TPLP: Page 1–17.

doi:10.1017/S1471068425100100

Non-Termination of Logic Programs Using Patterns

ETIENNE PAYET
LIM - Université de la Réunion, France
(*e-mail:* etienne.payet@univ-reunion.fr)

submitted 13 July 2025; revised 13 July 2025; accepted 27 July 2025

Abstract

In this paper, we consider an approach introduced in term rewriting for the automatic detection of non-looping non-termination from *patterns* of rules. We adapt it to logic programing by defining a new unfolding technique that produces patterns describing possibly infinite sets of finite rewrite sequences. We present an experimental evaluation of our contributions that we implemented in our tool NTI (Non-Termination Inference).

KEYWORDS: non-termination, non-loops, unfolding, logic programing

1 Introduction

This paper is concerned with non-termination in logic programing, where one rewrites finite sequences of terms (called *queries*) according to the operational semantics described, *for example*, by Apt (1997). Rewriting is formalized by binary relations $\Rightarrow_r$ indexed by rules r from the logic program under consideration and non-termination by the existence of an infinite rewrite sequence $Q_0 \Rightarrow_{r_1} Q_1 \Rightarrow_{r_2} \cdots$ (where the Q_is are queries). Our motivations are theoretical (study remarkable forms of infinite rewrite sequences) and practical (help programmers to detect bugs by providing queries that run forever).

Most papers related to this topic provide necessary or sufficient conditions for the existence of *loops*, *that is*, finite rewrite sequences $Q_0 \Rightarrow_{r_1} \cdots \Rightarrow_{r_n} Q_n$ where Q_n satisfies a condition $\mathcal{C}$ that entails the possibility of starting again, *that is*, $Q_n \Rightarrow_{r_1} \cdots \Rightarrow_{r_n} Q_{2n}$ holds and Q_{2n} also satisfies $\mathcal{C}$, and so on. For example, Payet and Mesnard (2006) present a sufficient condition, based on *neutral* argument positions of predicate symbols, which is applied to elements of the *binary unfolding* (a set of rules that exhibits the termination properties of logic programs, see the paper by Codish and Taboch (1999)).

In this paper, we are rather interested in *non-looping* non-termination, *that is*, infinite rewrite sequences that do not embed any loop. The non-periodic nature of such sequences makes them difficult to detect, while they can be produced from simple logic programs, as those used in our experiments (Sect. 5). We are inspired by the approach of Emmes *et al.* (2012), introduced in the context of term rewriting.[1] This approach

[1] The operational semantics of term rewriting differs from that of logic programing: a crucial difference is that the rewrite relation of term rewriting is based on instantiation while that of logic programing relies on unification.

considers *pattern terms* ("abstract" terms describing possibly infinite sets of concrete terms) as well as *pattern rules* built from pattern terms. From the term rewrite system under analysis, it produces pattern rules that are *correct*, *that is*, they describe sets of finite rewrite sequences w.r.t. the operational semantics of term rewriting. Nine inference rules are provided to derive correct pattern rules, as well as a strategy for their automated application and a sufficient condition to detect non-looping non-termination.

We adapt this approach to logic programing. Our main contributions are: (i) the definition of a new unfolding technique that produces correct pattern rules w.r.t. the operational semantics of logic programing (this gives a more compact presentation than the nine inference rules of Emmes *et al.* (2012) and we do not need application strategies); (ii) the definition of a restricted form of pattern terms, called *simple*, for which we provide a unification algorithm (needed to compute the unfolding) that we prove correct; (iii) an easily automatable sufficient condition to detect non-looping non-termination from pattern rules built from simple pattern terms; (iv) the implementation of a non-termination approach based on these notions in our tool NTI, that we have evaluated on logic programs resulting from the translation of term rewrite systems used in the experiments of Emmes *et al.* (2012). As far as we know, our approach is the first capable of proving non-termination of these logic programs automatically.

The paper is organized as follows. Sect. 2 introduces basic definitions and notations (a running example illustrating our contributions starts from Sect. 2.5), Sect. 3 presents our adaptation of patterns to logic programing, Sect. 4 considers the notion of simple pattern, Sect. 5 presents an experimental evaluation, Sect. 6 describes related work and Sect. 7 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, *for example*, $\langle a_0, a_1\rangle\,\langle a_2, a_3\rangle = \langle a_0, a_1, a_2, a_3\rangle$. We generally denote elements of $\overline{A}$ using lowercase letters with an overline, *for example*, $\overline{a}$.

2.1 Binary relations

A binary relation ϕ on a set A is a subset of $A^2 = A \times A$. For all $\varphi \subseteq A^2$, the *composition* of ϕ and φ is $\phi \circ \varphi = \big\{(a, a') \in A^2 \;\big|\; \exists a_1 \in A\colon\ (a, a_1) \in \phi \wedge (a_1, a') \in \varphi\big\}$. 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. $\phi^* = \phi^0 \cup \phi^+$) is the transitive (resp. reflexive and transitive) *closure* of ϕ. A *ϕ-chain* (or *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'. For binary relations that have the form of an arrow, *for example*, $\Rightarrow$, 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 Baader and Nipkow (1998) for terms. A *signature* is a set of *function symbols*, each element of which has an *arity* in $\mathbb{N}$ (the 0-ary

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

elements are called *constant symbols*). We denote function symbols by words in the *sans serif* font, *for example*, f, $\mathsf{0}$, $\mathsf{while}\ldots$

Let Σ be a signature and X be a set of *variables* disjoint from Σ. For all $m \in \mathbb{N}$, we let $\Sigma^{(m)}$ denote the set of all m-ary elements of Σ. The set $T(\Sigma, X)$ of all Σ-*terms over* X (or simply *terms* if Σ, X are clear from the context) 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, *for example*, $\mathsf{s}^3(\mathsf{0})$ 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 $T(\Sigma, X)$ is 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 let $Ran(\theta) = \bigcup\{ Var(\theta(x)) \mid x \in Dom(\theta)\}$ and $Var(\theta) = Dom(\theta) \cup Ran(\theta)$. We usually write θ as $\{x_1 \mapsto \theta(x_1), \ldots, x_m \mapsto \theta(x_m)\}$ 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 $s \in T(\Sigma, X)$, denoted as $s\theta$, is defined as: $s\theta = \theta(s)$ if $s \in X$ and $s\theta = \mathsf{f}(s_1\theta, \ldots, s_m\theta)$ if $s = \mathsf{f}(s_1, \ldots, s_m)$. Then, $s\theta$ is called an *instance* of s. Application is extended to finite sequences of terms: $\langle s_1, \ldots, s_m \rangle\, \theta = \langle s_1\theta, \ldots, s_m\theta \rangle$.

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, *that is*, for all $s \in T(\Sigma, X)$, $(s\sigma)\theta = s(\sigma\theta)$. We say that σ *commutes* with θ if $x\sigma\theta = x\theta\sigma$ for all $x \in X$. 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 s and t unify) if $s\sigma = t\sigma$ for some $\sigma \in S(\Sigma, X)$. Then, σ is a *unifier* of s and t. We let $mgu(s, t)$ denote the set of *most general unifiers* of s and t. All this is naturally extended to finite sequences of terms.

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 $T(\Sigma, X)$ and most of the time a substitution is an element of $S(\Sigma, X)$. Let n be a positive integer. An n-*context* is an element of $T(\Sigma \cup H, X)$ that contains occurrences of $\Box_1$, …, $\Box_n$ 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 $c(s_1, \ldots, s_n)$ 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 $n \in \mathbb{N}$, $c^{n+1} = c(c^n)$. We denote by $\chi^{(1)}$ the set of 1-contexts that contain no variable. Terms are generally denoted by a, s, t, u, v, variables by x, y, z and contexts by c, possibly with subscripts and primes.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

2.4 Logic programing

We refer to Apt (1997) for the basics of logic programing. To simplify our presentation, we place ourselves in the general framework of *term reduction systems*, *that is*, we do not distinguish *predicate/function* symbols, *terms/atoms*... and we do not always use the standard terminology and notations of logic programing (*e.g.*, *rule* instead of *clause*).

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*. For all sets of rules U, we let $[U] = \bigcup_{r \in U} [r]$. Moreover, for all rules or 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 finite sequences of terms. This is formalized by the following binary relation, which corresponds to the operational semantics of logic programing with the leftmost selection rule.

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

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

For all $s \in T(\Sigma, X)$, $calls_P(s) = \{t \in T(\Sigma, X) \mid \langle s \rangle \Rightarrow_P^+ \langle t, \ldots \rangle\} \cup \{\mathsf{e} \mid \langle s \rangle \Rightarrow_P^+ \mathsf{e}\}$ *is the set of calls in the* $\Rightarrow_P$*-chains that start from* s.

