Finding Finite Automata That Certify Termination of String Rewriting

Alfons Geser[1,⋆], Dieter Hofbauer[2], Johannes Waldmann[3], and Hans Zantema[4]

[1] National Institute of Aerospace, 144 Research Drive,
Hampton, Virginia 23666, USA
geser@nianet.org
[2] Mühlengasse 16, D-34125 Kassel, Germany
dieter@theory.informatik.uni-kassel.de
[3] Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig,
Fb IMN, PF 30 11 66, D-04251 Leipzig, Germany
waldmann@imn.htwk-leipzig.de
[4] Department of Computer Science, Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.zantema@tue.nl

Abstract. We present a technique based on the construction of finite automata to prove termination of string rewriting systems. Using this technique the tools `Matchbox` and `TORPA` are able to prove termination of particular string rewriting systems completely automatically for which termination was considered to be very hard until recently.

1 Introduction

Consider a finite string over $\{a, b\}$ and only one rule: if $aabb$ occurs in the string than it may be replaced by $bbbaaa$. The goal is to prove *termination*: prove that application of this rule cannot go on forever. This is a surprisingly hard problem for which only ad hoc proofs were available until recently [11, 13]. A set of such string replacement rules is called a string rewriting system (SRS) or semi-Thue system. In this paper we describe a technique based on the construction of finite automata by which termination of such SRSs including this example $\{aabb \to bbbaaa\}$ can be proved fully automatically.

It is widely accepted that being able to prove termination of programs is highly desirable. String rewriting is one of the simplest paradigms having full computational power, and is extensively studied, e.g., in Book and Otto [3]. For instance, Turing machine computation is easily seen to be a special case of string rewriting. Therefore it is natural to consider techniques for automatically proving termination of SRSs. On the other hand string rewriting can be seen as a

⋆ Partly supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while this author was in residence at the NIA.

M. Domaratzki et al. (Eds.): CIAA 2004, LNCS 3317, pp. 134–145, 2005.
© Springer-Verlag Berlin Heidelberg 2005

special case of term rewriting, for which a wide range of termination techniques has been developed; for a recent overview see Zantema [14].

In order to prove termination of an SRS R, we construct an infinite SRS $\mathsf{match}(R)$, obtained from R by labelling the symbols by natural numbers. By construction, on the one hand, $\mathsf{match}(R)$-steps simulate R-steps, and on the other hand, every finite subsystem of $\mathsf{match}(R)$ terminates. Now we construct a finite automaton such that an accepting computation of s is transformed into an accepting computation of t whenever s rewrites to t by a $\mathsf{match}(R)$-step. This closure property entails that the simulation of an R-derivation involves only a finite subsystem of $\mathsf{match}(R)$. Termination of R follows. Since in this way a bound on the labels occurring in derivations is established, this is called the *match-bound* approach.

The inspiration of match-bounds was taken from Ravikumar's *change-bounds*. Ravikumar [10] proposes an infinite SRS similar to $\mathsf{match}(R)$, and shows that change-bounded length-preserving SRSs preserve regular languages. It is easy to see that change-bounds imply match-bounds, while also the converse can be proved to hold. However, in contrast to change-bounds, match-bounds work also for SRSs that do not preserve lengths.

An earlier version of the match-bound approach was presented before [5, 6]. Here we describe the basic approach in a more general setting. The reason for doing this is twofold: the presentation is more modular and therefore hopefully simpler, and this generalization can be used for variants and extensions of the method. Indeed in this new setting we are able to give short proofs of the main theorems.

Our main new contribution is to describe new algorithms to construct appropriate automata. In the earlier approach [5, 6] some of us described how a suitable rewriting closure of a language may effectively preserve regularity. The main algorithm then consisted of constructing an automaton that accepts exactly the desired rewriting closure. A drawback of this approach is that even for very small SRSs like $aabb \to bbbaaa$ intermediate automata with thousands of states are constructed, while the final automaton for this example consists only of 42 states.

In the new approach, the constructed automaton need no longer be *exact*: it may accept any superset of the desired rewriting closure. Therefore the new approach is called *approximate*. Like the exact approach, the approximate approach is correct: the constructed automaton certifies termination of the SRS. In contrast to the exact approach, it may fail. However, we have observed that the approximate approach often succeeds, and even yields the same automaton as the exact approach. The approximate approach is usually much more economical: all intermediate automata are no bigger than the final one. This efficiency allows one to solve examples that the exact approach, in spite of its completeness, could not handle within reasonable time and memory.

The match-bound technique is one of the few techniques that are able to prove termination on a given language: no infinite rewrite sequence exists starting with some string in the language. Most other techniques only prove (uniform) termi-

nation: no infinite rewrite sequence exists at all. Therefore it profits from the theory of *forward closures*, according to which full termination can be concluded from termination on a particular language.

