Journal of Automated Reasoning (2024) 68:4
https://doi.org/10.1007/s10817-023-09693-z

Non-termination in Term Rewriting and Logic Programming

Étienne Payet[1]

Received: 30 November 2021 / Accepted: 21 December 2023 / Published online: 2 February 2024

Abstract
In this paper, we define two particular forms of non-termination, namely *loops* and *binary chains*, in an abstract framework that encompasses term rewriting and logic programming. The definition of loops relies on the notion of *compatibility* of binary relations. We also present a syntactic criterion for the detection of a special case of binary chains. Moreover, we describe our implementation NTI and compare its results at the Termination Competition 2023 with those of leading analyzers.

Keywords Abstract reduction systems · Term rewriting · Logic programming · Non-termination · Loop

1 Introduction

This paper is concerned with the abstract treatment of non-termination in structures where one rewrites elements using indexed binary relations. Such structures can be formalised by *abstract reduction systems* (ARSs) [2, 30], *i.e.*, pairs $(A, \Rightarrow_\Pi)$ where A is a set and $\Rightarrow_\Pi$ (the rewrite relation) is a union of binary relations on A, indexed by a set Π, *i.e.*, $\Rightarrow_\Pi = \bigcup\{\Rightarrow_\pi \mid \pi \in \Pi\}$. Non-termination in this context corresponds to the existence of an infinite rewrite sequence $a_0 \Rightarrow_{\pi_0} a_1 \Rightarrow_{\pi_1} \cdots$. In this introduction, we provide an extended, intuitive, description of the context and content of the paper, with several examples. In Sect. 1.1, we introduce two concrete instances of ARSs. Then, in Sects. 1.2 and 1.3, we consider two special forms of non-termination. Finally, in Sect. 1.4, we describe our implementation.

1.1 Term Rewriting and Logic Programming

Term rewriting and *logic programming* are concrete instances of ARSs, where Π indicates what *rule* is applied at what *position*. A crucial difference is that the rewrite relation of term rewriting (denoted by $\rightarrow_\Pi$) relies on instantiation while that of logic programming (denoted by $\rightsquigarrow_\Pi$) relies on narrowing, *i.e.*, on unification.

✉ Étienne Payet
etienne.payet@univ-reunion.fr

1 LIM, Université de la Réunion, Sainte-Clotilde, France

Example 1.1 In term rewriting, rules are pairs of finite terms. In this context, a *term rewrite system (TRS)* is a set of rules over a given signature (*i.e.*, a set of function symbols). Consider for instance the TRS that consists of the rules:

$$r_1 = \big(\mathsf{f}(x), \mathsf{g}(\mathsf{h}(x, 1), x)\big)$$
$$r_2 = \big(1, 0\big)$$
$$r_3 = \big(\mathsf{h}(x, 0), \mathsf{f}^2(x)\big) = (u_3, v_3)$$

Here, g, h (resp. f) are function symbols of arity 2 (resp. 1), 0, 1 are constant symbols (*i.e.*, function symbols of arity 0), x is a variable and f^2 denotes 2 successive applications of f. Consider the term $s = \mathsf{g}\big(\mathsf{h}(\mathsf{f}(x), 0), x\big)$. Its subterm $s' = \mathsf{h}(\mathsf{f}(x), 0)$ results from applying the substitution $\theta = \{x \mapsto \mathsf{f}(x)\}$ to u_3, denoted as $s' = u_3\theta$, hence it is an instance of u_3. Therefore, we have $s \rightarrow_{(r_3, p)} \mathsf{g}(v_3\theta, x)$, where p is the position of s' in s and $\mathsf{g}(v_3\theta, x)$ results from replacing s' by $v_3\theta$ in s, *i.e.*, $s \rightarrow_{(r_3, p)} \mathsf{g}(\mathsf{f}^3(x), x)$.

Example 1.2 In logic programming, one rewrites goals (*i.e.*, finite sequences of terms) into goals and a rule is a pair $(u, \overline{v})$ where u is a term and $\overline{v}$ is a goal. Moreover, the rewriting always takes place at the root position of an element of a goal. In this context, a *logic program (LP)* is a set of rules over a given signature.

Consider for instance the LP that consists of the rule:

$$r = \big(\underbrace{\mathsf{p}(\mathsf{f}(x, 0))}_{u}, \underbrace{\langle \mathsf{p}(x), \mathsf{q}(x)\rangle}_{\overline{v}}\big)$$

Here, p, q (resp. f) are function symbols of arity 1 (resp. 2), 0 is a constant symbol and x is a variable. Consider the goal $\overline{s} = \langle \mathsf{p}(0), \mathsf{p}(\mathsf{f}(x, x)), \mathsf{p}(x)\rangle$. The rule $(u_1, \langle v_1, v_1'\rangle) = \big(\mathsf{p}(\mathsf{f}(x_1, 0)), \langle \mathsf{p}(x_1), \mathsf{q}(x_1)\rangle\big)$ is obtained by renaming the variables of r with variables not occurring in $\overline{s}$. The term $s_1 = \mathsf{p}(\mathsf{f}(x, x))$ is an element of $\overline{s}$ and the substitution $\theta = \{x \mapsto 0, x_1 \mapsto 0\}$ is a unifier of s_1 and u_1, *i.e.*, $s_1\theta = u_1\theta$ (to be precise, θ is *the most general* unifier of s_1 and u_1, see Sect. 2.4). Therefore, we have $\overline{s} \rightsquigarrow_{(r,p)} \langle \mathsf{p}(0), v_1, v_1', \mathsf{p}(x)\rangle\theta$, where p is the position of s_1 in $\overline{s}$ and $\langle \mathsf{p}(0), v_1, v_1', \mathsf{p}(x)\rangle$ is the goal obtained from $\overline{s}$ by replacing s_1 by the elements of $\langle v_1, v_1'\rangle$, *i.e.*, $\overline{s} \rightsquigarrow_{(r,p)} \langle \mathsf{p}(0), \mathsf{p}(0), \mathsf{q}(0), \mathsf{p}(0)\rangle$. The operation which is used to rewrite $\overline{s}$ into $\langle \mathsf{p}(0), v_1, v_1', \mathsf{p}(x)\rangle\theta$ is called *narrowing*. In this example, it is applied at a given position of a goal using an LP rule, but it can be applied similarly at a position p of a term s using a TRS rule (*i.e.*, renaming of the rule with new variables, unification of the subterm of s at position p with the left-hand side of the renamed rule, replacement of the subterm by the right-hand side of the renamed rule and application of the most general unifier to the whole resulting term).

Another difference between term rewriting and logic programming is the effect of a rule application outside the position of the application. For instance, in Example 1.1, the application of r_3 at position p of s only affects the subterm of s at position p. In contrast, in Example 1.2, the application of r at position p of $\overline{s}$ also affects the terms at the other positions, because first s_1 is replaced by v_1 and v_1' and then the unifier θ is applied to the whole resulting goal.

Due to these differences, termination in logic programming does not seem related to termination in term rewriting, but rather to termination in the construction of the *overlap closure* [14], because both use narrowing. The overlap closure of a TRS $\mathcal{R}$, denoted as $OC(\mathcal{R})$, is the (possibly infinite) set defined inductively as follows. It exactly consists of the rules of $\mathcal{R}$, together with the rules resulting from narrowing elements of $OC(\mathcal{R})$ with elements of $OC(\mathcal{R})$. More precisely, suppose that (u_1, v_1) and (u_2, v_2) are in $OC(\mathcal{R})$. Then,

- (Forward closure) $(u_1\theta, v_1') \in OC(\mathcal{R})$ where $\big(v_1 \rightsquigarrow_{((u_2,v_2),p)} v_1'\big)$, θ is the unifier used and p is the position of a non-variable subterm of v_1;
- (Backward closure) $(u_2', v_2\theta) \in OC(\mathcal{R})$ where $\big(u_2 \rightsquigarrow_{((v_1,u_1),p)} u_2'\big)$, θ is the unifier used and p is the position of a non-variable subterm of u_2.

Consider for instance the TRS $\mathcal{R}$ that consists of the rule $r = \big(\mathsf{f}(\mathsf{s}(x), y), \mathsf{f}(x, \mathsf{s}(y))\big)$. It admits no infinite rewrite sequence, but the construction of its overlap closure will not stop as it produces each $\big(\mathsf{f}(\mathsf{s}^k(x), y), \mathsf{f}(x, \mathsf{s}^k(y))\big)$, $k > 0$ (where s^k denotes k successive applications of the unary function symbol s). Now, consider the LP that consists of r (with delimiters $\langle\cdot\rangle$ around the right-hand side). It admits the infinite rewrite

$$\langle \underbrace{\mathsf{f}(x, y)}_{a_0} \rangle \overset{\theta_1}{\underset{\pi}{\rightsquigarrow}} \langle \underbrace{\mathsf{f}(x_1, \mathsf{s}(y))}_{a_1} \rangle \overset{\theta_2}{\underset{\pi}{\rightsquigarrow}} \langle \underbrace{\mathsf{f}(x_2, \mathsf{s}^2(y))}_{a_2} \rangle \overset{\theta_3}{\underset{\pi}{\rightsquigarrow}} \langle \underbrace{\mathsf{f}(x_3, \mathsf{s}^3(y))}_{a_3} \rangle \overset{\theta_4}{\underset{\pi}{\rightsquigarrow}} \cdots$$

where $\pi = (r, \langle 1\rangle)$ ($\langle 1\rangle$ is the position of the leftmost term of a goal) and each step is decorated with the corresponding unifier:

$$\begin{aligned} \theta_1 &= \{x \mapsto \mathsf{s}(x_1), y_1 \mapsto y\} \\ \theta_2 &= \{x_1 \mapsto \mathsf{s}(x_2), y_2 \mapsto \mathsf{s}(y)\} \\ \theta_3 &= \{x_2 \mapsto \mathsf{s}(x_3), y_3 \mapsto \mathsf{s}^2(y)\} \\ &\ \vdots \end{aligned}$$

This rewrite sequence corresponds to the infinite construction of the overlap closure of $\mathcal{R}$. Indeed, for all $k > 0$, the pair $(a_0\theta_1 \ldots \theta_k, a_k\theta_1 \ldots \theta_k)$ corresponds to the rule $\big(\mathsf{f}(\mathsf{s}^k(x), y), \mathsf{f}(x, \mathsf{s}^k(y))\big)$:

$$\begin{aligned} (a_0\theta_1, a_1\theta_1) &= (\mathsf{f}(\mathsf{s}(x_1), y), \mathsf{f}(x_1, \mathsf{s}(y))) \\ (a_0\theta_1\theta_2, a_2\theta_1\theta_2) &= (\mathsf{f}(\mathsf{s}^2(x_2), y), \mathsf{f}(x_2, \mathsf{s}^2(y))) \\ &\ \vdots \end{aligned}$$

1.2 Loops

The vast majority of the papers dealing with non-termination in term rewriting provide conditions for the existence of *loops*. While sufficient conditions are generally used to design loop-detection approaches [11, 21, 33, 36, 37], necessary conditions are rather used to prove the absence of loops [9, 38]. In this context, a loop refers to a finite rewrite sequence $a_0 \rightarrow_\Pi \cdots \rightarrow_\Pi a_n$ where a_n embeds an instance of a_0. It is well known that then, one can go on from a_n, *i.e.*, there is a finite rewrite sequence $a_n \rightarrow_\Pi \cdots \rightarrow_\Pi a_{n+m}$ where a_{n+m} embeds an instance of a_n, hence an instance of a_0, and so on. Therefore, there is an infinite rewrite sequence starting from a_0 and in which a_0 is "reached" periodically (see Example 1.3 below).

The situation in logic programming is different, in the sense that there is no "official" definition of loops (at least, as far as we know). In contrast to term rewriting, where loopingness is a special form of non-termination, logic programming loops simply seem to correspond to infinite rewrite sequences, with no more precision. In this context, [25, 27] provide sufficient conditions for the existence of loops, while [3, 26–28] provide necessary conditions. The former are generally used to design static (*i.e.*, compile-time) loop-detection approaches, while the latter are used to design dynamic (*i.e.*, runtime) loop checks (*i.e.*, rewrite sequences are pruned at runtime when it seems appropriate, with a risk of pruning a finite rewrite).

In this paper, we provide a generic definition of loops in any ARS $(A, \Rightarrow_\Pi)$. It generalises TRS loops by considering any *compatible* relation ϕ, not necessarily the "embeds an instance of" relation (see Definition 4.1). Compatibility means that if $a \Rightarrow_\Pi a_1$ and a' is related to a via ϕ (denoted as $a' \in \phi(a)$) then we have $a' \Rightarrow_\Pi a'_1$ for some $a'_1 \in \phi(a_1)$ (see Definition 2.2).

