POP* and Semantic Labeling using SAT*

Martin Avanzini[1]

Institute of Computer Science, University of Innsbruck, Austria,
martin.avanzini@uibk.ac.at

Abstract. The *polynomial path order* (POP* for short) is a termination method that induces polynomial bounds on the innermost runtime complexity of term rewrite systems (TRSs for short). *Semantic labeling* is a transformation technique used for proving termination.
In this paper, we propose an efficient implementation of POP* together with *finite semantic labeling*. This automation works by a reduction to the problem of *boolean satisfiability*. We have implemented the technique and experimental results confirm the feasibility of our approach. By semantic labeling the analytical power of POP* is significantly increased.

1 Introduction

Term rewrite systems (TRSs for short) provide a conceptually simple but powerful abstract model of computation. In rewriting, proving termination is a long standing research field. Consequently, termination techniques applicable in an automated setting have been introduced quite early. Early research concentrated mainly on direct termination techniques [24]. One such technique is the use of *recursive path orders*, for instance the *multiset path order* (MPO for short) [11]. Nowadays, the emphasis shifted toward transformation techniques like the *dependency pair method* [2] or *semantic labeling* [26]. These methods significantly increase the possibility to automatically verify termination.

Many termination techniques can be used to analyse the complexity of rewrite systems. For instance, Hofbauer was the first to observe that termination via MPO implies the existence of a primitive recursive bound on the *derivational complexity* [15]. Here the derivational complexity of a TRS measures the maximal number of rewrite steps as a function in the size of the initial term. For the study of lower complexity bounds we recently introduced in [4] the *polynomial path order* (POP* for short). This order is in essence a miniaturization of MPO, carefully crafted to induce polynomial bounds on the number of rewrite steps (c.f. Theorem 1), whenever the initial term is argument-normalised (aka *basic*).

In this work, we show how to increase the analytical power of POP* by *semantic labeling* [26]. The idea behind semantic labeling is to label the function symbols of the analysed TRS $\mathcal{R}$ with semantic information so that proving termination of the labeled TRS $\mathcal{R}_{\mathsf{lab}}$ becomes easier. The transformation is termination preserving and reflecting. More precisely, every derivation of $\mathcal{R}$ is simulated

* This research is supported by FWF (Austrian Science Fund) projects P20133.

step-by-step by $\mathcal{R}_{\mathsf{lab}}$. Thus, besides analysing the termination behavior of $\mathcal{R}$, the TRS $\mathcal{R}_{\mathsf{lab}}$ can also be employed for investigating the complexity of $\mathcal{R}$.

In order to obtain the labeled TRS $\mathcal{R}_{\mathsf{lab}}$ from $\mathcal{R}$, one needs to define suitable interpretation- and labeling-functions for all function symbols appearing in $\mathcal{R}$. Naturally, these functions have to be chosen such that the employed direct technique — in our case POP*— is applicable to the labeled system. To find a properly labeled TRS $\mathcal{R}_{\mathsf{lab}}$ automatically, we extend the propositional encoding of POP* presented in [4]. Satisfiability of the constructed formula certifies the existence of a labeled system $\mathcal{R}_{\mathsf{lab}}$ that is compatible with POP*. As we have implemented the technique, the feasibility of our approach is confirmed. Moreover, experimental evidence indicates that the analytical power of polynomial path orders is significantly improved.

The automation of semantic labeling together with some base order is not essentially new. For instance, an automation of semantic labeling together with recursive path orders has already been given in [17]. Unfortunately, this approach is inapplicable in our context as the resulting TRS is usually infinite here. Like many syntactic techniques, soundness of polynomial path orders is restricted to finite TRSs. To achieve that $\mathcal{R}_{\mathsf{lab}}$ is finite, we restrict interpretation- and labeling-functions to finite domains.

We structure the remainder of this paper as follows: In Section 2 we recall basic notions and briefly introduce the reader to polynomial path orders POP*. In Section 3 we show how polynomial path orders together semantic labeling can be efficiently automated. In Section 4 we present experimental results, and we conclude in Section 5.

2 The Polynomial Path Order

We briefly recall the basic concepts of term rewriting, for details [8] provides a good resource. Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature, that is a set of function symbols with associated arities. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$. We write $\trianglelefteq$ for the subterm relation, the converse is denoted by $\trianglerighteq$, the strict part of $\trianglerighteq$ by $\triangleright$.

A *term rewrite system* (*TRS* for short) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a set of rewrite rules $l \to r$ such that $l, r \in \mathcal{T}(\mathcal{F},\mathcal{V})$, $l \notin \mathcal{V}$ and all variables of r also appear in l. In the following, $\mathcal{R}$ always denotes a TRS. If not mentioned otherwise, $\mathcal{R}$ is finite. A binary relation on $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a *rewrite relation* if it is closed under contexts and substitutions. The smallest extension of $\mathcal{R}$ that is a rewrite relation is denoted by $\to_{\mathcal{R}}$. The *innermost rewrite relation* $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is the restriction of $\to_{\mathcal{R}}$ where innermost terms have to be reduced first. The transitive and reflexive closure of a rewrite relation $\to$ is denoted by $\to^*$ and we write $s \to^n t$ for the contraction of s to t in n steps. We say that $\mathcal{R}$ is *(innermost) terminating* if there exists no infinite chain of terms $t_0, t_1, \dots$ such that $t_i \to_{\mathcal{R}} t_{i+1}$ ($t_i \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_{i+1}$) for all $i \in \mathbb{N}$.

