On a Correspondence between Predicative Recursion and Register Machines*

Martin Avanzini[1], Naohi Eguchi[2], and Georg Moser[1]

1 **Institute of Computer Science,**
University of Innsbruck, Austria
{martin.avanzini,georg.moser}@uibk.ac.at

2 **Mathematical Institute,**
Tohoku University, Japan
eguchi@math.tohoku.ac.jp

Abstract

We present the small polynomial path order sPOP*. Based on sPOP*, we study a class of rewrite systems, dubbed systems of predicative recursion of degree d, such that for rewrite systems in this class we obtain that the runtime complexity lies in $O(n^d)$. We show that predicative recursive rewrite systems of degree d define functions computable on a register machine in time $O(n^d)$.

1998 ACM Subject Classification F.2.2, F.4.1, F.4.2, D.2.4, D.2.8

Keywords and phrases Runtime Complexity, Polynomial Time Functions, Implicit Computational Complexity, Rewriting

1 Introduction

In [1] we propose the *small polynomial path order* (*sPOP** for short). The order sPOP* provides a characterisation of the class of *polynomial time computable function* (*polytime computable functions* for short) via term rewrite systems. Any polytime computable function gives rise to a rewrite system that is compatible with sPOP*. On the other hand any function defined by a rewrite system compatible with sPOP* is polytime computable. The proposed order embodies the principle of *predicative recursion* as proposed by Bellantoni and Cook [4]. Our result bridges the subject of (automated) complexity analysis of rewrite systems and the field of implicit computational complexity (*ICC* for short).

Based on sPOP*, one can delineate a class of rewrite systems, dubbed systems of *predicative recursion of degree d*, such that for rewrite systems in this class we obtain that the runtime complexity lies in $O(n^d)$. This is a tight characterisation in the sense that one can provide a family of systems of predicative recursion of depth d, such that their runtime complexity is bounded from below by $\Omega(n^d)$ [1]. In this note, we study the connection between functions f defined by predicative recursive term rewrite systems (TRSs) of degree d and register machines. We show that any such function can be computed by a register machine operating in time $O(n^d)$. This result further emphasises the fact that the runtime complexity of a TRS (cf. [7]) is an invariant cost model [3]. Our work was essentially motivated by Leivant's work on *predicative recurrence* [8] and Marion's *strict ramified primitive recursion* [10].

* This work is partially supported by FWF (Austrian Science Fund) project I-608-N18 and by a grant of the University of Innsbruck.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let $\mathcal{R}$ be a TRS and fix a (quasi)-precedence $\succcurlyeq := \succ \uplus \sim$ on the symbols of $\mathcal{R}$. We are assuming that the arguments of every function symbol are partitioned in to normal and safe ones. Notationally we write $f(t_1, \dots, t_k \,;\, t_{k+1}, \dots, t_{k+l})$ with *normal* arguments $t_1, \dots, t_k$ separated from *safe* arguments $t_{k+1}, \dots, t_{k+l}$ by a semicolon. We define the equivalence $\sim_{\mathsf{s}}$ on terms respecting this separation as follows: $s \sim_{\mathsf{s}} t$ holds if $s = t$ or $s = f(s_1, \dots, s_k \,;\, s_{k+1}, \dots, s_{k+l})$ and $t = g(t_1, \dots, t_k \,;\, t_{k+1}, \dots, t_{k+l})$ where $f \sim g$ and $s_i \sim_{\mathsf{s}} t_{\pi(i)}$ for all $i = 1, \dots, k+l$ such that the permutation π on $\{1, \dots, k+l\}$ maps normal to normal argument positions. We write $s \rhd_{\mathsf{n}} t$ if t is a proper subterm of s (modulo $\sim_{\mathsf{s}}$) at a normal argument position: $f(s_1, \dots, s_k \,;\, s_{k+1}, \dots, s_{k+l}) \rhd_{\mathsf{n}} t$ if $s_i \trianglerighteq \cdot \sim_{\mathsf{s}} t$ and $i \in \{1, \dots, k\}$.

