)^n(\emptyset)$*, where* $(T_P^\beta)^0(\emptyset) = \emptyset$.

▶ **Example 2.** Consider the logic program P that consists of the clauses

$$c_1 = \mathsf{p}(x,y) \leftarrow \mathsf{q}(x), \mathsf{p}(y,x) \qquad c_2 = \mathsf{q}(0) \leftarrow \mathsf{true}$$

Unfolding c_2 using $\mathsf{true} \leftarrow \mathsf{true} \in id$, one gets $c_2' = \mathsf{q}(0) \leftarrow \mathsf{true} \in T_P^\beta(\emptyset)$. Then, unfolding c_1 using c_2', $\mathsf{p}(x',y') \leftarrow \mathsf{p}(x',y') \in id$ and $i = 2$, one gets $c_3 = \mathsf{p}(0,y) \leftarrow \mathsf{p}(y,0) \in (T_P^\beta)^2(\emptyset)$. Finally, unfolding c_1 using c_2', c_3 and $i = 2$, one gets $c_4 = \mathsf{p}(0,0) \leftarrow \mathsf{p}(0,0) \in (T_P^\beta)^3(\emptyset)$.

It is proved in [4] that the binary unfoldings of a LP exhibit its termination properties:

▶ **Theorem 3** (Observing non-termination in the unfoldings)**.** *Let P be a LP and $\mathbf{Q}$ be a query. Then, $\mathbf{Q}$ is non-terminating w.r.t. P iff $\mathbf{Q}$ is non-terminating w.r.t. $unf(P)$.*

For instance, in Ex. 2, we have $c_4 = \mathsf{p}(0,0) \leftarrow \mathsf{p}(0,0) \in unf(P)$, so the query $\mathsf{p}(0,0)$ is non-terminating w.r.t. $unf(P)$. Hence, by Thm. 3, $\mathsf{p}(0,0)$ is non-terminating w.r.t. P.

The proof of Thm. 3 relies on the following definition and theorem.

▶ **Definition 4** (Calls-to relation $\rightsquigarrow$)**.** *Let P be a LP. For any atoms A and B, we say that B is a* call *in a derivation of $P \cup \{A\}$, denoted $A \underset{P}{\rightsquigarrow} B$, if $A \overset{+}{\underset{P}{\Rightarrow}} \langle B, \dots\rangle$; we also write $A \underset{L}{\rightsquigarrow} B$ to emphasize that L is the sequence of clauses of P used in a derivation from A to $\langle B, \dots\rangle$. A P*-chain *is a (possibly infinite) sequence of the form $A_0 \underset{P}{\rightsquigarrow} A_1 \underset{P}{\rightsquigarrow} A_2 \underset{P}{\rightsquigarrow} \cdots$*

▶ **Theorem 5** (Observing non-termination in the chains)**.** *A LP P is non-terminating iff there exists an infinite P-chain.*

For instance, in Ex. 2, we have the infinite P-chain $\mathsf{p}(0,0) \underset{P}{\rightsquigarrow} \mathsf{p}(0,0) \underset{P}{\rightsquigarrow} \cdots$

4 Observing non-termination in term rewriting

We consider the unfolding technique used in [8]. It is defined as a function over the domain $\Re$ of rewrite rules (viewed modulo renaming). It is based on forward and backward narrowing and also performs unfolding on variable positions (contrary to what is usually done in the literature). Note that in general, the unfoldings of a TRS are not finitely computable.

▶ **Definition 6** (Unfoldings)**.**

$$U_{\mathcal{R}} : \wp(\Re) \to \wp(\Re)$$

$$X \mapsto \underbrace{\left\{ (l \to r[p \leftarrow r'])\theta \;\middle|\; \begin{array}{l} l \to r \in X \\ p \in Pos(r) \\ l' \to r' \ll \mathcal{R} \\ \theta = mgu(r|_p, l') \end{array} \right\}}_{forward\ unfoldings} \cup \underbrace{\left\{ (l[p \leftarrow l'] \to r)\theta \;\middle|\; \begin{array}{l} l \to r \in X \\ p \in Pos(l) \\ l' \to r' \ll \mathcal{R} \\ \theta = mgu(l|_p, r') \end{array} \right\}}_{backward\ unfoldings}$$

and $unf(\mathcal{R}) = \bigcup_{n\in\mathbb{N}}(U_{\mathcal{R}})^n(\mathcal{R})$*, where* $(U_{\mathcal{R}})^0(\mathcal{R}) = \mathcal{R}$.

