Recurrent Pairs Revisited

Etienne Payet[0000−0002−3519−025X]

LIM - Université de la Réunion, France
etienne.payet@univ-reunion.fr
http://lim.univ-reunion.fr/staff/epayet/

Abstract. In this short paper, we consider non-looping non-termination in term rewriting and logic programming. We describe a strict generalisation of the recurrent pair approach that we introduced previously. We also present an experimental evaluation of our contribution that we implemented in our tool NTI.

Keywords: Non-Termination · Non-Looping · Term Rewriting · Logic Programming

1 Introduction

This paper is concerned with non-termination in term rewriting and logic programming, where one rewrites terms or sequences of terms according to the operational semantics described, *e.g.*, in [1, 2]. Rewriting is formalised by binary relations $\Rightarrow_r$ indexed by rules r from a given program and non-termination as the existence of an infinite rewrite sequence $t_0 \Rightarrow_{r_1} t_1 \Rightarrow_{r_2} \cdots$. The vast majority of the papers related to this topic provide necessary or sufficient conditions for the existence of *loops*, *i.e.*, finite rewrite sequences $t_0 \Rightarrow_{r_1} \cdots \Rightarrow_{r_n} t_n$ where t_n satisfies a condition C which entails the possibility of starting again, *i.e.*, $t_n \Rightarrow_{r_1} \cdots \Rightarrow_{r_n} t_{2n}$ where t_{2n} also satisfies C, and so on.

In this paper, we rather consider *non-looping* non-termination, *i.e.*, infinite rewrite sequences that do not embed any loop. Our contribution is a strict extension of an approach introduced previously [8, 9] for detecting non-termination of the form $t_0 \, (\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2}) \, t_1 \, (\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2}) \cdots$, where w_1 and w_2 are finite sequences of rules. We illustrate it by the following example.

Example 1. Consider the term rewrite system or binary logic program consisting of the following rules (*i.e.*, pairs of terms):

$$r_1 = (u_1, v_1) = \big(\mathsf{f}(x, \mathsf{s}^3(y)), \mathsf{f}(\mathsf{s}^2(x), \mathsf{s}(y))\big)$$
$$r_2 = (u_2, v_2) = \big(\mathsf{f}(x, \mathsf{s}^3(\mathsf{0})), \mathsf{f}(\mathsf{0}, \mathsf{s}^7(x))\big)$$

Here, x and y are variables and f, s and $\mathsf{0}$ are function symbols of arity 2, 1 and 0 respectively. Moreover, by s^n we mean n successive applications of s. Let $\Delta = 2$. We observe that there is a move of s's between the arguments of f. From u_1 to v_1, Δ occurrences of s are added at the root position of the first argument of f

while Δ occurrences are removed from the root position of the second one. On the other hand, suppose that in r_2 the term bound to x has the form $\mathsf{s}^m(\cdots)$. Then, in v_2, the number of occurrences of s at the root position of the second argument of f is $7 + m = \Delta \times 2 + m + 3$; so, if $m = \Delta \times n$ for some $n \in \mathbb{N}$, then we have $7 + m = \Delta \times (n + 2) + 3$. Therefore, for all naturals n, we have the infinite rewrite sequence:

$$\begin{aligned}\mathsf{f}\left(0, \mathsf{s}^{\Delta\times n+3}(0)\right) &\overset{n}{\underset{r_1}{\Rightarrow}} \mathsf{f}\left(\mathsf{s}^{\Delta\times n}(0), \mathsf{s}^3(0)\right) \underset{r_2}{\Rightarrow} \mathsf{f}\left(0, \mathsf{s}^{\Delta\times(n+2)+3}(0)\right)\\ &\overset{n+2}{\underset{r_1}{\Rightarrow}} \mathsf{f}\left(\mathsf{s}^{\Delta\times(n+2)}(0), \mathsf{s}^3(0)\right) \underset{r_2}{\Rightarrow} \mathsf{f}\left(0, \mathsf{s}^{\Delta\times(n+4)+3}(0)\right)\\ &\overset{n+4}{\underset{r_1}{\Rightarrow}} \cdots\end{aligned}$$

It has the form $t_0 \left(\Rightarrow_{r_1}^* \circ \Rightarrow_{r_2}\right) t_1 \left(\Rightarrow_{r_1}^* \circ \Rightarrow_{r_2}\right) \cdots$ and does not embed a loop, because the number of applications of r_1 gradually increases. The approach of this paper detects its existence (see Def. 3 and Cor. 1), contrary to that of [8, 9] which only considers the case $\Delta = 1$ (Def. 3 considers any $\Delta \in \mathbb{N}$).

The paper is organised as follows. Section 2 provides some basic definitions and notations, Sect. 3 describes an extension of the *recurrent pair* approach introduced in [8, 9], Sect. 4 presents an experimental evaluation, Sect. 5 briefly introduces related work and Sect. 6 concludes with future work.

2 Preliminaries

We let $\mathbb{N}$ denote the set of natural numbers. Let A be a set. Then, $\overline{A}$ is the set of finite sequences of elements of A, which includes the empty sequence, denoted as e. We use the delimiters $\langle$ and $\rangle$ for writing elements of $\overline{A}$ and juxtaposition to denote the concatenation operation, *e.g.*, $\langle a_0, a_1\rangle \langle a_2, a_3\rangle = \langle a_0, a_1, a_2, a_3\rangle$. We generally denote elements of $\overline{A}$ using lowercase letters with an overline, *e.g.*, $\overline{a}$.