Example 1.3 In Example 1.1, we have the rewrite sequence

$$\underbrace{\mathsf{f}(x)}_{a_0} \mathrel{\mathop{\rightarrow}_{(r_1,p_0)}} \underbrace{\mathsf{g}(\mathsf{h}(x,1),x)}_{a_1} \mathrel{\mathop{\rightarrow}_{(r_2,p_1)}} \underbrace{\mathsf{g}(\mathsf{h}(x,0),x)}_{a_2} \mathrel{\mathop{\rightarrow}_{(r_3,p_2)}} \underbrace{\mathsf{g}(\mathsf{f}^2(x),x)}_{a_3} \tag{1}$$

where $p_0 = \epsilon$ (*i.e.*, the *root position*, which is the position of a term inside itself), p_1 is the position of 1 in a_1 and p_2 is the position of $\mathsf{h}(x, 0)$ in a_2. Consider ϕ that relates any term s to the terms that embed an instance of s, *i.e.*, of the form $c[s\theta]$, where c is a context (a term with a hole in it), θ a substitution and $c[s\theta]$ the term resulting from replacing the hole in c by $s\theta$. It is well known that ϕ is compatible with the rewrite relation of term rewriting, *i.e.*, $s' \in \phi(s)$ and $s \rightarrow_{(r,p)} t$ imply that $s' \rightarrow_{(r,p')} t'$ for some position p' and some $t' \in \phi(t)$. We have $a_3 \in \phi(a_0)$ because $a_3 = c[a_0\theta]$ for $\theta = \{x \mapsto \mathsf{f}(x)\}$ and $c = \mathsf{g}(\square, x)$, where $\square$ is the hole. So, (1) is a loop and, by the compatibility property, there is an infinite rewrite sequence

$$a_0 \mathrel{\mathop{\rightarrow}_{(r_1,p_0)}} a_1 \mathrel{\mathop{\rightarrow}_{(r_2,p_1)}} a_2 \mathrel{\mathop{\rightarrow}_{(r_3,p_2)}} \underbrace{a_3}_{\in\phi(a_0)} \mathrel{\mathop{\rightarrow}_{(r_1,p_3)}} \underbrace{a_4}_{\in\phi(a_1)} \mathrel{\mathop{\rightarrow}_{(r_2,p_4)}} \underbrace{a_5}_{\in\phi(a_2)} \mathrel{\mathop{\rightarrow}_{(r_3,p_5)}} \underbrace{a_6}_{\in\phi(a_3)} \mathrel{\mathop{\rightarrow}_{(r_1,p_6)}} \cdots$$

which is a succession of loops.

Example 1.4 In Example 1.2, we have the rewrite sequence

$$\underbrace{\langle \mathsf{p}(\mathsf{f}(x,0))\rangle}_{\overline{a_0}} \mathrel{\mathop{\rightsquigarrow}_{(r,\langle 1\rangle)}} \underbrace{\langle \mathsf{p}(x), \mathsf{q}(x)\rangle}_{\overline{a_1}} \tag{2}$$

Consider ϕ that relates any goal $\overline{s}$ to the goals that embed a more general goal than $\overline{s}$, *i.e.*, of the form $\overline{c}[\overline{t}]$, where $\overline{c}$ is a context (a goal with a hole in it) and $\overline{t}$ is more general than $\overline{s}$, *i.e.*, $\overline{s} = \overline{t}\theta$ for some substitution θ (hence, $\overline{s}$ is an instance of $\overline{t}$). We prove in this paper (Lemma 2.17) that ϕ is compatible with the rewrite relation of logic programming, *i.e.*, $\overline{s'} \in \phi(\overline{s})$ and $\overline{s} \rightsquigarrow_{(r,p)} \overline{t}$ imply that $\overline{s'} \rightsquigarrow_{(r,p')} \overline{t'}$ for some position p' and some $\overline{t'} \in \phi(\overline{t})$. We have $\overline{a_1} \in \phi(\overline{a_0})$ because $\overline{a_1} = \overline{c}[\overline{b_0}]$ where $\overline{c} = \langle \square, \mathsf{q}(x)\rangle$ and $\overline{b_0} = \langle \mathsf{p}(x)\rangle$ is more general than $\overline{a_0}$ (we have $\overline{a_0} = \overline{b_0}\{x \mapsto \mathsf{f}(x, 0)\}$). So, (2) is a loop and, by the compatibility property, there is an infinite rewrite sequence

$$\overline{a_0} \mathrel{\mathop{\rightsquigarrow}_{(r,\langle 1\rangle)}} \underbrace{\overline{a_1}}_{\in\phi(\overline{a_0})} \mathrel{\mathop{\rightsquigarrow}_{(r,p_1)}} \underbrace{\overline{a_2}}_{\in\phi(\overline{a_1})} \mathrel{\mathop{\rightsquigarrow}_{(r,p_2)}} \underbrace{\overline{a_3}}_{\in\phi(\overline{a_2})} \mathrel{\mathop{\rightsquigarrow}_{(r,p_3)}} \cdots$$

which is a succession of loops.

1.3 Binary Chains

The infinite rewrite sequence of Example 1.3 (resp. Example 1.4) is a succession of loops that all involve the same sequence $\langle r_1, r_2, r_3\rangle$ (resp. $\langle r\rangle$) of rules. In this paper, we also consider infinite rewrite sequences that involve two sequences of rules. We call them *binary chains* (see Definition 5.1).

Example 1.5 Consider the contexts $c = \mathsf{g}(\square, \mathsf{0}, \square)$ and $c' = \mathsf{g}(\square, \mathsf{1}, \square)$ and the TRS that consists of the following rules:

$$r_1 = \big(\mathsf{f}(x, c[y], x), \mathsf{h}(x, y)\big) \qquad r_3 = \big(\mathsf{f}(x, \mathsf{0}, x), \mathsf{f}(c[x], c'[x], c[x])\big)$$
$$r_2 = \big(\mathsf{h}(x, y), \mathsf{f}(c[x], y, c[x])\big) \qquad r_4 = \big(\mathsf{1}, \mathsf{0}\big)$$

We have the infinite rewrite sequence

$$\begin{aligned}
&\underbrace{\mathsf{f}\big(c[\mathsf{0}], c[\mathsf{0}], c[\mathsf{0}]\big)}_{a_0} \underset{(r_1,\epsilon)}{\rightarrow} \mathsf{h}\big(c[\mathsf{0}], \mathsf{0}\big) \underset{(r_2,\epsilon)}{\rightarrow} \mathsf{f}\big(c^2[\mathsf{0}], \mathsf{0}, c^2[\mathsf{0}]\big) \\
&\quad \underset{(r_3,\epsilon)}{\rightarrow} \underbrace{\mathsf{f}\big(c^3[\mathsf{0}], c'[c^2[\mathsf{0}]], c^3[\mathsf{0}]\big)}_{a'_0} \underset{(r_4,p_0)}{\rightarrow} \underbrace{\mathsf{f}\big(c^3[\mathsf{0}], c^3[\mathsf{0}], c^3[\mathsf{0}]\big)}_{a_1} \\
&\quad \underset{(r_1,\epsilon)}{\rightarrow} \mathsf{h}\big(c^3[\mathsf{0}], c^2[\mathsf{0}]\big) \underset{(r_2,\epsilon)}{\rightarrow} \mathsf{f}\big(c^4[\mathsf{0}], c^2[\mathsf{0}], c^4[\mathsf{0}]\big) \\
&\quad \underset{(r_1,\epsilon)}{\rightarrow} \mathsf{h}\big(c^4[\mathsf{0}], c[\mathsf{0}]\big) \underset{(r_2,\epsilon)}{\rightarrow} \mathsf{f}\big(c^5[\mathsf{0}], c[\mathsf{0}], c^5[\mathsf{0}]\big) \\
&\quad \underset{(r_1,\epsilon)}{\rightarrow} \mathsf{h}\big(c^5[\mathsf{0}], \mathsf{0}\big) \underset{(r_2,\epsilon)}{\rightarrow} \mathsf{f}\big(c^6[\mathsf{0}], \mathsf{0}, c^6[\mathsf{0}]\big) \\
&\quad \underset{(r_3,\epsilon)}{\rightarrow} \underbrace{\mathsf{f}\big(c^7[\mathsf{0}], c'[c^6[\mathsf{0}]], c^7[\mathsf{0}]\big)}_{a'_1} \underset{(r_4,p_1)}{\rightarrow} \underbrace{\mathsf{f}\big(c^7[\mathsf{0}], c^7[\mathsf{0}], c^7[\mathsf{0}]\big)}_{a_2} \\
&\quad \underset{(r_1,\epsilon)}{\rightarrow} \cdots
\end{aligned}$$

where c^n denotes n embeddings of c into itself and p_n is the position of $\mathsf{1}$ (occurring in c') in a'_n, for all n in the set $\mathbb{N}$ of non-negative integers. We note that for all $n \in \mathbb{N}$, from a_n to a_{n+1} the sequence $w_1 = \langle r_1, r_2 \rangle$ is applied several times and then the sequence $w_2 = \langle r_3, r_4 \rangle$ is applied exactly once. This is written as $a_n \left(\rightarrow^*_{w_1} \circ \rightarrow_{w_2}\right) a_{n+1}$, and hence the infinite rewrite above relies on the two sequences of rules w_1 and w_2. In Sect. 5.1, we present a syntactic criterion for detecting a special case of this form of non-termination.

1.4 Implementation

Our tool NTI (Non-Termination Inference) [24] performs automatic proofs of non-termination in term rewriting and logic programming. It specifically considers logic programming with Prolog's leftmost selection rule, *i.e.*, always the leftmost term is chosen in a goal for narrowing (contrary to Example 1.2, where s_1 is not the leftmost term of $\overline{s}$). It applies program transformation techniques as the very first step, namely, the *dependency pair unfolding* [22] in term rewriting and the *binary unfolding* [4, 8] in logic programming. The idea is to compute "compressed" forms of finite rewrite sequences that can be used in a simplified setting not polluted by technicalities, such as deeper and deeper embedding of terms (*e.g.*, in Example 1.3, a_3 embeds an instance of a_0, a_6 embeds an instance of a_3, ...) This allows us to detect loops using a simplified version of the relations ϕ of Examples 1.3 and 1.4, and also to concentrate on rewrite sequences consisting of only one step (as they are "compressions" of longer sequences), see Examples 1.7, 1.8 and 1.10 below.

1.4.1 Dependency Pair Unfolding

The defined symbols of a TRS $\mathcal{R}$ are the function symbols f for which there is a rule of the form $(\mathsf{f}(\dots), \dots)$ in $\mathcal{R}$. Each defined symbol f is associated with a symbol F of the same arity that does not occur in $\mathcal{R}$. The set of dependency pairs of $\mathcal{R}$ is

$$\left\{ \big(\mathsf{F}(s_1,\dots,s_n),\mathsf{G}(t_1,\dots,t_m)\big) \;\middle|\; \begin{array}{l} \big(\mathsf{f}(s_1,\dots,s_n),t\big)\in\mathcal{R} \\ \mathsf{g}(t_1,\dots,t_m) \text{ is a subterm of } t \\ \mathsf{g} \text{ is a defined symbol of } \mathcal{R} \end{array} \right\}$$

The technique of [22] transforms a TRS $\mathcal{R}$ into a (possibly infinite) set $\mathit{Unf}(\mathcal{R})$ inductively defined as follows. It exactly consists of the dependency pairs of $\mathcal{R}$, together with the rules resulting from narrowing (forwards and backwards, as for $OC(\mathcal{R})$) elements of $\mathit{Unf}(\mathcal{R})$ with rules and dependency pairs of $\mathcal{R}$; while dependency pairs are only applied at the root position, between the application of two dependency pairs there can be an arbitrary number of narrowing steps below the root, using rules of $\mathcal{R}$. The computation of $\mathit{Unf}(\mathcal{R})$ is hence very similar to that of $OC(\mathcal{R})$, except that: it also considers the dependency pairs, it only uses the initial dependency pairs and rules for narrowing elements of $\mathit{Unf}(\mathcal{R})$ and it allows narrowing of variable subterms.

Theorem 1.6 [22] *Let $\mathcal{R}$ be a TRS. If a term $\mathsf{F}(s_1,\dots,s_n)$ starts an infinite rewrite sequence w.r.t. $\mathit{Unf}(\mathcal{R})$ then the term $\mathsf{f}(s_1,\dots,s_n)$ starts an infinite rewrite sequence w.r.t. $\mathcal{R}$.*

Note that this theorem only states an implication, not an equivalence, contrary to Theorem 1.9 below in logic programming (where an "if and only if" is stated). Therefore, we do not know whether it is restrictive to detect infinite rewrite sequences w.r.t. $\mathit{Unf}(\mathcal{R})$, instead of $\mathcal{R}$ (*i.e.*, we do not know whether the existence of an infinite rewrite sequence w.r.t. $\mathcal{R}$ necessarily implies that of an infinite rewrite sequence w.r.t. $\mathit{Unf}(\mathcal{R})$).

Example 1.7 (Example 1.3 continued) Let $\mathcal{R}$ be the TRS under consideration. The following rules are dependency pairs of $\mathcal{R}$ obtained from r_1 and r_3 respectively:

$$r'_1 = \big(\mathsf{F}(x),\mathsf{H}(x,1)\big) = (u'_1, v'_1) \qquad r'_3 = \big(\mathsf{H}(x,0),\mathsf{F}(\mathsf{f}(x))\big)$$

Here, r'_1 allows us to get rid of the context $c = \mathsf{g}(\square, x)$ that occurs in r_1. Let us simplify ϕ into the relation ϕ' that binds any term to its instances (so, compared to ϕ, there are no more contexts). Of course, ϕ' is also compatible with the rewrite relation of term rewriting.

We have $v'_1 \rightsquigarrow_{(r_2,p')} \mathsf{H}(x,0)$ where p' is the position of 1 in v'_1; so, from r'_1 and r_2, the technique of [22] produces $r''_1 = \big(\mathsf{F}(x),\mathsf{H}(x,0)\big) = (u''_1, v''_1)$. Moreover, $v''_1 \rightsquigarrow_{(r'_3,\epsilon)} \mathsf{F}(\mathsf{f}(x))$ so, from r''_1 and r'_3, we get $r = \big(\mathsf{F}(x),\mathsf{F}(\mathsf{f}(x))\big)$.

Let $\pi = (r,\epsilon)$. We have the following loop w.r.t. ϕ':

$$\overbrace{\mathsf{F}(x)}^{a'_0} \underset{\pi}{\rightarrow} \overbrace{\mathsf{F}(\mathsf{f}(x))}^{a'_1 \in \phi'(a'_0)}$$

Hence, there is an infinite rewrite sequence $a'_0 \rightarrow_\pi a'_1 \rightarrow_\pi a'_2 \rightarrow_\pi \cdots$ with $a'_1 \in \phi'(a'_0)$, $a'_2 \in \phi'(a'_1)$, ... It corresponds to that of Example 1.3, but each successive application of r_1, r_2, r_3 has been "compressed" into a single application of r. Moreover, it does not involve c, as well as the context that embeds an instance of a_1 in a_4, *etc*. By Theorem 1.6, $\mathsf{f}(x)$ starts an infinite rewrite sequence w.r.t. $\mathcal{R}$.

