Proving non-termination by finite automata

Jörg Endrullis[1] and Hans Zantema[2,3]

1 **Department of Computer Science, VU University Amsterdam, 1081 HV Amsterdam, The Netherlands, email:** j.endrullis@vu.nl
2 **Department of Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, email:** H.Zantema@tue.nl
3 **Institute for Computing and Information Sciences, Radboud University Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands**

Abstract

A new technique is presented to prove non-termination of term rewriting. The basic idea is to find a non-empty regular language of terms that is closed under rewriting and does not contain normal forms. It is automated by representing the language by a tree automaton with a fixed number of states, and expressing the mentioned requirements in a SAT formula. Satisfiability of this formula implies non-termination. Our approach succeeds for many examples where all earlier techniques fail, for instance for the S-rule from combinatory logic.

1998 ACM Subject Classification D.1.1 Applicative (Functional) Programming, D.3.1 Formal Definitions and Theory, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems, I.1.1 Expressions and Their Representation, I.1.3 Languages and Systems

Keywords and phrases Non-termination, finite automata, regular languages

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.160

1 Introduction

A basic approach for proving that a term rewriting system (TRS) is non-terminating is to prove that it admits a *loop*, that is, a reduction of the shape $t \to^+ C[t\sigma]$, see [26]. Indeed, such a loop gives rise to an infinite reduction $t \to^+ C[t\sigma] \to^+ C[(C[t\sigma])\sigma] \to \cdots$ in which in every step t is replaced by $C[t\sigma]$. In trying to prove non-termination, several tools ([1, 2]) search for a loop. An extension from [7], implemented in [1] goes a step further: it searches for reductions of the shape $t\sigma^n\mu \to^+ C[t\sigma^{f(n)}\mu\tau]$ for every n for a linear increasing function f, and some extensions. All of these patterns are chosen to be extended to an infinite reduction in an obvious way, hence proving non-termination. However, many non-terminating TRSs exist not admitting an infinite reduction of this regular shape, or the technique from [7] fails to find it.

A crucial example is the S-rule $a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$, one of the building blocks of Combinatory Logic. Although being only one single rule, and having nice properties like orthogonality, non-termination of this system is a hard issue. Infinite reductions are known, but are of a complicated shape, see [31]. So developing a general technique that can prove non-termination of the S-rule automatically is a great challenge. In this paper we succeed in presenting such a technique, and we describe a SAT-based approach by which non-termination of many TRSs, including the S-rule, is proved fully automatically.

The underlying idea is quite simple: non-termination immediately follows from the existence of a non-empty set of terms that is closed under rewriting and does not contain normal forms. Our approach is to find such a set being the language accepted by a finite tree automaton, and find this tree automaton from the satisfying assignment of a SAT formula

26th International Conference on Rewriting Techniques and Applications (RTA'15).
Editor: Maribel Fernández; pp. 160–176

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

describing the above requirements. Hence the goal is to describe the requirements, namely non-emptiness, closed under rewriting, and not containing normal forms, in a SAT formula.

We want to stress that having quick methods for proving non-termination of term rewriting also may be fruitful for proving termination. In a typical search for a termination proof, like using the dependency pair framework, the original problem is transformed in several ways to other termination problems that not all need to be terminating. Being able to quickly recognize non-termination of some of them makes a further search for termination proofs redundant, which may speed up the overall search for a termination proof.

We note that, like termination, non-termination is an undecidable property. However, while termination is Π^0_2-complete, non-termination is Σ^0_2-complete [11, 10].

The paper is organized as follows. In Section 2 we present our basic approach in the setting of abstract reduction systems on a set T, in which the language is just a subset of T. Surprisingly, being not weakly normalizing corresponds to (strongly) closed under rewriting, and being not strongly normalizing corresponds to weakly closed under rewriting. In Section 3 we give preliminaries on tree automata and show how string automata can be seen as an instance of tree automata. In Section 4 we present our basic methods, starting by how the requirements are expressed in SAT, and next how this is used to disprove weak normalization and strong normalization. In Section 5 we strengthen our approach by labeling the states of the tree automata by sets of rewrite rules and exploiting this in the method. In Section 6 we present experimental results of our implementation. We conclude in Section 7.

1.1 Related Work

The paper [26] introduces the notion of loops and investigates necessary conditions for the existence of them. The work [34] employs SAT solvers to find loops, [35] uses forward closures to find loops efficiently, and [33] introduces 'compressed loops' to find certain forms of very long loops. Non-termination beyond loops has been investigated in [29] and [7]. There the basic idea is the search for a particular generalization of loops, like a term t and substitutions σ, τ such that for every n there exist C, μ such that $t\sigma^n\tau$ rewrites to $C[t\sigma^{f(n)}\tau\mu]$, for some ascending linear function f. Although the S-rule admits such reductions, these techniques fail to find them. For other examples for which not even reductions exist of the shape studied in [29] and [7], we will be able to prove non-termination fully automatically.

Our approach can be summarized as searching for non-termination proofs based on regular (tree) automata. Regular (tree) automata have been fruitfully applied to a wide rage of properties of term rewriting systems: for proving termination [25, 21, 27], infinitary normalization [12], liveness [28], and for analyzing reachability and deciding the existence of common reducts [23, 13]. Local termination on regular languages, has been investigated in [9].

2 Abstract Rewriting

An *abstract reduction system* (ARS) is a binary relation $\to$ on a set T. We write $\to^+$ for the transitive closure, and $\to^*$ for the reflexive, transitive closure of $\to$.

Let $\to$ be an ARS on T. The ARS $\to$ is called *terminating* or *strongly normalizing* (SN) if no infinite sequence $t_0, t_1, t_2, \ldots \in T$ exists such that $t_i \to t_{i+1}$ for all $i \geq 0$. A *normal form* with respect to $\to$ is an element $t \in T$ such that no $u \in T$ exists satisfying $t \to u$. The set of all normal forms with respect to $\to$ is denoted by $\mathsf{NF}(\to)$. The ARS $\to$ is called *weakly normalizing* (WN) if for every $t \in T$ a normal form $u \in T$ exists such that $t \to^* u$.

▶ **Definition 1.** A set $L \subseteq T$ is called
- *closed under* $\to$ if for all $t \in L$ and all $u \in T$ satisfying $t \to u$ it holds $u \in L$, and
- *weakly closed under* $\to$ if for all $t \in L \setminus \mathsf{NF}(\to)$ there exists $u \in L$ such that $t \to u$.

It is straightforward from these definitions that if L is closed under $\to$, then L is weakly closed under $\to$ as well. The following theorems relate these notions to SN and WN.

▶ **Theorem 2.** *An ARS* $\to$ *on* T *is not* SN *if and only if a non-empty* $L \subseteq T$ *exists such that* $L \cap \mathsf{NF}(\to) = \varnothing$ *and* L *is weakly closed under* $\to^+$.

Proof. If $\to$ is not SN then an infinite sequence $t_0, t_1, t_2, \ldots \in T$ exists such that $t_i \to t_{i+1}$ for all $i \geq 0$. Then $L = \{t_i \mid i \geq 0\}$ satisfies the required properties.

Conversely, assume L satisfies the given properties. Since L is non-empty we can choose $t_0 \in L$, and using the other properties for $i = 0, 1, 2, 3, \ldots$ we can choose $t_{i+1} \in L$ such that $t_i \to^+ t_{i+1}$, proving that $\to$ is not SN. ◀

▶ **Theorem 3.** *An ARS* $\to$ *on* T *is not* WN *if and only if a non-empty* $L \subseteq T$ *exists such that* $L \cap \mathsf{NF}(\to) = \varnothing$ *and* L *is closed under* $\to$.

Proof. If $\to$ is not WN then $t \in T$ exists such that $L \cap \mathsf{NF}(\to) = \varnothing$ for $L = \{u \in T \mid t \to^* u\}$. Then L satisfies the required properties.

Conversely, assume L satisfies the given properties. Since L is non-empty we can choose $t_0 \in L$. Assume that $\to$ is WN, then $t_0 \to t_1 \to \cdots \to t_n$ exists such that $t_n \in \mathsf{NF}(\to)$. Since L is closed under $\to$ we obtain $t_i \in L$ for $i = 1, 2, \ldots, n$, contradicting $L \cap \mathsf{NF}(\to) = \varnothing$. ◀

A variant of Theorem 2, where $\to^+$ is replaced by $\to$, has been observed in [5]. To the best knowledge of the authors, Theorem 3 has not been observed in the literature.

