Solving Partial Order Constraints for LPO Termination

Michael Codish[1⋆], Vitaly Lagoon[2], and Peter J. Stuckey[2,3]

[1] Department of Computer Science, Ben-Gurion University, Israel
[2] Department of Computer Science and Software Engineering
The University of Melbourne, Australia
[3] NICTA Victoria Laboratory
mcodish@cs.bgu.ac.il, {lagoon,pjs}@cs.mu.oz.au

Abstract. This paper introduces a new kind of propositional encoding for reasoning about partial orders. The symbols in an unspecified partial order are viewed as variables which 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. We illustrate the application of our approach to determine LPO termination for term rewrite systems. Experimental results are unequivocal, indicating orders of magnitude speedups in comparison with current implementations for LPO termination. The proposed encoding is general and relevant to other applications which involve propositional reasoning about partial orders.

1 Introduction

This paper formalizes a propositional logic over partial orders. Formulæ in this logic are just like usual propositional formulæ except that propositions are 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. We refer to the formulæ of this logic as *partial order constraints*. There are many applications in computer science which involve reasoning about (the satisfiability of) partial order constraints. For example, in the contexts of termination analysis, theorem proving, and planning. The main contribution of this paper is a new kind of propositional encoding of partial order constraints in propositional logic.

Contemporary propositional encodings, such as the one considered in [14], model the atoms (primitive order relations such as $f = g$ or $f > h$ on symbols) in a partial order constraint as propositional variables. Then, propositional statements are added to encode the axioms of partial orders which 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 propose to model the symbols

⋆ Research performed while visiting the University of Melbourne

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.

We focus on the application to termination analysis for term rewrite systems (for a survey see [7]) and in particular on LPO termination [12, 6]. Experimental results are unequivocal, surpassing the performance of current termination analyzers such as TTT [11, 20] and AProVE [9, 2] (configured for LPO). The underlying approach is directly applicable to more powerful termination proving techniques, such as those based on dependency pairs [1], which basically involve the same kind of constraint solving.

Sections 2 and 3 introduce partial order constraints and their symbol-based propositional encoding. Section 4 introduces the LPO termination problem and its relation to partial order constraints. Section 5 describes and evaluates our implementation for LPO termination which is based on the application of a state-of-the-art propositional SAT solver [16]. Finally, we present related work and conclusions.

2 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 models. A model is an assignment of truth values to atoms which is required to satisfy both parts of the formula: the "propositional part" and the "partial order part".

Syntax: Let $\mathcal{F}$ be finite non-empty set of symbols and $\mathcal{R} = \{>, =\}$ consist of two binary relation symbols on $\mathcal{F}$. Since $\mathcal{R}$ is fixed we denote by $Atom_{\mathcal{F}}$ the set of atoms of the form $(f\ R\ g)$ where $R \in \mathcal{R}$ and $f, g \in \mathcal{F}$. 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 $Atom(\varphi)$.

Semantics: The symbols in $\mathcal{R}$ 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 models. Intuitively, 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. Before presenting a formal definition we illustrate this intuition by example.

Example 1. Let $\mathcal{F} = \{f, g, h\}$. The following are partial order constraints:

$$\varphi_1 = (f > g) \wedge ((f > h) \vee (h > f))$$
$$\varphi_2 = (f \geq g) \wedge (g \geq h) \wedge (h \geq g)$$
$$\varphi_3 = (f > g) \wedge \neg((h > g) \vee (f > h))$$

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: φ_1 evaluates to true when assigning the atoms in μ the value "true" and the others the value "false". It satisfies the partial order part: 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). 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:

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

The formula φ_2 has two models:

$$\{ (f = g), (g = f), (g = h), (h = g), (f = h), (h = f), (f = f), (g = g), (h = h) \}$$
$$\{ (f > g), (g = h), (h = g), (f > h), (f = f), (g = g), (h = h) \}$$

Focusing on φ_3 illustrates that there is an additional implicit condition for an assignment to satisfy a partial order constraint. We recall that a partial order can always be extended to a total order. The partial order $\mu = \{ (f > g) \}$ satisfies the propositional part of φ_3 and may appear at first sight to satisfy also the partial order part (it is a partial order). However, no extension of μ to a total order satisfies the propositional part of φ_3 and hence μ will not be considered a model of φ_3. To solve this, we will restrict models to be total orders.

The following formalizes the proposed semantics for partial order constraints.