Example 1.8 The following dependency pairs are obtained respectively from the rules r_1, r_2 and r_3 of the TRS $\mathcal{R}$ of Example 1.5:

$$\begin{aligned} r'_1 &= \big(\mathsf{F}(x, c[y], x), \mathsf{H}(x,y)\big) \\ r'_2 &= \big(\mathsf{H}(x,y), \mathsf{F}(c[x], y, c[x])\big) \\ r'_3 &= \big(\mathsf{F}(x,0,x), \mathsf{F}(c[x], c'[x], c[x])\big) \end{aligned}$$

From r'_1 and r'_2, the technique of [22] produces $r''_1 = \big(\mathsf{F}(x, c[y], x), \mathsf{F}(c[x], y, c[x])\big)$ and, from r'_3 and r_4, it produces $r''_3 = \big(\mathsf{F}(x, 0, x), \mathsf{F}(c[x], c[x], c[x])\big)$. Let $\pi_1 = (r''_1, \epsilon)$, $\pi_3 = (r''_3, \epsilon)$ and $A_n = \mathsf{F}\big(c^n[0], c^n[0], c^n[0]\big)$ for all $n \in \mathbb{N}$. We have the binary chain:

$$A_1 \underset{\pi_1}{\rightarrow} \mathsf{F}\big(c^2[0], 0, c^2[0]\big) \underset{\pi_3}{\rightarrow} A_3 \overset{3}{\underset{\pi_1}{\rightarrow}} \mathsf{F}\big(c^6[0], 0, c^6[0]\big) \underset{\pi_3}{\rightarrow} A_7 \overset{7}{\underset{\pi_1}{\rightarrow}} \cdots$$

It corresponds to that of Example 1.5, but each application of w_1 (resp. w_2) has been "compressed" into an application of r''_1 (resp. r''_3). By Theorem 1.6, $\mathsf{f}\big(c[0], c[0], c[0]\big)$ starts an infinite rewrite sequence w.r.t. $\mathcal{R}$.

1.4.2 Binary Unfolding

The binary unfolding [4, 8] is a program transformation technique that has been introduced in the context of Prolog's leftmost selection rule. It transforms a LP P into a (possibly infinite) set $\mathit{Binunf}(P)$ of *binary* rules (*i.e.*, rules, the right-hand side of which is a goal that contains at most one term) inductively defined as follows: $\mathit{Binunf}(P)$ exactly consists of the rules constructed by narrowing prefixes of right-hand sides of rules of P using elements of $\mathit{Binunf}(P)$; more precisely, for all $(u, \langle v_1, \ldots, v_n\rangle) \in P$:

(A) for each $1 \le i \le n$, the goals $\langle v_1\rangle, \ldots, \langle v_{i-1}\rangle$ are narrowed, respectively, with $(u_1, \epsilon), \ldots, (u_{i-1}, \epsilon)$ from $\mathit{Binunf}(P)$ (where ϵ is the empty goal) to obtain a corresponding instance of $(u, \langle v_i\rangle)$,

(B) for each $1 \le i \le n$, the goals $\langle v_1\rangle, \ldots, \langle v_{i-1}\rangle$ are narrowed, respectively, with $(u_1, \epsilon), \ldots, (u_{i-1}, \epsilon)$ from $\mathit{Binunf}(P)$ and $\langle v_i\rangle$ is also narrowed with $(u_i, \langle v\rangle)$ from $\mathit{Binunf}(P)$ to obtain a corresponding instance of $(u, \langle v\rangle)$,

(C) $\langle v_1\rangle, \ldots, \langle v_n\rangle$ are narrowed, respectively, with $(u_1, \epsilon), \ldots, (u_n, \epsilon)$ from $\mathit{Binunf}(P)$ to obtain a corresponding instance of (u, ϵ).

Intuitively, each generated binary rule $(u, \langle v\rangle)$ specifies that, w.r.t. the leftmost selection rule, $\langle u\rangle$ can be rewritten, using rules of P, to a goal $\langle v, \ldots\rangle$.

Theorem 1.9 [4] *Let P be a LP and $\overline{s_0}$ be a goal. Assume that the leftmost selection rule is used. Then, $\overline{s_0}$ starts an infinite rewrite sequence w.r.t. P* if and only if *it starts an infinite rewrite sequence w.r.t. $\mathit{Binunf}(P)$.*

So, in this context, suppose that $\overline{s_0}$ has the form $\langle v_0, \ldots, v_n\rangle$ and that it starts an infinite rewrite sequence w.r.t. P. Then, by Theorem 1.9, there is an infinite rewrite sequence

$$\overline{s_0} \overset{\theta_0}{\underset{(r_0, p_0)}{\rightsquigarrow}} \overline{s_1} \overset{\theta_1}{\underset{(r_1, p_1)}{\rightsquigarrow}} \cdots \tag{3}$$

where $r_0, r_1, \ldots$ are rules of $\mathit{Binunf}(P)$ (we decorate each step with the corresponding unifier). As the leftmost selection rule is used, p_i is the position of the leftmost term of $\overline{s_i}$, for all $i \in \mathbb{N}$. So, necessarily, there is a finite (possibly empty) prefix of (3) that "erases" a (possibly empty) prefix of $\overline{s_0}$, *i.e.*, it has the form $\overline{s_0} \overset{\theta_0}{\rightsquigarrow}_{(r_0, p_0)} \cdots \overset{\theta_{m-1}}{\rightsquigarrow}_{(r_{m-1}, p_{m-1})} \overline{s_m}$ with $\overline{s_m} = \langle v_k, \ldots, v_n\rangle\theta_0 \ldots \theta_{m-1}$ and $\overline{t_m} = \langle v_k\theta_0 \ldots \theta_{m-1}\rangle$ starts an infinite rewrite sequence of the form $\overline{t_m} \rightsquigarrow_{(r_m, p'_m)} \overline{t_{m+1}} \rightsquigarrow_{(r_{m+1}, p'_{m+1})} \cdots$. As the goal $\overline{t_m}$ is a singleton and the rules $r_m, r_{m+1}, \ldots$ are binary, we necessarily have that $\overline{t_{m+1}}, \overline{t_{m+2}}, \ldots$ are singletons and $p'_m = p'_{m+1} = \cdots = \langle 1\rangle$ (because, in logic programming, rewriting only takes place at the root position of a term of a goal).

So, from the non-termination point of view, and as far as Prolog's leftmost selection rule is concerned, it is not restrictive to consider a simplified form of logic programming where one only rewrites singleton goals using binary rules. Indeed, we have shown above that if P is non-terminating in the full setting then $\mathit{Binunf}(P)$ is necessarily non-terminating in this simplified one. Moreover, the leftmost selection rule is the most used by far (*e.g.*, it is the only considered one in the Termination Competition [31]). In Sect. 3, for all LPs P, we define a rewrite relation $\hookrightarrow_P$ that models this simplified form of logic programming. It enjoys an interesting closure property (Lemma 3.6) that we will use to detect binary chains (Sect. 5.1).

Example 1.10 In Example 1.4, the binary unfolding of the LP P under consideration produces the rule $r' = \big(\mathsf{p}(\mathsf{f}(x, 0)), \langle \mathsf{p}(x)\rangle\big)$ from r (using (A) in the definition of $\mathit{Binunf}(P)$ above, with $i = 1$). So, compared to r, we do not have the context $\overline{c} = \langle \square, \mathsf{q}(x)\rangle$ anymore and we can simplify ϕ, *i.e.*, we consider ϕ' that relates any goal $\overline{s}$ to all the goals that are more general than $\overline{s}$. Of course, ϕ' is also compatible with the rewrite relation of LPs. Let $\pi = (r', \langle 1\rangle)$. We have this loop w.r.t. ϕ':

$$\underbrace{\langle \mathsf{p}(\mathsf{f}(x, 0))\rangle}_{\overline{a_0}} \rightsquigarrow_{\pi} \underbrace{\langle \mathsf{p}(x)\rangle}_{\overline{b_0} \in \phi'(\overline{a_0})}$$

Hence, there is an infinite rewrite sequence $\overline{a_0} \rightsquigarrow_{\pi} \overline{b_0} \rightsquigarrow_{\pi} \overline{b_1} \rightsquigarrow_{\pi} \cdots$ where $\overline{b_0} \in \phi'(\overline{a_0})$, $\overline{b_1} \in \phi'(\overline{b_0})$, … It corresponds to that of Example 1.4, but it does not involve the context that embeds a more general goal than $\overline{a_0}$ in $\overline{a_1}$, the context that embeds a more general goal than $\overline{a_1}$ in $\overline{a_2}$, …; moreover, as $\overline{a_0}, \overline{b_0}, \overline{b_1}, \ldots$ consist of only one term, the narrowing steps are all performed at position $\langle 1\rangle$. By Theorem 1.9, $\overline{a_0}$ starts an infinite rewrite sequence w.r.t. P.

1.5 Content of the Paper

After presenting some preliminary material in Sect. 2 and abstract reduction systems in Sect. 3, we describe, in a generic way, loops in Sect. 4 and binary chains in Sect. 5, together with an automatable approach for the detection of the latter. Then, in Sect. 6, we compare our implementation NTI with the tools that participated in the Termination Competition 2023 [31]. Finally, we present related work in Sect. 7 and conclude in Sect. 8.

2 Preliminaries

We introduce some basic notations and definitions. First, in Sect. 2.1, we consider finite sequences and, in Sect. 2.2, binary relations and the related notions of chain and compatibility. Then, in Sects. 2.3 and 2.4, we define terms, contexts, substitutions and unifiers. Finally, in Sect. 2.5, we present term rewriting and logic programming.

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

2.1 Finite Sequences

Let A be a set. We let $\overline{A}$ denote the set of finite sequences of elements A; it includes the empty sequence, denoted as ϵ. We generally (but not always) use the delimiters $\langle\cdot\rangle$ for writing elements of $\overline{A}$. Moreover, we use juxtaposition to denote the concatenation operation, *e.g.*, $\langle a_0, a_1\rangle\langle a_2, a_3\rangle = \langle a_0, a_1, a_2, a_3\rangle$ and $a_0\langle a_1, a_2\rangle = \langle a_0, a_1, a_2\rangle$. The length of $w \in \overline{A}$ is denoted as $|w|$ and is inductively defined as: $|w| = 0$ if $w = \epsilon$ and $|w| = 1 + |w'|$ if $w = aw'$ for some $a \in A$ and $w' \in \overline{A}$.

2.2 Binary Relations

A *binary relation* ψ from a set A to a set B is a subset of $A \times B$. For all $a \in A$, we let $\psi(a) = \{b \in B \mid (a, b) \in \psi\}$. Instead of $(a, b) \in \psi$, we may write $a \; \psi \; b$ (*e.g.*, for binary relations that have the form of an arrow) or $b \in \psi(a)$. We let $\psi^{-1} = \{(b, a) \in B \times A \mid (a, b) \in \psi\}$ be the *inverse* of ψ. A *function* f from A to B is a binary relation from A to B which is such that for all $a \in A$, $f(a)$ consists of exactly one element.

A binary relation ϕ on a set A is a subset of $A^2 = A \times A$. For all $\varphi \subseteq A^2$, we let $\phi \circ \varphi$ denote the *composition* of ϕ and φ:

$$\phi \circ \varphi = \left\{(a, a') \in A^2 \;\middle|\; \exists a_1 \in A \; (a, a_1) \in \phi \wedge (a_1, a') \in \varphi\right\}$$

We let ϕ^0 be the identity relation and, for any $n \in \mathbb{N}$, $\phi^{n+1} = \phi^n \circ \phi$. Moreover, $\phi^+ = \cup\{\phi^n \mid n > 0\}$ (resp. $\phi^* = \phi^0 \cup \phi^+$) denotes the transitive (resp. reflexive and transitive) *closure* of ϕ.

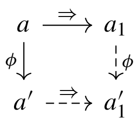
Definition 2.1 Let ϕ be a binary relation on a set A. A ϕ-*chain*, or simply *chain* if ϕ is clear from the context, is a (possibly infinite) sequence $a_0, a_1, \ldots$ of elements of A such that $a_{n+1} \in \phi(a_n)$ for all $n \in \mathbb{N}$. For binary relations that have the form of an arrow, *e.g.*, $\Rightarrow$, we simply write it as $a_0 \Rightarrow a_1 \Rightarrow \cdots$.

The next definition resembles that of a *simulation relation* [20] in a state transition system: $\Rightarrow$ corresponds to the transition relation of the system and ϕ to the simulation.

Definition 2.2 Let A be a set. We say that $\Rightarrow \subseteq A^2$ and $\phi \subseteq A^2$ are *compatible* if for all $a, a', a_1 \in A$ there exists $a'_1 \in A$ such that

$$\left(a' \in \phi(a) \wedge a \Rightarrow a_1\right) \text{ implies } \left(a'_1 \in \phi(a_1) \wedge a' \Rightarrow a'_1\right)$$

Equivalently, $(\phi^{-1} \circ \Rightarrow) \subseteq (\Rightarrow \circ \, \phi^{-1})$. This is illustrated by the following diagram:

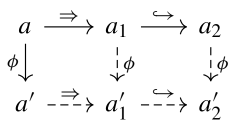


(solid (resp. dashed) arrows represent universal (resp. existential) quantification).

The following result is straightforward.

Lemma 2.3 *Let* $\Rightarrow$, $\hookrightarrow$ *and* ϕ *be binary relations on a set* A. *If* $\Rightarrow$ *and* ϕ *are compatible and* $\hookrightarrow$ *and* ϕ *are compatible then* $(\Rightarrow \circ \hookrightarrow)$ *and* ϕ *are compatible.*

Proof Using the same notations as in Definition 2.2, we have:



□

2.3 Terms

We use the same definitions and notations as [2] for terms.