2.1 Binary Relations

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, $\phi^+ = \bigcup \{\phi^n \mid n > 0\}$ (resp. $\phi^* = \phi^0 \cup \phi^+$) denotes the transitive (resp. reflexive and transitive) *closure* of ϕ. A *ϕ-chain*, or simply *chain* if ϕ is clear from the context, is a (possibly infinite) sequence of elements of A such that $(a, a') \in \phi$ for any two consecutive elements a, a' (hence, the empty sequence and the singletons are chains). For binary relations that have the form of an arrow, *e.g.*, $\Rightarrow$, we may write chains $a_0, a_1, \ldots$ as $a_0 \Rightarrow a_1 \Rightarrow \cdots$.

2.2 Terms and Substitutions

We use the same definitions and notations as [2] for terms. A *signature* is a set of *function symbols*, each element of which has an *arity* in $\mathbb{N}$. The 0-ary elements of a signature are called *constant symbols*. We denote function symbols by letters or digits in the *sans serif* font, *e.g.*, f, $\mathsf{0}\ldots$

Let Σ be a signature and X be a set of *variables* disjoint from Σ. For $m \in \mathbb{N}$, we let $\Sigma^{(m)}$ denote the set of all m-ary elements of Σ. The set $T(\Sigma, X)$ of Σ-*terms over* X is defined as: $X \subseteq T(\Sigma, X)$ and, for all $m \in \mathbb{N}$, all $\mathsf{f} \in \Sigma^{(m)}$ and all $s_1, \ldots, s_m \in T(\Sigma, X)$, $\mathsf{f}(s_1, \ldots, s_m) \in T(\Sigma, X)$. For all $s \in T(\Sigma, X)$, we let $Var(s)$ denote the set of variables occurring in s. We 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 $\mathsf{s}^0(\mathsf{0}) = \mathsf{0}$.

A $T(\Sigma, X)$-*substitution* (or simply *substitution* if the set of terms is irrelevant or clear from the context) is a function θ from X to $T(\Sigma, X)$ such that $\theta(x) \neq x$ for only finitely many variables x. The *domain* of θ is $Dom(\theta) = \{x \in X \mid \theta(x) \neq x\}$. We usually write θ as $\{x_1 \mapsto \theta(x_1), \ldots, x_n \mapsto \theta(x_m)\}$ where $\{x_1, \ldots, x_m\} = Dom(\theta)$ (hence, the identity substitution is written as $\emptyset$). A *(variable) renaming* is a substitution that is a bijection on X. We let $S(\Sigma, X)$ denote the set of all $T(\Sigma, X)$-substitutions.

The application of $\theta \in S(\Sigma, X)$ to $s \in T(\Sigma, X)$ is denoted as $s\theta$ and is defined as: $s\theta = \theta(s)$ if $s \in X$ and $s\theta = \mathsf{f}(s_1\theta, \ldots, s_m\theta)$ if $s = \mathsf{f}(s_1, \ldots, s_m)$. Application of θ is extended to finite sequences of terms, *i.e.*, $\langle s_1, \ldots, s_m \rangle \theta = \langle s_1\theta, \ldots, s_m\theta \rangle$.

The *composition* of $\sigma, \theta \in S(\Sigma, X)$ is the $T(\Sigma, X)$-substitution denoted as $\sigma\theta$ and defined as: for all $x \in X$, $\sigma\theta(x) = (\sigma(x))\theta$. This is an associative operation, *i.e.*, for all $s \in T(\Sigma, X)$, $(s\sigma)\theta = s(\sigma\theta)$. We say that σ is *more general* than θ if $\theta = \sigma\eta$ for some $\eta \in S(\Sigma, X)$.

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. Moreover, σ is called a *most general unifier* (mgu) of s and t if it is a unifier of s and t that is more general than all unifiers of s and t. We let $mgu(s, t)$ denote the set of mgu's of s and t.

2.3 The Signature Used in the Paper

We regard the symbol e denoting the empty sequence as a special constant symbol. To simplify the statements of this paper, from now on we fix a signature Σ and a set $H = \{\Box_n \mid n \in \mathbb{N} \setminus \{0\}\}$ of constant symbols (called *holes*) such that Σ, $\{\mathsf{e}\}$ and H are disjoint from each other. We also fix an infinite countable set X of variables disjoint from $\Sigma \cup \{\mathsf{e}\} \cup H$. A *term* is an element of $T(\Sigma, X)$. Let n be a positive integer. An n-*context* is an element of $T(\Sigma \cup H, X)$ that contains occurrences of $\Box_1, \ldots, \Box_n$ but no occurrence of another hole. For all n-contexts c and all $s_1, \ldots, s_n \in T(\Sigma \cup H, X)$, we let $c(s_1, \cdots, s_n)$ denote the element of $T(\Sigma \cup H, X)$ obtained from c by replacing all the occurrences of $\Box_i$ by s_i, for all $1 \leq i \leq n$. We use the superscript notation for denoting several successive embeddings of a 1-context c into itself: $c^0 = \Box_1$ and, for all $n \in \mathbb{N}$, $c^{n+1} = c(c^n)$.