The following definition introduces small polynomial path orders $>_{\mathsf{spop}*}$. The order allows recursive definitions only on *recursive symbols* $\mathcal{D}_{\mathsf{rec}} \subseteq \mathcal{D}$. Symbols in $\mathcal{D} \setminus \mathcal{D}_{\mathsf{rec}}$ are called *compositional* and denoted by $\mathcal{D}_{\mathsf{comp}}$. To retain the separation under $\sim_{\mathsf{s}}$, we require $\sim \subseteq \mathcal{C}^2 \cup \mathcal{D}^2_{\mathsf{rec}} \cup \mathcal{D}^2_{\mathsf{comp}}$. We set $\geqslant_{\mathsf{spop}*} := \sim_{\mathsf{s}} \cup >_{\mathsf{spop}*}$ and also write $>_{\mathsf{spop}*}$ for the product extension of $>_{\mathsf{spop}*}$ to tuples $\vec{s} = \langle s_1, \dots, s_n \rangle$ and $\vec{t} = \langle t_1, \dots, t_n \rangle$: $\vec{s} \geqslant_{\mathsf{spop}*} \vec{t}$ holds if $s_i \geqslant_{\mathsf{spop}*} t_i$ for all $i = 1, \dots, n$ and $\vec{s} >_{\mathsf{spop}*} \vec{t}$ holds if additionally $s_{i_0} >_{\mathsf{spop}*} t_{i_0}$ for some $i_0 \in \{1, \dots, n\}$. We denote by $\mathcal{T}(\mathcal{F}^{\prec f}, \mathcal{V})$ the set of terms build from variables and function symbols $\mathcal{F}^{\prec f} := \{g \mid f \succ g\}$.

▶ **Definition 1.1.** Let $s = f(s_1, \dots, s_k \,;\, s_{k+1}, \dots, s_{k+l})$. Then $s >_{\mathsf{spop}*} t$ if either

1) $s_i \geqslant_{\mathsf{spop}*} t$ for some argument s_i of s.
2) $f \in \mathcal{D}$, $t = g(t_1, \dots, t_m \,;\, t_{m+1}, \dots, t_{m+n})$ with $f \succ g$ and the following conditions hold: (i) $s \rhd_{\mathsf{n}} t_j$ for all normal arguments t_j of t, (ii) $s >_{\mathsf{spop}*} t_j$ for all safe arguments t_j of t, and (iii) $t_j \notin \mathcal{T}(\mathcal{F}^{\prec f}, \mathcal{V})$ for at most one $j \in \{1, \dots, k+l\}$.
3) $f \in \mathcal{D}_{\mathsf{rec}}$, $t = g(t_1, \dots, t_k \,;\, t_{k+1}, \dots, t_{k+l})$ with $f \sim g$ and the following conditions hold: (i) $\langle s_1, \dots, s_k \rangle >_{\mathsf{spop}*} \langle t_{\pi(1)}, \dots, t_{\pi(k)} \rangle$ for some permutation π, (ii) $\langle s_{k+1}, \dots, s_{k+l} \rangle \geqslant_{\mathsf{spop}*} \langle t_{\tau(k+1)}, \dots, t_{\tau(k+l)} \rangle$ for some permutation τ.

The *depth of recursion* $\mathsf{rd}(f)$ is inductively defined in correspondence to the rank of f in $\succcurlyeq$, but only takes recursive symbols into account: Let $n = \max\{0\} \cup \{\mathsf{rd}(g) \mid f \succ g\}$. Then $\mathsf{rd}(f) := 1 + n$ if $f \in \mathcal{D}_{\mathsf{rec}}$ and otherwise $\mathsf{rd}(f) := n$. We say a constructor TRS $\mathcal{R}$ is *predicative recursive of degree* d if $\mathcal{R}$ is compatible with an instance $>_{\mathsf{spop}*}$ and the maximal depth of recursion of a function symbol in $\mathcal{R}$ is d.

▶ **Theorem 1.2** ([1]). *Let $\mathcal{R}$ be predicative recursive of degree d. Then the innermost runtime complexity of $\mathcal{R}$ lies in $O(n^d)$. Moreover, this bound is tight.*

As one anonymous reviewer points out, Theorem 1.2 also holds with respect to full rewriting, if $\mathcal{R}$ is in addition a non-duplicating overlay system [6].

2 Register Machines Compute Predicative TRSs

Let $\mathbb{W}$ denote the set of *words* over a *binary* alphabet. Fix a predicative constructor TRS $\mathcal{R}$ of degree d that *computes* functions over $\mathbb{W}$. We will now show that the functions computed by $\mathcal{R}$ can be realised on a *register machine* (*RM*) [5], operating in time asymptotic to n^d where n is the size of the input.

