Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 246 (2009) 199–214
www.elsevier.com/locate/entcs

Transforming SAT into Termination of Rewriting [1]

Harald Zankl[2] Christian Sternagel[3] Aart Middeldorp[4]

Institute of Computer Science
University of Innsbruck
Innsbruck, Austria

Abstract

In this paper we propose different translations from SAT to termination of term rewriting, i.e., we translate a propositional formula φ into a generic rewrite system $\mathcal{R}^\varphi$ with the property that φ is satisfiable if and only if $\mathcal{R}^\varphi$ is (non)terminating. Our experiments reveal that the generated rewrite systems are challenging for automated termination provers. Furthermore, a large class of them seems to be just unprovable by current methods implemented in termination analyzers.

Keywords: term rewriting, termination, semantic labeling, SAT solving

1 Introduction

Termination of term rewrite systems (TRSs) is an undecidable property [12]. Nevertheless, nowadays powerful (incomplete) algorithms exist that can prove termination of many rewrite systems as can be witnessed by the international termination competition. [5] In 2004 Kurihara and Kondo were the first who encoded a termination method in propositional logic [19] and in 2006 the first tools (Jambox [6] and Matchbox [21]) employed SAT-solving techniques in the competition. They surprised the community by the gains in power and speed. Their success was mainly due to the so-called matrix-method [8] which can effectively be implemented using SAT-solvers. But even for very simple and ancient methods like the lexicographic path order [13,6] (LPO) the recent development in the SAT community allows way faster

[1] This research is supported by FWF (Austrian Science Fund) project P18763.
[2] Email: harald.zankl@uibk.ac.at
[3] Email: christian.sternagel@uibk.ac.at
[4] Email: aart.middeldorp@uibk.ac.at
[5] http://www.lri.fr/~marche/termination-competition/
[6] Available from http://joerg.endrullis.de.

doi:10.1016/j.entcs.2009.07.023

implementations [4] than some years ago. A similar speedup [22] is achieved for the Knuth-Bendix order (KBO) [14]. This is remarkable because KBO orientability is known to be decidable in polynomial time [18] whereas SAT is NP-complete [5]. In other words, the sophisticated algorithms for solving the computationally harder (unless P = NP) problem SAT outperform the dedicated methods for KBO [7,18]. In this paper we address the question whether a similar result also holds when translating the NP-complete SAT problem into the undecidable termination property of TRSs. However, the experiments reveal that at least for our translations the results are as expected. Concerning the transformation from SAT to termination, the dedicated SAT approaches perform much better. Even further, only the most simple propositional formulas produce TRSs which can be shown terminating by state-of-the-art termination provers. Therefore the translations can be used to generate a large set of difficult termination problems automatically.

The rest of the paper is organized as follows: In Section 2 propositional formulas are introduced and many-sorted rewriting is defined. In Section 3 we define TRSs $\mathcal{U}^\varphi$ that are terminating if and only if the propositional formula φ is unsatisfiable. In Section 4 the dual problem is considered for many-sorted TRSs $\mathcal{S}^\varphi$ and $\mathcal{T}^\varphi$ that are terminating if and only if φ is satisfiable. That even simple propositional formulas produce TRSs where termination analysis is challenging is demonstrated in Section 5 where it also becomes apparent that narrowing [10] is one method which can handle small instances. We conclude in Section 6.

2 Preliminaries

In this section we fix basic notation concerning propositional logic, introduce many-sorted TRSs and define the model variant of semantic labeling [24] in a many-sorted setting. Aoto and Yamada [1] already generalized semantic labeling to many-sorted rewriting but just for the quasi-model case.

2.1 Propositional Logic

Let $\mathcal{A}$ be a set of propositional variables (atoms). Sometimes we find it convenient to abbreviate the set of atoms $p_1, \ldots, p_n$ by $\mathcal{A}_n$. The set of propositional formulas $\mathcal{P}(\mathcal{A})$ is inductively defined by the following BNF

$$\varphi ::= p \in \mathcal{A} \mid (\varphi \wedge \varphi) \mid (\neg\varphi)$$

Note that we do not allow disjunction (which does not pose a restriction but allows to keep the presentation concise). The following convention is used to reduce the number of parentheses: (i) Outermost parenthesis are omitted, (ii) '$\wedge$' is left-associative, and (iii) '$\neg$' binds stronger than '$\wedge$'.

Let $\mathbb{B} := \{0, 1\}$. An *assignment* is a mapping $\alpha\colon \mathcal{A} \to \mathbb{B}$. It is lifted to an

interpretation of formulas: $[\alpha]\colon \mathcal{P}(\mathcal{A}) \to \mathbb{B}$ with

$$[\alpha](\varphi) = \begin{cases} \alpha(p) & \text{if } \varphi = p \text{ for some } p \in \mathcal{A} \\ [\alpha](\psi) \cdot [\alpha](\chi) & \text{if } \varphi = \psi \wedge \chi \\ \overline{[\alpha](\psi)} & \text{if } \varphi = \neg\psi \end{cases}$$

Here $(\cdot)\colon \mathbb{B}\times\mathbb{B} \to \mathbb{B}$ is defined as $x \cdot y = 1$ if and only if $x = y = 1$ and for $\overline{(*)}\colon \mathbb{B} \to \mathbb{B}$ we have $\overline{x} = 1$ if and only if $x = 0$. A formula φ is *satisfiable (unsatisfiable)* if an (no) assignment α exists such that $[\alpha](\varphi) = 1$. This problem is known as the satisfiability problem (SAT). For a propositional formula φ its *depth* is defined as follows: $\mathsf{depth}(p) = 0$ for $p \in \mathcal{A}$, $\mathsf{depth}(\varphi \wedge \psi) = 1 + \mathsf{max}(\mathsf{depth}(\varphi), \mathsf{depth}(\psi))$, and $\mathsf{depth}(\neg\varphi) = 1 + \mathsf{depth}(\varphi)$. Similarly the set of variables $\mathcal{V}\mathsf{ar}(\varphi)$ is defined recursively by: $\mathcal{V}\mathsf{ar}(p) = \{p\}$ for $p \in \mathcal{A}$, $\mathcal{V}\mathsf{ar}(\varphi \wedge \psi) = \mathcal{V}\mathsf{ar}(\varphi) \cup \mathcal{V}\mathsf{ar}(\psi)$, and $\mathcal{V}\mathsf{ar}(\neg\varphi) = \mathcal{V}\mathsf{ar}(\varphi)$. The well-known coincidence lemma states that when testing φ for satisfiability only the (finitely many) variables that actually occur in φ have to be considered which makes SAT decidable because the search space becomes finite. Furthermore, this allows us to relate assignments to substitutions (whose domain must be finite by definition) in the next section. Despite the fact that the search space for a satisfying assignment is finite, deciding SAT is difficult, more precisely, SAT is an NP-complete problem [5].

2.2 Many-Sorted Semantic Labeling

We assume basic familiarity with term rewriting [3]. Let $\mathcal{S}$ be a non-empty set of *sorts*. An *$\mathcal{S}$-sorted* signature is a set of function symbols $\mathcal{F}$, where each $f \in \mathcal{F}$ of arity n is associated with the *function signature* $\mathsf{sig}\colon \mathcal{F} \to \mathcal{S}^{n+1}$. Here the first n components of $\mathsf{sig}(f)$ give the sort (type) of each argument and the last gives the sort of the function's result. In the following, we write $f : s_1 \times \cdots \times s_n \to s_{n+1}$, to express that f has (function) signature $(s_1, \ldots, s_{n+1})$.

