Journal on Satisfiability, Boolean Modeling and Computation 5 (2008) 193–215

Solving Partial Order Constraints for LPO Termination

Michael Codish mcodish@cs.bgu.ac.il
Department of Computer Science,
Ben-Gurion University, Israel

Vitaly Lagoon lagoon@cs.mu.oz.au
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia

Peter J. Stuckey pjs@cs.mu.oz.au
National ICT Australia, Victoria Laboratory,
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia

Abstract

This paper introduces a propositional encoding for lexicographic path orders (LPOs) and the corresponding LPO termination property of term rewrite systems. Given this encoding, termination analysis can be performed using a state-of-the-art Boolean satisfiability solver. Experimental results are unequivocal, indicating orders of magnitude speedups in comparison with previous implementations for LPO termination. The results of this paper have already had a direct impact on the design of several major termination analyzers for term rewrite systems. The contribution builds on a symbol-based approach towards reasoning about partial orders. The symbols in an unspecified partial order are viewed as variables that take integer values and are interpreted as indices in the order. For a partial order statement on n symbols, each index is represented in $\lceil \log_2 n \rceil$ propositional variables and partial order constraints between symbols are modeled on the bit representations. The proposed encoding is general and relevant to other applications which involve propositional reasoning about partial orders.

KEYWORDS: *Boolean satisfiability, lexicographic path orders, proving termination, term rewrite systems*

Submitted February 2007; revised May 2008; published June 2008

1. Introduction

A term rewrite system is a collection of directed equations of the form $\ell \to r$ which computes on a given term by repeatedly replacing subterms that match the left side of a rule (ℓ) with corresponding instances of the right side of the rule (r). As a formalism, term rewrite systems have the full power of Turing machines. Termination is an important property of term rewrite systems which is, in general, undecidable. However, special cases are decidable, and over the years huge research efforts have been devoted to the study of such cases as well as to the development of termination proving techniques. In particular, a term

rewrite system terminates if there is a reduction order $\succ$ which orients the rules in the system. Namely, such that $\ell \succ r$ for each rule $\ell \to r$ in the system. There are many methods for defining such orders and one of the most studied is the lexicographic path order (LPO)[20, 10]. For details on term rewrite systems and their termination properties, see the paper by Baader and Nipkow [4] or the survey paper by Dershowitz and Jouannaud [11].

The increasing interest in automated termination analysis of term rewrite systems has led to an annual *International Competition of Termination Tools* [24] initiated in 2004.[1.] The rules of the competition allow competitors to apply their choice and combination of termination proving techniques within a time limit of 60 seconds. Prior to the results of this paper, techniques based on LPOs were considered too slow to be chosen with high priority in the strategies applied by the main tools participating in the competition.

This paper introduces a novel propositional encoding of LPOs and demonstrates that SAT solving can drastically speed up the solving for the related LPO termination problems. The key idea is that the encoding of a term rewrite system is satisfiable if and only if the system is LPO terminating and that each model of the encoding indicates a particular LPO which orients the rules in the system. We present unequivocal experimental results. Our approach surpasses in orders of magnitude the performance of previous implementations of LPO termination, such as those originally provided by the termination proving tools TTT [17, 32] and AProVE [15, 2], as well as a previously proposed propositional encoding for LPOs described in the paper by Kurihara and Kondo [23].

Deciding LPO termination for a term rewrite system $\mathcal{R}$ is about determining if there exists a partial order $>$ on the symbols occurring in $\mathcal{R}$ such that the induced LPO orients the rules in $\mathcal{R}$. This can be posed as a *partial order constraint* on the symbols. Partial order constraints are just like usual propositional formulae, except that propositions are atomic statements about a partial order on a finite set of symbols. For example, $(f = g) \wedge ((f > h) \vee (h > g))$ is a formula in this logic. It contains three atoms: $(f = g)$, $(f > h)$ and $(h > g)$. There are many other applications in computer science which involve reasoning about the satisfiability of partial order constraints. For example, in the verification of timed systems (e.g. [1]), and in the solving of scheduling and planning problems (e.g. [21]).

The results of this paper are obtained through an encoding of partial order constraints into propositional logic. A previous propositional encoding for LPO termination, considered in the paper by Kurihara and Kondo [23], is *atom-based*. It models the atoms in a partial order constraint as propositional variables. Then, propositional statements are added to encode the axioms of partial orders that the atoms are subject to. For a partial order constraint on n symbols, such encodings typically introduce $O(n^2)$ propositional variables and involve $O(n^3)$ propositional connectives to express the axioms. In contrast, we take a *symbol-based* approach modelling the symbols in a partial order constraint as integer values (in binary representation). For n symbols this requires $k = \lceil \log_2 n \rceil$ propositional variables for each symbol. The integer value of a symbol reflects its index in a total order extending the partial order. Constraints of the form $(f = g)$ or $(f > h)$ are then straightforward to encode in k-bit arithmetic and involve $O(\log n)$ connectives each.

This paper is an extended version of [6]. The recent papers [8] and [35] illustrate that the proposed underlying approach is directly applicable to the more powerful termination

1. See also http://www.lri.fr/~marche/termination-competition.

$not(gt(A,B)) \to ge(B,A)$
$not(ge(A,B)) \to gt(B,A)$
$not(or(A,B)) \to and(not(A), not(B))$
$not(and(A,B)) \to or(not(A), not(B))$
$and(A, or(B,C)) \to or(and(A,B), and(A,C))$
$and(or(B,C), A) \to or(and(B,A), and(C,A))$

Figure 1. An example term rewrite system: normalizing formulae with propositional connectives (and, or, not) and partial orders (gt and ge for $>$ and $\geq$).

proving technique based on dependency pairs and argument filterings [3]. The recent paper [26] shows that the proposed underlying approach is applicable to the more powerful recursive path order [9], which combines multi-set orders with LPOs with respect to permutations (instead of just left-to-right). These extensions involve more complex encodings, but basically the same kind of partial order and propositional constraint solving. In the past year, several additional papers [13, 14, 18, 37] have illustrated the huge potential in applying SAT solvers for other types of termination proving techniques for term rewrite systems. A common theme in all of these works is to represent (finite domain) integer variables as binary numbers in bit representation and to encode arithmetic constraints as Boolean functions on these representations. Results indicate uniformly that the SAT-based approach to proving termination is highly beneficial. At the time of writing, both TTT and AProVE apply SAT-based techniques for various types of termination analysis. Both systems have adopted techniques described in the preliminary version of this paper.

The rest of this paper is organized as follows: Section 2 provides a brief introduction to term rewrite systems and LPO termination. Sections 3 and 4 introduce partial order constraints and their symbol-based propositional encoding. Section 5 describes the encoding of the LPO termination problem to partial order constraints. Section 6 evaluates our SAT-based approach to LPO termination. Finally, we present related work and conclusions.

2. Preliminaries: Term Rewrite Systems and LPO Termination

A term rewrite system is a set of rules $\mathcal{R}$ of the form $\ell \to r$ where ℓ and r are terms constructed from given sets $\mathcal{F}$ and $\mathcal{V}$ of symbols and variables, respectively. There is an additional restriction that ℓ is not a variable and that r contains only variables also in ℓ. A rule $\ell \to r$ applies to a term t if a subterm s of t matches ℓ with some substitution σ (namely, $s = \ell\sigma$). The rule is applied by replacing the subterm s by $r\sigma$, resulting in a new term v. Such an application is called a rewrite step on t and denoted $t \to_{\mathcal{R}} v$. A derivation is a sequence of rewrite steps. A term rewrite system is said to be terminating if all of its derivations are finite. An example term rewrite system is depicted as Figure 1. An example derivation of this system is

$$\begin{aligned} not(and(gt(A,B), gt(B,A))) &\to_{\mathcal{R}} or(not(gt(A,B)), not(gt(B,A))) \to_{\mathcal{R}} \\ & or(ge(B,A), not(gt(B,A))) \to_{\mathcal{R}} or(ge(B,A), ge(A,B)) \end{aligned}$$

Termination of term rewrite systems is undecidable [19]. However, a term rewrite system terminates if there is a reduction order $\succ$ such that $\ell \succ r$ for each rule $\ell \to r$ in the system. A reduction order is an order that is well-founded, monotonic and stable (closed under contexts and substitutions). There are many methods for defining such orders. Many of them are based on so-called simplification orders and one of the most studied is the

lexicographic path order (LPO) [20, 10]. This section reviews the definition of LPOs and introduces the LPO termination problem.

We assume an algebra of terms $\mathcal{T}(\mathcal{F},\mathcal{V})$ constructed over a given finite set of symbols $\mathcal{F}$ and set of variables $\mathcal{V}$. Let $>_\mathcal{F}$ denote a (strict or non-strict) partial order on the symbols $\mathcal{F}$ (a so-called *precedence*) and let $\approx_\mathcal{F}$ denote the corresponding equivalence relation on symbols. A precedence $>_\mathcal{F}$ on the set of symbols $\mathcal{F}$ induces a lexicographic path order $\succ_{lpo}$ on the set of terms $\mathcal{T}(\mathcal{F},\mathcal{V})$.