2.5 Binary unfolding

The binary unfolding of a program P (see the paper by Codish and Taboch (1999)) is a set of binary rules, denoted as $binunf(P)$, that captures call patterns of P. It corresponds to the transitive closure of a binary relation which relates consecutive calls selected in a computation (Prop. 1 below). Non-termination for a specific sequence of terms implies the existence of a corresponding infinite chain in this relation (Thm. 1 below).

More precisely, $binunf(P)$ is defined as the least fixed point of a function T_P^β on the power set of $\Re$. For all $U \subseteq \Re$, $T_P^\beta(U)$ is constructed by unfolding prefixes of right-hand sides of rules from P using U. Let $(u, \langle v_1, \ldots, v_m \rangle) \in P$:

(i) for each $1 \leq i \leq m$, one unfolds $v_1, \ldots, v_{i-1}$ with $(u_1, \mathsf{e}), \ldots, (u_{i-1}, \mathsf{e})$ from U to obtain a corresponding instance of (u, v_i),
(ii) for each $1 \leq i \leq m$, one unfolds $v_1, \ldots, v_{i-1}$ with $(u_1, \mathsf{e}), \ldots, (u_{i-1}, \mathsf{e})$ from U and v_i with (u_i, v) from U to obtain a corresponding instance of (u, v),
(iii) one unfolds $v_1, \ldots, v_m$ with $(u_1, \mathsf{e}), \ldots, (u_m, \mathsf{e})$ from U to obtain a corresponding instance of (u, e).

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

This is formally expressed as follows, using the set *id* of *identity binary rules*, which consists of every pair $(\mathsf{f}(x_1,\ldots,x_m),\mathsf{f}(x_1,\ldots,x_m))$ where $\mathsf{f}\in\Sigma^{(m)}$ and $x_1,\ldots,x_m$ are distinct variables. The use of *id* allows one to cover case (i) above.

Definition 3.
For all programs P and all $U\subseteq\Re$, we let

$$T_P^\beta(U)=[(u,\mathsf{e})\in P]\cup\left[(u\theta,v\theta)\;\middle|\;\begin{array}{l} r=(u,\langle v_1,\ldots,v_m\rangle)\in P,\ 1\le i\le m\\ \langle(u_1,\mathsf{e}),\ldots,(u_{i-1},\mathsf{e}),(u_i,v)\rangle\ll_r U\cup id\\ \text{if } i<m \text{ then } v\neq\mathsf{e}\\ \theta\in mgu\,(\,\langle u_1,\ldots,u_i\rangle\,,\langle v_1,\ldots,v_i\rangle\,)\end{array}\right]$$

The binary unfolding of P is the set of binary rules $binunf(P)=(T_P^\beta)^*(\emptyset)$.

Intuitively, each $(u,v)\in binunf(P)$ specifies that some instance of v belongs to $calls_P(u)$. More generally, we have:

Proposition 1.
Let P be a program, $(u,v)\in binunf(P)$ and $\sigma\in S(\Sigma,X)$. Then, for some $\theta\in S(\Sigma,X)$, we have $v\theta\in calls_P(u\sigma)$.

Example 1.
Let P be the program which consists of the rules

$r_1=(\mathsf{while}(x,y),\langle\mathsf{gt}(x,y),\mathsf{add}(x,y,z),\mathsf{while}(z,\mathsf{s}(y))\rangle)$
$r_2=(\mathsf{gt}(\mathsf{s}(x),0),\mathsf{e})$ $\quad$ $r_3=(\mathsf{gt}(\mathsf{s}(x),\mathsf{s}(y)),\mathsf{gt}(x,y))$
$r_4=(\mathsf{add}(x,0,x),\mathsf{e})$ $\quad$ $r_5=(\mathsf{add}(x,\mathsf{s}(y),\mathsf{s}(z)),\mathsf{add}(x,y,z))$
$r_6=(\mathsf{while}(x,y),\mathsf{le}(x,y))$
$r_7=(\mathsf{le}(0,x),\mathsf{e})$ $\quad$ $r_8=(\mathsf{le}(\mathsf{s}(x),\mathsf{s}(y)),\mathsf{le}(x,y))$

and which corresponds to the imperative program fragment

```
while (x > y) { x = x + y; y = y + 1; }
```

Rule r_1 is used to continue the loop and r_6 is used to stop it. Note that this imperative fragment does not terminate if it is run from integers x,y such that $x>y>0$.

Let us compute some elements of $binunf(P)$ by applying Def. 3.

- *Obviously, we have $[r_2]\cup[r_4]\subseteq T_P^\beta(\emptyset)$.*
- *Let us unfold the whole right-hand side of r_1, that is let us consider $i=m=3$. We have*

$$\langle\,(\mathsf{gt}(\mathsf{s}(x_1),0),\mathsf{e}),(\mathsf{add}(x_2,0,x_2),\mathsf{e})\,,\\ (\mathsf{while}(x_3,y_3),\mathsf{while}(x_3,y_3))\,\rangle\ll_{r_1}[r_2]\cup[r_4]\cup id$$

  and $\theta=\{x\mapsto\mathsf{s}(x_1),y\mapsto 0,z\mapsto\mathsf{s}(x_1),x_2\mapsto\mathsf{s}(x_1),x_3\mapsto\mathsf{s}(x_1),y_3\mapsto\mathsf{s}(0)\}$ is the mgu of $\langle\mathsf{gt}(\mathsf{s}(x_1),0),\mathsf{add}(x_2,0,x_2),\mathsf{while}(x_3,y_3)\rangle$ and $\langle\mathsf{gt}(x,y),\mathsf{add}(x,y,z),\mathsf{while}(z,\mathsf{s}(y))\rangle$. Consequently, the set $(T_P^\beta)^2(\emptyset)$ contains the binary rule $(\mathsf{while}(x,y)\theta,\mathsf{while}(x_3,y_3)\theta)=(\mathsf{while}\,(\mathsf{s}(x_1),0)\,,\mathsf{while}\,(\mathsf{s}(x_1),\mathsf{s}(0)))$.
- *More generally, we have $[r'_n\mid n\in\mathbb{N}]\subseteq binunf(P)$ where, for all $n\in\mathbb{N}$, $r'_n=\big(\mathsf{while}(\mathsf{s}^{n+1}(x),\mathsf{s}^n(0)),\mathsf{while}(\mathsf{s}^{2n+1}(x),\mathsf{s}^{n+1}(0))\big)$.*

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

The binary unfolding exhibits the termination properties of a program:

Theorem 1
(see the paper by Codish and Taboch (1999)). *Let P be a program and $\overline{s}$ be a sequence of terms. Then, there is an infinite $\Rightarrow_P$-chain that starts from $\overline{s}$ if and only if there is an infinite $\Rightarrow_{binunf(P)}$-chain that starts from $\overline{s}$.*

Example 2
(Ex. 1 cont.). *For all $n > m > 0$, we have the infinite $\Rightarrow_P$-chain*

$$\langle \mathsf{while}(\mathsf{s}^n(0), \mathsf{s}^m(0)) \rangle \, (\underset{r_1}{\Rightarrow} \circ \overset{m}{\underset{r_3}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow} \circ \overset{m}{\underset{r_5}{\Rightarrow}} \circ \underset{r_4}{\Rightarrow}) \, \langle \mathsf{while}(\mathsf{s}^{n+m}(0), \mathsf{s}^{m+1}(0)) \rangle$$
$$(\underset{r_1}{\Rightarrow} \circ \overset{m+1}{\underset{r_3}{\Rightarrow}} \circ \underset{r_2}{\Rightarrow} \circ \overset{m+1}{\underset{r_5}{\Rightarrow}} \circ \underset{r_4}{\Rightarrow}) \cdots$$

and also the infinite $\Rightarrow_{binunf(P)}$-chain

$$\langle \mathsf{while}(\mathsf{s}^n(0), \mathsf{s}^m(0)) \rangle \underset{r'_m}{\Rightarrow} \langle \mathsf{while}(\mathsf{s}^{n+m}(0), \mathsf{s}^{m+1}(0)) \rangle \underset{r'_{m+1}}{\Rightarrow} \cdots$$

We note that none of these chains embeds a loop: in the $\Rightarrow_P$-chain, the number of applications of r_3 and r_5 gradually increases and, in the $\Rightarrow_{binunf(P)}$-chain, a new binary rule (not occurring before) is used at each step.

3 Patterns

In this section, we describe our adaptation to logic programing of the pattern approach introduced by Emmes *et al.* (2012). Our main idea is similar to that of Payet and Mesnard (2006), *that is*, unfold the program and try to prove its non-termination from the resulting set. To this end, based on the binary unfolding mentioned previously (Def. 3), we introduce a new unfolding technique that produces patterns of rules (Def. 10) and a sufficient condition to non-termination that we apply to the generated patterns (Thm. 3).