First we make precise the notion of computation on TRSs. We assume that the encoding of words $\mathbb{W}$ as terms makes use of dyadic successors s_0 and s_1 that append the corresponding character to its argument, as well as the constant ϵ to construct the empty word. Henceforth we set $\mathcal{C} := \{\mathsf{s}_0, \mathsf{s}_1, \epsilon\}$ and by the one-to-one correspondence between ground constructor terms and binary words $\mathbb{W}$ we allow ourselves to confuse these sets. Let f be a defined symbol

in $\mathcal{R}$ of arity k. Then $\mathcal{R}$ computes the function $f : \mathbb{W}^k \to \mathbb{W}$ defined as $f(w_1, \dots, w_k) = w$ if $\mathsf{f}(w_1, \dots, w_k) \to^!_{\mathcal{R}} w$. This notion is well-defined if $\mathcal{R}$ is orthogonal (hence confluent) and completely defined, i.e., normal forms and constructor terms coincide.

A RM over $\mathbb{W}$ contains a finite set of registers $\mathsf{R} = \{x_1, \dots, x_n\}$ that store words over $\mathbb{W}$. We use the notion of RM from [5] adapted from $\mathbb{N}$ to binary words $\mathbb{W}$ and identify RMs with goto-programs over variables R that allow to (i) copy (the content of) one variable to another, (ii) appending $0, 1$, or removing the last bit of a variable, and (iii) that can perform conditional branches based on the last bit of a variable. A RM *computes* the function $f : \mathbb{W}^k \to \mathbb{W}$ with $k \leqslant n$ defined as follows: $f(w_1, \dots, w_k) = w$ if on initial assignment w_i to x_i for all $i = 1, \dots, k$ and ε to x_i for all $i = k+1, \dots, n$, the associated goto-program halts and the content of a dedicated output-variable x_o equals w. The complexity of an RM is given by the number of executed instructions as function in the sum of sizes of the input.

To simplify matters, we *normalise* right-hand sides of rewrite rules. Throughout the following, we denote by $\vec{u}, \vec{v}, \vec{w}$, possibly extended by subscripts, vectors of constructor terms. Let $\mathcal{R}_\mathsf{n}$ denote some fixpoint on $\mathcal{R}$ of following *normalisation* operator: if the TRS contains a rule $\mathsf{f}(\vec{u_f}\,;\vec{v_f}) \to \mathsf{g}(\vec{u_g}\,;\vec{t_1}, \mathsf{h}(\vec{u_h}\,;\vec{t_2}), \vec{t_3})$ where $\mathsf{f} \succ \mathsf{h}$, $\mathsf{h} \in \mathcal{D}$ and $\vec{t_1}, \vec{t_2}$ or $\vec{t_3}$ contain at least one defined symbol, replace the rule with $\mathsf{f}(\vec{u_f}\,;\vec{v_f}) \to \mathsf{g}'(\vec{u_f}\,;\vec{v_f}, \vec{t_1}, \vec{t_2}, \vec{t_3})$ and $\mathsf{g}'(\vec{u_f}\,;\vec{v_f}, \vec{x_1}, \vec{x_2}, \vec{x_3}) \to \mathsf{g}(\vec{u_g}\,;\vec{x_1}, \mathsf{h}(\vec{u_h}\,;\vec{x_2}), \vec{x_3})$. Here g' is a fresh composition symbol so that $\mathsf{f} \succ \mathsf{g}' \succ \mathsf{g}, \mathsf{h}$ and variables $\vec{x_1}, \vec{x_2}, \vec{x_3}$ do not occur elsewhere. Note that $\mathcal{R}_\mathsf{n}$ is well-defined as in each step the number of defined symbols in right-hand sides are decreasing.

▶ **Lemma 2.1.** *We have (i) $\to_{\mathcal{R}} \subseteq \to^+_{\mathcal{R}_n}$ and (ii) $\mathcal{R}_n$ is predicative recursive of degree d.*

By Property (i) it is easy to verify that any function computed by $\mathcal{R}$ is also computed by $\mathcal{R}_\mathsf{n}$. Property (ii) and the definition of $\mathcal{R}_\mathsf{n}$ allows the classification of each $\mathsf{f}(\vec{u_l}\,;\vec{v_l}) \to r \in \mathcal{R}_\mathsf{n}$ into one of the following forms.