Definition 1 (LPO [20])**.** *The lexicographic path order $\succ_{lpo}$ on terms induced by the partial order $>_\mathcal{F}$ is defined as $s = f(s_1,\ldots,s_n) \succ_{lpo} t$ if and only if one of the following holds:*

1. *$t = g(t_1,\ldots,t_m)$ and $s \succ_{lpo} t_j$ for all $1 \le j \le m$ and either*
   (i) $f >_\mathcal{F} g$ or (ii) $f \approx_\mathcal{F} g$ and $\langle s_1,\ldots,s_n\rangle \succ^{lex}_{lpo} \langle t_1,\ldots,t_m\rangle$; or
2. *$s_i \succ_{lpo} t$ or $s_i \sim t$ for some $1 \le i \le n$.*

Here $\sim$ denotes the equality of terms up to the equivalence of symbols ($\approx_\mathcal{F}$). Namely, $s \sim t$ if and only if either (a) $s{=}t$; or (b) $s = f(s_1,\ldots,s_n)$, $t = g(t_1,\ldots,t_n)$, $f \approx_\mathcal{F} g$, and $s_i \sim t_i$ for all $1 \le i \le n$. The lexicographic extension of $\succ_{lpo}$ to tuples of terms is denoted $\succ^{lex}_{lpo}$ and defined by:

$$\langle s_1,\ldots,s_n\rangle \succ^{lex}_{lpo} \langle t_1,\ldots,t_m\rangle \Leftrightarrow \quad n > 0 \wedge \left(m = 0 \;\vee\; \left(m > 0 \;\wedge \begin{pmatrix} s_1 \succ_{lpo} t_1 \quad \vee \\ (s_1 \sim t_1 \wedge \langle s_2,\ldots,s_n\rangle \succ^{lex}_{lpo} \langle t_2,\ldots,t_m\rangle) \end{pmatrix}\right)\right)$$

It simply compares the terms in lexicographic order, using $\succ_{lpo}$ as the base order.

Example 1. *Consider the rewrite rule $not(gt(A,B)) \to ge(B,A)$ from Figure 1 and assume a partial order $>_\mathcal{F}$ on its symbols such that $gt >_\mathcal{F} ge$. Let $\succ_{lpo}$ denote the corresponding induced lexicographic partial order. It follows from definition 1 that $not(gt(A,B)) \succ_{lpo} ge(B,A)$: By case 2 of the definition we have that $not(gt(A,B)) \succ_{lpo} ge(B,A)$ if $gt(A,B) \succ_{lpo} ge(B,A)$; by case 1(i) of the definition $gt(A,B) \succ_{lpo} ge(B,A)$ if $gt >_\mathcal{F} ge$ (which is assumed) and if $gt(A,B) \succ_{lpo} B$ and $gt(A,B) \succ_{lpo} A$. Finally, these last two proof obligations follow directly from case 2 of Definition 1.*

Example 1 is about testing $s \succ_{lpo} t$ for a given partial order $>_\mathcal{F}$. LPO termination is instead about the existence of a partial order $>_\mathcal{F}$ such that $s \succ_{lpo} t$ for the induced lexicographic path order.

The LPO termination problem is to determine for a given term rewrite system with function symbols $\mathcal{F}$, if there exists a partial order $>_\mathcal{F}$ such that the induced lexicographic path order orients all of the rules in the system. Namely, such that $\ell \succ_{lpo} r$ for each $\ell \to r$ in the system. In this case the system is said to be LPO terminating. There are two variants of the problem: "strict-" and "quasi-LPO termination" depending on if the search for the partial order $>_\mathcal{F}$ is further restricted to find a strict partial order or not. Both imply termination of the corresponding term rewrite system. Quasi-LPO termination is the stronger property: if a system is LPO-terminating then it is also quasi-LPO terminating but the inverse does not hold. Quasi-LPO termination is also the harder problem, as the search for a non-strict partial order on $\mathcal{F}$ such that the corresponding LPO orients all of the rules is more extensive than the search for a strict partial order. Both of the corresponding decision problems, strict- and quasi- LPO termination, are decidable and NP complete [22].

3. Partial Order Constraints

Informally, a partial order constraint is just like a formula in propositional logic, except that propositions are atoms of the form $(f > g)$ or $(f = g)$. The semantics of a partial order constraint is a set of solutions. A solution is an assignment of truth values to atoms which is required to satisfy both parts of the formula: the propositional part—namely, viewing the formula as propositional; and the partial order part—namely, being consistent with the axioms of partial order with regards to the symbols in the atoms. Moreover, we require solutions to be closed under extension.

Syntax: Let $\mathcal{F}$ be a finite non-empty set of symbols and $\{ >, = \}$ be binary relation symbols on $\mathcal{F}$. We denote by $Atom_{\mathcal{F}}$ the set of atoms of the form $(f = g)$ and $(f > g)$ where $f, g \in \mathcal{F}$. A partial order constraint on $\mathcal{F}$ is a propositional formula in which the propositions are elements of $Atom_{\mathcal{F}}$. We sometimes write $(f \geq g)$ as shorthand for $(f > g) \vee (f = g)$. We denote the set of atoms occurring in a partial order constraint φ by $Atom(\varphi)$. We assume two special atoms denoted *true* and *false*. These may be viewed as shorthand for $(x = x)$ and $(x > x)$ (for a symbol $x \in \mathcal{F}$) respectively.

Semantics: The relations $>$ and $=$ are interpreted, respectively, as a strict partial order and as equality (both on $\mathcal{F}$). Let φ be a partial order constraint on $\mathcal{F}$. The semantics of φ is a set of solutions. A model of φ is a set of atoms from $Atom_{\mathcal{F}}$ which satisfies both parts of the formula: the propositional part and the partial order part. A solution of φ is a model μ of φ such that any extension of μ is also a model of φ. The following definition formalizes the semantics for partial order constraints.

Definition 2 (assignment, extension, model, solution)**.** *Let φ be a partial order constraint on $\mathcal{F}$. An* assignment *μ is a mapping from propositions of $Atom_{\mathcal{F}}$ to truth values, and can be identified with the set of propositions it assigns to the value true. We say that an assignment μ is a* partial order *if it satisfies the axioms for strict partial order and equality. More specifically, for all $f, g, h \in \mathcal{F}$:*

$$
\begin{array}{lllllll}
\textit{reflexivity:} & (f = f) \in \mu & & & & & \\
\textit{symmetry:} & (f = g) \in \mu & \Rightarrow & (g = f) \in \mu & & & \\
\textit{asymmetry:} & \neg((f > g) \in \mu \wedge (g > f) \in \mu) & & & & & \\
\textit{transitivity:} & (f > g) \in \mu & \wedge & (g > h) \in \mu & \Rightarrow & (f > h) \in \mu & \\
 & (f = g) \in \mu & \wedge & (g = h) \in \mu & \Rightarrow & (f = h) \in \mu & \\
\textit{identity:} & (f > g) \in \mu & \wedge & (g = h) \in \mu & \Rightarrow & (f > h) \in \mu & \\
 & (f = g) \in \mu & \wedge & (g > h) \in \mu & \Rightarrow & (f > h) \in \mu &
\end{array}
\tag{1}
$$

A partial order extension *(or simply* extension*) of an assignment μ is a partial order μ', such that $\mu \subseteq \mu'$. We say that an assignment μ is a* partial order model *for φ if it is a partial order and it is a model for φ. A* solution *μ of φ is a partial order model of φ, such that all extensions of μ are also models of φ. A partial order constraint φ is* satisfiable *if and only if it has a solution.*

Example 2. *Let $\mathcal{F} = \{f, g, h\}$. The following are partial order constraints:*

$$\begin{aligned} \varphi_1 &= (f > g) \wedge ((f > h) \vee (h > f)) \\ \varphi_2 &= (f \geq g) \wedge (g \geq h) \wedge (h \geq g) \\ \varphi_3 &= (f > g) \wedge \neg((h > g) \vee (f > h)) \end{aligned}$$

The set of atoms $\mu_1 = \{\ (f > g), (f > h), (f = f), (g = g), (h = h)\ \}$ is a model for φ_1. It satisfies the propositional part of φ_1: φ_1 evaluates to true when assigning the atoms in μ_1 the value true and the others the value false. It satisfies the partial order part of φ_1: it is a partial order. The set of atoms $\{\ (h > f), (f > g)\ \}$ is not a model (for any partial order constraint) because it is not closed under transitivity (nor reflexivity): it is not a partial order. However, its extension $\mu_2 = \{\ (h > f), (f > g), (h > g), (f = f), (g = g), (h = h)\ \}$ is a model for φ_1. Formula φ_1 has additional models which are extensions of μ_1 to a total order:

$$\begin{aligned} \mu_3 &= \{\ (f > g), (g > h), (f > h), (f = f), (g = g), (h = h)\ \}, \\ \mu_4 &= \{\ (f > h), (h > g), (f > g), (f = f), (g = g), (h = h)\ \}, \text{ and} \\ \mu_5 &= \{\ (f > g), (g = h), (h = g), (f > h), (f = f), (g = g), (h = h)\ \} \end{aligned}$$

Indeed, all of the extensions of μ_1 are models, so μ_1 is a solution of φ_1. Since μ_3, μ_4, and μ_5 are total orders and models they are also solutions.

