Automated Implicit Computational Complexity Analysis[⋆]

Martin Avanzini[1] and Georg Moser[2] and Andreas Schnabl[2]

[1] Master Program in Computer Science, University of Innsbruck, Austria, martin.avanzini@student.uibk.ac.at

[2] Institute of Computer Science, University of Innsbruck, Austria {georg.moser,andreas.schnabl}@uibk.ac.at

Abstract. Recent studies have provided many characterisations of the class of polynomial time computable functions through term rewriting techniques. In this paper we describe a (command-line based) system that implements the majority of these techniques and present experimental findings to simplify comparisons.

Introduction Recent studies have provided many characterisations of the class of polynomial time computable functions through term rewriting techniques. In this paper we are concerned with studies that employ term rewriting as abstract model of computation and consequently existing techniques from term rewriting are exploited to characterise several computational complexity classes, cf. [1,2,3,4,5,6]. The use of rewriting techniques opens the way for automatisation and below we describe a system that implements (almost all) known techniques in this area and compares their relative strength. The development of the system described was motivated by [7], where (for the first time) an implementation of complexity characterisation is discussed. For brevity and greater applicability, we restrict our description to those techniques that employs rewrite systems as (abstract) computational model and thus conceives the number of *rewrite steps* (perhaps allowing for a specific rewrite strategy) as suitable complexity measure, i.e. we will fore-mostly consider so-called *additive polynomial interpretations* and *polynomial path orders* as methods, cf. [2,6]. I.e., contrary to techniques like the *light multiset path order* [3] (*LMPO* for short) we prohibit ourselves to use (non-trivial) evaluation methods to show that the given TRS *essentially* describes a polytime computable function. We clarify the subtlety of this restriction through a simple example.

Example 1. Consider the following rewrite system $\mathcal{R}_{\mathsf{bin}}$. This is Example 2.21 in the Steinbach and Kühler's example collection [8]; It gives the first half of Pascal's rule for binomial coefficients.

$$\begin{aligned} \mathsf{bin}(x,\mathsf{0}) &\to \mathsf{s}(\mathsf{0}) \qquad & \mathsf{bin}(\mathsf{s}(x),\mathsf{s}(y)) &\to +(\mathsf{bin}(x,\mathsf{s}(y)),\mathsf{bin}(x,y)) \\ \mathsf{bin}(\mathsf{0},\mathsf{s}(y)) &\to \mathsf{0} & & \end{aligned}$$

[⋆] This research is supported by FWF (Austrian Science Fund) project P20133.

It is easy that the (innermost) derivation height of $\mathsf{bin}(\mathsf{s}^n(0), \mathsf{s}^m(0))$ is exponential in n. Hence the first approach cannot handle this example. On the other hand as the specification of addition $+$ is missing, we can simply *interpret* this symbol as the constant zero function and the function thus "computed" is trivially polytime computable.

In the here presented system all methods introduced in [1,2,3,9,6], thus (almost) all methods proposed in this particular approach to implicit computational complexity theory are implemented. We have not (yet) considered (quasi-friendly) sup-interpretations, cf. [4,5]. Our implementation is (partly) based on the termination prover $\mathsf{T_TT_2}$[3] In particular our (command-line) interface is compatible with $\mathsf{T_TT_2}$ and the modular design of our implementation allows for an immediated integration of the presented system as a plug-in to $\mathsf{T_TT_2}$. In order to compare the different methods proposed in the literature tested their applicability on (subsets of) the Termination Problem Data Base (TPDB for short) that is used in the annual termination competition.[4]Arguably this is an imperfect choice as the TPDB has been designed to test the strength of termination provers in rewriting, not as a testbed to analyse the implicit complexity of TRSs. On the other hand it is the only (relatively large) collection of TRSs that is publicly available and we have taken some effort in selecting an interesting and meaningful subsets of this collection as test-suites.

The above example may suggest that techniques like *additive polynomial interpretations* and *polynomial path orders* need to perform worse in comparison to less restrictive techniques like the LMPO or *quasi-interpretations* [1]. Interestingly our practical findings cannot confirm this: On the studied testbeds the (straightforward) technique of using restricted polynomial interpretations [2] to characterise polytime computable programs outperforms all other techniques in power, see below. One the other hand our experiments also indicate that a noticable part of the TPDB can be handled with the union of the implemented methods.