3 Tree Automata

▶ **Definition 4.** A *(non-deterministic finite) tree automaton* A *over a signature* Σ is a tuple $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ where

(i) Q is a finite set of *states*,
(ii) $F \subseteq Q$ is a set of *accepting states*, and
(iii) δ a set of rewrite rules, called *transition rules*, of the shape

$$f(q_1, \ldots, q_n) \rightsquigarrow q$$

where n is the arity of $f \in \Sigma$ and $q_1, \ldots, q_n, q \in Q$. We write $\rightsquigarrow$ for the rewrite relation generated by the rules δ.

Note that we use $\rightsquigarrow$ to distinguish automata transitions from term rewriting $\to$ with respect to some TRS R.

▶ **Definition 5.** The *language* $\mathcal{L}(A)$ *accepted by* A is the set

$$\mathcal{L}(A) = \{\, t \mid t \in T(\Sigma, \varnothing),\ q \in F,\ t \rightsquigarrow^* q \,\}$$

of ground terms that rewrite to a final state.

The kind of tree automata considered here is called *bottom up* in the literature. Sometimes in the definition of bottom-up tree automaton the right hand side q in the rule has arguments and the acceptance criterion is rewriting to a term with a final state as root. However, when

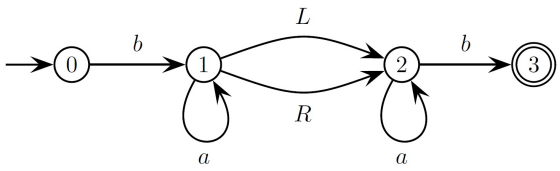
tree automata are only used for defining (term) languages as is the case in this paper, these definitions coincide.

Tree automata can be seen as a generalization of string automata as follows. For a string automaton (= NFA) S define the tree automaton A by

- taking the same sets of states and accepting states, and
- taking as signature the same signature in which all symbols are unary, extended by a single constant ε, and
- taking as transition rules $\varepsilon \rightsquigarrow q_0$ for q_0 being the initial state of S, and for every transition $q \xrightarrow{a} q'$ in S the rule $a(q) \rightsquigarrow q'$.

Form this definition it is immediate that a string $a_1 a_2 \cdots a_n$ is accepted by S if and only if $a_n(a_{n-1}(\cdots(a_1(\varepsilon))\cdots))$ is accepted by A. Here we assume that S reads the string from left to right (otherwise there is no need to reverse the order of the letters).

▶ **Example 6.** To define a tree automaton accepting the language $b\,a^*\,(L|R)\,a^*\,b$, that is, all words that start with b, end with b, contain one L or R and otherwise only a, we start by its corresponding string automaton



The above construction yields the tree automaton $A_{LR} = \langle\, Q, \Sigma, F, \delta \,\rangle$ where $\Sigma = \{b, L, R, a, \varepsilon\}$ in which b, L, R, a are unary and ε is a constant, $Q = \{0, 1, 2, 3\}$, $F = \{3\}$ and δ consists of the rules

$$\varepsilon \rightsquigarrow 0 \qquad a(1) \rightsquigarrow 1 \qquad b(0) \rightsquigarrow 1 \qquad R(1) \rightsquigarrow 2 \qquad L(1) \rightsquigarrow 2$$
$$a(2) \rightsquigarrow 2 \qquad b(2) \rightsquigarrow 3$$

▶ **Example 7.** The following is a tree automaton for the signature $\Sigma = \{a, S\}$ where a is binary and S is a constant. Let $A_S = \langle\, Q, \Sigma, F, \delta \,\rangle$ where $Q = \{0, 1, 2, 3, 4\}$, $F = \{4\}$ and

$$S \rightsquigarrow 0 \qquad a(0,0) \rightsquigarrow 1 \qquad a(1,0) \rightsquigarrow 2 \qquad a(2,2) \rightsquigarrow 3 \qquad a(3,3) \rightsquigarrow 3$$
$$a(0,2) \rightsquigarrow 2 \qquad\qquad\qquad a(2,3) \rightsquigarrow 3 \qquad a(3,3) \rightsquigarrow 4$$
$$a(0,3) \rightsquigarrow 2$$

As is usual in combinatory logic, ground terms are represented by omitting the a symbol and writing $uvw = (uv)w$. We show that this automaton accepts the term $SSS(SSS)(SSS(SSS))$:

$$SSS(SSS)(SSS(SSS)) \rightsquigarrow^{12} 000(000)(000(000))$$
$$\rightsquigarrow^{4} 10(10)(10(10)) \rightsquigarrow^{4} 22(22) \rightsquigarrow^{2} 33 \rightsquigarrow^{1} 4$$

Since $4 \in F$ the term is accepted by the automaton.

This automaton has been found automatically by our tool, and its language is closely related to the $\mathcal{QQQ}$-criterion of Waldmann [31, 3]. Roughly speaking, the language recognized by this automaton can be described as follows:

- state 0 accepts only the term S,
- state 1 accepts only the term SS,
- state 2 corresponds to terms that contain at least *one* occurrence of SSS,
- state 3 corresponds to terms that contain at least *two* occurrence of SSS, and
- state 4 accepts terms MN for which both M and N contain two occurrences of SSS.

4 Basic Methods

In this section, we are concerned with automating the abstract non-termination methods from Section 2. To this end, we use finite tree automata giving rise to regular tree languages. We first develop methods for disproving weak normalization and then for disproving strong normalization.

4.1 SAT Encoding of Properties

In this section, we collect decision procedures for the main properties of tree automata that we employ for proving non-termination, and we describe how we encode these procedures as Boolean satisfiability problems (SAT).

▶ **Remark 8** (SAT encoding of tree automata)**.** We encode the search for a tree automaton $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ over a signature Σ as a satisfiability problem as follows. We pick the number of states $n \in \mathbb{N}$ the automaton should have; the set of states is $Q = \{s_1, \ldots, s_n\}$. While the set of states Q is fix, we represent the final states $F \subseteq Q$ by n fresh Boolean variables

$$v_{F,s_1},\ v_{F,s_2},\ v_{F,s_3}, \ldots, v_{F,s_n}$$

and, for every $f \in \Sigma$, we represent the transition relation δ by fresh variables

$$v_{f,q_1,\ldots,q_{\#(f)},q} \qquad \text{for every } q_1, \ldots, q_{\#(f)}, q \in Q$$

For the moment, there are no constraints (formulas) and the interpretation of these variables can be chosen freely. The intention is that v_{F,s_i} is true if and only if s_i is a final state, and $v_{f,q_1,\ldots,q_{\#(f)},q}$ is true if and only if $f(q_1, \ldots, q_{\#(f)}) \rightsquigarrow q$ is a transition rule in δ.

▶ **Definition 9.** A state $q \in Q$ of a tree automaton $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ is called *reachable* if there exists a ground term $t \in T(\Sigma, \varnothing)$ such that $t \rightsquigarrow^* q$.

We assume, without loss of generality, that all states are reachable. Note that requiring that all states are reachable is not a restriction since we can always replace unreachable states by 'copies' of reachable states. We guarantee reachability as follows.

▶ **Lemma 10.** *Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ be a tree automaton. Then all states of A are reachable if and only if there exists a total well-founded order $<$ on the states Q such that for every $q \in Q$ there exists $f \in \Sigma$ and states $q_1 < q$, $q_2 < q$, $\ldots$, $q_{\#(f)} < q$ with $f(q_1, \ldots, q_{\#(f)}) \rightsquigarrow q$.*

Proof. The reachable states are the smallest set $Q' \subseteq Q$ that is closed under δ, that is, $q \in Q'$ whenever $f(q_1, \ldots, q_{\#(f)}) \rightsquigarrow q$ for some $f \in \Sigma$ and states $q_1, \ldots, q_{\#(f)} \in Q'$.

For the 'if'-part, assume that there was a non-reachable state. Let $q \in Q$ be the smallest non-reachable state with respect to the order $<$. By assumption there exist $f \in \Sigma$ and states $q_1 < q$, $q_2 < q$, $\ldots$, $q_{\#(f)} < q$ with $f(q_1, \ldots, q_{\#(f)}) \rightsquigarrow q$. By choice of q it follows that all states $q_1, q_2, \ldots, q_{\#(f)}$ are reachable, and hence q is reachable, contradicting the assumption.

