# Predictive Labeling with Dependency Pairs using SAT

Adam Koprowski[1] and Aart Middeldorp[2⋆]

[1] Department of Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[2] Institute of Computer Science
University of Innsbruck
6020 Innsbruck, Austria

**Abstract.** This paper combines predictive labeling with dependency pairs and reports on its implementation. Our starting point is the method of proving termination of rewrite systems using semantic labeling with infinite models in combination with lexicographic path orders. We replace semantic labeling with predictive labeling to weaken the quasi-model constraints and we combine it with dependency pairs (usable rules and argument filtering) to increase the power of the method. Encoding the resulting search problem as a propositional satisfiability problem and calling a state-of-the-art SAT solver yields a powerful technique for proving termination automatically.

## 1 Introduction

Termination is an important topic in term rewriting and over the years many techniques have been developed for proving termination. Recently the emphasis in the field is on automation and since 2004 an annual termination competition is being organized in which automatic termination provers compete on a set of termination problems.

One of the techniques for proving termination is a transformational method of *semantic labeling* due to Zantema [20]. The idea of this method is to give semantics to function symbols and use the semantics to label the function symbols in order that simpler termination methods become applicable. At first in automatic termination provers this method was used, if at all, with finite (typically two elements) model. In [14] the method of automating semantic labeling over infinite domains has been worked out and implemented in the termination prover TPA [13]. The approach was to find a quasi-model for the given rewrite system, transform it to a labeled rewrite system, and apply the recursive path order using the information in the labels to distinguish different occurrences of function symbols depending on their context.

The recent development of *predictive labeling* [11] aims at improving semantic labeling by weakening the quasi-model constraints—it allows to consider only

⋆ Partially supported by FWF (Austrian Science Fund) project P18763.

*usable rules* instead of all rules of the rewrite system under consideration when checking the quasi-model condition and it requires semantics only for the relevant part of the signature.

The first contribution of this paper is to increase the power of predictive labeling with natural numbers by incorporating it in the framework of *dependency pairs* [1, 6]. This requires an extension of the theory of predictive labeling which in [11] was presented for ordinary termination only. Furthermore, since labeling with natural numbers produces infinite systems over infinite signatures, powerful ingredients of the dependency pair method like usable rules with argument filterings are not directly applicable.

The second contribution is to extend the approach of automatically proving termination using semantic labeling with natural numbers, as in [14], to incorporate the improvement of predictive labeling. This is not completely straightforward due to the fact that apart from choosing semantics for function symbols one now also needs to decide which symbols to label which in turn influences the set of usable rules. This greatly enlarges the search space.

The third contribution is the presentation of insights used in the implementation of this approach in TPA. This is the first implementation of the predictive labeling method ever. It uses the increasingly popular method of encoding the search problems resulting from an application of a termination technique as propositional formulas and handing them over to a SAT solver.

The remainder of this paper is organized as follows. In the next section we recall basic notions and starting points of this paper. In Section 3 we present the combination of predictive labeling with dependency pairs. Section 4 describes the SAT encoding of the combination. In Section 5 we present experimental results and we conclude in Section 6.

## 2 Preliminaries

We begin by briefly recalling a few basic notions and refer to [2] for further details on term rewriting. This is followed by a presentation of dependency pairs in Section 2.1 and semantic and predictive labeling in Section 2.2.

We assume a signature $\mathcal{F}$ and a set of variables $\mathcal{V}$ and denote by $\mathcal{T}(\mathcal{F}, \mathcal{V})$ the set of terms over $\mathcal{F}$ and $\mathcal{V}$. By $\mathcal{F}\mathrm{un}(t)$ we denote the set of function symbols and by $\mathcal{V}\mathrm{ar}(t)$ the set of variables occurring in a term $t$. The root symbol of a term $t$ is denoted by $\mathrm{root}(t)$. A *rewrite rule* is a pair $(l, r)$, written $l \to r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $l \notin \mathcal{V}$, and $\mathcal{V}\mathrm{ar}(r) \subseteq \mathcal{V}\mathrm{ar}(l)$. A *term rewriting system* (TRS for short) is a set of rewrite rules. Given a TRS $\mathcal{R}$ and a function symbol $f$, the subset of $\mathcal{R}$ consisting of those rules that have $f$ as root of the left-hand side is denoted by $\mathcal{R}_f$: $\mathcal{R}_f = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) = f\}$. The subterm relation on terms is denoted by $\trianglelefteq$. The *rewrite relation* $\to_{\mathcal{R}}$ of a TRS $\mathcal{R}$ is defined as follows: $s \to_{\mathcal{R}} t$ if there exists a rewrite rule $l \to r \in \mathcal{R}$, a substitution $\sigma$, and a context $C$ such that $s = C[l\sigma]$ and $t = C[r\sigma]$. A TRS $\mathcal{R}$ is called *terminating* if there is no infinite reduction $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$.

We write $\mathcal{SN}$ for the subset of $\mathcal{T}(\mathcal{F},\mathcal{V})$ consisting of all terminating terms and $\mathcal{T}_\infty$ for the set of minimal non-terminating terms, that is, non-terminating terms all of whose arguments are terminating. For a TRS $\mathcal{R}$ and a relation $\succ$ we define $\mathcal{R}_\succ = \{s \to t \in R \mid s \succ t\}$.

*Example 1.* We will use the following TRS from [17] (AProVE/rta1.trs in TPDB [21]) to illustrate various developments throughout this paper:

$$
\begin{array}{llll}
(1) & \mathsf{plus}(\mathsf{0},y) \to y & (5) & \mathsf{plus}(\mathsf{s}(\mathsf{s}(x)),y) \to \mathsf{s}(\mathsf{plus}(x,\mathsf{s}(y))) \\
(2) & \mathsf{plus}(\mathsf{s}(\mathsf{0}),y) \to \mathsf{s}(y) & (6) & \mathsf{plus}(x,\mathsf{s}(\mathsf{s}(y))) \to \mathsf{s}(\mathsf{plus}(\mathsf{s}(x),y)) \\
(3) & \mathsf{ack}(\mathsf{0},y) \to \mathsf{s}(y) & (7) & \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}(x,\mathsf{plus}(y,\mathsf{ack}(\mathsf{s}(x),y))) \\
(4) & \mathsf{ack}(\mathsf{s}(x),\mathsf{0}) \to \mathsf{ack}(x,\mathsf{s}(\mathsf{0})) & &
\end{array}
$$

### 2.1 Dependency Pairs

The dependency pair method [1] is a powerful approach for proving termination of TRSs. The dependency pair framework [6] is a modular reformulation and improvement of this approach. We present a simplified version which is sufficient for our purposes. For further information on dependency pairs and more detailed explanations of the concepts introduced below the reader is referred to [1, 6, 7, 12].

Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. The set of *defined* symbols is defined as $\mathcal{D}_\mathcal{R} = \{\mathrm{root}(l) \mid l \to r \in \mathcal{R}\}$. The signature $\mathcal{F}$ is extended with symbols $f^\sharp$ for every symbol $f \in \mathcal{D}_\mathcal{R}$, resulting in the signature $\mathcal{F}^\sharp$. If $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ with $\mathrm{root}(t)$ defined then $t^\sharp$ denotes the term that is obtained from $t$ by replacing its root symbol with $\mathrm{root}(t)^\sharp$. If $l \to r \in \mathcal{R}$ and $t \trianglelefteq r$ with $\mathrm{root}(t)$ defined then the rule $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. The set of dependency pairs of $\mathcal{R}$ is denoted by $\mathrm{DP}(\mathcal{R})$. A *DP problem* is a pair of TRSs $(\mathcal{P},\mathcal{R})$. The problem is said to be *finite* if there is no infinite sequence $t_1 \to_\mathcal{R}^* s_1 \xrightarrow{\epsilon}_\mathcal{P} t_2 \to_\mathcal{R}^* s_2 \xrightarrow{\epsilon}_\mathcal{P} \cdots$ such that all terms $t_1, t_2, \ldots$ are terminating with respect to $\mathcal{R}$. Here $\epsilon$ in $\xrightarrow{\epsilon}_\mathcal{P}$ denotes that the application of the rule in $\mathcal{P}$ takes place at the root position. The main result underlying the dependency pair approach states that a TRS $\mathcal{R}$ is terminating iff the DP problem $(\mathrm{DP}(\mathcal{R}),\mathcal{R})$ is finite.

*Example 2.* For the TRS presented in Example 1 there are six dependency pairs:

$$
\begin{array}{rl}
(8) & \mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{0}) \to \mathsf{ack}^\sharp(x,\mathsf{s}(\mathsf{0})) \\
(9) & \mathsf{plus}^\sharp(\mathsf{s}(\mathsf{s}(x)),y) \to \mathsf{plus}^\sharp(x,\mathsf{s}(y)) \\
(10) & \mathsf{plus}^\sharp(x,\mathsf{s}(\mathsf{s}(y))) \to \mathsf{plus}^\sharp(\mathsf{s}(x),y) \\
(11) & \mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(x,\mathsf{plus}(y,\mathsf{ack}(\mathsf{s}(x),y))) \\
(12) & \mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{plus}^\sharp(y,\mathsf{ack}(\mathsf{s}(x),y)) \\
(13) & \mathsf{ack}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{ack}^\sharp(\mathsf{s}(x),y)
\end{array}
$$

In order to prove finiteness of a DP problem a number of so-called *DP processors* have been developed. DP processors are functions that take a DP problem as input and return a set of DP problems as output. In order to be employed to prove termination they need to be sound, that is, if all DP problems in a set returned by a DP processor are finite then the initial DP problem is finite.

Below we shortly introduce three key concepts of the dependency pair method that are important for our approach: reduction pairs, argument filtering, and usable rules [1, 7].

A reduction pair $(\gtrsim, >)$ consists of a preorder $\gtrsim$ which is closed under contexts and substitutions and a well-founded order $>$ which is closed under substitutions such that the inclusion $\gtrsim \cdot > \subseteq >$ or the inclusion $> \cdot \gtrsim \subseteq >$ holds. We say that a reduction pair $(\gtrsim, >)$ is $\mathcal{C}_\mathcal{E}$*-compatible* iff $\mathsf{c}(x, y) \gtrsim x$ and $\mathsf{c}(x, y) \gtrsim y$, where $\mathsf{c}$ is a new binary function symbol.

An argument filtering is a mapping $\pi$ that assigns to every $n$-ary function symbol $f$ an argument position $i \in \{1, \dots, n\}$ or a (possibly empty) list $[i_1, \dots, i_m]$ of argument positions with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. Every argument filtering $\pi$ induces a mapping on terms:

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable,} \\ \pi(t_i) & \text{if } t = f(t_1, \dots, t_n) \text{ and } \pi(f) = i, \\ f(\pi(t_{i_1}), \dots, \pi(t_{i_m})) & \text{if } t = f(t_1, \dots, t_n) \text{ and } \pi(f) = [i_1, \dots, i_m]. \end{cases}$$

Given a binary relation $\succ$ and an argument filtering $\pi$, we write $s \succ^\pi t$ iff $\pi(s) \succ \pi(t)$.

Next we introduce the concept of usable rules modulo argument filtering. Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and $\pi$ an argument filtering. We define the set of usable rules for $(\mathcal{P}, \mathcal{R})$ modulo $\pi$ as

$$\mathcal{U}_\pi(\mathcal{P}, \mathcal{R}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}_\pi(t, \mathcal{R})$$

with $\mathcal{U}_\pi(t, \mathcal{R}) = \varnothing$ if $t$ is a variable and

$$\mathcal{U}_\pi(t, \mathcal{R}) = \mathcal{R}_f \cup \bigcup_{l \to r \in \mathcal{R}_f} \mathcal{U}_\pi(r, \mathcal{R} \setminus \mathcal{R}_f) \cup \bigcup_{i \colon \pi(f) = i \vee i \in \pi(f)} \mathcal{U}_\pi(t_i, \mathcal{R} \setminus \mathcal{R}_f)$$

if $t = f(t_1, \dots, t_n)$. We illustrate usable rules on our leading example.

*Example 3.* Consider the TRS $\mathcal{R}$ from Example 1 and its dependency pairs $\mathcal{P}$, as presented in Example 2. Given argument filtering $\pi(\mathsf{ack}^\sharp) = 1$ and $\pi(f) = [1, \dots, n]$ for the remaining symbols $f \in \mathcal{F}^\sharp$, where $n$ is the arity of $f$. Then applying the above definition yields $\mathcal{U}_\pi(\mathcal{P}, \mathcal{R}) = \{3, 4, 7\}$.