▶ **Example 7.** Consider the TRS $\mathcal{R}$ introduced by Toyama [9] that consists of the rules

$$R_1 = \mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x) \quad R_2 = \mathsf{g}(x,y) \to x \quad R_3 = \mathsf{g}(x,y) \to y$$

We have $R_1 \in (U_{\mathcal{R}})^0(\mathcal{R})$. Unfolding R_1 backwards using R_2 and $p = 1$, one gets $R_4 = \mathsf{f}(\mathsf{g}(0,y'),1,x) \to \mathsf{f}(x,x,x) \in U_{\mathcal{R}}(\mathcal{R})$. Then, unfolding R_4 backwards using R_3 and $p = 2$, one gets $R_5 = \mathsf{f}(\mathsf{g}(0,y'),\mathsf{g}(x'',1),x) \to \mathsf{f}(x,x,x) \in (U_{\mathcal{R}})^2(\mathcal{R})$.

By [6], for all $s \to t \in \mathit{unf}(\mathcal{R})$ we have $s \overset{+}{\underset{\mathcal{R}}{\Rightarrow}} t$. So, as $\mathcal{R} \subseteq \mathit{unf}(\mathcal{R})$ also holds, the unfoldings of a TRS exhibit its termination properties:

▶ **Theorem 8** (Observing non-termination in the unfoldings)**.** *Let $\mathcal{R}$ be a TRS and s be a term. Then, s is non-terminating w.r.t. $\mathcal{R}$ iff s is non-terminating w.r.t. $\mathit{unf}(\mathcal{R})$.*

In Ex. 7 above, we have $R_5 = \mathsf{f}(\mathsf{g}(0, y'), \mathsf{g}(x'', 1), x) \to \mathsf{f}(x, x, x) \in \mathit{unf}(\mathcal{R})$, hence the term $s = \mathsf{f}(\mathsf{g}(0, 1), \mathsf{g}(0, 1), \mathsf{g}(0, 1))$ is non-terminating w.r.t. $\mathit{unf}(\mathcal{R})$ (we have $s \underset{R_5}{\Rightarrow} s \underset{R_5}{\Rightarrow} \cdots$). Consequently, by Thm. 8, s is non-terminating w.r.t. $\mathcal{R}$.

We refer to [2] for details on dependency pairs. The *defined symbols* of a TRS $\mathcal{R}$ are $\mathcal{D}_{\mathcal{R}} = \{\mathit{root}(l) \mid l \to r \in \mathcal{R}\}$. For every $\mathsf{f} \in \mathcal{F}$ we let $\mathsf{f}^{\#}$ be a fresh *tuple symbol* with the same arity as f. If $t = \mathsf{f}(t_1, \dots, t_m)$ is a term, we let $t^{\#}$ denote the construct $\mathsf{f}^{\#}(t_1, \dots, t_m)$. The set of *dependency pairs* of $\mathcal{R}$ is $\mathit{DP}(\mathcal{R}) = \{l^{\#} \to t^{\#} \mid l \to r \in \mathcal{R},\ t \text{ is a subterm of } r,\ \mathit{root}(t) \in \mathcal{D}_{\mathcal{R}}\}$ (viewed modulo renaming). A (possibly infinite) sequence $\mathfrak{C} = \langle s_1^{\#} \to t_1^{\#}, s_2^{\#} \to t_2^{\#}, \dots \rangle$ of dependency pairs of $\mathcal{R}$ is an *$\mathcal{R}$-chain* if there exist substitutions σ_i such that $t_i^{\#}\sigma_i \overset{*}{\underset{\mathcal{R}}{\Rightarrow}} s_{i+1}^{\#}\sigma_{i+1}$ holds for every two consecutive pairs $s_i^{\#} \to t_i^{\#}$ and $s_{i+1}^{\#} \to t_{i+1}^{\#}$ in the sequence. We may also write $\mathfrak{C}$ as $\langle (s_1^{\#} \to t_1^{\#}, \sigma_1), (s_2^{\#} \to t_2^{\#}, \sigma_2), \dots \rangle$ to emphasize that $\sigma_1, \sigma_2, \dots$ are substitutions associated with every two consecutive pairs. It is proved in [2] that the presence of an infinite $\mathcal{R}$-chain is a sufficient and necessary criterion for non-termination:

▶ **Theorem 9** (Observing non-termination in the chains)**.** *A TRS $\mathcal{R}$ is non-terminating* iff *there exists an infinite $\mathcal{R}$-chain.*

For instance, in Ex. 7, $\langle \mathsf{f}^{\#}(0, 1, x) \to \mathsf{f}^{\#}(x, x, x), \mathsf{f}^{\#}(0, 1, x) \to \mathsf{f}^{\#}(x, x, x), \dots \rangle$ is an infinite $\mathcal{R}$-chain because, for $\sigma = \{x / \mathsf{g}(0, 1)\}$, we have $\mathsf{f}^{\#}(x, x, x)\sigma \overset{*}{\underset{\mathcal{R}}{\Rightarrow}} \mathsf{f}^{\#}(0, 1, x)\sigma$.

5 Observing loopingness

The definitions presented below hold both in logic programming and in term rewriting, so we introduce a generic terminology. By a *program* (denoted by Π, $\Pi' \dots$) we mean a LP or a TRS, by a *rule* (denoted by π, $\pi' \dots$) we mean a clause or a rewrite rule, by a *goal* (denoted by α, $\alpha' \dots$) we mean a query or a term, by a *computation* we mean a derivation or a rewrite.

Let $L = \langle \pi_1, \dots, \pi_n \rangle$ be a finite non-empty sequence of rules. For any goals α, α' we write $\alpha \underset{L}{\hookrightarrow} \alpha'$ when $\alpha \underset{\pi_1}{\Rightarrow} \cdots \underset{\pi_n}{\Rightarrow} \alpha'$.

▶ **Definition 10** (Looping)**.** *Let Π be a program, L be a finite non-empty sequence of rules of Π and α be a goal. We say that a computation of $\Pi \cup \{\alpha\}$ is L-*looping *if it is infinite and has the form $\alpha \underset{L}{\hookrightarrow} \alpha_1 \underset{L}{\hookrightarrow} \alpha_2 \underset{L}{\hookrightarrow} \cdots$. We may drop the reference to L if it is not relevant, and simply say that the computation is* looping*. We say that α is* looping *w.r.t. Π if there exists a looping computation of $\Pi \cup \{\alpha\}$. We say that Π is* looping *if there exists a goal which is looping w.r.t. it.*

▶ **Example 11.** Consider the LP P which consists of the clauses $c_1 = \mathsf{p}_1 \leftarrow \mathsf{p}_2$, $c_2 = \mathsf{p}_2 \leftarrow \mathsf{p}_3$, $c_3 = \mathsf{p}_3 \leftarrow \mathsf{p}_1$ and $c_4 = \mathsf{p}_3 \leftarrow \mathsf{p}_4$. Then, p_1 is looping w.r.t. P as we have the infinite derivation $\mathsf{p}_1 \underset{c_1}{\Rightarrow} \mathsf{p}_2 \underset{c_2}{\Rightarrow} \mathsf{p}_3 \underset{c_3}{\Rightarrow} \mathsf{p}_1 \underset{c_1}{\Rightarrow} \mathsf{p}_2 \underset{c_2}{\Rightarrow} \mathsf{p}_3 \underset{c_3}{\Rightarrow} \mathsf{p}_1 \underset{c_1}{\Rightarrow} \cdots$ *i.e.*, for $L = \langle c_1, c_2, c_3 \rangle$, $\mathsf{p}_1 \underset{L}{\hookrightarrow} \mathsf{p}_1 \underset{L}{\hookrightarrow} \mathsf{p}_1 \underset{L}{\hookrightarrow} \cdots$

We extend the concept of loopingness to chains.

▶ **Definition 12** (Looping chain)**.** *Let P be a LP and $\mathcal{R}$ be a TRS.*

- *We say that a P-chain is* looping *if it is infinite and has the form $A_0 \underset{L}{\leadsto} A_1 \underset{L}{\leadsto} \cdots$ where L is a finite, non-empty, sequence of clauses of P.*
- *We say that an $\mathcal{R}$-chain is* looping *if it is infinite and has the form $\langle L, L, \dots \rangle$, where L is a finite, non-empty, sequence of elements of $DP(\mathcal{R}) \times$ Substitutions.*

▶ **Example 13** (Ex. 7 continued). Let $p = \big(\mathsf{f}^{\#}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}^{\#}(x,x,x), \{x/\,\mathsf{g}(\mathsf{0},\mathsf{1})\}\big)$. Then, $\langle p, p, \dots \rangle$ is a looping $\mathcal{R}$-chain.