The root symbols of left-hand sides of rewrite rules in $\mathcal{R}$ are called *defined symbols* and collected in $\mathcal{D}(\mathcal{R})$, while all other symbols are called *constructor*

symbols and collected in $\mathcal{C}(\mathcal{R})$. A term $f(s_1, \ldots, s_n)$ is *basic* if $f \in \mathcal{D}(\mathcal{R})$ and $s_1, \ldots, s_n \in \mathcal{T}(\mathcal{C}(\mathcal{R}), \mathcal{V})$. We write $\mathcal{T}_{\mathsf{b}}(\mathcal{R})$ for the set of all basic terms over $\mathcal{R}$. If every left-hand side of $\mathcal{R}$ is basic then $\mathcal{R}$ is called *constructor TRS*. Constructor TRSs allow us to model the computation of functions in a very natural way.

Example 1. Consider the constructor TRS $\mathcal{R}_{\mathsf{mult}}$ defined by

$$\begin{aligned} \mathsf{add}(\mathsf{0}, y) &\to y & \mathsf{mult}(\mathsf{0}, y) &\to \mathsf{0} \\ \mathsf{add}(\mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{add}(x, y)) & \mathsf{mult}(\mathsf{s}(x), y) &\to \mathsf{add}(y, \mathsf{mult}(x, y)). \end{aligned}$$

$\mathcal{R}_{\mathsf{mult}}$ defines the function symbols add and mult, i.e. $\mathcal{D}(\mathcal{R}) = \{\mathsf{add}, \mathsf{mult}\}$. Natural numbers are represented using the constructor symbols $\mathcal{C}(\mathcal{R}) = \{\mathsf{s}, \mathsf{0}\}$. Define the *encoding function* $\ulcorner\cdot\urcorner : \Sigma^* \to \mathcal{T}(\mathcal{C}(\mathcal{R}), \varnothing)$ by $\ulcorner 0 \urcorner = \mathsf{0}$ and $\ulcorner n + 1 \urcorner = \mathsf{s}(\ulcorner n \urcorner)$. Then for all $n, m \in \mathbb{N}$, $\mathsf{mult}(\ulcorner n \urcorner, \ulcorner m \urcorner) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} \ulcorner n * m \urcorner$. We say that $\mathcal{R}_{\mathsf{mult}}$ *computes* multiplication (and addition) on natural numbers. For instance, the system admits the innermost rewrite sequence

$$\mathsf{mult}(\mathsf{s}(\mathsf{0}), \mathsf{0}) \xrightarrow{\mathsf{i}} \mathsf{add}(\mathsf{0}, \mathsf{mult}(\mathsf{0}, \mathsf{0})) \xrightarrow{\mathsf{i}} \mathsf{add}(\mathsf{0}, \mathsf{0}) \xrightarrow{\mathsf{i}} \mathsf{0}\ ,$$

computing $1 * 0 = 0$. Note that for the second term, the innermost redex $\mathsf{mult}(\mathsf{0}, \mathsf{0})$ is reduced first.

In [19] it is proposed to conceive the complexity of a rewrite system $\mathcal{R}$ as the complexity of the functions computed by $\mathcal{R}$. Whereas this view falls into the realm of *implicit complexity analysis*, we conceive rewriting under $\mathcal{R}$ as the evaluation mechanism of the encoded function. Thus it is natural to define the runtime complexity based on the number of rewrite steps admitted by $\mathcal{R}$. Let $|s|$ denote the size of a term s. The *(innermost) runtime complexity* of a terminating rewrite system $\mathcal{R}$ is defined by

$$\mathsf{rc}_{\mathcal{R}}(m) = \max\{n \mid \exists s, t.\ s \xrightarrow{\mathsf{i}}{}^n t, s \in \mathcal{T}_{\mathsf{b}}(\mathcal{R}) \text{ and } |s| \leqslant m\}\ .$$

To verify whether the runtime complexity of a rewrite system $\mathcal{R}$ is polynomially bounded, we employ polynomial path order. Inspired by the recursion-theoretic characterization of the polytime functions given in [9], polynomial path orders rely on the separation of *safe* and *normal* inputs. For this, the notion of *safe mappings* is introduced. A safe mapping safe associates with every n-ary function symbol f the set of *safe argument positions*. If $f \in \mathcal{D}(\mathcal{R})$ then $\mathsf{safe}(f) \subseteq \{1, \ldots, n\}$, for $f \in \mathcal{C}(\mathcal{R})$ we fix $\mathsf{safe}(f) = \{1, \ldots, n\}$. The argument positions not included in $\mathsf{safe}(f)$ are called *normal* and denoted by $\mathsf{nrm}(f)$. A precedence is an irreflexive and transitive order on $\mathcal{F}$. The polynomial path order $>_{\mathsf{pop}*}$ is an extension of the auxiliary order $>_{\mathsf{pop}}$, both defined in the following two definitions. Here we write $>^=$ for the reflexive closure of an order $>$, further $(>)_{\mathsf{mul}}$ denotes its multiset-extension (c.f. [8]).