An $\mathcal{S}$-sorted set A is a family of sets $\{A_s\}_{s\in\mathcal{S}}$. For an $\mathcal{S}$-sorted set $\mathcal{V}$ of *variables* (where $\mathcal{V}_s \cap \mathcal{V}_t = \varnothing$ for $s \neq t$), let $\mathcal{T}(\mathcal{F},\mathcal{V})_s$ denote the set of terms with sort s over $\mathcal{F}$ and $\mathcal{V}$, which is defined inductively by the rules

$$\frac{x \in \mathcal{V}_s}{x} \qquad \frac{f \in \mathcal{F} \quad f : s_1 \times \cdots \times s_n \to s \quad t_i \in \mathcal{T}(\mathcal{F},\mathcal{V})_{s_i}}{f(t_1, \ldots, t_n)}$$

This yields the $\mathcal{S}$-sorted set $\mathcal{T}(\mathcal{F},\mathcal{V}) = \{\mathcal{T}(\mathcal{F},\mathcal{V})_s\}_{s\in\mathcal{S}}$. Associated with every term $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ is its *sort*, i.e., if $t \in \mathcal{T}(\mathcal{F},\mathcal{V})_s$ then $\mathsf{sort}(t) = s$. An $\mathcal{S}$-sorted TRS $\mathcal{R}$ is an $\mathcal{S}$-sorted set of pairs $(l, r) \in \mathcal{R}_s$—the so called *rewrite rules*—written as $l \to r$, such that there exists an $s \in \mathcal{S}$ with $l,\ r \in \mathcal{T}(\mathcal{F},\mathcal{V})_s$ and the usual restrictions that l is not a variable and all variables in r also occur in l are satisfied. In the sequel we identify one-sorted TRSs with unsorted ones and feel free to omit sort information where it is not essential.

Let $\mathcal{F}$ be an $\mathcal{S}$-sorted signature. An $\mathcal{S}$-sorted $\mathcal{F}$-algebra $\mathcal{A}$ consists of an $\mathcal{S}$-sorted *carrier* A (where each $A_s \in A$ is non-empty) and a set of *interpretations* $\{f_\mathcal{A}\}_{f\in\mathcal{F}}$,

such that for each function symbol $f : s_1 \times \cdots \times s_n \to s_{n+1}$ there is an interpretation $f_{\mathcal{A}} \colon A_{s_1} \times \cdots \times A_{s_n} \to A_{s_{n+1}}$. An $\mathcal{S}$-sorted *substitution* $\sigma \colon \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ is a set of mappings $\sigma_s \colon \mathcal{V}_s \to \mathcal{T}(\mathcal{F}, \mathcal{V})_s$ for every $s \in \mathcal{S}$ such that $\sigma(x) \neq x$ only for finitely many $x \in \mathcal{V}$. An $\mathcal{S}$-sorted *assignment* $\alpha \colon \mathcal{V} \to A$ is a set of mappings $\alpha_s \colon \mathcal{V}_s \to A_s$ for every $s \in \mathcal{S}$. For every $\mathcal{S}$-sorted term t and assignment $\alpha \colon \mathcal{V} \to A$, a mapping $[\alpha]_{\mathcal{A}} \colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to A$ is defined inductively

$$[\alpha]_{\mathcal{A}}(t) = \begin{cases} \alpha_s(x) & \text{if } t = x \text{ and } \mathsf{sort}(t) = s \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

An $\mathcal{S}$-sorted $\mathcal{F}$-algebra is a *model* of an $\mathcal{S}$-sorted TRS, if for all $\mathcal{S}$-sorted assignments α and rewrite rules $l \to r \in \mathcal{R}$ it holds that $[\alpha]_{\mathcal{A}}(l) = [\alpha]_{\mathcal{A}}(r)$. A *labeling* L chooses for every $f \in \mathcal{F}$ a set of *labels* L_f. The *labeled signature* is defined by

$$\mathcal{F}_{\mathsf{lab}} = \{f \mid f \in \mathcal{F}, L_f = \varnothing\} \cup \{f_a \mid f \in \mathcal{F}, a \in L_f\}$$

where the arity and function signature of f_a and f coincide. A *labeling* ℓ for an $\mathcal{S}$-sorted algebra $\mathcal{A}$ consists of a labeling L together with a *labeling function* $\ell_f \colon A_{s_1} \times \cdots \times A_{s_n} \to L_f$ for every $f \in \mathcal{F}$ with $L_f \neq \varnothing$ and $\mathsf{sig}(f) = s_1 \times \cdots \times s_n \to s_{n+1}$. Let $A^{\mathcal{V}}$ denote the set of all $\mathcal{S}$-sorted assignments from $\mathcal{V}$ to A. Let ℓ be a labeling for $\mathcal{A}$. For every assignment $\alpha \in A^{\mathcal{V}}$ a mapping $\mathsf{lab}_\alpha \colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}_{\mathsf{lab}}, \mathcal{V})$ is defined inductively as follows

$$\mathsf{lab}_\alpha(t) = \begin{cases} x & \text{if } t = x \\ f(\mathsf{lab}_\alpha(t_1), \ldots, \mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } L_f = \varnothing \\ f_a(\mathsf{lab}_\alpha(t_1), \ldots, \mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } L_f \neq \varnothing \end{cases}$$

with $a = \ell_f([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n))$. For any $\mathcal{S}$-sorted TRS $\mathcal{R}$ over $\mathcal{F}$, together with an $\mathcal{F}$-algebra $\mathcal{A}$ and a labeling ℓ, the $\mathcal{S}$-sorted TRS $\mathcal{R}_{\mathsf{lab}}$ over $\mathcal{F}_{\mathsf{lab}}$ is given by

$$\mathcal{R}_{\mathsf{lab}} = \{\mathsf{lab}_\alpha(l) \to \mathsf{lab}_\alpha(r) \mid l \to r \in \mathcal{R}, \alpha \in A^{\mathcal{V}}\}$$

An $\mathcal{S}$-sorted TRS $\mathcal{R}$ is *terminating* if it does not admit an infinite rewrite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$ starting at some $t_1 \in \mathcal{T}(\mathcal{F}, \mathcal{V})_s$ for some $s \in \mathcal{S}$.

Theorem 2.1 *Let $\mathcal{R}$ be an $\mathcal{S}$-sorted TRS. Let the algebra $\mathcal{A}$ be a model of $\mathcal{R}$ and let ℓ be a labeling for $\mathcal{A}$. Then $\mathcal{R}$ is terminating if and only if $\mathcal{R}_{\mathsf{lab}}$ is terminating.*

For the TRSs we are dealing with in the subsequent sections, many-sorted termination is equivalent to the one-sorted case [23] since the systems are non-collapsing. A TRS is collapsing if it contains a rule $l \to x$ for some variable x. Restricting to many-sortedness simplifies the proofs of Theorems 4.2 and 4.6 considerably.

3 Transforming Unsatisfiability to Termination

In the following we want to express SAT as a termination problem in rewriting, i.e., given a formula φ, we construct a TRS R^φ that is terminating if and only if φ is

satisfiable. This transformation is addressed in the next section. First we focus on the simpler dual problem, namely the construction of a TRS $\mathcal{R}^{\varphi}$ that is terminating if and only if φ is unsatisfiable.