First, we recall the definition of pattern term and pattern rule, that we formulate differently from Emmes *et al.* (2012) to fit our needs. In particular, we introduce the concept of pattern substitution.

Definition 4.
A pattern substitution is a pair $\theta = (\sigma, \mu) \in S(\Sigma, X)^2$, rather denoted as $\sigma \star \mu$. For all $n \in \mathbb{N}$, we let $\theta(n) = \sigma^n \mu$. We say that θ describes the set $\{\theta(n) \mid n \in \mathbb{N}\} \subseteq S(\Sigma, X)$.

For instance, if $\sigma = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\}$ and $\mu = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{0}\}$ then $\theta = \sigma \star \mu$ is a pattern substitution. For all $n \in \mathbb{N}$, we have $\theta(n) = \sigma^n \mu = \{x \mapsto \mathsf{s}^{n+1}(x), y \mapsto \mathsf{s}^n(\mathsf{0})\}$.

From pattern substitutions, we define pattern terms.

Definition 5.
A pattern term is a pair $p = (s, \theta)$ where $s \in T(\Sigma, X)$ and θ is a pattern substitution. We denote it as $s \star \theta$ or $s \star \sigma \star \mu$ if $\theta = \sigma \star \mu$. For all $n \in \mathbb{N}$, we let $p(n) = s\theta(n)$. We say that p describes the set $\{p(n) \mid n \in \mathbb{N}\} \subseteq T(\Sigma, X)$. For all $s \in T(\Sigma, X)$, we let $s^\star = s \star \emptyset \star \emptyset$.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

For instance, $p = \mathsf{gt}(x, y) \star \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\} \star \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{0}\}$ is a pattern term. For all $n \in \mathbb{N}$, we have $p(n) = \mathsf{gt}(\mathsf{s}^{n+1}(x), \mathsf{s}^n(\mathsf{0}))$.

Then, from pattern terms one can define pattern rules.

Definition 6.
A pattern rule is a pair $r = (p, q)$ of pattern terms. It describes the set of binary rules $rules(r) = \{(p(n), q(n)) \mid n \in \mathbb{N}\}$. We let $\Im$ denote the set of pattern rules.

Example 3
(Ex. 1 cont.). *Let $u = \mathsf{while}(x, y)$ be the left-hand side of r_1, $\sigma = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\}$, $\sigma' = \{x \mapsto \mathsf{s}(x)\}$ and $\mu = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{0}\}$. The pattern terms*

$$p = u\sigma \star \sigma \star \mu = \mathsf{while}(\mathsf{s}(x), \mathsf{s}(y)) \star \sigma \star \mu$$
$$q = u\sigma^2 \star \sigma\sigma' \star \mu = \mathsf{while}(\mathsf{s}^2(x), \mathsf{s}^2(y)) \star \{x \mapsto \mathsf{s}^2(x), y \mapsto \mathsf{s}(y)\} \star \mu$$

respectively describe the sets of terms $\{p(n) = \mathsf{while}\left(\mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(\mathsf{0})\right) \mid n \in \mathbb{N}\}$ and $\{q(n) = \mathsf{while}\left(\mathsf{s}^{2n+3}(x), \mathsf{s}^{n+2}(\mathsf{0})\right) \mid n \in \mathbb{N}\}$. Moreover,

$$\begin{aligned} rules((p, q)) &= \{\left(\mathsf{while}(\mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(\mathsf{0})), \mathsf{while}(\mathsf{s}^{2n+3}(x), \mathsf{s}^{n+2}(\mathsf{0}))\right) \mid n \in \mathbb{N}\} \\ &= \{r'_n \mid n > 0\} \subseteq \{r'_n \mid n \in \mathbb{N}\} \subseteq binunf(P) \text{ (see Ex. 1)} \end{aligned}$$

The notion of *correctness* of a pattern rule is defined by Emmes et al. (2012) in the context of term rewriting. We reformulate it as follows in logic programing.

Definition 7.
Let P be a program. A pattern rule r is correct w.r.t. P if $rules(r) \subseteq binunf(P)$. A set U of pattern rules is correct w.r.t. P if all its elements are.

So, if a pattern rule (p, q) is correct w.r.t. P then, for all $n \in \mathbb{N}$, we have $(p(n), q(n)) \in binunf(P)$, *that is*, by Prop. 1, $\langle p(n)\rangle \Rightarrow_P^+ \langle q(n)\theta, \ldots\rangle$ for some $\theta \in S(\Sigma, X)$. Intuitively, this means that for all $n \in \mathbb{N}$, a call to $p(n)$ necessarily leads to a call to $q(n)$. For instance, in Ex. 3, we have $rules((p, q)) \subseteq binunf(P)$, hence (p, q) is correct w.r.t. P and we have $\left\langle \mathsf{while}(\mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(\mathsf{0}))\right\rangle \Rightarrow_P^+ \left\langle \mathsf{while}(\mathsf{s}^{2n+3}(x), \mathsf{s}^{n+2}(\mathsf{0}))\right\rangle$ for all $n \in \mathbb{N}$.

The next result allows one to infer correct pattern rules from a program. It considers pairs of rules that have the same form as (r_2, r_3), (r_4, r_5) and (r_7, r_8) in Ex. 1. It uses the set of contexts $\chi^{(1)}$ (see Sect. 2.3).

Proposition 2.
Suppose that a program P contains two binary rules $r = (u, v)$ and $r' = (u', \mathsf{e})$ such that

- *$u = c(c_1(x_1), \ldots, c_m(x_m))$, $v = c(x_1, \ldots, x_m)$ and $u' = c(t_1, \ldots, t_m)$,*
- *$\{c_1, \ldots, c_m\} \subseteq \chi^{(1)}$ and c is an m-context with $Var(c) = \emptyset$,*
- *$x_1, \ldots, x_m$ are distinct variables and $t_1, \ldots, t_m$ are terms.*

Then, $(p, \mathsf{e}^\star)$ and $(q, v^\star)$ are correct w.r.t. P where $p = v \star \sigma \star \mu$, $q = u \star \sigma \star \emptyset$ and

$$\sigma = \{x_k \mapsto c_k(x_k) \mid 1 \le k \le m,\ c_k(x_k) \neq x_k\}$$
$$\mu = \{x_k \mapsto t_k \mid 1 \le k \le m,\ t_k \neq x_k\}$$

Example 4.
Let us regard (r_2, r_3) and (r_4, r_5) from Ex. 1.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

- $r_2 = (c(t_1, t_2), \mathsf{e})$ *and* $r_3 = (c(c_1(x), c_1(y)), c(x, y))$ *for* $t_1 = \mathsf{s}(x)$, $t_2 = 0$, $c = \mathsf{gt}(\Box_1, \Box_2)$ *and* $c_1 = \mathsf{s}(\Box_1)$. *So by Prop. 2,* $(p_1, \mathsf{e}^\star)$ *and* $(q_1, \mathsf{gt}(x, y)^\star)$ *are correct w.r.t.* P *where*

$$p_1 = \mathsf{gt}(x, y) \star \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\} \star \{x \mapsto \mathsf{s}(x), y \mapsto 0\}$$
$$q_1 = \mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) \star \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\} \star \emptyset$$

  Let $n \in \mathbb{N}$. *Then, we have* $(p_1(n), \mathsf{e}^\star(n)) \in binunf(P)$. *Hence, by Prop. 1,* $\mathsf{e}^\star(n)\eta \in calls_P(p_1(n))$ *for some* $\eta \in S(\Sigma, X)$. *But* $\mathsf{e}^\star(n)\eta = \mathsf{e}\eta = \mathsf{e}$ *so by Def. 2 (*$calls_P$*) we have* $\langle p_1(n) \rangle \Rightarrow_P^+ \mathsf{e}$ *where* $p_1(n) = \mathsf{gt}(\mathsf{s}^{n+1}(x), \mathsf{s}^n(0))$.
- $r_4 = (c'(t'_1, t'_2, t'_1), \mathsf{e})$ *and* $r_5 = (c'(c'_1(x), c'_2(y), c'_2(z)), c'(x, y, z))$ *for* $t'_1 = x$, $t'_2 = 0$, $c' = \mathsf{add}(\Box_1, \Box_2, \Box_3)$, $c'_1 = \Box_1$ *and* $c'_2 = \mathsf{s}(\Box_1)$. *So, by Prop. 2, the pattern rule* $(p_2, \mathsf{e}^\star)$ *is correct w.r.t.* P *where* $p_2 = \mathsf{add}(x, y, z) \star \{y \mapsto \mathsf{s}(y), z \mapsto \mathsf{s}(z)\} \star \{y \mapsto 0, z \mapsto x\}$.

