Automated Implicit Computational Complexity Analysis (System Description)*

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 (fully automatic and command-line based) system that implements the majority of these techniques and present experimental findings to simplify comparisons.

1 Introduction

Recent studies have provided many characterisations of the class of polynomial time computable functions. Our main interest lies in studies that employ term rewriting as abstract model of computation and consequently use existing techniques from rewriting to characterise several computational complexity classes, cf. [1,2,3,4,5,6]. The use of rewriting techniques opens the way for automatisation. In this paper, we present a fully automatic complexity tool that implements a majority of the techniques introduced in the cited literature. We describe the system and its implementation in some detail, and report on experimental results comparing the relative strength of the implemented techniques.

For brevity and greater applicability we consider techniques that directly classify runtime complexity, i.e., those techniques use the number of rewrite steps (perhaps allowing for a specific rewrite strategy) as complexity measure, namely *additive polynomial interpretations* [2] and *polynomial path orders* [6]. In Section 2, we recall the main theorems for these proof methods, and describe their implementation in some detail. Apart from that, we have also implemented the *light multiset path order* [3] (*LMPO* for short) and *quasi-interpretations* in conjunction with the *product path order* [1,7,8] (RPO^{QI}_{Pro} for short) in our tool. Opposed to the former methods, the latter two rely on non-trivial evaluation techniques, namely caching of intermediate results.

In order to compare the different methods proposed in the literature, we tested their applicability on (subsets of) the Termination Problem Data Base (TPDB for short), which is used in the annual termination competition. Arguably this is an imperfect choice as the TPDB has been designed to test the

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

strength of termination provers in rewriting, not as a testbed to analyse the implicit computational 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 interesting and meaningful subsets of this collection.

It seems natural that *additive polynomial interpretations* and *polynomial path orders*, which only use rewriting as an underlying model of computation, perform worse in comparison to *quasi-interpretations* and the *light multiset path order*. Interestingly, our practical findings cannot confirm this: on our studied testbeds, the former techniques outperform the latter two if we count the total number of TRSs that can be handled. Furthermore, a closer look on the testbeds reveals that every single one of the implemented techniques can handle at least one example that cannot be handled by any other method (see Section 4 for more details). This implies that a notable part of our testbeds is covered with the union of the implemented methods.

2 Methods that Directly Classify Polytime

In this section, we describe *additive polynomial interpretations* and *polynomial path orders* in more detail. To keep this system description short, we assume familiarity with rewriting (see for example [9]). Our first task is to fix what is meant by the function *computed* by a TRS. We write $s \rightarrow^!_{\mathcal{R}} t$ for $s \rightarrow^*_{\mathcal{R}} t$, whenever t is a normal form with respect to a TRS $\mathcal{R}$. Suppose $\ulcorner\cdot\urcorner$ is an encoding function and let $\mathcal{R}$ denote a completely defined and orthogonal constructor TRS. Let Σ be an alphabet. An n-ary function $f\colon (\Sigma^*)^n \to \Sigma^*$ is *computable* by $\mathcal{R}$ if there exists a defined function symbol f such that for all $w_1, \dots, w_n, v \in \Sigma^*$: $\mathsf{f}(\ulcorner w_1\urcorner, \dots, \ulcorner w_n\urcorner) \rightarrow^! \ulcorner v\urcorner$ 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. We say that a TRS $\mathcal{R}$ has a *simple signature* (cf. [3]) if we can type all constructors of $\mathcal{R}$ and define a mapping S from the set of sorts to $\mathbb{N}$ such that for every constructor of sort $s_1, \dots, s_n \to s$, there exists an i with $S(s_i) \leqslant S(s)$, and $S(s_j) < S(s)$ for all $j \neq i$. The *(innermost) runtime complexity function* with respect to a TRS $\mathcal{R}$ relates the length of the longest (innermost) derivation sequence in $\mathcal{R}$ to the size of the arguments of the initial term of the form $f(t_1, \dots, t_n)$, where f is a defined function symbol, and t_i are ground terms consisting only of constructor symbols.