`section: ‘‘System Description’’ fehlt`

The remainder of this paper is organised as follows. In the next section we will restate the definitions of the central techniques considered and describe the respective implementation in some detail. The following sections deals with the description of the system and present the experimental findings. To keep the presentation short, we assume familiarity with term rewriting [10].

Methods that Directly Classify Polytime In this section, we describe *additive polynomial interpretations* and *polynomial path orders* in more detail. Our first task is to fix what is meant by the function *computed* by a TRS. Let $\ulcorner \cdot \urcorner$ be an encoding function and let $\mathcal{R}$ denote a completely defined, orthogonal TRS. An n-ary function $f\colon (\Sigma^*)^n \to \Sigma^*$ is *computable* by $\mathcal{R}$ if there exists a defined

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

[4] We used version 4.0 of the TPDB, available online at `http://www.lri.fr/~marche/tpdb/`.

function symbol f such that for all $w_1, \dots, w_n, v \in \Sigma^*$: $\mathsf{f}(\ulcorner w_1 \urcorner, \dots, \ulcorner w_n \urcorner) \to^! \ulcorner v \urcorner$ if and only if $f(w_1, \dots, w_n) = v$. On the other hand we say that $\mathcal{R}$ *computes* f, if the function $f\colon (\Sigma^*)^n \to \Sigma^*$ is defined by the above equation. The *runtime complexity* (with respect to $\mathcal{R}$) is defined as follows: $\mathsf{rc}_{\mathcal{R}}(m) = \max\{\mathsf{dl}_{(\mathcal{R}, \xrightarrow{\mathsf{i}})}(t) \mid t = f(t_1, \dots, t_n) \in \mathcal{T}_{\mathsf{b}}$ and $\sum_i^n \mathsf{size}(t_i) \leqslant m\}$, where $\mathcal{T}_{\mathsf{b}}$ collects all constructor-based terms.

A suitable starting point into the classification of polytime computable functions are *polynomial interpretations.* Interestingly polynomial interpretations without further restrictions are too strong (cf. [11]). Hence restricted polynomial interpretations are studied in the literature. A polynomial $P(x_1, \dots, x_n)$ (over the natural numbers) is called *additive* if $P(x_1, \dots, x_n) = x_1 + \cdots + x_n + c$ where $c \in \mathbb{N}$, $c \geqslant 1$. In [12] polynomials are further classified into *linear*, *simple*, and *simple-mixed.* We exploit this classification in the next definition. A polynomial interpretation $\mathcal{A}$ is called *simple-mixed, constructor-restricted* (*SMC* for short) if all interpretation functions $c_{\mathcal{A}}$ are additive polynomials, whenever $c \in \mathcal{C}$. The following theorem is an easy consequence of [2, Lemma 3].

Theorem 2. *Let $\mathcal{R}$ be a finite, constructor TRS compatible with an SMC-interpretations. Then the runtime complexity with respect to $\mathcal{R}$ is polynomial*

As a direct consequence of Theorem 2 we obtain that SMC-interpretations induced polytime computable functions, cf. [2, Theorem 4].

Theorem 3. *The functions computable by a constructor and orthogonal TRS that is compatible with a SMC-interpretations is polytime computable.*

In the implementation of this technique we follow well-established methods stemming from termination analysis. In [12] Contejean et al. describe how-to mechanise the search for polynomial interpretation through *constraint propagation.* A central idea is the use of abstract polynomial interpretations. For SMC, that is: $f_{\mathcal{A}}(x_1, \dots, x_n) = \sum_{i_j} a_{f,i_1,\dots,i_n} x_1^{i_1} \cdot \dots \cdot x_n^{i_n} + \sum_{i=1}^n b_i x_i^2$ if $f \in \mathcal{D}$ and $f_{\mathcal{A}}(x_1, \dots, x_n) = \sum_{i=1}^n x_i + c + 1$, otherwise. Here the degree of the monomial $x_1^{i_1} \cdot \dots \cdot x_n^{i_n}$ is at most 1. The variables $a_{f,i_1,\dots,i_n}$ ($i_j \in \{0, 1\}$) and c are called *coefficient variables.*