Unification for pattern terms is not considered by Emmes *et al.* (2012). As we need it in our development (see Def. 10 below), we define it here.

Definition 8.
Let p *and* q *be pattern terms and* θ *be a pattern substitution. Then,* θ *is a unifier of* p *and* q *if, for all* $n \in \mathbb{N}$, *we have* $p(n)\theta(n) = q(n)\theta(n)$. *Moreover,* θ *is a most general unifier (mgu) of* p *and* q *if, for all* $n \in \mathbb{N}$, $\theta(n) \in mgu(p(n), q(n))$. *We let* $mgu(p, q)$ *denote the set of all mgu's of* p *and* q.

for example, if $p = \mathsf{f}(x, y) \star \{x \mapsto \mathsf{s}(x)\} \star \{x \mapsto 0\}$ and $q = \mathsf{f}(x, y) \star \{y \mapsto \mathsf{s}(y)\} \star \{y \mapsto 1\}$, then $\theta = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\} \star \{x \mapsto 0, y \mapsto 1\}$ is a unifier of p and q. Indeed, for all $n \in \mathbb{N}$, $\theta(n) = \{x \mapsto \mathsf{s}^n(0), y \mapsto \mathsf{s}^n(1)\}$ is a unifier of $p(n) = \mathsf{f}(\mathsf{s}^n(0), y)$ and $p(n) = \mathsf{f}(x, \mathsf{s}^n(1))$. All these notions are naturally extended to finite sequences of pattern terms.

We also need to adapt the notion of equivalence class modulo renaming (see Sect. 2.4).

Definition 9.
For all $U \subseteq \Im$, *we let* $[U] = \bigcup_{r \in U} [r]$ *where* $[r] = \{r' \in \Im \mid rules(r') \subseteq [rules(r)]\}$.

Hence, $[r]$ consists of all pattern rules r' that describe a subset of $[rules(r)]$. For instance, if $p = \mathsf{f}(x, y) \star \{x \mapsto \mathsf{s}(x)\} \star \{x \mapsto 0\}$ and $p' = \mathsf{f}(\mathsf{s}(x), y') \star \{x \mapsto \mathsf{s}(x)\} \star \{x \mapsto 0\}$, then $(p', \mathsf{e}^\star) \in [(p, \mathsf{e}^\star)]$. Indeed, we have $rules((p', \mathsf{e}^\star)) = \{(\mathsf{f}(\mathsf{s}^{n+1}(0), y'), \mathsf{e}) \mid n \in \mathbb{N}\} \subseteq [(\mathsf{f}(\mathsf{s}^n(0), y), \mathsf{e}) \mid n \in \mathbb{N}] = [rules((p, \mathsf{e}^\star))]$.

Now we provide a counterpart of Def. 3 (binary unfolding) for pattern rules. A notable difference, however, is the use of an arbitrary set B instead of $E = \{(u^\star, \mathsf{e}^\star) \mid (u, \mathsf{e}) \in P\}$. The set B plays a similar role to the pattern creation inference rules of Emmes *et al.* (2012). Using suitable sets B's (as those consisting of rules provided by Prop. 2), we get an approach that computes pattern rules finitely describing infinite subsets of $binunf(P)$, *that is*, an approach that unfolds "faster" (see Ex. 5). We let *patid* denote the set of all pairs $(\mathsf{f}(x_1, \ldots, x_m)^\star, \mathsf{f}(x_1, \ldots, x_m)^\star)$ where $\mathsf{f} \in \Sigma^{(m)}$ and $x_1, \ldots, x_m$ are distinct variables. For all rules r, the notation $\ll_r$ is naturally extended to sets of pattern rules, according to the following definitions: the set of variables of a pattern term $p = s \star \sigma \star \mu$ is $Var(p) = Var(s) \cup Var(\sigma) \cup Var(\mu)$ and that of a pattern rule $r = (p, q)$ is $Var(r) = Var(p) \cup Var(q)$.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

Definition 10.
For all programs P and all $B, U \subseteq \Im$, we let

$$T^{\pi}_{P,B}(U) = [B] \cup \left[\begin{array}{l|l} (u \star \sigma \star \mu, & r = (u, \langle v_1, \ldots, v_m \rangle) \in P,\ 1 \le i \le m \\ \ v \star \sigma_i \sigma \star \mu_i \mu) & \langle (p_1, \mathsf{e}^\star), \ldots, (p_{i-1}, \mathsf{e}^\star), (p_i, v \star \sigma_i \star \mu_i) \rangle \\ & \qquad \ll_r U \cup patid \\ & \text{if } i < m \text{ then } v \neq \mathsf{e} \\ & \sigma \star \mu \in mgu\left(\langle p_1, \ldots, p_i \rangle, \langle {v_1}^\star, \ldots, {v_i}^\star \rangle\right) \\ & \sigma \text{ commutes with } \sigma_i \text{ and } \mu_i \end{array} \right]$$

The pattern unfolding of P using B is the set $patunf(P, B) = (T^{\pi}_{P,B})^*(\emptyset)$.

Example 5.
Let B be the set consisting of the rules $(p_1, \mathsf{e}^\star)$ and $(p_2, \mathsf{e}^\star)$ of Ex. 4. Let us compute some elements of $patunf(P, B)$ by applying Def. 10. We have $\{(p'_1, \mathsf{e}^\star), (p'_2, \mathsf{e}^\star)\} \subseteq [B] \subseteq T^{\pi}_{P,B}(\emptyset)$ where p'_1 and p'_2 are renamed versions of p_1 and p_2 respectively:

$$p'_1 = \mathsf{gt}(x_1, y_1) \star \{x_1 \mapsto \mathsf{s}(x_1), y_1 \mapsto \mathsf{s}(y_1)\} \star \{x_1 \mapsto \mathsf{s}(x_1), y_1 \mapsto \mathsf{0}\}$$
$$p'_2 = \mathsf{add}(x_2, y_2, z_2) \star \{y_2 \mapsto \mathsf{s}(y_2), z_2 \mapsto \mathsf{s}(z_2)\} \star \{y_2 \mapsto \mathsf{0}, z_2 \mapsto x_2\}$$

Let us unfold the whole right-hand side of $r_1 \in P$, i.e., let us consider $i = m = 3$. We have $\langle (p'_1, \mathsf{e}^\star), (p'_2, \mathsf{e}^\star), (p'_3, p'_3) \rangle \ll_{r_1} T^{\pi}_{P,B}(\emptyset) \cup patid$ where $p'_3 = \mathsf{while}(x_3, y_3)^\star$. The right-hand side of r_1 is $\langle v_1, v_2, v_3 \rangle = \langle \mathsf{gt}(x, y), \mathsf{add}(x, y, z), \mathsf{while}(z, \mathsf{s}(y)) \rangle$. Let $S = \langle {v_1}^\star, {v_2}^\star, {v_3}^\star \rangle$ and $S' = \langle p'_1, p'_2, p'_3 \rangle$. We show in Ex. 11 that $\rho \star \nu \in mgu(S, S')$ where

$$\rho = \{x \mapsto \mathsf{s}(x),\ y \mapsto \mathsf{s}(y),\ z \mapsto \mathsf{s}^2(z),\ x_2 \mapsto \mathsf{s}(x_2),\ x_3 \mapsto \mathsf{s}^2(x_3),\ y_3 \mapsto \mathsf{s}(y_3)\}$$
$$\nu = \{x \mapsto \mathsf{s}(x_1),\ y \mapsto \mathsf{0},\ z \mapsto \mathsf{s}(x_1),\ x_2 \mapsto \mathsf{s}(x_1),\ x_3 \mapsto \mathsf{s}(x_1),\ y_3 \mapsto \mathsf{s}(\mathsf{0})\}$$

So, $r'' = (u \star \rho \star \nu, \mathsf{while}(x_3, y_3) \star \rho \star \nu) \in (T^{\pi}_{P,B})^2(\emptyset)$ where $u = \mathsf{while}(x, y)$ is the left-hand side of r_1. It describes the set of binary rules

$$\left\{ r''_n = \left(\mathsf{while}(\mathsf{s}^{n+1}(x_1), \mathsf{s}^n(\mathsf{0})), \mathsf{while}(\mathsf{s}^{2n+1}(x_1), \mathsf{s}^{n+1}(\mathsf{0}))\right) \mid n \in \mathbb{N} \right\}$$

and we have $[r''_n \mid n \in \mathbb{N}] = [r'_n \mid n \in \mathbb{N}]$ (see Ex. 1).

The following result corresponds to the Soundness Thm. 7 of Emmes *et al.* (2012).

Theorem 2.
Let P be a program and $B \subseteq \Im$ be correct w.r.t. P. Then, $patunf(P, B)$ is correct w.r.t. P.