Versions of the approximate approach have been implemented in three tools: `Matchbox`, `TORPA`, and `AProVE`. The tool `Matchbox` [12] was the first tool that implemented the match-bound approach, and it offers a variety of match-bound related computations. The tool `TORPA` [15] is a tool for proving termination of string rewriting by various techniques: polynomial interpretations, recursive path order, semantic labelling, dependency pairs and finally one particular version of match-bounds for forward closures. The tool `AProVE` [7] is a tool for proving termination of term rewriting mainly by dependency pairs, but also covering various other techniques. In the most recent version the approach using match-bounds for forward closures was copied from `TORPA`. In the category "string rewriting" in the termination competition of the 7th International Workshop on Termination in 2004 these three tools ranked third, first, and second, respectively.

The paper is organized as follows. In Section 2 the basic theory is presented, including preliminaries and all proofs, except for the proofs of forward closure theory. Next in Section 3 the various ways of finding compatible automata are discussed: the exact approach and the approximate approach. For the latter, two variants are discussed: the one that has been implemented in `TORPA`, and the one that has been implemented in `Matchbox`.

2 Basic Theory

A string rewrite system (SRS) over an alphabet Σ is a set $R \subseteq \Sigma^* \times \Sigma^*$. Elements $(\ell, r) \in R$ are called *rules* and are written as $\ell \to r$; the string ℓ is called the left hand side (lhs) and r is called the right hand side (rhs) of the rule. A string $s \in \Sigma^*$ rewrites to a string $t \in \Sigma^*$ with respect to an SRS R, written as $s \to_R t$ if strings $u, v \in \Sigma^*$ and a rule $\ell \to r \in R$ exist such that $s = u\ell v$ and $t = urv$. The reflexive transitive closure of $\to_R$ is written as $\to_R^*$. In this paper we consider both finite and infinite SRSs over both finite and infinite alphabets, on the other hand all automata we consider are finite.

A sequence $t_1, t_2, t_3, \ldots$ is called an *R-derivation* if $t_i \to_R t_{i+1}$ for all $i = 1, 2, 3, \ldots$. An SRS R is called *terminating* on a language $L \subseteq \Sigma^*$ if no infinite R-derivation $t_1, t_2, t_3, \ldots$ exists such that $t_1 \in L$. An SRS R is called *terminating* if no infinite R-derivation exists at all, i.e., it is terminating on Σ^*. Any SRS having an empty lhs is non-terminating, hence we generally assume that each lhs is non-empty.

For a map $h : \Sigma' \to \Sigma$ we reuse the notation h for its morphism extension $h : \Sigma'^* \to \Sigma^*$ by $h(\epsilon) = \epsilon$ and $h(uv) = h(u)h(v)$, and to languages over Σ' by $h(L) = \{h(u) \mid u \in L\}$.

Let R be an SRS over an alphabet Σ, and let $L \subseteq \Sigma^*$. Let R' be an SRS over an alphabet Σ', and let $L' \subseteq \Sigma'^*$. The triple (Σ', R', L') is called an *enrichment* of (Σ, R, L) by $h : \Sigma' \to \Sigma$ if $L = h(L')$ and

$$h(\ell') = \ell \wedge (\ell \to r) \in R \Rightarrow \exists r' \in \Sigma'^*.\, (\ell' \to r') \in R' \wedge h(r') = r$$

for all $\ell' \in \Sigma'^*$ and $(\ell \to r) \in R$. From the enrichment property, it follows that

$$h(s') \to_R t \Rightarrow \exists t' \in \Sigma'^*.\, s' \to_{R'} t' \wedge t = h(t')$$

for all $s', t' \in \Sigma'^*$, and so if R' terminates on L' then R terminates on L.

For an SRS R over an alphabet Σ for which all lhs's are non-empty we define the infinite SRS $\mathsf{match}(R)$ over $\Sigma \times \mathbf{N}$ to consist of all rules $(a_1, n_1) \cdots (a_p, n_p) \to (b_1, m_1) \cdots (b_q, m_q)$ for which $a_1 \cdots a_p \to b_1 \cdots b_q \in R$ and $m_i = 1 + \min_{j=1,\ldots,p} n_i$ for all $i = 1, \ldots, q$. For instance, if R contains the rule $aa \to aba$, then $(a,3)(a,1) \to (a,2)(b,2)(a,2)$ is a rule of $\mathsf{match}(R)$.