Terms are generally denoted by s, t, u, v, variables by x, y and contexts by c, possibly with subscripts and primes.

2.4 Term Rewriting and Logic Programming

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

Definition 1. *A* program *is a subset of* $T(\Sigma, X) \times \overline{T(\Sigma, X)}$*, every element of which is called a* rule*. A rule* $(u, \overline{v})$ *is* binary *if* $\overline{v}$ *is empty or is a singleton. We let* $\Re$ *denote the set of binary rules. For the sake of readability, we omit the delimiters* $\langle$ *and* $\rangle$ *in the right-hand side of a binary rule, which amounts to considering that* $\Re \subseteq T(\Sigma, X) \times (T(\Sigma, X) \cup \{\mathsf{e}\})$*.*

Given a rule $(u, \overline{v})$, we let $[(u, \overline{v})] = \{(u\gamma, \overline{v}\gamma) \mid \gamma \text{ is a renaming}\}$ denote its *equivalence class modulo renaming*. Moreover, for all sets of rules U and sequences of terms S, we write $\overline{r} \ll_S U$ to denote that $\overline{r}$ is a sequence of elements of U variable disjoint from S and from each other.

The rules of a program allow one to rewrite terms and finite sequences of terms. This is formalised by the following binary relations.

Definition 2. *For all programs* P *and all* $\Rightarrow \in \{\rightarrow, \rightsquigarrow, \hookrightarrow\}$*, we let* $\Rightarrow_P = \bigcup\{\Rightarrow_r \mid r \in P\}$ *where, for all* $r \in P$*,*

$$\underset{r}{\rightarrow} = \left\{ \big(s, c(v\theta)\big) \in T(\Sigma, X)^2 \;\middle|\; \begin{array}{l} r = (u, \langle v \rangle),\ s = c(u\theta),\ \theta \in S(\Sigma, X) \\ c \text{ is a 1-context with exactly one } \Box_1 \end{array} \right\}$$

$$\underset{r}{\rightsquigarrow} = \left\{ \big(\langle s \rangle\, \overline{s}, (\overline{v}\,\overline{s})\theta\big) \in \overline{T(\Sigma, X)}^2 \;\middle|\; \langle (u, \overline{v}) \rangle \ll_{\langle s \rangle \overline{s}} [r],\ \theta \in mgu(u, s) \right\}$$

$$\underset{r}{\hookrightarrow} = \left\{ \big(u\theta, v\theta\big) \in \Re \;\middle|\; r = (u, v) \in \Re,\ Var(v) \subseteq Var(u),\ \theta \in S(\Sigma, X) \right\}$$

For instance, in Ex. 1, we have $Var(v_1) \subseteq Var(u_1)$. So, for $\theta = \{x \mapsto \mathsf{0}, y \mapsto \mathsf{s}(x)\}$, we have $u_1\theta \hookrightarrow_{r_1} v_1\theta$ where $u_1\theta = \mathsf{f}(\mathsf{0}, \mathsf{s}^4(x))$ and $v_1\theta = \mathsf{f}(\mathsf{s}^2(\mathsf{0}), \mathsf{s}^2(x))$. We also have $u_1\theta = c(u_1\theta)$ and $v_1\theta = c(v_1\theta)$ for the 1-context $c = \Box_1$. Hence, we have $u_1\theta \rightarrow_{r_1} v_1\theta$. On the other hand, let $u_1' = \mathsf{f}(x', \mathsf{s}^3(y))$ and $v_1' = \mathsf{f}(\mathsf{s}^2(x'), \mathsf{s}(y))$. Then, we have $\langle (u_1', v_1') \rangle \ll_{\langle u_1\theta \rangle} [r_1]$ and $\theta' \in mgu(u_1', u_1\theta)$ where $\theta' = \{x' \mapsto \mathsf{0}, y \mapsto \mathsf{s}(x)\}$. So, we have $\langle u_1\theta \rangle \rightsquigarrow_{r_1} \langle v_1'\theta' \rangle$ where $v_1'\theta' = v_1\theta$ (in this example, the sequence $\overline{s}$ of the definition of $\rightsquigarrow$ is empty).

The binary relation $\rightarrow_P$ (resp. $\rightsquigarrow_P$) corresponds to the operational semantics of term rewriting (resp. logic programming with the leftmost selection rule). In the proofs of Sect. 3, we need the *closure under substitutions* property (Lemma 1 below). Contrary to $\rightarrow_P$, the relation $\rightsquigarrow_P$ does not satisfy it. This is why we introduce $\hookrightarrow_P$: it satisfies this property and is a restriction of $\rightsquigarrow_P$ to binary rules (*i.e.*, for all $r \in P$, $s \hookrightarrow_r t$ implies $\langle s \rangle \rightsquigarrow_r \overline{t}$ where $\overline{t} = \mathsf{e}$ if $t = \mathsf{e}$ and $\overline{t} = \langle t \rangle$ otherwise, see Lemma 2.15 of [9]). We also note that $\hookrightarrow_P$ is a restriction of $\rightarrow_P$ (*i.e.*, $\hookrightarrow_r \subseteq \rightarrow_r$ for all $r \in P$) where one rewrites terms at the root position only.