Note that there exist infinite computations which are not looping, *i.e.*, do not correspond to the infinite repetition of the same sequence of rules.

▶ **Example 14.** Let P be the LP which consists of the clauses $c_1 = \mathsf{p}(\mathsf{0},y) \leftarrow \mathsf{p}(\mathsf{s}(y),\mathsf{s}(y))$ and $c_2 = \mathsf{p}(\mathsf{s}(x),y) \leftarrow \mathsf{p}(x,y)$. We have the following infinite derivation of $P \cup \{\mathsf{p}(\mathsf{0},\mathsf{0})\}$:

$$\mathsf{p}(\mathsf{0},\mathsf{0}) \underset{c_1}{\Rightarrow} \mathsf{p}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})) \underset{c_2}{\Rightarrow} \mathsf{p}(\mathsf{0},\mathsf{s}(\mathsf{0})) \underset{c_1}{\Rightarrow} \mathsf{p}(\mathsf{s}^2(\mathsf{0}),\mathsf{s}^2(\mathsf{0})) \overset{2}{\underset{c_2}{\Rightarrow}} \mathsf{p}(\mathsf{0},\mathsf{s}^2(\mathsf{0})) \underset{c_1}{\Rightarrow} \cdots$$

It is not looping as it follows the path $\langle c_1, c_2, c_1, c_2, c_2, c_1, \dots \rangle$. We also have the infinite, non-looping, P-chain:

$$\mathsf{p}(\mathsf{0},\mathsf{0}) \underset{c_1}{\leadsto} \mathsf{p}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})) \underset{c_2}{\leadsto} \mathsf{p}(\mathsf{0},\mathsf{s}(\mathsf{0})) \underset{c_1}{\leadsto} \mathsf{p}(\mathsf{s}^2(\mathsf{0}),\mathsf{s}^2(\mathsf{0})) \underset{\langle c_2,c_2\rangle}{\leadsto} \mathsf{p}(\mathsf{0},\mathsf{s}^2(\mathsf{0})) \underset{c_1}{\leadsto} \cdots$$

▶ **Example 15.** Let $\mathcal{R}$ be the TRS which consists of the rules $R_1 = \mathsf{f}(\mathsf{0},y) \to \mathsf{f}(\mathsf{s}(y),\mathsf{s}(y))$ and $R_2 = \mathsf{f}(\mathsf{s}(x),y) \to \mathsf{f}(x,y)$. We have the following infinite rewrite of $\mathcal{R} \cup \{\mathsf{f}(\mathsf{0},\mathsf{0})\}$:

$$\mathsf{f}(\mathsf{0},\mathsf{0}) \underset{R_1}{\Rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{s}(\mathsf{0})) \underset{R_2}{\Rightarrow} \mathsf{f}(\mathsf{0},\mathsf{s}(\mathsf{0})) \underset{R_1}{\Rightarrow} \mathsf{f}(\mathsf{s}^2(\mathsf{0}),\mathsf{s}^2(\mathsf{0})) \overset{2}{\underset{R_2}{\Rightarrow}} \mathsf{f}(\mathsf{0},\mathsf{s}^2(\mathsf{0})) \underset{R_1}{\Rightarrow} \cdots$$

It is not looping as it follows the path $\langle R_1, R_2, R_1, R_2, R_2, R_1, \dots \rangle$. We also have the infinite, non-looping, $\mathcal{R}$-chain $\langle (R_1^{\#}, \sigma_1), (R_2^{\#}, \theta_1), (R_1^{\#}, \sigma_2), (R_2^{\#}, \theta_2), \dots \rangle$ where $R_1^{\#} = s_1^{\#} \to t_1^{\#} = \mathsf{f}^{\#}(\mathsf{0},y) \to \mathsf{f}^{\#}(\mathsf{s}(y),\mathsf{s}(y))$, $R_2^{\#} = s_2^{\#} \to t_2^{\#} = \mathsf{f}^{\#}(\mathsf{s}(x),y) \to \mathsf{f}^{\#}(x,y)$ and, for all $i > 0$, $\sigma_i = \{y/\,\mathsf{s}^{i-1}(\mathsf{0})\}$ and $\theta_i = \{x/\mathsf{0}, y/\,\mathsf{s}^i(\mathsf{0})\}$. Indeed, we have $t_1^{\#}\sigma_1 \overset{*}{\underset{\mathcal{R}}{\Rightarrow}} s_2^{\#}\theta_1$, $t_2^{\#}\theta_1 \overset{*}{\underset{\mathcal{R}}{\Rightarrow}} s_1^{\#}\sigma_2$, ...

