Binary Non-Termination in Term Rewriting and Logic Programming

Etienne Payet

Binary Non-Termination in Term Rewriting and Logic Programming

Étienne Payet
LIM - Université de la Réunion, France

Abstract

We present a new syntactic criterion for the automatic detection of non-termination in an abstract setting that encompasses a simplified form of term rewriting and logic programming.

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

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

1 Introduction

This paper is concerned with non-termination in structures where one rewrites elements using indexed binary relations. Such structures can be formalised by *abstract reduction systems* (ARSs) [3], *i.e.*, couples $(A, \Rightarrow_I)$ where A is a set and $\Rightarrow_I$ (the rewrite relation) is the union of binary relations on A, indexed by a set I, *i.e.*, $\Rightarrow_I = \bigcup\{\Rightarrow_\iota \mid \iota \in I\}$. Non-termination in these structures can be formalised as the existence of an infinite rewrite sequence $a_0 \Rightarrow_{\iota_0} a_1 \Rightarrow_{\iota_1} \cdots$. *Term rewrite systems (TRSs)* and *logic programs (LPs)* are concrete instances of ARSs: A is the set of finite terms and I indicates what rule (= a couple of finite terms) is applied at what position. A crucial difference is that the rewrite relation of TRSs relies on instantiation while that of LPs relies on narrowing, *i.e.*, on unification. In this paper, we present a new syntactic criterion for the automatic detection of non-termination in an abstract setting that encompasses a simplified form of term rewriting and logic programming. Namely, we suppose that the rewriting always takes place at the root position of terms (see Def. 4 below). There exist program transformation techniques that make it possible to place oneself in such a context, *e.g.*, the overlap closure [8] in term rewriting or the binary unfoldings [4, 6] in logic programming preserve the non-termination property of the original program.

2 Preliminaries

We let $\mathbb{N}$ denote the set of non-negative integers.

2.1 Binary Relations

If $\Rightarrow$ and $\hookrightarrow$ are binary relations on a set A, then $\Rightarrow \circ \hookrightarrow$ denotes their *composition*. We let $\Rightarrow^0$ be the identity relation and, for all $n \in \mathbb{N}$, $\Rightarrow^{n+1} = (\Rightarrow^n \circ \Rightarrow)$. Moreover, $\Rightarrow^* = \bigcup\{\Rightarrow^n \mid n \geq 0\}$ is the *reflexive and transitive closure* of $\Rightarrow$. We formalise non-termination as the existence of an infinite sequence of connected elements:

▶ **Definition 1.** *Let $\Rightarrow$ be a binary relation on a set A. A* $\Rightarrow$-chain *is a (possibly infinite) sequence $a_0, a_1, \ldots$ of elements of A such that $a_n \Rightarrow a_{n+1}$ for all $n \in \mathbb{N}$. We simply write it as $a_0 \Rightarrow a_1 \Rightarrow \cdots$.*

2.2 Terms

We use the same definitions and notations as [3] for terms. From now on, we fix a *signature* Σ (the *function symbols*) together with an infinite countable set X of *variables*, with $\Sigma \cap X = \emptyset$.

We let $\mathsf{f}, \mathsf{g}, \mathsf{s}$ be function symbols of positive arity and $\mathsf{0}$ be a constant symbol. The set of all *terms* built from Σ and X is denoted by $T(\Sigma, X)$. A *context* is a term with at least one "hole", represented by $\square$, in it. For all terms or contexts t, we let $Var(t)$ denote the set of variables occurring in t and, for all contexts c, we let $c[t]$ denote the term or context obtained from c by replacing all the occurrences of $\square$ by t. For all contexts c, we let $c^0 = \square$ and, for all $n \in \mathbb{N}$, $c^{n+1} = c[c^n]$. Terms are generally denoted by a, s, t, u, v, variables by x, y and contexts by c, possibly with subscripts and quotes.