For the 'only if'-part, assume that all states are reachable. Then Q is the result of stepwise closing $\varnothing$ under δ. There exists a sequence of states $\varnothing = Q_0 \subseteq Q_1 \subseteq \ldots \subseteq Q_{|Q|} = Q$ such that for every $0 \leq i < |Q|$ we have $Q_{i+1} = Q_i \cup \{q_i\}$ for some $q_i \in Q \setminus Q_i$ such that there are $f_i \in \Sigma$ and states $q_{i,1}, \ldots, q_{i,\#(f_i)} \in Q_i$ with $f_i(q_{i,1}, \ldots, q_{i,\#(f)}) \rightsquigarrow q_i$. The order $<$ induced by $q_0 < q_1 < \ldots < q_{|Q|-1}$ is a total order on the states with the desired property. ◀

▶ **Remark 11** (SAT encoding of reachability of all states)**.** We extend the encoding of tree automata as described in Remark 8. We want to guarantee that all states are reachable by

employing Lemma 10. However, instead of encoding an arbitrary well-founded relation, we make use of the fact that the names of states are irrelevant. Hence, without loss of generality (modulo renaming of states), we may assume that $s_1 < s_2 < \ldots < s_n$. We then encode the condition of Lemma 10 by formulas

$$\bigvee_{f\in\Sigma,\ q_1<q,\ldots,\ q_{\#(n)}<q} v_{f,q_1,\ldots,q_n,q}$$

for every $q \in Q$.

The following lemma is immediate.

▶ **Lemma 12.** *Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ be a tree automaton such that all states are reachable. Then $\mathcal{L}(A) \neq \varnothing$ if and only if $F \neq \varnothing$.*

▶ **Remark 13** (SAT encoding of $\mathcal{L}(A) \neq \varnothing$)**.** In a setting where all states are reachable, the encoding of $\mathcal{L}(A) \neq \varnothing$ as satisfiability problem trivializes to: $\bigvee_{q\in Q} v_{F,q}$.

The following lemma gives a simple criterion for closure under rewriting.

▶ **Lemma 14** (Genet [24, Proposition 12])**.** *Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ be a tree automaton and R a left-linear term rewriting system. Then $\mathcal{L}(A)$ is closed under rewriting with respect to R if for every $\ell \to r \in R$, $\alpha : \mathcal{X} \to Q$ and $q \in Q$ we have $\ell\alpha \rightsquigarrow_A^* q \implies r\alpha \rightsquigarrow_A^* q$.*

Note that left-linearity of R is crucial for the Lemma 14 since A can be a non-deterministic automaton. If R would contain non-left-linear rules $\ell \to r$ then we would need to check set-assignments $\alpha : \mathcal{X} \to \wp(Q)$ instead $\alpha : \mathcal{X} \to Q$. That is, we would need to take into account, that a non-deterministic automaton can interpret the same term by different states.

For terms t, we use $\mathit{Var}(t)$ to denote the set of variables occurring in t.

▶ **Remark 15** (SAT encoding of closure under rewriting)**.** We encode the conditions of Lemma 14. Let the automaton A be encoded as in Remark 8. Let U be the set of all non-variable subterms of left-hand sides and right-hand sides of rules in R. For every $t \in U$, assignment $\alpha : \mathit{Var}(t) \to Q$ and $q \in Q$ we introduce a fresh variable

$$v_{t,\alpha,q} \qquad \text{with the intended meaning: } v_{t,\alpha,q} \text{ is true } \iff t\alpha \rightsquigarrow^* q\ .$$

We ensure this meaning by the following formulas: for terms $t = f(t_1, \ldots, t_n) \in U$

$$v_{t,\alpha,q} \iff \bigvee_{q_1,\ldots,q_n\in Q} \big(v_{t_1,\alpha_1,q_1} \wedge \ldots \wedge v_{t_n,\alpha_n,q_n} \wedge v_{f,q_1,\ldots,q_n,q}\big)$$

where α_i is the restriction of α to the domain $\mathit{Var}(t_i)$. For variables $x \in U$, we stipulate $v_{x,\alpha,q} \iff \alpha(x) = q$; note that we can immediately evaluate and fill in these truth values. Finally, we encode $\ell\alpha \rightsquigarrow_A^* q \implies r\alpha \rightsquigarrow_A^* q$ by formulas

$$v_{\ell,\alpha,q} \to v_{r,\alpha,q}$$

for every $\ell \to r \in R$, $\alpha : \mathit{Var}(\ell) \to Q$ and $q \in Q$.

The following modification of Lemma 14 gives a simple criterion for weak closure under rewriting. The requirement $r\alpha \rightsquigarrow_A^* q$ of Lemma 14 is weakened to $t\alpha \rightsquigarrow_A^* q$ for some reduct t the left-hand side ℓ.

▶ **Lemma 16.** *Let $A = \langle Q, \Sigma, F, \delta \rangle$ be a tree automaton and R a left-linear term rewriting system. Then $\mathcal{L}(A)$ is weakly closed under rewriting with respect to R if for every $\ell \to r \in R$, $\alpha : \mathcal{X} \to Q$ and $q \in Q$ we have $\ell\alpha \rightsquigarrow_A^* q \implies t\alpha \rightsquigarrow_A^* q$ for some term t such that $\ell \to_R^+ t$.*

Proof. Let $s \in \mathcal{L}(A) \setminus \mathsf{NF}(\to_R)$. Then $s = C[\ell\sigma]$ for a context C, rewrite rule $\ell \to r \in R$ and substitution $\sigma : \mathcal{X} \to T(\Sigma, \varnothing)$. Since $s \in \mathcal{L}(A)$ there exists $q \in Q$ such that $\ell\sigma \rightsquigarrow^* q$ and $C[q] \rightsquigarrow^* q'$ with $q' \in F$. By left-linearity ℓ does not contain duplicated occurrences of variables. As a consequence, there exists $\alpha : \mathcal{X} \to Q$ such that $\sigma(x) \rightsquigarrow^* \alpha(x)$ and $\ell\alpha \rightsquigarrow^* q$. By the assumptions of the lemma, a term t exists such that $\ell \to_R^+ t$ and $t\alpha \rightsquigarrow^* q$. Hence $t\sigma \rightsquigarrow^* q$ and $C[t\sigma] \rightsquigarrow^* C[q] \rightsquigarrow^* q'$. Thus $C[t\sigma] \in \mathcal{L}(A)$. Since $s = C[\ell\sigma] \to_R^+ C[t\sigma]$, this proves that $\mathcal{L}(A)$ is weakly closed under rewriting with respect to R. ◀

▶ **Remark 17** (SAT encoding of Lemma 16)**.** The conditions of Lemma 16 can be encoded similar to Lemma 14 (described in Remark 15). In Lemma 16 the condition $r\alpha \rightsquigarrow_A^* q$ is weakened to: $t\alpha \rightsquigarrow_A^* q$ for some reduct t the left-hand side ℓ. We can pick a finite set of reducts $U \subseteq \{t \mid \ell \to^+ t\}$ of the left-hand side ℓ, and encode the disjunction $\bigvee_{t \in U} t\alpha \rightsquigarrow_A^* q$ as a Boolean satisfiability problem. Note that $U \neq \varnothing$ since $r \in U$.

Next, we want to guarantee that the language $\mathcal{L}(A)$ contains no normal forms, in other words, that every term in the language contains a redex. For left-linear term rewriting systems R, we can reduce this problem to language inclusion $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ where B is a tree automaton that accepts the language of reducible terms. If R is a left-linear rewrite system, then the set of ground terms containing redex occurrences is a regular tree language. A deterministic automaton B for this language can be constructed using the overlap-closure of subterms of left-hand sides, see further [15, 16]. Here, we do not repeat the construction, but state the lemma that we will employ:

▶ **Lemma 18.** *Let $\{\ell_1, \ldots, \ell_n\}$ be a set of linear terms over Σ. Then we can construct a deterministic and complete automaton $B = \langle Q, \Sigma, F, \delta \rangle$ and sets $F_{\ell_1}, \ldots, F_{\ell_n} \subseteq Q$ such that for every term $t \in T(\Sigma, \varnothing)$ and $i \in \{1, \ldots, n\}$ we have:*

- *$t \rightsquigarrow^* q$ with $q \in F_{\ell_i}$ if and only if t is an instance of ℓ_i.*

Note that by choosing $F = F_{\ell_1} \cup \ldots \cup F_{\ell_n}$ we obtain: $t \rightsquigarrow^* q$ with $q \in F$ if and only if t is an instance ℓ_i for some $i \in \{1, \ldots, n\}$.