The formula φ_2 has two example solutions:

$$\begin{aligned} &\{\ (f > g), (g = h), (h = g), (f > h), (f = f), (g = g), (h = h)\ \}, \text{ and} \\ &\{\ (f = g), (g = f), (g = h), (h = g), (f = h), (h = f), (f = f), (g = g), (h = h)\ \} \end{aligned}$$

Focusing on φ_3, we illustrate an additional condition for an assignment to satisfy a partial order constraint. When searching for solutions of a formula, we are only interested in assignments if their extensions are also solutions. The partial order $\mu = \{\ (f > g), (f = f), (g = g), (h = h)\ \}$ is a model: φ_3 evaluates to true when assigning the atoms in μ the value true and the others the value false. However, no extension of μ satisfies the propositional part of φ_3 and hence μ will not be considered a solution of φ_3.

The following proposition enables us to restrict attention to partial order models which are total orders. A *total order* μ is a partial order that also satisfies the following additional axiom. For all $f, g \in \mathcal{F}$:

$$\text{comparability:} \quad (f > g) \in \mu \ \vee \ (g > f) \in \mu \ \vee \ (f = g) \in \mu \tag{2}$$

Proposition 1. *Partial order constraint φ has a solution if and only if it has a partial order model which is a total order.*

The proof of this proposition is straightforward. Given that we focus on total order solutions, we have that $\neg(f > g) \equiv (g > f) \vee (g = f)$ and that $\neg(f = g) \equiv (f > g) \vee (g > f)$. Hence we may assume without loss of generality that partial order constraints are negation free. For example, the formula φ_3 from Example 2 is equivalent to $\varphi_3' = (f > g) \wedge (g \geq h) \wedge (h \geq f)$.

Satisfiability: In this paper, we are concerned with the question of satisfiability of partial order constraints: given a partial order constraint φ, does it have a solution? Similar to the general SAT problem, the satisfiability of partial order constraints is NP-complete. The reduction from SAT is straightforward.

Integer-based interpretation: We propose a finite domain integer-based interpretation of partial order constraints. In this approach, the semantics of a partial order constraint is a set of integer solutions.

Definition 3 (integer assignment and solution)**.** *Let φ be a partial order constraint on $\mathcal{F}$ and let $|\mathcal{F}| = n$. An integer* assignment *for φ is a mapping $\theta : \mathcal{F} \to \{0, \ldots, n-1\}$. An integer* solution *of φ is an integer assignment which satisifes φ under the standard interpretations of $>$ and $=$ on the non-negative integers.*

Example 3. *Consider again the partial order constraints from Example 2. The assignments mapping $\langle f, g, h\rangle$ to $\langle 2, 1, 1\rangle$, $\langle 2, 0, 0\rangle$ and $\langle 0, 0, 0\rangle$ are solutions for φ_2. But only the first two are solutions to φ_1. The formula φ_3 has no integer solutions. Any such solution would imply a cycle: $f > g$, $g \geq h$ and $h \geq f$.*

Theorem 2. *A partial order constraint φ is satisfiable if and only if it has an integer solution.*

The theorem is a direct consequence of the following two lemmata.

Lemma 3. *Let θ be an integer solution of φ. The assignment*

$$\mu = \left\{ \; (f > g) \;\middle|\; \{f, g\} \subseteq \mathcal{F}, (\theta(f) > \theta(g)) \; \right\} \bigcup \left\{ \; (f = g) \;\middle|\; \{f, g\} \subseteq \mathcal{F}, (\theta(f) = \theta(g)) \; \right\}$$

is a solution of φ.

Proof. Observe that μ satisfies both the propositional and partial order parts of φ, since the integer relation $>$ is a total order. Hence μ is a solution for φ by definition. By construction, μ is total and hence any extension of μ is also a model of φ. □

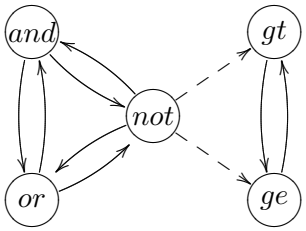
Lemma 4. *Let φ be a satisfiable partial order constraint for $\mathcal{F}$ with n symbols. Then there exists an integer solution θ of φ in $\left\{ \; 0, \ldots, n-1 \; \right\}$.*

Proof. By Proposition 1, φ has a total order model μ. Assume $\mathcal{F} = \{f_1, \ldots, f_n\}$. By comparability, for each $0 \leq i < j \leq n-1$ exactly one of $f_i > f_j$ or $f_i = f_j$ or $f_j > f_i$ holds. We can linearize the symbols in $\mathcal{F}$: $f_{k_n} \rhd_{n-1} \cdots \rhd_2 f_{k_2} \rhd_1 f_{k_1}$ where for each $1 \leq i < n$, $\rhd_i \in \{>, =\}$ and $(f_{k_{i+1}} \rhd_i f_{k_i}) \in \mu$, since μ models transitivity, symmetry, and identity. We can then construct a solution θ, using values from 0 to no more than $n-1$, where

$$\begin{aligned} &\theta(f_{k_1}) = 0 \\ &\theta(f_{k_{j+1}}) = \begin{cases} \theta(f_{k_j}) & \text{where } \rhd_{j-1} \equiv (=) \\ \theta(f_{k_j}) + 1 & \text{where } \rhd_{j-1} \equiv (>) \end{cases} \quad \text{for } 0 \leq j < n-1 \end{aligned}$$

□

$$\begin{aligned}\varphi \;=\; & ((gt > ge) \vee (not > ge)) \wedge ((ge > gt) \vee (not > gt)) \wedge \\ & (((or > and) \wedge (or > not)) \vee (not > and)) \wedge \\ & (((and > or) \wedge (and > not)) \vee (not > or)) \wedge (and > or)\end{aligned}$$



(a) (b)

Figure 2. (a) A partial order constraint, and (b) its domain graph. The graph has two strongly connected components: $\{gt, ge\}$ and $\{not, and, or\}$. Arcs between the components are dashed.

Simplifying partial order constraint satisfaction: The atoms in a formula φ induce a graph G_φ on the symbols in $\mathcal{F}$. The strongly connected components of G_φ indicate where the partial order constraints can cause unsatisfiability. This graph, G_φ, captures all possible cycles in the partial order and hence all potential contradictions. The following definition is inspired by [23]. We can simplify a (negation free) partial order constraint by making use of G_φ.

Definition 4 (domain graph)**.** *Let φ be a (negation free) partial order constraint on $\mathcal{F}$. The domain graph $G_\varphi = (V, E)$ is a directed graph with vertices $V = \mathcal{F}$ and edges $E = \{\ (f, g) \mid \{\ (f > g), (f = g), (g = f)\ \} \cap Atom(\varphi) \neq \emptyset\ \}$.*

Figure 2 illustrates a partial order constraint (a) and its domain graph (b).

Definition 5 (simplifying a partial order constraint)**.** *Let φ be a (negation free) partial order constraint on $\mathcal{F}$. The* simplification *of φ is the formula obtained by substituting true for any atom $(f > g)$, such that f and g appear in different strongly connected components (SCCs) of G_φ.*

Example 4. *Consider the partial order constraint φ and its domain graph G_φ depicted as Figure 2. The graph G_φ has two strongly connected components. The simplification for φ, as prescribed by Definition 5, is obtained as:*

$$\begin{aligned}\varphi' \;\equiv\; & ((gt > ge) \vee true) \wedge ((ge > gt) \vee true) \wedge \\ & (((or > and) \wedge (or > not)) \vee (not > and)) \wedge \\ & (((and > or) \wedge (and > not))) \vee (not > or)) \\ \equiv\; & (((or > and) \wedge (or > not)) \vee (not > and)) \wedge \\ & ((((and > or) \wedge (and > not))) \vee (not > or))\end{aligned}$$

which is satisfied by a partial order in which $(not > or)$ and $(not > and)$.