The set $S(\Sigma, X)$ of all *substitutions* consists of the functions θ from X to $T(\Sigma, X)$ such that $Dom(\theta) = \{x \in X \mid \theta(x) \neq x\}$ is finite. A substitution θ is usually written as $\{x \mapsto \theta(x) \mid x \in Dom(\theta)\}$ and its application to a term s as $s\theta$. A *renaming* is a substitution that is a bijection on X. The *composition* of substitutions σ and θ is denoted as $\sigma\theta$. We say that σ is *more general than* θ if $\theta = \sigma\eta$ for some substitution η. We let $\theta^0 = \emptyset$ (the identity substitution) and, for all $n \in \mathbb{N}$, $\theta^{n+1} = \theta^n\theta$.

A term s is an *instance* of a term t if $s = t\theta$ for some $\theta \in S(\Sigma, X)$. On the other hand, s *unifies* with t if $s\theta = t\theta$ for some $\theta \in S(\Sigma, X)$; then, θ is called a *unifier* of s and t and $mgu(s, t)$ denotes the *most general unifier* (mgu) of s and t.

2.3 Term Rewriting and Logic Programming

We refer to [3] (resp. [1]) for the basics of term rewriting (resp. logic programming).

▶ **Definition 2.** *A* program *is a subset of* $T(\Sigma, X)^2$*, every element* (u, v) *of which is called a* rule*, where u (resp. v) is the* left-hand side *(resp.* right-hand side*). For each program P, we let $\overline{P}$ denote the set of all finite, non-empty, sequences of elements of P.*

In this paper, we only consider ARSs $(A, \Rightarrow_I)$ such that $A = T(\Sigma, X)$ and I is a program. Hence the following simplified definition.

▶ **Definition 3.** *An* abstract reduction system (ARS) *is a union of binary relations on $T(\Sigma, X)$ indexed by a program, i.e., it has the form $\Rightarrow_P = \bigcup\{\Rightarrow_r \subseteq T(\Sigma, X)^2 \mid r \in P\}$ for some program P. For each ARS $\Rightarrow_P$ and each $\omega = (r_1, \dots, r_n)$ in $\overline{P}$, we let $\Rightarrow_\omega = (\Rightarrow_{r_1} \circ \cdots \circ \Rightarrow_{r_n})$.*

The next definition introduces term rewrite systems and logic programs as concrete instances of ARSs. For all terms s and rules (u, v) and (u', v'), we write $(u, v) \ll_s (u', v')$ to denote that (u, v) is a *variant* of (u', v') *variable disjoint* with s, *i.e.*, for some renaming γ, we have $u = u'\gamma$, $v = v'\gamma$ and $Var(u) \cap Var(s) = Var(v) \cap Var(s) = \emptyset$.

▶ **Definition 4.** *For each program P, we let $\rightarrow_P = \bigcup\{\rightarrow_r \mid r \in P\}$ and $\rightsquigarrow_P = \bigcup\{\rightsquigarrow_r \mid r \in P\}$ where, for all $r \in P$,*

$$\underset{r}{\rightarrow} = \left\{\left(u\theta, v\theta\right) \in T(\Sigma, X)^2 \;\middle|\; (u, v) = r,\ \theta \in S(\Sigma, X)\right\} \qquad \textit{(Term Rewriting)}$$

$$\underset{r}{\rightsquigarrow} = \left\{\left(s, v\theta\right) \in T(\Sigma, X)^2 \;\middle|\; (u, v) \ll_s r,\ \theta = mgu(s, u)\right\} \qquad \textit{(Logic Programming)}$$

We say that $\rightarrow_P$ (resp. $\rightsquigarrow_P$) is a term rewrite system *(resp. a* logic program*).*