▶ **Example 19.** The following tree automaton $B_S = \langle Q, \Sigma, F, \delta \rangle$ accepts the language of ground terms that contain a redex occurrence with respect to the S-rule $a(a(a(S, x), y), z) \to a(a(x, z), a(y, z))$. Here $Q = \{0, 1, 2, 3\}$, $\Sigma = \{a, S\}$, $F = \{3\}$ and

$$S \rightsquigarrow 0 \qquad a(0, q) \rightsquigarrow 1 \qquad a(1, q) \rightsquigarrow 2 \qquad a(2, q) \rightsquigarrow 3 \qquad a(3, q) \rightsquigarrow 3 \qquad a(q', 3) \rightsquigarrow 3$$

for all $q \in \{0, 1, 2\}$ and $q' \in \{0, 1, 2, 3\}$.

Since the automaton B_S is deterministic and complete, we can obtain an automaton $\overline{B_S} = \langle Q, \Sigma, \overline{F}, \delta \rangle$ that accepts the complement of the language (the language of ground normal forms) by taking the complement $\overline{F} = \{0, 1, 2\}$ of the set of final states.

The following is crucial for feasibility of our approach. Deciding language inclusion of non-deterministic automata is known to be EXPTIME complete, see [30]. However, to guarantee that a language contains no normal forms, it suffices to check whether two non-deterministic automata have a non-empty intersection. This property can be decided in polynomial time by constructing the product automaton and considering the reachable states.

▶ **Definition 20.** The *product* $A \times B$ of tree automata $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ and $B = \langle\, Q', \Sigma, F', \delta' \,\rangle$ is the tree automaton $C = \langle\, Q \times Q', \Sigma, F \times F', \gamma \,\rangle$ where the transition relation γ is given by

$$f(\,(q_1,p_1),\ldots,(q_n,p_n)\,) \rightsquigarrow_\gamma (q',p') \quad\iff\quad f(q_1,\ldots,q_n) \rightsquigarrow_\delta q' \;\wedge\; f(p_1,\ldots,p_n) \rightsquigarrow_{\delta'} p'$$

for every $f \in \Sigma$ of arity n and states $q_1,\ldots,q_n,q' \in Q$ and $p_1,\ldots,p_n,p' \in Q'$.

▶ **Lemma 21.** *Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ and $B = \langle\, Q', \Sigma, F', \delta' \,\rangle$ be tree automata. Then we have $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$ if and only if in $A \times B$ no state in $F \times F'$ is reachable.*

Proof. Let $A \times B = \langle\, Q \times Q', \Sigma, \varnothing, \gamma \,\rangle$. For the 'if'-part, assume that $\mathcal{L}(A) \cap \mathcal{L}(B) \neq \varnothing$. Let $t \in \mathcal{L}(A) \cap \mathcal{L}(B)$. Then $t \rightsquigarrow_\delta^* q$ for some $q \in F$ and $t \rightsquigarrow_{\delta'}^* q'$ for some $q' \in F'$. But then $t \rightsquigarrow_\gamma^* (q,q')$ and hence $(q,q') \in F \times F'$ is reachable in $A \times B$; this contradicts the assumption.

For the 'only if'-part, assume, for a contradiction, that $t \rightsquigarrow_\gamma^* (q,q')$ in $A \times B$ with $q \in F$ and $q' \in F'$. Then this directly translates to $t \rightsquigarrow_\delta^* q$ in A and $t \rightsquigarrow_{\delta'}^* q'$ in B. Hence $t \in \mathcal{L}(A)$ and $t \in \mathcal{L}(B)$, contradicting $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$. ◀

We can use Lemma 21 to check that the language $\mathcal{L}(A)$ of an automaton A does not contain normal forms. To this end, we only need an automaton B that accepts all ground normal forms. Then $\mathcal{L}(A)$ contains no normal forms if $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$.

▶ **Example 22.** The reachable states of the product $A_S \times \overline{B_S}$ of the automata A_S from Example 7 and $\overline{B_S}$ from Example 19 are $(0,0),(1,1),(2,2),(2,1),(3,3),(3,2),(2,3),(4,3)$. The only state (q,q') such that q is accepting in A_S is $(4,3)$ and 3 is not an accepting state of $\overline{B_S}$. The conditions of Lemma 21 are fulfilled and hence $\mathcal{L}(A_S) \cap \mathcal{L}(\overline{B_S}) = \varnothing$. Recall that $\overline{B_S}$ accepts all ground normal forms, and thus every term accepted by A_S contains a redex.

▶ **Remark 23** (SAT encoding of language inclusion)**.** Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ and $B = \langle\, Q', \Sigma, F', \delta' \,\rangle$ be tree automata. Let $A \times B = \langle\, Q \times Q', \Sigma, \varnothing, \gamma \,\rangle$.

First, note that reachability of all states in the automata A and B does not imply that all states in the product automaton $A \times B$ are reachable. As a consequence, we have to 'compute' the set of reachable states using Boolean satisfiability problems. For this purpose, we reformulate Lemma 21 in the following equivalent way: …, then $\mathcal{L}(A) \cap \mathcal{L}(B) = \varnothing$ if and only if there exists a set of states $P \subseteq Q \times Q'$ such that

(a) P is *closed under transitions* in $A \times B$, that is, $q \in P$ whenever $f(q_1,\ldots,q_n) \rightsquigarrow_\gamma q$ for some $q_1,\ldots,q_n \in P$, and

(b) for all $(q,q') \in P$ it holds that $q \in F$ implies $q' \notin F'$.

Note that this statement is equivalent to Lemma 21. Item (a) guarantees that P contains all reachable states, and hence (b) is required for at least the reachable states. Thus the conditions imply those of Lemma 21. On the other hand, we can take P to be precisely the set of reachable states, and then the conditions are exactly those of Lemma 21.

The idea is that the reformulated statement has a much more efficient encoding as Boolean satisfiability problem. We only need to encode the closure of P under transitions, but there is no longer the need for encoding the property that P is the smallest such set (which is a statement of second-order logic).

Assume that we have a SAT encoding of the automata A and B as in Remark 8; we write $v_{A,\ldots}$ for the variables encoding A, and $v_{B,\ldots}$ for the variables encoding B. To represent the set P, we introduce variables $p_{(q,q')}$ for every $(q,q') \in Q \times Q'$ and the properties are translated into the following formulas:

(a) for every $f \in \Sigma$ with arity n and $(q_1, q_1'), \dots, (q_n, q_n'), (q, q') \in Q \times Q'$:

$$(v_{A,f,q_1,\dots,q_n,q} \wedge v_{B,f,q_1',\dots,q_n',q'} \wedge p_{(q_1,q_1')} \wedge p_{(q_2,q_2')} \wedge \dots \wedge p_{(q_n,q_n')}) \to p_{(q,q')}$$

(b) for every $(q, q') \in Q \times Q'$: $(p_{(q,q')} \wedge v_{A,F,q}) \to \neg v_{B,F',q'}$.

Each of these formulas simplifies to a single clause (a disjunction of literals).

We remark that we will employ this translation for the case that B consists of the set of terms containing redex occurrences with respect to a given rewrite system R. Then B is known and fixed before the translation to a satisfiability problem. As a consequence, we know the truth values of $v_{B,f,q_1',\dots,q_n',q'}$ and $v_{B,F',q'}$ in the formulas above, and can immediate skip the generation of formulas that are trivially true (the large majority in case (a)).