Definition 1. *Let $>$ be a precedence and let safe be a safe mapping. We define the order $>_{\mathsf{pop}}$ inductively as follows: $s = f(s_1, \ldots, s_n) >_{\mathsf{pop}} t$ if one of the following alternatives hold:*

1. $s_i >^{=}_{\mathsf{pop}} t$ *for some* $i \in \{1,\dots,n\}$*, and if* $f \in \mathcal{D}(\mathcal{R})$ *then* $i \in \mathsf{nrm}(f)$*, or*
2. $t = g(t_1,\dots,t_m)$*,* $f \in \mathcal{D}(\mathcal{R})$*,* $f > g$ *and* $s >_{\mathsf{pop}} t_i$ *for all* $1 \leqslant i \leqslant m$*.*

Definition 2. *Let* $>$ *be a precedence and let* safe *be a safe mapping. We define the* polynomial path order $>_{\mathsf{pop}*}$ *inductively as follows:* $s = f(s_1,\dots,s_n) >_{\mathsf{pop}*} t$ *if one of the following alternatives hold:*

1. $s >_{\mathsf{pop}} t$*, or*
2. $s_i >^{=}_{\mathsf{pop}*} t$ *for some* $i \in \{1,\dots,n\}$*, or*
3. $t = g(t_1,\dots,t_m)$*,* $f \in \mathcal{D}(\mathcal{R})$*,* $f > g$ *and*
   - $s >_{\mathsf{pop}*} t_{j_0}$ *for some* $j_0 \in \mathsf{safe}(g)$*, and*
   - *for all* $j \neq j_0$*, either* $s >_{\mathsf{pop}} t_j$ *or* $s \rhd t_j$ *and* $j \in \mathsf{safe}(g)$*, or*
4. $t = f(t_1,\dots,t_m)$*,* $f \in \mathcal{D}(\mathcal{R})$ *and*
   - $[s_{i_1},\dots,s_{i_p}]\ (>_{\mathsf{pop}*})_{\mathsf{mul}}\ [t_{i_1},\dots,t_{i_p}]$ *for* $\mathsf{nrm}(f) = \{i_1,\dots,i_p\}$*, and*
   - $[s_{j_1},\dots,s_{j_q}]\ (>^{=}_{\mathsf{pop}*})_{\mathsf{mul}}\ [t_{j_1},\dots,t_{j_q}]$ *for* $\mathsf{safe}(f) = \{j_1,\dots,j_q\}$*.*

Here $[t_1,\dots,t_n]$ denotes the multiset with elements $t_1,\dots,t_n$. When $\mathcal{R} \subseteq >_{\mathsf{pop}*}$ holds, we say that $>_{\mathsf{pop}*}$ is *compatible* with $\mathcal{R}$. The main theorem from [4] states:

Theorem 1. *Let* $\mathcal{R}$ *be a* finite*, constructor TRS compatible with* $>_{\mathsf{pop}*}$*, i.e.,* $\mathcal{R} \subseteq >_{\mathsf{pop}*}$*. Then the runtime complexity of* $\mathcal{R}$ *is polynomial. The polynomial depends only on the cardinality of* $\mathcal{F}$ *and the sizes of the right-hand sides in* $\mathcal{R}$*.*

We conclude this section by demonstrating the application of POP* on the TRS $\mathcal{R}_{\mathsf{mult}}$. Below we write $\langle i \rangle$ for the i-th case of Definition 2.

Example 2. Reconsider the rewrite system $\mathcal{R}_{\mathsf{mult}}$ from Example 1. Consider the safe mapping safe where the second argument of addition is safe ($\mathsf{safe}(\mathsf{add}) = \{2\}$) and all arguments of multiplication are normal ($\mathsf{safe}(\mathsf{mult}) = \varnothing$). Furthermore, let the precedence $>$ be defined as $\mathsf{mult} > \mathsf{add} > \mathsf{s} > 0$.

In order to verify compatibility for this particular instance $>_{\mathsf{pop}*}$ we need to show that all the rules in $\mathcal{R}_{\mathsf{mult}}$ are strictly decreasing, i.e., $l >_{\mathsf{pop}*} r$ holds for $l \to r \in \mathcal{R}_{\mathsf{mult}}$. To exemplify this, consider the rule $\mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y))$. From $\mathsf{s}(x) >_{\mathsf{pop}*} x$ by rule $\langle 2 \rangle$ we infer $[\mathsf{s}(x)]\ (>_{\mathsf{pop}*})_{\mathsf{mul}}\ [x]$. Furthermore $[y]\ (>^{=}_{\mathsf{pop}*})_{\mathsf{mul}}\ [y]$ holds and thus by rule $\langle 4 \rangle$ we obtain $\mathsf{add}(\mathsf{s}(x), y) >_{\mathsf{pop}*} \mathsf{add}(x, y)$. From $\mathsf{add} > \mathsf{s}$ we finally conclude $\mathsf{add}(\mathsf{s}(x), y) >_{\mathsf{pop}*} \mathsf{s}(\mathsf{add}(x, y))$ by one application of rule $\langle 3 \rangle$. As a consequence of Theorem 1, the number of rewrite steps starting from $\mathsf{mult}(\ulcorner n \urcorner, \ulcorner m \urcorner)$ is polynomially bounded in n and m.

3 A Propositional Encoding of POP* and Finite Semantic Labeling

Before we investigate the propositional encoding of polynomial path orders and semantic labeling, we briefly explain basic notions of semantic labeling as introduced in [26].

Semantics is given to a TRS $\mathcal{R}$ by defining a *model*. A model is an $\mathcal{F}$*-algebra* $\mathcal{A}$, i.e. a carrier A equipped with operations $f_{\mathcal{A}} : A^n \to A$ for every n-ary symbol

$f \in \mathcal{F}$, such that for every rule $l \to r \in \mathcal{R}$ and any *assignment* $\alpha : \mathcal{V} \to A$, the equality $[\alpha]_{\mathcal{A}}(l) = [\alpha]_{\mathcal{A}}(r)$ holds. Here $[\alpha]_{\mathcal{A}}(t)$ denotes the *interpretation* of t with assignment α, recursively defined by

$$[\alpha]_{\mathcal{A}}(t) = \begin{cases} \alpha(t) & \text{if } t \in \mathcal{V} \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}.$$

