Nontermination of String Rewriting using SAT[⋆]

Harald Zankl and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, Austria

Abstract. This note presents a new approach for proving nontermination of SRSs. We encode rewrite sequences as propositional formulae which are satisfiable whenever the corresponding sequence includes a looping reduction, which can be constructed from the assignment.[1]

1 Introduction

Establishing termination of rewrite systems is a challenging endeavor since it is undecidable in general. Nevertheless these days many powerful algorithms exist which are able to automatically prove termination for a huge class of rewrite systems as witnessed by the annual termination competition.[2] The few techniques known for proving nontermination are based on narrowing [4], unfolding [5], matchbounds [6], and ancestor graphs [7].

2 Encoding Looping Reductions in SAT

A *string rewrite system* (SRS) $\mathcal{S}$ is called looping if there exist a context C, a substitution σ, and a term t with $t \to_{\mathcal{S}}^{+} C[t\sigma]$. When applying the dependency pair approach [1] one can forget about leading contexts due to minimality considerations, and a looping reduction takes the form $u \to_{\mathcal{S}\cup\mathsf{DP}(\mathcal{S})}^{+} u\sigma$ where rules in $\mathsf{DP}(\mathcal{S})$ are applied at root positions and the ones in $\mathcal{S}$ below the root.

Our idea is to encode a looping rewrite sequence in a matrix of dimension $n \times m$ with $(0,0)$ the top leftmost and $(n-1, m-1)$ the bottom rightmost entry. Every row in the matrix corresponds to a string and from row i to row $i+1$ a rewrite step is performed. We are now going to define the necessary properties one by one after the following example.

Example 1. Consider the SRS $\mathcal{S} = \{\mathsf{ab} \to \mathsf{bbaa}\}$ with the dependency pair $\mathsf{Ab} \to \mathsf{Aa}$ which forms a cycle in the dependency graph. In a 3×5 matrix a looping reduction is possible. The entries marked with $\cdot$ indicate that any symbol might appear at these positions.

$$\begin{array}{ccccc}
\mathsf{A} & \mathsf{b} & \mathsf{b} & \cdot & \cdot \\
\mathsf{A} & \mathsf{a} & \mathsf{b} & \cdot & \cdot \\
\mathsf{A} & \mathsf{b} & \mathsf{b} & \mathsf{a} & \mathsf{a}
\end{array}$$

[⋆] This research is supported by FWF (Austrian Science Fund) project P18763.

[1] The idea of encoding computation as propositional satisfiability goes back to [2].

[2] `www.lri.fr/~marche/termination-competition`

For the propositional representation we need the following variables:

M_{ij}^a	indicating that symbol a occurs at position (i,j) in the matrix,
$R_i^{l\to r}$	indicating that in row i of the matrix rule $l \to r$ applies,
$\mathbf{p}_i$	indicating the position (= column) in row i where the rule is applied (in the example $\mathbf{p}_0 = 0$ and $\mathbf{p}_1 = 1$), and
$\mathbf{e}_i$	pointing to the last symbol of the term (in the example possible values for $\mathbf{e}_0$, $\mathbf{e}_1$, and $\mathbf{e}_2$ are 2, 2, and 4).

The variables $\mathbf{p}_i$ and $\mathbf{e}_i$ are not boolean but represent integer values which are implemented as lists of boolean variables encoding the actual integer value in binary. To distinguish them from proper propositional variables they are denoted in boldface.

Exactly one function symbol: To get exactly one function symbol per matrix entry, multiple symbols at the same entry have to be banned and at least one symbol must occur. Note that dependency pair symbols (those in $\mathcal{F}^\sharp \setminus \mathcal{F}$) can only appear at the root position. We express this property below where $X = \mathcal{F}$ if $j > 0$ and $X = \mathcal{F}^\sharp \setminus \mathcal{F}$ otherwise:

$$\alpha_{ij} = \Big(\bigvee_{a \in X} M_{ij}^a\Big) \wedge \bigwedge_{a \in X} \big(M_{ij}^a \to \bigwedge_{b \in X \setminus \{a\}} \neg M_{ij}^b\big)$$