▶ **Example 5.** Let $r = \big(\mathsf{f}(x), \mathsf{s}(x)\big) = (u, v)$. Then, $\mathsf{f}^2(x) \rightarrow_r \mathsf{s}(\mathsf{f}(x))$ because $\mathsf{f}^2(x) = u\theta$ and $\mathsf{s}(\mathsf{f}(x)) = v\theta$ for $\theta = \{x \mapsto \mathsf{f}(x)\}$. Let $r' = \big(\mathsf{f}(\mathsf{g}(x, \mathsf{0})), \mathsf{f}(x)\big)$ and $s = \mathsf{f}(\mathsf{g}(x, x))$. The rule $(u', v') = (\mathsf{f}(\mathsf{g}(x', \mathsf{0})), \mathsf{f}(x'))$ is a variant of r' variable disjoint with s. Let $\theta' = \{x \mapsto \mathsf{0}, x' \mapsto \mathsf{0}\}$. Then, $\theta' = mgu\,(s, u')$ and we have $s \rightsquigarrow_{r'} v'\theta'$, *i.e.*, $\mathsf{f}(\mathsf{g}(x, x)) \rightsquigarrow_{r'} \mathsf{f}(\mathsf{0})$.

In term rewriting and in logic programming (modulo a condition), the left-hand side of a rule can be rewritten to the corresponding instance of the right-hand side.

▶ **Lemma 6.** *Let $r = (u, v)$ be a rule and θ be a substitution. We have $u\theta \rightarrow_r v\theta$ and, if $Var(v) \subseteq Var(u)$, $u\theta \rightsquigarrow_r v\theta$.*

3 Binary Non-Termination

We are interested in *binary chains*, *i.e.*, infinite chains that consist of the repetition of two sequences of rules. There are ARSs that admit such chains but no infinite chain consisting of the repetition of a single sequence (see, *e.g.*, $\rightarrow_P$ in Ex. 8 and Ex. 9 below). More precisely:

▶ **Definition 7.** *Let* $\Rightarrow_P$ *be an ARS and* $\omega_1, \omega_2 \in \overline{P}$. *A* $(\omega_1, \omega_2, \Rightarrow_P)$-chain *is an infinite* $(\Rightarrow^*_{\omega_1} \circ \Rightarrow_{\omega_2})$*-chain.*

▶ **Example 8.** Let $\Rightarrow_P \in \{\rightarrow_P, \rightsquigarrow_P\}$ where P is the program that consists of the rules

$$r_1 = \big(\mathsf{f}(x, \mathsf{s}(y)), \mathsf{f}(\mathsf{s}^2(x), y)\big) \qquad r_2 = \big(\mathsf{f}(x, \mathsf{0}), \mathsf{f}(\mathsf{s}(\mathsf{0}), x)\big)$$

(see [13] and `TRS_Standard/Zantema_15/ex12.xml` in [11]). We have the $(r_1, r_2, \Rightarrow_P)$-chain:

$$\mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{0}) \overset{0}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{0}) \underset{r_2}{\Rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0})) \overset{1}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}^3(\mathsf{0}), \mathsf{0}) \underset{r_2}{\Rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}^3(\mathsf{0})) \overset{3}{\underset{r_1}{\Rightarrow}} \cdots$$

▶ **Example 9.** Let $\Rightarrow_P \in \{\rightarrow_P, \rightsquigarrow_P\}$ where P is the program that consists of the rules

$$r_1 = \big(\mathsf{f}(x, \mathsf{s}(y)), \mathsf{f}(\mathsf{s}(x), y)\big) \qquad r_2 = \big(\mathsf{f}(x, \mathsf{0}), \mathsf{f}(x, \mathsf{s}(x))\big)$$

(see [13] and `TRS_Standard/Zantema_15/ex14.xml` in [11]). We have the $(r_1, r_2, \Rightarrow_P)$-chain:

$$\mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \overset{1}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{0}) \underset{r_2}{\Rightarrow} \mathsf{f}(\mathsf{s}(\mathsf{0}), \mathsf{s}^2(\mathsf{0})) \overset{2}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}^3(\mathsf{0}), \mathsf{0}) \underset{r_2}{\Rightarrow} \mathsf{f}(\mathsf{s}^3(\mathsf{0}), \mathsf{s}^4(\mathsf{0})) \overset{4}{\underset{r_1}{\Rightarrow}} \cdots$$

Now, we present a criterion for the detection of binary chains. It is tailored to deal with specific sequences ω_1 and ω_2 that each consist of a single rule of a particular form. Intuitively, the rule r_1 of ω_1 and the rule r_2 of ω_2 are mutually recursive; in r_1, a context c is removed from the left-hand side to the right-hand side while, in r_2, c is added again. Ex. 8 and Ex. 9 are concrete instances, with $c = \mathsf{s}(\square)$. This is formalised as follows.