All the examples given above seem to indicate that the *Observing non-termination* results of Sect. 3 and Sect. 4 can be instantiated to loopingness.

▶ **Lemma 16** (Observing loopingness in the chains). *If, for a program Π, there exists a looping Π-chain then Π is looping.*

Proof. For LPs, the result immediately follows from Def. 4 and Def. 12. For TRSs, it is proved in [2] that any infinite Π-chain $\mathfrak{C} = \langle (s_1^{\#} \to t_1^{\#}, \sigma_1), (s_2^{\#} \to t_2^{\#}, \sigma_2), \dots \rangle$ corresponds to an infinite rewrite $\mathfrak{C}' = (s_1\sigma_1 \underset{R_1}{\Rightarrow} C_1[t_1]\sigma_1 \overset{*}{\underset{\Pi}{\Rightarrow}} C_1[s_2]\sigma_2 \underset{R_2}{\Rightarrow} C_1[C_2[t_2]]\sigma_2 \overset{*}{\underset{\Pi}{\Rightarrow}} \cdots)$ where $R_1 = s_1 \to C_1[t_1]$, $R_2 = s_2 \to C_2[t_2]$, ... are rewrite rules of Π and the rewrites in $\overset{*}{\underset{\Pi}{\Rightarrow}}$ do not occur in the C_i's (*i.e.*, they are those of $t_i^{\#}\sigma_i \overset{*}{\underset{\Pi}{\Rightarrow}} s_{i+1}^{\#}\sigma_{i+1}$). Hence, if $\mathfrak{C}$ is looping, so is $\mathfrak{C}'$. ◀

▶ **Conjecture 17** (Observing loopingness in the chains). *If a program Π is looping then there exists a looping Π-chain.*

Proof sketch for a LP P. Let $\mathbf{Q}_0 \underset{L}{\hookrightarrow} \mathbf{Q}_1 \underset{L}{\hookrightarrow} \cdots$ be a looping derivation of $P \cup \{\mathbf{Q}_0\}$. Then, in each step $\mathbf{Q}_i \underset{L}{\hookrightarrow} \mathbf{Q}_{i+1}$ there is a query $\langle A, \dots \rangle$ that has an infinite derivation. For all $i \in \mathbb{N}$, let $\langle A_i, \dots \rangle$ be the leftmost such query in $\mathbf{Q}_i \underset{L}{\hookrightarrow} \mathbf{Q}_{i+1}$. Then, for all $i \in \mathbb{N}$ we have $A_i \underset{P}{\leadsto} A_{i+1}$. Let L' be the sequence of clauses used in $A_0 \overset{+}{\underset{P}{\Rightarrow}} \langle A_1, \dots \rangle$. We prove by induction on i that, for all $i \in \mathbb{N}$, L' is used in $A_i \overset{+}{\underset{P}{\Rightarrow}} \langle A_{i+1}, \dots \rangle$ and hence that $A_i \underset{L'}{\leadsto} A_{i+1}$. ◀

▶ **Lemma 18** (Observing loopingness in the unfoldings)**.** *A term is looping w.r.t. a TRS $\mathcal{R}$ iff it is looping w.r.t. $unf(\mathcal{R})$.*

Proof. ($\Rightarrow$) As $\mathcal{R} \subseteq unf(\mathcal{R})$, any rewrite with $\mathcal{R}$ is also a rewrite with $unf(\mathcal{R})$. ($\Leftarrow$) For all $s \to t \in unf(\mathcal{R})$ we have $s \overset{+}{\underset{\mathcal{R}}{\Rightarrow}} t$ [6], so replacing, in a looping rewrite with $unf(\mathcal{R})$, each step by the corresponding finite sequence of steps in $\mathcal{R}$, one gets a looping rewrite with $\mathcal{R}$. ◀