Rule application: If a rule $l \to r$ applies in row i we have to match the left-hand side somewhere in row i and the right-hand side in row $i+1$ of the matrix (see $\mathit{applies}_i^{l\to r}$). Furthermore all entries which are not affected by a rule application must be copied from row i to row $i+1$ (see $\mathit{copy}_{ij}^{l\to r}$). The position of the rule application is fixed by $\mathbf{p_i}$ and satisfying $\mathit{copy}_{ij}^{l\to r}$ has the side effect that only one rule is applied.

$$\beta_i^{l\to r} = R_i^{l\to r} \to \big(\mathit{applies}_i^{l\to r} \wedge \bigwedge_{0 \leqslant j < m} \mathit{copy}_{ij}^{l\to r}\big)$$

where in case of $l \to r \in \mathcal{S}$ we have

$$\begin{aligned} \mathit{applies}_i^{l\to r} = \bigwedge_{0 \leqslant j < |l|} M_{i(\mathbf{p}_i + j)}^{l_{j+1}} \wedge \bigwedge_{0 \leqslant j < |r|} M_{(i+1)(\mathbf{p}_i + j)}^{r_{j+1}} \\ \wedge (\mathbf{e}_{i+1} + |l| = \mathbf{e}_i + |r|) \wedge (\mathbf{e}_i \geqslant \mathbf{p}_i + |l|) \end{aligned}$$

and if $l \to r \in \mathsf{DP}(\mathcal{S})$ then $\mathbf{p}_i$ specializes to 0 and the conjunct $\mathbf{p}_i = 0$ is added. The formula $\mathit{applies}_i^{l\to r}$ applies the left-hand side in row i and the right-hand side in row $i+1$ at the abstract position $\mathbf{p}_i$. The last but one conjunct demands that the end pointer in line $i+1$ takes the value of $\mathbf{e}_i - |l| + |r|$. To ensure that the contracted redex fits in line i the last conjunct must be satisfied.

The formula for $\mathit{copy}_{ij}^{l\to r}$ is defined as $\top$ if $j + \max\{|l|, |r|\} \geqslant m$ (these entries would be outside the matrix) and otherwise as

$$\Big((j < \mathbf{p}_i) \wedge \bigwedge_{a \in X} \big(M_{ij}^a \leftrightarrow M_{(i+1)j}^a\big)\Big) \vee \Big((j \geqslant \mathbf{p}_i) \wedge \bigwedge_{a \in \mathcal{F}} \big(M_{i(j+|l|)}^a \leftrightarrow M_{(i+1)(j+|r|)}^a\big)\Big)$$

(where $X = \mathcal{F}^\sharp \setminus \mathcal{F}$ if $j = 0$ and $X = \mathcal{F}$ otherwise) if $l \to r \in \mathcal{S}$ and in case $l \to r \in \mathsf{DP}(\mathcal{S})$ as $\bigwedge_{a \in \mathcal{F}} \left(M^a_{i(j+|l|)} \leftrightarrow M^a_{(i+1)(j+|r|)}\right)$.

All entries in the matrix before the position where the rule is applied are copied from row i to $i+1$. The second disjunct copies the entries after $\mathbf{p}_i$ which are unaffected when applying the rule. The position of these entries change if the applied rule is not length-preserving. For rules $l \to r \in \mathsf{DP}(\mathcal{S})$ we know that $\mathbf{p}_i = 0$ and hence the formula simplifies to the one shown above.

Initial term is reached again: To recognize a loop the string in some line $i > 0$ must match the one in line zero. Furthermore the end pointer for line i is demanded not to be smaller than the one of line zero.

$$\gamma = \bigvee_{0<i<n} \left((\mathbf{e}_i \geqslant \mathbf{e}_0) \wedge \bigwedge_{a \in \mathcal{F}^\sharp \setminus \mathcal{F}} \left(M^a_{00} \leftrightarrow M^a_{i0}\right) \wedge \bigwedge_{\substack{0<j<m \\ a \in \mathcal{F}}} \left(M^a_{0j} \leftrightarrow M^a_{ij}\right) \right)$$

Putting everything together: The formula $NT_{\mathcal{S}}$ is defined as

$$\left(\bigwedge_{0 \leqslant i<n} (\bigwedge_{0 \leqslant j<m} \alpha_{ij}) \wedge (\mathbf{e}_i < m) \wedge \beta_i \right) \wedge \gamma$$

with

$$\beta_i = \bigvee_{l \to r \in \mathcal{S} \cup \mathsf{DP}(\mathcal{S})} R_i^{l \to r} \ \wedge \bigwedge_{l \to r \in \mathcal{S} \cup \mathsf{DP}(\mathcal{S})} \beta_i^{l \to r}$$

expressing that one rule must apply and it is applied properly. The condition $\mathbf{e}_i < m$ ensures that all rewrite steps are performed within the allowed matrix dimensions.

