Complexity Analysis by Rewriting*

Martin Avanzini[1] and Georg Moser[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@uibk.ac.at

Abstract. In this paper we introduce a restrictive version of the multiset path order, called *polynomial path order*. This recursive path order induces polynomial bounds on the maximal number of innermost rewrite steps. This result opens the way to automatically verify for a given program, written in an eager functional programming language, that the maximal number of evaluation steps starting from any function call is polynomial in the input size. To test the feasibility of our approach we have implemented this technique and compare its applicability to existing methods.

1 Introduction

Term rewriting is a conceptually simple but powerful abstract model of computation that underlies much of declarative programming. In rewriting, proving termination is an important research field. Powerful methods have been introduced to establish termination of a given term rewrite system. One of the most natural ways to proof termination is the use of *interpretations*. Consequentially this technique has been introduced quite early. Moreover, if one is interested in automatically proving termination, *polynomial interpretations* provide a natural starting point, cf. [10]. However, termination proofs via polynomial interpretations are limited as the longest possible rewrite sequences admitted by rewrite systems compatible with a polynomial interpretation are double-exponential (in the size of the initial term), see [13]. Another well-studied (and direct) termination technique is the use of reduction orders—for example simplification orders. Still this technique is limited, which can again be shown by the analysis of the induced derivation length, cf. [12,25,15]. In recent years the emphasis shifted towards transformation techniques like the dependency pair method or semantic labeling. Transformation techniques have significantly increased the possibility to automatically prove termination.

Once we have established termination of a given rewrite system $\mathcal{R}$, it seems natural to direct the attention to the analysis of the *complexity* of $\mathcal{R}$. In rewriting the complexity of a rewrite system $\mathcal{R}$ is measured as the *maximal derivation length* with respect to $\mathcal{R}$. As mentioned above for *direct* termination methods

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

a significant amount of investigations has been conducted, providing a suitable foundation for further research. Unfortunately, almost nothing is known about the length of derivations induced by state-of-the-art termination techniques like the dependency pair method or semantic labeling. For the dependency pair method no results on the induced derivation length are known. Partial result with respect to semantic labeling are reported in [18].

In this paper we introduce a restriction of the multiset path order, called *polynomial path order* (denoted as $>_{\mathsf{pop}*}$). Our main result states that this recursive path order induces polynomial bounds on the maximal length of *innermost* rewrite steps. As we have successfully implemented this technique, we thus can *automatically* verify for a given term rewrite system $\mathcal{R}$ that $\mathcal{R}$ admits at most polynomial innermost derivation length (on the set of *constructor-based* terms). This opens the way to automatically verify for a given program—written in an eager functional programming language—that its runtime complexity is polynomial (in the input size). The only restrictions in the applicability of the result are that (i) the functional program P is transformable into a term rewrite system $\mathcal{R}$ and (ii) a feasible (i.e., polynomial) derivation length with respect to $\mathcal{R}$ gives rise to a feasible runtime complexity of P. In short the transformation has to be *non-termination* and *complexity preserving*.

The definition of polynomial path orders employs the idea of *tiered recursion* [6]. Syntactically this amount to a separation of arguments into *normal* and *safe* argument. (Below this will be governed by the presences of mappings safe and nrm associating with each function symbol a list of argument positions.) We explain our approach by an example rewrite system that clearly admits at most polynomial derivation length.

Example 1. Consider the following rewrite system $\mathcal{R}_{\mathsf{mult}}$.

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

We suppose that all arguments of the successor (s) are safe ($\mathsf{safe}(\mathsf{s}) = \{1\}$), that the second argument of addition (add) is safe ($\mathsf{safe}(\mathsf{add}) = \{2\}$) and that all arguments of multiplication (mult) are normal ($\mathsf{safe}(\mathsf{mult}) = \varnothing$). Furthermore let the (strict) precedence $>$ be defined as $\mathsf{mult} > \mathsf{add} > \mathsf{s}$. Then $\mathcal{R}_{\mathsf{mult}}$ is compatible with $>_{\mathsf{pop}*}$ (see Definition 4) and as a consequence of our main theorem (see Section 3) we conclude that the number of rewrite steps starting from $\mathsf{mult}(\mathsf{s}^n(\mathsf{0}),\mathsf{s}^m(\mathsf{0}))$ is polynomially bounded in n and m. (Here we write $\mathsf{s}^n(\mathsf{0})$ as abbreviation of $\mathsf{s}(\ldots(\mathsf{s}(\mathsf{0})\ldots))$ with n occurrences of the successor symbol s.)