The system $\mathcal{R}$ is labeled according to a *labeling* ℓ for $\mathcal{A}$, i.e. a set of mappings $\ell_f : A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$.[1] For every assignment α, the mapping $\mathsf{lab}_\alpha(t)$ is defined by

$$\mathsf{lab}_\alpha(t) = \begin{cases} t & \text{if } t \in \mathcal{V} \\ f_a(\mathsf{lab}_\alpha(t_1), \ldots, \mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

where $a = \ell_f([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n))$. The *labeled* TRS $\mathcal{R}_{\mathsf{lab}}$ is obtained by labeling all rules for all assignments α:

$$\mathcal{R}_{\mathsf{lab}} = \{\mathsf{lab}_\alpha(l) \to \mathsf{lab}_\alpha(r) \mid l \to r \in \mathcal{R} \text{ and assignment } \alpha\}.$$

The main theorem from [26] states that $\mathcal{R}_{\mathsf{lab}}$ is terminating if and only if $\mathcal{R}$ is terminating. In particular, it is shown that

$$s \to_{\mathcal{R}} t \quad \Longleftrightarrow \quad \mathsf{lab}_\alpha(s) \to_{\mathcal{R}_{\mathsf{lab}}} \mathsf{lab}_\alpha(t)$$

holds for α an arbitrary assignment.

To simplify the presentation, we consider only algebras $\mathcal{B}$ with carrier $\mathbb{B} = \{\top, \bot\}$ here, although in principle the approach works for arbitrary finite carriers. To encode a Boolean function $b : \mathbb{B}^n \to \mathbb{B}$, we make use of unique propositional atoms $b_{w_1,\ldots,w_n}$ for every sequence of arguments $w_1, \ldots, w_n \in \mathbb{B}^n$. The atom $b_{w_1,\ldots,w_n}$ denotes the result of applying arguments $w_1, \ldots, w_n$ to b. For each sequence $a_1, \ldots, a_n$ of propositional formulas, we denote by $\ulcorner b \urcorner(a_1, \ldots, a_n)$ the following formula: when $n = 0$, we set $\ulcorner b \urcorner = b_\varepsilon$. For $n > 0$, we set

$$\ulcorner b \urcorner(a_1, \ldots, a_n) = \bigwedge_{w_1,\ldots,w_n \in \mathbb{B}^n} \Big((\bigwedge_{i=1}^{n} w_i \leftrightarrow a_i) \to b_{w_1,\ldots,w_n} \Big) .$$

Consider the constraint $\ulcorner b \urcorner(a_1, \ldots, a_n) \leftrightarrow r$, and suppose ν is a satisfying assignment. One easily verifies that the encoded function b satisfies $b(w_1, \ldots, w_n) = \nu(b_{w_1,\ldots,w_n}) = \nu(r)$ for $w_1 = \nu(a_1), \ldots, w_n = \nu(a_n)$. We use this observation below to impose restrictions on interpretation- and labeling-functions.

For every assignment $\alpha : \mathcal{V} \to \mathbb{B}$ and term t appearing in $\mathcal{R}$ we introduce the atoms $\mathrm{int}_{\alpha,t}$ and $\mathrm{lab}_{\alpha,t}$ for $t \notin \mathcal{V}$. The meaning of $\mathrm{int}_{\alpha,t}$ is the result of $[\alpha]_{\mathcal{B}}(t)$ for the encoded model $\mathcal{B}$, $\mathrm{lab}_{\alpha,t}$ denotes the label of the root symbol of the labeled

[1] The definition from [26] allows the labeling of a subset of $\mathcal{F}$ and leave other symbols unchanged. In our context, this has no consequence and simplifies the translation.

term $\mathsf{lab}_\alpha(t)$. To ensure for terms $t = f(t_1, \ldots, t_n)$ and assignments α a correct valuation of $\mathrm{int}_{\alpha,t}$ and $\mathrm{lab}_{\alpha,t}$ respectively, we introduce constraints

$$\begin{aligned}\mathrm{INT}_\alpha(t) &= \mathrm{int}_{\alpha,t} \leftrightarrow \ulcorner f_{\mathcal{B}} \urcorner(\mathrm{int}_{\alpha,t_1}, \ldots, \mathrm{int}_{\alpha,t_n}), \text{ and}\\ \mathrm{LAB}_\alpha(t) &= \mathrm{lab}_{\alpha,t} \leftrightarrow \ulcorner \ell_f \urcorner(\mathrm{int}_{\alpha,t_1}, \ldots, \mathrm{int}_{\alpha,t_n}).\end{aligned}$$

Furthermore, we set $\mathrm{INT}_\alpha(t) = \mathrm{int}_{\alpha,t} \leftrightarrow \alpha(t)$[2] for $t \in \mathcal{V}$. The above constraints have to be enforced for every term appearing in $\mathcal{R}$. This is covered by

$$\mathrm{LAB}(\mathcal{R}) = \bigwedge_\alpha \Big(\bigwedge_{\mathcal{R} \trianglerighteq t} (\mathrm{INT}_\alpha(t) \wedge \mathrm{LAB}_\alpha(t)) \wedge \bigwedge_{l \to r \in \mathcal{R}} (\mathrm{int}_{\alpha,l} \leftrightarrow \mathrm{int}_{\alpha,r}) \Big) \,.$$

Above $\trianglerighteq$ is extended to TRSs in the obvious way: $\mathcal{R} \trianglerighteq t$ if $l \trianglerighteq t$ or $r \trianglerighteq t$ for some rule $l \to r \in \mathcal{R}$. Notice that $\bigwedge_{l \to r \in \mathcal{R}} (\mathrm{int}_{\alpha,l} \leftrightarrow \mathrm{int}_{\alpha,r})$ enforces the model condition.