▶ **Definition 10.** *A* recurrent pair *for a program* P *is a pair* $(r_1, r_2) \in P^2$ *such that*

- $r_1 = \big(\mathsf{f}(x, c[y]), \mathsf{f}(c^{n_1}[x], y)\big)$ *and* $r_2 = \big(\mathsf{f}(x, s), \mathsf{f}(c^{n_2}[t], c^{n_3}[x])\big)$
- $x \neq y$
- $\mathit{Var}(c) = \mathit{Var}(s) = \emptyset$
- $t \in \{x, s\}$

▶ **Example 11.** In Ex. 8, we have $(n_1, n_2, n_3) = (2, 1, 0)$, $c = \mathsf{s}(\square)$ and $s = t = \mathsf{0}$. In Ex. 9, we have $(n_1, n_2, n_3) = (1, 0, 1)$, $c = \mathsf{s}(\square)$, $s = \mathsf{0}$ and $t = x$.

We show that the existence of a recurrent pair leads to that of a binary chain (see Prop. 20), provided that property (1) below is satisfied. The rest of this section is parametric in an ARS $\Rightarrow_P$ and a recurrent pair (r_1, r_2) for P as in Def. 10, with $r_1 = (u_1, v_1)$ and $r_2 = (u_2, v_2)$. We suppose that we have

$$\forall \theta \in S(\Sigma, X)\ (u_1\theta \underset{r_1}{\Rightarrow} v_1\theta) \wedge (u_2\theta \underset{r_2}{\Rightarrow} v_2\theta) \tag{1}$$

As $\mathit{Var}(v_1) \subseteq \mathit{Var}(u_1)$ and $\mathit{Var}(v_2) \subseteq \mathit{Var}(u_2)$, by Lem. 6 both $\rightarrow_p$ and $\rightsquigarrow_P$ satisfy (1).

For the sake of readability, we introduce the following notation.

▶ **Definition 12.** *For all* $m, n \in \mathbb{N}$, *we let* $\mathsf{f}(m, n)$ *denote the term* $\mathsf{f}(c^m[s], c^n[s])$.

Then, we have the following two lemmas. Lem. 13 states that r_1 allows one to iteratively move a tower of c's from the second to the first argument of f. Conversely, Lem. 14 states that r_2 allows one to copy a tower of c's from the first to the second argument of f in just one step.

▶ **Lemma 13.** *For all $m, n \in \mathbb{N}$, $\mathsf{f}(m,n) \Rightarrow_{r_1}^{n} \mathsf{f}(n_1 \times n + m, 0)$.*

Proof. We proceed by induction on n.

- (Base: $n = 0$) Here, $\Rightarrow_{r_1}^{n}$ is the identity. Hence, for all $m \in \mathbb{N}$, we have $\mathsf{f}(m,n) \Rightarrow_{r_1}^{n} \mathsf{f}(m,n)$, where $\mathsf{f}(m,n) = \mathsf{f}(n_1 \times n + m, 0)$.
- (Induction) Suppose that for some $n \in \mathbb{N}$ we have $\mathsf{f}(m,n) \Rightarrow_{r_1}^{n} \mathsf{f}(n_1 \times n + m, 0)$ for all $m \in \mathbb{N}$. Let $m \in \mathbb{N}$. Then, $\mathsf{f}(m, n+1) = \mathsf{f}(c^m[s], c^{n+1}[s]) = u_1\{x \mapsto c^m[s], y \mapsto c^n[s]\}$. Therefore, by (1), we have $\mathsf{f}(m, n+1) \Rightarrow_{r_1} v_1\{x \mapsto c^m[s], y \mapsto c^n[s]\}$ where $v_1\{x \mapsto c^m[s], y \mapsto c^n[s]\} = \mathsf{f}(c^{n_1+m}[s], c^n[s]) = \mathsf{f}(n_1 + m, n)$. But, by induction hypothesis, we have $\mathsf{f}(n_1 + m, n) \Rightarrow_{r_1}^{n} \mathsf{f}(n_1 \times n + (n_1 + m), 0)$, *i.e.*, $\mathsf{f}(n_1 + m, n) \Rightarrow_{r_1}^{n} \mathsf{f}(n_1 \times (n+1) + m, 0)$. Finally, $\mathsf{f}(m, n+1) \Rightarrow_{r_1}^{n+1} \mathsf{f}(n_1 \times (n+1) + m, 0)$.