Lemma 5. *A (negation free) partial order constraint is φ satisfiable if and only its simplification φ' is satisfiable.*

Proof. Any solution of φ is a solution of φ', as we have only replaced literals occurring positively by *true*. Hence, if θ is an integer solution of φ, then θ is an integer solution of φ'. Take an integer solution θ' of φ'. We can create a new solution θ'' of φ' by shifting the mapping of all symbols in any given strongly connected component of G_φ by a constant c. This holds since there are no atoms that involve symbols from two SCCs. Consider a total order on the strongly connected components of G_φ. It is straightforward to shift the values indicated by θ' of the symbols in corresponding components such that $f > g$ holds for each atom $(f > g)$ with f and g from different components. The mapping obtained is an integer solution of φ. □

4. A Symbol-Based Propositional Encoding

This section presents a propositional encoding of partial order constraints. We refer to this encoding as *symbol-based*. A partial order constraint φ on a set of symbols $\mathcal{F}$ is encoded by a propositional formula φ' such that each solution of φ corresponds to a model of φ' and in particular such that φ is satisfiable if and only if φ' is. The idea is to construct the encoding in terms of the integer-based interpretation of partial order constraints. We view the symbols in $\mathcal{F}$ as integer variables and interpret atoms $(f > g)$ and $(f = g)$ as integer constraints. By Lemma 4, if $\mathcal{F}$ has n symbols then it suffices to consider the finite integer domain $\{\ 0, \ldots, n-1\ \}$ for the symbols in $\mathcal{F}$. To obtain a propositional encoding, each symbol is modeled using $k = \lceil \log_2 n \rceil$ propositional variables which encode the binary representation of its value and the constraints $(f > g)$ and $(f = g)$ are encoded in k-bit arithmetic as detailed below.

To introduce the encoding we assume the following notation: With each atom $a \in \mathit{Atom}_\mathcal{F}$, we associate the propositional variable denoted $[\![a]\!]$. The propositional part of the partial order constraint φ is denoted $[\![\varphi]\!]$. This is the propositional formula obtained by replacing each atom $a \in \mathit{atom}(\varphi)$ with its associated propositional variable $[\![a]\!]$.

Example 5. *Consider the partial order constraint*

$$\begin{aligned}\varphi' \quad = \quad & ((or > and) \wedge (or > not) \vee (not > and)) \wedge \\ & ((and > or) \wedge (and > not) \vee (not > or))\end{aligned}$$

from Example 4. Its propositional part is

$$\begin{aligned}[\![\varphi']\!] \quad = \quad & ([\![or > and]\!] \wedge [\![or > not]\!] \vee [\![not > and]\!]) \wedge \\ & ([\![and > or]\!] \wedge [\![and > not]\!] \vee [\![not > or]\!])\end{aligned}$$

The propositional encoding of a partial order constraint φ over an alphabet $|\mathcal{F}|$ of n symbols with $k = \lceil \log_2 n \rceil$ is denoted $\|\varphi\|_k$ (we sometimes omit the subscript k) and defined as follows:

1. The k-bit representation for $f \in \mathcal{F}$ is $\langle f_k, \ldots, f_1 \rangle$ with f_k the most significant bit.
2. A constraint of the form $(f = g)$ is encoded in k-bits by

$$\|(f = g)\|_k = \bigwedge_{i=1}^{k} (f_i \leftrightarrow g_i).$$

A constraint of the form $(f > g)$ is encoded in k-bits by

$$\|(f > g)\|_k = \begin{cases} (f_1 \wedge \neg g_1) & k = 1 \\ (f_k \wedge \neg g_k) \vee ((f_k \leftrightarrow g_k) \wedge \|(f > g)\|_{k-1}) & k > 1 \end{cases}$$

3. A partial order constraint φ is encoded in k bits by the conjunction of the propositional part of φ with the formulae defining the propositional variables associated with the atoms in φ:

$$\|\varphi\|_k = \quad [\![\varphi]\!] \quad \wedge \bigwedge_{a \in Atom(\varphi)} ([\![a]\!] \leftrightarrow \|a\|_k) \tag{3}$$

Example 6. *Consider the partial order constraint φ' from Example 5 and recall that:*

$$\begin{aligned} [\![\varphi']\!] \quad = \quad & ([\![or > and]\!] \wedge [\![or > not]\!] \vee [\![not > and]\!]) \wedge \\ & ([\![and > or]\!] \wedge [\![and > not]\!] \vee [\![not > or]\!]) \end{aligned}$$

Each of the three symbols in φ' is represented in 2 bits and the propositional encoding of φ' is obtained as:

$$\begin{aligned} \|\varphi'\|_2 \quad = \quad & [\![\varphi']\!] \ \wedge \ ([\![not > or]\!] \leftrightarrow ((not_2 \wedge \neg or_2) \vee (not_2 \leftrightarrow or_2 \wedge not_1 \wedge \neg or_1))) \wedge \\ & ([\![and > or]\!] \leftrightarrow ((and_2 \wedge \neg or_2) \vee (and_2 \leftrightarrow or_2 \wedge and_1 \wedge \neg or_1))) \wedge \\ & ([\![not > and]\!] \leftrightarrow ((not_2 \wedge \neg and_2) \vee (not_2 \leftrightarrow and_2 \wedge not_1 \wedge \neg and_1))) \end{aligned}$$

Proposition 6. *The symbol-based encoding of partial order constraint φ with n symbols, m connectives and $r = |Atom(\varphi)|$ distinct atoms involves $O(r + n \log n)$ propositional variables and $O(m + r \log n)$ connectives.*

Recall that SAT solvers typically consider propositional formulae in conjunctive normal form. The transformation to CNF of a propositional formula (such as the encoding in Equation 3) is performed using a (linear) Tseitin transformation [31]. For a formula with m connectives, this transformation will add m new propositional variables and result in a CNF with $O(m)$ clauses of size 3 or less (see [31] for details).

The encoding of φ is a Boolean formula, $\|\varphi\|$, polynomial in the size of φ. Satisfying the formula provides a solution to φ.

Theorem 7. *A partial order constraint φ on symbols $\mathcal{F}$ is satisfiable if and only if its symbol-based propositional encoding $\|\varphi\|$ is.*

Proof. Assume that $\mathcal{F}$ contains n symbols and let $k = \lceil \log_2 n \rceil$.

($\Rightarrow$) Since φ is satisfiable, it has a total order model μ and a corresponding integer solution θ (constructed as in the proof of Lemma 4). Let θ' be the Boolean assignment on the (propositional) variables $\{ f_j \mid f \in \mathcal{F},\ 1 \leq j < k \}$ such that for each $f \in \mathcal{F}$, $\langle \theta'(f_k), \ldots, \theta'(f_1) \rangle$ is the k-bit representation of the integer $\theta(f)$. Then $\mu \cup \theta'$ is a model of $\|\varphi\|_k$ because: (1) by Definition 2, μ models $[\![\varphi]\!]$; and (2) from the construction of θ', for each $a \in Atom(\varphi)$, $(\mu \models [\![a]\!]) \Leftrightarrow (\theta' \models \|a\|)$.

($\Leftarrow$) Consider a model σ of $\|\varphi\|_k$ and its partition $\sigma = \mu \cup \theta'$ where $\mu \subseteq Atom_{\mathcal{F}}$ and $\theta' \subseteq \{ f_j \mid f \in \mathcal{F}, 1 \leq j \leq k \}$. Let θ be the corresponding mapping of symbols in $\mathcal{F}$ to

integers, where $\theta(f)$ is equal to the integer represented in k-bits by $\langle \theta'(f_k), \ldots, \theta'(f_1)\rangle$. Then θ is an integer solution of φ because: (1) for each $[\![f > g]\!] \in \mu$, $\sigma \models ([\![f > g]\!] \leftrightarrow \|f > g\|_k)$ and therefore $\theta(f) > \theta(g)$; (2) for each $[\![f = g]\!] \in \mu$, $\sigma \models ([\![f = g]\!] \leftrightarrow \|f = g\|_k)$ and therefore $\theta(f) = \theta(g)$; and (3) $\mu \models [\![\varphi]\!]$. φ. □

The simplification discussed in the previous section also gives us an immediate basis for reducing the size of integers we need to consider in finding a solution to φ. In order to find an integer solution to the simplification of φ, φ', we only need to consider values from $\{0, \ldots, |S| - 1\}$ for each symbol f in an SCC S of G_φ, since symbols in different SCCs do not interact.

5. Encoding LPO Termination

This section illustrates how an LPO termination problem can be expressed in terms of a partial order constraint. Hirokawa and Middeldorp [16] observe that finding $>_\mathcal{F}$ such that $s \succ_{lpo} t$ is tantamount to solving a constraint obtained by unfolding the definition of $s \succ_{lpo} t$. The following definition specifies such an unfolding in terms of a transformation τ mapping a pair of terms s and t containing symbols from $\mathcal{F}$ to a partial order constraint $\tau(s \succ t)$ on $\mathcal{F}$. A solution of $\tau(s \succ t)$ is a partial order $>_\mathcal{F}$ such that $s \succ_{lpo} t$ holds for the induced lexicographic path order. If $\tau(s \succ t)$ has no solution then it is not possible to orient the terms s and t in a lexicographic path order. The definition of τ follows precisely the structure of Definition 1 The constraint obtained as $\tau(s \succ t)$ is essentially the same referred to in [16].