Lemma 1. *Let P be a program and $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$. Then, $\Rightarrow_P$ is* closed under substitutions*, i.e., $s \Rightarrow_P t$ implies $s\theta \Rightarrow_P t\theta$ for all terms s, t and substitutions θ.*

In the rest of this paper, given a program P, we only consider the relations $\rightarrow_P$ and $\hookrightarrow_P$. In logic programming with the leftmost selection rule, one may consider the *binary unfolding* [3]of P, denoted as $binunf(P)$, which is a set of binary rules obtained from P that enjoys the following property:

Theorem 1 ([3]). *Let P be a program and $\overline{s}$ be a sequence of terms. Then, there is an infinite $\rightsquigarrow_P$-chain that starts from $\overline{s}$* if and only if *there is an infinite $\rightsquigarrow_{binunf(P)}$-chain that starts from $\overline{s}$.*

Hence, the existence of an infinite $\hookrightarrow_{binunf(P)}$-chain that starts from a term s implies that of an infinite $\rightsquigarrow_{binunf(P)}$-chain that starts from $\langle s \rangle$, which itself implies that of an infinite $\rightsquigarrow_P$-chain that starts from $\langle s \rangle$.

3 Extending Recurrent Pairs

Let P be a program and $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$. In this section, we describe a technique for detecting infinite $\Rightarrow_P$-chains using a strict extension of recurrent pairs [8, 9]. For all $w \in \overline{P}$, we let $\Rightarrow_w$ be the identity relation if w is empty and $\Rightarrow_w = (\Rightarrow_{r_1} \circ \cdots \circ \Rightarrow_{r_n})$ if $w = \langle r_1, \ldots, r_n \rangle$ with $n \geq 1$. So, any finite non-empty $\Rightarrow_P$-chain has the form $s \Rightarrow_w t$ for some $s, t \in T(\Sigma, X)$ and some $w \in \overline{P}$. Moreover, for all $m, n \in \mathbb{N}$, we let $[n]_m$ be the set of integers greater than n that are *congruent to n modulo m, i.e.*, $[n]_m = \{n + k \times m \mid k \in \mathbb{N}\}$.

Definition 3. *Let P be a program and $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$. A* recurrent pair for $\Rightarrow_P$ *is a pair $(u_1 \Rightarrow_{w_1} v_1, u_2 \Rightarrow_{w_2} v_2)$ of finite non-empty $\Rightarrow_P$-chains such that*

- $u_1 = c_1(x, c_2^{m_1}(y))$ *and* $v_1 = c_1(c_2^{n_1}(x), c_2^{n_2}(y))$,
- $u_2 = c_1(x, c_2^{m_2}(s))$ *and* $v_2 = c_1(c_2^{n_3}(t), c_2^{n_4}(x))$ *with* $t \in \{s, x\}$,
- c_1 *is a 2-context,* c_2 *is a 1-context,* $x, y \in X$, $s \in T(\Sigma, X)$,
- $x \neq y$ *and* $\{x, y\} \cap Var(c_1, c_2, s) = \emptyset$,
- $\{m_1, m_2, n_1, n_2, n_3, n_4\} \subseteq \mathbb{N}$,
- $n_2 \leq m_1$, $n_2 \leq m_2$ *and* $\{n_1, n_3, n_4 - m_2\} \subseteq [0]_\Delta$ *where* $\Delta = (m_1 - n_2)$.

This definition is a strict generalisation of that provided in [9], which itself is a strict generalisation of that of [8] (we always have $(m_1, n_2, m_2) = (1, 0, 0)$ in [8] and $(m_1, n_2) = (1, 0)$ in [9]). Intuitively, $u_1 \Rightarrow_{w_1} v_1$ and $u_2 \Rightarrow_{w_2} v_2$ are "mutually recursive" $\Rightarrow_P$-chains where occurrences of context c_2 are moved between $\Box_1$ and $\Box_2$ in c_1 (see Lemma 2 and Lemma 4 below). In some very special situations, the existence of a recurrent pair implies that of a loop, *e.g.*, if $n_3 = 0$ and $m_2 = n_4$ then we have $c_1(s, c_2^{m_2}(s)) \Rightarrow_{w_2} c_1(s, c_2^{m_2}(s)) \Rightarrow_{w_2} \cdots$. But this is not always the case: in this section, we prove that the existence of a recurrent pair implies that of an infinite $(\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2})$-chain (see Cor. 1).

Example 2. In Ex. 1, let $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$ and $P = \{r_1, r_2\}$. We have $u_1 \Rightarrow_{r_1} v_1$ and $u_2 \Rightarrow_{r_2} v_2$. Moreover, the pair $(u_1 \Rightarrow_{r_1} v_1, u_2 \Rightarrow_{r_2} v_2)$ is recurrent for $\Rightarrow_P$ with $c_1 = \mathsf{f}(\Box_1, \Box_2)$, $c_2 = \mathsf{s}(\Box_1)$, $s = t = \mathsf{0}$, $m_1 = m_2 = 3$, $(n_1, n_2, n_3, n_4) = (2, 1, 0, 7)$ and $\Delta = (m_1 - n_2) = 2$.