◀

▶ **Lemma 14.** *For all $m \in \mathbb{N}$, $\mathsf{f}(m,0) \Rightarrow_{r_2} \mathsf{f}(m' + n_2, m + n_3)$ where $m' = 0$ if $t = s$ and $m' = m$ if $t = x$.*

Proof. Let $m \in \mathbb{N}$. We have $\mathsf{f}(m,0) = \mathsf{f}(c^m[s], s) = u_2\{x \mapsto c^m[s]\}$. Hence, by (1), we have $\mathsf{f}(m,0) \Rightarrow_{r_2} v_2\{x \mapsto c^m[s]\}$.

- If $t = s$ then $v_2\{x \mapsto c^m[s]\} = \mathsf{f}(c^{n_2}[s], c^{m+n_3}[s]) = \mathsf{f}(n_2, m + n_3)$.
- If $t = x$ then $v_2\{x \mapsto c^m[s]\} = \mathsf{f}(c^{m+n_2}[s], c^{m+n_3}[s]) = \mathsf{f}(m + n_2, m + n_3)$.

◀

We consider the following polynomials in the indeterminate $i \in \mathbb{N}$. We define them in a mutually recursive way, which reflects the mutually recursive nature of r_1 and r_2 and hence facilitates the proof of the existence of a $(r_1, r_2, \Rightarrow_P)$-chain (Prop. 20 below).

▶ **Definition 15.** *We let*

- $\Pi_0(i) = n_2$ *and* $\Pi'_0(i) = n_3$
- $\Pi_{n+1}(i) = \Delta_n(i) + n_2$ *and* $\Pi'_{n+1}(i) = \Delta'_n(i) + n_3$ *for all* $n \in \mathbb{N}$

where, for all $n \in \mathbb{N}$,

- $\Delta_n(i) = 0$ *if* $t = s$ *and* $\Delta_n(i) = \Delta'_n(i)$ *if* $t = x$
- $\Delta'_n(i) = i\Pi'_n(i) + \Pi_n(i)$.

▶ **Example 16.** In Ex. 9, we have $t = x$ and $(n_1, n_2, n_3) = (1, 0, 1)$. Hence:

- $\Pi_0(i) = n_2 = 0$ and $\Pi'_0(i) = n_3 = 1$
- $\Pi_1(i) = \Delta_0(i) + n_2 = \Delta'_0(i) = i\Pi'_0(i) + \Pi_0(i) = i$
- $\Pi'_1(i) = \Delta'_0(i) + n_3 = i + 1$
- $\Pi_2(i) = \Delta_1(i) + n_2 = \Delta'_1(i) = i\Pi'_1(i) + \Pi_1(i) = i^2 + i + i = i^2 + 2i$
- $\Pi'_2(i) = \Delta'_1(i) + n_3 = i^2 + 2i + 1$

The next lemma provides a simpler form of Π and Π' for the case $t = s$ (the case $t = x$ is more intricate).