A suitable starting point for the classification of polytime computable functions is to look at *polynomial interpretations*. However, as shown in [10], polynomial interpretations without further restrictions still admit a double exponential upper bound on the runtime complexity of a given TRS. Hence restricted polynomial interpretations are studied in the literature. A polynomial $P(x_1, \dots, x_n)$ (over $\mathbb{N}$) is called *additive* (cf. [2]) if $P(x_1, \dots, x_n) = x_1 + \cdots + x_n + a$ where $a \in \mathbb{N}$. A polynomial interpretation $\mathcal{A}$ is called *simple-mixed, constructor-restricted* (*SMC* for short) if all interpretation functions $f_{\mathcal{A}}$ are additive polynomials whenever f is a constructor symbol, and *simple-mixed polynomials* (cf. [11]) otherwise. The following theorem is shown in [2, Lemma 3, Theorem 4].

Theorem 1. *Let $\mathcal{R}$ be a finite constructor TRS that is compatible with an SMC-interpretation. Then the runtime complexity with respect to $\mathcal{R}$ is polynomial. Furthermore, the functions computable by a finite and orthogonal constructor TRS that is compatible with an SMC-interpretation are polytime computable.*

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 $\sim$. We require that the arity of f equals the arity of g whenever $f \sim g$. POP* relies on the separation of *safe* and *normal* inputs. A safe mapping safe associates with every n-ary function symbol f the set of *safe argument positions.* If f is a defined function symbol then $\mathsf{safe}(f) \subseteq \{1,\ldots,n\}$, otherwise 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. 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. We define $f(s_1,\ldots,s_n) \sim_{\mathsf{pop*}} g(t_1,\ldots,t_n)$ (respectively $f(s_1,\ldots,s_n) \sim_{\mathsf{pop}} g(t_1,\ldots,t_n)$) if and only if $f \sim g$, and the safe and normal part of the multisets $[s_1,\ldots,s_n]$ and $[t_1,\ldots,t_n]$ are equivalent over $\sim_{\mathsf{pop*}}$ (respectively $\sim_{\mathsf{pop}}$). We define the order $\succ_{\mathsf{pop}}$ inductively as follows: $s = f(s_1,\ldots,s_n) \succ_{\mathsf{pop}} t$ if either

1. $f \in \mathcal{C}$ 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 $f \in \mathcal{D}$, $f \succ g$, and $s \succ_{\mathsf{pop}} t_i$ for all $1 \leqslant i \leqslant m$.

Here $\mathcal{D}$ ($\mathcal{C}$) denotes the defined (constructor) symbols, respectively. Based on $\succ_{\mathsf{pop}}$ we define $\succ_{\mathsf{pop*}}$ 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 $f \in \mathcal{D}$, $f \succ g$, and the following properties hold: (i) $s \succ_{\mathsf{pop*}} t_{i_0}$ for some $i_0 \in \mathsf{safe}(g)$ and (ii) 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 $\mathsf{nrm}(f) = \{i_1,\ldots,i_p\}$, $\mathsf{safe}(f) = \{j_1,\ldots,j_q\}$), $\mathsf{nrm}(g) = \{i'_1,\ldots,i'_{p'}\}$, $\mathsf{safe}(g) = \{j'_1,\ldots,j'_{q'}\}$, both $[s_{i_1},\ldots,s_{i_p}]$ $(\succ_{\mathsf{pop*}})_{\mathsf{mul}}$ $[t_{i'_1},\ldots,t_{i'_{p'}}]$ and $[s_{j_1},\ldots,s_{j_q}]$ $(\succsim_{\mathsf{pop*}})_{\mathsf{mul}}$ $[t_{j'_1},\ldots,t_{j'_{q'}}]$ hold.

Here $(\succ)_{\mathsf{mul}}$ denotes the multiset extension of an order $\succ$. We arrive at the following theorem, which is an easy consequence of the main results from [6].