▶ **Remark 24** (Complexity of the SAT encoding)**.** While the encoding is efficient for string rewriting systems, it suffers from an 'encoding explosion' for term rewriting systems containing symbols of higher arity. The problem arises from the SAT encoding of the recursive computation of the interpretation of terms (described in Remark 15). The computation of the interpretation of a term $f(t_1, \dots, t_n)$ containing m variables needs $O(|Q|^{m+n+1})$ clauses: m for the quantification over the variable assignments, n for the possible states of $t_1, \dots, t_n$ and 1 for the possible result states. To some extend, this problem can be overcome by 'uncurrying' the system, that is, for every symbol f of arity $n > 2$ we introduce fresh symbols $f_1, \dots, f_{n-1}$ of arity 2 and then replace all occurrences of $f(t_1, \dots, t_n)$ by $f_{n-1}(\dots f_2(f_1(t_1, t_2), t_3) \dots, t_n)$. This transformation helps to bring the complexity down to $O(|Q|^{m+3})$. Nevertheless, for example for the S-rule, which only contains binary symbols, we still need $|Q|^6$ clauses. We note that after the uncurrying transformation, an automaton with more states may be needed to generate 'the same' language.

4.2 Disproving Weak Normalization

We are now ready to use Theorem 3 in combination with tree automata for automatically disproving weak normalization. The language L in the theorem is described by a non-deterministic tree automaton. In the previous section, we have seen how the relevant properties of tree automata can be checked. Here, we summarize the procedure:

▶ **Technique 25.** Let R be a left-linear TRS. We search for a tree automaton $A = \langle Q, \Sigma, F, \delta \rangle$ such that $\mathcal{L}(A)$ fulfills the properties of Theorem 3:

(i) We guarantee $\mathcal{L}(A) \cap \mathsf{NF}(\to) = \varnothing$ by the following steps:
- We employ Lemma 18 to construct a deterministic, complete automaton $B = \langle Q, \Sigma, F, \delta \rangle$ that accepts the set of terms containing redex occurrences with respect to R.
- Then the automaton $\overline{B} = \langle Q, Q \setminus \Sigma, F, \delta \rangle$ accepts all ground normal forms.
- We use Lemma 21 to check that $\mathcal{L}(A) \cap \mathcal{L}(\overline{B}) = \varnothing$ (thus $\mathcal{L}(A) \subseteq \mathcal{L}(B)$).

(ii) We guarantee that $\mathcal{L}(A)$ is closed under $\to$ by Lemma 14.

(iii) We use Lemma 12 to ensure that $\mathcal{L}(A) \neq \varnothing$.

These conditions can be encoded as satisfiability problems as described in Remarks 8, 11, 23, 13 and 15. This enables us to utilize SAT solvers to search for suitable automata A.

▶ **Remark 26.** The technique 25 can be slightly strengthened by first eliminating collapsing rules, that is, rules of the form $\ell \to x$ with $x \in \mathcal{X}$. Assume that the TRS R contains a collapsing rule $\ell \to x$. For every $f \in \Sigma$ we define the substitution $\sigma_f : \mathcal{X} \to T(\Sigma, \mathcal{X})$ by $\sigma_f(x) = f(x_1, \dots, x_{\#(f)})$ for fresh variables $x_1, \dots, x_{\#(f)}$ and $\sigma_f(y) = y$ for all $y \neq x$. We define $R' = (R \setminus \{\ell \to x\}) \cup \{\ell\sigma_f \to x\sigma_f \mid f \in \Sigma\}$. Then $\to_R$ and $\to_{R'}$ coincide on ground terms and hence R is (weakly) ground normalizing if and only if R is.

▶ **Remark 27.** We note the combination of Technique 25 with Remark 26 is complete with respect to disproving weak normalization on regular languages: if there exists a regular language L fulfilling the conditions of Theorem 3, then weak normalization can be disproved using Technique 25 after eliminating collapsing rules as in Remark 26.

This can be seen as follows. In the work [8, 9] a generalized method for ensuring closure of the language of automata under rewriting has been proposed. Thereby the condition $\ell\alpha \rightsquigarrow_A^* q \implies r\alpha \rightsquigarrow_A^* q$ of Lemma 14 is weakened to

$$\ell\alpha \rightsquigarrow_A^* q \implies r\alpha \rightsquigarrow_A^* p \quad \text{for some } p \geq q\,. \tag{1}$$

Here $\leq$ is a quasi-order on the states Q and the automaton must be monotonic with respect to this order, see Definition 34. The monotonicity guarantees that the language of the automaton is closed under rewriting.

In [23] it has been shown that this monotonicity property is strong enough to characterize and decide the closure of the regular languages under rewriting. In particular, the language of a deterministic tree automaton is closed under rewriting if and only if there exists such a monotonic quasi-order on the states.

Let R be a TRS such that there exists a regular language that satisfies the conditions of Theorem 3. Then there exists a deterministic, complete automaton A accepting this language and a quasi-order $\leq$ on the states satisfying (1) and monotonicity. Let R' be obtained from R by eliminating collapsing rules as described in Remark 26. We obtain a non-deterministic automaton A' that fulfils the requirements of Technique 25 for R' by closing the transition relation of A under $\leq$: we add $f(q_1,\dots,q_n) \rightsquigarrow q$ whenever $q \leq p$ and $f(q_1,\dots,q_n) \rightsquigarrow p$. As a consequence of monotonicity and using induction over the term structure, we obtain for all terms $t \in T(\Sigma,\mathcal{X})$ with $t \notin \mathcal{X}$ and $\alpha : \mathcal{X} \to Q$ that

($\star$) $t \rightsquigarrow_{A'}^* q$ if and only if $t \rightsquigarrow_A^* p$ for some p with $q \leq p$.

As a consequence of ($\star$) and monotonicity we have $\mathcal{L}(A') = \mathcal{L}(A)$ (roughly speaking, if $q \leq p$, then q accepts a subsets of the language of p). Thus $\mathcal{L}(A') \cap \mathsf{NF}(\to) = \varnothing$ and $\mathcal{L}(A') \neq \varnothing$ are guaranteed. Finally, we show that Lemma 14 is applicable for R' and A'. Let $\ell \to r \in R'$, $\alpha : \mathcal{X} \to Q$ and $q \in Q$ such that $\ell\alpha \rightsquigarrow_{A'}^* q$. Then by ($\star$) we get $\ell\alpha \rightsquigarrow_A^* q'$ for some $q' \in Q$ with $q \leq q'$. By (1) we have that $r\alpha \rightsquigarrow_A^* q''$ for some $q'' \in Q$ with $q' \leq q''$. Again by ($\star$) we obtain that $r\alpha \rightsquigarrow_A^* q$. Hence the conditions of Technique 25 are fulfilled for R' and A'.

▶ **Example 28.** We consider the following string rewriting system:

$$aL \to La \qquad Ra \to aR \qquad bL \to bR \qquad Rb \to Lab$$

This rewrite system is neither strongly nor weakly normalizing, but does not admit looping reductions, that is, reductions of the form $s \to^+ \ell sr$. An example of an infinite reduction is:

$$bLb \to bRb \to bLab \to bRab \to baRb \to baLab \to bLaab \to bRaab \to \cdots$$

It is easy to check that the automaton A_{LR} from Example 6 fulfills the requirements of Technique 25. Hence, the system is not weakly normalizing.

▶ **Example 29.** We consider the S-rule from combinatory logic:

$$a(a(a(S,x),y),z) \to a(a(x,z),a(y,z))$$

For the S-rule it is known that there are no reductions $t \to^* C[t]$ for ground terms t, see [31]. For open terms t the existence of reductions $t \to^* C[t\sigma]$ is open.

It is straightforward to verify that the automaton A_S from Example 7 fulfills the requirements of Technique 25, and hence the S-rule, and in particular the term $SSS(SSS)(SSS(SSS))$, are not weakly normalizing.

▶ **Example 30.** The δ-rule (known as Owl in Combinatory Logic) is even simpler:

$$\delta xy \to y(xy), \qquad \text{or equivalently} \qquad a(a(\delta, x), y) \to a(y, a(x, y)) \ .$$

As shown in [4], this rule does not admit loops, , and the techniques in [7] fail for this system. The Technique 25 can be applied to automatically disprove weak-normalization for this rule. Our tool finds a tree automaton that has 3 states and accepts the language of all ground terms with two occurrences of $\delta\delta$. In fact, this is precisely the language of non-terminating ground δ-terms, see further [4].