- *Construction Rule*: r is a constructor term;
- *Recursion Rule*: $r = \mathsf{g}(\vec{u_g}\,;\vec{v_g}, \mathsf{f}'(\vec{u_r}\,;\vec{v_r}), \vec{w_g})$ where $\mathsf{f} \succ \mathsf{g}$ and $\mathsf{f} \sim \mathsf{f}'$;
- *Composition Rule*: $r = \mathsf{g}(\vec{u_g}\,;\vec{v_g}, \mathsf{h}(\vec{u_r}\,;\vec{v_r}), \vec{w_g})$ where $\mathsf{f} \succ \mathsf{g}, \mathsf{h}$.

In the latter two cases the context $\mathsf{g}(\vec{u_g}\,;\vec{v_g}, \Box, \vec{w_g})$ might also be missing. Note that for recursion rules, the sum of sizes of $\vec{u_l}$ is strictly greater than the sum of the sizes of $\vec{u_r}$.

▶ **Theorem 2.2.** *Let $\mathcal{R}$ be an orthogonal and completely defined predicative system of degree d. Every function f computed by $\mathcal{R}$ is computed by a register machine RM_f operating in time $\mathsf{O}(n^d)$, where n refers to the sum of the sizes of* normal *arguments.*

Proof. Consider a function f computed by $\mathcal{R}$, and let f be the corresponding defined symbol. We define a program P_f which, on *input variables* $\vec{I_f}$ initialised with $\vec{v}$, computes $f(\vec{v})$ in a dedicated *output variable* O_f, executing no more than $\mathsf{O}(n^{\mathsf{rd(f)}})$ instructions. The program P_f works by reduction according to the *normalised* TRS $\mathcal{R}_\mathsf{n}$. For this note that $\mathcal{R}_\mathsf{n}$ is orthogonal, hence Lemma 2.1 (1) gives that $\mathcal{R}_\mathsf{n}$ reduces $\mathsf{f}(\vec{v})$ to $f(\vec{v})$ independent on the evaluation strategy. The construction is by induction on the rank f in $\succcurlyeq$ (on the extended signature of $\mathcal{R}_\mathsf{n}$). We only consider the more involved inductive step. By induction hypothesis for each g below f in the precedence there exist a program P_g that compute the function defined by g operating in time $\mathsf{O}(n^{\mathsf{rd(g)}})$, where n is the sum of sizes of normal arguments to g.

Suppose the input variables $\vec{I_f}$ hold the arguments $\vec{v}$. Due to linearity, pattern matches can be hard-coded by looking at suffixes in $\vec{I_f}$ bounded in size by a constant. Consequently in a constant number of steps P_f can check which rules applies on $\mathsf{f}(\vec{v})$. First suppose $\mathsf{f} \in \mathcal{D}_{\mathsf{comp}}$, thus $\mathsf{f}(\vec{u})$ reduces either using a construction or composition rule.

The interesting case is when $\mathsf{f}(\vec{u}) \xrightarrow{\mathsf{i}}_{\mathcal{R}_\mathsf{n}} \mathsf{g}(\vec{v_1}, \mathsf{h}(\vec{w}), \vec{v_2})$ due to a composition rule. Since $\mathsf{f} \succ \mathsf{g}, \mathsf{h}$, induction hypothesis gives programs P_g and P_h that compute the functions defined

by g and h respectively. The program P_f first stores the arguments to h in the dedicated input registers $\vec{I_h}$ and executes the code of P_h. Since $\vec{w}$ are constructor terms, initialisation of $\vec{I_h}$ requires only constant time similar to above. Further the sum of sizes of normal inputs in $\vec{u}$ and $\vec{w}$ differ only by a constant factor c_1, hence executing P_h takes time $\mathsf{O}((c_1 \cdot n)^{\mathsf{rd}(\mathsf{h})}) = \mathsf{O}(n^{\mathsf{rd}(\mathsf{h})})$. We repeat the procedure using a program P_g in time $\mathsf{O}(n^{\mathsf{rd}(\mathsf{g})})$. Here we employ that due to separation of safe and normal arguments, the complexity of computing the call of g does not depend on the result of $\mathsf{h}(\vec{w})$. Overall, employing $\mathsf{rd}(\mathsf{f}) \geqslant \mathsf{rd}(\mathsf{g}), \mathsf{rd}(\mathsf{h})$, the runtime is in $\mathsf{O}(n^{\mathsf{rd}(\mathsf{h})} + n^{\mathsf{rd}(\mathsf{g})}) \subseteq \mathsf{O}(n^{\mathsf{rd}(\mathsf{f})})$.