Theorem 2. *Let $\mathcal{R}$ be a finite constructor TRS compatible with $\succ_{\mathsf{pop*}}$, i.e., $\mathcal{R} \subseteq \succ_{\mathsf{pop*}}$. Then the induced innermost runtime complexity is polynomial. Furthermore, the functions computable by a finite and orthogonal constructor TRS with a simple signature that is compatible with an instance of POP* are exactly the polytime computable functions.*

In order to increase the applicability of POP*, we employ semantic labeling [12] with Boolean models. It is straightforward to check that the maximal lengths of derivations in the original and the labeled system are equal. In order to apply Theorem 2 it is mandatory to restrict to *finite* models. We discuss the implementation of POP* with semantic labeling below in Section 3.

3 Implementation

The here presented fully automatic system `icct`[3] implements the techniques introduced in [1,2,3,6,7]. In addition to the techniques described in Section 2, our system `icct` can handle the methods LMPO [3] and RPO^{QI}_{Pro} [1]. We have not (yet) considered (quasi-friendly) sup-interpretations, cf. [4,5], but plan to do so in the future. Furthermore our system `icct` only searches for additive polynomial quasi-interpretations, without the max-function. This contrasts with the system `crocus`[4] implementing quasi-interpretations as defined in [8]. We also plan to add this in the future, possibly using the methods described in [13].

Our implementation is (partly) based on the termination prover $\mathsf{T_TT_2}$[5] and therefore our (command-line) interface is compatible with $\mathsf{T_TT_2}$. The modular design of our implementation allows for an immediate integration of the presented system as a plug-in to $\mathsf{T_TT_2}$. Like $\mathsf{T_TT_2}$, our methods are written fully in `OCaml`, and the system `icct` extends $\mathsf{T_TT_2}$ by around 3500 lines of code. In the remainder of this section, we describe the implementation of SMC and POP*.

In order to search for SMC interpretations, we follow well-established methods stemming from termination analysis. Our starting point is the use of abstract polynomial interpretations, cf. [14]. For SMC, that is: $f_{\mathcal{A}}(x_1,\dots,x_n) = \sum_{i_j \in \{0,1\}} a_{i_1,\dots,i_n} x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} + \sum_{i=1}^{n} b_i x_i^2$ if $f \in \mathcal{D}$, and $f_{\mathcal{A}}(x_1,\dots,x_n) = \sum_{i=1}^{n} x_i + c$ otherwise. The variables $a_{i_1,\dots,i_n}$, b_i, 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. Putting an upper bound on the coefficient variables makes the problem finite, allowing it to be transformed into SAT. The encoding into SAT essentially follows [15]. To actually solve the satisfiability problem, MiniSAT[6] is invoked. A satisfying assignment is used to instantiate the coefficient variables suitably.

To prove compatibility of a given TRS $\mathcal{R}$ with polynomial path orders we have to find a quasi-precedence $\succsim$ and a *safe mapping* such that the induced order is compatible with $\mathcal{R}$. Due to the huge search space, this is difficult to implement efficiently. Thus, we translate this problem into SAT. Here, we focus on the encoding of POP* for strict precedence with semantic labeling for models over the carrier $\mathbb{B} = \{0,1\}$. To encode a Boolean function $b : \mathbb{B}^n \to \mathbb{B}$, we make use of propositional atoms b_w for every sequence of arguments $w = w_1,\dots,w_n \in \mathbb{B}^n$. Let the formula $\ulcorner b \urcorner(a_1,\dots,a_n)$ be such that for a satisfying assignment ν, the equality $\nu(\ulcorner b \urcorner(a_1,\dots,a_n)) = b_{\nu(a_1),\dots,\nu(a_n)}$ is asserted. For every assignment α and term

[3] Available online at `http://cl-informatik.uibk.ac.at/software/icct`, where full evidence of the experiments presented below can be found.

[4] `http://libresource.inria.fr/projects/crocus`.