Definition 2.4 A *signature* Σ is a set of *function symbols*, each element of which has an *arity* in $\mathbb{N}$, which is the number of its arguments. For all $n \in \mathbb{N}$, we denote the set of all n-ary elements of Σ by $\Sigma^{(n)}$. The elements of $\Sigma^{(0)}$ are called *constant symbols*.

Function symbols are denoted by letters or non-negative integers in the *sans serif* font, *e.g.*, f, g or $\mathsf{0}, \mathsf{1}$. We frequently use the superscript notation to denote several successive applications of a unary function symbol, *e.g.*, $\mathsf{s}^3(\mathsf{0})$ is a shortcut for $\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$ and $\mathsf{s}^0(\mathsf{0}) = \mathsf{0}$.

Definition 2.5 Let Σ be a signature and X be a set of *variables* disjoint from Σ. The set $T(\Sigma, X)$ is defined as:

- $X \subseteq T(\Sigma, X)$,
- for all $n \in \mathbb{N}$, all $\mathsf{f} \in \Sigma^{(n)}$ and all $s_1, \dots, s_n \in T(\Sigma, X)$, $\mathsf{f}(s_1, \dots, s_n) \in T(\Sigma, X)$.

For all $s \in T(\Sigma, X)$, we let $Var(s)$ denote the set of variables occurring in s. Moreover, for all $\langle s_1, \dots, s_n \rangle \in \overline{T(\Sigma, X)}$, we let $Var(\langle s_1, \dots, s_n \rangle) = Var(s_1) \cup \cdots \cup Var(s_n)$.

In order to simplify the statement of the definitions and theorems of this paper, from now on we fix a signature Σ, an infinite countable set X of variables disjoint from Σ and a constant symbol $\square$ which does not occur in $\Sigma \cup X$.

Definition 2.6 A *term* is an element of $T(\Sigma, X)$ and a *goal* an element of $\overline{T(\Sigma, X)}$. Moreover, a *context* is an element of $T(\Sigma \cup \{\square\}, X)$ that contains at least one occurrence of $\square$ and a *goal-context* is a finite sequence of the form $\langle s_1, \dots, s_i, \square, s_{i+1}, \dots, s_n \rangle$ where all the s_i's are terms. A context c can be seen as a term with "holes", represented by $\square$, in it. For all $t \in T(\Sigma \cup \{\square\}, X)$, we let $c[t]$ denote the element of $T(\Sigma \cup \{\square\}, X)$ obtained from c by replacing all the occurrences of $\square$ by t. We use the superscript notation for denoting several successive embeddings of a context into itself: $c^0 = \square$ and, for all $n \in \mathbb{N}$, $c^{n+1} = c[c^n]$. Identically, a goal-context $\overline{c}$ can be seen as a goal with a hole, represented by $\square$. For all $\overline{t} \in \overline{T(\Sigma, X)}$, we let $\overline{c}[\overline{t}]$ denote the goal obtained from $\overline{c}$ by replacing $\square$ by the elements of $\overline{t}$.

Terms are generally denoted by a, s, t, u, v, variables by x, y, z and contexts by c, possibly with subscripts and primes. Goals and goal-contexts are denoted using an overbar.

The notion of position in a term is needed to define the operational semantics of term rewriting (see Definition 2.14).

Definition 2.7 The set of *positions* of $s \in T(\Sigma \cup \{\square\}, X)$, denoted as $Pos(s)$, is a subset of $\overline{\mathbb{N}}$ which is inductively defined as:

- if $s \in X$, then $Pos(s) = \{\epsilon\}$,
- if $s = \mathsf{f}(s_1, \dots, s_n)$ then $Pos(s) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip \mid p \in Pos(s_i)\}$.

The position ϵ is called the *root position* of s and the function or variable symbol at this position is the *root symbol*. For all $p \in Pos(s)$, the *subterm of s at position p*, denoted by $s|_p$ is inductively defined as: $s|_\epsilon = s$ and $\mathsf{f}(s_1, \dots, s_n)|_{ip'} = s_i|_{p'}$. Moreover, for all $t \in T(\Sigma, X)$, we denote by $s[t]_p$ the term that is obtained from s by replacing the subterm at position p by t, *i.e.*, $s[t]_\epsilon = t$ and $\mathsf{f}(s_1, \dots, s_n)[t]_{ip'} = \mathsf{f}(s_1, \dots, s_i[t]_{p'}, \dots, s_n)$.

2.4 Substitutions

Definition 2.8 The set $S(\Sigma, X)$ of all *substitutions* consists of the functions θ from X to $T(\Sigma, X)$ such that $\theta(x) \neq x$ for only finitely many variables x. The *domain* of θ is the finite set $Dom(\theta) = \{x \in X \mid \theta(x) \neq x\}$. We usually write θ as $\{x_1 \mapsto \theta(x_1), \ldots, x_n \mapsto \theta(x_n)\}$ where $\{x_1, \ldots, x_n\} = Dom(\theta)$. A *(variable) renaming* is a substitution that is a bijection on X.

The application of a substitution θ to a term or context s is denoted as $s\theta$ and is defined as:

- $s\theta = \theta(s)$ if $s \in X$,
- $s\theta = \mathsf{f}(s_1\theta, \ldots, s_n\theta)$ if $s = \mathsf{f}(s_1, \ldots, s_n)$.

This is extended to goals, *i.e.*, $\langle s_1, \ldots, s_n\rangle\theta = \langle s_1\theta, \ldots, s_n\theta\rangle$.

Definition 2.9 Let $s, t \in T(\Sigma, X)$. We say that t is an *instance* of s if $t = s\theta$ for some $\theta \in S(\Sigma, X)$. Then, we also say that s is *more general than* t. If θ is a renaming, then t is also called a *variant* of s. These definitions straightforwardly extend to all $\overline{s}, \overline{t} \in \overline{T(\Sigma, X)}$.

Definition 2.10 The *composition* of substitutions σ and θ is the substitution denoted as $\sigma\theta$ and defined as: for all $x \in X$, $\sigma\theta(x) = (\sigma(x))\theta$. We say that σ is more general than θ if $\theta = \sigma\eta$ for some substitution η.

The composition of substitutions is an associative operation, *i.e.*, for all terms s and all substitutions σ and θ, $(s\sigma)\theta = s(\sigma\theta)$. We use the superscript notation for denoting several successive compositions of a substitution with itself: $\theta^0 = \emptyset$ (the identity substitution) and, for all $n \in \mathbb{N}$, $\theta^{n+1} = \theta\theta^n = \theta^n\theta$.

Definition 2.11 Let $s, t \in T(\Sigma, X)$. We say that s *unifies* with t (or that s and t unify) if $s\sigma = t\sigma$ for some $\sigma \in S(\Sigma, X)$. Then, σ is called a *unifier* of s and t and $mgu(s, t)$ denotes the *most general unifier* (mgu) of s and t, which is unique (up to variable renaming).

We will frequently refer to the relations "embeds an instance of" (denoted as *ins*) and "embeds a more general term than" (denoted as *mg*) defined as:

Definition 2.12 Using the usual notations for terms (s, t), goals $(\overline{s}, \overline{t})$, contexts (c) and goal-contexts $(\overline{c})$, we define:

$$ins = \{(s, c[t]) \mid t \text{ is an instance of } s\} \cup \left\{\left(\overline{s}, \overline{c}[\overline{t}]\right) \,\middle|\, \overline{t} \text{ is an instance of } \overline{s}\right\}$$
$$mg = \{(s, c[t]) \mid t \text{ is more general than } s\} \cup \left\{\left(\overline{s}, \overline{c}[\overline{t}]\right) \,\middle|\, \overline{t} \text{ is more general than } \overline{s}\right\}$$

2.5 Term Rewriting and Logic Programming

We refer to [2, 30] for the basics of term rewriting and to [1, 17] for those of logic programming. For the sake of simplicity and harmonisation, we consider the following notion of rule that encompasses TRS rules and LP rules (usually, the right-hand side of a TRS rule is a term, see Sect. 1.1).

Definition 2.13 A *program* is a subset of $T(\Sigma, X) \times \overline{T(\Sigma, X)}$, every element $(u, \overline{v})$ of which is called a *rule*. The term u (resp. the (possibly empty) goal $\overline{v}$) is the *left-hand side* (resp. *right-hand side*).

The rules of a program allow one to rewrite terms and goals. This is formalised by the following binary relations, where $\rightarrow_P$ corresponds to the operational semantics of term rewriting and $\rightsquigarrow_P$ to that of logic programming. For all goals $\overline{s}$ and rules $(u, \overline{v})$ and $(u', \overline{v}')$, we write $(u, \overline{v}) \ll_{\overline{s}} (u', \overline{v}')$ to denote that $(u, \overline{v})$ is a variant of $(u', \overline{v}')$ variable disjoint with $\overline{s}$, *i.e.*, for some renaming γ, we have $u = u'\gamma$, $\overline{v} = \overline{v}'\gamma$ and $Var(u) \cap Var(\overline{s}) = Var(\overline{v}) \cap Var(\overline{s}) = \emptyset$.

Definition 2.14 For all programs P, we let

$$\underset{P}{\rightarrow} = \bigcup \left\{ \underset{r}{\rightarrow} \;\middle|\; r \in P \right\} \quad \text{and} \quad \underset{P}{\rightsquigarrow} = \bigcup \left\{ \underset{r}{\rightsquigarrow} \;\middle|\; r \in P \right\}$$

where, for all $r \in P$,

$$\underset{r}{\rightarrow} = \bigcup \left\{ \underset{(r,p)}{\rightarrow} \;\middle|\; p \in \overline{\mathbb{N}} \right\} \quad \text{and} \quad \underset{r}{\rightsquigarrow} = \bigcup \left\{ \underset{(r,p)}{\rightsquigarrow} \;\middle|\; p \in \overline{\mathbb{N}} \right\}$$

and, for all $p \in \overline{\mathbb{N}}$,

$$\underset{(r,p)}{\rightarrow} = \left\{ (s, t) \in T(\Sigma, X)^2 \;\middle|\; \begin{array}{l} r = (u, \langle v \rangle),\ p \in Pos(s) \\ s|_p = u\theta,\ \theta \in S(\Sigma, X) \\ t = s[v\theta]_p \end{array} \right\}$$

$$\underset{(r,p)}{\rightsquigarrow} = \left\{ (\overline{s}, \overline{t}) \in \overline{T(\Sigma, X)}^2 \;\middle|\; \begin{array}{l} \overline{s} = \langle s_1, \ldots, s_n \rangle,\ p \in \{\langle 1 \rangle, \ldots, \langle n \rangle\} \\ (u, \langle v_1, \ldots, v_m \rangle) \ll_{\overline{s}} r,\ \theta = mgu(s_p, u) \\ \overline{t} = \langle s_1, \ldots, s_{p-1}, v_1, \ldots, v_m, s_{p-1}, \ldots, s_n \rangle\theta \end{array} \right\}$$

Concrete examples are provided in Sect. 1.1. We note that if the rule r has the form $(u, \overline{v})$ with $|\overline{v}| \neq 1$ then $\rightarrow_{(r,p)} = \emptyset$ for all $p \in \overline{\mathbb{N}}$. On the other hand, in the definition of $\rightsquigarrow_{(r,p)}$, we may have $\langle v_1, \ldots, v_m \rangle = \epsilon$ (*i.e.*, $m < 1$).

The next lemma states some closure properties of $\rightarrow_{(r,p)}$ and $\rightsquigarrow_{(r,p)}$ under substitutions. It is needed to prove compatibility of $\rightarrow_r$ and mg (Lemma 2.17 below) as well as closure properties of abstract reduction systems (see Sect. 3.1). As $\rightarrow_{(r,p)}$ relies on instantiation, its closure property is almost straightforward. In contrast, $\rightsquigarrow_{(r,p)}$ relies on narrowing, so its closure property is more restricted and, moreover, it is more complicated to prove.

Lemma 2.15 *Let $r = (u, \langle v \rangle)$ be a rule and θ be a substitution.*

- *For all positions p and all terms s, t, we have: $s \rightarrow_{(r,p)} t$ implies $s\theta \rightarrow_{(r,p)} t\theta$.*
- *$Var(v) \subseteq Var(u)$ implies $\langle u\theta \rangle \rightsquigarrow_{(r,\langle 1 \rangle)} \langle v\theta \rangle$.*

Proof Let p be a position and s, t be some terms such that $s \rightarrow_{(r,p)} t$. Then, by Definition 2.14, we have $p \in Pos(s)$, $s|_p = u\sigma$ and $t = s[v\sigma]_p$ for some substitution σ. We have $p \in Pos(s\theta)$ and $s\theta|_p = s|_p\theta = u\sigma\theta$. So, by Definition 2.14, $s\theta \rightarrow_{(r,p)} s\theta[v\sigma\theta]_p$ where $s\theta[v\sigma\theta]_p = s[v\sigma]_p\theta = t\theta$. Consequently, we have $s\theta \rightarrow_{(r,p)} t\theta$.

Now, suppose that $Var(v) \subseteq Var(u)$ and let us prove that $\langle u\theta \rangle \rightsquigarrow_{(r,\langle 1 \rangle)} \langle v\theta \rangle$. Let $(u\gamma, \langle v\gamma \rangle)$ be a variant of r variable disjoint with $u\theta$, for some variable renaming γ. Let $\eta = \{x\gamma \mapsto x\theta \mid x \in Var(u),\ x\gamma \neq x\theta\}$.