We define $\mathsf{base} : \Sigma \times \mathbf{N} \to \Sigma$ by $\mathsf{base}((a,n)) = a$ for all $a \in \Sigma$, $n \in \mathbf{N}$, and $\mathsf{lift}_0 : \Sigma \to \Sigma \times \mathbf{N}$ by $\mathsf{lift}_0(a) = (a, 0)$ for all $a \in \Sigma$. By construction, $(\Sigma \times \mathbf{N}, \mathsf{match}(R), \mathsf{lift}_0(L))$ is an enrichment of (Σ, R, L) by base.

An SRS R' over Σ' is called *locally terminating* if for every finite alphabet $\Sigma'_0 \subseteq \Sigma'$ the SRS $R'_0 = \{\, \ell' \to r' \in R' \mid \ell', r' \in \Sigma'^*_0 \,\}$ is terminating.

Theorem 1. *Let R be any finite SRS over Σ. Then $\mathsf{match}(R)$ is locally terminating.*

Proof. Let $\Sigma'_0 \subseteq \Sigma' = \Sigma \times \mathbf{N}$ be finite; we have to prove that R'_0 as defined above for $R' = \mathsf{match}(R)$ is terminating. Let n be the maximum value for which there is $a \in \Sigma$ such that $(a, n) \in \Sigma'_0$. Assume that R'_0 admits an infinite derivation, then there is also an infinite R'_0-derivation in which all symbols (a, k) satisfy $k \leq n$. Let m be a number such that for every rhs of R the length is less than m. Now for a symbol (a, k) define its weight $W((a, k)) = m^{n-k}$, and for a string of symbols the weight is the sum of the weights of the symbols. For every rule $\ell' \to r'$ in R'_0 we have $r' = (b_1, k), (b_2, k), \ldots, (b_q, k)$ while ℓ' contains a symbol $(a, k-1)$. Hence

$$W(\ell') \geq W((a, k-1)) = m^{n-k+1} > q \cdot m^{n-k} = W(r').$$

Hence in every step of the infinite derivation the weight in $\mathbf{N}$ strictly decreases, contradiction. □

Finiteness of R is not essential for validity of Theorem 1: using multisets easily a proof can be given not requiring finiteness. However, we intend to use Theorem 1 only for finite systems and we will use the present proof using weights to conclude a result on derivational complexity.

All automata we consider in this paper are standard non-deterministic finite-state automata. For two states p, q in an automaton A over Σ and a string $u \in \Sigma^*$ we write $p \xrightarrow{u}_A q$ if there is a path from p to q in A in which the transitions are successively labelled by the symbols in u. More precisely, the transition relation $\xrightarrow{a}_A$ for $a \in \Sigma$ is extended to Σ^* by defining inductively $p \xrightarrow{au}_A q$ if a state r exists such that $p \xrightarrow{a}_A r$ and $r \xrightarrow{u}_A q$, and $p \xrightarrow{\epsilon}_A p$ for all states p. Let A, L and R be a finite automaton, a language and an SRS over an alphabet Σ, respectively. Then A is called *compatible* with L and R if

- $L \subseteq L(A)$, and
- A is closed under R, i.e., if $\ell \to r \in R$ and $p \xrightarrow{\ell}_A q$ for two states p, q of A, then also $p \xrightarrow{r}_A q$.

A direct consequence of this definition is that if A is compatible with L and R, then we have $\to_R^*(L) \subseteq L(A)$, where the set of R-successors $\to_R^*(L)$ is defined by

$$\to_R^*(L) \;=\; \{u \mid \exists t \in L :\; t \to_R^* u\}.$$

If Σ is finite then such a compatible automaton trivially exists: take the automaton for Σ^* consisting of one state, and a-transitions from that state to itself for every $a \in \Sigma$. We will focus on finding finite compatible automata for infinite SRSs over infinite alphabets, starting from a language described by a finite automaton.

Theorem 2. *Let (Σ', R', L') be an enrichment of (Σ, R, L) by h, let R' be locally terminating, and assume that L' and R' admit a compatible finite automaton. Then R terminates on $h(L')$.*

Proof. Suppose there is an infinite R-derivation $u_1 \to_R u_2 \to_R u_3 \to_R \cdots$ for which $u_1 \in h(L')$. Then there exists $v_1 \in L'$ such that $h(v_1) = u_1$. By repeated application of the definition of enrichment this gives rise to an infinite R'-derivation $v_1 \to_{R'} v_2 \to_{R'} v_3 \to_{R'} \cdots$ for which $h(v_i) = u_i$ for $i = 1, 2, 3, \ldots$. Let A be a finite automaton compatible with L' and R'; let Σ'_0 be the finite set of transition labels occurring in A. Since $L' \subseteq L(A)$ we have $p_0 \xrightarrow{v_1}_A p_f$ for the initial state p_0 and a final state p_f, since A is closed under R' we obtain by induction on i that $p_0 \xrightarrow{v_i}_A p_f$ for all $i = 1, 2, 3, \ldots$. Hence for every rule $\ell' \to r' \in R'$ that is applied in the infinite derivation $v_1 \to_{R'} v_2 \to_{R'} v_3 \to_{R'} \cdots$ there are states p, q in A satisfying $p \xrightarrow{\ell'}_A q$ and $p \xrightarrow{r'}_A q$. Hence in the infinite derivation only rules from R'_0 are applied, contradicting the assumption that R' is locally terminating. □