Definition 1 (assignment, model). *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 "true". Let φ be a partial order constraint on $\mathcal{F}$. We say that an assignment μ is a model for φ if: (1) it makes φ true as a propositional formula; (2) it satisfies the axioms for strict partial order and equality; and (3) it defines a total order on $\mathcal{F}$. More specifically, in (2) and in (3), an assignment μ is required to satisfy (for all $f, g, h \in \mathcal{F}$):*

reflexivity:	$(f = f) \in \mu$					
symmetry:	$(f = g) \in \mu$	$\Rightarrow$	$(g = f) \in \mu$			
asymmetry:	$\neg((f > g) \in \mu \wedge (g > f) \in \mu)$					
transitivity:	$(f > g) \in \mu$	$\wedge$	$(g > h) \in \mu$	$\Rightarrow$	$(f > h) \in \mu$	
	$(f = g) \in \mu$	$\wedge$	$(g = h) \in \mu$	$\Rightarrow$	$(f = h) \in \mu$	
identity:	$(f > g) \in \mu$	$\wedge$	$(g = h) \in \mu$	$\Rightarrow$	$(f > h) \in \mu$	
	$(f = g) \in \mu$	$\wedge$	$(g > h) \in \mu$	$\Rightarrow$	$(f > h) \in \mu$	
comparability:	$(f > g) \in \mu$	$\vee$	$(g > f) \in \mu$	$\vee$	$(f = g) \in \mu$	

Given that we fix the models of a partial order constraint to be total orders, 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 1 is equivalent to $\varphi_3' = (f > g) \wedge (g \geq h) \wedge (h \geq f)$ which is clearly unsatisfiable.

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 model? Similarly to the general SAT problem, the satisfiability of partial order constraints is NP-complete, and the reduction from SAT is straightforward.

Solution-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 2 (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 $\mu : \mathcal{F} \to \{1, \ldots, n\}$. An integer* solution *of φ is an assignment θ which makes φ true under the natural interpretations of $>$ and $=$ on the natural numbers.*

Example 2. Consider again the partial order constraints from Example 1. The assignments mapping $\langle f, g, h\rangle$ to $\langle 3, 2, 2\rangle$, $\langle 3, 1, 1\rangle$ and $\langle 1, 1, 1\rangle$ are solutions for φ_2. But only the first two are solutions to φ_1. The formula φ_3 has no solutions.

Theorem 1. *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 lemmata.

Lemma 1. *Let θ be a solution of φ. The assignment*

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

is a model of φ.

Proof. Clearly μ satisfies both the propositional and partial order parts of φ since the integer relation $>$ is a total order. Hence μ is a model for φ by definition.

Lemma 2. *Let μ be a model of φ on $\mathcal{F}$ with n symbols. Then there exists a solution θ of φ in $\{1, \ldots, n\}$.*

Proof. Assume $\mathcal{F} = \{f_1, \ldots, f_n\}$ and let μ be a model of φ. By asymmetry, identity and comparability, for each $1 \leq i < j \leq n$ exactly one of $f_i > f_j$ or $f_i = f_j$ or $f_j > f_i$ hold. We can linearize the symbols in $\mathcal{F}$: $f_{k_n} \; R_{n-1} \; \cdots R_2 \; f_{k_2} \; R_1 \; f_{k_1}$ where for each $1 \leq i < n$, $R_i \in \{>, =\}$ and $(f_{k_{i+1}} \; R_i \; f_{k_i}) \in \mu$, since μ models transitivity, symmetry, and identity. We can then construct a solution θ, using values from 1 to no more than n, where

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

$$\varphi = ((gt > ge) \vee (- > ge)) \wedge ((ge > gt) \vee (- > gt)) \wedge$$
$$(((+ > *) \wedge (+ > -)) \vee (- > *)) \wedge$$
$$(((* > +) \wedge (* > -)) \vee (- > +)) \wedge (* > +)$$

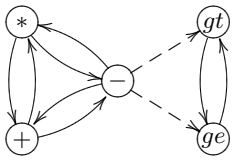

Fig. 1. A partial order constraint (left) and its domain graph (right). The graph has two strongly connected components: $\{gt, ge\}$ and $\{-, *, +\}$. Arcs between the components are dashed.

Decomposing partial order constraint satisfaction: The atoms in a formula φ induce a graph G_φ on the symbols in $\mathcal{F}$ such that φ is satisfiable if and only if the formulae corresponding to the strongly connected components of G_φ are all satisfiable. Considering this graph facilitates the decomposition of a test for satisfiability to a set of smaller instances. This graph captures all possible cycles in the partial order and hence all potential contradictions. The following definition is inspired by [14].

Definition 3 (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 1 illustrates a partial order constraint (a) and its domain graph (b).