For this purpose we consider a $\{\mathsf{bool}\}$-sorted signature $\mathcal{F} = \{\star, -\}$ containing a binary function symbol $(\star) : \mathsf{bool} \times \mathsf{bool} \to \mathsf{bool}$ and a unary function symbol $- : \mathsf{bool} \to \mathsf{bool}$. Furthermore we assume that the propositional atoms in $\mathcal{A}$ are contained in the set of term variables $\mathcal{V}_{\mathsf{bool}}$. Although '$\star$' will represent (on the term level) the same as '$\wedge$' does on formulas, we use different function symbols because we want to clearly separate between the two different concepts. The same holds for the symbols '$-$' and '$\neg$'. The obvious encoding $\ulcorner * \urcorner \colon \mathcal{P}(\mathcal{A}) \to \mathcal{T}(\{\star, -\}, \mathcal{V})$ transforms formulas into terms as follows: $\ulcorner p \urcorner = p$ for $p \in \mathcal{A}$, $\ulcorner \varphi \wedge \psi \urcorner = \ulcorner \varphi \urcorner \star \ulcorner \psi \urcorner$, and $\ulcorner \neg \varphi \urcorner = - \ulcorner \varphi \urcorner$. Now every well-formed formula in $\mathcal{P}(\mathcal{A})$ has a corresponding term representation in $\mathcal{T}(\mathcal{F}, \mathcal{V})$.

The next goal is to mimic the task of assignments for formulas on the term level. Thus the signature $\mathcal{F}$ is extended by two constant symbols of sort bool, namely '$\bot$' and '$\top$'. We say that an assignment $\alpha \colon \mathcal{A}_n \to \mathbb{B}$ and a substitution $\sigma \colon \mathcal{V}_n \to \{\bot, \top\}$ are *corresponding* if $\alpha(p_i) = 0$ if and only if $\sigma(p_i) = \bot$ for all $1 \leqslant i \leqslant n$ (here $\mathcal{V}_n = \mathcal{A}_n$).

In order to perform the work $[\alpha]$ does on formulas the six rewrite rules

$$\bot \star \bot \to \bot \quad \bot \star \top \to \bot \quad \top \star \bot \to \bot \quad \top \star \top \to \top \quad - \bot \to \top \quad - \top \to \bot$$

referred to as the TRS $\mathcal{S}\mathsf{imp}$ are employed. The next lemma formalizes the interplay of assignments and substitutions.

Lemma 3.1 *Let $\varphi \in \mathcal{P}(\mathcal{A}_n)$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $\ulcorner \varphi \urcorner = t$. If the assignment α and the substitution σ are corresponding, then*

- $[\alpha](\varphi) = 0$ *implies* $t\sigma \to^*_{\mathcal{S}\mathsf{imp}} \bot$ *and dually*
- $[\alpha](\varphi) = 1$ *implies* $t\sigma \to^*_{\mathcal{S}\mathsf{imp}} \top$

Proof By induction on the structure of φ. □

The following example already contains the main idea for constructing nonterminating sequences.

Example 3.2 Consider the formula $\varphi = p_1 \wedge \neg p_2$ with the corresponding term $t = \ulcorner \varphi \urcorner = p_1 \star (- p_2)$. Then the TRS $\mathcal{S}\mathsf{imp}$ together with the rewriting rule

$$\mathsf{unsat}(p_1, p_2, \top) \to \mathsf{unsat}(p_1, p_2, p_1 \star (- p_2))$$

admits the cyclic reduction

$$\mathsf{unsat}(\top, \bot, \top) \to \mathsf{unsat}(\top, \bot, \top \star (- \bot)) \to \mathsf{unsat}(\top, \bot, \top \star \top) \to \mathsf{unsat}(\top, \bot, \top)$$

which proves nontermination of this TRS. The reason for nontermination is that for a satisfying assignment α (in this case $\alpha(p_1) = 1$ and $\alpha(p_2) = 0$) there is

$$\begin{aligned} \bot_{\mathcal{B}} &= 0 & \top_{\mathcal{B}} &= 1 \\ \star_{\mathcal{B}}(x,y) &= x \cdot y & -_{\mathcal{B}}(x) &= \overline{x} \\ \mathsf{unsat}_{\mathcal{B}}(p_1,\ldots,p_n,y) &= 0 \end{aligned}$$

Table 1
A model for the TRS $\mathcal{U}^{\varphi}$.

a *corresponding* substitution σ such that the term $\ulcorner\varphi\urcorner\sigma = t\sigma$ rewrites to $\top$ by Lemma 3.1.

The next theorem formally establishes the relation between satisfiable formulas and nontermination of corresponding TRSs.

Theorem 3.3 *Let $\varphi \in \mathcal{P}(\mathcal{A}_n)$. The parametrized TRS $\mathcal{U}^{\varphi}$ that consists of all rules in $\mathcal{S}$imp and*

$$\mathsf{unsat}(p_1,\ldots,p_n,\top) \to \mathsf{unsat}(p_1,\ldots,p_n,\ulcorner\varphi\urcorner) \tag{1}$$

is terminating if and only if φ is unsatisfiable.

Proof For the direction from left to right assume $\mathcal{U}^{\varphi}$ to be terminating and φ to be satisfiable to arrive at a contradiction. Since φ is satisfiable there must be a satisfying assignment α and a corresponding substitution σ. But then there is the cyclic reduction

$$t = \mathsf{unsat}(\sigma(p_1),\ldots,\sigma(p_n),\top) \to \mathsf{unsat}(\sigma(p_1),\ldots,\sigma(p_n),\ulcorner\varphi\urcorner\sigma) \to^* t$$

where the first rewrite step is an application of rule (1) and the rest of the sequence holds by Lemma 3.1 since $\ulcorner\varphi\urcorner\sigma \to^*_{\mathcal{S}\mathsf{imp}} \top$. Contradiction.

For the direction from right to left we assume unsatisfiability of φ and show termination of $\mathcal{U}^{\varphi}$. For this purpose we apply semantic labeling. Note that we consider $\mathcal{U}^{\varphi}$ as one-sorted. The idea is to label the symbol unsat by the value which φ evaluates to—under all possible assignments. To obtain a model, the function symbols are interpreted in the Boolean algebra. The interpretation $\mathcal{B}$ over the carrier $\mathbb{B}$ depicted in Table 1 is a model for $\mathcal{U}^{\varphi}$. Next the labeling for $\mathcal{U}^{\varphi}$ is defined. For this purpose only the function symbol unsat gets labeled, i.e., $L_{\star} = L_{-} = \varnothing$ [7] and $L_{\mathsf{unsat}} = \mathbb{B}$. The labeling function $\ell_{\mathsf{unsat}} \colon \mathbb{B}^{n+1} \to \mathbb{B}$ is defined as: $\ell_{\mathsf{unsat}}(p_1,\ldots,p_n,y) = y$. By assumption the formula φ evaluates to 0 under all assignments. Hence the labeled variant of rule (1) looks like

$$\mathsf{unsat}_1(p_1,\ldots,p_n,\top) \to \mathsf{unsat}_0(p_1,\ldots,p_n,\ulcorner\varphi\urcorner) \tag{2}$$

Termination of the labeled system can then easily be shown by some basic method, e.g., LPO; choosing the precedence $\mathsf{unsat}_1 > \mathsf{unsat}_0, \star, -$ allows to orient rule (2) from left to right and $- > \bot, \top$ handles the rules in $\mathcal{S}\mathsf{imp}$. So $\mathcal{U}^{\varphi}_{\mathsf{lab}}$ is terminating. Theorem 2.1 yields the termination of $\mathcal{U}^{\varphi}$. □

[7] Labeling constants is superfluous and hence we implicitly set $L_{\bot} = L_{\top} = \varnothing$.