Theorem 2 will be used as follows. If termination of R on a language L over Σ has to be proved then we define $L_0 = \mathsf{lift}_0(L) = \{(s, 0) \mid s \in L\}$. By definition, $\mathsf{base}(L_0) = L$. Now according to Theorems 1 and 2 it suffices to find a compatible automaton for L_0 and $\mathsf{match}(R)$. Due to the form of the weights we used in the proof of Theorem 1 this does not only prove termination of R on L but even linear derivational complexity: there is a constant C such that every R-derivation starting in $s \in L$ has length at most $C \cdot |s|$.

To prove termination on Σ^* (usually simply called *termination*) we may apply this approach for $L = \Sigma^*$. However, by a result of Dershowitz [4] on forward closures, we may also choose another language for L that may be smaller. Thus we can prove termination for SRSs that do not satisfy linear derivational complexity, like $\{ab \to ba\}$ or $\{ab \to bba\}$.

We describe *forward closures* by rewriting using an extended SRS $R_\#$ [5, 6]; this way we characterize termination on Σ^* by termination on a small regular

set, which makes it amenable to automation. A self-contained presentation of this $R_\#$-approach including all proofs will appear in [16].

For an SRS R over an alphabet Σ we define the SRS $R_\#$ over $\Sigma \cup \{\#\}$ by

$$R_\# \;=\; R \;\cup\; \{\; \ell_1 \# \to r \mid \ell \to r \in R \;\wedge\; \ell = \ell_1\ell_2 \;\wedge\; \ell_1 \neq \epsilon \neq \ell_2 \;\}.$$

Write $\mathsf{rhs}(R)$ for the set of rhs's of R.

Theorem 3. *Let R be a finite SRS. Then R is terminating if and only if $R_\#$ is terminating on $\mathsf{rhs}(R) \cdot \{\#\}^*$.*

We omit the proof. Combining Theorems 1, 2, and 3 now for proving (uniform) termination of a finite SRS R it suffices to find a compatible automaton for $\mathsf{lift}_0(\mathsf{rhs}(R) \cdot \{\#\}^*)$ and $\mathsf{match}(R_\#)$.

3 Finding a Compatible Automaton

Due to the observations we made termination of an SRS can be proved by proving the existence of a finite automaton A compatible with a language L and a (usually infinite) SRS R. Such an automaton A is called *exact* if $L(A) = \to_R^*(L)$.

We describe two basic ways of constructing a compatible automaton: an *exact* approach that always yields an exact automaton in case of success, and an *approximate* approach that yields a compatible automaton that need not be exact. Each approach starts from an automaton that accepts L.

3.1 The Exact Approach

The exact approach is based on the notion of a *deleting* string rewriting system [8, 9]. A string rewriting system R over an alphabet Σ is called *deleting* if it has no empty lhs and there is an irreflexive partial order $>$ on Σ (a *precedence*) such that for each rule $\ell \to r$ in R and for each letter a in r, there is some letter b in ℓ with $b > a$.

Similar to the proof of Theorem 1 it is easy to see that every deleting string rewriting system over a finite alphabet is terminating. The class of deleting string rewriting systems enjoys the following strong effective decomposition property. An SRS is called context-free if every lhs has length at most 1.

Theorem 4 ([9]). *If R is a finite deleting string rewriting system over a finite alphabet Σ, then there are, effectively, an extended alphabet $\Gamma \supseteq \Sigma$, a terminating, context-free SRS T over Γ, and a context-free SRS C over Γ, such that for each language $L \subseteq \Sigma^*$, $\to_R^*(L) = \leftarrow_C^*(\to_T^*(L)) \cap \Sigma^*$.*

The exact approach consists of using this decomposition to construct an automaton that accepts $\to_R^*(L)$ from an automaton that accepts L. Let $\mathsf{match}_k(R)$ be the restriction of $\mathsf{match}(R)$ to the (finite) alphabet $\Sigma \times \{0, \ldots, k\}$. It is obvious that $\mathsf{match}_k(R)$ is deleting, so Theorem 4 applies. Now a finite automaton A_k is

constructed such that $L(A_k) = \mathsf{base}(\rightarrow^*_{\mathsf{match}_k(R)}(\mathsf{lift}_0(L)))$. We do this construction successively for $k = 0, 1, 2, \ldots$. If some k is found that satisfies $L_k = L_{k+1}$ then A_k is the desired exact automaton compatible with $\mathsf{match}(R)$ and L.