Finally, we adapt the non-termination criterion of Emmes *et al.* (2012) to our setting.

Theorem 3.
Let P be a program and $B \subseteq \Im$ be correct w.r.t. P. Suppose that $patunf(P, B)$ contains a pattern rule of the form $(u \star \sigma \star \mu, u\sigma^a \star \sigma^b \sigma' \star \mu\mu')$ where σ' commutes with σ and μ. Then, for all $n \in \mathbb{N}$ and all $\theta \in S(\Sigma, X)$, there is an infinite $\Rightarrow_P$-chain that starts from $\langle u\sigma^n \mu\theta \rangle$.

Example 6.
Let us regard the pattern rule (p, q) of Ex. 3. As $rules((p, q)) \subseteq \{r'_n \mid n \in \mathbb{N}\}$ with $\{r'_n \mid n \in \mathbb{N}\} \subseteq [r'_n \mid n \in \mathbb{N}] = [r''_n \mid n \in \mathbb{N}] = [rules(r'')]$ (see Ex. 5), we have $(p, q) \in [r''] \subseteq patunf(P, B)$. Moreover, B is correct w.r.t. P (see Ex. 4) and $(p, q) =$

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

$(u\sigma \star \sigma \star \mu, (u\sigma)\sigma \star \sigma\sigma' \star \mu)$ *(see Ex. 3) where* σ' *commutes with* σ *and* μ*. By Thm. 3, for all* $m, n \in \mathbb{N}$ *and* $\theta = \{x \mapsto \mathsf{s}^n(\mathsf{0})\}$*, the sequence* $\langle (u\sigma)\sigma^m\mu\theta \rangle$ *starts an infinite* $\Rightarrow_P$*-chain, with*

$$\langle (u\sigma)\sigma^m\mu\theta \rangle = \left\langle u\sigma^{m+1}\mu\theta \right\rangle = \left\langle \mathsf{while}\left(\mathsf{s}^{(n+1)+(m+1)}(\mathsf{0}), \mathsf{s}^{m+1}(\mathsf{0})\right) \right\rangle$$

Hence, for all $n > m > 0$*, the sequence* $\langle \mathsf{while}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0})) \rangle$ *starts an infinite* $\Rightarrow_P$*-chain. This had already been observed in Ex. 2.*

4 Simple patterns

In practice, to implement the approach presented in the previous section, one has to find a way to compute mgu's of pattern terms and to check the non-termination condition of Thm. 3. In this section, we introduce a class of pattern terms of a special form, called *simple*, that is more restrictive but for which we provide a unification algorithm as well as a non-termination criterion that is easier to check than that of Thm. 3. We describe them using a new signature that consists of unary symbols only:

$$\Upsilon = \left\{ c^{a,b} : \text{a unary symbol} \;\middle|\; c \in \chi^{(1)}, (a,b) \in \mathbb{N}^2 \right\}$$

Any symbol $c^{a,b} \in \Upsilon$ represents all successive embeddings of c into itself of the form $c^{a\times n+b}$ where $n \in \mathbb{N}$. Hence, for all $u \in T(\Sigma \cup \Upsilon, X)$ and all $n \in \mathbb{N}$, we let $u(n)$ be the element of $T(\Sigma, X)$ obtained from u by replacing every $c^{a,b} \in \Upsilon$ by $c^{a\times n+b}$. Moreover, for all $\theta \in S(\Sigma \cup \Upsilon, X)$ and all $n \in \mathbb{N}$, we let $\theta(n)$ be the element of $S(\Sigma, X)$ defined as: for all $x \in X$, $(\theta(n))(x) = (\theta(x))(n)$. In the rest of this section, we consider elements of $T(\Sigma \cup \Upsilon, X)$ modulo the following equivalence relation.

Definition 11.
The binary relation $\sim \subseteq T(\Sigma \cup \Upsilon, X)^2$ *is defined as:* $u \sim v$ *iff* $u(n) = v(n)$ *for all* $n \in \mathbb{N}$*. It is an equivalence relation and we let* $[u]$ *denote the equivalence class of* u *w.r.t.* $\sim$*.*

The following straightforward result can be used to simplify $(\Sigma \cup \Upsilon)$-terms.

Lemma 1.
For all $c \in \chi^{(1)}$, $a, b, a', b' \in \mathbb{N}$ *and* $u \in T(\Sigma \cup \Upsilon, X)$ *we have* $c^{a,b}(c^{a',b'}(u)) \sim c^{a+a',b+b'}(u)$ *and* $c(u) \sim c^{0,1}(u)$*.*

Example 7.
Let $c = \mathsf{f}(\square_1, \mathsf{0}, \square_1) \in \chi^{(1)}$*. Then,* $c^{1,1} \in \Upsilon$*. Let* $u = c^{1,1}(\mathsf{1}) \in T(\Sigma \cup \Upsilon, X)$*. For all* $n \in \mathbb{N}$*, we have* $u(n) = c^{n+1}(\mathsf{1})$*. For instance,* $u(1) = c^2(\mathsf{1}) = \mathsf{f}(\mathsf{f}(\mathsf{1}, \mathsf{0}, \mathsf{1}), \mathsf{0}, \mathsf{f}(\mathsf{1}, \mathsf{0}, \mathsf{1}))$*. We note that* $v = c^{1,0}(c(\mathsf{1})) \sim c^{1,0}(c^{0,1}(\mathsf{1})) \sim u$*; indeed, for all* $n \in \mathbb{N}$*,* $v(n) = c^n(c(\mathsf{1})) = u(n)$*. Let* $\theta = \{x \mapsto u\} \in S(\Sigma \cup \Upsilon, X)$ *and* $n \in \mathbb{N}$*. We have* $(\theta(n))(x) = (\theta(x))(n) = u(n)$ *and, for all* $y \in X \setminus \{x\}$*,* $(\theta(n))(y) = (\theta(y))(n) = y(n) = y$*. Hence,* $\theta(n) = \{x \mapsto u(n)\}$*.*

Definition 12.
A pattern term $p = s \star \sigma \star \mu$ *is called simple if, for all* $x \in Var(s)$, $\sigma(x) = c^a(x)$ *and* $\mu(x) = c^b(t)$ *for some* $c \in \chi^{(1)}$, $a, b \in \mathbb{N}$ *and* $t \in T(\Sigma, X)$*. Then, we let* $v(p) = [s\theta_p]$ *where*

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

$$\theta_p = \left\{ x \mapsto u \;\middle|\; \begin{array}{l} x \in Var(s),\ \sigma(x) = c^a(x),\ \mu(x) = c^b(t) \\ \text{if } \sigma(x) = x \text{ then } u = \mu(x) \text{ else } u = c^{a,b}(t) \end{array} \right\}$$

A pattern rule (p, q) is called simple if p and q are simple.

The next result follows from Def. 5 and Def. 12.

Lemma 2.
For all simple pattern terms p, all $u \in \upsilon(p)$ and all $n \in \mathbb{N}$ we have $p(n) = u(n)$.

Example 8.
The pattern term $p = s \star \sigma \star \mu = \mathsf{f}(\mathsf{s}(x), y) \star \{x \mapsto \mathsf{s}^2(x)\} \star \{x \mapsto \mathsf{s}(x_1), y \mapsto \mathsf{0}\}$ is simple. For $c = \mathsf{s}(\square_1)$, we have $\langle \sigma(x), \mu(x) \rangle = \langle c^2(x), c(x_1) \rangle$ and $\sigma(y) = y$. So $\upsilon(p) = [s\theta_p]$ where $\theta_p = \{x \mapsto c^{2,1}(x_1), y \mapsto \mathsf{0}\}$. Moreover, $s\theta_p = \mathsf{f}(\mathsf{s}(c^{2,1}(x_1)), \mathsf{0}) = \mathsf{f}(c(c^{2,1}(x_1)), \mathsf{0}) \sim \mathsf{f}(c^{2,2}(x_1), \mathsf{0})$. For all $n \in \mathbb{N}$, $p(n) = s\sigma^n\mu = \mathsf{f}(\mathsf{s}^{2n+2}(x_1), \mathsf{0}) = (s\theta_p)(n)$.

We note that the pattern rules $(p, \mathsf{e}^\star)$ and $(q, v^\star)$ produced from Prop. 2 are simple. In Ex. 4, $\upsilon(p_1) = [\mathsf{gt}(c_1^{1,1}(x), c_1^{1,0}(\mathsf{0}))]$ and $\upsilon(p_2) = [\mathsf{add}(x, c_1^{1,0}(\mathsf{0}), c_1^{1,0}(x))]$ where $c_1 = \mathsf{s}(\square_1)$.