4 Transforming Satisfiability to Termination

In the previous section the task was somehow simpler since there it sufficed to construct a nonterminating sequence if there exists a satisfying assignment. Hence by guessing a satisfying assignment for φ one could construct an infinite sequence in the TRS $\mathcal{U}^{\varphi}$. In this section the endeavor is more challenging, because one has to guarantee that one cycles if *no* satisfying assignment exists. Hence, the parametrized TRS will have to test all assignments before entering a loop if none of them satisfied the formula φ. Thus we have to provide the possibility to generate all assignments successively. The following three rules, referred to as $\mathcal{N}\mathsf{ext}$ do this job by representing assignments as bitlists (consequently the signature $\mathcal{F}$ is extended by the binary function symbol $(::) : \mathsf{bool} \times \mathsf{list} \to \mathsf{list}$, the constant $\mathsf{nil} : \mathsf{list}$, and the unary function symbol $\mathsf{next} : \mathsf{list} \to \mathsf{list}$):

$$\begin{aligned} \mathsf{next}(\mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{next}(\bot :: xs) &\to \top :: xs \\ \mathsf{next}(\top :: xs) &\to \bot :: \mathsf{next}(xs) \end{aligned}$$

To ease notation we will encode lists over $\bot$ and $\top$ as natural numbers. Therefore, lists are interpreted as little endian representation of binary numbers where $\bot$ corresponds to 0 and $\top$ to 1. Let $\mathcal{G}$ be the signature $\{\bot, \top, ::, \mathsf{nil}\}$. The mapping $\mathsf{enc}\colon \mathcal{T}(\mathcal{G}) \to \mathbb{N} \times \mathbb{N}$, $\mathsf{enc}(\mathsf{nil}) = (0, 0)$ and $\mathsf{enc}(x :: xs) = (x + 2i, l + 1)$ where $\mathsf{enc}(xs) = (i, l)$, uniquely associates lists with entries $\bot$ or $\top$ to pairs. The first component of the pair is the little endian representation of the bitlist whereas the second component is the length of the list. For convenience we denote (i, l) by $\mathbf{i_l}$. Furthermore if l is irrelevant or fixed we feel free to omit it. Taking the above conventions into account the bitlist $[\top; \bot; \top; \top]$[8] can be written as $\mathbf{13_4}$ or more sloppily as $\mathbf{13}$. But we do not only identify these bitlists with natural numbers, they also encode substitutions. Hence, a bitlist $[t_1; \ldots; t_n]$ gives rise to a substitution σ with $\sigma(p_i) = t_i$ for $1 \leqslant i \leqslant n$. Using this convention a term t indexed with a bold face integer denotes the result of applying the substitution to the term, i.e., $(p_1 \star ((- p_2) \star p_3))_{\mathbf{13}}$ denotes $\top \star ((- \bot) \star \top)$.

Lemma 4.1 *For a bitlist t, $\mathsf{next}(t)$ rewrites to the successor of t:*

$$\textit{If } \mathsf{enc}(t) = \mathbf{i_l} \textit{ then } \mathsf{next}(t) \to^*_{\mathcal{N}\mathsf{ext}} t' \textit{ with } \mathsf{enc}(t') = (\mathbf{i + 1} \bmod \mathbf{2^l})_{\mathbf{l}}.$$

Proof By induction on the structure of t and unfolding the definition of $\mathbf{i_l}$. □

To proceed we explicitly state the function signature sig, i.e., the sort for each function symbol, in the left column of Table 2. In the theorem below the variables $p_1, \ldots, p_n$ are of sort bool and xs is of sort list.

[8] To ease readability, lists $x :: (y :: (z :: \mathsf{nil}))$ are abbreviated by $[x; y; z]$.

$\bot : \mathsf{bool}$	$\bot_{\mathcal{A}} = 0$
$\top : \mathsf{bool}$	$\top_{\mathcal{A}} = 1$
$\star : \mathsf{bool} \times \mathsf{bool} \to \mathsf{bool}$	$\star_{\mathcal{A}}(x, y) = x \cdot y$
$- : \mathsf{bool} \to \mathsf{bool}$	$-_{\mathcal{A}}(x) = \overline{x}$
$\mathsf{nil} : \mathsf{list}$	$\mathsf{nil}_{\mathcal{A}} = (0, 0)$
$:: : \mathsf{bool} \times \mathsf{list} \to \mathsf{list}$	$::_{\mathcal{A}}(x, (i, l)) = (x + 2i, l + 1)$
$\mathsf{next} : \mathsf{list} \to \mathsf{list}$	$\mathsf{next}_{\mathcal{A}}((i, l)) = (i + 1 \bmod 2^l, l)$
$\mathsf{sat} : \mathsf{list} \times \mathsf{bool} \to \mathsf{bool}$	$\mathsf{sat}_{\mathcal{A}}((i, l), b) = 0$

Table 2
A model for the $\{\mathsf{bool}, \mathsf{list}\}$-sorted TRS $\mathcal{S}^{\varphi}$.

Theorem 4.2 *Let $\varphi \in \mathcal{P}(\mathcal{A}_n)$. Then the parametrized $\{\mathsf{bool}, \mathsf{list}\}$-sorted TRS $\mathcal{S}^{\varphi}$ that contains all rules of $\mathcal{S}\mathsf{imp}$, $\mathcal{N}\mathsf{ext}$, and additionally*

$$\mathsf{sat}([p_1; \ldots; p_n], \bot) \to \mathsf{sat}(\mathsf{next}([p_1; \ldots; p_n]), \ulcorner\varphi\urcorner) \tag{3}$$

is terminating if and only if the formula φ is satisfiable.

Proof For the direction from left to right assume for the sake of a contradiction unsatisfiability of φ. The cyclic reduction

$$\begin{aligned}&\mathsf{sat}(\mathbf{0}, \bot) \to \\ &\quad \mathsf{sat}(\mathsf{next}(\mathbf{0}), \ulcorner\varphi\urcorner_{\mathbf{0}}) \to^* \mathsf{sat}(\mathbf{1}, \ulcorner\varphi\urcorner_{\mathbf{0}}) \to^* \mathsf{sat}(\mathbf{1}, \bot) \to^* \cdots \to^* \\ &\quad \mathsf{sat}(\mathsf{next}(\mathbf{2^n - 1}), \ulcorner\varphi\urcorner_{\mathbf{2^n-1}}) \to^* \mathsf{sat}(\mathbf{0}, \ulcorner\varphi\urcorner_{\mathbf{2^n-1}}) \to^* \mathsf{sat}(\mathbf{0}, \bot)\end{aligned}$$

proves nontermination of $\mathcal{S}^{\varphi}$ where $\mathsf{next}(\mathbf{i_n}) \to^*_{\mathcal{N}\mathsf{ext}} (\mathbf{i + 1} \bmod \mathbf{2^n})_{\mathbf{n}}$ follows from Lemma 4.1 and since we assumed that φ is unsatisfiable $\ulcorner\varphi\urcorner_{\mathbf{i_n}} \to^*_{\mathcal{S}\mathsf{imp}} \bot$ by Lemma 3.1, for all $0 \leqslant i < 2^n$.

For the direction from right to left we will again give a proof using semantic labeling. The difference this time is that we exploit the many-sorted version of semantic labeling. Now for every sort $s \in \{\mathsf{bool}, \mathsf{list}\}$ we have to specify a carrier. The choices are $A_{\mathsf{bool}} = \mathbb{B}$ and $A_{\mathsf{list}} = P := \{(i, l) \in \mathbb{N} \times \mathbb{N} \mid i < 2^l\}$. Then the interpretation in the right column of Table 2 is a model for $\mathcal{S}^{\varphi}$. We show this for the $\mathcal{N}\mathsf{ext}$-rules. Let us fix an arbitrary value $(i, l) \in P$ for xs. The three $\mathcal{N}\mathsf{ext}$-rules generate the three equalities

$$(0 + 1 \bmod 2^0, 0) = (0, 0) \tag{4}$$

$$((0 + 2i) + 1 \bmod 2^{l+1}, l + 1) = (1 + 2i, l + 1) \tag{5}$$

$$((1 + 2i) + 1 \bmod 2^{l+1}, l + 1) = (0 + 2(i + 1 \bmod 2^l), l + 1) \tag{6}$$

Equation (4) is trivially valid. Since $i < 2^l$ by definition of P equation (5) holds since the modulo operation can be omitted. Validity of equation (6) is shown

by case distinction. For $i = 2^l - 1$ it simplifies to $1 + 2(2^l - 1) + 1 \bmod 2^{l+1} = 0 + 2(2^l - 1 + 1 \bmod 2^l)$ where both sides equal 0. For the other case we know that $i < 2^l - 1$ and consequently $2i + 2 < 2^{l+1}$. Hence, the modulo operation does no harm and both sides evaluate to the same value.