- First, we prove that η is a substitution. Let $(x \mapsto s)$ and $(y \mapsto t)$ be some bindings in η. By definition of η, we have $(x \mapsto s) = (x'\gamma \mapsto x'\theta)$ and $(y \mapsto t) = (y'\gamma \mapsto y'\theta)$ for some variables x' and y' in $Var(u)$. As γ is a variable renaming, it is a bijection on X, so if $x = y$ then $x' = y'$ and hence $(x \mapsto s) = (y \mapsto t)$. Consequently, for any bindings $(x \mapsto s)$ and $(y \mapsto t)$ in η, $(x \mapsto s) \neq (y \mapsto t)$ implies $x \neq y$. Moreover, by definition of η, for any $(x \mapsto s) \in \eta$ we have $x \neq s$. Therefore, η is a substitution.

- Then, we prove that η is a unifier of $u\gamma$ and $u\theta$.
  - Let $x \in Var(u)$. Then, by definition of η, $x\gamma\eta = x\theta$. So, $u\gamma\eta = u\theta$.
  - Let $y \in Dom(\eta)$. Then, $y = x\gamma$ for some $x \in Var(u)$. Hence, $y \in Var(u\gamma)$. As $u\gamma$ is variable disjoint with $u\theta$, we have $y \notin Var(u\theta)$. Therefore, we have $Dom(\eta) \cap Var(u\theta) = \emptyset$, so $u\theta\eta = u\theta$.

  Consequently, we have $u\gamma\eta = u\theta\eta$.
- Finally, we prove that η is more general than any other unifier of $u\gamma$ and $u\theta$. Let σ be a unifier of $u\gamma$ and $u\theta$. Then, for all $x \in Var(u)$, we have $x\gamma\sigma = x\theta\sigma$. For all variables y that do not occur in $Dom(\eta)$, we have $y\eta\sigma = y\sigma$ and, for all $y \in Dom(\eta)$, we have $y = x\gamma$ for some $x \in Var(u)$, so $y\eta\sigma = x\gamma\eta\sigma = x\theta\sigma = x\gamma\sigma = y\sigma$. Hence, $\eta\sigma = \sigma$, *i.e.*, η is more general than σ.

Therefore, $\eta = mgu(u\gamma, u\theta)$. So, by Definition 2.14, we have $\langle u\theta\rangle \rightsquigarrow_{(r,\langle 1\rangle)} \langle v\gamma\eta\rangle$. Note that for all $x \in Var(v)$, we have $x \in Var(u)$, so $x\gamma\eta = x\theta$ by definition of η. Hence, $v\gamma\eta = v\theta$. Finally, we have $\langle u\theta\rangle \rightsquigarrow_{(r,\langle 1\rangle)} \langle v\theta\rangle$. □

Example 2.16 Consider the rule $r = (u, \langle v\rangle) = (\mathsf{f}(x, 1), \langle \mathsf{f}(1, x)\rangle)$ and the substitution $\theta = \{x \mapsto 0, y \mapsto 0\}$. We have $\langle u\theta\rangle = \langle \mathsf{f}(0, 1)\rangle$, $\langle v\theta\rangle = \langle \mathsf{f}(1, 0)\rangle$ and $\langle u\theta\rangle \rightsquigarrow_{(r,\langle 1\rangle)} \langle v\theta\rangle$.

For $s = \mathsf{f}(0, y)$, we also have $\langle s\rangle \rightsquigarrow_{(r,\langle 1\rangle)} \langle \mathsf{f}(1, 0)\rangle$, but there is no rewriting of $\langle s\theta\rangle$ with r because $s\theta = \mathsf{f}(0, 0)$ does not unify with any variant of u.

The next compatibility results follow from Definitions 2.12 and 2.14 and from Lemma 2.15.

Lemma 2.17 *For all rules r, $\rightarrow_r$ and ins are compatible, and so are $\rightsquigarrow_r$ and mg.*

Proof Let r be a rule.

- Let s, t, s' be terms such that $s' \in ins(s)$ and $s \rightarrow_r t$.
  Then, by Definition 2.12, we have $s' = c[s\sigma]$ for some context c and some substitution σ. Moreover, by Definition 2.14, $s \rightarrow_{(r,p)} t$ for some $p \in \overline{\mathbb{N}}$ and r has the form $(u, \langle v\rangle)$. So, by Lemma 2.15, we have $s\sigma \rightarrow_{(r,p)} t\sigma$. Let p' be the position of an occurrence of $\square$ in c. Then, $c[s\sigma] \rightarrow_{(r,p'p)} c[t\sigma]$ where $c[s\sigma] = s'$ and $c[t\sigma] \in ins(t)$. So, we have proved that $s' \rightarrow_r t'$ for some $t' \in ins(t)$, *i.e.*, that $\rightarrow_r$ and *ins* are compatible.
- Let $\overline{s}, \overline{t}, \overline{s'}$ be goals such that $\overline{s'} \in mg(\overline{s})$ and $\overline{s} \rightsquigarrow_r \overline{t}$. Then, by Definition 2.12, we have $\overline{s'} = \overline{c}[\overline{a}]$ for some goal-context $\overline{c}$ and some goal $\overline{a}$ that is more general than $\overline{s}$. Moreover, by Definition 2.14, we have $\overline{s} \rightsquigarrow_{(r,p)} \overline{t}$ for some $p \in \overline{\mathbb{N}}$. Let r' be a variant of r that is variable disjoint with $\overline{s'}$. Then, r' is also variable disjoint with $\overline{a}$. So, by the One Step Lifting Lemma 3.21 at page 59 of [1], for some goal $\overline{b}$ that is more general than $\overline{t}$, we have $\overline{a} \rightsquigarrow^{\theta}_{(r,p)} \overline{b}$, where r' and θ are respectively the variant of r and the unifier used. As r' is also variable disjoint with $\overline{c}$, we have $\overline{c}[\overline{a}] \rightsquigarrow^{\theta}_{(r,p'+p)} (\overline{c}\theta)[\overline{b}]$ where r' is the variant of r used and p' is the position of $\square$ in $\overline{c}$. We note that $\overline{c}[\overline{a}] = \overline{s'}$ and $(\overline{c}\theta)[\overline{b}] \in mg(\overline{t})$. So, we have proved that $\overline{s'} \rightsquigarrow_r \overline{t'}$ for some $\overline{t'} \in mg(\overline{t})$, *i.e.*, that $\rightsquigarrow_r$ and *mg* are compatible.

□

Example 2.18 In Example 1.1, $s = \mathsf{g}\big(\mathsf{h}(\mathsf{f}(x), 0), x\big)$ and $s \rightarrow_{(r_3,p)} \mathsf{g}(\mathsf{f}^3(x), x)$ where $p = \langle 1\rangle$. Let $\sigma = \{x \mapsto 0\}$. Then, $s\sigma = \mathsf{g}\big(\mathsf{h}(\mathsf{f}(0), 0), 0\big)$ and $s\sigma \rightarrow_{(r_3,p)} \mathsf{g}(\mathsf{f}^3(0), 0)$, *i.e.*, $s\sigma \rightarrow_{(r_3,p)} \mathsf{g}(\mathsf{f}^3(x), x)\sigma$, where $s\sigma \in ins(s)$ and $\mathsf{g}(\mathsf{f}^3(x), x)\sigma \in ins(\mathsf{g}(\mathsf{f}^3(x), x))$.

3 Abstract Reduction Systems

The following notion (see, *e.g.*, Chapter 2 of [2] or Chapter 1 of [30]) generalises the semantics of term rewriting and logic programming presented above.

Definition 3.1 An *abstract reduction system (ARS)* is a pair $(A, \Rightarrow_\Pi)$ consisting of a set A and a *rewrite relation* $\Rightarrow_\Pi$, which is the union of binary relations on A indexed by a set Π, *i.e.*, $\Rightarrow_\Pi = \bigcup\{\Rightarrow_\pi \mid \pi \in \Pi\}$.

We have $A = T(\Sigma, X)$ in term rewriting and $A = \overline{T(\Sigma, X)}$ in logic programming; moreover, Π is a program in both cases.

We formalise non-termination as the existence of an infinite chain in an ARS.

Definition 3.2 A *chain* in an ARS $\mathcal{A} = (A, \Rightarrow_\Pi)$ is a (possibly infinite) $\Rightarrow_\Pi$-chain.

3.1 Closure Under Substitutions

In Sect. 5.1, we prove that the existence of finite chains $u_1 \Rightarrow_\Pi^+ v_1$ and $u_2 \Rightarrow_\Pi^+ v_2$ of a special form implies that of an infinite chain that involves instances of u_1, v_1, u_2 and v_2 (see Definition 5.5 and Corollary 5.13). The proof relies on the following property of ARSs. In the rest of this paper, for all ARSs $(A, \Rightarrow_\Pi)$ and all $w = \langle \pi_1, \dots, \pi_n \rangle$ in $\overline{\Pi}$, we let $\Rightarrow_w = (\Rightarrow_{\pi_1} \circ \cdots \circ \Rightarrow_{\pi_n})$, where $\Rightarrow_\epsilon$ is the identity relation.

Definition 3.3 Let $\mathcal{A} = (A, \Rightarrow_\Pi)$ be an ARS where $A \subseteq T(\Sigma, X) \cup \overline{T(\Sigma, X)}$. We say that $\mathcal{A}$ is *closed under substitutions* if, for all $s, t \in A$, all $w \in \overline{\Pi}$ and all substitutions θ, $s \Rightarrow_w t$ implies $s\theta \Rightarrow_w t\theta$.

The following result is a consequence of Lemma 2.15.

Lemma 3.4 *For all programs P, $(T(\Sigma, X), \rightarrow_P)$ is closed under substitutions.*

Proof Let P be a program, s, t be terms, $w \in \overline{P}$ and θ be a substitution. Suppose that $s \rightarrow_w t$. We prove by induction on $|w|$ that $s\theta \rightarrow_w t\theta$.

- (Base: $|w| = 0$) Here, $\rightarrow_w$ is the identity relation. As $s \rightarrow_w t$, we have $s = t$, hence $s\theta = t\theta$, and so $s\theta \rightarrow_w t\theta$.
- (Induction) Suppose that $|w| = n + 1$ for some $n \in \mathbb{N}$. Suppose also that for all terms s', t', all $w' \in \overline{P}$ with $|w'| = n$ and all substitutions σ, $s' \rightarrow_{w'} t'$ implies $s'\sigma \rightarrow_{w'} t'\sigma$. As $|w| = n + 1$, we have $w = rw'$ for some $r \in P$ and some $w' \in \overline{P}$ with $|w'| = n$. Therefore, $s \rightarrow_r s' \rightarrow_{w'} t$ for some term s'. By definition of $\rightarrow_r$, we have $r = (u, \langle v \rangle)$, so, by Lemma 2.15, $s\theta \rightarrow_r s'\theta$. As $|w'| = n$, by induction hypothesis we have $s'\theta \rightarrow_{w'} t\theta$. Hence, $s\theta \rightarrow_r s'\theta \rightarrow_{w'} t\theta$, *i.e.*, $s\theta \rightarrow_w t\theta$. □

However, for all programs P, $(\overline{T(\Sigma, X)}, \rightsquigarrow_P)$ is not closed under substitutions, see Example 2.16. Hence, based on Lemma 2.15, we introduce the following restricted form of logic programming, where one only rewrites singleton goals using rules, the right-hand side of which is a singleton goal.

Definition 3.5 For all programs P, we let $\hookrightarrow_P = \bigcup\{\hookrightarrow_r \mid r \in P\}$ where, for all $r \in P$, $\hookrightarrow_r = \big\{(u\theta, v\theta) \in T(\Sigma, X)^2 \bigm| r = (u, \langle v \rangle),\ Var(v) \subseteq Var(u),\ \theta \in S(\Sigma, X)\big\}$.

We note that $\hookrightarrow_r \subseteq \rightarrow_{r,\epsilon}$. Moreover, $\hookrightarrow_r = \emptyset$ if $r = (u, \overline{v})$ with $Var(\overline{v}) \not\subseteq Var(u)$ or $|\overline{v}| \neq 1$. Now, we have a counterpart of Lemma 3.4 in logic programming:

Lemma 3.6 *For all programs P, $(T(\Sigma, X), \hookrightarrow_P)$ is closed under substitutions.*

Proof Let P be a program, s, t be terms, $w \in \overline{P}$ and θ be a substitution. Suppose that $s \hookrightarrow_w t$. We prove by induction on $|w|$ that $s\theta \hookrightarrow_w t\theta$.

- (Base: $|w| = 0$) Here, $\hookrightarrow_w$ is the identity relation. As $s \hookrightarrow_w t$, we have $s = t$, hence $s\theta = t\theta$, and so $s\theta \hookrightarrow_w t\theta$.
- (Induction) Suppose that $|w| = n + 1$ for some $n \in \mathbb{N}$. Suppose also that for all terms s', t', all $w' \in \overline{P}$ with $|w'| = n$ and all substitutions σ, $s' \hookrightarrow_{w'} t'$ implies $s'\sigma \hookrightarrow_{w'} t'\sigma$. As $|w| = n + 1$, we have $w = rw'$ for some $r \in P$ and some $w' \in \overline{P}$ with $|w'| = n$. Therefore, $s \hookrightarrow_r s' \hookrightarrow_{w'} t$ for some term s'. By Definition 3.5, we have $r = (u, \langle v \rangle)$, $Var(v) \subseteq Var(u)$, $s = u\sigma$ and $s' = v\sigma$ for some substitution σ; moreover, $u\sigma\theta \hookrightarrow_r v\sigma\theta$; so, $s\theta \hookrightarrow_r s'\theta$. As $|w'| = n$, by induction hypothesis we have $s'\theta \hookrightarrow_{w'} t\theta$. Hence, $s\theta \hookrightarrow_r s'\theta \hookrightarrow_{w'} t\theta$, *i.e.*, $s\theta \hookrightarrow_w t\theta$. □

It follows from the next result that the existence of an infinite $\hookrightarrow_P$-chain implies that of an infinite $\rightsquigarrow_P$-chain, *i.e.*, non-termination in the restricted form of logic programming implies non-termination in full logic programming.