Definition 6 (encoding LPO)**.** *In the following let $\bar{s} = \langle s_1, \ldots, s_n\rangle$ and $\bar{t} = \langle t_1, \ldots, t_m\rangle$. The encodings of $s \succ_{lpo} t$, $\bar{s} \succ^{lex}_{lpo} \bar{t}$ and $s \sim t$ are defined recursively by the encoding τ and denoted $\tau(s \succ t)$, $\tau(\bar{s} \succ \bar{t})$ and $\tau(s \sim t)$ respectively.*

1. $\tau(s \succ t) = \begin{cases} \mathit{false} & \mathit{if\ s\ is\ a\ variable} \\ (t = g(\bar{t}) \wedge \mu_1) \bigvee \mu_2 & \mathit{if\ } s = f(\bar{s}) \end{cases}$ *where*

   - $\mu_1 = \bigwedge_{1 \le j \le m} \tau(s \succ t_j) \bigwedge \Big((f >_\mathcal{F} g) \bigvee ((f \approx_\mathcal{F} g) \wedge \tau(\bar{s} \succ \bar{t})) \Big)$, *and*
   - $\mu_2 = \bigvee_{1 \le i \le n} (\tau(s_i \succ t) \vee \tau(s_i \sim t))$

2. $\tau(\langle s_1, \ldots, s_n\rangle \succ \langle t_1, \ldots, t_m\rangle) = \quad n > 0 \wedge \Big(m = 0 \ \vee \ \Big(m > 0 \ \wedge \begin{pmatrix} \tau(s_1 \succ t_1) & \vee \\ (\tau(s_1 \sim t_1) \wedge \tau(\langle s_2, \ldots, s_n\rangle \succ \langle t_2, \ldots, t_m\rangle)) \end{pmatrix} \Big) \Big)$

3. $\tau(s \sim t) = \begin{cases} \mathit{true} & \mathit{if\ } s = t \\ (f \approx_\mathcal{F} g) \wedge \bigwedge_{1 \le i \le n} \tau(s_i \sim t_i) & \mathit{if\ } s = f(s_1, \ldots, s_n) \ \mathit{and\ } t = g(t_1, \ldots, t_n) \\ \mathit{false} & \mathit{otherwise} \end{cases}$

The encoding specified in Definition 6 can be further specialized in case we are interested in a strict partial order $>_{\mathcal{F}}$. In this case $\approx_{\mathcal{F}}$ and $\sim$ are the syntactic identity of symbols and terms respectively. As such atoms of the form $(f \approx_{\mathcal{F}} g)$ and constraints of the form $(s \sim t)$ can be immediately evaluated to *true* or *false*.

Example 7. *Consider the term rewrite system of Figure 1 and assume that we are interested in a strict precedence $>_{\mathcal{F}}$. Applying Definition 6 to the left and right sides of each of the rules results in the following partial order constraints:*

$$\begin{aligned}
not(gt(A,B)) >_{lpo} ge(B,A) &= (gt > ge) \vee (not > ge)\\
not(ge(A,B)) >_{lpo} gt(B,A) &= (ge > gt) \vee (not > gt)\\
not(or(A,B)) >_{lpo} and(not(A), not(B)) &= ((or > and) \wedge (or > not)) \vee (not > and)\\
not(and(A,B)) >_{lpo} or(not(A), not(B)) &= ((and > or) \wedge (and > not)) \vee (not > or)\\
and(A, or(B,C)) >_{lpo} or(and(A,B), and(A,C)) &= and > or\\
and(or(B,C), A) >_{lpo} or(and(B,A), and(C,A)) &= and > or
\end{aligned}$$

The partial order constraint, satisfiability of which, determines strict- or quasi-LPO termination of a constraint system is obtained as the conjunction of the encodings $\tau(\ell \succ r)$ of the rules in the system. Coming back to the system of Figure 1, the conjunction of constraints illustrated in Example 7 is precisely the partial order constraint φ from Figure 2. Coming back to Example 4, it is straightforward to observe that it is satisfiable. Hence the system is LPO-terminating.

The next example illustrates a term rewrite system which is quasi-LPO terminating but not strict-LPO terminating.

Example 8. *Consider the following term rewrite system.*

$$\begin{aligned}
&div(X,e) \to i(X)\\
&i(div(X,Y)) \to div(Y,X)\\
&div(div(X,Y),Z) \to div(Y, div(i(X),Z))
\end{aligned}$$

Definition 6 for strict-LPO gives

$$\begin{aligned}
div(X,e) >_{lpo} i(X) &= div > i\\
i(div(X,Y)) >_{lpo} div(Y,X) &= i > div\\
div(div(X,Y),Z) >_{lpo} div(Y, div(i(X),Z)) &= div > i
\end{aligned}$$

The conjunction of the constraints on the right sides is not satisfiable indicating that there does not exist any strict partial order on $\mathcal{F}$ such that the corresponding lexicographic path order decreases on the three rules. The system is however quasi-LPO terminating. Definition 6 for quasi-LPO gives a satisfiable partial order constraint equivalent to $(div \geq i) \wedge (i \geq div)$ which indicates that taking $div \approx i$ provides a proof of quasi-LPO termination (and hence termination of the given system).

We conclude this section with a note on the size of the partial order constraint $\tau(s \succ_{lpo} t)$ for terms s and t involving a total of n symbol occurrences. Consider one step of unfolding the recursive definition for the given terms $s = f(s_1, \ldots, s_p)$ and $t = g(t_1, \ldots, t_q)$. We obtain a formula of the form

$$\tau(s \succ t) \;=\; \bigwedge_j \tau(s \succ t_j) \wedge \Big((f >_{\mathcal{F}} g) \vee (f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \succ \bar{t})\Big) \vee \bigvee_i \Big(\tau(s_i \succ t) \vee \tau(s_i \sim t)\Big)$$

This formula contains $p + q$ subformulae of the form $\tau(s \succ t')$ or $\tau(s' \succ t)$ each involving one of the original terms s or t. Further evaluation steps, one for each of these subterms further multiplies the number of occurrences of subformula containing one of the original terms s or t. Hence, a direct application of Definition 6 may result in a constraint of size exponential in n.

Example 9. *The straightforward application of Definition 6 for*

$$\tau(f(f(f(c,d),f(c,d)),f(f(c,d),f(c,d))) \succ g(g(g(a,b),g(a,b)),g(g(a,b),g(a,b))))$$

results in a partial order constraint containing 2228 Boolean connectives and 4458 occurrences of symbols from $\mathcal{F} = \{f, g, a, b, c, d\}$.

To obtain a polynomial size encoding we introduce sharing of common sub-formula into the partial order constraint. The approach is similar to that proposed by Tseitin to obtain linear CNF transformation of Boolean formula [31]. Instead of encoding $s' \succ_{lpo} t'$ "in place" for each subformula encountered when encoding $s \succ_{lpo} t$, we introduce a fresh Boolean variable of the form $X_{(s' \succ t')}$ to represent its encoding. Similarly we introduce $X_{(s' \sim t')}$. Viewed this way, the encoding takes the form:

$$\begin{aligned}
X_{(s \succ t)} &\leftrightarrow \bigwedge_j X_{(s \succ t_j)} \wedge \Big((f > g) \vee (f = g) \wedge \tau(\bar{s} \succ \bar{t})\Big) \vee \bigvee_i \Big(X_{(s_i \succ t)} \vee X_{(s_i \sim t)}\Big) \\
&\cdots \\
X_{(s' \succ t')} &\leftrightarrow \qquad\qquad \cdots \\
&\cdots
\end{aligned}$$

where s' and t' are subterms of s and t respectively and for the lexicographic order on tuples we evaluate the definition of $\tau(\bar{s} \succ \bar{t})$ in-line. It takes the following form:

$$\tau(\bar{s} \succ \bar{t}) = X_{(s_1 \succ t_1)} \vee (X_{(s_1 \sim t_1)} \wedge X_{(s_2 \succ t_2)} \vee (X_{(s_2 \sim t_2)} \wedge \cdots \vee (X_{(s_{n-1} \sim t_{n-1})} \wedge X_{(s_n \succ t_n)}) \cdots))$$

Let us now consider the (total) size of this system of equations:

- The number of rows is $O(n^2)$ considering all pairs of subterms s' and t'; and each row contains two partial order atoms, $(f >_{\mathcal{F}} g)$ and $(f \approx_{\mathcal{F}} g)$, as well as some number of variables of the form $X_{(s \succ t)}$ and $X_{(s \sim t)}$.
- Consider a variable of the form $X_{(s \succ t_j)}$ on the right side of $\leftrightarrow$ referring to term s (e.g. in the first row). It will not appear again on the right side in another row. This is because (looking in the "recursive calls" of the first row) it will not occur in the specifications of variables of the form $X_{(\bar{s} \succ \bar{t})}$ nor of $X_{(s_i \succ t)}$ which refer to proper subterms of s. Likewise variables of the form $X_{(s_i \succ t)}$ (e.g. from the first row) will not occur again on the right side in other rows of the system.
- Now consider a variable of the form $X_{(s_i \succ t_i)}$ occurring in the unfolding of $\tau(\bar{s} \succ \bar{t})$ in the first row. It will occur 3 times (on the right side of a $\leftrightarrow$) in the system: Once in the unfolding of $\tau(\bar{s} \succ \bar{t})$, once in the specification of $X_{(s_i \succ t)} \leftrightarrow \cdots$, and once in the specification of $X_{(s \succ t_i)} \leftrightarrow \cdots$.