The following sets of labels are employed: $L_\star = L_- = L_{::} = L_{\mathsf{next}} = \varnothing$ and $L_{\mathsf{sat}} = \mathbb{N} \times \mathbb{B}$. Then, the labeling function $\ell_{\mathsf{sat}} \colon P \times \mathbb{N} \to \mathbb{N} \times \mathbb{B}$ with $\ell_{\mathsf{sat}}((i,l),b) = (i,b)$ is used which produces the following labeled variants of rule (3)

$$\mathsf{sat}_{i,0}([p_1;\ldots;p_n],\bot) \to \mathsf{sat}_{(i+1 \bmod 2^n),\varphi_{\mathbf{i_n}}}(\mathsf{next}([p_1;\ldots;p_n]),\ulcorner\varphi\urcorner) \tag{7}$$

where $0 \leqslant i < 2^n$. In the right-hand side of the generic rule (7) the expression $\varphi_{\mathbf{i_n}}$ means that φ is evaluated by the assignment corresponding to the bitlist $\mathbf{i_n}$.

If for at least one assignment φ evaluates to 1 then the system can be proved terminating. Assume that the j-th assignment satisfies φ. Then the precedence

$$\begin{aligned}
&\mathsf{sat}_{(j+1),0} > \mathsf{sat}_{(j+2),0} > \cdots > \mathsf{sat}_{(2^n-1),0} > \mathsf{sat}_{0,0} > \mathsf{sat}_{1,0} > \cdots > \mathsf{sat}_{j,0}\\
&\quad \mathsf{sat}_{i,0} > \mathsf{sat}_{(i+1 \bmod 2^n),1} \quad (0 \leqslant i < 2^n)\\
&\quad \mathsf{sat}_{j,0} > \mathsf{next}, \star, - > \bot, \top
\end{aligned}$$

is well-founded and allows LPO to orient all rules of the labeled TRS $\mathcal{S}^\varphi_{\mathsf{lab}}$ from left to right. Termination of $\mathcal{S}^\varphi$ follows from Theorem 2.1. □

As an example consider the transformation of the formula $p_1 \wedge \neg p_2$ below.

Example 4.3 The system $\mathcal{S}^{p_1 \wedge \neg p_2}$ gives rise to the labeled rules

$$\begin{aligned}
\mathsf{sat}_{0,0}([p_1;p_2],\bot) &\to \mathsf{sat}_{1,0}(\mathsf{next}([p_1;p_2]),p_1 \star (-p_2))\\
\mathsf{sat}_{1,0}([p_1;p_2],\bot) &\to \mathsf{sat}_{2,1}(\mathsf{next}([p_1;p_2]),p_1 \star (-p_2))\\
\mathsf{sat}_{2,0}([p_1;p_2],\bot) &\to \mathsf{sat}_{3,0}(\mathsf{next}([p_1;p_2]),p_1 \star (-p_2))\\
\mathsf{sat}_{3,0}([p_1;p_2],\bot) &\to \mathsf{sat}_{0,0}(\mathsf{next}([p_1;p_2]),p_1 \star (-p_2))
\end{aligned}$$

Note that because in the second line the term $(p_1 \star (-p_2))_{\mathbf{1}}$ is interpreted as 1 and hence the system can easily be proved terminating by LPO with the precedence

$$\mathsf{sat}_{2,0} > \mathsf{sat}_{3,0} > \mathsf{sat}_{0,0} > \mathsf{sat}_{1,0} > \mathsf{sat}_{2,1}, \mathsf{next}, \star, - \quad - > \bot, \top$$

In this translation the TRS S^φ_{lab} gets exponentially larger (in the number of variables in φ) than the original unlabeled system. More precisely, rule (3) gives rise to 2^n different labeled variants due to the n Boolean variables in the list $[p_1;\ldots;p_n]$. But the resulting TRS is still finite, in contrast to the one from the next subsection.

4.1 An Alternative Transformation

In the transformation $\mathcal{S}^\varphi$ the formula φ gets assigned the values implicitly by pattern matching because the same variables $p_1,\ldots,p_n$ are used in the formula and in the assignment. One not so nice side-effect is that in rule (3) the list of variables

occurring in φ must be specified as the first argument to sat. Here we present a different translation where the variables $p_1, \ldots, p_n$ are considered as constants $\mathsf{v_1}, \ldots, \mathsf{v_n}$ in the signature $\mathcal{F}$. For terms that represent formulas on the syntactic level, the sort $\mathsf{formula}$ is used, i.e., $\mathsf{v_i} : \mathsf{formula}$ for $1 \leqslant i \leqslant n$. Furthermore a close inspection of the systems $\mathcal{U}^\varphi$ and $\mathcal{S}^\varphi$ reveals that there is no clear separation between syntax and semantics when formulas are represented as terms. To differentiate these two concepts we employ different function symbols for the two layers. Once more the signature $\mathcal{F}$ is augmented by a binary function symbol $\mathsf{and} : \mathsf{formula} \times \mathsf{formula} \to \mathsf{formula}$ and a unary function symbol $\mathsf{not} : \mathsf{formula} \to \mathsf{formula}$. Consequently also the encoding $\ulcorner * \urcorner$ must now map formulas to their syntactic representation on the term level. Hence the function $\ulcorner * \urcorner$ is redefined accordingly, i.e., $\ulcorner * \urcorner \colon \mathcal{P}(\mathcal{A}_n) \to \mathcal{T}(\{\mathsf{v_1}, \ldots, \mathsf{v_n}, \mathsf{and}, \mathsf{not}\})$ with $\ulcorner p_i \urcorner = \mathsf{v_i}$ for $1 \leqslant i \leqslant n$, $\ulcorner \varphi \wedge \psi \urcorner = \mathsf{and}(\ulcorner \varphi \urcorner, \ulcorner \psi \urcorner)$, and $\ulcorner \neg \varphi \urcorner = \mathsf{not}(\ulcorner \varphi \urcorner)$. Thus, for the formula $p_1 \wedge \neg p_2$ the (syntactic) term representation $\ulcorner \varphi \urcorner$ is $\mathsf{and}(\mathsf{v_1}, \mathsf{not}(\mathsf{v_2}))$.