Example 9.
We illustrate the fact that non-termination detection with simple pattern terms is more restrictive than with the full class of pattern terms. Let P be the program consisting of

$$r_1 = (\mathsf{while}(x, y), \langle \mathsf{isList}(y), \mathsf{while}(x, \mathsf{cons}(x, y)) \rangle)$$
$$r_2 = (\mathsf{isList}(\mathsf{nil}), \mathsf{e})$$
$$r_3 = (\mathsf{isList}(\mathsf{cons}(x, y)), \mathsf{isList}(y))$$

Let $c = \mathsf{cons}(x, \square_1)$. Then, $\langle \mathsf{while}(x, c^n(\mathsf{nil})) \rangle\ (\Rightarrow_{r_1} \circ \Rightarrow_{r_3}^n \circ \Rightarrow_{r_2}) \langle \mathsf{while}(x, c^{n+1}(\mathsf{nil})) \rangle$ holds for all $n \in \mathbb{N}$. Hence, for all $n \in \mathbb{N}$, $\langle \mathsf{while}(x, c^n(\mathsf{nil})) \rangle$ starts an infinite $\Rightarrow_P$-chain. To detect that, one would need an unfolded pattern rule of the form $(\mathsf{while}(x, y) \star \sigma \star \mu, \mathsf{while}(x, y)\sigma \star \sigma \star \mu)$ where $\sigma = \{y \mapsto c(y)\}$ and $\mu = \{y \mapsto \mathsf{nil}\}$. Such a rule satisfies the condition of Thm. 3 but is not simple because $c \notin \chi^{(1)}$ ($Var(c) \neq \emptyset$).

We also define simple substitutions.

Definition 13.
A substitution $\theta \in S(\Sigma \cup \Upsilon, X)$ is called simple if, for all $x \in X$, $\theta(x) \in [c^{a,b}(t)]$ for some $c \in \chi^{(1)}$, $a, b \in \mathbb{N}$ and $t \in T(\Sigma, X)$. Then, we let $\upsilon^{-1}(\theta)$ denote the pattern substitution $\sigma \star \mu$ such that: for all $x \in X$, if $\theta(x) \in [c^{a,b}(t)]$ then $\sigma(x) = c^a(x)$ and $\mu(x) = c^b(t)$.

The next result follows from Def. 4 and Def. 13.

Lemma 3.
For all simple substitutions θ and all $n \in \mathbb{N}$, $\theta(n) = (\upsilon^{-1}(\theta))(n)$.

Example 10.
Let $\theta = \{x \mapsto c^2(1), y \mapsto c(c^{2,1}(c^{1,2}(c(\mathsf{0}))))\}$ where $c = \mathsf{s}(\square_1)$. We have $\theta(x) \in [c^{0,2}(1)]$, $\theta(y) \in [c^{3,5}(\mathsf{0})]$ and $\theta(z) \in [c^{0,0}(z)]$ for all $z \in X \setminus \{x, y\}$. So, $\upsilon^{-1}(\theta) = \{y \mapsto c^3(y)\} \star \{x \mapsto c^2(1), y \mapsto c^5(\mathsf{0})\}$. For all $n \in \mathbb{N}$, we have $\theta(n) = \{x \mapsto c^2(1), y \mapsto c(c^{2n+1}(c^{n+2}(c(\mathsf{0}))))\} = \{x \mapsto c^2(1), y \mapsto c^{3n+5}(\mathsf{0})\} = (\upsilon^{-1}(\theta))(n)$.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

4.1 Unification of simple pattern terms

For all sequences $S = \langle p_1, \ldots, p_m \rangle$ of simple pattern terms, we define $\upsilon(S) = \{\langle u_1, \ldots, u_m \rangle \mid u_1 \in \upsilon(p_1), \ldots, u_m \in \upsilon(p_m)\}$.

Unification Algorithm 1

Let S and S' be sequences of simple pattern terms. Let $S_1 \in \upsilon(S)$ and $S'_1 \in \upsilon(S')$.

- If $mgu(S_1, S'_1)$ contains a simple substitution θ then return $\upsilon^{-1}(\theta)$
- else halt with failure.

Partial correctness follows from the next theorem.

Theorem 4.
If the unification algorithm successfully terminates then it produces a pattern substitution which is an mgu of the input sequences.

In practice, as S_1 and S'_1 are sequences of elements of $T(\Sigma \cup \Upsilon, X)$, one can use any classical unification algorithm (Robinson, Martelli-Montanari...) to compute $\theta \in mgu(S_1, S'_1)$. Then, it suffices to check whether θ is simple, for instance using Lem. 1.

Example 11
(Related to Ex. 5). *Consider the sequences of simple pattern terms* $S = \langle v_1{}^\star, v_2{}^\star, v_3{}^\star \rangle$ *and* $S' = \langle p'_1, p'_2, p'_3 \rangle$ *where* $v_1 = \mathsf{gt}(x, y)$, $v_2 = \mathsf{add}(x, y, z)$, $v_3 = \mathsf{while}(z, \mathsf{s}(y))$ *and*

$$p'_1 = \mathsf{gt}(x_1, y_1) \star \{x_1 \mapsto \mathsf{s}(x_1), y_1 \mapsto \mathsf{s}(y_1)\} \star \{x_1 \mapsto \mathsf{s}(x_1), y_1 \mapsto \mathsf{0}\}$$
$$p'_2 = \mathsf{add}(x_2, y_2, z_2) \star \{y_2 \mapsto \mathsf{s}(y_2), z_2 \mapsto \mathsf{s}(z_2)\} \star \{y_2 \mapsto \mathsf{0}, z_2 \mapsto x_2\}$$
$$p'_3 = \mathsf{while}(x_3, y_3)^\star$$

Let $c = \mathsf{s}(\square_1)$, $S_1 = \langle \mathsf{gt}(x, y), \mathsf{add}(x, y, z), \mathsf{while}(z, c(y)) \rangle$ *and*

$$S'_1 = \langle \mathsf{gt}(c^{1,1}(x_1), c^{1,0}(\mathsf{0})), \mathsf{add}\,(x_2, c^{1,0}(\mathsf{0}), c^{1,0}(x_2)), \mathsf{while}(x_3, y_3) \rangle$$

We have $S_1 \in \upsilon(S)$ *and* $S'_1 \in \upsilon(S')$. *Moreover,* $\theta \in mgu(S_1, S'_1)$ *where*

$$\theta = \{x \mapsto c^{1,1}(x_1),\ y \mapsto c^{1,0}(\mathsf{0}),\ z \mapsto c^{1,0}(c^{1,1}(x_1)),$$
$$x_2 \mapsto c^{1,1}(x_1),\ x_3 \mapsto c^{1,0}(c^{1,1}(x_1)),\ y_3 \mapsto c(c^{1,0}(\mathsf{0}))\}$$

We note that $\theta(x) \in [c^{1,1}(x_1)]$, $\theta(y) \in [c^{1,0}(\mathsf{0})]$, $\theta(z) \in [c^{2,1}(x_1)]$, $\theta(x_2) \in [c^{1,1}(x_1)]$, $\theta(x_3) \in [c^{2,1}(x_1)]$ *and* $\theta(y_3) \in [c^{1,1}(\mathsf{0})]$. *So,* θ *is a simple substitution and the algorithm produces the pattern substitution* $\upsilon^{-1}(\theta) = \rho \star \nu$ *where*

$$\rho = \{x \mapsto \mathsf{s}(x),\ y \mapsto \mathsf{s}(y),\ z \mapsto \mathsf{s}^2(z),\ x_2 \mapsto \mathsf{s}(x_2),\ x_3 \mapsto \mathsf{s}^2(x_3),\ y_3 \mapsto \mathsf{s}(y_3)\}$$
$$\nu = \{x \mapsto \mathsf{s}(x_1),\ y \mapsto \mathsf{0},\ z \mapsto \mathsf{s}(x_1),\ x_2 \mapsto \mathsf{s}(x_1),\ x_3 \mapsto \mathsf{s}(x_1),\ y_3 \mapsto \mathsf{s}(\mathsf{0})\}$$

By Thm. 4, $\rho \star \nu \in mgu(S, S')$.

A natural choice for S_1 and S'_1 in our unification algorithm is to consider, for all $p = s \star \sigma \star \mu$ in $S \cup S'$, the term $s\theta_p \in [s\theta_p]$ (see Def. 12). But this leads to an incomplete approach, *that is*, an approach that may fail to find a unifier even if one exists.