Given such abstract interpretations we transform the compatibility and monotonicity tests into Diophantine (in)equalities in the coefficient variables. It is well-known that solving Diophantine (in)equalities in undecidable in general, but as an easy remedy, we put an upper bound on the coefficient variables. This makes the problem finite, allowing it to be transformed into a satisfiable problem of proposition logic. The encoding as SAT problem essentially follows[13] and we employ the following (straightforward) optimisation steps: (i) if possible Boolean expressions are simplified by applying simplifications of partially evaluated connectives and (ii) in the encoding of natural numbers sequences of propositional formulas are employed. The maximum value that can be possibly reached in such a sequence is recorded, which to minimise the number of bits needed in the encoding. To actually solve the satisfiability problem, MiniSAT[5]

[5] `http://minisat.se`.

is invoked. The finally obtained satisfying assignment is used to instantiate the values of coefficient variables suitably.

Recently in [6] a restriction of the multiset path order, called *polynomial path order* (*POP** for short) has been introduced. In [6] POP* is defined for a strict precedence. Below we extend this definition to quasi-precedences $\succsim$, whose equivalence part is denoted as $\sim$. If $f \sim g$ then the arity of symbol f equals the arity of g. POP* relies 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 $\mathsf{nrm}(f)$.

Let $\succsim$ be a quasi-precedence and safe a safe mapping respecting $\succsim$. The polynomial path order $\succsim_{\mathsf{pop}*}$ is an extension of the auxiliary order $\succsim_{\mathsf{pop}}$. In order to define $\succsim_{\mathsf{pop}*}$ we give the definition of its strict part $\succ_{\mathsf{pop}*}$ and its equivalence part $\sim_{\mathsf{pop}*}$ separately. I.e., we define $f(s_1,\ldots,s_n) \sim_{\mathsf{pop}*} g(t_1,\ldots,t_n)$ ($f(s_1,\ldots,s_n) \sim_{\mathsf{pop}} g(t_1,\ldots,t_n)$) if and only if $f \sim g$, the the safe and normal part of the multisets $[s_1,\ldots,s_n]$ and $[t_1,\ldots,t_n]$ are equal (over $\sim_{\mathsf{pop}*}$), and $\mathsf{safe}(f) = \mathsf{safe}(g)$. We define the order $\succ_{\mathsf{pop}}$ inductively as follows: $s = f(s_1,\ldots,s_n) \succ_{\mathsf{pop}} t$ if one of the following alternatives hold:

1. $f \in \mathcal{C}(\mathcal{R})$ and $s_i \succsim_{\mathsf{pop}} t$, for some $i \in \{1,\ldots,n\}$, or
2. $s_i \succsim_{\mathsf{pop}} t$ for some $i \in \mathsf{nrm}(f)$, or
3. $t = g(t_1,\ldots,t_m)$ with $f \in \mathcal{D}(\mathcal{R})$ and $f \succ g$ and $s \succ_{\mathsf{pop}} t_i$ for all $1 \leqslant i \leqslant m$.

Based on $\succ_{\mathsf{pop}}$ we define the *polynomial path order* $\succ_{\mathsf{pop}*}$ inductively as follows: $s = f(s_1,\ldots,s_n) \succ_{\mathsf{pop}*} t$ if either

1. $s \succ_{\mathsf{pop}} t$, or
2. $s_i \succsim_{\mathsf{pop}*} t$ for some $i \in \{1,\ldots,n\}$, or
3. $t = g(t_1,\ldots,t_m)$, with $f \in \mathcal{D}(\mathcal{R})$, $f \succ g$, and the following properties hold:
   – $s \succ_{\mathsf{pop}*} t_{i_0}$ for some $i_0 \in \mathsf{safe}(g)$ and
   – either $s \succ_{\mathsf{pop}} t_i$ or $s \rhd t_i$ and $i \in \mathsf{safe}(g)$ for all $i \neq i_0$, or
4. $t = g(t_1,\ldots,t_n)$, $f \sim g$ and for $\mathsf{nrm}(f) = \{i_1,\ldots,i_p\}$, $\mathsf{safe}(f) = \{j_1,\ldots,j_q\}$ both $[s_{i_1},\ldots,s_{i_p}]\ (\succ_{\mathsf{pop}*})_{\mathsf{mul}}\ [t_{i_1},\ldots,t_{i_p}]$ and $[s_{j_1},\ldots,s_{j_q}]\ (\succsim_{\mathsf{pop}*})_{\mathsf{mul}}\ [t_{j_1},\ldots,t_{j_q}]$ holds.