Example 3. Let $\Rightarrow \in \{\to, \hookrightarrow\}$ and P be the program consisting of the rules

$$r_1 = (u_1, v_1) = \big(\mathsf{f}(x, \mathsf{s}^2(y)), \mathsf{f}(x, \mathsf{s}(y))\big)$$
$$r_2 = (u_2, v_2) = (\mathsf{f}(x, \mathsf{s}(\mathsf{0})), \mathsf{f}(\mathsf{s}(x), \mathsf{s}(x)))$$

We have $u_1 \Rightarrow_{r_1} v_1$ and $u_2 \Rightarrow_{r_2} v_2$. Moreover, the pair $(u_1 \Rightarrow_{r_1} v_1, u_2 \Rightarrow_{r_2} v_2)$ is recurrent for $\Rightarrow_P$ with $c_1 = \mathsf{f}(\Box_1, \Box_2)$, $c_2 = \mathsf{s}(\Box_1)$, $s = \mathsf{0}$, $t = x$, $(m_1, m_2) = (2, 1)$, $(n_1, n_2, n_3, n_4) = (0, 1, 1, 1)$ and $\Delta = (m_1 - n_2) = 1$.

Example 4. Let $\Rightarrow \in \{\to, \hookrightarrow\}$ and P be the program consisting of the rules

$$r_1 = (u_1, v_1) = \big(\mathsf{f}(x, \mathsf{s}^7(y)), \mathsf{f}(\mathsf{s}^3(x), \mathsf{s}^4(y))\big)$$
$$r_2 = (u_2, v_2) = \big(\mathsf{f}(x, \mathsf{s}^6(\mathsf{0})), \mathsf{f}(x, \mathsf{s}^{11}(x))\big)$$

We have $u_1 \Rightarrow_{r_1} v_1$ and $u_2 \Rightarrow_{r_2} v_2$. Moreover, the pair $(u_1 \Rightarrow_{r_1} v_1, u_2 \Rightarrow_{r_2} v_2)$ is recurrent for $\Rightarrow_P$ with $c_1 = \mathsf{f}(\Box_1, \Box_2)$, $c_2 = \mathsf{s}(\Box_1)$, $s = \mathsf{s}(\mathsf{0})$, $t = x$, $(m_1, m_2) = (7, 5)$, $(n_1, n_2, n_3, n_4) = (3, 4, 0, 11)$ and $\Delta = (m_1 - n_2) = 3$. Note that if we choose $s = \mathsf{0}$ instead then we get $m_2 = 6$ and we have $(n_4 - m_2) \notin [0]_\Delta$.

The next statements are parametric in a program P, in $\Rightarrow \in \{\to, \hookrightarrow\}$ and in a recurrent pair for $\Rightarrow_P$, with the notations of Def. 3 as well as this new one, introduced for the sake of readability:

Definition 4. *For all $m, n \in \mathbb{N}$, $c_1(m, n)$ denotes the term $c_1(c_2^m(s), c_2^n(s))$.*

First, we introduce a couple of technical lemmas stating properties of the relations $\Rightarrow_{w_1}$ and $\Rightarrow_{w_2}$. Their proofs are based on the fact that $\to$ and $\hookrightarrow$ are closed under substitutions (see Lemma 1).

Lemma 2. *For all $m, n \in \mathbb{N}$ with $m_1 \leq n$, we have*

$$c_1(m, n) \underset{w_1}{\Rightarrow} c_1(m + n_1, n - \Delta)$$

Proof. Let $m, n \in \mathbb{N}$ with $m_1 \leq n$. Then, $c_1(m, n) = c_1(c_2^m(s), c_2^n(s)) = u_1\theta$ where $\theta = \{x \mapsto c_2^m(s), y \mapsto c_2^{n-m_1}(s)\}$. So, as $u_1 \Rightarrow_{w_1} v_1$, by Lemma 1 we have $c_1(m, n) \Rightarrow_{w_1} v_1\theta$ where $v_1\theta = c_1\left(c_2^{m+n_1}(s), c_2^{n_2+n-m_1}(s)\right) = c_1(m+n_1, n_2+n-m_1)$, where $n_2 + n - m_1 = n - \Delta$. □

By iterating the application of Lemma 2, one gets the following result.

Lemma 3. *For all $m \in \mathbb{N}$ and $n \in [m_2]_\Delta$, there exists $k \in \mathbb{N}$ such that*

$$c_1(m, n) \underset{w_1}{\overset{k}{\Rightarrow}} c_1(m + k \times n_1, m_2)$$

Proof. Let $m \in \mathbb{N}$ and $n \in [m_2]_\Delta$. Then, we have $n = m_2 + k\Delta$ for some $k \in \mathbb{N}$.

- Suppose that $m_1 \leq m_2$. Then, we have $m_1 \leq m_2 \leq n$. Therefore, one can apply Lemma 2 k times to get $c_1(m, n) \Rightarrow_{w_1}^k c_1(m + k \times n_1, n - k\Delta)$, with $(n - k\Delta) = m_2$.