**Theorem 4 (Reduction Pair Processor).** *Let $\mathcal{P}$ and $\mathcal{R}$ be (possibly infinite) TRSs. Let $(\gtrsim, >)$ be a $\mathcal{C}_\mathcal{E}$-compatible reduction pair and let $\pi$ be an argument filtering. If $\mathcal{R}$ is finitely branching, $\mathcal{P} = \mathcal{P}_{\gtrsim^\pi} \cup \mathcal{P}_{>^\pi}$, and $\mathcal{U}_\pi(\mathcal{P}, \mathcal{R}) \subseteq \gtrsim^\pi$ then the DP processor $(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P} \setminus \mathcal{P}_{>^\pi}, \mathcal{R})\}$ is sound.* □

In [7] the above theorem is stated and proved for finite TRSs $\mathcal{P}$ and $\mathcal{R}$. In our setting we deal with infinite TRSs obtained by labeling finite TRSs. So finiteness of $\mathcal{R}$ is too restrictive. A careful inspection of the proof in [7] as well as the proofs of related statements in [12, 18] reveals that it is sufficient that $\mathcal{R}$ is finitely branching. The reason is that then the sets $\{t \mid s \to_{\mathcal{R}}^* t\}$ of reducts of terminating terms $s$ are still guaranteed to be finite.

### 2.2 Semantic and Predictive Labeling

We start by presenting semantic labeling [20] in the setting of monotone algebras. Let $\mathcal{R}$ be a TRS over signature $\mathcal{F}$ and let $\mathcal{A} = (A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ be a well-founded weakly monotone $\mathcal{F}$-algebra, that is a quadruple consisting of

- a non-empty carrier set $A$,
- a set of algebra operations on $A$ that are weakly monotone in all coordinates: $f_A(a_1, \ldots, a_i, \ldots, a_n) \gtrsim f_A(a_1, \ldots, b, \ldots, a_n)$ for all $n$-ary $f \in \mathcal{F}$, $a_1, \ldots, a_n, b \in A$, and $i \in \{1, \ldots, n\}$ with $a_i \gtrsim b$,
- a well-founded order $>$ on $A$, and
- a relation $\gtrsim$ such that $> \cdot \gtrsim \subseteq >$ or $\gtrsim \cdot > \subseteq >$.

A weakly monotone *labeling* $\ell$ for $\mathcal{A}$ consists of a set of labels $L_f \subseteq A$ together with a mapping $\ell_f : A^n \to L_f$ for every $n$-ary function symbol $f \in \mathcal{F}$ such that $\ell_f$ is weakly monotone in all coordinates. The labeled signature $\mathcal{F}_{\mathsf{lab}}$ consist of $n$-ary function symbols $f_a$ for every $n$-ary function symbol $f \in \mathcal{F}$ and every label $a \in L_f$ together with all function symbols $f \in \mathcal{F}$ such that $L_f = \varnothing$. We extend an assignment of variables $\alpha\colon \mathcal{V} \to A$ to the mapping $\mathsf{lab}_\alpha : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}_{\mathsf{lab}}, \mathcal{V})$ in the following way:

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

where $a$ denotes the label $\ell_f([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n))$. We extend the relation $>$ on $A$ to $>_{\mathcal{A}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ as follows: $s >_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) > [\alpha]_{\mathcal{A}}(t)$ for all variable assignments $\alpha$; $\gtrsim$ is similarly extended to $\gtrsim_{\mathcal{A}}$. We say that $\mathcal{A}$ is a quasi-model for $\mathcal{R}$ if $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$. We define the TRS $\mathcal{D}\mathrm{ec}$ to consist of all rewrite rules $f_a(x_1, \ldots, x_n) \to f_b(x_1, \ldots, x_n)$ with $f \in \mathcal{F}$ an $n$-ary function symbol, $a, b \in L_f$ with $a > b$, and pairwise different variables $x_1, \ldots, x_n$.

The following is the straightforward generalization of the result for ordinary termination from [20] to DP problems. The only observation is that interpretations of dependency pair symbols do not contribute to labels so by choosing them to be the same constant we get the quasi-model constraints for DP rules for free. We omit the easy proof.

**Theorem 5.** *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ a weakly monotone quasi-model for $\mathcal{R}$, and $\ell$ a weakly monotone labeling for $\mathcal{A}$. The DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ is finite iff the DP problem $(\mathrm{DP}(\mathcal{R})_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is finite.* □

In the remainder of this subsection we recall some definitions pertinent to predictive labeling that are needed for the developments in the next section. Let $\mathcal{R}$ be a TRS. For function symbols $f$ and $g$ we write $f \rhd_{\mathrm{d}} g$ if there exist a rewrite rule $l \to r \in \mathcal{R}$ such that $f = \mathrm{root}(l)$ and $g$ is a function symbol in $\mathcal{F}\mathrm{un}(r)$. Let $\ell$ be a labeling and $t$ a term. We define

$$\mathcal{G}_\ell(t) = \begin{cases} \varnothing & \text{if } t \text{ is a variable,} \\ \mathcal{F}\mathrm{un}(t_1)^* \cup \cdots \cup \mathcal{F}\mathrm{un}(t_n)^* & \text{if } t = f(t_1, \ldots, t_n) \text{ and } L_f \neq \varnothing, \\ \mathcal{G}_\ell(t_1) \cup \cdots \cup \mathcal{G}_\ell(t_n) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } L_f = \varnothing \end{cases}$$

where $F^*$ denotes the set $\{g \mid f \rhd_{\mathrm{d}}^* g \text{ for some } f \in F\}$. Furthermore we define

$$\mathcal{G}_\ell(\mathcal{R}) = \bigcup_{l \to r \in \mathcal{R}} \mathcal{G}_\ell(l) \cup \mathcal{G}_\ell(r)$$

and the set of *usable rules* for $\ell$ is defined as $\mathcal{U}(\ell) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) \in \mathcal{G}_\ell(\mathcal{R})\}$. Typically, $\mathcal{U}(\ell)$ is a proper subset of $\mathcal{R}$. The point of predictive labeling is to replace the condition $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$ in semantic labeling by the easier to satisfy condition $\mathcal{U}(\ell) \subseteq \gtrsim_{\mathcal{A}}$. We illustrate this on an example.

*Example 6.* Consider the TRS $R$ from Example 1. Suppose $L_{\mathsf{s}} \neq \varnothing$ and $L_{\mathsf{plus}} = L_{\mathsf{ack}} = \varnothing$. Then applying the above definition gives $\mathcal{G}_\ell(\mathcal{R}) = \{\mathsf{plus}, \mathsf{s}\}$ and $\mathcal{U}(\ell) = \{1, 2, 5, 6\}$.

The weakly monotone algebras $\mathcal{A} = (A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ used to determine the labeling and to satisfy the quasi-model constraints in connection with predictive labeling (i.e., $\mathcal{U}(\ell) \subseteq \gtrsim_{\mathcal{A}}$), must satisfy the additional property that for every finite subset $X \subseteq A$ there exists a least upper bound $\bigsqcup X$ of $X$ in $A$ (with respect to $\gtrsim$). Such algebras are called $\sqcup$-*algebras* in [11]. The main result of [11] can now be stated.