The extended alphabet may turn out to be rather large (a few hundred letters even for small systems), so the automaton for $\rightarrow^*_T(L)$ has many states as well. We do not give details here; to give a flavor we sketch a small fragment of this approach as it occurs for computing A_2 for the single rule $aa \to aba$.

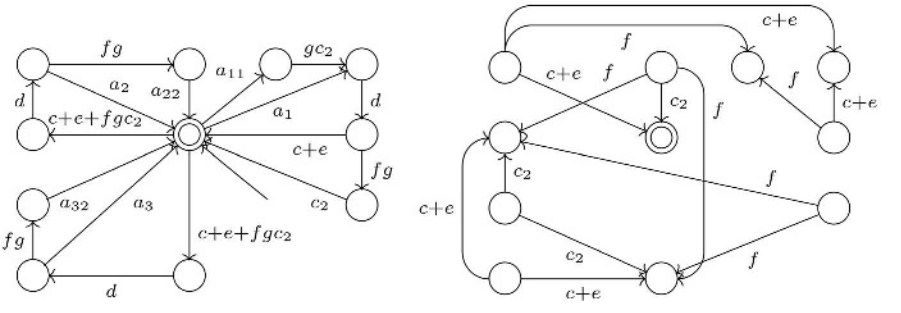
Let R be the SRS consisting of the rules $aa \to cdc, ac \to cdc, ca \to cdc, cc \to fgf$. This SRS is a renaming of (the accessible part–see [5, 6]–of) the system $\mathsf{match}_2(\{aa \to aba\})$. Observe that R is deleting with $a > c > d > f > g$. Using Theorem 4 we get the decomposition $\Gamma = \Sigma \cup \{b, e, a_1, a_2, a_3, a_{22}, a_{32}, a_{11}, c_2\}$;

$$T = \begin{cases} a \to b, & a \to cda_2, & a \to cda_3, & a \to a_1dc, & c \to e, \\ c \to fgc_2, & a_2 \to fga_{22}, & a_3 \to fga_{32}, & a_1 \to a_{11}gc_2; \end{cases}$$

$$C = \begin{cases} c \to a_2b, & e \to a_2b, & c \to a_3c, & e \to a_3c, & c \to ca_1, \\ e \to ca_1, & f \to c_2e, & c_2 \to a_{22}b, & c_2 \to a_{32}c, & f \to ca_{11}. \end{cases}$$

Now let $L = \{a\}^*$. The automaton for $\rightarrow^*_T(L)$ is constructed by supplementing, as long as possible, a path $p \xrightarrow{r} q$ for each transition $p \xrightarrow{x} q$ and rule $x \to r$ in T.

For a better overview, we render the initial state in the center of the left figure without the looping transitions labelled by a and b.



The automaton for $\leftarrow^*_C(\rightarrow^*_T(L))$ is obtained by adding, as long as possible, a transition $p \xrightarrow{x} q$ for each path $p \xrightarrow{r} q$ and rule $x \to r$ in C, see [1, 2]. The result is given in the right figure, rendering only the new transitions. Finally, the automaton for $\rightarrow^*_R(L)$ is obtained by dropping all transitions labelled with $\Gamma \setminus \Sigma$ letters.

3.2 The Approximate Approach

The intermediate automata constructed during the exact approach may be much bigger than the final compatible automaton. For simple examples these intermediate automata may have thousands of nodes and may take minutes to compute.

We therefore introduce an approximate approach that avoids the construction of automata that are bigger than the result.

The basic idea of this approach is to start with an automaton that accepts exactly L, and then to add only states and transitions as needed for closure under rewriting. More precisely, the procedure systematically looks for counterexamples to closure under rewriting, i.e., for states p, q and rules $\ell \to r \in R$ such that $p \xrightarrow{\ell}_A q \wedge p \not\xrightarrow{r}_A q$. States and transitions are added to the automaton such that $p \xrightarrow{r}_A q$. The procedure repeats this step until either there are no counterexamples left, in which case the resulting automaton is compatible; or the resources are exceeded and the construction has failed.

There are various strategies how to add states and transitions suitably; we will describe two of them.

Our approach is currently restricted to the case where each right hand side is non-empty. Empty right hand sides can be included by automata having epsilon-transitions. Alternatively, empty right hand sides can be eliminated by preprocessing the SRS as we will see in an example.