Lemma 3.7 *For all programs P, terms s, t and $w \in \overline{P}$, $s \hookrightarrow_w t$ implies $\langle s \rangle \rightsquigarrow_w \langle t \rangle$.*

Proof Let P be a program, s, t be terms and $w \in \overline{P}$. Suppose that $s \hookrightarrow_w t$. We prove by induction on $|w|$ that $\langle s \rangle \rightsquigarrow_w \langle t \rangle$.

- (Base: $|w| = 0$) Here, $\hookrightarrow_w$ and $\rightsquigarrow_w$ are the identity relation. As $s \hookrightarrow_w t$, we have $s = t$, hence $\langle s \rangle = \langle t \rangle$, and so $\langle s \rangle \rightsquigarrow_w \langle t \rangle$.
- (Induction) Suppose that $|w| = n+1$ for some $n \in \mathbb{N}$. Suppose also that for all terms s', t' and all $w' \in \overline{P}$ with $|w'| = n$, $s' \hookrightarrow_{w'} t'$ implies $\langle s' \rangle \rightsquigarrow_{w'} \langle t' \rangle$. As $|w| = n + 1$, we have $w = rw'$ for some $r \in P$ and some $w' \in \overline{P}$ with $|w'| = n$. Therefore, $s \hookrightarrow_r s' \hookrightarrow_{w'} t$ for some term s'. By definition of $\hookrightarrow_r$, we have $r = (u, \langle v \rangle)$, $Var(v) \subseteq Var(u)$, $s = u\theta$ and $s' = v\theta$ for some substitution θ. So, by Lemma 2.15, we have $\langle s \rangle \rightsquigarrow_r \langle s' \rangle$. As $|w'| = n$, by induction hypothesis we have $\langle s' \rangle \rightsquigarrow_{w'} \langle t \rangle$. Hence, $\langle s \rangle \rightsquigarrow_r \langle s' \rangle \rightsquigarrow_{w'} \langle t \rangle$, *i.e.*, $\langle s \rangle \rightsquigarrow_w \langle t \rangle$.

□

4 Loops

In Sect. 1.2, we have provided an informal description of loops. Now, we propose a formal definition in an abstract setting.

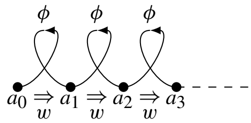
Definition 4.1 Let $\mathcal{A} = (A, \Rightarrow_\Pi)$ be an ARS, $w \in \overline{\Pi}$ and ϕ be a binary relation on A which is compatible with $\Rightarrow_\Pi$. A (w, ϕ)-*loop* in $\mathcal{A}$ is a pair $(a, a') \in A^2$ such that $a \Rightarrow_w a'$ and $a' \in \phi(a)$.

By definition of compatibility (Definition 2.2), the existence of a loop immediately leads to that of an infinite chain.

Lemma 4.2 *Suppose that there is a (w, ϕ)-loop (a, a') in an ARS $(A, \Rightarrow_\Pi)$. Then, there is an infinite $\Rightarrow_w$-chain that starts with $a \Rightarrow_w a'$.*

Proof Let $a_0 = a$ and $a_1 = a'$. As (a, a') is a (w, ϕ)-loop, we have $a_0 \Rightarrow_w a_1$ and $a_1 \in \phi(a_0)$. Moreover, $\Rightarrow_\Pi$ and ϕ are compatible. Therefore, by the definition of $\Rightarrow_w$ and Lemma 2.3, $\Rightarrow_w$ and ϕ are compatible. So, we have $a_1 \Rightarrow_w a_2$ for some $a_2 \in \phi(a_1)$. Again by compatibility of $\Rightarrow_w$ and ϕ, we have $a_2 \Rightarrow_w a_3$ for some $a_3 \in \phi(a_2)$, *etc.* □

We note that the infinite chain $a_0 \Rightarrow_w a_1 \Rightarrow_w \cdots$ corresponding to a (w, ϕ)-loop is such that for all $n \in \mathbb{N}$, $a_0 \Rightarrow_w^n a_n$ and $a_n \in \phi^n(a_0)$, *i.e.*, elements that are somehow "similar" to a_0 w.r.t. ϕ (*i.e.*, $a_n \in \phi^n(a_0)$) are periodically "reached" via $\Rightarrow_w$. The name *loop* stems from this observation.



$$\forall n \in \mathbb{N}\ \left(a_0 \Rightarrow_w^n a_n\right) \wedge \left(a_n \in \phi^n(a_0)\right)$$

Moreover, the chain $a_0 \Rightarrow_w a_1 \Rightarrow_w \cdots$ only relies on a single sequence w of elements of Π. In the next section, we will consider more elaborated chains based on two sequences.

Example 4.3 In Example 1.3, we have

$$\underbrace{\mathsf{f}(x)}_{a_0} \underset{r_1}{\rightarrow} \mathsf{g}(\mathsf{h}(x, 1), x) \underset{r_2}{\rightarrow} \mathsf{g}(\mathsf{h}(x, 0), x) \underset{r_3}{\rightarrow} \underbrace{\mathsf{g}(\mathsf{f}^2(x), x)}_{a_3}$$

i.e., $a_0 \rightarrow_w a_3$ for $w = \langle r_1, r_2, r_3\rangle$. Moreover, $a_3 \in \mathit{ins}(a_0)$ and $\rightarrow_P$ and *ins* are compatible, where $P = \{r_1, r_2, r_3\}$ (see Lemma 2.17). So, (a_0, a_3) is a (w, ins)-loop in $(T(\Sigma, X), \rightarrow_P)$. Therefore, by Lemma 4.2, there is an infinite $\rightarrow_w$-chain that starts with $a_0 \rightarrow_w a_3$. Indeed, we have

$$\underbrace{\mathsf{f}(x)}_{a_0} \underset{w}{\rightarrow} \underbrace{\mathsf{g}(\mathsf{f}^2(x), x)}_{a_3} \underset{w}{\rightarrow} \underbrace{\mathsf{g}\big(\mathsf{g}(\mathsf{f}^3(x), \mathsf{f}(x)), x\big)}_{a_6} \underset{w}{\rightarrow} \cdots$$

where $a_3 \in \mathit{ins}(a_0)$, $a_6 \in \mathit{ins}(a_3)$, ... We note that in this chain, the rules are applied at positions that vary gradually (*e.g.*, r_1 is applied to subterms of the form $\mathsf{f}(\cdots)$ that occur at deeper and deeper positions).

Example 4.4 In Example 1.4, we have

$$\underbrace{\langle \mathsf{p}(\mathsf{f}(x, 0))\rangle}_{\overline{a_0}} \underset{r}{\rightsquigarrow} \underbrace{\langle \mathsf{p}(x), \mathsf{q}(x)\rangle}_{\overline{a_1}}$$

i.e., $\overline{a_0} \rightarrow_w \overline{a_1}$ for $w = \langle r\rangle$. Moreover, $\overline{a_1} \in \mathit{mg}(\overline{a_0})$ and $\rightsquigarrow_P$ and *mg* are compatible, where $P = \{r\}$ (see Lemma 2.17). So, $(\overline{a_0}, \overline{a_1})$ is a (w, mg)-loop in $(\overline{T(\Sigma, X)}, \rightsquigarrow_P)$. Therefore, by Lemma 4.2, there is an infinite $\rightsquigarrow_w$-chain that starts with $\overline{a_0} \rightsquigarrow_w \overline{a_1}$. Indeed, we have

$$\underbrace{\langle \mathsf{p}(\mathsf{f}(x, 0))\rangle}_{\overline{a_0}} \underset{w}{\rightsquigarrow} \underbrace{\langle \mathsf{p}(x), \mathsf{q}(x)\rangle}_{\overline{a_1}} \underset{w}{\rightsquigarrow} \underbrace{\langle \mathsf{p}(x_1), \mathsf{q}(x_1), \mathsf{p}(\mathsf{f}(x_1, 0))\rangle}_{\overline{a_2}} \underset{w}{\rightsquigarrow} \cdots$$

where $\overline{a_1} \in \mathit{mg}(\overline{a_0})$, $\overline{a_2} \in \mathit{mg}(\overline{a_1})$, ...

5 Binary Chains

In Sect. 4 we have considered infinite chains that rely on a single sequence w of elements of Π. A natural way to extend this class is to consider infinite chains based on two sequences w_1 and w_2 as in Definition 5.1 below. In principle, one could also define many other forms of similar chains. In this paper, we concentrate on this form only as it covers interesting examples found in the literature or in [32] (*e.g.*, Examples 5.2, 5.3, 5.4 below) and, moreover, we are able to provide an automatable approach for the detection of a special case (see Sect. 5.1); this case allows one for instance to encode the semantics of vector addition systems [16].

Definition 5.1 A *binary chain* in an ARS $(A, \Rightarrow_\Pi)$ is an infinite $(\Rightarrow^*_{w_1} \circ \Rightarrow_{w_2})$-chain for some $w_1, w_2 \in \overline{\Pi}$.

Of course, as $\Rightarrow_w = (\Rightarrow^0_w \circ \Rightarrow_w)$, the infinite chain $a_0 \Rightarrow_w a_1 \Rightarrow_w \cdots$ corresponding to a (w, ϕ)-loop is also binary. Moreover, any infinite chain of the form $a_0 (\Rightarrow^*_{w_1} \circ \Rightarrow^*_{w_2})\, a_1 (\Rightarrow^*_{w_1} \circ \Rightarrow^*_{w_2}) \cdots$ is binary, because any sequence $a_i \Rightarrow_{w_2} a'_i \Rightarrow_{w_2} a''_i$ has the form $a_i \Rightarrow^0_{w_1} a_i \Rightarrow_{w_2} a'_i \Rightarrow^0_{w_1} a'_i \Rightarrow_{w_2} a''_i$.

Example 5.2 [5, 9, 34] Consider the rules

$$r_1 = \big(\mathsf{b}(\mathsf{c}), \langle \mathsf{d}(\mathsf{c})\rangle\big) \qquad r_2 = \big(\mathsf{b}(\mathsf{d}(x)), \langle \mathsf{d}(\mathsf{b}(x))\rangle\big) \qquad r_3 = \big(\mathsf{a}(\mathsf{d}(x)), \langle \mathsf{a}(\mathsf{b}^2(x))\rangle\big)$$

For the sake of readability, let us omit parentheses and write for instance adc instead of a(d(c)). Let $w_1 = \langle r_2\rangle$ and $w_2 = \langle r_3, r_1\rangle$. We have the infinite $(\rightarrow^*_{w_1} \circ \rightarrow_{w_2})$-chain

$$\begin{aligned}
\mathsf{adc} \xrightarrow[r_2]{0} \mathsf{adc} &\xrightarrow[r_3]{} \mathsf{ab^2c} \xrightarrow[r_1]{} \mathsf{abdc}\\
&\xrightarrow[r_2]{1} \mathsf{adbc} \xrightarrow[r_3]{} \mathsf{ab^3c} \xrightarrow[r_1]{} \mathsf{ab^2dc}\\
&\xrightarrow[r_2]{2} \mathsf{adb^2c} \xrightarrow[r_3]{} \mathsf{ab^4c} \xrightarrow[r_1]{} \mathsf{ab^3dc}\\
&\xrightarrow[r_2]{3} \cdots
\end{aligned}$$

We note that in this chain, the rule r_3 is always applied at the root position but that r_1 and r_2 are applied at positions that vary.

Example 5.3 (`TRS_Standard/Zantema_15/ex11.xml` in [32]) Consider the rules

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

Let $w_1 = \langle r_1\rangle$ and $w_2 = \langle r_2\rangle$. We have the infinite $(\hookrightarrow^*_{w_1} \circ \hookrightarrow_{w_2})$-chain

$$\begin{aligned}
\mathsf{f}(\mathsf{s}(0), 0) \overset{0}{\underset{r_1}{\hookrightarrow}} \mathsf{f}(\mathsf{s}(0), 0) &\underset{r_2}{\hookrightarrow} \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0))\\
&\overset{1}{\underset{r_1}{\hookrightarrow}} \mathsf{f}(\mathsf{s}^2(0), 0) \underset{r_2}{\hookrightarrow} \mathsf{f}(\mathsf{s}(0), \mathsf{s}^2(0))\\
&\overset{2}{\underset{r_1}{\hookrightarrow}} \mathsf{f}(\mathsf{s}^3(0), 0) \underset{r_2}{\hookrightarrow} \mathsf{f}(\mathsf{s}(0), \mathsf{s}^3(0))\\
&\overset{3}{\underset{r_1}{\hookrightarrow}} \cdots
\end{aligned}$$

For instance, $u_1 = \mathsf{f}(\mathsf{s}(0), \mathsf{s}(0))$ and $u_2 = \mathsf{f}(\mathsf{s}^2(0), 0)$ correspond to the integer vectors $v_1 = (1, 1)$ and $v_2 = (2, 0)$, respectively, and the chain $u_1 \hookrightarrow_{r_1} u_2$ models the component-wise addition of the vector $(1, -1)$ to u_1. The programs `ex12.xml` and `ex14.xml` in the same directory of [32] are similar.