**Theorem 7.** *Let $\mathcal{R}$ be a finitely branching TRS, $\mathcal{A}$ a weakly monotone $\sqcup$-algebra, and $\ell$ a weakly monotone labeling for $\mathcal{A}$ such that $\mathcal{U}(\ell) \subseteq \gtrsim_{\mathcal{A}}$. If $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}$ is terminating then $\mathcal{R}$ is terminating.* □

Due to the restriction to $\sqcup$-algebras, predictive labeling is less powerful than semantic labeling in theory. However, since the algebras used in current termination tools are $\sqcup$-algebras, in practice predictive labeling is to be preferred as it has the clear advantage of weakening the quasi-model condition; instead of all rules only the usable rules need to be oriented, which brings improvements in proving power as well as efficiency.

## 3 Predictive Labeling and Dependency Pairs

The following theorem constitutes the main theoretical result of this paper.

**Theorem 8.** *Let $\mathcal{R}$ and $\mathcal{P} \subseteq \mathrm{DP}(\mathcal{R})$ be TRSs. Let $\mathcal{A}$ be a weakly monotone $\sqcup$-algebra and $\ell$ a weakly monotone labeling for $\mathcal{A}$ such that $\mathcal{U}(\ell) \subseteq \gtrsim_{\mathcal{A}}$. If $\mathcal{R}$ is finitely branching then the DP processor $(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})\}$ is sound.*

Before presenting a proof sketch, we make some clarifying remarks. The condition $\mathcal{P} \subseteq \mathrm{DP}(\mathcal{R})$ ensures that the root symbols of the rules in $\mathcal{P}$ are dependency pair symbols that occur nowhere else. This implies that we do not have to worry about the semantics of the rules $\mathcal{P}$ and thus $\mathcal{U}(\ell)$ will be a subset of $\mathcal{R}$. Nevertheless, dependency pair symbols in $\mathcal{P}$ can be labeled and this may influence the usable rules. It follows that the definition of $\mathcal{U}(\ell)$ given in the preceding section has to be slightly modified: $\mathcal{U}(\ell) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) \in \mathcal{G}_\ell(\mathcal{P} \cup \mathcal{R})\}$. The $\mathcal{D}\mathrm{ec}$ rules are computed for all labeled symbols in $\mathcal{F}^\sharp$, where $\mathcal{F}$ is the signature of $\mathcal{R}$.

*Proof (sketch).* Suppose the DP processor $(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})\}$ is not sound. So the DP problem $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is finite whereas $(\mathcal{P}, \mathcal{R})$ is not. Hence there exists an infinite sequence

$$t_1 \to_{\mathcal{R}}^* u_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* u_2 \xrightarrow{\epsilon}_{\mathcal{P}} \cdots$$

such that the terms $t_1$, $t_2$, $\ldots$ are terminating with respect to $\mathcal{R}$. Let $\alpha$ be an arbitrary assignment. We recall the following definitions from [11].

- Let $t \in \mathcal{SN}$. The interpretation $[\alpha]^*_{\mathcal{A}}(t)$ is inductively defined as follows:

$$[\alpha]^*_{\mathcal{A}}(t) = \begin{cases} \alpha(t) & \text{if } t \text{ is a variable,} \\ f_{\mathcal{A}}([\alpha]^*_{\mathcal{A}}(t_1), \ldots, [\alpha]^*_{\mathcal{A}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \in \mathcal{G}_\ell, \\ \bigsqcup \{[\alpha]^*_{\mathcal{A}}(u) \mid t \to^+_{\mathcal{R}} u\} & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \notin \mathcal{G}_\ell. \end{cases}$$

- Let $t \in \mathcal{SN} \cup \mathcal{T}_\infty$. The labeled term $\mathsf{lab}^*_\alpha(t)$ is inductively defined as follows:

$$\mathsf{lab}^*_\alpha(t) = \begin{cases} t & \text{if } t \text{ is a variable,} \\ f(\mathsf{lab}^*_\alpha(t_1), \ldots, \mathsf{lab}^*_\alpha(t_n)) & \text{if } L_f = \varnothing, \\ f_a(\mathsf{lab}^*_\alpha(t_1), \ldots, \mathsf{lab}^*_\alpha(t_n)) & \text{if } L_f \neq \varnothing \end{cases}$$

  where $a = \ell_f([\alpha]^*_{\mathcal{A}}(t_1), \ldots, [\alpha]^*_{\mathcal{A}}(t_n))$.
- Given a substitution $\sigma$ such that $\sigma(x) \in \mathcal{SN}$ for all variables $x$, the assignment $\alpha^*_\sigma$ is defined as $[\alpha]^*_{\mathcal{A}} \circ \sigma$ and the substitution $\sigma_{\mathsf{lab}^*_\alpha}$ as $\mathsf{lab}^*_\alpha \circ \sigma$.

We will apply $\mathsf{lab}^*_\alpha(\cdot)$ to the terms in the above sequence. Fix $i \geqslant 1$. Repeated application of Lemma 17 in [11] yields $\mathsf{lab}^*_\alpha(t_i) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \mathsf{lab}^*_\alpha(u_i)$. We have $u_i = l\sigma$ and $t_{i+1} = r\sigma$ for some $l \to r \in \mathcal{P}$. We use Lemma 15 in [11] to obtain

$$\mathsf{lab}^*_\alpha(l\sigma) \to^*_{\mathcal{D}\mathrm{ec}} \mathsf{lab}_{\alpha^*_\sigma}(l)\sigma_{\mathsf{lab}^*_\alpha}.$$

Since $\mathsf{lab}_{\alpha^*_\sigma}(l) \to \mathsf{lab}_{\alpha^*_\sigma}(r) \in \mathcal{P}_{\mathsf{lab}}$, $\mathsf{lab}_{\alpha^*_\sigma}(l)\sigma_{\mathsf{lab}^*_\alpha} \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \mathsf{lab}_{\alpha^*_\sigma}(r)\sigma_{\mathsf{lab}^*_\alpha}$. A variation of Lemma 16 in [11] gives $\mathsf{lab}_{\alpha^*_\sigma}(r)\sigma_{\mathsf{lab}^*_\alpha} = \mathsf{lab}^*_\alpha(r\sigma)$. Putting things together

yields $\mathsf{lab}^*_\alpha(t_i) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \cdot \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \mathsf{lab}^*_\alpha(t_{i+1})$. Hence, the above infinite sequence is transformed into

$$\mathsf{lab}^*_\alpha(t_1) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \cdot \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \mathsf{lab}^*_\alpha(t_2) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \cdot \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \cdots$$

If we can show that the terms $\mathsf{lab}^*_\alpha(t_1), \mathsf{lab}^*_\alpha(t_2), \cdots$ are terminating with respect to $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}$ then the DP problem $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is not finite, providing the desired contradiction. Suppose $\mathsf{lab}^*_\alpha(t_i)$ for some $i$ admits an infinite reduction with respect to $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}$. Because $\mathcal{D}\mathrm{ec}$ is a terminating TRS, there must be infinitely many $\mathcal{R}_{\mathsf{lab}}$-steps in this sequence. If we remove all labels, the $\mathcal{D}\mathrm{ec}$-steps disappear and the $\mathcal{R}_{\mathsf{lab}}$-steps are turned into $\mathcal{R}$-steps. It follows that $t_i$ is not terminating with respect to $\mathcal{R}$. This completes the proof. □

## 4 SAT Encoding

We start this section by giving an outline of the main steps of our termination proving procedure. Afterwards we explain which parts are encoded in SAT and how this is actually achieved.