In all examples until now infinite reductions exist of the regular shape based on $t\sigma^n\tau$ rewriting to a term having $t\sigma^{f(n)}\tau\mu$ as a sub-term, for every n, for some term t and substitutions σ, τ, μ and an ascending linear function f. For instance, the S rule (Example 29) admits an infinite reduction implied by $t\sigma^n\tau$ rewriting to a super-term of $t\sigma^{n+1}\tau$, for $t = a(x, x)$, $\sigma(x) = Ax$, $\tau(x) = SA(SAA)$, for $A = SSS$.

▶ **Example 31.** The following example does not have an infinite reduction of this regular shape, neither of the more general patterns from [29] and [7].

$$aL \to La \qquad Raa \to aaaR \qquad bL \to bRa \qquad Rb \to Lb \qquad Rab \to Lab.$$

In this system bRa^nb rewrites to $bRa^{f(n)}b$ for f defined by $f(2n) = 3n + 1$ and $f(2n + 1) = 3n + 2$ for all n. This obviously yields an infinite reduction, but f is not linear, by which this example is outside the scope of [29] and [7]. In our approach a proof of non-termination and even non-weak-normalization is extremely simple: $ba^*(L \mid R)a^*b$ is non-empty, closed under rewriting and does not contain normal forms.

4.3 Disproving Strong Normalization

For disproving strong normalization based on Theorem 2, the only difference with Technique 25 is that checking that L is closed under $\to$ by Lemma 14 has to be replaced by checking that L is weakly closed under $\to$ by Lemma 16. The technique is applicable to string and term rewriting systems, and can be automated as described in Technique 25 and Remark 17.

▶ **Example 32.** Let us consider the rewrite system

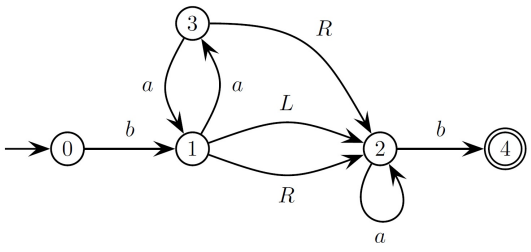
$$aaL \to Laa \qquad Ra \to aR \qquad bL \to bR \qquad Rb \to Lab \qquad Rb \to aLb$$

This system is non-looping and non-terminating. However, in contrast to Example 28, this system is weakly normalizing, since by always choosing the fourth rule the last rule is never used, and the first four rules are terminating. Hence the Technique 25 is not applicable for this TRS. However, the following pattern extends to an infinite reduction

$$bRa^{2n}b \to^{2n} ba^{2n}Rb \to ba^{2n}Lab \to^{n} bLa^{2n+1}b \to$$

$$bRa^{2n+1}b \to^{2n+1} ba^{2n+1}Rb \to ba^{2n+2}Lb \to^{n+1} bLa^{2n+2}b \to bRa^{2n+2}b.$$

Instead of finding this pattern explicitly, non-termination is also concluded from checking that $b(aa)^*(L \mid R \mid aR)a^*b$ describes a language satisfying all conditions from Theorem 2. A corresponding automaton is given on the right.



The conditions of Theorem 2 are now checked as follows. Non-emptiness follows from the existence of a path from state 0 to state 4. Every path from 0 to 4 either contains one of the patterns aaL, Ra, bL or Rb, so it remains to show weakly closedness under rewriting by Lemma 16. In the setting of string automata this means that for every left hand side ℓ and every ℓ-path from a state p to a state q we should find a u-path from p to q for a string u such that ℓ rewrites to u in one or more steps. For $\ell = aaL$ the only path is from 1 to 2, for which there is also an Laa path. For $\ell = Ra$ there is a path from 1 to 2, for which there is also an aR path via 3. The only other option for $\ell = Ra$ is a path from 3 to 2, for which there is also an aR path via 1. For $\ell = bL$ the only path is from 0 to 2, for which there is also a bR path. Finally, for $\ell = Rb$ there is a path from 1 to 4, for which there is also an Lab path and a path from 3 to 4, for which there is also an aLb path, by which all conditions have been verified. Note that for the Rb-path from 1 to 4 it is essential to use the 4th rule, while for the Rb-path from 3 to 4 it is essential to use the last rule.

This example can also be treated by the technique introduced in the following section.

5 Improved Methods for Disproving Strong Normalization

In this section, we improve the method for proving non-termination. The methods introduced so far are not able to handle the following example.

▶ **Example 33.** We consider the following string rewriting system:

$$zL \to Lz \qquad Rz \to zR \qquad zLL \to zLR \qquad RRz \to LzRz$$

This rewrite system is weakly normalizing but not strongly normalizing. The non-termination criteria introduced in the previous sections are not applicable for this system. Let us consider the first steps of an infinite reduction:

$$\begin{aligned} &\underline{zLL}zzRz \\ \to\ &zLRzzRz \to zLzRzRz \to zLzzRRz \\ \to\ &zLzzLzRz \to zLzLzzRz \to \underline{zLL}zzzRz \\ \to\ &\dots \end{aligned}$$

Note the underlined occurrences of zLL. Due to the rule $zLL \to zLR$, the word zL is the marker for 'turning' on the left; However, this marker zL is itself a redex. To obtain an infinite reduction, this marker must not be reduced.

The idea for proving non-termination of systems like Example 33 is to let the automaton determine which redex to contract. To this end, we introduce a 'redex selection' function

$$\xi : Q \to \mathcal{P}(R)$$

that maps states of the automaton to sets of rules that may be contracted at the corresponding position in the term. The idea is that a redex $\ell\sigma$ in a term $C[\ell\sigma]$ with respect to a rule $\ell \to r$ is allowed to be contracted if $\ell\sigma \rightsquigarrow^* q$ with $\ell \to r \in \xi(q)$. In this way, the automaton determines what redexes are to be contracted. Then the automaton only needs to fulfill the property $\ell\alpha \rightsquigarrow_A^* q \implies r\alpha \rightsquigarrow_A^* q$ for the *selected rules*:

$$\ell \to r \in \xi(q) \ \wedge\ \ell\alpha \rightsquigarrow_A^* q \quad \implies \quad r\alpha \rightsquigarrow_A^* q$$

for every rule $\ell \to r \in R$, state $q \in Q$ and $\alpha : \mathcal{X} \to Q$. Moreover, as proposed in [8, 9, 23], we weaken the requirement $r\alpha \rightsquigarrow_A^* q$ to $r\alpha \rightsquigarrow_A^* p$ for some $p \geq q$. Here $\leq$ is a quasi-order on the states and the automaton must be monotonic with respect to this order (see Definition 34). The monotonicity guarantees that the language of the automaton is closed under rewriting. For the present paper, this closure property holds only for the rules selected by ξ.

▶ **Definition 34** (Monotonicity)**.** A tree automaton $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ is *monotonic* with respect to a quasi-order $\leq$ on the states Q if the following properties hold:

(i) For all $f \in \Sigma$ with arity n and states $a_1 \leq b_1$, $a_2 \leq b_2$, …, $a_n \leq b_n$, it holds

$$f(a_1, \ldots, a_n) \rightsquigarrow_A q \quad \Longrightarrow \quad f(b_1, \ldots, b_n) \rightsquigarrow_A p \text{ for some } p \in Q \text{ with } q \leq p$$

(ii) Whenever $q \in F$ and $q \leq p$, then $p \in F$.

The following lemma is immediate by induction on the size of the context.

▶ **Lemma 35.** *Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ be a tree automaton that is monotonic with respect to a quasi-order $\leq$ on the states Q. Let $a, b \in Q$ with $a \leq b$. Then for all ground contexts C we have that $C[a] \rightsquigarrow a'$ with $a' \in Q$ implies that $C[b] \rightsquigarrow b'$ for some $b' \in Q$ with $a' \leq b'$.*

▶ **Definition 36** (Runs)**.** Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ be a tree automaton and $t \in T(\Sigma, \varnothing)$. A *run* of A on t is a function $\rho : Pos(t) \to Q$ such that for every $p \in Pos(t)$ and $t(p) = f \in \Sigma$ there is a rule $f(\rho(p1), \ldots, \rho(pn)) \rightsquigarrow \rho(p)$ in δ. The run ρ is accepting if $\rho(\varepsilon) \in F$.

Note that there is a direct correspondence between runs on t and rewrite sequences $t \rightsquigarrow^* q$. We are now ready to state the generalized theorem for disproving strong normalization.