In summary, it means that the total number of variables on all of the right sides of the $\leftrightarrow$'s is $O(n^2)$. Hence the total size of the encoding is determined by the $O(n^2)$ number of rows and the atoms $(f > g)$, $(f = g)$ which encode to $O(\log n)$ bits. Namely, the size of the encoding is $O(n^2 \log n)$. In practice it is often the case that $|\mathcal{F}| \ll n$. In this case we may view the encoding of the partial order atoms as constant size and the overall encoding as $O(n^2)$.

Strictly speaking, the unfoldings obtained now contain partial order atoms like $(f > g)$, as well as Boolean Tseitin variables, and as such are not partial order constraints. However, this may be safely viewed as an implementation detail with the Tseitin variables used to directly obtain the encoding of $s \succ_{lpo} t$ in CNF.

6. Implementation and Experimental Results

A prototype analyzer, lpoSAT, for strict- and quasi- LPO termination is described in [6]. It is based directly on the (polynomial sized) encoding of an LPO termination problem to a partial order constraint described in Section 5 and then on the encoding of this constraint to a CNF as described in Section 4. The CNF is delegated to a SAT solver. A satisfying assignment, if one exists, is then interpreted to provide the user with a precedence which proves LPO termination.

The prototype is written in SWI Prolog [34, 28] and applies the MiniSat solver [12, 25] through its Prolog interface described in the paper by Codish, Lagoon and Stuckey [7]. The experiments described in the preliminary version of this paper [6] indicate orders of magnitude speed-up when compared with the two main termination-proving tools incorporating LPO termination: TTT [32] and AProVE [2].

These experiments are summarized in Table 1, where parts (a) and (b) of the table present the results for strict- and quasi- LPO termination analyses, respectively. For these experiments, lpoSAT was run on a 1.5-GHz laptop with Linux FC4 and compared against TTT run through its Web interface.[2.] The TTT analyzer was run on a Xeon 2.24-GHz dual-core machine. At the time, experiments with AProVE gave results which were considerably slower than TTT and hence the comparison focused on TTT. The columns in the tables contain times (in seconds) with TTT configured to run with a timeout of 600 seconds, the maximum allowed by its Web interface. The comparison involves 751 term rewrite systems from the Termination Problem Data Base (TPDB) version 2.0 [30]. The times presented in the table are taken on different machines which makes the precise comparison impossible. Nevertheless, the results are indicative showing that lpoSAT is fast in absolute terms.

In the past year, the SAT-based techniques described in [6] were integrated within the AProVE tool [2]. Moreover, also the dedicated (non SAT-based) version of the LPO termination analysis in AProVE and TTT was improved (partially in view of recommendations presented in [6]). These changes facilitate a better comparison between the SAT and dedicated approaches to LPO termination analysis.

Table 2 provides a current comparison for strict- and quasi- LPO termination analyses in parts (a) and (b) of the table respectively. The first two columns in each part of the table provide the results for the SAT-based and dedicated analyses of AProVE. The third and fourth columns of each part illustrate the results for our Prolog implementation, as well as

2. http://colo6-c703.uibk.ac.at/ttt/cgi-bin/index.cgi

Table 1. Summary of experimental results for strict and quasi LPO termination as reported in [6]: total, average and maximum times (sec) for 751 tests of TPDB version 2.0. Timeout is 600 seconds, the maximum allowed by the web interface of TTT.

Time (sec)	lpoSAT	TTT	lpoSAT	TTT
Total	8.983	647.48	8.609	2167.44
Average	0.012	0.86	0.011	2.89
Max	0.477	317.63	0.544	600.00
Timeouts	–	–	–	1

(a) strict LPO (2006) **(b)** quasi-LPO (2006)

Table 2. Summary of experimental results (2007) for strict- and quasi- LPO termination: total, average and maximum times (sec) for 865 tests of TPDB version 3.2. Timeout is 60 seconds, following the same rules of the International Termination Competition.

Time(sec)	AProVE (SAT)	AProVE (ded.)	lpoSAT (SAT)	TTT (ded.)	AProVE (SAT)	AProVE (ded.)	lpoSAT (SAT)	TTT (ded.)
Total	5.86	328.49	3.59	232.52	9.31	1330.78	10.59	1752.81
Average	0.007	0.38	0.004	0.27	0.011	1.54	0.012	2.03
Max	0.31	60.00	0.20	60.00	0.50	60.00	2.80	60.00
Timeouts	–	4	–	2	–	14	–	9

(a) strict LPO (2007) **(b)** quasi-LPO (2007)

for the TTT analyzer. Except for TTT, all runtimes are measured on a dual-core 2.16-GHz laptop, running GNU/Linux 2.6 (Ubuntu 7.04). The TTT analyzer is running through its Web interface. The tables indicate total, average and maximum runtimes for the 865 term rewrite systems in the Termination Problem Data Base version 3.2 [30]. AProVE and TTT are configured to timeout after 60 seconds, following the same rules as in the *International Termination Competition* [24].

Table 1 shows that the SAT based approach was orders of magnitude faster than the dedicated non-SAT-based algorithm in TTT in 2006. Table 2 shows that the advantage of the SAT-based approach over the non-SAT-based algorithms continue to hold in 2007. The reader should focus on the results obtained within AProVE comparing the LPO termination analysis, applying the SAT-based approach, and the dedicated non-SAT-based algorithm. This gives the fairest possible comparison, since the remainder of the system is identical. Clearly the SAT-based approach is superior. In spite of the few timeouts encountered when applying the dedicated analysers, both provide the same number of termination proofs: from the 865 term rewrite systems, 123 and 127 are found terminating when applying the strict- and quasi- LPO techniques, respectively. Tables 1 and 2 are not comparable: The analyzers have been improved and are running on different (faster) machines; and the benchmark suite has also been extended.

Table 3. Execution times (sec) for the 25 hardest tests for the dedicated solver in AProVE with 60 second timeouts indicated by ∞.

	LPO		quasi-LPO	
Test	AProVE (SAT)	AProVE (dedicated)	AProVE (SAT)	AProVE (dedicated.)
AG01/#3.10	0.01	0.12	0.03	17.55
AProVE/AAECC	0.01	0.37	0.03	∞
AProVE/AAECC-ring	0.04	1.50	0.10	∞
AProVE/JFP_Ex51	0.01	0.05	0.01	25.07
Cime/mucrl1	0.13	∞	0.50	∞
currying/AG01/#3.13	0.07	2.33	0.08	40.59
higher-order/Bird/BTreeMember	0.01	0.70	0.02	∞
higher-order/Bird/Hamming	0.03	0.24	0.04	19.10
HM/t009	0.05	5.75	0.34	∞
TRCSR/Ex15_Luc98_C	0.03	0.44	0.04	27.75
TRCSR/Ex1_2_AEL03_C	0.05	59.12	0.08	∞
TRCSR/Ex1_GL02a_C	0.02	2.06	0.03	∞
TRCSR/Ex1_GM03_C	0.02	1.23	0.04	∞
TRCSR/Ex26_Luc03b_C	0.03	0.37	0.04	32.75
TRCSR/Ex2_Luc02a_C	0.03	0.58	0.04	34.37
TRCSR/Ex2_Luc03b_C	0.02	0.25	0.04	15.09
TRCSR/Ex49_GM04_C	0.02	4.75	0.03	∞
TRCSR/Ex49_GM04_FR	0.01	0.03	0.02	21.34
TRCSR/Ex5_7_Luc97_C	0.05	∞	0.07	∞
TRCSR/Ex6_15_AEL02_C	0.08	∞	0.10	∞
TRCSR/Ex6_15_AEL02_Z	0.02	0.02	0.05	∞
TRCSR/ExAppendixB_AEL03_C	0.06	∞	0.10	∞
TRCSR/ExIntrod_GM99_C	0.04	1.76	0.07	52.81
TRCSR/ExIntrod_Zan97_C	0.03	0.92	0.04	∞
TRCSR/ExSec11_1_Luc02a_C	0.04	0.54	0.05	49.76
Total (including 60 sec timeouts)	0.91	323.13	1.99	1176.18

Table 3 focuses on the 25 most challenging examples for the dedicated solver of AProVE (chosen by maximum total time for strict and quasi- LPO analyses in the dedicated solver). The AProVE SAT-based times are provided for comparison. The columns indicate runtimes (in seconds) for the LPO and quasi-LPO analyses for the SAT-based and dedicated solvers of AProVE. Clearly the problems that are hard for the dedicated solver are easy for the SAT-based solver, none requiring more than half a second. The results demonstrate the clear benefit of the new SAT-based method in AProVE.