Example 5.4 [38] Consider the rules

$$r_1 = \big(\mathsf{f}(\mathsf{c}, \mathsf{a}(x), y), \langle \mathsf{f}(\mathsf{c}, x, \mathsf{a}(y))\rangle\big) \qquad r_2 = \big(\mathsf{f}(\mathsf{c}, \mathsf{a}(x), y), \langle \mathsf{f}(x, y, \mathsf{a}^2(\mathsf{c}))\rangle\big)$$

Let $w_1 = \langle r_1 \rangle$ and $w_2 = \langle r_2 \rangle$. We have the infinite $(\hookrightarrow^*_{w_1} \circ \hookrightarrow_{w_2})$-chain

$$\begin{aligned}
\mathsf{f}(\mathsf{c}, \mathsf{a}(\mathsf{c}), \mathsf{a}^2(\mathsf{c})) &\overset{0}{\underset{r_1}{\hookrightarrow}} \mathsf{f}(\mathsf{c}, \mathsf{a}(\mathsf{c}), \mathsf{a}^2(\mathsf{c})) \underset{r_2}{\hookrightarrow} \mathsf{f}(\mathsf{c}, \mathsf{a}^2(\mathsf{c}), \mathsf{a}^2(\mathsf{c})) \\
&\overset{1}{\underset{r_1}{\hookrightarrow}} \mathsf{f}(\mathsf{c}, \mathsf{a}(\mathsf{c}), \mathsf{a}^3(\mathsf{c})) \underset{r_2}{\hookrightarrow} \mathsf{f}(\mathsf{c}, \mathsf{a}^3(\mathsf{c}), \mathsf{a}^2(\mathsf{c})) \\
&\overset{2}{\underset{r_1}{\hookrightarrow}} \mathsf{f}(\mathsf{c}, \mathsf{a}(\mathsf{c}), \mathsf{a}^4(\mathsf{c})) \underset{r_2}{\hookrightarrow} \mathsf{f}(\mathsf{c}, \mathsf{a}^4(\mathsf{c}), \mathsf{a}^2(\mathsf{c})) \\
&\overset{3}{\underset{r_1}{\hookrightarrow}} \cdots
\end{aligned}$$

5.1 Recurrent Pairs

Now, we present a new criterion for the detection of binary chains. It is based on two specific chains $u_1 \Rightarrow^+_\Pi v_1$ and $u_2 \Rightarrow^+_\Pi v_2$ such that a context is removed from u_1 to v_1 while it is added again from u_2 to v_2. This is formalised as follows. In the next definition, we consider a new hole symbol $\Box'$ that does not occur in $\Sigma \cup X \cup \{\Box\}$ and we let c_1 be a context with at least one occurrence of $\Box$ and $\Box'$; moreover, for all terms t, t', we let $c_1[t, t']$ be the term obtained from c_1 by replacing the occurrences of $\Box$ (resp. $\Box'$) by t (resp. t'). On the other hand, we let c_2 be a context with occurrences of $\Box$ only (as in Definition 2.6).

Definition 5.5 Let $\mathcal{A} = (T(\Sigma, X), \Rightarrow_\Pi)$ be an ARS closed under substitutions. A *recurrent pair in* $\mathcal{A}$ is a pair $(u_1 \Rightarrow_{w_1} v_1, u_2 \Rightarrow_{w_2} v_2)$ of finite chains in $\mathcal{A}$ such that

- $u_1 = c_1[x, c_2[y]]$, $v_1 = c_1[c_2^{n_1}[x], y]$, $u_2 = c_1[x, c_2^{n_2}[s]]$ and $v_2 = c_1[c_2^{n_3}[t], c_2^{n_4}[x]]$
- $x \neq y$ and $\{x, y\} \cap \mathit{Var}(c_1) = \emptyset$
- $\mathit{Var}(c_2) = \mathit{Var}(s) = \emptyset$
- $t \in \{x, s\}$
- $n_4 \geq n_2$

The ARS $\big(T(\Sigma, X), \to_{\{r_1, r_2, r_3\}}\big)$ of Example 5.2 is not covered by this definition because it only involves function symbols of arity 0 or 1 (hence, one cannot find a context c_1 with at least one occurrence of $\Box$ and $\Box'$).

Example 5.6 In Example 1.5, we have the chains $\mathsf{f}(x, c[y], x) \to_{w_1} \mathsf{f}(c[x], y, c[x])$ and $\mathsf{f}(x, \mathsf{0}, x) \to_{w_2} \mathsf{f}(c[x], c[x], c[x])$. As $\big(T(\Sigma, X), \to_{\{r_1, r_2, r_3, r_4\}}\big)$ is closed under substitutions (Lemma 3.4), these chains form a recurrent pair, with $c_1 = \mathsf{f}(\Box, \Box', \Box)$, $c_2 = c = \mathsf{g}(\Box, \mathsf{0}, \Box)$, $(n_1, n_2, n_3, n_4) = (1, 0, 1, 1)$, $s = \mathsf{0}$ and $t = x$.

Example 5.7 In Example 5.3, we have the chains $\mathsf{f}(x, \mathsf{s}(y)) \hookrightarrow_{w_1} \mathsf{f}(\mathsf{s}(x), y)$ and $\mathsf{f}(x, \mathsf{0}) \hookrightarrow_{w_2} \mathsf{f}(\mathsf{s}(\mathsf{0}), x)$. As $\big(T(\Sigma, X), \hookrightarrow_{\{r_1, r_2\}}\big)$ is closed under substitutions (Lemma 3.6), these chains form a recurrent pair, with $c_1 = \mathsf{f}(\Box, \Box')$, $c_2 = \mathsf{s}(\Box)$, $(n_1, n_2, n_3, n_4) = (1, 0, 1, 0)$ and $s = t = \mathsf{0}$.

Example 5.8 In Example 5.4, we have the chains $\mathsf{f}(\mathsf{c}, \mathsf{a}(x), y) \hookrightarrow_{w_1} \mathsf{f}(\mathsf{c}, x, \mathsf{a}(y))$ and $\mathsf{f}(\mathsf{c}, \mathsf{a}(\mathsf{c}), y) \hookrightarrow_{w_2} \mathsf{f}(\mathsf{c}, y, \mathsf{a}^2(\mathsf{c}))$. As $\big(T(\Sigma, X), \hookrightarrow_{\{r_1, r_2\}}\big)$ is closed under substitutions (Lemma 3.6), these chains form a recurrent pair, with $c_1 = \mathsf{f}(\mathsf{c}, \Box', \Box)$, $c_2 = \mathsf{a}(\Box)$, $(n_1, n_2, n_3, n_4) = (1, 0, 1, 0)$ and $s = t = \mathsf{a}(\mathsf{c})$.

It is proved in [23] that the existence of a recurrent pair of a more restricted form (*i.e.*, $|w_1| = |w_2| = 1$, $c_1 = \mathsf{f}(\Box, \Box')$ and $n_2 = 0$, as in Example 5.3) leads to that of a binary chain. The generalisation to any sequences w_1, w_2, any context c_1 and any $n_2 \in \mathbb{N}$ satisfying the constraints above is presented below (Corollary 5.13). In some special situations, the obtained binary chain actually relies on a single sequence (*e.g.*, if $n_2 = n_3 = n_4 = 0$ then we have $c_1[s, s] \rightarrow_{w_2} c_1[s, s] \rightarrow_{w_2} \cdots$), but this is not always the case (see the examples above). The next statements are parametric in an ARS $\mathcal{A}$ closed under substitutions and a recurrent pair in $\mathcal{A}$, with the notations of Definition 5.5 as well as this new one (introduced for the sake of readability):

Definition 5.9 For all $m, n \in \mathbb{N}$, we let $c_1[m, n]$ denote the term $c_1[c_2^m[s], c_2^n[s]]$.

Then, we have the following two results. Lemma 5.10 states that w_1 allows one to iteratively move a tower of c_2's from the positions of $\square'$ to those of $\square$ in c_1. Conversely, Lemma 5.11 states that w_2 allows one to copy a tower of c_2's from the positions of $\square$ to those of $\square'$ in just one application of $\Rightarrow_{w_2}$.

Lemma 5.10 *For all* $m, n \in \mathbb{N}$, $c_1[m, n+1] \Rightarrow_{w_1} c_1[m+n_1, n]$. *Consequently, for all* $m, n \in \mathbb{N}$ *with* $n \geq n_2$, *we have* $c_1[m, n] \Rightarrow_{w_1}^{n-n_2} c_1[m + (n - n_2) \times n_1, n_2]$.

Proof Let $m, n \in \mathbb{N}$. Then, $c_1[m, n+1] = c_1[c_2^m[s], c_2^{n+1}[s]] = u_1\theta$ where $\theta = \{x \mapsto c_2^m[s], y \mapsto c_2^n[s]\}$. So, as $u_1 \Rightarrow_{w_1} v_1$ and $\mathcal{A}$ is closed under substitutions, we have $c_1[m, n+1] \Rightarrow_{w_1} v_1\theta$ where $v_1\theta = c_1[c_2^{m+n_1}[s], c_2^n[s]] = c_1[m + n_1, n]$.

Now, we prove the second part of the lemma by induction on n.

- (Base: $n = n_2$) Here, $\Rightarrow_{w_1}^{n-n_2}$ is the identity relation. Hence, for all $m \in \mathbb{N}$, we have $c_1[m, n] \Rightarrow_{w_1}^{n-n_2} c_1[m, n]$, where $c_1[m, n] = c_1[m + (n - n_2) \times n_1, n_2]$.
- (Induction) Suppose that for some $n \geq n_2$ we have $c_1[m, n] \Rightarrow_{w_1}^{n-n_2} c_1[m + (n - n_2) \times n_1, n_2]$ for all $m \in \mathbb{N}$. Let $m \in \mathbb{N}$. By the first part of the lemma, $c_1[m, n+1] \Rightarrow_{w_1} c_1[m+n_1, n]$. Moreover, by induction hypothesis, $c_1[m+n_1, n] \Rightarrow_{w_1}^{n-n_2} c_1[(m+n_1)+(n-n_2) \times n_1, n_2]$, *i.e.*, $c_1[m + n_1, n] \Rightarrow_{w_1}^{n-n_2} c_1[m + (n + 1 - n_2) \times n_1, n_2]$. Consequently, finally we have $c_1[m, n+1] \Rightarrow_{w_1}^{n+1-n_2} c_1[m + (n + 1 - n_2) \times n_1, n_2]$. □

Lemma 5.11 *For all* $m \in \mathbb{N}$, *we have* $c_1[m, n_2] \Rightarrow_{w_2} c_1[m' + n_3, m + n_4]$ *where* $m' = 0$ *if* $t = s$ *and* $m' = m$ *if* $t = x$.

Proof Let $m \in \mathbb{N}$. We have $c_1[m, n_2] = c_1[c_2^m[s], c_2^{n_2}[s]] = u_2\{x \mapsto c_2^m[s]\}$. Hence, as $u_2 \Rightarrow_{w_2} v_2$ and $\mathcal{A}$ is closed under substitutions, we have $c_1[m, n_2] \Rightarrow_{w_2} v_2\{x \mapsto c_2^m[s]\}$.

- If $t = s$ then $v_2\{x \mapsto c_2^m[s]\} = c_1[c_2^{n_3}[s], c_2^{m+n_4}[s]] = c_1[n_3, m + n_4]$.
- If $t = x$ then $v_2\{x \mapsto c_2^m[s]\} = c_1[c_2^{m+n_3}[s], c_2^{m+n_4}[s]] = c_1[m + n_3, m + n_4]$. □

By combining Lemmas 5.10 and 5.11, one gets:

Proposition 5.12 *For all* $m, n \in \mathbb{N}$ *with* $n \geq n_2$, *there exist* $m', n' \in \mathbb{N}$ *such that* $n' \geq n_2$ *and* $c_1[m, n] (\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2}) c_1[m', n']$.

Proof Let $m, n \in \mathbb{N}$ with $n \geq n_2$. By Lemma 5.10, $c_1[m, n] \Rightarrow_{w_1}^{n-n_2} c_1[l, n_2]$ where $l = m + (n - n_2) \times n_1$. Moreover, by Lemma 5.11, $c_1[l, n_2] \Rightarrow_{w_2} c_1[l' + n_3, l + n_4]$ where $l' = 0$ if $t = s$ and $l' = l$ if $t = x$. Therefore, for $m' = l' + n_3$ and $n' = l + n_4$, we have $c_1[m, n] (\Rightarrow_{w_1}^{n-n_2} \circ \Rightarrow_{w_2}) c_1[m', n']$ and we note that $n' \geq n_2$ because, by Definition 5.5, $n_4 \geq n_2$. □

The next result is a straighforward consequence of Proposition 5.12.

Table 1 Termination Competition 2023—Category *TRS Standard*

Participant	NO's	% VBS	(E12)	(P21)	(P23) (Z15)	
AProVE	278 (22)	81.8 (6.5)	35 (17)	0	0	2
AutoNon	233 (20)	68.5 (5.9)	19 (3)	3	0	14 (9)
MnM	253 (9)	74.4 (2.6)	0	0	0	0
MU-TERM	136	40.0	0	0	0	0
NaTT	170	50.0	0	0	0	0
NTI	263 (10)	77.4 (2.9)	5	3	10 (10)	3
$\mathsf{T_TT_2}$	194	57.1	0	0	0	0

Corollary 5.13 *For all* $m, n \in \mathbb{N}$ *with* $n \geq n_2$, $c_1[m, n]$ *starts an infinite* $(\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2})$*-chain.*

Example 5.14 (Example 5.6 continued) An infinite $(\rightarrow_{w_1}^* \circ \rightarrow_{w_2})$-chain that starts from $\mathsf{f}(c[0], c[0], c[0]) = c_1[1, 1]$ is provided in Example 1.5.

Example 5.15 (Example 5.7 (resp 5.8) continued) An infinite $(\hookrightarrow_{w_1}^* \circ \hookrightarrow_{w_2})$-chain that starts from $\mathsf{f}(\mathsf{s}(0), 0) = c_1[1, 0]$ (resp. $\mathsf{f}(\mathsf{c}, \mathsf{a}(\mathsf{c}), \mathsf{a}^2(\mathsf{c})) = c_1[1, 0]$) is provided in Example 5.3 (resp. 5.4).

6 Experimental Evaluation

In [21, 22, 25], we have provided details and experimental evaluations with numbers about the loop detection approach of NTI. Since then, the tool has evolved, as we have fixed several bugs and implemented the syntactic criterion of Sect. 5.1 for detecting binary chains. Hence, we present an updated evaluation based on the results of NTI at the Termination Competition 2023 [31]. NTI participated in two categories.