▶ **Theorem 37.** *Let R be a left-linear TRS. Let $A = \langle\, Q, \Sigma, F, \delta \,\rangle$ be a tree automaton with $\mathcal{L}(A) \neq \varnothing$, $\leq$ a quasi-order on the states Q, and $\xi : Q \to \mathcal{P}(R)$ a function, called* redex selection function*. Assume that the following properties hold:*

(a) *The automaton A is monotonic with respect to $\leq$.*

(b) *For every state $q \in Q$, rule $\ell \to r \in \xi(q)$ and $\alpha : \mathcal{X} \to Q$ it holds that:*

$$\begin{aligned} \ell\alpha \rightsquigarrow_A^* q \quad \Longrightarrow \quad & (\exists p \in Q.\ q \leq p \ \wedge\ r\alpha \rightsquigarrow_A^* p) \vee \\ & (\exists r' \trianglelefteq r.\ \exists q' \in F.\ r'\alpha \rightsquigarrow_A^* q') \end{aligned}$$

(c) *For every term $t \in T(\Sigma, \varnothing)$ and accepting run ρ on t there is a position p such that $t|_p$ is an instance of the left-hand side of a rule $\ell \to r \in \xi(\rho(p))$.*

Then R is not strongly normalizing.

Proof. Assume that the conditions of the theorem are fulfilled. To disprove strong normalization of $\to$ it suffices to disprove strong normalization of $\to \circ \trianglerighteq$ where $\trianglerighteq$ is the (non-strict) sub-term relation. We show that $\mathcal{L}(A)$ and $\to \circ \trianglerighteq$ fulfill the requirements of Theorem 2. Let $t \in \mathcal{L}(A)$. Then there exists an accepting run ρ of A on t. By item 3 there exists a position $p \in Pos(t)$ and a rule $\ell \to r \in \xi(\rho(p))$ such that $t|_p$ is an instance of ℓ. Then $t|_p = \ell\sigma$ for some substitution σ. By left-linearity, we can define $\alpha : Var(\ell) \to Q$ by $\alpha(x) = \rho(pp')$ whenever $\ell|_{p'} \in \mathcal{X}$. Then $\ell\alpha \rightsquigarrow^* \rho(p)$ and we distinguish cases according to item 2:

1. There exists $q \in Q$ with $\rho(p) \leq q$ and $r\alpha \rightsquigarrow^* q$. We know that $t[\ell\sigma]_p = t \rightsquigarrow^* \rho(\varepsilon)$ and define $t' = t[r\sigma]_p$. Note that $t \to t'$ and $t \to \circ \trianglerighteq t'$. We have $t \rightsquigarrow t[\rho(p)] \rightsquigarrow \rho(\varepsilon)$ and $\rho(p) \leq q$. By Lemma 35 we have $t[q] \rightsquigarrow q'$ for some $q' \geq \rho(\varepsilon)$ and by monotonicity $q' \in F$. Hence $t' = t[r\sigma]_p \rightsquigarrow t[q] \rightsquigarrow^* q'$. Thus $t' \in \mathcal{L}(A)$ and $t \to \circ \trianglerighteq t'$.
2. There exist $r' \trianglelefteq r$ and $q \in F$ such that $r'\alpha \rightsquigarrow^* q$. Then $r'\sigma \rightsquigarrow^* q$ and hence $r'\sigma \in \mathcal{L}(A)$. Moreover, $t \to \circ \trianglerighteq r'\sigma$.

This shows that $\mathcal{L}(A)$ contains no normal forms and is weakly closed under $\to \circ \trianglerighteq$. By Theorem 2, $\to \circ \trianglerighteq$ is not strongly normalizing and hence $\to$ is not strongly normalizing. ◀

▶ **Remark 38** (SAT encoding of the conditions of Theorem 37)**.** The encoding of condition 1 of Theorem 37 is straightforward, and condition 2 is an easy extension of Remark 15. The requirement 3 can be encoded similar to Remark 23, as follows. Let $\ell_1, \ldots, \ell_n$ be the left-hand sides of rules in R. Let B be the automaton and $F_{\ell_1}, \ldots, F_{\ell_n}$ the sets of states obtained from Lemma 18. We construct the product automaton $A \times B$, and then we compute those states that are reachable without passing states (q, q') for which there exists $\ell \to r \in \xi(q)$ such that $q' \in F_\ell$ (that is, the rule $\ell \to r$ is selected by A and B confirms that the term is an instance of ℓ).

6 Experimental Results

We have implemented the improved method for disproving strong normalization (Theorem 37) presented in this paper. For the purpose of evaluating our techniques, the tool applies only the methods presented in this paper, and no other non-termination method like loop checks. The SAT solver employed for the evaluation results in this section is MiniSat [6]. Our tool can be downloaded from `http://joerg.endrullis.de/non-termination/`.

Our tool can automatically prove non-termination of all examples in this paper, including the S-rule and the δ-rule. The following table shows the size of the automata that are found by the tool as witnesses for non-termination for the examples in our paper:

Example	28	29	30	31	32	33
Number of states	4	5	3	4	5	6

Each of these automata has been found within less than a second on a dual core laptop.

We have also evaluated our methods on the database used in [7], consisting of 58 non-terminating term rewriting systems that do not admit loops. The tool AProVE recognizes 44 systems as non-terminating; an impressive 76%. An extension of AProVE with our method would increase the recognition by 8.5% to 84.5%. In other words, our method succeeds on 36% (that is 5 systems) of the remaining 14 systems for which AProVE did not find a proof. These 5 systems are:

- `nonloop/TRS/emmes/ex3_4.trs`
- `nonloop/TRS/own/challenge_fab.trs`
- `nonloop/TRS/own/downfrom.trs`
- `nonloop/TRS/own/ex_payet.trs`
- `nonloop/TRS/own/isList-List.trs`

In total, our tool succeeds for 26 of the 58 non-looping examples from [7]. The results suggest that our method and that of [7] are complementary and should be combined for maximum strength. The paper [7] explicitly mentions that the following example is beyond their techniques (this example is not part of the database above):

$$
\begin{aligned}
f(\mathrm{true}, \mathrm{true}, x, \mathrm{s}(y)) &\to f(\mathit{isNat}(x), \mathit{isNat}(y), \mathrm{s}(x), \mathit{double}(\mathrm{s}(y))) \\
\mathit{isNat}(0) &\to \mathrm{true} \\
\mathit{isNat}(\mathrm{s}(x)) &\to \mathit{isNat}(x) \\
\mathit{double}(0) &\to 0 \\
\mathit{double}(\mathrm{s}(x)) &\to \mathrm{s}(\mathrm{s}(\mathit{double}(x)))
\end{aligned}
$$

Our non-termination techniques can handle this system: the tool finds an automaton with 6 states within 3 seconds (using the transformation from Remark 24).

Finally, we have evaluated the tool on the termination problem database (TPDB). We have run our tool on all string and term rewriting systems (of the standard categories) that remained unsolved during the last full run of all tools in December 2013. For string rewriting, our tool was able to disprove termination for 13, and for term rewriting, for 8 systems of the unsolved systems. This corresponds to an increase of strength of 11.5% (114 + 13) for string rewriting and of 3% (274 + 8) for term rewriting. Let us mention that many of the 13 string rewriting systems actually admit loops, but very complicated ones, that are not found by the standard tools. These loops have been found in previous competitions by the tools Matchbox [32] and Knocked for Loops [36].

7 Conclusions and Future Work

In this paper, we have employed regular languages for proving non-termination. Instead of searching for an infinite reduction explicitly we search for a regular language with properties from which non-termination easily follows. After encoding these properties in a propositional formula, the actual search is done by a SAT solver. In some examples, like Example 31, a very simple corresponding regular language is quickly found by our approach, while the actual infinite reductions have a non-linear pattern being beyond earlier approaches.

For future work, it is interesting to investigate whether this approach can be extended to context-free (tree) languages; such an approach could potentially also generalize [7]. The question is whether there are efficient criteria to check the conditions of Theorem 2. For example, consider the following string rewriting system:

$$\begin{array}{lllll} & bB \to Bb & bcd \to BcD & Dd \to dD & \\ aX \to abb & BX \to Xb & bcd \to XcY & YD \to dY & Ye \to dde \end{array}$$

This system admits for every $n > 1$ reductions of the form

$$a\,b^n\,c\,d^n\,e \to^* a\,B^{n-1}\,bcd\,D^{n-1}\,e \to^* a\,B^{n-1}\,XcY\,D^{n-1}\,e \to^* a\,b^{n+1}\,c\,d^{n+1}\,e$$

As a description of this pattern needs a context-free language, it is unlikely that a regular language exists that fulfills the requirements of Theorem 2.