▶ **Lemma 17.** *If $t = s$ then, for all $n \in \mathbb{N}$, $\Pi_n(i) = n_2$ and $\Pi'_n(i) = n_3 i^n + \sum_{k=0}^{n-1}(n_2 + n_3)i^k$.*

Proof. Suppose that $t = s$. Then, for all $n \in \mathbb{N}$, $\Delta_n(i) = 0$, so $\Pi_{n+1}(i) = n_2$. As $\Pi_0(i) = n_2$ also, for all $n \in \mathbb{N}$ we have $\Pi_n(i) = n_2$. Now, we prove that $\Pi'_n(i) = n_3 i^n + \sum_{k=0}^{n-1}(n_2 + n_3)i^k$. We proceed by induction on n.

- (Base: $n=0$) We have $\Pi'_n(i) = n_3 = n_3 i^n + \sum_{k=0}^{n-1}(n_2+n_3)i^k$.
- (Induction) Suppose that the property holds for some $n \in \mathbb{N}$. We have $\Pi'_{n+1}(i) = \Delta'_n(i) + n_3 = i\Pi'_n(i) + \Pi_n(i) + n_3$. But, as $t = s$, $\Pi_n(i) = n_2$ and, by induction hypothesis, $\Pi'_n(i) = n_3 i^n + \sum_{k=0}^{n-1}(n_2+n_3)i^k$. So, $\Pi'_{n+1}(i) = i(n_3 i^n + \sum_{k=0}^{n-1}(n_2+n_3)i^k) + n_2 + n_3 = n_3 i^{n+1} + \sum_{k=0}^{n}(n_2+n_3)i^k$.

◀

▶ **Example 18.** In Ex. 8, we have $t = s$ and $(n_1, n_2, n_3) = (2, 1, 0)$. Hence, by Lem. 17, we have $\Pi_n(i) = 1$ and $\Pi'_n(i) = \sum_{k=0}^{n-1} i^k$ for all $n \in \mathbb{N}$.

Using Π and Π', we define the set of terms A:

▶ **Definition 19.** *We let* $A = \{a_n = \mathsf{f}(\Pi_n(n_1), \Pi'_n(n_1)) \mid n \in \mathbb{N}\}$.

Now we prove the existence of the $(r_1, r_2, \Rightarrow_P)$-chain

$$a_0 \left(\overset{\Pi'_0(n_1)}{\underset{r_1}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow}\right) a_1 \left(\overset{\Pi'_1(n_1)}{\underset{r_1}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow}\right) a_2 \left(\overset{\Pi'_2(n_1)}{\underset{r_1}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow}\right) \cdots$$

▶ **Proposition 20.** *For all $n \in \mathbb{N}$, we have* $a_n \left(\Rightarrow_{r_1}^{\Pi'_n(n_1)} \circ \Rightarrow_{r_2}\right) a_{n+1}$.

Proof. Let $n \in \mathbb{N}$. We have $a_n = \mathsf{f}(\Pi_n(n_1), \Pi'_n(n_1))$. By Lem. 13 and Lem. 14,

$$a_n \overset{\Pi'_n(n_1)}{\underset{r_1}{\Rightarrow}} \mathsf{f}\left(\underbrace{n_1 \times \Pi'_n(n_1) + \Pi_n(n_1)}_{\Delta'_n(n_1)}, 0\right) \underset{r_2}{\Rightarrow} \mathsf{f}\left(m, \underbrace{\Delta'_n(n_1) + n_3}_{\Pi'_{n+1}(n_1)}\right)$$

where $m = n_2 = \Pi_{n+1}(n_1)$ if $t = s$ and $m = \Delta'_n(n_1) + n_2 = \Pi_{n+1}(n_1)$ if $t = x$. Hence, $a_n \left(\Rightarrow_{r_1}^{\Pi'_n(n_1)} \circ \Rightarrow_{r_2}\right) a_{n+1}$. ◀