Definition 4 (restricting a partial order constraint). *Let φ be a partial order constraint on $\mathcal{F}$ and let $\mathcal{F}' \subseteq \mathcal{F}$. The* restriction *of φ to the symbols in $\mathcal{F}'$ is the formula obtained by substituting "true" for any atom $(f\ R\ g)$ such that $(f,g) \notin \mathcal{F}' \times \mathcal{F}'$. The* SCC-partition *of φ is the set of graphs obtained by restricting φ to the nodes in each of the strongly connected components of G_φ.*

Example 3. Consider the partial order constraint φ and its domain graph G_φ depicted as Figure 1. The graph G_φ has two strongly connected components. The SCC-partition for φ gives:

$$\begin{aligned}
\varphi_1 &= ((gt > ge) \vee true) \wedge ((ge > gt) \vee true) \quad \equiv \quad true \\
\varphi_2 &= (* > +) \wedge (((+ > *) \wedge (+ > -)) \vee (- > *)) \wedge \\
&\qquad ((((* > +) \wedge (* > -))) \vee (- > +)) \\
&\equiv (* > +) \wedge (- > *) \wedge (- > +)
\end{aligned}$$

Lemma 3. *A partial order constraint is satisfiable if and only if each of the formula in its SCC-partition is satisfiable.*

Proof. (idea) You can only get a contradiction if $x > x$ along some path in the graph. Any such path will be contained in a single SCC.

3 A Symbol-based propositional encoding

This section presents a propositional encoding of partial order constraints. A partial order constraint φ on a set of symbols $\mathcal{F}$ is encoded by a propositional formula φ' such that each model of φ corresponds to a model of φ' and in particular such that φ is satisfiable if and only if φ' is. The novelty is to construct the encoding in terms of the solution-based interpretation of partial order constraints. We view the n symbols in $\mathcal{F}$ as integer variables taking finite domain values from the set $\{1, \ldots, n\}$. Each symbol is thus modeled using $k = \lceil \log_2 n \rceil$ propositional variables which encode the binary representation of its value. Constraints of the form $(f > g)$ or $(f = g)$ on $\mathcal{F}$ are interpreted as constraints on integers and it is straightforward to encode them in k-bit arithmetic.

The symbol-based propositional encoding for partial order constraints is defined as follows. For $|\mathcal{F}| = n$ we need $k = \lceil \log_2 n \rceil$ bits per symbol. We denote by $[\![\mathrm{a}]\!]$ the propositional variable corresponding to an atom $a \in Atom_{\mathcal{F}}$ and by $[\![\varphi]\!]$ the propositional formula obtained when replacing atoms by propositional variables in partial order constraint φ.

1. For $f \in \mathcal{F}$, the k-bit representation is $f = \langle f_k, \ldots, f_1 \rangle$ 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

$$\|\varphi\|_k = [\![\varphi]\!] \quad \wedge \bigwedge_{a \in Atoms(\varphi)} ([\![\mathrm{a}]\!] \leftrightarrow \|a\|_k) \tag{1}$$

Proposition 1. *The symbol-based encoding of partial order constraint φ with n symbols involves $O(n \log n)$ propositional variables and $O(|\varphi| \log n)$ connectives.*

Example 4. Consider the partial order constraint $\varphi_2 = (*{>}+) \wedge (-{>}*) \wedge (-{>}+)$ from Example 3. Denote $[\![*{>}+]\!] = p$, $[\![-{>}*]\!] = q$ and $[\![-{>}+]\!] = r$. Each of the three symbols in φ_2 is represented in 2 bits and the propositional encoding of φ_2 is obtained as

$$\begin{aligned} \varphi_2' = p \wedge q \wedge r \wedge (p \leftrightarrow (*_2 \wedge \neg +_2) \vee (*_2 \leftrightarrow +_2 \wedge *_1 \wedge \neg +_1)) \wedge \\ (q \leftrightarrow (-_2 \wedge \neg *_2) \vee (-_2 \leftrightarrow *_2 \wedge -_1 \wedge \neg *_1)) \wedge \\ (r \leftrightarrow (-_2 \wedge \neg +_2) \vee (-_2 \leftrightarrow +_2 \wedge -_1 \wedge \neg +_1)) \end{aligned}$$

The proof of the following theorem is straightforward.

Theorem 2. *A partial order constraint φ on symbols $\mathcal{F}$ is satisfiable if and only if its symbol-based propositional encoding encode(φ) is.*

$$-gt(A,B) \to ge(B,A) \qquad -(A * B) \to (-A) + (-B)$$
$$-ge(A,B) \to gt(B,A) \qquad A * (A + B) \to (A * B) + (A * C)$$
$$-(A + B) \to (-A) * (-B) \qquad (B + C) * A \to (B * A) + (C * A)$$

Fig. 2. An example term rewrite system: normalizing formulæ with propositional connectives: $*$,$+$,$-$ (for: and, or, not); and partial orders: gt, ge (for: $>$,$\geq$).