As described in Remark 24, the SAT encoding of (non-deterministic) automata is not efficient for symbols of higher arity. We think that these problems can be overcome by more efficient encodings of automata. For example, the uncurrying transformation mentioned in Remark 24 can be seen as a restriction of the shape of the automata (the transition is computed argument by argument) instead of a transformation on the system. It would be interesting to investigate what other restrictions would lead to a more efficient representation of automata as Boolean satisfiability problems. Results in this direction can be of interest in various areas where automata are applied.

We think that it is also interesting to investigate whether the characterization of strong and weak normalization (Theorems 2 and 3) can be adapted to the setting of infinitary rewriting [20, 22, 14] with infinite terms and ordinal-length reductions; the interesting properties then are infinitary strong and weak normalization.

Finally, we note that equality of streams [17, 18, 37, 19] (infinite sequences of symbols) can be rendered as a non-termination problem (a comparison program running indefinitely if the streams are equal, and terminating as soon as a difference is found). It remains to be investigated whether non-termination techniques can be employed fruitfully for proving stream equality.

References

1 Homepage of AProVE, 2015. http://aprove.informatik.rwth-aachen.de.
2 Homepage of TTT2, 2015. http://cl-informatik.uibk.ac.at/software/ttt2/.
3 H.P. Barendregt, J. Endrullis, J.W. Klop, and J. Waldmann. Dance of the Starlings. 2015. To be published on arXiv.org.
4 H.P. Barendregt, J. Endrullis, J.W. Klop, and J. Waldmann. Songs of the Starling and the Owl. 2015. To be published on arXiv.org.
5 B. Cook. Priciples of program termination. http://research.microsoft.com/en-us/um/cambridge/projects/terminator/principles.pdf.
6 N. Eén and A. Biere. Effective Preprocessing in SAT through Variable and Clause Elimination. In *Proc. Conf. on Theory and Applications of Satisfiability Testing (SAT '05)*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.
7 F. Emmes, T. Enger, and J. Giesl. Proving Non-looping Non-termination Automatically. In *International Joint Conference on Automated Reasoning (IJCAR 2012)*, volume 7364 of *LNCS*, pages 225–240. Springer, 2012.
8 J. Endrullis, R. C. de Vrijer, and J. Waldmann. Local Termination. In *Proc. Conf. on Rewriting Techniques and Applications (RTA)*, volume 5595 of *LNCS*, pages 270–284. Springer, 2009.
9 J. Endrullis, R.C. de Vrijer, and J. Waldmann. Local Termination: Theory and Practice. *Logical Methods in Computer Science*, 6(3), 2010.
10 J. Endrullis, H. Geuvers, J. G. Simonsen, and H. Zantema. Levels of Undecidability in Rewriting. *Information and Computation*, 209(2):227–245, 2011.
11 J. Endrullis, H. Geuvers, and H. Zantema. Degrees of Undecidability in Term Rewriting. In *Proc. Int. Workshop on Computer Science Logic (CSL 2009)*, volume 5771 of *LNCS*, pages 255–270. Springer, 2009.
12 J. Endrullis, C. Grabmayer, D. Hendriks, J.W. Klop, and R.C. de Vrijer. Proving Infinitary Normalization. In *Postproc. Int. Workshop on Types for Proofs and Programs (TYPES 2008)*, volume 5497 of *LNCS*, pages 64–82. Springer, 2009.
13 J. Endrullis, C. Grabmayer, J.W. Klop, and V. van Oostrom. On Equal μ-Terms. *Theoretical Computer Science*, 412(28):3175–3202, 2011.
14 J. Endrullis, H. Hvid Hansen, D. Hendriks, A. Polonsky, and A. Silva. A Coinductive Framework for Infinitary Rewriting and Equational Reasoning. In *Proc. Conf. on Rewriting Techniques and Applications (RTA 2015)*, Leibniz International Proceedings in Informatics. Schloss Dagstuhl, 2015.
15 J. Endrullis and D. Hendriks. Transforming Outermost into Context-Sensitive Rewriting. *Logical Methods in Computer Science*, 6(2), 2010.
16 J. Endrullis and D. Hendriks. Lazy Productivity via Termination. *Theoretical Computer Science*, 412(28):3203–3225, 2011.
17 J. Endrullis, D. Hendriks, and R. Bakhshi. On the Complexity of Equivalence of Specifications of Infinite Objects. In *ACM SIGPLAN International Conference on Functional Programming (ICFP 2012)*, pages 153–164. ACM, 2012.
18 J. Endrullis, D. Hendriks, R. Bakhshi, and G. Roşu. On the complexity of stream equality. *Journal of Functional Programming*, 24(2–3):166–217, 2014.
19 J. Endrullis, D. Hendriks, and M. Bodin. Circular Coinduction in Coq Using Bisimulation-Up-To Techniques. In *Proc. Conf. on Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 354–369. Springer, 2013.
20 J. Endrullis, D. Hendriks, and J.W. Klop. Highlights in Infinitary Rewriting and Lambda Calculus. *Theoretical Computer Science*, 464:48–71, 2012.
21 J. Endrullis, D. Hofbauer, and J. Waldmann. Decomposing Terminating Rewrite Relations. In *Proc. Workshop on Termination (WST '06)*, pages 39–43, 2006.

22 J. Endrullis and A. Polonsky. Infinitary Rewriting Coinductively. In *Proc. Types for Proofs and Programs (TYPES 2012)*, volume 19 of *Leibniz International Proceedings in Informatics*, pages 16–27. Schloss Dagstuhl, 2013.

23 B. Felgenhauer and R. Thiemann. Reachability Analysis with State-Compatible Automata. In *LATA*, volume 8370 of *LNCS*, pages 347–359. Springer, 2014.

24 T. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In *Proc. Conf. on Rewriting Techniques and Applications (RTA '98)*, volume 1379 of *LNCS*, pages 151–165. Springer, 1998.

25 A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Information and Computation*, 205(4):512–534, 2007.

26 A. Geser and H. Zantema. Non-looping String Rewriting. *RAIRO Theoretical Informatics and Applications*, 33(3):279–302, 1999.

27 M. Korp and A. Middeldorp. Match-bounds revisited. *Information and Computation*, 207(11):1259–1283, 2009.

28 M. Mousazadeh, B. T. Ladani, and H. Zantema. Liveness verification in trss using tree automata and termination analysis. *Computing and Informatics*, 29(3):407–426, 2010.

29 M. Oppelt. Automatische Erkennung von Ableitungsmustern in nichtterminierenden Wortersetzungssystemen. Technical report, HTWK Leipzig, Germany, 2008. Diploma Thesis.

30 Helmut Seidl. Deciding equivalence of finite tree automata. In *STACS 89*, volume 349 of *LNCS*, pages 480–492. Springer, 1989.

31 J. Waldmann. The Combinator S. *Information and Computation*, 159(1–2):2–21, 2000.

32 J. Waldmann. Matchbox: A Tool for Match-Bounded String Rewriting. In *Proc. Conf. on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *LNCS*, pages 85–94. Springer, 2004.

33 J. Waldmann. Compressed loops (draft), 2012.

34 H. Zankl and A. Middeldorp. Nontermination of String Rewriting using SAT, 2007.

35 H. Zankl, C. Sternagel, D. Hofbauer, and A. Middeldorp. Finding and certifying loops. In *Proc. Conf. on Theory and Practice of Computer Science (SOFSEM 2010)*, volume 5901 of *LNCS*, pages 755–766. Springer, 2010.

36 H. Zankl, C. Sternagel, D. Hofbauer, and A. Middeldorp. Finding and certifying loops. In *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901 of *LNCS*, pages 755–766. Springer, 2010.

37 H. Zantema and J. Endrullis. Proving Equality of Streams Automatically. In *Proc. Conf. on Rewriting Techniques and Applications (RTA 2011)*, pages 393–408, 2011.