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

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

t appearing in $\mathcal{R}$ we introduce the atoms $\mathrm{int}_{\alpha,t}$ and $\mathrm{lab}_{\alpha,t}$ for $t \notin \mathcal{V}$. 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 $\mathrm{lab}_{\alpha,t}$ expresses the label of the root symbol of t 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 $\mathrm{INT}_{\alpha}(t) = \mathrm{int}_{\alpha,t} \leftrightarrow \alpha(t)$. 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 $\mathcal{R}$. The model condition is formalised by

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

Assume ν is a satisfying assignment for $\mathrm{LAB}(\mathcal{R})$ and $\mathcal{R}_{\mathsf{lab}}$ denotes the system obtained by labeling $\mathcal{R}$. In order to show compatibility of $\mathcal{R}_{\mathsf{lab}}$ with POP^*, we need to find a precedence $>$ and a safe mapping such that $\mathcal{R}_{\mathsf{lab}} \subseteq >_{\mathsf{pop}*}$ holds. To compare the labelled versions of two terms s, t under 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 encoding 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 similarly.

Set $\ulcorner f(s_1, \ldots, s_n) >^{(2)}_{\mathsf{pop}*} t \urcorner_{\alpha} = \top$ if $s_i = t$ holds for some s_i. Otherwise, $\ulcorner f(s_1, \ldots, s_n) >^{(2)}_{\mathsf{pop}*} t \urcorner_{\alpha} = \bigvee_{i=1}^{n} \ulcorner s_i \succ_{\mathsf{pop}*} t \urcorner_{\alpha}$. For any $f \in \mathcal{F}_{\mathsf{lab}}$ and an argument position i of f we encode $i \in \mathsf{safe}(f)$ by an atom $\mathsf{safe}_{f,i}$. For $f \in \mathcal{F}$ and a formula a denoting the label of f, the formula $\mathrm{SF}(f_a, i)$ ($\mathrm{NRM}(f_a, i)$) assesses that the i-th position of f_a is safe (normal), respectively. Furthermore, for $f, g \in \mathcal{F}$ and formulae a and b, 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 $\langle 3 \rangle$ as follows:

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

In order to guarantee that f_a is defined we add a rule $f_a(x_1, \ldots, x_n) \to \mathsf{c}$ to the labelled TRS, where c is a fresh constant. In practical tests this has shown to be more effective than allowing that f_a becomes undefined. A careful analysis revealed that in order to encode the restricted multiset comparison, the notion of *multiset covers* [16] is adaptable, cf. [6]. A multiset cover is a pair of total mappings $\gamma \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$ and $\varepsilon \colon \{1, \ldots, n\} \to \mathbb{B}$, encoded using fresh atoms $\gamma_{i,j}$ and ε_i. To assert a correct encoding of (γ, ε), we introduce the formula $\ulcorner (\gamma, \varepsilon) \urcorner$. We define $\ulcorner f(s_1, \ldots, s_n) >^{(4)}_{\mathsf{pop}*} f(t_1, \ldots, t_n) \urcorner_{\alpha}$ by

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

Finally $\mathrm{POP}^*_{\mathrm{SL}}(\mathcal{R}) = \bigwedge_\alpha \bigwedge_{l \to r \in \mathcal{R}} \ulcorner l \succ_{\mathsf{pop}*} r \urcorner_\alpha \wedge \mathrm{SM}(\mathcal{R}) \wedge \mathrm{STRICT}(\mathcal{R}) \wedge \mathrm{LAB}(\mathcal{R})$ asserts the existence of a model $\mathcal{B}$ and labeling ℓ such that the labelled TRS is compatible with $\succ_{\mathsf{pop}*}$. Here $\mathrm{STRICT}(\mathcal{R})$ formalises the strictness of $>$, while $\mathrm{SM}(\mathcal{R})$ enforces a valid encoding of the safe mapping safe.

4 Experiments and Conclusions

We have tested our complexity tool `icct` on two testbeds: first, we filtered the TPDB for constructor TRS, this testbed is denoted as **C** below. Secondly we considered TRS from the example collections [17,18,19] and filtered for orthogonal constructor (hence confluent) TRS, we call this testbed **FP**.[7] The first test suite (containing 592 systems) aims to compare the relative power of the methods on a decently sized testbed. The second one (containing 71 systems) tries to focus on natural examples of confluent TRS. Success on the testbed **FP** implies that the function encoded by the given TRS is polytime computable — however, for LMPO and POP^* one has to verify in addition that the signature is simple. The tests were run on a standard PC with 512 MB of memory and a 2.4 GHz Intel® Pentium™ IV processor. We summarise the results of the test in Table 1.