1. First the dependency pairs of $\mathcal{R}$ are computed. Then the strongly connected components (SCCs) in an over-approximation of the dependency graph of $\mathcal{R}$ are determined.
2. In the next step the subterm criterion [12] is applied in connection with the recursive SCC algorithm [10]. The purpose of this step is to quickly remove components which do not pose a challenge for the termination proof.
3. At this point we deal with a number of problems of the form $(\mathcal{P}, \mathcal{R})$ where $\mathcal{P}$ is a set of dependency pairs of the original TRS $\mathcal{R}$. Both $\mathcal{P}$ and $\mathcal{R}$ are finite systems over a finite signature. Each of these problems is subjected to the following steps.
4. Predictive labeling (Theorem 8) transforms $(\mathcal{P}, \mathcal{R})$ into $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$. This new problem generally consists of infinite systems over infinite signatures.
5. Next we apply the reduction pair processor with argument filtering (Theorem 4). In our implementation this step is specialized by taking LPO as the underlying reduction order. In order to make progress, there must be at least one rule in $\mathcal{P}$ with the property that all its labeled versions in $\mathcal{P}_{\mathsf{lab}}$ are strictly decreasing.
6. In the next step we return to the problem $(\mathcal{P}, \mathcal{R})$ and remove those rules from $\mathcal{P}$ that were identified in the preceding step. Then we repeat the algorithm on the resulting problem from step 2 onward.

We illustrate this procedure on an example.

*Example 9.* We continue with our leading example. The estimated dependency graph has two SCCs: $\{(8), (11), (13)\}$ and $\{(9), (10)\}$. The former is taken care of by two applications of the subterm criterion, first with projection $\pi(\mathsf{ack}^\sharp) = 1$

and then with $\pi(\mathsf{ack}^\sharp) = 2$. So in step 3 the problem $(\mathcal{P}, \mathcal{R})$ with $\mathcal{P} = \{(9), (10)\}$ and $\mathcal{R} = \{(1), \ldots, (7)\}$ remains. We will label function symbol $\mathsf{plus}^\sharp$, so $\mathcal{G}_\ell(\mathcal{P} \cup \mathcal{R}) = \{\mathsf{s}\}$ and thus $\mathcal{U}_\ell(\mathcal{R}) = \varnothing$. Taking the successor function as semantics for $\mathsf{s}$ together with the labeling function $\ell_{\mathsf{plus}^\sharp}(x, y) = x + y$ produces in step 4 the DP problem $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ with $\mathcal{P}_{\mathsf{lab}}$ consisting of the rules

$$\mathsf{plus}^\sharp_{i+j+2}(\mathsf{s}(\mathsf{s}(x)), y) \to \mathsf{plus}^\sharp_{i+j+1}(x, \mathsf{s}(y))$$
$$\mathsf{plus}^\sharp_{i+j+2}(x, \mathsf{s}(\mathsf{s}(y))) \to \mathsf{plus}^\sharp_{i+j+1}(\mathsf{s}(x), y)$$

for all $i, j \geqslant 0$, $\mathcal{R}_{\mathsf{lab}} = \mathcal{R}$, and $\mathcal{D}\mathrm{ec} = \{\mathsf{plus}^\sharp_i(x, y) \to \mathsf{plus}^\sharp_j(x, y) \mid i > j \geqslant 0\}$. The DP problem $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is taken care of in step 5 by the argument filtering $\pi(\mathsf{plus}^\sharp_i) = [\,]$ for all $i$ in combination with the well-founded LPO precedence $\mathsf{plus}^\sharp_i \succ \mathsf{plus}^\sharp_j$ whenever $i > j$. Note that the rules in $\mathcal{R}_{\mathsf{lab}}$ are ignored as they are not usable. Since all rules in $\mathcal{P}_{\mathsf{lab}}$ are strictly decreasing, there is nothing left to do in step 6.

We use SAT for steps 4 and 5 of our algorithm. The starting point of our encoding is the approach to semantic labeling with natural numbers and LPO from [14] and the encoding of specific instances of Theorem 4 in [3, 19]. The main challenges are the encoding of

- the search for interpretations $f_A$ and labeling functions $\ell_f$,
- the choice of function symbols to be labeled and the corresponding computation of usable rules,
- the induced quasi-model constraints,
- the precedence constraints over the infinite signature of the labeled system, and
- the finite branching condition in Theorem 4.

For the first problem, we adopt the SAT encoding of matrix interpretations [5].

To address the second problem we introduce a new propositional variable $\mathsf{L}_f$ for every function symbol $f$ that will indicate whether $L_f \neq \varnothing$. Given those variables we need to compute the set of usable rules for predictive labeling according to the definitions at the end of Section 2.2. To this end we introduce propositional variables $\mathsf{U}_f$ for every defined symbol $f$ of $\mathcal{R}$ indicating whether the rewrite rules defining $f$ are usable. For a DP problem $(\mathcal{P}, \mathcal{R})$ the encoding of usable rules is expressed as:

$$\omega_{\mathrm{UR}}(\mathcal{P}, \mathcal{R}) = \bigwedge_{f \in \mathcal{F}^\sharp} \Big( \mathsf{L}_f \implies \bigwedge_{g \in \Delta_f(\mathcal{P}, \mathcal{R})^*} \mathsf{U}_g \Big)$$

where

$$\Delta_f(\mathcal{P}, \mathcal{R}) = \bigcup_{l \to r \in \mathcal{P} \cup \mathcal{R}} \{g \in \mathcal{F}\mathrm{un}(t) \mid \mathrm{root}(t) = f \text{ and } t \trianglelefteq l \text{ or } t \trianglelefteq r\}.$$