Example 12.
Let $p = s \star \{x \mapsto c(x)\} \star \emptyset$ *and* $q = s \star \{x \mapsto c_2(x)\} \star \{x \mapsto y\}$ *where* $s = \mathsf{f}(x)$, $c = \mathsf{s}(\square_1)$ *and* $c_2 = c^2$. *Then,* p *and* q *are simple. Let* $\theta = \{x \mapsto c(x)\} \star \{x \mapsto y\}$. *For all* $n \in \mathbb{N}$, *we have*

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

$p(n) = \mathsf{f}(c^n(x))$ and $q(n) = \mathsf{f}(c^{2n}(y))$, hence $\theta(n) = \{x \mapsto c^n(y)\}$ is a unifier of $p(n)$ and $q(n)$. Therefore, θ is a unifier of p and q. On the other hand, we have $s\theta_p = \mathsf{f}(c^{1,0}(x))$ and $s\theta_q = \mathsf{f}(c_2^{1,0}(y))$. As $c^{1,0}$ and $c_2^{1,0}$ are different symbols, $mgu(s\theta_p, s\theta_q) = \emptyset$. So, from $s\theta_p$ and $s\theta_q$, the unification algorithm fails to find a unifier for p and q. Now, let us choose the term $u = \mathsf{f}(c^{1,0}(c^{1,0}(y)))$ in $[s\theta_q]$. The substitution $\eta = \{x \mapsto c^{1,0}(y)\}$ is simple and belongs to $mgu(s\theta_p, u)$. So, the unification algorithm succeeds and returns $\upsilon^{-1}(\eta) = \theta$.

4.2 A non-termination criterion

Now, we provide a non-termination criterion that is simpler to implement than that of Thm. 3. It relies on pattern rules of the following form, which is easy to check in practice.

Definition 14.
We say that a pattern rule $r = (p, q)$ is special if it is simple and there exists

$$c(c_1^{a_1,b_1}(t_1), \ldots, c_m^{a_m,b_m}(t_m)) \in \upsilon(p) \quad \text{and} \quad c(c_1^{a'_1,b'_1}(t_1\rho), \ldots, c_m^{a'_m,b'_m}(t_m\rho)) \in \upsilon(q)$$

such that c is an m-context with $Var(c) = \emptyset$, $\rho \in S(\Sigma, X)$ and

1. $\forall i : (t_i \in X) \vee (t_i \in T(\Sigma, X) \wedge Var(t_i) = \emptyset)$,
2. $\forall i, j : (t_i \in X \wedge t_i = t_j) \Rightarrow c_i = c_j$,
3. $\{(a_i, a'_i) \mid Var(t_i) = \emptyset\} = \{(e, e)\}$ *with* $0 < e$, $\{(a_i, a'_i) \mid t_i \in X\} = \{(a, a')\}$ *with* $a \leq a'$,
4. $\{(b_i, b'_i) \mid Var(t_i) = \emptyset\} = \{(b, b')\}$ *with* $b \leq b'$, $\{(b_i, b'_i) \mid t_i \in X\} = \{(d, d')\}$,
5. $k = (b' - b)/e \in \mathbb{N}$ *and* $a = a' \Rightarrow 0 \leq (d' - d) - a \times k$.

Then, we let $\alpha(r) = 0$ if $a = a'$ and $\alpha(r) = \frac{a \times k - (d' - d)}{a' - a}$ otherwise.

The existence of a special pattern rule implies non-termination:

Theorem 5.
Let P be a program and $B \subseteq \Im$ be correct w.r.t. P. Suppose that $patunf(P, B)$ contains a special pattern rule $r = (p, q)$. Then, for all $n \in \mathbb{N}$ such that $n \geq \alpha(r)$ and all $\theta \in S(\Sigma, X)$, there is an infinite $\Rightarrow_P$-chain that starts from $\langle p(n)\theta \rangle$.

Example 13.
In Ex. 5, the set $patunf(P, B)$ contains the pattern rule $r'' = (p, q)$ and we have

$$\mathsf{while}(\mathsf{s}^{1,1}(x_1), \mathsf{s}^{1,0}(0)) = c\left(c_1^{a_1,b_1}(t_1), c_2^{a_2,b_2}(t_2)\right) \in \upsilon(p)$$

$$\mathsf{while}(\mathsf{s}^{2,1}(x_1), \mathsf{s}^{1,1}(0)) = c\left(c_1^{a'_1,b'_1}(t_1), c_2^{a'_2,b'_2}(0)\right) \in \upsilon(q)$$

(with a slight abuse of notation when writing $\mathsf{s}^{1,1}$, $\mathsf{s}^{1,0}$ and $\mathsf{s}^{2,1}$). Moreover,

- $\{(a_i, a'_i) \mid Var(t_i) = \emptyset\} = \{(a_2, a'_2)\} = \{(1, 1)\} = \{(e, e)\}$ *with* $0 < e$,
- $\{(a_i, a'_i) \mid t_i \in X\} = \{(a_1, a'_1)\} = \{(1, 2)\} = \{(a, a')\}$ *with* $a < a'$,
- $\{(b_i, b'_i) \mid Var(t_i) = \emptyset\} = \{(b_2, b'_2)\} = \{(0, 1)\} = \{(b, b')\}$ *with* $b \leq b'$,
- $\{(b_i, b'_i) \mid t_i \in X\} = \{(1, 1)\} = \{(d, d')\}$,
- $k = (b' - b)/e = (1 - 0)/1 = 1 \in \mathbb{N}$.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

So, $\alpha(r) = \frac{1\times 1-(1-1)}{2-1} = 1$. *Then, by Thm. 5, for all* $n \in \mathbb{N}$ *such that* $n \geq 1$ *and all* $\theta \in S(\Sigma, X)$, *there is an infinite* $\Rightarrow_P$*-chain that starts from* $\langle p(n)\theta\rangle = \langle \mathsf{while}(\mathsf{s}^{n+1}(x_1), \mathsf{s}^n(\mathsf{0}))\theta\rangle$. *This corresponds to what we observed in Ex. 2 and Ex. 6. For instance, from* $n = 1$ *and* $\theta = \{x_1 \mapsto \mathsf{0}\}$, *we get: there is an infinite* $\Rightarrow_P$*-chain that starts from* $\langle \mathsf{while}(\mathsf{s}^2(\mathsf{0}), \mathsf{s}(\mathsf{0}))\rangle$.

Def. 14 requires that $A_1 = \{i \mid Var(t_i) = \emptyset\}$ and $A_2 = \{i \mid t_i \in X\}$ are not empty. This can be lifted as follows. If $A_1 = \emptyset$ or $A_2 = \emptyset$ then we demand that $a_1 = \ldots = a_m = a$, $b_1 = \ldots = b_m = b$, $a'_1 = \ldots = a'_m = a'$ and $b'_1 = \ldots = b'_m = b'$. Moreover:

- if $A_1 = \emptyset$ then we replace 3–5 in Def. 14 by $a \leq a'$ and $a = a' \Rightarrow b \leq b'$; we also let $\alpha(r) = 0$ if $a = a'$ and $\alpha(r) = \frac{b-b'}{a'-a}$ otherwise;
- if $A_2 = \emptyset$ then we demand that $a = a'$ and we replace 3–5 in Def. 14 by $0 < a$ and $k = (b' - b)/a \in \mathbb{N}$; we also let $\alpha(r) = 0$.

5 Experimental evaluation

We have implemented the approach of Sect. 4 in our tool NTI, which is the only tool participating in the *International Termination Competition*[2] capable of disproving termination of logic programs (LPs). We used the natural choice for S_1 and S'_1 in the unification algorithm, even if it leads to an incomplete approach (see end of Sect. 4.1). We ran NTI on 41 LPs obtained by translating term rewrite systems (TRSs) of the *Termination Problem Data Base*[3] (TPDB) that are known to be non-looping non-terminating. These LPs are small but they are representative of the kind of non-termination that we want to capture. We used the following configuration: MacBook Pro 2020 with Apple M1 chip, 16 GB RAM, macOS Sequoia 15.4.1. Table 1 shows the results for 7 LPs obtained from directory `AProVE_10` (which consists of 14 TRSs but we discarded those that translate to LPs that are not in the scope of our technique, *i.e.*, LPs that terminate or involve 1-contexts which are not elements of $\chi^{(1)}$, see Ex. 9). Table 2 shows the results for 34 LPs obtained from directory `EEG_IJCAR_12`, originally proposed to evaluate the approach of Emmes *et al.* (2012) (it consists of 49 TRSs but, again, we discarded those that translate to LPs that are out of scope). The tables have the following structure: column "Program" gives the name of the program together with its number of rules and relations, "Mode" gives the mode of interest (`i` means *input*, *i.e.*, a term with no variable), "NTI" gives the non-terminating term provided by NTI, "#unf" gives the number of generated unfolded rules and "Time(ms)" gives the time in milliseconds (we used a time-out of 10 s). The 4 programs marked with † are LP translations of TRSs that have not been proven non-terminating by any TRS analyzer participating in the competition until 2024. The results show that our approach succeeds on them. On the other hand, our approach fails on 5 programs. Our results can be reproduced using our tool and the benchmarks available at https://github.com/etiennepayet/nti.