Here $(\succ_{\mathsf{pop}*})_{\mathsf{mul}}$ denotes the multiset extension of $\succ_{\mathsf{pop}*}$. We arrive at the following theorem, whose proof is an easy consequence of the main result established in [6].

Theorem 4. *Let $\mathcal{R}$ be a finite, constructor TRS compatible with $\succ_{\mathsf{pop}*}$, i.e., $\mathcal{R} \subseteq \succ_{\mathsf{pop}*}$. Then the induced runtime complexity is polynomial.*

In order to increase the applicability of POP* we employ *finite* semantic labelling [14]. More precisely we employ Boolean models in the labelling. It is well-known that semantic labelling is non-termination preserving and it is easy to show that semantic labelling (using arbitrary models) doesn't affect the maximal length of derivations. However, in order apply Theorem 4 it is mandatory to restrict to *finite* models. For the next theorem, we need to restrict to a *simple* signature, see [3,6] for a definition. A sufficient, but not necessary condition for a signature to be simple, is that all constructors are unary. For a proof of Theorem 5, the reader is referred to [15].

Theorem 5. *Each finite, orthogonal and constructor TRS based on a simple signature that is compatible with POP* is computable in polynomial time and vice versa each polynomial computable function is computable by a finite, left-linear, orthogonal and constructor TRS compatible with POP* that is based on a simple signature.*

To prove compatibility of a given TRS $\mathcal{R}$ with recursive path orders we have to find a quasi-precedence $\succsim$ such that the induced order is compatible with $\mathcal{R}$. When we want to orient $\mathcal{R}$ by a polynomial path order $\succ_{\mathsf{pop*}}$ we additionally require a suitable *safe mapping*. The latter requirement turned out to be a challenging task. We encode the constraint $s \succ_{\mathsf{pop*}} t$ into a propositional formula and employ recent advances in the use of logic-based techniques in termination analysis, see for example [16,17]. A careful analysis revealed that in order to encode safe mappings the notion of *multiset covers* [16], originally invented to deal with multiset comparison can be adapted, cf. [6].

As the principal idea of the encoding is already described in [6], for brevity we focus on the encoding of semantic labelling with Boolean models together with POP* for strict precedence. To encode a Boolean function $b \colon \mathbb{B}^n \to \mathbb{B}$, we make use of propositional atoms b_w for every sequence of arguments $w = w_1, \ldots, w_n \in \mathbb{B}^n$. For every assignment α and term t appearing in $\mathcal{R}$ we introduce the atoms $\mathrm{int}_{\alpha,t}$ and $\mathrm{lab}_{\alpha,t}$ for $t \not\in \mathcal{V}$. The formula $\mathrm{int}_{\alpha,t}$ encodes $[\alpha]_{\mathcal{B}}(t)$, where $[\alpha]_{\mathcal{B}}$ denotes the evaluation function of the Boolean model $\mathcal{B}$, while $\mathrm{lab}_{\alpha,t}$ expressed the label of the root symbol of t under with respect to the assignment α. We define $\mathrm{INT}_\alpha(t) = \mathrm{int}_{\alpha,t} \leftrightarrow \ulcorner f_{\mathcal{B}} \urcorner(\mathrm{int}_{\alpha,t_1}, \ldots, \mathrm{int}_{\alpha,t_n})$ and $\mathrm{LAB}_\alpha(t) = \mathrm{lab}_{\alpha,t} \leftrightarrow \ulcorner \ell_f \urcorner(\mathrm{int}_{\alpha,t_1}, \ldots, \mathrm{int}_{\alpha,t_n})$. Further for $t \in \mathcal{V}$ we define $\mathrm{INT}_\alpha(t) = \alpha(t)$. These constraints have to be enforced for every term appearing in $\mathcal{R}$. We write $\mathcal{R} \trianglerighteq t$ to denote that t is a subterm of a left- or right-hand side of a rule in $\mathcal{R}$. To formalise the the model condition, we employ:

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

Assume ν is a satisfying assignment for $\mathrm{LAB}(\mathcal{R})$ and $\mathcal{R}_{\mathsf{lab}}$ denotes the system obtained by labelling $\mathcal{R}$ according to the encoded labelling 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 labelled versions of two concrete terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ under an assignment α, we define

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

Here $\ulcorner s >^{(i)}_{\mathsf{pop*}} t \urcorner$ refers to the encodings of the case $\langle i \rangle$ from the definition of $\succ_{\mathsf{pop*}}$. We discuss the cases $\langle 2 \rangle$ – $\langle 4 \rangle$, as case $\langle 1 \rangle$ is obtained similar.

Set $\ulcorner f(s_1,\dots,s_n) >^{(2)}_{\mathsf{pop*}} t \urcorner_\alpha = \top$ if $s_i = t$ holds for some s_i Otherwise, $\ulcorner f(s_1,\dots,s_n) >^{(2)}_{\mathsf{pop*}} t \urcorner_\alpha = \bigvee_{i=1}^{n} \ulcorner s_i \succ_{\mathsf{pop*}} t \urcorner_\alpha$. For any $f \in \mathcal{F}_{\mathsf{lab}}$ and argument position i of f we encode $i \in \mathsf{safe}(f)$ by a atom $\mathsf{safe}_{f,i}$. For $f \in \mathcal{F}$ and formula a, the formula $\mathrm{SF}(f_a, i)$ assesses that depending on the valuation of a, either the i-th position of f_{true} or f_{false} is safe. Similarly we define $\mathrm{NRM}(f_a, i)$ to assert that the i-th argument position of the labelled symbol is normal. For formulas a and b, $f, g \in \mathcal{F}$, and ν an assignment, the formula $\ulcorner f_a > g_b \urcorner$ represents the comparison $f_{\nu(a)} > g_{\nu(b)}$. Assume $f \in \mathcal{D}$, We translate case $\langle 3 \rangle$ for $f \neq g$ to

$$\ulcorner f(s_1,\dots,s_n) >^{(3)}_{\mathsf{pop*}} g(t_1,\dots,t_m) \urcorner_\alpha = \ulcorner f_{\mathrm{lab}_{\alpha,s}} > g_{\mathrm{lab}_{\alpha,t}} \urcorner \wedge$$
$$\wedge \bigvee_{i_0} \Big(\ulcorner s \succ_{\mathsf{pop*}} t_{i_0} \urcorner_\alpha \wedge \mathrm{SF}(g_{\mathrm{lab}_{\alpha,t}}, i_0) \wedge \bigvee_{i \neq i_0} \big(\ulcorner s >^{(1)}_{\mathsf{pop*}} t_i \urcorner_\alpha \vee (\mathrm{SF}(g_{\mathrm{lab}_{\alpha,t}}, i) \wedge \ulcorner s \rhd t_i \urcorner)\big)\Big) \ .$$

In order to guarantee that f_a is defined we add a rule $f_a(x_1,\dots,x_n) \to \mathsf{c}$ to the labelled TRS, where c is a fresh constant. In practical test this has shown to be more effective than allowing that f_a becomes undefined. To encode multiset comparisons, we make use of *multiset covers*, c.f. [16]. A multiset cover is a pair of total mappings $\gamma\colon \{1,\dots,m\} \to \{1,\dots,n\}$ and $\varepsilon\colon \{1,\dots,n\} \to \mathbb{B}$, encoded using fresh atoms $\gamma_{i,j}$ and ε_i. To assert a correct encoding of (γ, ε), we introduce the formula $\ulcorner (\gamma, \varepsilon) \urcorner$. Based upon this, we define $\ulcorner f(s_1,\dots,s_n) >^{(4)}_{\mathsf{pop*}} f(t_1,\dots,t_n) \urcorner_\alpha$ by

$$(\mathrm{lab}_{\alpha,s} \leftrightarrow \mathrm{lab}_{\alpha,t}) \wedge \bigvee_{i=1}^{n} (\mathrm{NRM}(f_{\mathrm{lab}_{\alpha,s}}, i) \wedge \neg\varepsilon_i) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \big(\gamma_{i,j} \to (\varepsilon_i \to \ulcorner s_i = t_j \urcorner) \wedge$$
$$\wedge\, (\neg\varepsilon_i \to \ulcorner s_i \succ_{\mathsf{pop*}} t_j \urcorner_\alpha) \wedge (\mathrm{SF}(f_{\mathrm{lab}_{\alpha,s}}, i) \leftrightarrow \mathrm{SF}(f_{\mathrm{lab}_{\alpha,t}}, j))\big) \wedge \ulcorner (\gamma, \varepsilon) \urcorner \ .$$