Counting the total number of yes-instances reveals that the most powerful methods for showing computability of functions in polynomial time are SMC and $\mathrm{POP}^*_{\mathrm{SL}}$. As $\mathrm{POP}^*_{\mathrm{SL}}$is a mostly syntactic method, it is much faster than the fully semantic SMC. Note that SMC is slightly more powerful than $\mathrm{POP}^*_{\mathrm{SL}}$ on testbed **FP**, while the order is reversed for test suite **C**. This indicates that the techniques can handle different TRS, which can be verified by a close look at the full experimental evidence[3]. More precisely, any of the methods implemented in our system `icct` can handle (at least) one example that cannot be handled by any other method. In particular SMC is the only technique that can handle the example `AG01/#3.7`, an encoding of the logarithm function also studied in [8]. In addition we compared `icct` with the results for `crocus` published at the `crocus` site[4], but found that `crocus` in conjunction with PPO [7] cannot handle more TRS than listed in the respective column.

Table 1. Experimental results

	POP^*		$\mathrm{POP}^*_{\sim}$		$\mathrm{POP}^*_{\mathrm{SL}}$		LMPO		RPO^{QI}_{Pro}		SMC	
	FP	**C**	**FP**	**C**	**FP**	**C**	**FP**	**C**	**FP**	**C**	**FP**	**C**
Yes	7	41	8	43	14	74	13	54	13	51	15	69
Maybe	64	551	63	549	57	511	58	538	58	540	49	394
Timeout	0	0	0	0	0	7	0	0	0	1	7	129
Avg. Time (ms)	53		47		192		44		279		507	

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

As already mentioned, possible future work would be to extend `icct` by implementing sup-interpretations [4,5] and adding the max-function [13] to quasi-interpretations and SMC. In addition we want to investigate whether recent work on context-dependent interpretations [20] can be extended to implicit computational complexity analysis.

References

1. Marion, J.Y., Moyen, J.Y.: Efficient first order functional program interpreter with time bound certifications. In: Proc. 7th LPAR. Volume 1955 of LNCS. (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.Y.: 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 of LNCS. (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. Volume 4989 of LNCS. (2008) 130–146
7. Bonfante, G., Marion, J.Y., Moyen, J.Y.: Quasi-intepretations and small space bounds. In: Proc. 16th RTA. Volume 3467 of LNCS. (2005) 150–164
8. Bonfante, G., Marion, J.Y., Péchoux, R.: Quasi-interpretation synthesis by decomposition. In: Proc. 4th ICTAC. Volume 4711 of LNCS. (2007) 410–424
9. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
10. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Proc. 3rd RTA. Volume 355 of LNCS. (1989) 167–177
11. Steinbach, J.: Generating polynomial orderings. IPL **49** (1994) 85–93
12. Zantema, H.: Termination of term rewriting by semantic labelling. FI **24** (1995) 89–105
13. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: Maximal termination. In: Proc. 19th RTA. (2008) To appear.
14. Contejean, E., Marché, C., Tomás, A.P., Urbain, X.: Mechanically proving termination using polynomial interpretations. JAR **34**(4) (2005) 325–363
15. 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
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. Volume 4720 of LNCS. (2007) 267–282
17. Steinbach, J., Kühler, U.: Check your ordering - termination proofs and open problems. Technical Report SR-90-25, University of Kaiserslautern (1990)
18. Dershowitz, N.: 33 examples of termination. In: Term Rewriting, French Spring School of Theoretical Computer Science, Advanced Course, Springer (1995) 16–26
19. Arts, T., Giesl, J.: A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-09-2001, RWTH Aachen (2001)
20. Moser, G., Schnabl, A.: Proving quadratic derivational complexities using context dependent interpretations. In: Proc. 19th RTA. (2008) To appear.