- Category *Logic Programming* (315 LPs, 2 participants: AProVE[10, 12] and NTI). Here, NTI was the only tool capable of detecting non-termination. In total, it found 58 non-terminating LPs; 45 of them were detected from a loop and the other 13 from a recurrent pair.
- Category *TRS Standard* (1523 TRSs, 7 participants: AProVE, AutoNon[7], MnM[15], MU-TERM[18], NaTT[35], NTI, $\mathsf{T_TT_2}$ [29]).
  We report some results in Table 1. For each participant, column "NO's" reports the number of NO answers, *i.e.*, of TRSs proved non-terminating (with, in parentheses, the number of these TRSs detected by the participant only). The *Virtual Best Solver (VBS)* collects the best consistent claim for each benchmark since at least 2018; column "% VBS" reports the percentage of NO's of each participant w.r.t. the 340 NO's that the VBS collected in total. Then, the following columns report the number of NO's for 4 particular directories of [32]: (E12) corresponds to `TRS_Standard/EEG_IJCAR_12` (49 TRSs), (P21) to `TRS_Standard/payet_21` (3 TRSs), (P23) to `TRS_Standard/Payet_23` (10 TRSs) and (Z15) to `TRS_Standard/Zantema_15` (16 TRSs).
  Globally, the number of non-terminating TRSs detected by NTI is the second highest, just behind AProVE.
  As far as we know, MnM, MU-TERM, NaTT and $\mathsf{T_TT_2}$ only detect loops. We note that the number of loops found by MnM is the highest among these 4 participants. It is even

very likely that it is the highest among all the participants but we could not verify this precisely.

On the other hand, AProVE, AutoNon and NTI are able to detect loops as well as other forms of non-termination (usually called *non-loops*).

In order to detect non-loops, NTI tries to find recurrent pairs while AProVE implements the approach of [6, 19] and AutoNon that of [7] (see Sect. 7 below). To illustrate their technique, the authors of these tools submitted several TRSs admitting no loop to the organizers of the competition (see directory (E12) for AProVE, (P21)+(P23) for NTI and (Z15) for AutoNon). The 10 non-terminating TRSs that NTI is the only one to detect are those of (P23). Moreover, we note that NTI finds 3 recurrent pairs in (Z15), see Example 5.3, and none in (E12)[1], while AProVE fails on the TRSs of (P21)+(P23) and AutoNon succeeds on those of (P21) only. We guess that AutoNon fails on (P23) because all the TRSs of this directory are not left-linear.

The version of NTI that participated in the Termination Competition 2023 relies on a weaker variation of Definition 5.5 where $n_2 = 0$ (actually, the idea of adding n_2 to this definition came after the competition). Since then, we have extended the code to handle n_2 and now NTI is able to find recurrent pairs for 5 TRSs in the directory `TRS_Standard/Waldmann_06` of [32] (`jwno2.xml`, `jwno3.xml`, `jwno5.xml`, `jwno7.xml`, `jwno8.xml`) and also for the TRS `TRS_Standard/Mixed_TRS/6.xml`. Therefore, together with the 3 TRSs in `TRS_Standard/Zantema_15` (see Example 5.3), NTI is able to find recurrent pairs for 9 TRSs that already occurred in [32] before we added (P21) and (P23). All these 9 TRSs are also proved non-terminating by AutoNon but not by AProVE.

7 Related Work

In [21, 25] we have presented an extended description of related work in term rewriting and in logic programming. The state of the art has not evolved much since then. To the best of our knowledge, the only significant new results have been introduced in [6, 7, 19, 34] for string and term rewriting.

- In [19], the author presents an approach to detect infinite chains in string rewriting. It uses rules between string patterns of the form $uv^n w$ where u, v, w are strings and n can be instantiated by any natural number. This idea is extended in [6] to term rewriting. String patterns are replaced by *pattern terms* that describe sets of terms and have the form $t\sigma^n\mu$ where t is a term, σ, μ are substitutions and n is any natural number. A *pattern rule* $(t_1\sigma_1^n\mu_1, t_2\sigma_2^n\mu_2)$ is correct if, for all $n \in \mathbb{N}$, $t_1\sigma_1^n\mu_1 \rightarrow_P^+ t_2\sigma_2^n\mu_2$ holds, where P is the TRS under consideration. Several inference rules are introduced to derive correct pattern rules from a TRS automatically. A sufficient condition on the derived pattern rules is also provided to detect non-termination.
  Currently, we do not have a clear idea of the links between the infinite chains considered in these papers and those of Sect. 5. The experimental results reported in Sect. 6 suggest that these papers address a form of non-termination which is not connected to recurrent pairs. Indeed, AProVE fails on the TRSs of (P21)+(P23) while NTI fails to find recurrent pairs for those of (E12).
- In [7], an automatable approach is presented to prove the existence of infinite chains for TRSs. The idea is to find a non-empty regular language of terms that is closed under

[1] Actually, NTI succeeds on 5 TRSs of (E12) because it implements a simplistic version of [6, 19]

rewriting and does not contain normal forms. It is automated by representing the language by a finite tree automaton and expressing these requirements in a SAT formula whose satisfiability implies non-termination. A major difference w.r.t. our work is that this technique only addresses left-linear TRSs; in contrast, the approach of Sect. 5.1 applies to any ARS which is closed under substitution, and left-linearity is not needed. On the other hand, using the notations of Sect. 5.1, the set $\{c_1[m,n] \mid m,n \in \mathbb{N},\ n \geq n_2\}$ is a non-empty regular language of terms which is closed under $\Rightarrow_{\Pi}^{+}$ and does not contain normal forms (see Proposition 5.12), *i.e.*, it is precisely the kind of set that the approach of [7] tries to find.

- The concept of *inner-looping chain* is presented and studied in [34]. It corresponds to infinite chains that have the form

$$c_1[c_2^{n_0} s\theta^{n_0}] \xrightarrow[P]{+} c_1[c_2^{n_1} s\theta^{n_1}] \xrightarrow[P]{+} \cdots$$

where P is a program, c_1, c_2 are contexts, s is a term, θ is a substitution and the n_i's are non-negative integers (here, $c_2^n s\theta^n = c_2[\cdots c_2[c_2[s\theta]\theta]\theta \cdots]$ where c_2 and θ repeat n times). For instance, the infinite chain of Example 5.2 is inner-looping because it has the form $a_0 \rightarrow_P^+ a_1 \rightarrow_P^+ \cdots$ where, for all $n \in \mathbb{N}$, $a_n = \mathsf{ab}^n\mathsf{dc} = c_1[c_2^n s\theta^n]$ with $c_1 = \mathsf{a}(\square)$, $c_2 = \mathsf{b}(\square)$, $s = \mathsf{dc}$ and $\theta = \emptyset$.
We do not know how inner-looping chains are connected to the infinite chains considered in Sect. 5.

The notion of *recurrent set* [13] used to prove non-termination of imperative programs is also related to our work. It can be defined as follows in our formalism.

Definition 7.1 Let $\mathcal{A} = (A, \Rightarrow_{\Pi})$ be an ARS. A set B is *recurrent for* $\mathcal{A}$ if $B \neq \emptyset$, $B \subseteq A$ and $(\Rightarrow_{\Pi}^{+}(b) \cap B) \neq \emptyset$ for all $b \in B$.

For instance, in Sect. 5.1, the set $\{c_1[m,n] \mid m,n \in \mathbb{N},\ n \geq n_2\}$ is recurrent (see Proposition 5.12). Moreover, the regular languages of terms computed by the approach of [7] are non-empty and closed under rewriting, hence they are also recurrent.

8 Conclusion

We have considered two forms of non-termination, namely, loops and binary chains, in an abstract framework that encompasses term rewriting and logic programming. We have presented a syntactic criterion to detect a special case of binary chains and implemented it successfully in our tool NTI. As for future work, we plan to investigate the connections between the infinite chains considered in [6, 34] and in this paper.

Acknowledgements The author is very grateful to the anonymous reviewers for their insightful and constructive comments and suggestions on this work.

References

1. Apt, K.R.: From Logic Programming to Prolog. Prentice Hall International series in computer science. Prentice Hall, Hoboken (1997)
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
3. Bol, R.N., Apt, K.R., Klop, J.W.: An analysis of loop checking mechanisms for logic programs. Theoret. Comput. Sci. **86**(1), 35–79 (1991)

4. Codish, M., Taboch, C.: A semantic basis for the termination analysis of logic programs. J. Logic Program. **41**(1), 103–123 (1999)
5. Dershowitz, N.: Termination of rewriting. J. Symbol. Comput. **3**(1/2), 69–116 (1987)
6. Emmes, F., Enger, T., Giesl, J.: Proving non-looping non-termination automatically. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR'12). LNCS, vol. 7364, pp. 225–240. Springer, Berlin (2012)
7. Endrullis, J., Zantema, H.: Proving non-termination by finite automata. In: Fernández, M. (ed.) Proceedings of the 26th International Conference on Rewriting Techniques and Applications (RTA'15). Leibniz International Proceedings in Informatics (LIPIcs), vol. 36, pp. 160–176. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015)
8. Gabbrielli, M., Giacobazzi, R.: Goal independency and call patterns in the analysis of logic programs. In: Berghel, H., Hlengl, T., Urban, J.E. (eds.) Proceedings of the 1994 ACM Symposium on Applied Computing (SAC'94), pp. 394–399. ACM, New York (1994)
9. Geser, A., Zantema, H.: Non-looping string rewriting. RAIRO Theoret. Inf. Appl. **33**(3), 279–302 (1999)
10. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. Journal of Automated Reasoning **58**(1), 3–31 (2017)
11. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Gramlich, B. (ed.) Proceedings of the 5th International Workshop on Frontiers of Combining Systems (FroCoS'05). LNAI, vol. 3717, pp. 216–231. Springer, Berlin (2005)
12. Giesl, J., et al.: AProVE (Automated Program Verification Environment). http://aprove.informatik.rwth-aachen.de/ (2023)
13. Gupta, A., Henzinger, T.A., Majumdar, R., Rybalchenko, A., Xu, R.-G.: Proving non-termination. In: Necula, G.C., Wadler, P. (eds.) Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08), pp. 147–158. ACM, New York (2008)
14. Guttag, J.V., Kapur, D., Musser, D.R.: On proving uniform termination and restricted termination of rewriting systems. SIAM J. Comput. **12**(1), 189–214 (1983)
15. Hofbauer, D.: MnM (MultumNonMulta) (2023)
16. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
17. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Berlin (1987)
18. Lucas, S., Gutiérrez, R.: MU-TERM. http://zenon.dsic.upv.es/muterm/ (2019)
19. Oppelt, M.: Automatische Erkennung von Ableitungsmustern in nichtterminierenden Wortersetzungssystemen. Diploma Thesis, HTWK Leipzig, Germany (2008)
20. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) Proceedings of the 5th GI-Conference on Theoretical Computer Science. LNCS, vol. 104, pp. 167–183. Springer, Berlin (1981)
21. Payet, É.: Loop detection in term rewriting using the eliminating unfoldings. Theoret. Comput. Sci. **403**(2–3), 307–327 (2008)
22. Payet, É.: Guided unfoldings for finding loops in standard term rewriting. In: Mesnard, F., Stuckey, P.J. (eds.) Proceedings of the 28th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'18), Revised Selected Papers, LNCS, vol. 11408, pp. 22–37. Springer, Berlin (2018)
23. Payet, É.: Binary non-termination in term rewriting and logic programming. In: Yamada, A. (ed.) Proceedings of the 19th International Workshop on Termination (WST'23) (2023)
24. Payet, É.: NTI (Non-Termination Inference). http://lim.univ-reunion.fr/staff/epayet/Research/NTI/NTI.html and https://github.com/etiennepayet/nti (2023)
25. Payet, É., Mesnard, F.: Nontermination inference of logic programs. ACM Trans. Program. Lang. Syst. **28**(2), 256–289 (2006)
26. Sahlin, D.: The Mixtus approach to automatic partial evaluation of full Prolog. In: Debray, S.K., Hermenegildo, M.V. editors, Proc. of the 1990 North American Conference on Logic Programming, pages 377–398. MIT Press (1990)
27. De Schreye, D., Verschaetse, K., Bruynooghe, M.: A practical technique for detecting non-terminating queries for a restricted class of Horn clauses, using directed, weighted graphs. In: Warren, D.H.D., Szeredi, P. (eds.) Proceedings of the 7th International Conference on Logic Programming (ICLP'90), pp. 649–663. MIT, New York (1990)
28. Shen, Y.-D.: An extended variant of atoms loop check for positive logic programs. New Gen Comput. **15**(2), 187–204 (1997)
29. Sternagel, C., Middeldorp, A.: T_TT_2 (Tyrolean Termination Tool 2). http://cl-informatik.uibk.ac.at/software/ttt2/ (2020)

30. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
31. The Annual International Termination Competition. http://termination-portal.org/wiki/Termination_Competition
32. Termination Problems Data Base. http://termination-portal.org/wiki/TPDB
33. Waldmann, J.: Matchbox: a tool for match-bounded string rewriting. In: van Oostrom, V. (ed.) Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04). LNCS, vol. 3091, pp. 85–94. Springer, Berlin (2004)
34. Wang, Y., Sakai, M.: On non-looping term rewriting. In: Geser, A., Søndergaard, H. (ed.) Proceedings of the 8th International Workshop on Termination (WST'06), pp. 17–21 (2006)
35. Yamada, A.: NaTT (Nagoya Termination Tool). https://www.trs.css.i.nagoya-u.ac.jp/NaTT/ (2023)
36. Zankl, H., Middeldorp, A.: Nontermination of string rewriting using SAT. In: Hofbauer, D., Serebrenik, A. (ed.) Proceedings of the 9th International Workshop on Termination (WST'07), pp. 52–55 (2007)
37. Zantema, H.: Termination of string rewriting proved automatically. J. Automat. Reason. **34**(2), 105–139 (2005)
38. Zantema, H., Geser, A.: Non-looping rewriting. Universiteit Utrecht. UU-CS, Department of Computer Science. Utrecht University, The Netherlands (1996)