4 LPO termination

A term rewrite system is a set of rules of the form $\ell \to r$ where ℓ and r are terms constructed from given sets of symbols $\mathcal{F}$ and variables $\mathcal{V}$, and such that ℓ is not a variable and r only contains 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 $r\sigma$. Such an application is called a rewrite step on t. 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 2.

Termination of term rewrite systems is undecidable. 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. There are many methods for defining such orders. Many of them are based on so-called simplification orders and one such order is the lexicographic path order (LPO)[12, 6].

We assume an algebra of terms constructed over given sets of symbols $\mathcal{F}$ and variables $\mathcal{V}$. Let $>_\mathcal{F}$ denote a (strict or non-strict) partial order on $\mathcal{F}$ (a so-called *precedence*) and let $\approx_\mathcal{F}$ denote the corresponding equivalence relation. We denote by $\sim$ the equality of terms up to equivalence of symbols. Observe that if $>_\mathcal{F}$ is strict then $\approx_\mathcal{F}$ and $\sim$ are the identity of symbols and terms respectively. Each precedence $>_\mathcal{F}$ on the symbols induces a lexicographic path order $\succ_{lpo}$ on terms. If for each of the rules $\ell \to r$ in a system, $\ell \succ_{lpo} r$ then the system is LPO terminating.

Definition 5 (LPO [12]). *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 $1 \leq j \leq m$ 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. *$s_i \succsim_{lpo} t$ for some $1 \leq i \leq n$.*

Here $\succ^{lex}_{lpo}$ is the lexicographic extension of $\succ_{lpo}$ to tuples of terms and $\succsim_{lpo}$ is the union of $\succ_{lpo}$ and $\sim$.

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 $\ell \succ_{lpo} r$ for each of the rules with the induced lexicographic path order. There are two variants of the problem: "strict-" and "quasi-LPO termination" depending on if we require $>_\mathcal{F}$ to be strict or not. The corresponding decision problems, strict-

and quasi- LPO termination, are decidable and NP complete [13]. In [10], the authors observe that finding $>_{\mathcal{F}}$ such that $s \succ_{lpo} t$ is tantamount to solving a constraint obtained by unfolding the definition of $s \succ_{lpo} t$ with details depending on whether $>_{\mathcal{F}}$ is a strict or non-strict partial order. The strict- and quasi-LPO termination problems are to decide if conjunctions of these unfoldings are satisfiable — one conjunct for each rule in the given term rewrite system.

Example 5. Consider the term rewrite system of Figure 2. Unfolding Definition 5 for strict-LPO termination, we obtain the following:

$$\begin{aligned}
-(gt(A,B)) >_{lpo} ge(B,A) &\iff (gt > ge) \vee (- > ge)\\
-(ge(A,B)) >_{lpo} gt(B,A) &\iff (ge > gt) \vee (- > gt)\\
-(A+B) >_{lpo} (-(A)) * (-(B)) &\iff ((+ > *) \wedge (+ > -)) \vee (- > *)\\
-(A*B) >_{lpo} (-(A)) + (-(B)) &\iff ((* > +) \wedge (* > -)) \vee (- > +)\\
A*(B+C) >_{lpo} (A*B) + (A*C) &\iff * > +\\
(B+C)*A >_{lpo} (B*A) + (C*A) &\iff * > +
\end{aligned}$$

The term rewrite system is LPO terminating if and only if the conjunction of the constraints on the right sides is satisfiable. This conjunction is precisely the partial order constraint φ from Figure 1 which by Lemma 3, is satisfiable if and only if the formula in its SCC-partition are. Coming back to Example 3, it is straightforward to observe that they are.

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

Example 6. Consider the following term rewrite system.

$div(X,e) \to i(X)$
$i(div(X,Y)) \to div(Y,X)$
$div(div(X,Y),Z) \to div(Y,div(i(X),Z))$

Unfolding Definition 5 for strict-LPO gives

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

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. Unfolding Definition 5 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.

5 Implementation and Experimentation

We have implemented a prototype analyzer, poSAT, for strict- and quasi- LPO termination based on the encoding proposed in Section 3. The implementation

is a written primarily in SWI-Prolog [21, 17] and interfaces the MiniSat solver [8, 16] for solving SAT instances.

We have integrated MiniSat and SWI-Prolog through ≈190 lines of C-code and ≈140 lines of Prolog code. For details concerning this interface see [5]. SAT solvers typically consider propositional formulæ in conjunctive normal form. The transformation of a propositional formula with m connectives and n literals is performed using a (linear) Tseitin transformation [19] (for details on our implementation see [5]) and results in a conjunctive normal form with $O(m + n)$ variables and $O(m)$ clauses.