Assume ν is a satisfying assignment for $\mathrm{LAB}(\mathcal{R})$ and $\mathcal{R}_{\mathsf{lab}}$ denotes the system obtained by labeling $\mathcal{R}$ according to the encoded labeling and model. In order to show compatibility of $\mathcal{R}_{\mathsf{lab}}$ with POP*, we need to find a precedence $>$ and safe mapping safe such that $\mathcal{R}_{\mathsf{lab}} \subseteq >_{\mathsf{pop}*}$ holds for the induced order $>_{\mathsf{pop}*}$. To compare the labeled versions $\mathsf{lab}_\alpha(s)$ and $\mathsf{lab}_\alpha(t)$ of two concrete terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ under a particular assignment α, i.e., to check $\mathsf{lab}_\alpha(s) >_{\mathsf{pop}*} \mathsf{lab}_\alpha(t)$, we define

$$\ulcorner s >_{\mathsf{pop}*} t \urcorner_\alpha = \ulcorner s >^{(1)}_{\mathsf{pop}*} t \urcorner_\alpha \vee \ulcorner s >^{(2)}_{\mathsf{pop}*} t \urcorner_\alpha \vee \ulcorner s >^{(3)}_{\mathsf{pop}*} t \urcorner_\alpha \vee \ulcorner s >^{(4)}_{\mathsf{pop}*} t \urcorner_\alpha.$$

Here $\ulcorner s >^{(i)}_{\mathsf{pop}*} t \urcorner$ refers to the encodings of the case $\langle i \rangle$ from Definition 2. We discuss the cases $\langle 2 \rangle$ – $\langle 4 \rangle$, case $\langle 1 \rangle$, the comparison using the weaker order $>_{\mathsf{pop}}$, is obtained similarly. The above definition relies on the following auxiliary constraints. For every labeled symbol $f_a \in \mathcal{F}_{\mathsf{lab}}$ and argument position i of f, we encode $i \in \mathsf{safe}(f_a)$ by a propositional atom $\mathsf{safe}_{f_a,i}$. For every unlabeled symbol $f \in \mathcal{F}$ and formula a representing the label, the formula $\mathrm{SF}(f_a, i)$ (respectively $\mathrm{NRM}(f_a, i)$) assesses that depending on the valuation of a, the i-th position of $f_\top$ or $f_\bot$ is safe (normal). Similar, for $f, g \in \mathcal{F}$ and propositional formulas a and b, the formula $\ulcorner f_a > g_b \urcorner$ ensures $f_{\nu(a)} > f_{\nu(b)}$ in the precedence for satisfying assignment ν. For the latter, we follow the standard approach of encoding precedences on function symbols, compare for instance [23].

Notice that $s_i = t$ if and only if $\mathsf{lab}_\alpha(s_i) = \mathsf{lab}_\alpha(t)$. Thus case $\langle 2 \rangle$ is perfectly captured by $\ulcorner f(s_1, \ldots, s_n) >^{(2)}_{\mathsf{pop}*} t \urcorner_\alpha = \top$ if $s_i = t$ holds for some s_i. Otherwise, we define $\ulcorner f(s_1, \ldots, s_n) >^{(2)}_{\mathsf{pop}*} t \urcorner_\alpha = \bigvee_{i=1}^n \ulcorner s_i >_{\mathsf{pop}*} t \urcorner_\alpha$. For the encoding of the third clause in Definition 2, we introduce fresh atoms δ_j for each argument position j of g. The formula $\mathrm{one}(\delta_1, \ldots, \delta_m)$ assures that exactly one atom δ_j is true. This particular atom marks the unique safe argument position j of $g(t_1, \ldots, t_m)$ with the strong comparison $\mathsf{lab}_\alpha(s) >_{\mathsf{pop}*} \mathsf{lab}_\alpha(t_j)$ allowed. We express clause

[2] We also use $\top$ and $\bot$ to denote truth and falsity in propositional formulas.

$\langle 3\rangle$ by the propositional formula

$$\begin{aligned}\ulcorner f(s_1,\dots,s_n) >^{(3)}_{\mathsf{pop}*} g(t_1,\dots,t_m)\urcorner_\alpha = \ulcorner f_{\mathrm{lab}_{\alpha,s}} > g_{\mathrm{lab}_{\alpha,t}}\urcorner \wedge \mathrm{one}(\delta_1,\dots,\delta_m)\\ \wedge \bigwedge_{j=1}^{m}\Big(\big(\delta_j \to \ulcorner s >_{\mathsf{pop}*} t_j\urcorner_\alpha \wedge \mathrm{SF}(g_{\mathrm{lab}_{\alpha,t}},j)\big)\\ \wedge \big(\neg\delta_j \to \ulcorner s >_{\mathsf{pop}} t_j\urcorner_\alpha \vee \ulcorner s \rhd t_j\urcorner \wedge \mathrm{SF}(g_{\mathrm{lab}_{\alpha,t}},j)\big)\Big)\end{aligned}$$

for $g \in \mathcal{D}(\mathcal{R})$. Here $\ulcorner s \rhd t_i\urcorner = \top$ when $s \rhd t_i$ holds, and otherwise $\ulcorner s \rhd t_i\urcorner = \bot$. This is justified as the subterm relation is closed under labeling. Note that in the above encoding of clause $\langle 3\rangle$, we assume that the labeled root symbol $f_{\mathrm{lab}_{\alpha,s}}$ is a defined symbol of $\mathcal{R}_{\mathsf{lab}}$. For the case that $f_{\mathrm{lab}_{\alpha,s}}$ is not defined, we add a rule $f_{\mathrm{lab}_{\alpha,s}}(x_1,\dots,x_n) \to \mathsf{c}$ with c a fresh constant to the analysed system $\mathcal{R}_{\mathsf{lab}}$. The latter rule is oriented if we additionally require $f_{\mathrm{lab}_{\alpha,s}} > \mathsf{c}$ in the precedence. For instance, the constraint $\ulcorner \mathsf{mult}(\mathsf{s}(x),y) >^{(3)}_{\mathsf{pop}*} \mathsf{add}(y,\mathsf{mult}(x,y))\urcorner_\alpha$ unfolds to