▶ **Example 21.** In Ex. 8, we have $\Pi_n(i) = 1$ and $\Pi'_n(i) = \sum_{k=0}^{n-1} i^k$ for all $n \in \mathbb{N}$ (see Ex. 18). We also have $n_1 = 2$ and the $(r_1, r_2, \Rightarrow_P)$-chain:

$$\underbrace{\mathsf{f}(\mathsf{s}(0), 0)}_{a_0} \overset{\Pi'_0(n_1)}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}(0), 0) \underset{r_2}{\Rightarrow} \underbrace{\mathsf{f}(\mathsf{s}(0), \mathsf{s}(0))}_{a_1} \overset{\Pi'_1(n_1)}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}^3(0), 0) \underset{r_2}{\Rightarrow} \underbrace{\mathsf{f}(\mathsf{s}(0), \mathsf{s}^3(0))}_{a_2} \overset{\Pi'_2(n_1)}{\underset{r_1}{\Rightarrow}} \cdots$$

▶ **Example 22.** In Ex. 9, we have $\Pi_0(n_1) = 0$, $\Pi'_0(n_1) = 1$, $\Pi_1(n_1) = 1$, $\Pi'_1(n_1) = 2$, $\Pi_2(n_1) = 3$, $\Pi'_2(i) = 4$, ... (see Ex. 16). We have the $(r_1, r_2, \Rightarrow_P)$-chain:

$$\underbrace{\mathsf{f}(0, \mathsf{s}(0))}_{a_0} \overset{\Pi'_0(n_1)}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}(0), 0) \underset{r_2}{\Rightarrow} \underbrace{\mathsf{f}(\mathsf{s}(0), \mathsf{s}^2(0))}_{a_1} \overset{\Pi'_1(n_1)}{\underset{r_1}{\Rightarrow}} \mathsf{f}(\mathsf{s}^3(0), 0) \underset{r_2}{\Rightarrow} \underbrace{\mathsf{f}(\mathsf{s}^3(0), \mathsf{s}^4(0))}_{a_2} \overset{\Pi'_2(n_1)}{\underset{r_1}{\Rightarrow}} \cdots$$

4 Future Work and Implementation

We plan to investigate how our work relates to the forms of non-termination detected by the approaches of [5, 7, 12]. We have no clear idea for the moment.

Our tool NTI (Non-Termination Inference) [9] is designed to automatically prove the existence of infinite chains in TRSs and in LPs. It first transforms the original program P into a program P': for TRSs, it uses the dependency pairs combined with a variant of the overlap closure [10] and, for LPs, it uses the binary unfolding [4, 6]. By [2, 4, 8], non-termination of P' implies that of P. Then, it detects recurrent pairs (Def. 10), hence binary chains (Prop. 20), in P'.

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.

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 F. Emmes, T. Enger, and J. Giesl. Proving non-looping non-termination automatically. In B. Gramlich, D. Miller, and U. Sattler, editors, *Proc. of the 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, volume 7364 of *LNCS*, pages 225–240. Springer, 2012. doi:10.1007/978-3-642-31365-3_19.

6 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 the 1994 ACM Symposium on Applied Computing (SAC'94)*, pages 394–399. ACM Press, 1994. doi:10.1145/326619.326789.

7 A. Geser and H. Zantema. Non-looping string rewriting. *RAIRO Theoretical Informatics and Applications*, 33(3):279–302, 1999. doi:10.1051/ita:1999118.

8 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.

9 NTI (Non-Termination Inference). http://lim.univ-reunion.fr/staff/epayet/Research/NTI/NTI.html and https://github.com/etiennepayet/nti.

10 É. Payet. Guided unfoldings for finding loops in standard term rewriting. In F. Mesnard and P. J. Stuckey, editors, *Proc. of the 28th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'18), Revised Selected Papers*, volume 11408 of *LNCS*, pages 22–37. Springer, 2018. doi:10.1007/978-3-030-13838-7_2.

11 Termination Problems Data Base. http://termination-portal.org/wiki/TPDB.

12 Y. Wang and M. Sakai. On non-looping term rewriting. In A. Geser and H. Søndergaard, editors, *Proc. of the 8th International Workshop on Termination (WST'06)*, pages 17–21, 2006.

13 H. Zantema and A. Geser. *Non-looping rewriting.* Universiteit Utrecht. UU-CS, Department of Computer Science. Utrecht University, Netherlands, 1996.