Table 4 presents a detailed analysis for the 25 most challenging examples for lpoSAT (chosen by maximum total time for strict and quasi- LPO analyses). The two "Time" columns provide the runtimes of the lpoSAT analyzer for LPO and quasi-LPO, respectively. The V and C columns list the number of Boolean variables and the number of clauses in the SAT instance generated in each case. Finally, the "CSP%" columns characterize the percentage of runtime spent on the actual partial order constraint solving in each case. This figure includes conversion of the partial order constraints into a SAT instance, solving the

Table 4. The 25 hardest tests for lpoSAT, with benchmark directory TRCSR indicated by (T), showing execution time (sec), and number of variable V and number of clauses C in the SAT instance generated, as well as the percentage of runtime spent in constraint solving for each case.

	LPO				quasi-LPO			
Test	Time	V	C	CSP%	Time	V	C	CSP%
AProVE/AAECC-ring	0.034	480	1478	38	0.049	1088	3525	49
Cime/mucrl1	0.121	0	1	0	0.158	0	1	0
currying/AG01/#3.13	0.013	0	1	0	0.201	0	1	0
currying/D33/30	0.014	0	1	0	1.060	0	1	0
HM/t009	0.050	589	1767	32	0.116	2504	8192	59
secret05/cime1	0.036	0	1	0	0.568	0	1	0
secret05/cime5	0.032	0	1	0	0.061	0	1	0
Thiemann/nonterm	0.013	0	1	0	2.826	0	1	0
(T)/Ex1_2_AEL03_C	0.036	480	1476	33	0.083	1834	5972	59
(T)/Ex26_Luc03b_C	0.020	278	830	40	0.050	1189	3598	60
(T)/Ex2_Luc02a_C	0.020	282	846	30	0.053	1229	3720	60
(T)/Ex3_3_25_Bor03_C	0.017	209	603	29	0.040	873	2648	57
(T)/Ex4_7_37_Bor03_C	0.021	209	611	24	0.048	921	2715	48
(T)/Ex5_7_Luc97_C	0.032	474	1411	41	0.074	1873	6018	64
(T)/Ex6_15_AEL02_C	0.050	692	2135	46	0.106	2563	8301	61
(T)/Ex6_15_AEL02_FR	0.019	453	1343	47	0.040	1152	3776	68
(T)/Ex6_15_AEL02_GM	0.026	557	1727	54	0.052	1457	4789	71
(T)/Ex6_15_AEL02_Z	0.018	443	1313	56	0.039	1164	3812	72
(T)/Ex7_BLR02_C	0.016	215	635	31	0.040	967	2939	57
(T)/Ex9_BLR02_C	0.018	220	626	44	0.039	891	2646	54
(T)/ExAppendixB_AEL03_C	0.040	532	1652	32	0.093	2016	6541	61
(T)/ExIntrod_GM99_C	0.027	309	915	37	0.063	1271	3837	59
(T)/ExIntrod_Zan97_C	0.020	248	735	40	0.050	1104	3316	62
(T)/ExSec11_1_Luc02a_C	0.025	319	950	36	0.063	1445	4303	60
Zantema/z30	0.023	65	104	9	0.228	174	350	0
Average	0.030	282.2	846.5	28	0.248	1028.6	3240.2	43
Max	0.121	692	2135	56	2.826	2563	8301	72
Total	0.741				6.200			

SAT instance with MiniSat and interpreting the result in terms of the constraint symbols. We also provide the averages and the maximum values for each of the columns and the total runtimes.

Six examples result in trivial CNF instances (0 Boolean variables / 1 CNF clause). In these cases, the encoding of the termination problem to a partial order constraint as described in Section 5 results in the `false` constraint. Obviously, the challenge in these examples is not in solving the constraints but rather in obtaining them by unfolding Definition 1. Note that in our implementation, when unfolding based on Definition 1, the only Boolean simplifications applied are to replace expressions where a Boolean constant true or false is encountered by the equivalent expression without this constant: e.g., $\phi \wedge \mathit{false}$ becomes false and $\phi \wedge \mathit{true}$ becomes ϕ.

The results in the "CSP%" columns of Table 4 and their maximum and average values lead to an interesting observation. Namely, solving of partial order constraints is not the hardest step in proving termination, at least not in our prototype implementation. Even for the hardest cases—the average solving time is as low as 28% of the total for strict LPO and 43% for quasi-LPO. Corresponding averages for the complete benchmarking suite are even lower—24% and 30% respectively.

There are 14 tests appearing in both Tables 3 and 4, indicating that the hardest problems are somewhat independent of the actual solver (dedicated or SAT-based). All of the tests in Tables 3 and 4 are neither strict- nor quasi-LPO terminating. This is not surprising for the hardest tests, as proving unsatisfiability is typically harder than finding a solution for a satisfiable formula.

Finally we comment that the prototype does not simplify partial order constraints with respect to their SCC-components (Lemma 5). The experimental results reported in [6] indicate that the implementation would not benefit from that: (a) Most of the tests are very fast without simplification; and (b) It is typical for hard cases of LPO termination to have a large strongly connected component including the majority of the symbols.

7. Related Work

The first encoding of a termination problem into propositional logic is presented in a paper by Kurihara and Kondo [23]. This work considers only strict LPO termination and assumes that partial order constraints are negation free and contain only atoms of the form $(f > g)$ (no equality). The approach is based on an atom-based encoding and a representation in terms of BDDs. It does not provide competitive results. A SAT-based implementation of the atom-based approach of [23] is described in the recent report by Zankl [36] together with an experimental evaluation and a comparison with our symbol-based approach. It shows that the symbol-based approach is orders of magnitude faster for the LPO termination benchmark set.

The atom-based encoding of a partial order constraint φ is obtained by: (a) viewing the atoms in φ as propositional variables, and (b) making the axioms for partial order explicit in the encoding. We can formalize this in our notation of Section 4 where $[\![\varphi]\!]$ denotes the propositional part of the partial order constraint φ obtained by replacing the atoms by corresponding propositional variables. The atom-based propositional encoding of a (negation free) partial order constraint φ on symbols $\mathcal{F}$ which does not involve equality is obtained as

$$[\![\varphi]\!] \ \wedge \ T^{>}_{\mathcal{F}} \ \wedge \ A^{>}_{\mathcal{F}} \tag{4}$$

where

$$A^{>}_{\mathcal{F}} = \bigwedge_{\substack{f,g \in \mathcal{F} \\ f \neq g}} \neg([\![f > g]\!] \wedge [\![g > f]\!]) \qquad \text{and} \qquad T^{>}_{\mathcal{F}} = \bigwedge_{\substack{f,g,h \in \mathcal{F} \\ f \neq g \neq h \neq f}} [\![f > g]\!] \wedge [\![g > h]\!] \rightarrow [\![f > h]\!]$$

For the more general case where φ may contain also equality the encoding must make all of the partial order axioms of Equations (1 – 2) explicit. In general, atom-based propositional encodings result in large propositional formulas. For $|\mathcal{F}| = n$ they introduce $O(n^2)$ propositional variables and involve $O(n^3)$ connectives (e.g., for transitivity).

It is insightful to compare the two encodings of a partial order constraint φ given as Equations (3) and (4). The common part in both encodings is the sub-formula $[\![\varphi]\!]$ in which atoms are viewed as propositional variables. The difference is that Equation (4) introduces explicit axioms to relate the atoms in a partial order, while Equation (3) interprets the n symbols as indices represented in $\lceil \log_2 n \rceil$-bits. This is why the symbol-based encoding introduces $O(n \log n)$ new propositional variables instead of the $O(n^2)$ for the atom-based approach. Moreover the symbol-based encoding does not require the expensive encoding of the axioms because the encoding as integers ensures that they hold "for free" since this is a property of integers. For a partial order constraint φ with m connectives and r distinct atoms, the symbol-based encoding involves $O(m + r \log n)$ connectives instead of $O(m + n^3)$ for the atom-based encoding. More importantly, in practice the search for a solution is extremely sensitive to the number of variables. The $O(n \log n)$ additional variables of the symbol-based encoding is clearly superior to the $O(n^2)$ for the atom-based approach.

In [23] Kurihara and Kondo propose two optimizations. They note that for a given formula φ, the domain graph G_φ is often sparse and hence they propose to specialize the explicit representation of the axioms for those symbols from $\mathcal{F}$ actually occurring in φ. In a second optimization Kurihara and Kondo observe that the axioms for transitivity and asymmetry can be replaced by a simpler axiom (they call it A^*), introducing a single clause of the form $\neg((f_1 > f_2) \wedge (f_2 > f_3) \wedge \cdots \wedge (f_{k-1} > f_k) \wedge (f_k > f_1)$ for each simple cycle $(f_1 > f_2), (f_2 > f_3), \ldots, (f_{k-1} > f_k), (f_k > f_1)$ in G_φ to in order to prevent the cycle from being present in a model. They claim correctness of the encoding and report considerable speedups when it is applied. The problem with this approach is that in general there may be an exponential number of simple cycles to consider. Hence, their encoding either requires $O(n^2)$ propositional variables and introduces $O(n^3)$ connectives or else relies on a potentially exponential phase of processing the simple loops in the domain graph.