– Suppose that $m_2 < m_1$.
  - If $k = 0$ then $n = m_2$ and $c_1(m,n) \Rightarrow_{w_1}^k c_1(m + k \times n_1, m_2)$.
  - Otherwise, $m_2 + \Delta = m_2 + (m_1 - n_2) = m_1 + (m_2 - n_2)$, *i.e.*, $m_2 + \Delta - m_1 = m_2 - n_2$. But, as $n_2 \leq m_2$, we have $0 \leq (m_2 - n_2)$. So, $0 \leq (m_2 + \Delta - m_1)$, *i.e.*, $m_1 \leq (m_2 + \Delta)$. Consequently, we have $m_2 < m_1 \leq (m_2 + \Delta) \leq n$. So, one can apply Lemma 2 k times to get $c_1(m,n) \Rightarrow_{w_1}^k c_1(m + k \times n_1, n - k\Delta)$, with $(n - k\Delta) = m_2$. □

Lemma 4. *For all $m \in \mathbb{N}$, we have $c_1(m, m_2) \Rightarrow_{w_2} c_1(n_3 + m', n_4 + m)$ where $m' = 0$ if $t = s$ and $m' = m$ if $t = x$.*

Proof. Let $m \in \mathbb{N}$. We have $c_1(m, m_2) = c_1(c_2^m(s), c_2^{m_2}(s)) = u_2\{x \mapsto c_2^m(s)\}$. So, as $u_2 \Rightarrow_{w_2} v_2$, by Lemma 1 we have $c_1(m, m_2) \Rightarrow_{w_2} v_2\{x \mapsto c_2^m(s)\}$.

– If $t = s$ then $v_2\{x \mapsto c_2^m(s)\} = c_1(c_2^{n_3}(s), c_2^{n_4+m}(s)) = c_1(n_3, n_4 + m)$.
– If $t = x$ then $v_2\{x \mapsto c_2^m(s)\} = c_1(c_2^{n_3+m}(s), c_2^{n_4+m}(s)) = c_1(n_3 + m, n_4 + m)$.
□

By combining Lemma 3 and Lemma 4, one gets the next proposition.

Proposition 1. *For all $m \in [0]_\Delta$ and $n \in [m_2]_\Delta$, there exist $m' \in [0]_\Delta$ and $n' \in [m_2]_\Delta$ such that $c_1(m,n)\,(\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2})\, c_1(m', n')$.*

Proof. Let $m \in [0]_\Delta$ and $n \in [m_2]_\Delta$. By Lemma 3, there exists $k \in \mathbb{N}$ such that $c_1(m,n) \Rightarrow_{w_1}^k c_1(l, m_2)$ and $l = m + k \times n_1$. By Lemma 4, $c_1(l, m_2) \Rightarrow_{w_2} c_1(n_3 + l', n_4 + l)$ where $l' = 0$ if $t = s$ and $l' = l$ if $t = x$. Therefore, for $m' = n_3 + l'$ and $n' = n_4 + l$, we have $c_1(m,n)\,(\Rightarrow_{w_1}^k \circ \Rightarrow_{w_2})\, c_1(m', n')$. We note that $m' \in [0]_\Delta$ because $\{m, n_1, n_3\} \subseteq [0]_\Delta$. Moreover, $(n_4 - m_2) \in [0]_\Delta$ implies that $n_4 \in [m_2]_\Delta$; hence, as $l \in [0]_\Delta$ (because $\{m, n_1\} \subseteq [0]_\Delta$), we have $(n_4 + l) \in [m_2]_\Delta$, *i.e.*, $n' \in [m_2]_\Delta$. □

The main result of this paper is a straightforward consequence of Prop. 1:

Corollary 1. *Let P be a program, $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$ and $(u_1 \Rightarrow_{w_1} v_1, u_2 \Rightarrow_{w_2} v_2)$ be a recurrent pair for $\Rightarrow_P$, with the notations of Def. 3. Then, for all $m \in [0]_\Delta$ and $n \in [m_2]_\Delta$, the term $c_1(m,n)$ starts an infinite $(\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2})$-chain.*

This result implies the existence of infinite chains in the above examples. Each of these chains does not embed any loop because the number of applications of r_1 gradually increases.

Example 5. In Ex. 1-2, we have the following infinite $(\Rightarrow_{r_1}^* \circ \Rightarrow_{r_2})$-chain:

$$\overbrace{\mathsf{f}(0, \mathsf{s}^3(0))}^{c_1(0,m_2)}\,(\underset{r_1}{\overset{0}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow})\,\mathsf{f}(0, \mathsf{s}^7(0))\underset{r_1}{\overset{2}{\Rightarrow}}\mathsf{f}(\mathsf{s}^4(0), \mathsf{s}^3(0))\underset{r_2}{\Rightarrow}\mathsf{f}(0, \mathsf{s}^{11}(0))\underset{r_1}{\overset{4}{\Rightarrow}}\cdots$$

It corresponds to the case $n = 0$ in Ex. 1.

Example 6. In Ex. 3, we have the following infinite $(\Rightarrow_{r_1}^* \circ \Rightarrow_{r_2})$-chain:

$$\overbrace{\mathsf{f}(0,\mathsf{s}(0))}^{c_1(0,m_2)} (\underset{r_1}{\overset{0}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow}) \mathsf{f}(\mathsf{s}(0),\mathsf{s}(0)) (\underset{r_1}{\overset{0}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow}) \mathsf{f}(\mathsf{s}^2(0),\mathsf{s}^2(0)) \underset{r_1}{\overset{1}{\Rightarrow}} \mathsf{f}(\mathsf{s}^2(0),\mathsf{s}(0)) \underset{r_2}{\Rightarrow} \mathsf{f}(\mathsf{s}^3(0),\mathsf{s}^3(0)) \underset{r_1}{\overset{2}{\Rightarrow}} \cdots$$