The rest of poSAT is implemented in ≈700 lines of Prolog code. This includes a parser (for term rewrite systems, modules to translate strict- and quasi-LPO termination problems into partial order constraints, the module converting partial order constraints into SAT instances, and finally a head module processing the command line, running the components, pretty-printing the results etc. The current implementation does not decompose partial order constraints to their SCC-components (Lemma 3). The experimental results indicate that the implementation would not benefit from that: (a) Most of the tests are very fast without this decomposition; and (b) It is typical for hard cases of LPO termination (see Table 2) to have a large strongly connected component including the majority of the symbols.

For experimentation we have taken all 751 term rewrite systems from the Termination Problem Data Base [18] which do not specify a "theory" or a "strategy". In the following, the names of term rewrite systems are indicated in typewriter font and can be found in [18]. We report on the comparison of poSAT for both strict- and quasi- LPO termination analysis with the TTT analyzer[20].

For the experiments, poSAT runs on a 1.5GHz laptop running GNU/Linux FC4. The TTT analyzer is applied via its Web interface [20] and runs on a Xeon 2.24GHz dual-CPU platform which is a considerably faster machine than ours. Experiments with AProVE running on our local (laptop) platform give results which are consistently slower than TTT (on its faster machine). Hence for comparison with poSAT we provide the numbers only for TTT.

With regards to precision, as expected, both analyzers give the same results (with the exception of a single test which TTT cannot handle within the maximum timeout allocation). From the 751 example systems, 128 are LPO terminating and 132 are quasi-LPO terminating. For poSAT, run times include the complete cycle of processing each test: reading and parsing the file, translation to partial order constraints and then to propositional formula, solving by the SAT solver and printing the results. The run time of each test is computed as an average of ten identical runs.

Table 1(a) summarizes the results for strict LPO termination analysis. The columns contain times (in seconds) for our poSAT analyzer and for TTT configured to run with a timeout of 10 minutes (the maximum allowed by its Web interface). Note that the times are taken on different machines which makes the precise comparison impossible. Nevertheless, the results are indicative showing that poSAT is fast in absolute terms and scales better for hard cases. Notably,

	poSAT	TTT
Total	8.983	647.48
Average	0.012	0.86
Max	0.477	317.63

(a) strict LPO termination

	poSAT	TTT
Total	8.609	2167.44
Average	0.011	2.89
Max	0.544	600.00

(b) quasi-LPO termination

Table 1. Summary of experimental results: total, average and maximum times (sec) for 751 tests.

the hardest test of LPO termination for poSAT (`HM/t005.trs`) completes in under a half second, while the hardest test for TTT (`various_14.trs`) takes more than 5 minutes.

Table 1(b) presents the results for quasi-LPO termination analysis. For this variant, poSAT completes the 751 tests in 8.6sec. The same task takes TTT over 34 *minutes* with one test (`currying/Ste92/hydra.trs`) running out of 10 minutes timeout. The next hardest test for TTT is `currying/AG01_No_3.13.trs` which completes in 182.6sec (3min). The same two tests take poSAT 0.054sec and 0.021sec respectively. The hardest quasi-LPO test for poSAT is `Zantema/z30` which takes 0.54sec in our analyzer and 5.02sec in TTT.

Once again, the timings are indicative despite the fact that the two analyzers run on different machines. By comparing the results in Table 1(a) and (b) we observe that for quasi-LPO, TTT runs about an order of magnitude slower than for strict LPO. In contrast, poSAT demonstrates similar performance for both LPO and quasi-LPO.

Table 2 presents a detailed analysis for the 25 most challenging examples for poSAT chosen by maximum total time for strict- and quasi- LPO analysis. The two parts of the table present the respective results for strict- and quasi-LPO termination analyses. The following information is provided: The columns labeled "Sym" and "CNF" characterize the partial order constraints derived from the given term rewrite systems. "Sym" indicates the number of symbols in the complete formula and in the largest component of its SCC-partition (0/0 in this column means that the partial order constraint is trivial i.e., true or false). "CNF" indicates the numbers of propositional variables and clauses in the translation of the propositional (symbol-based) encoding to conjunctive normal form. The columns labeled "poSAT" and "TTT" indicate run times (in seconds) for the poSAT and TTT solvers.

All of the tests in Table 2 are not strict- nor quasi-LPO terminating. This is not surprising for the 25 hardest tests, as proving unsatisfiability is typically harder than finding a solution for a satisfiable formula. It is interesting to note that four examples among the hardest 25, result in trivial partial order constraints. Obviously, the challenge in these examples is not in solving the constraints but rather in obtaining them by unfolding Definition 5. Interestingly, our translation and simplification mechanisms are sometimes more powerful than those of TTT. For instance, `currying/AG01_No_3.13` is simplified to