In the TRS $\mathcal{S}^\varphi$ the assignment was applied automatically by pattern matching of the variables. Now we employ separate rewrite rules that perform that step. Note that these rules at the same time execute the transformation from the syntactic to the semantic level. The TRS $\mathcal{A}\mathsf{ssign}$

$$
\begin{aligned}
\mathsf{assign}(xs, \mathsf{and}(x, y)) &\to \mathsf{assign}(xs, x) \star \mathsf{assign}(xs, y) \\
\mathsf{assign}(xs, \mathsf{not}(x)) &\to - \mathsf{assign}(xs, x) \\
\mathsf{assign}(xs, \mathsf{v_i}) &\to \mathsf{nth}(xs, \mathsf{s}^i(\mathsf{0})) \qquad 1 \leqslant i \leqslant n \\
\mathsf{nth}(\bot :: xs, \mathsf{0}) &\to \bot \\
\mathsf{nth}(\top :: xs, \mathsf{0}) &\to \top \\
\mathsf{nth}(b :: xs, \mathsf{s}(j)) &\to \mathsf{nth}(xs, j)
\end{aligned}
$$

performs the task of $[\alpha]$ on the term representation of propositional formulas. The way how assignments were generated in the previous subsection is no longer suitable. There all variables occurring in φ had to be specified in sat's first argument. Since we want to get rid of that requirement the idea is to start with an empty assignment (empty list) and increase its length repeatedly. Hence in this section the assignments are no longer computed modulo some length but the *overflow* is simply taken into account by increasing the length of the list. The three rules below are referred to as the TRS $\mathcal{N}\mathsf{ext2}$:

$$
\begin{aligned}
\mathsf{next}(\mathsf{nil}) &\to \top :: \mathsf{nil} \\
\mathsf{next}(\bot :: xs) &\to \top :: xs \\
\mathsf{next}(\top :: xs) &\to \bot :: \mathsf{next}(xs)
\end{aligned}
$$

Similar to before a more readable notation for bitlists is employed, i.e., they are identified with natural numbers as follows: $\mathsf{enc} \colon \mathcal{T}(\mathcal{G}) \to \mathbb{N}$ with $\mathsf{enc}(\mathsf{nil}) = \mathsf{enc}(\bot) = 0$, $\mathsf{enc}(\top) = 1$, and $\mathsf{enc}(x :: xs) = \mathsf{enc}(x) + 2\,\mathsf{enc}(xs)$. This encoding is not injective because the lists $[\top; \bot; \bot]$ and $[\top]$ are both denoted by $\mathbf{1}$. In our setting these (more or less) leading zeros do not pose a problem.

$\bot : \mathsf{bool}$	$\top_{\mathcal{A}} = 1$
$\top : \mathsf{bool}$	$\bot_{\mathcal{A}} = 0$
$\star : \mathsf{bool} \times \mathsf{bool} \to \mathsf{bool}$	$\star_{\mathcal{A}}(x, y) = x \cdot y$
$- : \mathsf{bool} \to \mathsf{bool}$	$-_{\mathcal{A}}(x) = \overline{x}$
$\mathsf{nil} : \mathsf{list}$	$\mathsf{nil}_{\mathcal{A}} = 0$
$:: : \mathsf{bool} \times \mathsf{list} \to \mathsf{list}$	$::_{\mathcal{A}}(x, i)) = x + 2i$
$\mathsf{next} : \mathsf{list} \to \mathsf{list}$	$\mathsf{next}_{\mathcal{A}}(i) = i + 1$
$\mathsf{v}_i : \mathsf{formula} \quad 1 \leqslant i \leqslant n$	$\mathsf{v}_{i\mathcal{A}} = p_i$
$\mathsf{and} : \mathsf{formula} \times \mathsf{formula} \to \mathsf{formula}$	$\mathsf{and}_{\mathcal{A}}(x, y) = x \wedge y$
$\mathsf{not} : \mathsf{formula} \to \mathsf{formula}$	$\mathsf{not}_{\mathcal{A}}(x) = \neg x$
$0 : \mathsf{nat}$	$0_{\mathcal{A}} = 0$
$\mathsf{s} : \mathsf{nat} \to \mathsf{nat}$	$\mathsf{s}_{\mathcal{A}}(x) = x + 1$
$\mathsf{assign} : \mathsf{list} \times \mathsf{formula} \to \mathsf{bool}$	$\mathsf{assign}_{\mathcal{A}}(i, \varphi) = [\alpha_i](\varphi)$
$\mathsf{nth} : \mathsf{list} \times \mathsf{nat} \to \mathsf{bool}$	$\mathsf{nth}_{\mathcal{A}}(i, j) = \alpha_i(p_j)$
$\mathsf{sat} : \mathsf{list} \times \mathsf{bool} \to \mathsf{bool}$	$\mathsf{sat}_{\mathcal{A}}(i, b) = 0$

Table 3
A model for the $\{\mathsf{bool}, \mathsf{formula}, \mathsf{list}, \mathsf{nat}\}$-sorted TRS $\mathcal{T}^{\varphi}$.

Lemma 4.4 *For a bitlist t, $\mathsf{next}(t)$ rewrites to the successor of t:*

$$\textit{If } \mathsf{enc}(t) = \mathbf{i} \textit{ then } \mathsf{next}(t) \to^*_{\mathcal{N}\mathsf{ext2}} t' \textit{ with } \mathsf{enc}(t') = \mathbf{i} + \mathbf{1}.$$

Proof By induction on the structure of t and unfolding the definition of $\mathbf{i}$. □

The desired property that the rules in $\mathcal{A}\mathsf{ssign}$ evaluate the term representation $\ulcorner\varphi\urcorner$ for a given bitlist $\mathbf{i}$ is formalized in the lemma below.

Lemma 4.5 *Let $\varphi \in \mathcal{P}(\mathcal{A}_n)$ and let $\mathbf{i}$ be the encoding of an assignment α with $[\alpha](\varphi) = 0$. Then $\mathsf{assign}(\mathbf{i}, \ulcorner\varphi\urcorner) \to^*_{\mathcal{A}\mathsf{ssign} \cup \mathcal{S}\mathsf{imp}} \bot$.*

Proof By induction on the structure of $\ulcorner\varphi\urcorner$ and unfolding the definition of $\mathbf{i}$. □

Now, we will establish a theorem similar to Theorem 4.2. Again, we prove the theorem in a many-sorted setting. The full information is depicted in Table 3. The variables in the TRS are associated to sorts as follows: $b \in \mathcal{V}_{\mathsf{bool}}$, $xs \in \mathcal{V}_{\mathsf{list}}$, $j \in \mathcal{V}_{\mathsf{nat}}$, and x, $y \in \mathcal{V}_{\mathsf{formula}}$.

Theorem 4.6 *Let $\varphi \in \mathcal{P}(\mathcal{A}_n)$. Then the parametrized $\{\mathsf{bool}, \mathsf{formula}, \mathsf{list}, \mathsf{nat}\}$-sorted TRS $\mathcal{T}^{\varphi}$ consisting of the $\mathcal{S}$imp-, $\mathcal{N}$ext2-, and $\mathcal{A}$ssign-rules plus additionally*

$$\mathsf{sat}(xs, \bot) \to \mathsf{sat}(\mathsf{next}(xs), \mathsf{assign}(xs, \ulcorner\varphi\urcorner)) \tag{8}$$

is terminating if and only if φ is satisfiable.

Proof Concerning the direction from left to right one can again construct a nonterminating reduction for any unsatisfiable formula φ. In order not to get stuck while evaluating $\mathsf{assign}(\mathbf{i}, \ulcorner\varphi\urcorner)$ a sufficiently large $\mathbf{i}$ is taken (e.g., $\mathbf{i} = 2^{n+1}$). Then there is the infinite sequence

$$\begin{aligned}\mathsf{sat}(\mathbf{i}, \bot) &\to \mathsf{sat}(\mathsf{next}(\mathbf{i}), \mathsf{assign}(\mathbf{i}, \ulcorner\varphi\urcorner)) \to^* \\ &\mathsf{sat}(\mathbf{i}+\mathbf{1}, \bot) \to \mathsf{sat}(\mathsf{next}(\mathbf{i}+\mathbf{1}), \mathsf{assign}(\mathbf{i}+\mathbf{1}, \ulcorner\varphi\urcorner)) \to^* \\ &\mathsf{sat}(\mathbf{i}+\mathbf{2}, \bot) \to \mathsf{sat}(\mathsf{next}(\mathbf{i}+\mathbf{2}), \mathsf{assign}(\mathbf{i}+\mathbf{2}, \ulcorner\varphi\urcorner)) \to^* \cdots\end{aligned}$$

where the $\to$-steps are applications of rule 8 and the $\to^*$-steps can be performed because of Lemmata 4.4 and 4.5.