In an earlier work, [5], Bryant and Velev also consider an atom-based approach for Boolean satisfiability with transitivity constraints. They too propose methods to reduce the number of clauses required to express the transitivity axiom for partial orders and report significant speed-ups for the benchmarks considered in [5].

Partial order constraints can be seen as an instance of the more general formulae of separation logic as described by Talapur *et al.* [29]. In separation logic the atoms are of the form $f \geq g + c$ with f and g a pair of symbols and c a constant. In [29], the authors take a symbol based approach to test for the satisfiability of formulae in this logic. Our notion of domain graph is a simplification of the *inequalities graph* introduced in that paper. Talapur *et al.* propose an analysis of the inequalities graph which enables to restrict the range of the values which need be considered for the symbols in a constraint. This is a more general result than that obtained via simplification in our Lemma 5. Given the current speed of our LPO termination analyzer, it is unlikely that range restriction could further substantially improve solving times. For the LPO termination benchmark suite, the analyses are so fast that the overhead in computing SCC's does not pay off.

Wang *et al.* [33] also consider an encoding of separation logic to SAT and improve on the results of [29]. They also consider symbol-based and atom-based encodings (termed "small-domain" and "per-constraint") and quote [27] for an experimental study indicating that the atom-based approach is often faster than the symbol-based approach for their benchmarks. Wang *et al.* propose a lazy atom-based approach in which the clauses required to express

the transitivity axioms are added on a "by-need" basis. A main contribution of [33] is the incremental approach to cycle detection. Their approach has been successful for applications involving large (or unbounded) integer valued variables where representing the variables in terms of their bits might be prohibitive.

The LPO termination problems are quite different than those tackled using separation logic. The number of symbols is not large, there are no constants and hence the range of the corresponding integer variables is also not large. Our straightforward symbol-based encoding gives extremely fast results without any need for a specialized SAT solver. As Table 4 shows, the maximum CNF instance solved in our 25 hardest tests includes 2563 propositional variables and 8301 CNF clauses. This is well below the capacity limits of MiniSat, which is reported to handle benchmarks with hundreds of thousands of variables and clauses [25]. Moreover, as indicated by our experimental results, The major part in solving LPO termination problems is in the encoding to partial order constraints and not in solving these constraints.

8. Conclusion

We introduced a new kind of propositional encoding for reasoning about LPO termination of term rewrite systems. The key idea was to transform an LPO termination problem into a partial order constraint on its symbols. We solved partial order constraints applying a symbol-based encoding to propositional logic and using a state-of-the-art SAT solver. Experimental results were unequivocal indicating orders of magnitude speedups in comparison with previous implementations for LPO termination analysis.

This paper has had a direct impact on the design of several major termination analysers for term rewrite systems [2, 32]. Moreover, it was the basis for extensions to more powerful termination proving techniques involving new types of encodings but the same underlying partial order constraint solver.

Acknowledgment

We thank Bart Demoen and Samir Genaim for insights regarding the Prolog implementation. Jürgen Giesl, Peter Schneider-Kamp and Aart Middeldorp assisted with the use of AProVE and TTT. Masahito Kurihara provided the test cases from [23]. Yefim Dinitz and anonymous reviewers provided many useful comments on an earlier version of this paper.

References

[1] Pomiv '96: Proceedings of the DIMACS workshop on partial order methods in verification, 1997.

[2] AProVe: Automated program verification environment. http://www-i2.informatik.rwth-aachen.de/AProVE. Viewed June 2007.

[3] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, **236**(1-2):133–178, 2000.

[4] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge, 1998.

[5] Randal E. Bryant and Miroslav N. Velev. Boolean satisfiability with transitivity constraints. *ACM Transactions on Computational Logic*, **3**(4):604–627, 2002.

[6] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Solving partial order constraints for LPO termination. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA)*, **4098** of *Lecture Notes in Computer Science*, pages 4–18, 2006.

[7] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Logic programming with satisfiability. *Theory and Practice of Logic Programming*, **8**(1):121–128, 2008.

[8] Michael Codish, Peter Schneider-Kamp, Vitaly Lagoon, René Thiemann, and Jürgen Giesl. SAT solving for argument filterings. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, **4246** of *Lecture Notes in Artificial Intelligence*, pages 30–44. Springer, November 2006.

[9] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, **17**:279–301, 1982.

[10] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, **3**(1/2):69–116, 1987.

[11] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, **B: Formal Models and Semantics**, pages 2435–320. Elsevier and MIT Press, 1990.

[12] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003 (Selected Revised Papers)*, **2919** of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[13] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR)*, **4130** of *Lecture Notes in Computer Science*, pages 574–588. Springer, 2006.

[14] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, **4501** of *Lecture Notes in Computer Science*, pages 340–354, 2007.

[15] Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Automated termination proofs with AProVE. In Vincent van Oostrom, editor, *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA)*, **3091** of *Lecture Notes in Computer Science*, pages 210–220, Aachen, Germany, 2004. Springer.

[16] N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In R. Nieuwenhuis, editor, *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA)*, **2706** of *Lecture Notes in Computer Science*, pages 311–320, Valencia, Spain, 2003.

[17] Nao Hirokawa and Aart Middeldorp. Tyrolean termination tool. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA)*, **3467** of *Lecture Notes in Computer Science*, pages 175–184, Nara, Japan, 2005. Springer.

[18] D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA)*, **4098** of *Lecture Notes in Computer Science*, pages 328–342, 2006.

[19] G. Huet and D.S. Lankford. On the uniform halting problem for term rewriting systems. Technical Report Rapport laboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France, 1978. Available at http://www.ens-lyon.fr/LIP/REWRITING/OLD_PUBLICATIONS_ON_TERMINATION (viewed May 2008).

[20] Sam Kamin and Jean-Jacques Levy. Two generalizations of the recursive path ordering. Department of Computer Science, University of Illinois, Urbana, IL. Available at http://www.ens-lyon.fr/LIP/REWRITING/OLD_PUBLICATIONS_ON_TERMINATION (viewed December 2005), 1980.

[21] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1194–1201, 1996.

[22] M. Khrishnamoorthy and P. Narendran. On recursive path ordering. *Theoretical Computer Science*, **40**:323–328, 1985.

[23] Masahito Kurihara and Hisashi Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Innovations in Applied Artificial Intelligence, 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Proceedings*, **3029** of *Lecture Notes in Computer Science*, pages 827–837, Ottawa, Canada, 2004. Springer.

[24] Claude Marché and Hans Zantema. The termination competition. In Franz Baader, editor, *RTA*, **4533** of *Lecture Notes in Computer Science*, pages 303–313. Springer, 2007.

[25] MiniSAT. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat. Viewed December 2005.

[26] Peter Schneider-Kamp, René Thiemann, Elena Annov, Michael Codish, and Jürgen Giesl. Proving termination using recursive path orders and sat solving. In Boris Konev and Frank Wolter, editors, *FroCos*, **4720** of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2007.

[27] S. A. Seshia, S. K. Lahiri, and R.E. Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proceedings of the 40th Design Automation Conference, (DAC '03)*, pages 425–430. ACM Press, June 2003.

[28] SWI-prolog. http://http://www.swi-prolog.org. Viewed December 2005.

[29] Muralidhar Talupur, Nishant Sinha, Ofer Strichman, and Amir Pnueli. Range allocation for separation logic. In Rajeev Alur and Doron Peled, editors, *CAV*, **3114** of *Lecture Notes in Computer Science*, pages 148–161. Springer, 2004.

[30] The termination problems data base. http://www.lri.fr/~marche/tpdb/. Viewed June 20007.

[31] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. 1968. Reprinted in J. Siekmann and G. Wrightson (editors), Automation of Reasoning, vol. 2, pp. 466-483, Springer-Verlag Berlin, 1983.

[32] Tyrolean termination tool. http://cl2-informatik.uibk.ac.at/ttt. Viewed June 2007.

[33] Chao Wang, Franjo Ivancic, Malay Ganai, and Aari Gupta. Deciding separation logic formulae by SAT and incremental negative cycle elimination. In G. Sutcliffe and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence and Reasoning (LPAR '05)*, pages 322–336. Springer-Verlab, 2005.

[34] Jan Wielemaker. An overview of the SWI-Prolog programming environment. In Fred Mesnard and Alexander Serebenik, editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium, December 2003. Katholieke Universiteit Leuven. CW 371.

[35] H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proceedings of the 33rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '07)*, **4362** of *Lecture Notes in Computer Science*, pages 579–590, 2007.

[36] Harald Zankl. SAT techniques for lexicographic path orders. http://arxiv.org/abs/cs.SC/0605021, May 2006.

[37] Harald Zankl and Aart Middeldorp. Satisfying KBO constraints. In F. Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications*, **4533** of *Lecture Notes in Computer Science*, pages 389–403, 2007.