▶ **Conjecture 19** (Observing loopingness in the binary unfoldings)**.** *A query is looping w.r.t. a LP P iff it is looping w.r.t. $unf(P)$.*

Proof sketch. Use Lem. 16 + Conj. 17 and the fact that $\underset{P}{\rightsquigarrow} = \underset{unf(P)}{\rightsquigarrow}$ [4]. ◀

▶ **Example 20.** In Ex. 2, we have the $\langle c_4 \rangle$-looping derivation $\mathsf{p}(0,0) \underset{c_4}{\Rightarrow} \mathsf{p}(0,0) \underset{c_4}{\Rightarrow} \cdots$ of $unf(P) \cup \{\mathsf{p}(0,0)\}$ and the $\langle c_1, c_2 \rangle$-looping derivation $\mathsf{p}(0,0) \underset{c_1}{\Rightarrow} \langle \mathsf{q}(0), \mathsf{p}(0,0) \rangle \underset{c_2}{\Rightarrow} \mathsf{p}(0,0) \underset{c_1}{\Rightarrow} \cdots$ of $P \cup \{\mathsf{p}(0,0)\}$. Note that $c_1 \notin unf(P)$ (c_1 is not binary) hence this derivation of $P \cup \{\mathsf{p}(0,0)\}$ is not a derivation of $unf(P) \cup \{\mathsf{p}(0,0)\}$. In Ex. 7, for $s = \mathsf{f}(\mathsf{g}(0,1), \mathsf{g}(0,1), \mathsf{g}(0,1))$, we have the $\langle R_5 \rangle$-looping rewrite $s \underset{R_5}{\Rightarrow} s \underset{R_5}{\Rightarrow} \cdots$ of $unf(\mathcal{R}) \cup \{s\}$ and the $\langle R_2, R_3, R_1 \rangle$-looping rewrite $s \underset{R_2}{\Rightarrow} \mathsf{f}(0, \mathsf{g}(0,1), \mathsf{g}(0,1)) \underset{R_3}{\Rightarrow} \mathsf{f}(0,1,\mathsf{g}(0,1)) \underset{R_1}{\Rightarrow} s \underset{R_2}{\Rightarrow} \cdots$ of $\mathcal{R} \cup \{s\}$. As $\mathcal{R} \subseteq unf(\mathcal{R})$, this rewrite of $\mathcal{R} \cup \{s\}$ is also a rewrite of $unf(\mathcal{R}) \cup \{s\}$.

6 Acknowledgement and future work

We thank the anonymous referees for their valuable comments and constructive criticisms.

Besides finishing the proofs (Conj. 19 and Conj. 17), we plan to extend the results to dependency pairs in logic programming [7] and to inner-loopingness [10]. We also plan to unify more concepts from termination analysis of LPs and TRSs.

References

1. K. R. Apt. *From logic programming to Prolog*. Prentice Hall International series in computer science. Prentice Hall, 1997.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
4. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999. doi:10.1016/S0743-1066(99)00006-0.
5. M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In H. Berghel, T. Hlengl, and J. E. Urban, editors, *Proc. of SAC'94*, pages 394–399. ACM Press, 1994. doi:10.1145/326619.326789.
6. J. V. Guttag, D. Kapur, and D. R. Musser. On proving uniform termination and restricted termination of rewriting systems. *SIAM Journal of Computing*, 12(1):189–214, 1983. doi:10.1137/0212012.
7. M. T. Nguyen, J. Giesl, P. Schneider-Kamp, and D. De Schreye. Termination analysis of logic programs based on dependency graphs. In A. King, editor, *Proc. of LOPSTR'07*, volume 4915 of *LNCS*, pages 8–22. Springer, 2007. doi:10.1007/978-3-540-78769-3_2.
8. É. Payet. Loop detection in term rewriting using the eliminating unfoldings. *Theoretical Computer Science*, 403(2-3):307–327, 2008. doi:10.1016/j.tcs.2008.05.013.
9. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987. doi:10.1016/0020-0190(87)90122-0.
10. Y. Wang and M. Sakai. On non-looping term rewriting. In A. Geser and H. Søndergaard, editors, *Proc. of WST'06*, pages 17–21, 2006.