For the direction from right to left once more a semantic labeling approach is followed. The interpretation given in Table 3 models the $\{\mathsf{bool}, \mathsf{formula}, \mathsf{list}, \mathsf{nat}\}$-sorted TRS $\mathcal{T}^\varphi$. What remains to be defined is an enumeration α_i of assignments as follows: $\alpha_i(p_j) = f^j(i) \bmod 2$ with $f^0(i) = i$ and $f^{j+1}(i) = f^j(\lceil i \div 2 \rceil)$. Checking that $\mathcal{A}$ models $\mathcal{T}^\varphi$ is straightforward.

Again, only the function symbol sat is labeled. Note that the labeled TRS T^φ_{lab} is infinite since all possible instances of bitlists are considered (compared to finitely many bitlists of a specified length in the previous subsection). The labeling function $\ell_{\mathsf{sat}}(i, b) = (i, b)$ gives rise to infinitely many rules of the following shape

$$\mathsf{sat}_{i,0}(xs, \bot) \to \mathsf{sat}_{(i+1),[\alpha_i](\varphi)}(\mathsf{next}(xs), \mathsf{assign}(xs, \ulcorner\varphi\urcorner))$$

Similar to before a precedence of the shape $\mathsf{sat}_{i,0} > \mathsf{sat}_{(i+1),0}$ if $[\alpha_i](\varphi) = 0$ and $\mathsf{sat}_{i,0} > \mathsf{sat}_{(i+1),1}$ if $[\alpha_i](\varphi) = 1$ for all $i \geqslant 0$ is needed which in general might not be well-founded since it can contain the infinite sequence

$$\mathsf{sat}_{0,0} > \mathsf{sat}_{1,0} > \mathsf{sat}_{2,0} > \dots$$

but due to the assumption that φ is satisfiable, not all of these precedence comparisons are necessary. If $[\alpha_j](\varphi) = 1$ then there is no labeled rule which demands $\mathsf{sat}_{j,0} > \mathsf{sat}_{(j+1),0}$. Without loss of generality we can assume $0 \leqslant j < 2^n$. Due to the construction of α_j also α_{j+2^n}, $\alpha_{j+2^{n+1}}$, $\dots$ satisfy φ and hence removing all superfluous comparisons $\mathsf{sat}_{(j+2^{n+m}),0} > \mathsf{sat}_{(j+2^{n+m}+1),0}$ for all $m \in \mathbb{N}$ produces a well-founded precedence (because for any $i \in \mathbb{N}$ one can find a $k \in \mathbb{N}$ such that $i \leqslant j + 2^{n+k}$). It follows that $\mathcal{T}^\varphi_{\mathsf{lab}}$ is terminating. The termination of $\mathcal{T}^\varphi$ follows from Theorem 2.1. □

Although the transformations $\mathcal{S}^\varphi$ and $\mathcal{T}^\varphi$ look very similar at first, they are quite different. Concerning the number of rewrite rules, $\mathcal{S}^\varphi$ does not depend on φ whereas $\mathcal{T}^\varphi$ depends linearly on the number of variables in φ. On the other hand, the list of variables $p_1, \dots, p_n$ must be given as an argument to sat in $\mathcal{S}^\varphi$. In Section 5 it becomes apparent that proving (non)termination automatically is much more challenging for $\mathcal{T}^\varphi$ than for $\mathcal{S}^\varphi$. The main reason is that by separating syntax from semantics, there is less structure that can be exploited by termination tools. The nontermination proofs become more challenging because for $\mathcal{S}^\varphi$ an infinite rewrite

tool	2 variables, depth 3			3 variables, depth 4			4 variables, depth 5		
	$\mathcal{S}^\varphi$	$\mathcal{T}^\varphi$	$\mathcal{U}^\varphi$	$\mathcal{S}^\varphi$	$\mathcal{T}^\varphi$	$\mathcal{U}^\varphi$	$\mathcal{S}^\varphi$	$\mathcal{T}^\varphi$	$\mathcal{U}^\varphi$
	T/ N	T/N	T/ N	T/N	T/N	T/ N	T/N	T/N	T/ N
AProVE	81/19	0/ 0	19/81	34/ 0	0/ 0	10/88	14/ 0	0/ 0	5/79
Jambox	16/ 0	0/ 0	19/ 0	24/ 0	0/ 0	12/ 0	15/ 0	0/ 0	11/ 0
NTI	0/19	0/ 0	0/81	0/ 5	0/ 0	0/74	0/ 0	0/ 0	0/11
TPA	0/ 0	0/ 0	1/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0	0/ 0
TTT2	10/ 0	0/ 0	0/ 0	6/ 0	0/ 0	0/ 0	5/ 0	0/ 0	0/ 0

Table 4
Experimental Results.

sequence can be captured by considering cyclic reductions of ground terms, i.e., $t \to^+ t$ for a ground term t (cf. the proof of Theorem 4.2). In contrast $\mathcal{T}^\varphi$ really demands looping reductions, i.e., $t \to^+ C[t\sigma]$ where the context C is empty but t may no longer be ground since the lengths of the bitlists are increased.

5 Evaluation

For experimental results [9] we considered all automated (non)termination analyzers that participated in the 2007 edition of the international termination competition for term rewrite systems augmented with TPA [15], a tool with strong support for termination proofs via semantic labeling. To our knowledge none of these tools supports analysis of sorted TRSs. Consequently we provide our examples unsorted. As already stated in the beginning, dropping sorts does not affect termination of the TRSs we propose. Furthermore we stress that the proofs of Theorems 4.2 and 4.6 can be modified to work on unsorted TRSs. For the TRS $\mathcal{S}^\varphi$ this means that the interpretations range over the set of pairs P whereas the proof of Theorem 4.6 can be generalized to one sort by using the natural numbers as a carrier and representing formulas via a Gödel encoding [11].

It turned out that even for rather small formulas (some of) our transformations produce rewrite systems whose termination analysis is challenging. We considered 100 randomly generated formulas of different shapes. Table 4 summarizes the results, e.g., formulas of depth three using two different propositional variables are considered in the leftmost block, etc. Every tool was run on all TRSs resulting from transforming the formula φ to $\mathcal{S}^\varphi$, $\mathcal{T}^\varphi$, and $\mathcal{U}^\varphi$ for at most 60 seconds to analyze termination (T) or nontermination (N) of each system. Globally speaking, for TRSs originating from very small formulas AProVE [9] performs best. This is due to its support for narrowing which allows to exploit the structure of $\mathcal{S}^\varphi$ and $\mathcal{U}^\varphi$. Jambox solves some instances by semantic labeling over Boolean models (which is very close to the way how we proved termination) and by the matrix method. The latter

[9] Further details to be found at http://colo6-c703.uibk.ac.at/ttt2/hz/sat2trs/.

systems could also be handled by T$_{\text{T}}$T$_2$.[10] NTI [20] supports only nontermination analysis, using an unfolding operator. Semantic labeling based on Boolean models and (quasi-)models over the naturals is implemented in TPA [17,16] which usually performs very well on standard examples. The experiments reveal that the latter is not powerful for the systems obtained from the transformations proposed in this paper.