The Strategy in TORPA [15]. If A has to be extended in order to satisfy $p \xrightarrow{r}_A q$ in TORPA this is done as follows. As stated above, we assume that r is non-empty, so we may write $r = au$ for $a \in \Sigma, u \in \Sigma^*$. It is checked whether a state n exists satisfying $n \xrightarrow{u}_A q$. If so, then only one single transition $p \xrightarrow{a}_A n$ is added. If not, then a completely fresh path from p to q is constructed: if $r = a_1 \cdots a_k$ then $k-1$ fresh states $n_1, \ldots, n_{k-1}$ and k fresh transitions

$$p \xrightarrow{a_1}_A n_1, \;\; n_1 \xrightarrow{a_2}_A n_2, \;\; \ldots, \; n_{k-2} \xrightarrow{a_{k-1}}_A n_{k-1}, \;\; n_{k-1} \xrightarrow{a_k}_A q$$

are added. In both cases the extended automaton A indeed satisfies $p \xrightarrow{r}_A q$.

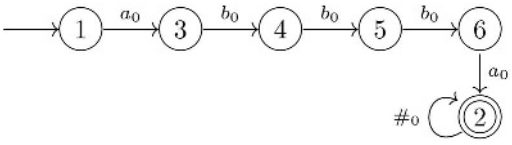
This strategy is particularly powerful for proving termination using $\mathsf{match}(R)$ and forward closures. As a simple example consider the SRS R consisting of the single rule $aba \to abbba$. Due to Theorem 3 termination of R may be proved by proving that $R_\#$ is terminating on $\{abbba\} \cdot \{\#\}^*$, where $R_\#$ consists of the rules

$$aba \to abbba, \;\; a\# \to abbba, \;\; ab\# \to abbba.$$

Due to Theorems 1 and 2 it now suffices to find a compatible automaton for $\{a_0b_0b_0b_0a_0\} \cdot \{\#_0\}^*$ and $\mathsf{match}(R_\#)$. Here we shortly write x_i rather than (x, i) for $x \in \{a, b, \#\}, i \in \mathbf{N}$, and $\mathsf{match}(R_\#)$ consists of the rules

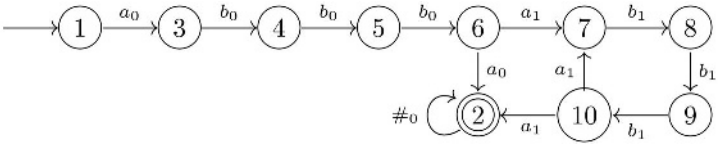
$$\begin{array}{rll} a_ib_ja_k \to a_mb_mb_mb_ma_m & \text{for } i,j,k,m \in \mathbf{N},\ m = \min\{i,j,k\}+1 \\ a_i\#_0 \to a_1b_1b_1b_1a_1 & \text{for } i \in \mathbf{N} \\ a_ib_j\#_0 \to a_1b_1b_1b_1a_1 & \text{for } i,j \in \mathbf{N}. \end{array}$$

Formally $\mathsf{match}(R_\#)$ also contains rules with $\#_i$ in the left hand side for $i > 0$, but it is easy to see that these will never be involved in derivations starting from a string not containing $\#_i$ for $i > 0$. This observation holds for every SRS, not only this example. Now the search for a compatible automaton may start. We start by the following automaton for $\{a_0b_0b_0b_0a_0\} \cdot \{\#_0\}^*$:



The numbering of the nodes is as generated in TORPA where the initial node is always 1 and the final node is always 2.

The first counterexample we find is $6 \overset{a_0\#_0}{\rightarrow}_A 2$ for the rule $a_0\#_0 \rightarrow a_1b_1b_1b_1a_1$ in $\mathsf{match}(R_\#)$. We have to construct a path from 6 to 2 labelled by $a_1b_1b_1b_1a_1$. As there is no path from any state to 2 labelled by $b_1b_1b_1a_1$, a fresh path is added with the fresh states 7, 8, 9, 10 and transitions between them. The next counterexample is $10 \overset{a_1\#_0}{\rightarrow}_A 2$ for the rule $a_1\#_0 \rightarrow a_1b_1b_1b_1a_1$ of $\mathsf{match}(R_\#)$. Here a path from 10 to 2 labelled by $a_1b_1b_1b_1a_1$ has to be created. Since there is already a path from 7 to 2 labelled by $b_1b_1b_1a_1$, only the single transition from 10 to 7 is added. There are no further counterexamples. This yields the following compatible automaton:

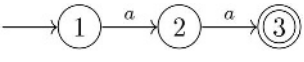


This simple strategy was found during a trial to reconstruct by pencil and paper the exact automaton for $\mathsf{match}(R_\#)$ for $R = \{aabb \rightarrow bbbaaa\}$ as it was given in [5, 6]. Using this strategy, TORPA generates the same automaton, only a few hundred times faster. Other, more involved strategies turned out unsatisfactory for forward closures.