The polynomial path order is an extension of the *path order for* **FP** introduced by Arai and the second author in [1]. A central motivation of this research is the observation that the direct application of the latter order is only successful on a handful of (very simple) rewrite systems. The path order for **FP** gains only power if additional transformations are performed. Unfortunately, such powerful transformations are difficult to find automatically.

Further note that the polynomial path order is to some extent related to the *light multiset path order* introduced by Marion [17]. Roughly speaking the light

multiset path order is a tamed version of the multiset path order, characterising the functions computable in polytime. It seems important to stress that the below stated main theorem fails for the light multiset path order. This can be easily seen from the next example.

Example 2. Consider the following rewrite system $\mathcal{R}_{\mathsf{bin}}$. (This is Example 2.21 about binomial coefficients from [22].)

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

For a precedence that fulfills $\mathsf{bin} > \mathsf{s}$, $\mathsf{bin} > +$ and separations of arguments $\mathsf{safe}(\mathsf{bin}) = \varnothing$, $\mathsf{safe}(+) = \{1,2\}$, we obtain that $\mathcal{R}_{\mathsf{bin}}$ is compatible with the light multiset path order, cf. [17]. However it is straightforward to verify that the (innermost) derivation height of $\mathsf{bin}(\mathsf{s}^n(\mathsf{0}),\mathsf{s}^m(\mathsf{0}))$ is exponential in n.

To test the feasibility of our approach we have implemented a small complexity analyser based on the polynomial path order and compare its applicability to existing techniques. To do so, we also have implemented the light multiset path order and a restricted form of polynomial interpretations, so-called *additive polynomial interpretations*, cf. [7]. Note that compatibility with addivite polynomial interpretations induces polynomial derivation length for constructor-based terms, cf. [7].

The research in [7,17] falls into the realm of *implicit complexity theory*. In this context related work to our research is due to Bonfante *et al.* [8] but see also seminal work by Hofmann [14] and Schwichtenberg [20]. While [14,20] are incomparable to our techniques, a comparison to [8] is also not straightforward. Our principal concern is that the *termination* techniques employed allow for an *complexity analysis* of the subjected program. On the other hand the crucial feature of *quasi-interpretations* (the central contribution of [8]) is their weak monotonicity, hence termination can only be shown in conjunction with other termination techniques. For example the class of polytime computable functions can be characterised as the class of functions computable by confluent constructor rewrite systems compatible with the multiset path order *and* that admit only additive quasi-interpretations, cf. [8]. This interesting result renders an insightful implicit characterisation of the polytime computable function, but it is of little help, if one wants to obtain a complexity analysis of a term rewrite system subjected to a modern termination prover. Recently an interesting application of quasi-interpretations has been reported by Lucas and Peña [16]. Here the dependency pair method is used in conjunction with quasi-interpretations to obtain bounds on the *memory consumption* of Safe programs. This method is easily automatable, but new ideas are necessary to yield bounds on the *runtime behaviour* of functional programs.

The remainder of this paper is organised as follows. In the next section we recall basic notions and starting points of this paper. In Section 3 we have collected our main results. In order to prove these results we extend results originally presented in [1]. Our findings in this direction are presented in Section 4. The central

argument to prove the main theorem is then given in Section 5. In Section 6 we give the experimental evidence mentioned above. In Section 7 we touch upon an application of our main theorem in recent work (together with Hirokawa and Middeldorp) where we study the termination behaviour of Scheme programs. Finally in Section 8 we conclude and mention possible future work.