Finally satisfiability of

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

asserts the existence of a model $\mathcal{B}$ and labelling ℓ such that the labelled (and extended) TRS $\mathcal{R}'_{\mathsf{lab}}$ is compatible with $\succ_{\mathsf{pop*}}$. Here $\mathrm{STRICT}(\mathcal{R})$ formalises the strictness and other minor properties of the precedence $>$ while $\mathrm{SM}_{\mathsf{SL}}(\mathcal{R})$ guarantees that safe is indeed a safe mapping.

Experiments

`clean up and shorten`

`test suites beschreiben/motivieren`

Besides the two methods described above in Sect. **??**, we have also implemented the *light multiset path order* (LMPO) [3] and Quasi-Interpretations [9]. LMPOis, like POP*, a tamed version of the multiset path order. It is slightly less restrictive than POP*, but additionally requires the evaluation strategy to cache the results of all intermediate computations in order to induce a polynomial upper bound on the required time (see Example **??**). Our implementation of LMPOis rather similar to the implementation of POP*: like in the latter case, we encode the requirements of the two orders into propositional logic. The important differences between LMPOand POP*lie in the last two cases in Def. **??**. Furthermore, the relation $\sim$ has to be changed: instead of the multiset comparison in POP*, only a one-to-one comparison of the arguments is allowed for LMPO. Furthermore, for constructors, the subterms may not be permuted for this comparison.

For Quasi-Interpretations, we have implemented additive RPO^{QI}_{Pro}. This method succeeds if the given rewrite system is compatible with both an additive quasi-interpretation and an RPO where each function symbol has a *product* status (which compares tuples of terms by $s_1, \ldots, s_n \succ^p t_1, \ldots, t_n$ iff $s_i \succeq t_i$ for all i and $s_i \succ t_i$ for at least on i). Additive quasi-interpretations use the following interpretation functions:

$$
\begin{aligned}
f_{\mathcal{A}}(x_1, \ldots, x_n) &= \sum_{i_j \in \{0,1\}} a_{f,i_1,\ldots,i_n} x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} + \sum_{i=1}^{n} b_i x_i^2 && \text{if } f \in \mathcal{D} \\
f_{\mathcal{A}}(x_1, \ldots, x_n) &= \sum_{i=1}^{n} x_i + c + 1 && \text{if } f \in \mathcal{C}, \mathsf{ar}\, f > 0 \\
f_{\mathcal{A}}(x_1, \ldots, x_n) &= 0 && \text{if } f \in \mathcal{C}, \mathsf{ar}\, f = 0
\end{aligned}
$$

Unlike other polynomial interpretations, this only has to orient all rewrite rules weakly. We search for a suitable interpretation by constructing constraints in propositional logic in the same way as for SMC: first, we build a suitable set of Diophantine constraints according to the algorithm given in [12]. These are then encoded into SAT, following the ideas presented in [13]. RPO with product status is implemented in a similar way as POP*and SMC. Note that, due to our approach of building Diophantine constraints, we can only handle quasi-interpretations which use standard polynomials, but not the max function.

We have tested the approaches on two testbeds: first, we have used all constructor based systems from the current version of the termination problems database [6] (we denote this testbed by **C**). Second, we have specifically tested all constructor systems from the folders `AG01`, `D33`, and `SK90` of the TPDB (we denote this testbed by **F**). The tests were run single-threaded on a 2.1 GHz Intel Core 2 Duo with 1 GB of memory. The results of the tests are shown in Table 1[7].

As we can see, the most powerful methods for showing computability of functions in polynomial time are POP$^*_{\mathrm{SL}}$and SMC. However, since POP$^*_{\mathrm{SL}}$is a

[6] `http://www.lri.fr/~marche/tpdb/`.