Example 7. In Ex. 4, we have the following infinite $(\Rightarrow_{r_1}^* \circ \Rightarrow_{r_2})$-chain:

$$\overbrace{\mathsf{f}(\mathsf{s}(0),\mathsf{s}^6(0))}^{c_1(0,m_2)} (\underset{r_1}{\overset{0}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow}) \mathsf{f}(\mathsf{s}(0),\mathsf{s}^{12}(0)) \underset{r_1}{\overset{2}{\Rightarrow}} \mathsf{f}(\mathsf{s}^7(0),\mathsf{s}^6(0)) \underset{r_2}{\Rightarrow} \mathsf{f}(\mathsf{s}^7(0),\mathsf{s}^{18}(0)) \underset{r_1}{\overset{4}{\Rightarrow}} \cdots$$

4 Experimental Evaluation

We have implemented the approach of Sect. 3 in our tool NTI for term rewrite systems (TRSs) and logic programs (LPs). More precisely, NTI unfolds the program under analysis (using a user-provided time limit to stop this possibly infinite process) and tries to detect pairs of unfolded rules $((u_1, v_1), (u_2, v_2))$ that satisfy Def. 3. Since we started to work on this topic a few years ago, we have added several examples of non-looping non-terminating TRSs and LPs to the TPDB.[1] They are all similar to the programs of Ex. 1-4. In Table 1, we report the results of the analysers participating in the *International Termination Competition*[2] that are capable of detecting non-looping non-terminating TRSs (NTI'24 refers to the version of NTI that implements the approach of this paper). We point out that AProVE and NTI are the only two tools that participate in the "logic programming" category of the competition and that AProVE is not able to detect non-termination of LPs. We note that NTI'24 is the only tool that succeeds on all benchmarks. Our results can be reproduced using our tool and the benchmarks available online.[3]

5 Related Work

Apart from [8, 9], we are aware of only a few papers dealing with non-looping non-termination: [7, 12] in the field of string rewriting, [4, 5, 11, 12] in term rewriting and [10] in logic programming. How all these approaches are related to ours is an open question that we leave for future work.

6 Conclusion

We have presented a natural, strict, generalisation of the recurrent pairs of [8, 9] to encompass a broader range of infinite rewrite sequences, specifically those

[1] Termination Problem Data Base: http://termination-portal.org/wiki/TPDB
[2] http://termination-portal.org/wiki/Termination_Competition
[3] https://github.com/etiennepayet/nti

Table 1. Results of the Termination Competition 2024 on the TRSs and LPs we added to the TPDB (time limit = 300 seconds, see https://termcomp.github.io/Y2024/). The last column on the right gives the values of Δ for the recurrent pairs found by NTI.

Directory of the TPDB	AProVE [6]	AutoNon [5]	NTI'23 [9]	NTI'24	Δ
TRS_Standard/payet_21	0/3	3/3	3/3	3/3	$\{1\}$
TRS_Standard/Payet_23	0/10	0/10	10/10	10/10	$\{1\}$
TRS_Standard/Payet_24	1/5	0/5	0/5	5/5	$\{1,2,3,5\}$
Logic_Programming/Payet_22	0/5	-	4/5	5/5	$\{1\}$
Logic_Programming/Payet_23	0/9	-	9/9	9/9	$\{1\}$
Logic_Programming/Payet_24	0/4	-	0/4	4/4	$\{2,3,5\}$

where Δ can be any natural number, rather than being restricted to 1. On the theoretical side, this leads to the definition of a wider class of non-terminating behaviours, compared to [8, 9]. On the practical side, this allows the detection of infinite rewrite sequences that were not detectable by previous approaches. We still have to investigate whether these infinite sequences correspond to situations arising in real programs or in programs introduced by other sources than ourselves (in the TPDB, our extension only solves problems that we specifically added to illustrate our contribution, contrary to the approach of [8, 9] that solves problems from, *e.g.*, [12]).

Future work will also focus on extending our approach to deal with more general situations, *e.g.*, by considering recurrent tuples with more than two rules to detect infinite rewrite sequences $t_0 \rightarrowtail t_1 \rightarrowtail \cdots$ where $\rightarrowtail$ has the form $(\Rightarrow_{w_1}^* \circ \cdots \circ \Rightarrow_{w_{n-1}}^* \circ \Rightarrow_{w_n})$ for some natural $n \geq 1$, some $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$ and some finite sequences of rules $w_1, \ldots, w_n$.

Example 8. Consider the following variant of Ex. 1:

$$\begin{aligned} r_1 = (u_1, v_1) &= \big(\mathsf{f}(x, \mathsf{s}^3(y), z), \mathsf{f}(\mathsf{s}^2(x), \mathsf{s}(y), z)\big) \\ r_2 = (u_2, v_2) &= \big(\mathsf{f}(x, y, \mathsf{s}^3(z)), \mathsf{f}(x, y, \mathsf{s}(z))\big) \\ r_3 = (u_3, v_3) &= \big(\mathsf{f}(x, \mathsf{s}^3(\mathsf{0}), \mathsf{s}^3(\mathsf{0})), \mathsf{f}(\mathsf{0}, \mathsf{s}^7(x), \mathsf{s}^7(x))\big) \end{aligned}$$