Now the encoding of quasi-model constraints can easily be expressed as

$$\omega_{\mathrm{QM}}(\mathcal{R}) = \bigwedge_{f \in \mathcal{D}_{\mathcal{R}}} \Big( \mathsf{U}_f \implies \bigwedge_{l \to r \in \mathcal{R}_f} \ulcorner [l]_{\mathcal{A}} \gtrsim_{\mathcal{A}} [r]_{\mathcal{A}} \urcorner \Big).$$

Here $\ulcorner \ldots \urcorner$ converts inequalities into formulas. For that we need to be able to compute term interpretations in the algebra $\mathcal{A}$ and compare them by $\gtrsim_{\mathcal{A}}$. To that end we can use any technique following the weakly monotone algebra approach from Section 2.2, like polynomial interpretations [15] or matrix interpretations [5]. In our implementation we use matrix interpretations with dimensions 1 (which correspond to linear polynomial interpretations) and 2. The reader is referred to [5] for details on how the corresponding constraints can be encoded. We note that both polynomial and matrix interpretations give $\sqcup$-algebras, which is required for the soundness of predictive labeling (Theorem 8).

Note that we encode quasi-model constraints for $\mathcal{R}$ but not for $\mathcal{P}$. The reason is that every DP problem $(\mathcal{P}, \mathcal{R})$ encountered during the execution of our algorithm has the property that the root symbols of the left- and right-hand sides of rules in $\mathcal{P}$ are dependency pair symbols, which do not occur elsewhere in the problem. Hence they are not part of $\mathcal{G}_\ell(\mathcal{P} \cup \mathcal{R})$ and consequently no rule from $\mathcal{P}$ is usable.

The next question is how to restrict the spectrum of possible precedence relations for infinite labeled TRSs in such a way that they have finite representation, can be searched for easily, and ensuring their well-foundedness is feasible. With every (unlabeled) function symbol $f$ we associate a pair of natural numbers $(f_{\mathrm{L}}, f_{\mathrm{SL}})$, the *level* and *sublevel* of $f$. Such an assignment induces a precedence $\succ_{\mathcal{F}_{\mathsf{lab}}}$ on labeled function symbols in the following way. Firstly, no matter what the labels are, if the level of $f$ is greater than the level of $g$ then $f \succ_{\mathcal{F}_{\mathsf{lab}}} g$. If two symbols have the same level but the label of $f$ is greater (with respect to $>_{\mathcal{A}}$) than that of $g$ then again $f \succ_{\mathcal{F}_{\mathsf{lab}}} g$. Finally, if the levels and labels of $f$ and $g$ are equal but the sublevel of $f$ is bigger than the sublevel of $g$ then again $f \succ_{\mathcal{F}_{\mathsf{lab}}} g$. Note that $\succ_{\mathcal{F}_{\mathsf{lab}}}$ is well-founded since it is obtained as the lexicographic comparison of three well-founded orders.

The straightforward encoding of the (strict) precedence comparisons is presented below. For the computation of labels and their comparison with $>_{\mathcal{A}}$ and $\gtrsim_{\mathcal{A}}$ we may use any approach following the weakly monotone algebra framework.

$$\ulcorner f_i \succ_{\mathcal{F}_{\mathsf{lab}}} g_j \urcorner \;=\; \ulcorner f_{\mathrm{L}} >_{\mathbb{N}} g_{\mathrm{L}} \urcorner \vee \Big( \ulcorner f_{\mathrm{L}} =_{\mathbb{N}} g_{\mathrm{L}} \urcorner \wedge \mathsf{L}_f \wedge \mathsf{L}_g \wedge \big( \ulcorner i >_{\mathcal{A}} j \urcorner \vee \big( \ulcorner i \gtrsim_{\mathcal{A}} j \urcorner \wedge \ulcorner f_{\mathrm{SL}} >_{\mathbb{N}} g_{\mathrm{SL}} \urcorner \big) \big) \Big)$$

$$\ulcorner f_i \succ_{\mathcal{F}_{\mathsf{lab}}} f_j \urcorner \;=\; \mathsf{L}_f \wedge \ulcorner i >_{\mathcal{A}} j \urcorner$$

The use of sublevels allows us to represent more precedences on the signatures of infinite labeled TRSs, which increases the termination proving power with only a small reduction in efficiency.

A natural question is how this setting for precedences compares to the one from [14]. It is easy to observe that every precedence in our setting has a counterpart in the setting of [14]. The converse is also true. More precisely, for every

well-founded precedence $\succ_{\mathcal{F}_{\mathsf{lab}}}$ from [14] there exists a precedence $\succ'_{\mathcal{F}_{\mathsf{lab}}}$ in our setting (well-founded by definition), such that $\succ_{\mathcal{F}_{\mathsf{lab}}} \subseteq \succ'_{\mathcal{F}_{\mathsf{lab}}}$. So the expressive power of the two approaches is the same.

The last challenge that we address is the requirement of Theorem 4 that the TRS $\mathcal{R}_{\mathsf{lab}}$ is finitely branching. For the type of infinite but well structured, parameterized TRSs obtained by labeling finite TRSs this is easy to check. The only source of violation may be a parameterized (labeled) rule where a single labeled instance of a left-hand sides has infinitely many corresponding labeled right-hand sides. In the case of weakly monotone polynomial interpretations this means that a variable must be present in some labels in the right-hand side but not in any label in the corresponding left-hand side. So we define

$$\omega_{\mathrm{FB}}(\mathcal{R}) = \bigwedge_{l\to r\in\mathcal{R}} \bigwedge_{x\in\mathcal{V}\mathrm{ar}(r)} \big(\Phi_x(r) \implies \Phi_x(l)\big)$$

with

$$\Phi_x(t) = \bigvee_{f(t_1,\ldots,t_n)\trianglelefteq t} \mathsf{L}_f \wedge a > 0$$

Here $a$ is the coefficient of $x$ when (symbolically) computing the label of $f$ in $f(t_1,\ldots,t_n)$. So $\Phi_x(t)$ evaluates to true when the variable $x$ occurs in some label in term $t$. Thus $\omega_{\mathrm{FB}}(\mathcal{R})$ expresses that if the value of $x$ is used for obtaining the label of a function symbol occurring in $r$ then this must also be true for a function symbol occurring in $l$, for every rule $l \to r \in \mathcal{R}$ and every variable $x$ occurring in $r$.

When using matrix interpretations instead of polynomial interpretations we obtain a similar formula $\omega_{\mathrm{FB}}(\mathcal{R})$, only now variables are interpreted as finite vectors of natural numbers. So in addition to $x$ we must also propagate the position in the vector on which the label depends. We omit the straightforward details.