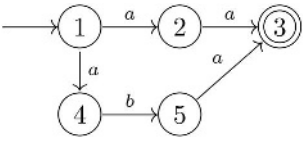
The Strategy in Matchbox [12]. For a rule $\ell \rightarrow r \in R$ and states p, q in A for which $p \overset{\ell}{\rightarrow}_A q$ holds but not $p \overset{r}{\rightarrow}_A q$, this strategy considers all decompositions $r = xyz$ and states p', q' such that $p \overset{x}{\rightarrow}_A p'$ and $q' \overset{z}{\rightarrow}_A q$ and $y \neq \epsilon$. Then among all possibilities one is chosen for which y has minimal length, and a new path $p' \overset{y}{\rightarrow}_A q'$ is constructed.

The TORPA strategy can also be seen as a variant of this decomposition approach, constrained by $|x| = 0 \wedge (|y| = 1 \vee |z| = 0)$.

As an illustration of both strategies consider the automaton

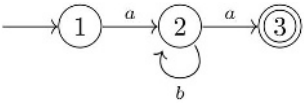


for $L = \{aa\}$ and the rule $aa \rightarrow aba$. Then a path $1 \overset{aba}{\rightarrow}_A 3$ has to be created. In the TORPA strategy it is observed that no state n exists satisfying $n \overset{ba}{\rightarrow}_A 3$. As a consequence two fresh states 4, 5 and three fresh transitions are created:



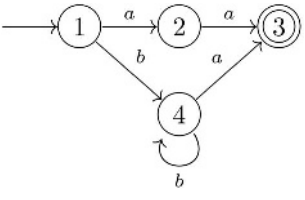
After this single step the automaton is closed under rewriting.

In contrast, in the `Matchbox` strategy no fresh states are required at all: by $1 \xrightarrow{a}_A 2$ and $2 \xrightarrow{a}_A 3$ only one fresh b-transition from 2 to 2 is added:



and again after a single step the automaton is closed under rewriting. This illustrates non-exactness of the `Matchbox` strategy: by the latter automaton A the string $abba$ is in $L(A)$ but not in $\to_R^*(L)$.

Similar non-exactness occurs in the `TORPA` strategy, for instance by starting from the automaton



and the same rule $aa \to aba$ in the `TORPA` strategy the transition $1 \xrightarrow{a}_A 4$ is added, by which again the string $abba$ is in $L(A)$ but not in $\to_R^*(L)$.

We have experimented with several conceivable variants of such a strategy for re-using transitions. Roughly speaking in using forward closures the `TORPA`-strategy is often the most successful, while in not using forward closures the `Matchbox`-strategy is often more powerful. In any case, the order in which paths are handled strongly influences the process; and there does not seem to be a straightforward *complete* strategy: one that terminates in all cases where the exact computation is successful.

3.3 An Example with Empty Right Hand Sides

In an SRS one or more rules $\ell \to r$ may be replaced by $a\ell \to ar$ for all $a \in \Sigma$ without changing the termination behavior. As an example, consider the SRS R consisting of the rules

$$Ab \to baBA,\ Ba \to abAB,\ Aa \to \epsilon,\ Bb \to \epsilon.$$

Due to empty rhs's the approximate approach cannot be applied directly. However, by using the above observation termination of R may be concluded from termination of the SRS consisting of the rules

$$Ab \to baBA,\ Ba \to abAB,\quad aAa \to a,\ bAa \to b,\ AAa \to A,\ BAa \to B,$$

$$aBb \to a,\ bBb \to b,\ ABb \to A,\ BBb \to B.$$

This is easily proved using our approximate approach for forward closures by an automaton having only 14 states.

4 Conclusions

For an extensive class of string rewriting systems, termination can be shown by the construction of a compatible automaton, i.e., a finite automaton that has a suitable closure property. Whereas in theory the construction of an exact compatible automaton always succeeds if it exists, i.e., if the system is match-bounded, it may be prohibitively expensive. So we proposed an approximate approach that is more efficient but may fail. For instance, this new approach allows to prove termination of the single rule SRSs

$$\{babaa \to aaababab\},\ \{babaa \to abaabbaba\},\ \{baaabbaa \to aaabbaaabb\}$$

within fractions of seconds by automata having 72, 98 and 155 states, respectively. The exact approach, in contrast, fails due to lack of memory. All standard techniques for proving termination [14] fail, too, for these examples.