	LPO				quasi-LPO			
Test	Sym	CNF	poSAT	TTT	Sym	CNF	poSAT	TTT
AProVE_AAECC-ring	28/10	642/2369	0.088	0.11	28/25	786/2951	0.093	0.12
Cime_mucrl1	0/0	0/1	0.298	2.56	0/0	0/1	0.248	13.88
currying_AG01_No_3.13	0/0	0/1	0.127	39.24	0/0	0/1	0.027	184.24
higher-order_Bird_H*	0/0	0/1	0.089	0.15	0/0	0/1	0.025	1.30
HM_t005	0/0	0/1	0.477	11.75	0/0	0/1	0.040	2.13
HM_t009	19/11	773/2779	0.167	0.14	19/18	1388/4880	0.175	0.16
Ex1_2_AEL03_C	19/17	630/2301	0.115	0.23	19/19	1286/4877	0.141	88.30
Ex1_2_AEL03_GM	22/17	506/1805	0.058	0.04	22/22	693/2475	0.060	19.39
Ex26_Luc03b_C	15/12	384/1307	0.055	0.08	15/15	816/2847	0.079	6.00
Ex2_Luc02a_C	15/12	390/1332	0.063	0.08	15/15	838/2939	0.086	6.09
Ex3_3_25_Bor03_C	12/10	285/945	0.050	0.06	12/12	605/2100	0.061	0.72
Ex4_7_37_Bor03_C	13/11	287/962	0.061	0.11	13/13	577/2067	0.072	0.83
Ex5_7_Luc97_C	18/15	614/2181	0.093	0.15	18/18	1341/4871	0.139	92.51
Ex6_15_AEL02_C	23/22	906/3312	0.159	0.37	23/23	1862/6756	0.215	123.47
Ex6_15_AEL02_FR	26/20	599/2146	0.060	0.05	26/26	867/3152	0.065	40.01
Ex6_15_AEL02_GM	29/25	745/2761	0.079	0.07	29/29	1074/3920	0.099	155.26
Ex6_15_AEL02_Z	26/20	587/2105	0.060	0.05	26/26	869/3196	0.061	18.31
Ex7_BLR02_C	14/11	299/1013	0.044	0.07	14/14	627/2289	0.064	1.70
Ex8_BLR02_C	12/10	280/930	0.048	0.07	12/12	546/1906	0.060	0.38
Ex9_BLR02_C	12/9	296/968	0.054	0.06	12/12	608/2071	0.065	0.37
ExAppendixB_AEL03_C	20/18	700/2576	0.121	0.29	20/20	1410/5294	0.152	109.39
ExIntrod_GM99_C	16/13	423/1428	0.080	0.11	16/16	848/3017	0.088	21.36
ExIntrod_Zan97_C	15/12	344/1167	0.051	0.08	15/15	709/2544	0.069	2.02
ExSec11_1_Luc02a_C	16/13	439/1490	0.067	0.12	16/16	985/3353	0.098	29.32
Zantema_z30	2/2	65/106	0.119	2.91	3/3	12827/18205	0.544	5.02

Table 2. The 25 hardest tests for poSAT

false in poSAT but not in TTT, leading to a long search for TTT. The difference is due to the fact that in the case of poSAT the generation of a partial order formula never introduces trivial sub-formula ("true" or "false"), these are evaluated on-the-fly.

Another observation based on the results of Table 2 is that the partial order constraints derived from the tests typically have domain graphs with large strongly-connected components. Almost every test in the table has a "core" component including the majority of the symbols. Therefore, it is unlikely that the performance of poSAT for the presented tests can be improved by using the SCC-based decomposition of the formula.

As Table 2 shows, the maximum CNF instance solved in our tests includes 12827 propositional variables and 18205 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 [16].

6 Related and Future Works

The idea of mapping LPO termination problems to a corresponding propositional formula is first addressed in [14] where the authors assume that partial order constraints contain only disjunction and conjunction of atoms of the form $(f > g)$ (no equality and no negation). This suffices for strict-LPO termination analysis. We present here a generalization of that approach which can be applied also for quasi-LPO termination and then compare it with the approach proposed in this paper.

The basic strategy is the same as in Section 3: to encode a partial order constraint φ on $\mathcal{F}$ by an equivalent propositional formula φ' such that each model of φ corresponds to a model of φ' and in particular such that φ is satisfiable if and only if φ' is. The main difference is that the approach in [14] is "atom-based". The encoding for 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. As in Section 3, we let $[\![\mathrm{a}]\!]$ denote the propositional variable corresponding to an atom $a \in Atom_{\mathcal{F}}$ and $[\![\varphi]\!]$ the propositional formula obtained by replacing each atom a in partial order constraint φ by the propositional variable $[\![\mathrm{a}]\!]$. For a set of symbols $\mathcal{F}$ the following propositional formulæ make the axioms explicit:

- $R^{=}_{\mathcal{F}} = \bigwedge_{f \in \mathcal{F}} [\![\mathrm{f} = \mathrm{f}]\!]$
- $A^{>}_{\mathcal{F}} = \bigwedge_{f,g \in \mathcal{F}} \neg([\![\mathrm{f} > \mathrm{g}]\!] \wedge [\![\mathrm{g} > \mathrm{f}]\!])$
- $T^{=}_{\mathcal{F}} = \bigwedge_{\substack{f,g,h \in \mathcal{F} \\ f \neq g \neq h \neq f}} [\![\mathrm{f} = \mathrm{g}]\!] \wedge [\![\mathrm{g} = \mathrm{h}]\!] \rightarrow [\![\mathrm{f} = \mathrm{h}]\!]$
- $I^{2}_{\mathcal{F}} = \bigwedge_{\substack{f,g,h \in \mathcal{F} \\ f \neq g \neq h \neq f}} [\![\mathrm{f} = \mathrm{g}]\!] \wedge [\![\mathrm{g} > \mathrm{h}]\!] \rightarrow [\![\mathrm{f} > \mathrm{h}]\!]$
- $S^{=}_{\mathcal{F}} = \bigwedge_{f,g \in \mathcal{F}} [\![\mathrm{f} = \mathrm{g}]\!] \rightarrow [\![\mathrm{g} = \mathrm{f}]\!]$
- $T^{>}_{\mathcal{F}} = \bigwedge_{\substack{f,g,h \in \mathcal{F} \\ f \neq g \neq h \neq f}} [\![\mathrm{f} > \mathrm{g}]\!] \wedge [\![\mathrm{g} > \mathrm{h}]\!] \rightarrow [\![\mathrm{f} > \mathrm{h}]\!]$
- $I^{1}_{\mathcal{F}} = \bigwedge_{\substack{f,g,h \in \mathcal{F} \\ f \neq g \neq h}} [\![\mathrm{f} > \mathrm{g}]\!] \wedge [\![\mathrm{g} = \mathrm{h}]\!] \rightarrow [\![\mathrm{f} > \mathrm{h}]\!]$
- $C^{\geq}_{\mathcal{F}} = \bigwedge_{\substack{f,g \in \mathcal{F} \\ f \neq g \neq h \neq f}} [\![\mathrm{f} > \mathrm{g}]\!] \vee [\![\mathrm{g} > \mathrm{f}]\!] \vee [\![\mathrm{f} = \mathrm{g}]\!]$

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 $encode(\varphi) = [\![\varphi]\!] \wedge T^{>}_{\mathcal{F}} \wedge A^{>}_{\mathcal{F}}$ [14]. In the general case when φ may contain also equality the encoding is obtained as

$$encode(\varphi) = [\![\varphi]\!] \wedge R^{=}_{\mathcal{F}} \wedge S^{=}_{\mathcal{F}} \wedge A^{=}_{\mathcal{F}} \wedge T^{>}_{\mathcal{F}} \wedge T^{=}_{\mathcal{F}} \wedge A^{>}_{\mathcal{F}} \wedge I^{1}_{\mathcal{F}} \wedge I^{2}_{\mathcal{F}} \wedge C^{\geq}_{\mathcal{F}} \qquad (2)$$

The two variants of atom-based propositional encodings both result in large propositional formula. For $|\mathcal{F}| = n$ they introduce $O(n^2)$ propositional variables and involve $O(n^3)$ connectives (e.g., for transitivity).

In [14] 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. However, in view of Lemma 3 we may assume that we are testing satisfiability for partial order constraints which have strongly connected domain graphs. Moreover, as indicated by our experimental evaluation (see Table 2), the domain graphs for some of the more challenging examples have strongly connected components with up to 30 symbols.

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 assert that that cycle is not 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, the encoding described in [14] 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.

It is insightful to compare the two encodings of a partial order constraint φ given as Equations (1) and (2). The common part in both encodings is the subformula $[\![\varphi]\!]$ in which atoms are viewed as propositional variables. The difference is that Equation (2) introduces explicit axioms to relate the atoms in a partial order where Equation (1) interprets the n symbols as indices represented in $\lceil \log_2 n \rceil$-bits. This is why the symbol-based encoding introduces $O(n \log n)$ propositional variables instead of $O(n^2)$ 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". Hence the number of connectives is $O(|\varphi| \log n)$ instead of $O(n^3 + |\varphi|)$. Obviously for small n the symbol based encoding can be larger than the atom-based encoding. However, the search space is determined by the number of variables, where the $O(n \log n)$ of the symbol-based encoding is clearly superior to the $O(n^2)$ for the atom-based approach.

An implementation of the atom-based approach of [14] is described in the recent report [22] together with an experimental evaluation and comparison with our symbol-based approach. It shows the symbol-based approach is orders of magnitude faster on its benchmark set.