[2] http://termination-portal.org/wiki/Termination˙Competition
[3] http://termination-portal.org/wiki/TPDB

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

Table 1. *Logic programs obtained from* `TPDB/TRS_Standard/AProVE_10`

Program (#rules, #rel)	Mode	NTI	#unf	Time(ms)
`andIsNat` (3, 2)	`f(i,i)`	`f(0,0)`	5	100
`double` (3, 2)	`f(i)`	`f(0)`	4	100
`ex1` (3, 2)	`f(i,i)`	`f(0,0)`	4	100
`ex2` (3, 2)	`g(i)`	`g(0)`	4	100
`ex3` (4, 2)	`g(i,i)`	`g(0,0)`	12	100
`halfdouble` (7, 4)	`f(i)`	`f(0)`	10	100
`isNat` (3, 2)	`f(i)`	`f(0)`	4	100

Table 2. *Logic programs obtained from* `TPDB/TRS_Standard/EEG_IJCAR_12`

Program (#rules, #rel)	Mode	NTI	#unf	Time(ms)
`emmes-nonloop-ex1_1` (5, 4)	`f(i,i)`	`f(s(0),0)`	6	90
`emmes-nonloop-ex1_2` (7, 5) †	`f(i,i)`	`f(s(0),0)`	38	120
`emmes-nonloop-ex1_3` (7, 5) †	`f(i,i)`	`f(s(0),0)`	38	120
`emmes-nonloop-ex1_4` (7, 5)	`f(i,i)`	`f(s(0),0)`	29	120
`emmes-nonloop-ex1_5` (7, 5)	`f(i,i)`	`f(s(0),0)`	29	130
`emmes-nonloop-ex2_1` (6, 4) †	`f(i,i)`	`f(s(0),0)`	26	120
`emmes-nonloop-ex2_2` (5, 3)	`f(i,i)`	`f(s(0),0)`	8	100
`emmes-nonloop-ex2_3` (6, 4) †	`f(i,i)`	`f(s(0),0)`	26	120
`emmes-nonloop-ex2_4` (8, 5)	`f(i,i)`	`f(s(0),0)`	140	240
`emmes-nonloop-ex2_5` (8, 5)	`f(i,i)`	`f(s(0),0)`	140	270
`emmes-nonloop-ex3_1` (6, 4)	`f(i)`	`f(s(0))`	30	120
`emmes-nonloop-ex3_2` (6, 4)	`f(i)`	`f(s(0))`	30	110
`emmes-nonloop-ex3_3` (8, 5)	`f(i)`	`f(s(0))`	78	180
`emmes-nonloop-ex3_4` (8, 5)	`f(i)`	`f(s(0))`	178	270
`emmes-nonloop-ex4_1` (7, 4)	`f(i)`	`f(0)`	12	100
`emmes-nonloop-ex4_2` (9, 5)	`f(i)`	`f(0)`	40	140
`emmes-nonloop-ex4_3` (9, 5)	`f(i)`	`f(0)`	40	130
`emmes-nonloop-ex4_4` (9, 5)	`f(i)`	`f(0)`	39	140
`emmes-nonloop-ex5_1` (9, 5)	`f(i)`	`f(s(0))`	40	140
`emmes-nonloop-ex5_2` (8, 4)	`f(i)`	`f(s(0))`	14	100
`emmes-nonloop-ex5_3` (9, 5)	`f(i)`	`f(s(0))`	40	140
`enger-nonloop-ex_payet` (6, 3)	`while(i,i)`	?	1296	time out
`enger-nonloop-isDNat` (3, 2)	`f(i)`	`f(0)`	4	90
`enger-nonloop-isTrueList` (3, 2)	`f(i)`	`f(nil)`	4	90
`enger-nonloop-swap_decr` (5, 3)	`f(i)`	?	41017	time out
`enger-nonloop-swapX` (3, 2)	`g(i)`	`g(0)`	4	90
`enger-nonloop-swapXY` (3, 2)	`g(i,i)`	`g(0,0)`	4	90
`enger-nonloop-swapXY2` (3, 2)	`g(i,i)`	`g(0,0)`	4	90
`enger-nonloop-toOne` (5, 3)	`f(i)`	`f(s(0))`	7	90
`enger-nonloop-unbounded` (3, 2)	`h(i,i)`	`h(s(0),0)`	4	90
`enger-nonloop-while-lt` (3, 2)	`while(i,i)`	`while(0,0)`	4	90
`rybalchenko-nonloop-popl08` (15, 7)	`while(i,i)`	?	9505	time out
`velroyen-nonloop-AlternatingIncr_c` (11, 6)	`while(i)`	?	2035	time out
`velroyen-nonloop-ConvLower_c` (12, 6)	`while(i)`	?	1341	time out

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

6 Related work

The only other approach we are aware of for proving non-looping non-termination of logic programs is that of Payet (2024). Roughly, it detects infinite chains of the form $\overline{s}_0\,(\Rightarrow^*_{r_1} \circ \Rightarrow_{r_2})\,\overline{s}_1\,(\Rightarrow^*_{r_1} \circ \Rightarrow_{r_2}) \cdots$ where (r_1, r_2) is a *recurrent pair* of binary rules (we note that the infinite $\Rightarrow_P$-chain of Ex. 2 does not have this form). This approach seems to address another class of non-loopingness (compared to that of this paper): it is not able to disprove termination of the programs of Tables 1 and 2 and, on the other hand, it is able to disprove termination of programs[4] on which our approach fails.

Loop checking (see, *e.g.*, the paper by Bol *et al.* (1991)) is also related to our work. It attempts to prune infinite rewrites at runtime using necessary conditions for the existence of infinite chains (hence, there is a risk of pruning a finite rewrite). In contrast, our approach uses a sufficient condition (see Thm. 3 and Thm. 5) to *prove* the existence of atomic goals that start a non-terminating chain.

Tabling (see, *e.g.*, the paper by Sagonas *et al.* (1994)) is another related technique that avoids infinite rewrites by storing intermediate results in a table and reusing them when needed. Only loops of a particular simple form are detected (*i.e.*, when a variant occurs in the evaluation of a subgoal). We are not aware of any tabling-based approach that can capture non-looping non-termination.

Another related technique is that of Payet and Mesnard (2006) which proves the existence of loops using neutral argument positions of predicate symbols.

7 Conclusion

We have presented a new approach, based on a new unfolding technique that generates correct patterns of rules, to disprove termination of logic programs. We have implemented it in our tool NTI and we have successfully evaluated it on logic programs obtained by translating TRSs from the TPDB.

Future work will be concerned with completeness of our unification algorithm, extending Prop. 2 to get more initial pattern rules and taking into account 1-contexts with variables in simple pattern terms (to deal with programs as that of Ex. 9). We also plan to adapt our approach to TRSs and to compare it to that of Emmes *et al.* (2012). Moreover, we will compare our approach to that of Payet (2024) on a theoretical level.

Acknowledgements

The author has no competing interests to declare. He thanks the anonymous reviewers for their insightful and constructive criticisms.

References

Apt, K. R. 1997. *From Logic Programming to Prolog*. Prentice Hall International series in computer science. Prentice Hall.

[4] *E.g.*, those in directories `TPDB/Logic_Programming/Payet_22`, `TPDB/Logic_Programming/Payet_23` and `TPDB/Logic_Programming/Payet_24`, proposed to evaluate the approach of Payet (2024).

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press

Baader, F. and Nipkow, T. 1998. *Term Rewriting and All That.* Cambridge University Press.

Bol, R. N., Apt, K. R. and KLOP, J. W. 1991. An analysis of loop checking mechanisms for logic programs. *Theoretical Computer Science* 86, 1, 35–79.

Codish, M. and Taboch, C. 1999. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming* 41, 1, 103–123.

Emmes, F., Enger, T. and GIESL, J. 2012. Proving non-looping non-termination automatically. In Proc. of the 6th International Joint Conference on Automated Reasoning (IJCAR'12), B. Gramlich, D. Miller and U. Sattler, Eds. LNCS, Springer, Vol. 7364, 225–240.

Payet, E. 2024. Non-termination in term rewriting and logic programming. *Journal of Automated Reasoning* 68, 4, 24.

Payet, E. and Mesnard, F. 2006. Nontermination inference of logic programs. *ACM Transactions on Programming Languages and Systems* 28, 2, 256–289.

Sagonas, K., Swift, T. and WARREN, D. S. 1994. XSB as an efficient deductive database engine. In Proc. of the 1994 ACM SIGMOD International Conference on Management of Data, R. T. Snodgrass and M. Winslett, Ed. ACM Press, 442–453.

https://doi.org/10.1017/S1471068425100100 Published online by Cambridge University Press