The notion of match-bounds was inspired by Ravikumar [10], who showed that *change-bounded* string rewriting preserves regularity of languages. Similar to $\mathsf{match}(R)$, he defined a related system over the alphabet $\Sigma \times \mathbf{N}$ to consist of all rules $(a_1, n_1) \cdots (a_p, n_p) \to (b_1, n_1 + 1) \cdots (b_p, n_p + 1)$ for which $a_1 \cdots a_p \to b_1 \cdots b_p$ is in the system R over Σ. This definition, however, is only meaningful for length-preserving systems, where $|\ell| = |r|$ for every rule $\ell \to r$. For this particular class of systems it can be shown that match-boundedness actually coincides with change-boundedness.

Presently we work on extending these techniques to term rewriting. To this end the automata are replaced by finite tree automata.

References

1. R. V. Book, M. Jantzen, and C. Wrathall. Monadic Thue systems. *Theoret. Comput. Sci.*, 19:231–251, 1982.
2. R. V. Book and F. Otto. Cancellation rules and extended word problems. *Inform. Process. Lett.*, 20:5–11, 1985.
3. R. V. Book and F. Otto. *String-Rewriting Systems.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1993.
4. N. Dershowitz. Termination of linear rewriting systems. In S. Even and O. Kariv (Eds.), *Proc. 8th Int. Coll. on Automata, Languages and Programming ICALP-81*, *Lecture Notes in Comput. Sci.* Vol. 115, pp. 448-458. Springer-Verlag, 1981.
5. A. Geser, D. Hofbauer and J. Waldmann. Match-bounded string rewriting systems. In B. Rovan and P. Vojtas (Eds.), *Proc. 28th Int. Symp. Mathematical Foundations of Computer Science MFCS-03*, *Lecture Notes in Comput. Sci.* Vol. 2747, pp. 449-459. Springer-Verlag, 2003.

6. A. Geser, D. Hofbauer and J. Waldmann. Match-bounded string rewriting systems. NIA Report 2003-09, National Institute of Aerospace, Hampton, VA, USA. Available at `http://research.nianet.org/~geser/papers/nia-matchbounded.html`. Accepted for *Appl. Algebra Engrg. Comm. Comput.*.
7. J. Giesl, R. Thiemann, P. Schneider-Kamp and S. Falke. Automated termination proofs with AProVE. In V. van Oostrom (Ed.), *Proc. 15th Int. Conf. Rewriting Techniques and Applications RTA-04*, *Lecture Notes in Comp. Sci.*, Springer, 2004. Tool and description available at `http://www-i2.informatik.rwth-aachen.de/AProVE/`.
8. T. N. Hibbard. Context-limited grammars. *J. ACM*, 21(3):446–453, 1974.
9. D. Hofbauer and J. Waldmann. Deleting string rewriting systems preserve regularity. In *Proc. 7th Int. Conf. Developments in Language Theory DLT-03*, *Lect. Notes Comp. Sci.*, Springer-Verlag, 2003. Accepted for *Theoret. Comput. Sci.*.
10. B. Ravikumar. Peg-solitaire, string rewriting systems and finite automata. In H.-W. Leong, H. Imai, and S. Jain (Eds.), *Proc. 8th Int. Symp. Algorithms and Computation ISAAC-97*, *Lecture Notes in Comput. Sci.* Vol. 1350, pp. 233–242. Springer-Verlag, 1997.
11. E. Tahhan Bittar. Complexité linéaire du problème de Zantema. *C. R. Acad. Sci. Paris Sér. I Inform. Théor.*, t. 323:1201–1206, 1996.
12. J. Waldmann. Matchbox: a tool for match-bounded string rewriting, In V. van Oostrom (Ed.), *Proc. 15th Int. Conf. Rewriting Techniques and Applications RTA-04*, *Lecture Notes in Comp. Sci.*, Springer, 2004. Tool and description available at `http://theo1.informatik.uni-leipzig.de/matchbox/`.
13. H. Zantema and A. Geser. A complete characterization of termination of $0^p1^q \to 1^r0^s$. *Appl. Algebra Engrg. Comm. Comput.*, 11(1):1–25, 2000.
14. H. Zantema. Termination. In *Term Rewriting Systems, by Terese*, pages 181–259. Cambridge University Press, 2003.
15. H. Zantema. TORPA: Termination of Rewriting Proved Automatically. In V. van Oostrom (Ed.), *Proc. 15th Int. Conf. Rewriting Techniques and Applications RTA-04*, *Lecture Notes in Comp. Sci.*, Springer, 2004. Tool and description available at `http://www.win.tue.nl/~hzantema/torpa.html`.
16. H. Zantema. Termination of string rewriting proved automatically. Accepted for *J. Automat. Reason.*, 2004.