Testing for satisfiability of partial order constraints comes up in many other applications. First of all in the context of term rewrite systems where LPO is just one example of a simplification order and analyses based on other types of orders may also be encoded into propositional logic. Moreover, for programs which cannot be shown to terminate using these kinds of simplification orders, the dependency pairs approach [1] has proven very successful in generating sets of constraints such that the existence of a (quasi-)order satisfying them is a sufficient condition for termination. Our constraint solving technique is directly applicable and improves considerably the performance of implementations for these techniques. Initial results are described in [**?**].

In practice LPO termination tests are often performed in an incremental fashion, adding constraints to orient the rules in a term rewrite system one rule at a time. Methods that construct a partial order thus seek to incrementally extend that partial order if possible. In our approach, we construct a linearization of the partial order and are hence less likely to be able to extend a previous order to satisfy new constraints. However, both approaches make choices which may have

to be undone to satisfy all constraints. For poSAT, the encoding (which often takes a good proportion of the analysis time) is clearly incremental. Moreover, given the raw speed advantages, and the fact that the hardest instances are unsatisfiable, where incrementality is not useful, we are confident that in an incremental context the poSAT is still superior.

7 Conclusion

We have introduced a new kind of propositional encoding for reasoning about partial orders. Previous works propose to represent the atoms in a formula as propositional variables and to explicitly encode the axioms for partial order. Our novel approach is to interpret the symbols in a formula as finite domain variables corresponding to the indices in the partial order. We illustrate the application of our approach for LPO termination analysis for term rewrite systems. Experimental results are unequivocal indicating orders of magnitude speedups in comparison with current implementations for LPO termination analysis. The proposed technique is directly applicable to more powerful termination proving techniques, such as those based on dependency pairs [1], which basically involve the same kind of constraint solving.

Acknowledgment

We are grateful to Bart Demoen for useful insights regarding the implementation and to Samir Genaim who donated the Prolog parser for term rewrite systems. Jürgen Giesl and Aart Middeldorp assisted with the use of the AProVE and TTT analyzers. Yefim Dinitz and anonymous reviewers provided many useful comments on the presentation.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
2. Automated program verification environment (AProVe). http://www-i2.informatik.rwth-aachen.de/AProVE/. Viewed December 2005.
3. M. Codish, V. Lagoon, P. Schachte, and P. J. Stuckey. Size-change termination analysis in k-bits. In P. Sestoft, editor, *Proceedings of the European Symposium on Programming*, volume 3924 of *LNCS*, pages 230–245. Springer, 2006.
4. M. Codish, V. Lagoon, and P. J. Stuckey. Testing for termination with monotonicity constraints. In M. Gabbrielli and G. Gupta, editors, *Proceedings of the 21st International Logic Programming Conference*, volume 3668 of *LNCS*, pages 326–340. Springer, 2005.
5. M. Codish, V. Lagoon, and P. J. Stuckey. Logic programming with satisfiability. `http://www.cs.bgu.ac.il/~mcodish/Papers/Sources/lpsat.pdf`, (submitted).
6. N. Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1/2):69–116, 1987.
7. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 2435–320. Elsevier and MIT Press, 1990.

8. N. Eén and N. Sörensson. An extensible sat-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003 (Selected Revised Papers)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
9. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In V. van Oostrom, editor, *Proc. of the 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 210–220, Aachen, Germany, 2004. Springer.
10. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In R. Nieuwenhuis, editor, *Proc. of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 311–320, Valencia, Spain, 2003.
11. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In *Proc. of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *LNCS*, pages 175–184, Nara, Japan, 2005. Springer.
12. S. Kamin and J.-J. 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.
13. M. Khrishnamoorthy and P. Narendran. On recursive path ordering. *Theoretical Computer Science*, 40:323–328, 1985.
14. M. Kurihara and H. 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*, volume 3029 of *LNCS*, pages 827–837, Ottawa, Canada, 2004. Springer.
15. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. *ACM SIGPLAN Notices*, 36(3):81–92, 2001. Proceedings of POPL'01.
16. MiniSAT solver. http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat. Viewed December 2005.
17. Swi-prolog. http://http://www.swi-prolog.org/. Viewed December 2005.
18. The termination problems data base. http://www.lri.fr/∼marche/tpdb/. Viewed December 2005.
19. 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.
20. Tyrolean termination tool. http://cl2-informatik.uibk.ac.at/ttt/. Viewed December 2005.
21. J. Wielemaker. An overview of the SWI-Prolog programming environment. In F. Mesnard and A. Serebenik, editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium, Dec. 2003. Katholieke Universiteit Leuven. CW 371.
22. H. Zankl. Sat techniques for lexicographic path orders. `http://arxiv.org/abs/cs.SC/0605021`, May 2006.