There are two types of variables keeping track of function symbols—concrete (M^a_{ij}) and abstract ones ($M^a_{i\mathbf{x}}$) where $\mathbf{x}$ is a list of propositional variables representing an integer in binary. The latter ones are needed when a rule is applied at the abstract position $\mathbf{p}_i$. In the current encoding, abstract variables $M^{\mathsf{a}}_{3[x_1;x_0]}$ and $M^{\mathsf{a}}_{3[y_1;y_0]}$ denote two different expressions and hence may take different values. If the assignments for x_1 and y_1 as well as x_0 and y_0 are the same, we want to enforce that the variables take identical values. In the implementation we test for every such abstract variable whether it matches a concrete one and we identify them if that is the case:

$$\varphi_{aux} = \bigwedge_{0 \leqslant i<n} \bigwedge_{0 \leqslant j<m} \bigwedge_{M^a_{i\mathbf{p}}} \left(\mathbf{p} = j \to (M^a_{ij} \leftrightarrow M^a_{i\mathbf{p}})\right)$$

Lemma 2. *If $NT_{\mathcal{S}} \wedge \varphi_{aux}$ is satisfiable then $\mathcal{S}$ admits a looping reduction.* $\square$

3 Experimental Results

The 322 SRSs from the termination problem data base 2006 have been considered. All tests have been performed on a laptop with a clock rate of 2.2GHz and

1GB memory. A time limit of 60 seconds was used. Satisfiability of the propositional formulae was tested by the award winning SAT solver MiniSat [3]. Our implementation is build on top of T_TT2[3], the successor of T_TT.

After computing the SCCs in the estimated dependency graph our implementation tries to remove rules which cannot contribute to a nonterminating sequence (to be precise, linear polynomial interpretations with coefficients in $\{0, 1\}$ are applied to get smaller SCCs). We anticipate that applying more advanced termination criteria will result in a speedup of our implementation. Our solver tries matrices of different dimensions. The heuristic first tries small dimensions, then increases the number of rows and columns, respectively. The maximum number of rows and columns we try is 28. Needless to say, it is to some extent designed for the given database. Counterexamples are found mostly within a few seconds if they exist but the SAT solver seems to run forever if the SRS is not looping.

In the SRS category of the 2006 termination competition Jambox could disprove 25 SRSs terminating, Matchbox and AProVE came second with 12 systems followed by TORPA with 4 systems. TEPARLA proved nontermination of a single system whereas CiME, MultumNonMulta, TPA, and T_TTbox appear not to have implemented any method for proving nontermination. Our implementation proves nontermination of 24 SRSs within the given time limit. All of these SRSs are also handled by Jambox. The missing system is secret2006_matchbox_1 (which takes our tool 13 seconds using dimension 15×20). Nevertheless, our SAT based approach is not subsumed by the methods implemented in Jambox. None of the tools participating in last year's competition can handle any of the 335 challenging and small SRSs collected by Johannes Waldmann,[4] whereas we can disprove termination of several of them.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. S.A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd STOC*, pages 151–158. ACM, 1971.
3. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. 7th SAT*, volume 2919 of *LNCS*, pages 502–518, 2004.
4. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proc. 5th FroCoS*, volume 3717 of *LNAI*, pages 216–231, 2005.
5. É. Payet. Detecting non-termination of term rewriting systems using an unfolding operator. In *Proc. 16th LOPSTR*, volume 4407 of *LNCS*, pages 194–209, 2006.
6. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. 15th RTA*, volume 3091 of *LNCS*, pages 85–94, 2004.
7. H. Zantema. Termination of rewriting proved automatically. *Journal of Automated Reasoning*, 34:105–139, 2005.

[3] colo6-c703.uibk.ac.at/ttt2

[4] dfa.imn.htwk-leipzig.de/matchbox/new-problems/srs-2007-02-28/