Let $\Delta = 2$ and $\Rightarrow \in \{\rightarrow, \hookrightarrow\}$. For all $n \in \mathbb{N}$, we have the infinite rewrite sequence

$$\begin{aligned} \mathsf{f}\left(\mathsf{0}, \mathsf{s}^{\Delta\times n+3}(\mathsf{0}), \mathsf{s}^{\Delta\times n+3}(\mathsf{0})\right) &(\overset{n}{\underset{r_1}{\Rightarrow}} \circ \overset{n}{\underset{r_2}{\Rightarrow}}) \mathsf{f}\left(\mathsf{s}^{\Delta\times n}(\mathsf{0}), \mathsf{s}^3(\mathsf{0}), \mathsf{s}^3(\mathsf{0})\right) \\ &\underset{r_3}{\Rightarrow} \mathsf{f}\left(\mathsf{0}, \mathsf{s}^{\Delta\times(n+2)+3}(\mathsf{0}), \mathsf{s}^{\Delta\times(n+2)+3}(\mathsf{0})\right) \\ &(\overset{n+2}{\underset{r_1}{\Rightarrow}} \circ \overset{n+2}{\underset{r_2}{\Rightarrow}}) \mathsf{f}\left(\mathsf{s}^{\Delta\times(n+2)}(\mathsf{0}), \mathsf{s}^3(\mathsf{0}), \mathsf{s}^3(\mathsf{0})\right) \\ &\underset{r_3}{\Rightarrow} \mathsf{f}\left(\mathsf{0}, \mathsf{s}^{\Delta\times(n+4)+3}(\mathsf{0}), \mathsf{s}^{\Delta\times(n+4)+3}(\mathsf{0})\right) \\ &(\overset{n+4}{\underset{r_1}{\Rightarrow}} \circ \overset{n+4}{\underset{r_2}{\Rightarrow}}) \cdots \end{aligned}$$

It has the form $t_0 \rightarrowtail t_1 \rightarrowtail \cdots$ where $\rightarrowtail = (\Rightarrow_{r_1}^* \circ \Rightarrow_{r_2}^* \circ \Rightarrow_{r_3})$ and it does not embed any loop, because the number of applications of r_1 and r_2 gradually

increases. Moreover, it is not a $(\Rightarrow_{w_1}^* \circ \Rightarrow_{w_2})$-chain for some finite sequences of rules w_1 and w_2 (see Cor. 1). We observe that the pair (r_1, r_3) is not recurrent: if it was then, in r_1, the context c_1 would be $\mathsf{f}(\Box_1, \Box_2, z)$ and, in r_3, it would be $\mathsf{f}(\Box_1, \Box_2, \mathsf{s}^3(\mathsf{0}))$ or $\mathsf{f}(\Box_1, \mathsf{s}^3(\mathsf{0}), \Box_2)$, which is not possible because these contexts are different. For the same reasons, the pair (r_2, r_3) is not recurrent either. Hence, our approach does not apply to this example.

Acknowledgments. The author thanks the anonymous reviewers for their insightful and constructive criticisms.

Disclosure of Interests. The author has no competing interests to declare that are relevant to the content of this article.

References

1. Apt, K.R.: From Logic Programming to Prolog. Prentice Hall International series in computer science, Prentice Hall (1997)
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
3. Codish, M., Taboch, C.: A semantic basis for the termination analysis of logic programs. Journal of Logic Programming **41**(1), 103–123 (1999). https://doi.org/10.1016/S0743-1066(99)00006-0
4. Emmes, F., Enger, T., Giesl, J.: Proving non-looping non-termination automatically. In: Gramlich, B., Miller, D., Sattler, U. (eds.) Proc. of the 6th International Joint Conference on Automated Reasoning. LNCS, vol. 7364, pp. 225–240. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_19
5. Endrullis, J., Zantema, H.: Proving non-termination by finite automata. In: Fernández, M. (ed.) Proc. of the 26th International Conference on Rewriting Techniques and Applications. Leibniz International Proceedings in Informatics, vol. 36, pp. 160–176. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015). https://doi.org/10.4230/LIPIcs.RTA.2015.160
6. Giesl, J., *et al.*: Automated Program Verification Environment (2025). http://aprove.informatik.rwth-aachen.de/
7. Oppelt, M.: Automatische Erkennung von Ableitungsmustern in nichtterminierenden Wortersetzungssystemen. Diploma Thesis, HTWK Leipzig, Germany (2008)
8. Payet, E.: Binary non-termination in term rewriting and logic programming. In: Yamada, A. (ed.) Proc. of the 19th International Workshop on Termination (2023). https://doi.org/10.48550/arXiv.2307.11549
9. Payet, E.: Non-termination in term rewriting and logic programming. Journal of Automated Reasoning **68**(4), 24 pages (2024). https://doi.org/10.1007/S10817-023-09693-Z
10. Payet, E.: Non-termination of logic programs using patterns. Accepted for presentation at the 41st International Conference on Logic Programming (2025)
11. Wang, Y., Sakai, M.: On non-looping term rewriting. In: Geser, A., Søndergaard, H. (eds.) Proc. of the 8th International Workshop on Termination. pp. 17–21 (2006)
12. Zantema, H., Geser, A.: Non-looping rewriting. Universiteit Utrecht. UU-CS, Department of Computer Science, Utrecht University, Netherlands (1996)