$$\begin{aligned}&\ulcorner \mathsf{mult}_{l_1} > \mathsf{add}_{l_2}\urcorner_\alpha \wedge \mathrm{one}(\delta_1,\delta_2)\\ &\wedge \big(\ \delta_1 \to \ulcorner \mathsf{mult}(\mathsf{s}(x),y) >_{\mathsf{pop}*} y\urcorner_\alpha \wedge \mathrm{SF}(g_{l_2},1)\big)\\ &\wedge \big(\neg\delta_1 \to \ulcorner \mathsf{mult}(\mathsf{s}(x),y) >_{\mathsf{pop}} y\urcorner_\alpha \vee \top \wedge \mathrm{SF}(g_{l_2},1)\big)\\ &\wedge \big(\ \delta_2 \to \ulcorner \mathsf{mult}(\mathsf{s}(x),y) >_{\mathsf{pop}*} \mathsf{mult}(x,y)\urcorner_\alpha \wedge \mathrm{SF}(g_{l_2},2)\big)\\ &\wedge \big(\neg\delta_2 \to \ulcorner \mathsf{mult}(\mathsf{s}(x),y) >_{\mathsf{pop}} \mathsf{mult}(x,y)\urcorner_\alpha \vee \bot \wedge \mathrm{SF}(g_{l_2},2)\big)\end{aligned}$$

for corresponding labels l_1 and l_2 depending on the encoded model. Additionally we require $\ulcorner \mathsf{mult}_\top > \mathsf{c}\urcorner$ and $\ulcorner \mathsf{mult}_\bot > \mathsf{c}\urcorner$ to orient the added rules.

To encode the final clause $\langle 4\rangle$ from Definition 2, we make use of *multiset covers* [23]. A multiset cover is a pair of total mappings $\gamma\colon \{1,\dots,n\} \to \{1,\dots,n\}$ and $\varepsilon\colon \{1,\dots,n\} \to \mathbb{B}$, encoded using fresh atoms $\gamma_{i,j}$ and ε_i. The underlying idea is that for the comparison $[s_1,\dots,s_n]\ (>^{=}_{\mathsf{pop}*})_{\mathsf{mul}}\ [t_1,\dots,t_n]$ to hold, every term t_j has to be *covered* by some term s_i (encoded by γ_{ij}), either by $s_i = t_j$ or $s_i >_{\mathsf{pop}*} t_j$. The former situation is encoded by $\neg\varepsilon_i$, the latter by ε_i. For the case $s_i = t_j$, s_i must not cover any element besides t_j. We set

$$\ulcorner(\gamma,\varepsilon)\urcorner = \bigwedge_{j=1}^{m}\mathrm{one}(\gamma_{1,j},\dots,\gamma_{n,j}) \wedge \bigwedge_{i=1}^{n}(\varepsilon_i \to \mathrm{one}(\gamma_{i,1},\dots,\gamma_{i,m}))\ .$$

Based on this encoding of multiset covers, case $\langle 4\rangle$ is now expressible as

$$\begin{aligned}\ulcorner f(s_1,\dots,s_n) >^{(4)}_{\mathsf{pop}*} f(t_1,\dots,t_n)\urcorner_\alpha =\\ (\mathrm{lab}_{\alpha,s} \leftrightarrow \mathrm{lab}_{\alpha,t}) \wedge \ulcorner(\gamma,\varepsilon)\urcorner \wedge \bigvee_{i=1}^{n}\big(\mathrm{NRM}(f_{\mathrm{lab}_{\alpha,s}},i) \wedge \neg\varepsilon_i\big)\\ \wedge \bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\Big(\gamma_{i,j} \to \big((\mathrm{SF}(f_{\mathrm{lab}_{\alpha,s}},i) \leftrightarrow \mathrm{SF}(f_{\mathrm{lab}_{\alpha,t}},j))\\ \wedge (\varepsilon_i \to \ulcorner s_i = t_j\urcorner) \wedge (\neg\varepsilon_i \to \ulcorner s_i >_{\mathsf{pop}*} t_j\urcorner_\alpha)\big)\Big)\ .\end{aligned}$$

The constraint $\bigvee_{i=1}^{n}\big(\mathrm{NRM}(f_{\mathrm{lab}_{\alpha,s}}, i) \wedge \neg\varepsilon_i\big)$ is used so that at least one normal argument decreases. Assuming $\mathrm{STRICT}(\mathcal{R})$ and $\mathrm{SM}_{\mathsf{SL}}(\mathcal{R})$ cover the restrictions on the precedence and safe mapping, satisfiability of

$$\mathrm{POP}^*_{\mathrm{SL}}(\mathcal{R}) = \bigwedge_{\alpha} \bigwedge_{l\to r\in\mathcal{R}} \ulcorner l >_{\mathsf{pop}*} r \urcorner_\alpha \wedge \mathrm{SM}(\mathcal{R}) \wedge \mathrm{STRICT}(\mathcal{R}) \wedge \mathrm{LAB}(\mathcal{R})$$

certifies the existence of a model $\mathcal{B}$ and labeling ℓ such that the rewrite system

$$\mathcal{R}'_{\mathsf{lab}} = \mathcal{R}_{\mathsf{lab}} \cup \{f_a(x_1,\ldots,x_n) \to \mathsf{c} \mid f \in \mathcal{D}(\mathcal{R}) \text{ and } f_a \in \mathcal{C}(\mathcal{R}_{\mathsf{lab}})\}$$

is compatible with $>_{\mathsf{pop}*}$. The encoding is sound in the following sense.