Combining all ingredients gives us now the final formula for executing steps 4 and 5 of our termination procedure simultaneously:

$$\omega_{\mathrm{UR}}(\mathcal{P},\mathcal{R}) \wedge \omega_{\mathrm{QM}}(\mathcal{R}) \wedge \omega_{\mathrm{FB}}(\mathcal{R}) \wedge \omega_{\mathrm{LAB}}(\mathcal{P},\mathcal{R}) \wedge \omega_{\mathrm{LPO}}(\mathcal{P},\mathcal{R})$$

Some clarifying remarks are in order.

The subformula $\omega_{\mathrm{LAB}}(\mathcal{P},\mathcal{R})$ takes care of computing the labels for all occurrences of all function symbols. (Since the choice of which symbols will be actually labeled is left to the SAT solver, the calculation of labels needs to be encoded for *all* symbols.) This is very similar to the encoding of the algebra computations in $\omega_{\mathrm{QM}}(\mathcal{R})$ and actually we can share most of the code.

The subformula $\omega_{\mathrm{LPO}}(\mathcal{P},\mathcal{R})$ is the encoding of the specialization of Theorem 8 to LPO. We adopt the encoding given in [3] but since we deal with infinite systems we use as basic building blocks the precedence comparisons sketched on page 10. We compute usable rules with respect to original (unlabeled) system and assume that all labeled versions of a usable unlabeled rule are usable. This gives a correct over-approximation of the usable rules of the labeled TRS.

One thing that seems to be missing in the above formula is the treatment of the rules in $\mathcal{D}$ec. Indeed they are not part of the formula in any way and that is because in the present setting they can be ignored. Regardless of the argument filtering and the precedence (within the constraints of the level/sublevel encoding), the rules in $\mathcal{D}$ec are all (weakly) oriented, do not contribute to the computation of usable rules, and cannot make the system infinitely branching.

The above formula is given to a SAT solver. Three things can happen as a result:

- the SAT solver returns a satisfying assignment, which is translated back to obtain concrete parameters required to execute steps 4 and 5 of our algorithm, or
- the SAT solver returns "unsatisfiable", in which case we know that our approach is not applicable, or
- the SAT solver runs out of time or other resources, in which case we give up without being able to conclude anything.

## 5 Experimental Results

We implemented the technique described in the preceding section in the termination prover TPA, using the MiniSat SAT solver [4]. In this section we evaluate our method on a number of examples from the Termination Problem Database (TPDB, see [21]). All experiments involving TPA were performed on a machine equipped with an Intel ® Xeon$^{\text{TM}}$ 2.80 GHz processor. Experimental data for other termination tools are taken from the respective publications (due to the difficulty of obtaining those tools in the configuration required for our experiments).

A very natural benchmark for our approach would be the comparison with results from [14]. Unfortunately the substantial difference in the approach to semantic labeling with natural numbers makes any decent comparison difficult. We are convinced however that the direct approach from [14] would be absolutely infeasible for exploring the much larger search space resulting from using arbitrary interpretations with bounded coefficients instead of only a small number of predefined interpretations.

We begin by evaluating two basic ingredients of our implementation, matrix interpretations and LPO with argument filtering (both without semantic or predictive labeling), against reference implementations: [5] for the former and [3] for the latter. Both implementations use more or less the same setup: dependency pairs with usable rules and subterm criterion. The results in Table 1 are based on version 2.0 of the TPDB, more precisely on the 773 TRSs from the termination category of this database. The columns "yes", "time", and "timeout" indicate the number of successful termination proofs, the total time (in seconds) spent on the TRSs in the problem set and the number of timeouts that occurred. We used a 60 seconds time limit.

The experiments for matrix interpretations use $2 \times 2$ matrices, 2 bits for the matrix entries, and 3 bits to represent the values of intermediate results.

**Table 1.** Comparison to other tools.

| technique | tool | yes | time | timeout |
|---|---|---|---|---|
| matrix interpretations | Jambox | 505 | N/A | N/A |
| | TPA | 498 | 3541 | 30 |
| LPO | AProVE | 380 | 193 | 0 |
| | TPA | 372 | 191 | 0 |

**Table 2.** Experiments with TPA on TPDB version 3.2.

| | | 60 seconds timeout | | | 10 minutes timeout | | |
|---|---|---|---|---|---|---|---|
| | technique | yes | time | timeout | yes | time | timeout |
| 1×1 | SL | 440 | 1178 | 2 | 440 | 1351 | 0 |
| | PL | 456 | 1193 | 2 | 456 | 1316 | 0 |
| | PL′ | 426 | 752 | 1 | 426 | 893 | 0 |
| 2×2 | SL | 503 | 6905 | 51 | 506 | 24577 | 30 |
| | PL | 527 | 6906 | 53 | 532 | 25582 | 32 |
| | PL′ | 522 | 5211 | 33 | 524 | 11328 | 8 |

The slightly lower score of TPA compared to Jambox is probably due to a more sophisticated approximation of the dependency graph in the latter. No timing information is given in [5] but the authors write "[...] we took the time limit of 1 minute [...] this time was hardly ever consumed [...] average computation time for all proofs is around 1 second". This suggests that our implementation is far from optimal. Indeed, we did not invest much time in optimizing the encoding. This seems to be a good starting point for improving the results for predictive labeling presented below.

The slightly higher score of AProVE in the LPO experiments is likely due to a different graph approximation algorithm. The execution speeds are almost the same but one needs to keep in mind that the results were obtained on different machines and hence cannot be compared directly.

Table 2 summarizes the experiments performed with TPA on the 864 TRSs in version 3.2 of the TPDB. All experiments were performed with time limits of 60 seconds and 10 minutes. The first group of results is based on semantic/predictive labeling with matrix interpretations of dimension 1 (equivalent to linear polynomials) used for both interpretations and labels:

- SL means semantic labeling (Theorem 5) where all symbols are labeled and all rules are considered for the quasi-model requirement,
- PL stands for predictive labeling and corresponds to the approach described in this paper,
- PL′ is a variant with a simple heuristic for the choice of labeled symbols; instead of leaving this decision to SAT all dependency pair symbols are labeled and only them.

A first observation is that predictive labeling is more powerful than semantic labeling—it proves termination of an additional 16 TRSs—without incurring any significant increase in execution speed. The heuristic brings a considerable speedup at the expense of termination proving power.

For the second group of results we use the same methods as for the first group but now $2 \times 2$ matrices are used for interpretations and labels. Again predictive labeling performs better than semantic labeling. It is interesting to observe that the price in termination proving power of the heuristic is much less than for dimension 1. This can be intuitively explained by the more powerful algebraic structure used for the labeling functions, which makes it possible to put more information in the labels and hence counter the reduced flexibility in the choice of function symbols to label. A similar line of reasoning could lead to the belief that in this case it is also easier to satisfy quasi-model constraints and thereby diminishing the improvement of predictive labeling but the difference of 26 TRSs between SL and PL proves this hypothesis wrong.