[7] Full experimental evidence can be found at `http://homepage.uibk.ac.at/~csae2496/ijcar/index.html`.

Table 1. Experimental results

		POP^*	$POP^*_\sim$	POP^*_{SL}	LMPO	$LMPO^*$	RPO^{QI}_{Pro}	SMC	$MPO_\sim$
F	Yes	7	8	14	13	13	13	15	17
	Maybe	64	63	57	58	58	58	49	54
	Timeout	0	0	0	0	0	0	7	0
	Avg. Time (ms)	32	34	122	32	32	191	773	37
C	Yes	41	43	74	54	55	51	69	69
	Maybe	551	549	511	549	537	541	415	523
	Timeout	0	0	7	0	0	0	108	0
	Avg. Time (ms)	31	36	105	37	32	204	350	40

primarily syntactic termination criterion which only uses semantic labelling as a semantic component, while SMC is a fully semantic method. This is reflected by the huge difference between these two methods in the used time and the number of timeouts. $MPO_\sim$in this table denotes the standard multiset path order with quasi-precedence, as implemented in $\mathsf{T_TT_2}$. We can also see that $POP^*$and LMPOare a notable restriction of $MPO_\sim$, but can still solve a good portion of the examples solved by $MPO_\sim$. Furthermore, a comparison between $MPO_\sim$and RPO^{QI}_{Pro}shows that the quasi interpretations can also solve a lot of the examples that are shown terminating by $MPO_\sim$. Considering that RPO with product status is a further restriction of $MPO_\sim$, this shows that the quasi-interpretations are currently certainly not a bottleneck in our implementation of RPO^{QI}_{Pro}, even though our implementation is currently not using the max function yet.

References

1. Marion, J.Y., Moyen, J.Y.: Efficient first order functional program interpreter with time bound certifications. In: Proc. 7th LPAR. Volume 1955. (2000) 25–42
2. Bonfante, G., Cichon, A., Marion, J.Y., Touzet, H.: Algorithms with polynomial interpretation termination proof. JFP **11**(1) (2001) 33–53
3. Marion, J.: Analysing the implicit complexity of programs. IC **183** (2003) 2–18
4. Marion, J.Y., , Péchoux, R.: Resource analysis by sup-interpretation. In: FLOPS. Volume 3945. (2006) 163–176
5. Marion, J.Y., , Péchoux, R.: Quasi-friendly sup-interpretations. CoRR **abs/cs/0608020** (2006)
6. Avanzini, M., Moser, G.: Complexity analysis by rewriting. In: Proc. 9th FLOPS. (2008) To appear. Available at `http://cl-informatik.uibk.ac.at/~georg/list.publications.html`.
7. Bonfante, G., Marion, J.Y., Péchoux, R.: Quasi-interpretation synthesis by decomposition. In: Proc. 4th ICTAC. Volume 4711 of LNCS. (2007) 410–424
8. Steinbach, J., Kühler, U.: Check your ordering - termination proofs and open problems. Technical Report SEKI-Report SR-90-25, University of Kaiserslautern (1990)

9. Bonfante, G., Marion, J.Y., Moyen, J.Y.: Quasi-intepretations and small space bounds. In: Proc. 16th RTA. Number 3467 in LNCS (2005) 150–164
10. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
11. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Proc. 3rd RTA. Number 355 in LNCS (1989) 167–177
12. Contejean, E., Marché, C., Tomás, A.P., Urbain, X.: Mechanically proving termination using polynomial interpretations. JAR **34**(4) (2005) 325–363
13. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: Proc. 10th SAT. Volume 4501 of LNCS. (2007) 340–354
14. Zantema, H.: Termination of term rewriting by semantic labelling. FI **24** (1995) 89–105
15. Avanzini, M., Moser, G.: Complexity analysis by rewriting. Draft. Availabe at `http://cl-informatik.uibk.ac.at/~georg/list.publications.html` (2007)
16. Schneider-Kamp, P., Thiemann, R., Annov, E., Codish, M., Giesl, J.: Proving termination using recursive path orders and SAT solving. In: Proc. 6th FroCos. Number 4720 in LNCS (2007) 267–282
17. Zankl, H., Middeldorp, A.: Satisfying KBO constraints. In: Proc. of 18th RTA. Number 4533 in LNCS (2007) 389–403