Theorem 2. *Suppose the propositional formula* $\mathrm{POP}^*_{\mathrm{SL}}(\mathcal{R})$ *is satisfiable. Then* $\mathcal{R}_{lab} \subseteq >_{\mathsf{pop}*}$ *for some (finite) labeled rewrite system* $\mathcal{R}_{lab}$ *and polynomial path order* $>_{\mathsf{pop}*}$.

Since every rewrite sequence of $\mathcal{R}$ is simulated step-by-step by $\mathcal{R}_{\mathsf{lab}}$ we obtain:

Corollary 1. *Let* $\mathcal{R}$ *be a finite constructor TRS. Suppose the propositional formula* $\mathrm{POP}^*_{\mathrm{SL}}(\mathcal{R})$ *is satisfiable. Then the induced (innermost) runtime complexity of* $\mathcal{R}$ *is polynomial.*

4 Experimental Results

We have implemented the encoding of POP^* with semantic labeling (denoted by $\mathrm{POP}^*_{\mathrm{SL}}$ below) in OCaml. We compare this implementation to the implementation without labeling from [4] (denoted by POP^*) and an implementation of a restricted class of polynomial interpretations (denoted by SMC). To check satisfiability of the obtained formulas we employ the MiniSat SAT-solver [12].

SMC refers to a restrictive class of polynomial interpretations: Every constructor symbol is interpreted by a *strongly linear* polynomial, i.e., a polynomial of shape $P(x_1,\ldots,x_n) = \Sigma_{i=1}^n x_i + c$ with $c \in \mathbb{N}$, $c \geqslant 1$. Furthermore, each defined symbol is interpreted by a *simple-mixed* polynomial $P(x_1,\ldots,x_n) = \Sigma_{i_j\in 0,1} a_{i_1\ldots i_n} x_1^{i_1}\ldots x_n^{i_n} + \Sigma_{i=1}^n b_i x_i^2$ with coefficients in $\mathbb{N}$. Unlike for the general case, these restricted interpretations induce polynomial bounds on the runtime complexity. To find such interpretation functions automatically, we employ cdiprover3 [20].

Table 1 presents experimental results based on two testbeds. Testbed **T** constitutes of the 957 examples from the Termination Problem Database 4.0[3] (TPDB) that were automatically verified terminating in the competition of 2007[4]. Testbed **C** is a restriction of **T** where only constructor TRSs have been considered (449 in total). All experiments were conducted on a PC with 512 MB of RAM and a 2.4 GHz Intel® Pentium™ IV processor.

[3] Available at `http://www.lri.fr/~marche/tpdb`.
[4] C.f. `http://www.lri.fr/~marche/termination-competition/2007/`.

Table 1. Experimental results on TPDB 4.0.

	POP^*		POP^*_{SL}		SMC	
	T	**C**	**T**	**C**	**T**	**C**
Yes	65	41	128	74	156	83
Maybe	892	408	800	370	495	271
Timeout (60 sec.)	0	0	29	5	306	95
Average Time Yes (sec.)	0.037		0.130		0.183	

Table 1 confirms that semantic labeling significantly increases the power of POP^*, yielding comparable results to SMC. What is noteworthy is that the union of yes-instances of the three methods constitutes of 218 examples for testbed **T** and 112 for testbed **C**. For these 112 out of 449 constructor TRSs we are able to conclude a polynomial runtime complexity. Interestingly, POP^*_{SL} and SMC succeed on a quite different range of systems. There are 29 constructor TRSs that only POP^*_{SL} can deal with, whereas 38 constructor yes-instances of SMC cannot be handled by POP^*_{SL}. Table 1 reflects that for both suites SMC runs into a timeout for approximately every fourth system. This indicates that purely semantic methods similar to SMC tend to get impractical when the size of the input system increases. Compared to this, the number of timeouts of POP^*_{SL} is rather low.

We perform various optimizations in our implementation: First of all, the constraints can be reduced during construction. Further, it is beneficial to lazily construct the overall constraint. For example, the formula $\ulcorner f(s_1, \ldots, s_n) >^{(2)}_{\mathsf{pop}*} s_i \urcorner_\alpha$ reduces to $\top$. Hence $\ulcorner f(s_1, \ldots, s_n) >_{\mathsf{pop}*} s_i \urcorner_\alpha = \top$ can be concluded without constructing encodings for the remaining cases in Definition 2. Furthermore, $s >_{\mathsf{pop}*} t$ is doomed to failure if t contains variables not appearing in s. For this case, we replace the corresponding constraint by $\bot$. SAT-solvers expect their input in CNF (worst case exponential in size). We employ the transformation proposed in [21] to obtain an equisatisfiable CNF linear in size. This approach is analogous to Tseitin's transformation [25] but the resulting CNF is usually shorter as the plurality of atoms is taken into account.

5 Conclusion

In this paper we have shown howto automatically verify polynomial runtime complexities of rewrite systems. For that we employ semantic labeling and polynomial path orders. Our automation works by a reduction to SAT and employing a state-of-the-art SAT-solver. To our best knowledge, this is the first SAT encoding of some recursive path order with finite semantic labeling. The experimental results confirm the feasibility of our approach. Moreover, they demonstrate that by semantic labeling we significantly increase the power of polynomial path orders.