But narrowing is expensive which can be seen by comparing the different blocks of Table 4. AProVE can handle all TRSs resulting from the $\mathcal{S}^\varphi$ translation if formulas are of depth three but for depth four (five) the performance decreases to 34% (14%). For the other translations the effect is not so tremendous, well, for $\mathcal{T}^\varphi$ the surprising outcome is that no tool could handle any system at all and the systems in $\mathcal{U}^\varphi$ are generally a bit easier since they do not iterate over the assignments. Needless to say, the formulas φ which are considered for our experiments are a very trivial task for any SAT-solver.

We conclude this section by a sketch of how AProVE solves many instances by considering the TRS S^φ for $\varphi = p_1 \wedge p_2$. After some preliminary analysis based on dependency pairs [2], AProVE concludes that any infinite sequence applies the rule

$$\mathsf{sat}([p_1; p_2], \bot) \to \mathsf{sat}(\mathsf{next}([p_1; p_2]), p_1 \star p_2)$$

indefinitely. Narrowing the above rule at position 1 allows to replace it by the two rules

$$\begin{aligned}\mathsf{sat}([\bot; p_2], \bot) &\to \mathsf{sat}(\mathsf{next}([\bot; p_2]), \bot \star p_2)\\ \mathsf{sat}([\top; p_2], \bot) &\to \mathsf{sat}(\mathsf{next}([\top; p_2]), \top \star p_2)\end{aligned}$$

and narrowing these rules at position 1 gives

$$\begin{aligned}\mathsf{sat}([\bot; \bot], \bot) &\to \mathsf{sat}(\mathsf{next}([\bot; \bot]), \bot \star \bot)\\ \mathsf{sat}([\bot; \top], \bot) &\to \mathsf{sat}(\mathsf{next}([\bot; \top]), \bot \star \top)\\ \mathsf{sat}([\top; \bot], \bot) &\to \mathsf{sat}(\mathsf{next}([\top; \bot]), \top \star \bot)\\ \mathsf{sat}([\top; \top], \bot) &\to \mathsf{sat}(\mathsf{next}([\top; \top]), \top \star \top)\end{aligned}$$

After this state is reached the right-hand sides can be rewritten [10] using the $\mathcal{S}$imp and $\mathcal{N}$ext rules which allows the dependency graph processor [2] to conclude termination.

6 Conclusion

In this paper we proposed three different transformations from propositional formulas φ to confluent—since orthogonal—term rewrite systems $\mathcal{S}^\varphi$, $\mathcal{T}^\varphi$, and $\mathcal{U}^\varphi$ such that φ is satisfiable (unsatisfiable) if and only if $\mathcal{S}^\varphi$, $\mathcal{T}^\varphi$ ($\mathcal{U}^\varphi$) is terminating. Although the systems can be proved (non)terminating by semantic labeling using

[10] http://colo6-c703.uibk.ac.at/ttt2/

intuitive models, state-of-the-art termination tools fail even on very small and simple TRSs. Especially the transformation $\mathcal{T}^{\varphi}$ produces unsolvable rewrite systems which might be due to the fact that it preserves much less structure than $\mathcal{S}^{\varphi}$ does. If tool authors investigate the reasons why the generated problems are that hard, new termination techniques could emerge.

References

[1] Aoto, T. and T. Yamada, *Termination of simply typed term rewriting by translation and labelling*, in: *Proc. 14th International Conference on Rewriting Techniques and Applications*, LNCS **2706**, 2003, pp. 380–394.

[2] Arts, T. and J. Giesl, *Termination of term rewriting using dependency pairs*, Theoretical Computer Science **236** (2000), pp. 133–178.

[3] Baader, F. and T. Nipkow, "Term Rewriting and All That," Cambridge University Press, 1998.

[4] Codish, M., V. Lagoon and P. Stuckey, *Solving partial order constraints for LPO termination*, in: *Proc. 17th International Conference on Rewriting Techniques and Applications*, LNCS **4098**, 2006, pp. 4–18.

[5] Cook, S., *The complexity of theorem-proving procedures*, in: *Proc. 3rd annual ACM symposium on theory of computing* (1971), pp. 151–158.

[6] Dershowitz, N., *Orderings for term-rewriting systems*, Theoretical Computer Science **17** (1982), pp. 279–301.

[7] Dick, J., J. Kalmus and U. Martin, *Automating the Knuth-Bendix ordering*, Acta Informatica **28** (1990), pp. 95–119.

[8] Endrullis, J., J. Waldmann and H. Zantema, *Matrix interpretations for proving termination of term rewriting*, Journal of Automated Reasoning **40** (2008), pp. 195–220.

[9] Giesl, J., P. Schneider-Kamp and R. Thiemann, *AProVE 1.2: Automatic termination proofs in the dependency pair framework*, in: *Proc. 3rd International Joint Conference on Automated Reasoning*, LNAI **4130**, 2006, pp. 281–286.

[10] Giesl, J., R. Thiemann and P. Schneider-Kamp, *The dependency pair framework: Combining techniques for automated termination proofs*, in: *Proc. 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LNAI **3452**, 2005, pp. 301–331.

[11] Gödel, K., *Über formal unentscheidbare sätze der principia mathematica und verwandter systeme*, Monatshefte für Mathematik und Physik **38** (1931), pp. 173–198.

[12] Huet, G. and D. Lankford, *On the uniform halting problem for term rewriting systems*, Technical Report 282, INRIA, Le Chesnay, France (1978).

[13] Kamin, S. and J. Lévy, *Two generalizations of the recursive path ordering*, Unpublished manuscript, University of Illinois (1980).

[14] Knuth, D. and P. Bendix, *Simple word problems in universal algebras*, in: J. Leech, editor, *Computational Problems in Abstract Algebra*, Pergamon Press, 1970 pp. 263–297.

[15] Koprowski, A., *TPA: Termination proved automatically.*, in: *Proc. 17th International Conference on Rewriting Techniques and Applications*, LNCS **4098**, 2006, pp. 257–266.

[16] Koprowski, A. and A. Middeldorp, *Predictive labeling with dependency pairs using SAT*, in: *Proc. 21st International Conference on Automated Deduction*, LNAI **4603**, 2007, pp. 410–425.

[17] Koprowski, A. and H. Zantema, *Automation of recursive path ordering for infinite labelled rewrite systems*, in: *Proc. 3rd International Joint Conference on Automated Reasoning*, LNAI **4130**, 2006, pp. 332–346.

[18] Korovin, K. and A. Voronkov., *Orienting rewrite rules with the Knuth-Bendix order*, Information and Computation **183** (2003), pp. 165–186.

[19] Kurihara, M. and H. Kondo, *Efficient BDD encodings for partial order constraints with application to expert systems in software verification*, in: *Proc. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, LNAI **3029**, 2004, pp. 827–837.

[20] Payet, É., *Detecting non-termination of term rewriting systems using an unfolding operator*, in: *Proc. 11th International Symposium on Logic-Based Program Synthesis and Transformation*, LNCS **4407**, 2007, pp. 194–209.

[21] Waldmann, J., *Matchbox: A tool for match-bounded string rewriting*, in: *Proc. 15th International Conference on Rewriting Techniques and Applications*, LNCS **3091**, 2004, pp. 85–94.

[22] Zankl, H. and A. Middeldorp, *Satisfying KBO constraints*, in: *Proc. 18th International Conference on Rewriting Techniques and Applications*, LNCS **4533**, 2007, pp. 389–403.

[23] Zantema, H., *Termination of term rewriting: Interpretation and type elimination*, Journal of Symbolic Computation **17** (1994), pp. 23–50.

[24] Zantema, H., *Termination of term rewriting by semantic labelling*, Fundamenta Informaticae **24** (1995), pp. 89–105.