It is worth noting that even for the slowest variant (PL with matrices of dimension $2 \times 2$) the average time for successful proof is around 3 seconds. Moreover, there are three TRSs which can be proved terminating using this method but not with any tool that participated in the 2006 Termination Competition: Ex26_Luc03b_Z, Ex49_GM04_FR, and ExSec11_1_Luc02a_GM from the TRCSR subcollection.

## 6 Conclusion and Further Research

In this paper we extended the theory of predictive labeling to a dependency pair setting and we presented the ideas behind the SAT based implementation of this technique in the termination prover TPA. Experimental results confirm the feasibility of our approach. Our technique extends the earlier TPA implementation of semantic labeling with infinite quasi-models described in [14] in several ways:

- the quasi-model restriction is relaxed by using predictive labeling,
- the integration with dependency pairs makes the approach more powerful,
- the SAT encoding enables the use of unrestricted polynomial and matrix interpretations for function symbols, whereas in [14] the interpretations were restricted to a small predefined set in order not to blow up the search space.

There is however one extension in [14] that we fail to cover here and that is the possibility of using min and max as interpretations for binary symbols. This feature really adds power to the whole approach, allowing for instance to easily prove termination of the TRS SUBST [9]. The approach in [14] is to allow at most one binary function symbol to be interpreted as min or max, and to perform a case analysis on all occurrences of that symbol in combination with rule splitting. This approach seems to be difficult to incorporate in our new setting as the whole search procedure is encoded as a SAT problem and we have no way of knowing for which symbols min or max will be chosen and hence cannot do this case analysis and rule splitting in advance. We leave this issue as future work.

An important theoretical question is whether the finite branching condition in Theorem 4 is essential. Disabling the $\omega_{\mathrm{FB}}(\mathcal{R})$ conjunct in our encoding allows to "prove" the termination of two more TRSs from the TPDB. One of these TRSs is presented below.

*Example 10.* Consider the following TRS (Thiemann/factorial1.trs) computing the factorial function:

$$\begin{aligned}
\mathsf{plus}(0,x) &\to x & \mathsf{plus}(\mathsf{s}(x),y) &\to \mathsf{s}(\mathsf{plus}(\mathsf{p}(\mathsf{s}(x)),y))\\
\mathsf{times}(0,y) &\to 0 & \mathsf{times}(\mathsf{s}(x),y) &\to \mathsf{plus}(y,\mathsf{times}(\mathsf{p}(\mathsf{s}(x)),y))\\
\mathsf{p}(\mathsf{s}(0)) &\to 0 & \mathsf{p}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))\\
\mathsf{fac}(0,x) &\to x & \mathsf{fac}(\mathsf{s}(x),y) &\to \mathsf{fac}(\mathsf{p}(\mathsf{s}(x)),\mathsf{times}(\mathsf{s}(x),y))\\
\mathsf{factorial}(x) &\to \mathsf{fac}(x,s(0))
\end{aligned}$$

There are ten dependency pairs and the estimated dependency graph contains four single node SCCs. Only one of them, consisting of the dependency pair

$$\mathsf{fac}^\sharp(\mathsf{s}(x),y) \to \mathsf{fac}^\sharp(\mathsf{p}(\mathsf{s}(x)),\mathsf{times}(\mathsf{s}(x),y))$$

is problematic. It could be solved using matrices of dimension 2 by labeling $\mathsf{s}$ and $\mathsf{p}$, but it is essential that the interpretation of $\mathsf{plus}$ depends on its second argument. Then the label of the root symbol of the right-hand side of the rule $\mathsf{plus}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{plus}(\mathsf{p}(\mathsf{s}(x)),y))$ depends on the assignment to $y$ whereas in the left-hand side there is only one labeled $\mathsf{s}$ symbol with $x$ as its argument so its label is necessarily independent of the value of $y$. This makes $\mathcal{R}_{\mathsf{lab}}$ non-finitely branching and hence the termination proof is out of reach with our approach.

At the end of Section 2.1 we already remarked that the proof of Theorem 4 relies on the finite branching condition. The key idea in the proof goes back to a modularity result for termination of Gramlich [8], in which the same finite branching condition is required. By using a much more complicated construction, Ohlebusch [16] showed that the finite branching condition in the modularity result is not essential. It is worthwhile to investigate whether the proof technique in [16] can be used to generalize Theorem 4.

Needless to say, it is not a single technique but rather a careful combination of techniques that makes a successful termination tool. Hence the effect of combining our approach with other techniques should be investigated.

**Acknowledgments**

We are grateful to Hans Zantema for helpful discussions. The comments of the referees improved the presentation.

## References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236:133–178, 2000.

2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. 13th LPAR*, volume 4246 of *LNAI*, pages 30–44, 2006.
4. N. Eén and N Sörensson. An extensible SAT-solver. In *Proc. 6th SAT*, volume 2919 of *LNCS*, pages 502–518, 2003.
5. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proc. 3rd IJCAR*, volume 4130 of *LNAI*, pages 574–588, 2006.
6. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 11th LPAR*, volume 3452 of *LNCS*, pages 301–331, 2005.
7. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *JAR*, 37(3):155–203, 2006.
8. B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *AAECC*, 5:131–158, 1994.
9. T. Hardin and A. Laville. Proof of termination of the rewriting system SUBST on CCL. *TCS*, 46(2–3):305–312, 1986.
10. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
11. N. Hirokawa and A. Middeldorp. Predictive labeling. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 313–327, 2006.
12. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
13. A. Koprowski. TPA: Termination proved automatically. In *Proc. 17th RTA*, volume 4098 of *LNCS*, pages 257–266, 2006.
14. A. Koprowski and H. Zantema. Automation of recursive path ordering for infinite labelled rewrite systems. In *Proc. 3rd IJCAR*, volume 4130 of *LNAI*, pages 332–346, 2006.
15. D. Lankford. On proving term rewrite systems are noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
16. E. Ohlebusch. On the modularity of termination of term rewriting systems. *TCS*, 136(2):333–360, 1994.
17. R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *AAECC*, 16(4):229–270, 2005.
18. X. Urbain. Modular & incremental automated termination proofs. *JAR*, 32:315–355, 2004.
19. H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proc. 33rd SOFSEM*, volume 4362 of *LNCS*, pages 579–590, 2007.
20. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.
21. Termination problem data base, version 3.2, 2006. `www.lri.fr/~marche/tpdb`.