Our research seems comparable to [10], where recursive path orders together with strongly linear polynomial quasi-interpretations are employed in the complexity analysis. In particular, they have a fully automatable (but of course incomplete) procedure to verify whether the functions computed by the TRS under consideration are feasibly, i.e., polytime, computable. Opposed to [10], we study the length of derivations here. In [7] it is shown that polynomially bounded innermost runtime-complexity entails polytime computability of the functions defined. As a by-product of Corollary 1, [7] gives us a procedure for the complexity analysis of the functions defined. Finally, we also mention that semantic labeling over a Boolean carrier has been implemented in the termination prover TPA [16], where heuristics are used to find an appropriately labeled TRS $\mathcal{R}_{\mathsf{lab}}$. Unlike their approach, we leave all choices concerning the labeling to a state-of-the-art SAT-solver.

In the meantime, polynomial path orders have been extended in various ways. Inspired by the concept of *predicative recursion and parameter substitution* (see [9]), [6] extends polynomial path orders, widening their applicability. Our integration of semantic labeling naturally translates to this extension. Second, polynomial path orders can also be defined over *quasi-precedences*, compare [5]. Further, in [5] polynomial path orders have been combined with *weak dependency pairs* [14], a version of the dependency pair method suitably adapted for the study of runtime-complexities. In principle, this allows the use of those techniques that were developed in the context of dependency pairs for termination analysis, also for complexity analysis. In [5] we exploit two such techniques, namely *argument filterings* [18] and the *usable rules criterion* [2]. All above mentioned extensions have been implemented in the *Tyrolean Complexity Tool*, an open source complexity analyser for TRSs[5].

Finally, we conclude with an application of our research. There is a long interest in the functional programming community to automatically verify complexity properties of programs. For brevity, we just mention [22,1,10]. Rewriting naturally models the evaluation of functional programs, and the termination behavior of functional programs via transformations to rewrite systems has been extensively studied. For instance, one recent approach is described in [13] where Haskell programs are covered. In joint work with Hirokawa, Middeldorp and Moser [3] we propose a translation from (a pure subset of higher-order) Scheme programs to term rewrite systems. The transformation is designed to be complexity preserving and thus allows the study of the complexity of a Scheme program P by the analysis of the transformed rewrite system $\mathcal{R}$. Hence from compatibility of $\mathcal{R}$ with POP* we can directly conclude that the number of evaluation steps of the Scheme program P is polynomially bounded with respect to the input sizes. All necessary steps can be performed mechanically and thus we arrive at a completely automatic complexity analysis for (a pure subset of) Scheme, and eagerly evaluated functional programs in general.

[5] For further information, see `http://cl-informatik.uibk.ac.at/software/tct/`.

References

1. H. Anderson, S. Khoo, S. Andrei, and B. Luca. Calculating polynomial runtime properties. In *Proc. 3th APLAS*, volume 3780 of *LNCS*, pages 230–246. Springer, 2005.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1-2):133–178, 2000.
3. M. Avanzini, N. Hirokawa, A. Middeldorp, and G. Moser. Proving termination of scheme programs by rewriting. Draft available at `http://cl-informatik.uibk.ac.at/~zini/publications/SchemeTR07.pdf`.
4. M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proc. 9th FLOPS*, volume 4989 of *LNCS*, pages 130–146. Springer, 2008.
5. M. Avanzini and G. Moser. Dependency pairs and polynomial path orders. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 48–62. Springer, 2009.
6. M. Avanzini and G. Moser. Polynomial path orders and the rules of predicative recursion with parameter substitution. In *Proc. 10th WST*, 2009.
7. M. Avanzini and G. Moser. Complexity analysis by graph rewriting. In *Proc. 11th FLOPS*, LNCS. Springer, 2010. To appear.
8. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
9. S. Bellantoni and S. A. Cook. A new recursion-theoretic characterization of the polytime functions. *CC*, 2:97–110, 1992.
10. G. Bonfante, J. Marion, and R. Pchoux. Quasi-interpretation synthesis by decomposition. In *Proc. 4th ICTAC*, volume 4711 of *LNCS*, pages 410–424. Springer, 2007.
11. N. Dershowitz. Orderings for term-rewriting systems. In *20th Annual Symposium on Foundations of Computer Science*, pages 123–131. IEEE, 1979.
12. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. 6th SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
13. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 297–312. Springer, 2006.
14. N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. 4th IJCAR*, volume 5195 of *LNCS*, pages 364–380. Springer, 2008.
15. D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *TCS*, 105(1):129–140, 1992.
16. A. Koprowski. Tpa: Termination proved automatically. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 297–312. Springer, 2006.
17. A. Koprowski and A. Middeldorp. Predictive labeling with dependency pairs using SAT. In *Proc. 21th CADE*, volume 4603 of *LNCS*, pages 410–425. Springer, 2007.
18. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. 1th PPDP*, volume 1702 of *LNCS*, pages 47–61. Springer, 1999.
19. P. Lescanne. Termination of rewrite systems by elementary interpretations. *Formal Aspects of Computing*, 7(1):77–90, 1995.
20. G. Moser and A. Schnabl. Proving quadratic derivational complexities using context dependent interpretations. In *Proc. 19th RTA*, volume 5117 of *LNCS*, pages 276–290. Springer, 2008.
21. D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986.

22. M. Rosendahl. Automatic complexity analysis. In *Proc. 4th FPCA*, pages 144–156, 1989.
23. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. 6th FroCoS*, volume 4720 of *LNCS*, pages 267–282. Springer, 2007.
24. TeReSe. *Term Rewriting Systems*, volume 55 of *CTTCS*. Cambridge University Press, 2003.
25. G. Tseitin. On the complexity of derivation in propositional calculus. *SCML, Part 2*, pages 115–125, 1968.
26. H. Zantema. Termination of term rewriting by semantic labelling. *FI*, 24(1/2):89–105, 1995.