Now suppose $\mathsf{f} \in \mathcal{D}_\mathsf{rec}$ and thus $\mathsf{rd}(\mathsf{f}) \geqslant 1$. Consider an innermost reduction of f. Wlog

$$\mathsf{f}(\vec{v}) = \mathsf{f}_0(\vec{v_0}) \xrightarrow{\mathsf{i}}_{\mathcal{R}_\mathsf{n}} \mathsf{g}_1(\vec{u_1}, \mathsf{f}_1(\vec{v_1}), \vec{w_1}) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{g}_1(\vec{u_1}, \mathsf{g}_2(\vec{u_2}, \dots, \mathsf{g}_k(\vec{u_k}, \mathsf{f}_k(\vec{v_k}), \vec{w_k}), \dots, \vec{w_2}), \vec{w_1})$$

where the first k applications follow from applying recursive rules, and $\mathsf{f}_k(\vec{v_k})$ matches either a construction or composition rule. By definition the sum of sizes of normal arguments in the *recursion arguments* $\vec{v_0}, \dots, \vec{v_k}$ is strictly decreasing, and conclusively k is bounded by n. To compute $f(\vec{v})$, the program P_f evaluates the last term inside out, starting from $\mathsf{f}_k(\vec{v_k})$. Since we have only a constant number of registers at our disposal, we cannot program the machine to memorise or recompute all recursion arguments $\vec{v_0}, \dots, \vec{v_k}$ in time linear in n. Instead, we employ per argument position of f an additional register and exploit the following one-to-one correspondence between arguments $\vec{v_{i+1}}$ and $\vec{v_i}$: $\vec{v_{i+1}}$ is obtained from $\vec{v_i}$ by flipping and chopping a constant number of bits according to the rewrite rule applied in step i. For $i = 0, \dots, n$, the machine performs this operation on the input registers storing $\vec{v_i}$, pushing the chopped bits onto the corresponding auxiliary registers in constant time. To recall the rule applied in step i, we associate each rule with a binary number of fixed size, and push this number on an additional register that we abuse as a *call stack*. Since $\vec{v_{i+1}}$ is obtained from $\vec{v_i}$ by executing a constant number of instructions, $\vec{v_k}$ is constructed in time $k = \mathsf{O}(n)$, allowing stepwise reconstruction of recursion arguments starting from $\vec{v_k}$.

Recall that the sum of sizes of normal recursion arguments $\vec{v_i}$ $(i = 1, \dots, k)$ is decreasing and consequently bounded by n. Consider the evaluation of $\mathsf{f}_k(\vec{v_k})$ that reduces by construction either using a composition or projection rule. In both cases we conclude that $\mathsf{f}_k(\vec{v_k})$ is computed in time $\mathsf{O}(n^{\mathsf{rd}(\mathsf{f})-1})$ as in the case $\mathsf{f} \in \mathcal{D}_\mathsf{comp}$, employing $\mathsf{rd}(\mathsf{f}) > \mathsf{rd}(\mathsf{g})$ for all g such that $\mathsf{f} \succ \mathsf{g}$. The evaluation is then continued inside out exactly as in the case $\mathsf{f} \in \mathcal{D}_\mathsf{comp}$, recovering the arguments $\vec{v_i}$ from $\vec{v_{i+1}}$ after each step in constant time. Employing $\mathsf{rd}(\mathsf{f}) > \mathsf{rd}(\mathsf{g_i})$ we see that the application of $\mathsf{g_i}$ is bounded by $\mathsf{O}(n^{\mathsf{rd}(\mathsf{g})}) \subseteq \mathsf{O}(n^{\mathsf{rd}(\mathsf{f})-1})$. Overall, employing $k = \mathsf{O}(n)$, the procedure stops after executing at most $\mathsf{O}(n) + \mathsf{O}(n \cdot n^{\mathsf{rd}(\mathsf{f})-1}) = \mathsf{O}(n^{\mathsf{rd}(\mathsf{f})})$ instructions. This concludes the final case. ◀

3 Experimental Results

We have implemented sPOP* in the *Tyrolean Complexity Tool* T$_\mathsf{C}$T[1]. In Table 1 we contrast sPOP* to its predecessors *lightweight multiset path orders* (LMPO for short) [9] and *polynomial path orders* [2] (POP* for short)[2]. LMPO characterises the class of polytime computable functions, also by embodying the principle of predicative recursion. Since LMPO allows simultaneous recursion it fails at binding the runtime complexity polynomially. POP* characterises predicative recursive systems but cannot give a precise bound on the runtime

[1] T$_\mathsf{C}$T is open source and available from `http://cl-informatik.uibk.ac.at/software/tct`.
[2] See `http://cl-informatik.uibk.ac.at/software/tct/experiments/wst2012` for full experimental evidence and explanation on the setup.

complexity. Finally we also included *multiset path orders* (MPO for short) in the table, as all mentioned orders are essentially syntactic restrictions of MPO.

Comparing LMPO and MPO, the experiments reveal that enforcing predicative recursion limits the power of our techniques by roughly one fourth on our testbed. Comparing POP* with sPOP* we see an increase in precision accompanied with only a minor decrease in power. Of the four systems that can be handled by POP* but not by sPOP*, two fail to be oriented because sPOP* weakens the multiset status to product status, and two fail because sPOP* enforces a more restrictive composition scheme.

bound	MPO	LMPO	POP*	sPOP*
$\mathsf{O}(1)$				$9_{\backslash 0.06}$
$\mathsf{O}(n^1)$				$32_{\backslash 0.07}$
$\mathsf{O}(n^2)$				$38_{\backslash 0.09}$
$\mathsf{O}(n^3)$				$39_{\backslash 0.20}$
$\mathsf{O}(n^k)$			$43_{\backslash 0.05}$	$39_{\backslash 0.20}$
yes	$76_{\backslash 0.09}$	$57_{\backslash 0.05}$	$43_{\backslash 0.05}$	$39_{\backslash 0.07}$
maybe	$681_{\backslash 0.16}$	$700_{\backslash 0.11}$	$714_{\backslash 0.11}$	$718_{\backslash 0.11}$

Figure 1 Number of oriented problems and average execution time in seconds.

4 Conclusion and Future Work

We have shown that predicative TRSs of recursion depth d can be computed by RMs operating in time $O(n^d)$. One question that remains open is the reverse direction on the correspondence between RMs and predicative TRSs. Using a pairing constructor for collecting the contents of the registers, the simulation of $\mathsf{O}(n^d)$ time-bounded RMs is straight forward to define using recursion up to depth d. Without such a constructor however, the proof gets significantly more involved. Still, we are sufficiently convinced of our argument to conjecture that also the reverse direction on the correspondence between RMs and predicative TRSs holds. More precisely, suppose f is computable by a RM in time $\mathsf{O}(n^d)$. Then there exists a predicative recursive TRS $\mathcal{R}$ of degree d that computes f. In future work we also want to investigate whether we can weaken the assumptions in Theorem 2.2 so that compatibility with sPOP* is no longer required.

References

1 M. Avanzini, N. Eguchi, and G. Moser. A New Order-theoretic Characterisation of the Polytime Computable Functions. *CoRR*, cs/CC/1201.2553, 2012.

2 M. Avanzini and G. Moser. Complexity Analysis by Rewriting. In *Proc. of 9th FLOPS*, volume 4989 of *LNCS*, pages 130–146, 2008.

3 M. Avanzini and G. Moser. Closing the Gap Between Runtime Complexity and Polytime Computability. In *Proc. of 21st RTA*, volume 6 of *LIPIcs*, pages 33–48, 2010.

4 S. Bellantoni and S. Cook. A new Recursion-Theoretic Characterization of the Polytime Functions. *CC*, 2(2):97–110, 1992.

5 K. Erk and L. Priese. *Theoretische Informatik: Eine umfassende Einführung*. Springer Verlag, 3te auflage edition, 2008.

6 N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for Termination and Complexity. *JAR*, 2012. To appear.

7 N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. *CoRR*, abs/1102.3129, 2011. submitted.

8 D. Leivant. Stratified Functional Programs and Computational Complexity. In *Proc. 20th POPL*, pages 325–333, 1993.

9 J.-Y. Marion. Analysing the Implicit Complexity of Programs. *IC*, 183:2–18, 2003.

10 J.-Y. Marion. On Tiered Small Jump Operators. *LMCS*, 5(1), 2009.