2 Preliminaries

We assume familiarity with term rewriting [4,23]. Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$. We always assume that $\mathcal{F}$ contains at least one constant. The *arity* of a function symbol f is denoted as $\mathsf{ar}(f)$. Let $>$ be a precedence on the signature $\mathcal{F}$. The *rank* of a function symbol is defined inductively as follows: $\mathsf{rk}(f) = 1+\max\{\mathsf{rk}(g) \mid g \in \mathcal{F} \wedge f > g\}$. (Here we employ the convention that the maximum of an empty set equals 0.) We write $\trianglelefteq$ to denote the subterm relation and $\trianglerighteq$ for its converse. The strict part of $\trianglerighteq$ is denoted by $\rhd$. $\mathsf{Var}(t)$ denotes the set of variables occurring in a term t. The *size* (*depth*) of a term t is denoted as $\mathsf{size}(t)$ ($\mathsf{dp}(t)$). The width of a term t is defined inductively as follows: $\mathsf{wd}(t) = 1$, if t is a variable or a constant, otherwise if $t = f(t_1,\ldots,t_n)$ with $n > 0$, we set $\mathsf{wd}(t) = \max\{n, \mathsf{wd}(t_1),\ldots,\mathsf{wd}(t_n)\}$. The *Buchholz norm* of a term t is defined inductively as follows: $\|t\| = 1$, if t is a variable and for $t = f(t_1,\ldots,t_n)$ we set $\|t\| = 1 + \max\{n, \|t_1\|,\ldots,\|t_n\|\}$. We write $[t_1,\ldots,t_n]$ to denote multisets and $\uplus$ for the summation of multisets.

A *term rewrite system* (*TRS* for short) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a set of rewrite rules $l \to r$, such that $l \notin \mathcal{V}$ and $\mathsf{Var}(l) \supseteq \mathsf{Var}(r)$. (If not mentioned otherwise, we assume $\mathcal{R}$ is *finite*.) The root symbols of left-hand sides of rewrite rules are called *defined*, while all other function symbols are called *constructors*. For a given signature $\mathcal{F}$ the defined symbols are denoted as $\mathcal{D}$, while the constructor symbol are collected in $\mathcal{C}$. The smallest rewrite relation that contains $\mathcal{R}$ is denoted by $\to_{\mathcal{R}}$. We simply write $\to$ for $\to_{\mathcal{R}}$ if $\mathcal{R}$ is clear from context. Let s and t be terms. If exactly n steps are preformed to contract s to t we write $s \to^n t$. A term $s \in \mathcal{T}(\mathcal{F},\mathcal{V})$ is called a *normal form* if there is no $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ such that $s \to t$. The *innermost rewrite relation* $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ of a TRS $\mathcal{R}$ is defined on terms as follows: $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t$ if there exist a rewrite rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s = C[l\sigma]$, $t = C[r\sigma]$, and all proper subterms of $l\sigma$ are normal forms of $\mathcal{R}$. A TRS is called *confluent* if for all $s,t_1,t_2 \in \mathcal{T}(\mathcal{F},\mathcal{V})$ with $s \to^* t_1$ and $s \to^* t_2$ there exists a term t_3 such that $t_1 \to^* t_3$ and $t_2 \to^* t_3$. A TRS is *non-overlapping* if it has no critical pairs, cf. [4]. A TRS $\mathcal{R}$ is *left-linear* if for all rules $l \to r \in \mathcal{R}$, all variables in l occur at most once. If $\mathcal{R}$ is additionally non-overlapping, then $\mathcal{R}$ is called *orthogonal*. Note that every orthogonal TRS is confluent. A *constructor* TRS is a TRS whose signature $\mathcal{F}$ can be partitioned into the defined symbols $\mathcal{D}$ and constructor symbols $\mathcal{C}$ in such a way that the left-hand side of each rule has the form $f(s_1,\ldots,s_n)$ with $f \in \mathcal{D}$ and for all i: $s_i \in \mathcal{T}(\mathcal{C},\mathcal{V})$. A defined function symbol is *completely defined* if it does not occur in any ground term in normal form. A TRS is *completely defined* if each

defined symbol is completely defined. An element of $\mathcal{T}(\mathcal{C},\mathcal{V})$ is called a *value*; we set $\mathsf{Val}(\mathcal{R}) = \mathcal{T}(\mathcal{C},\mathcal{V})$. We call a TRS *terminating* if no infinite rewrite sequence exists. The *derivation length* of a term t with respect to a terminating TRS $\mathcal{R}$ and rewrite relation $\to_{\mathcal{R}}$ is defined as usual: $\mathsf{DI}_{(\mathcal{R},\to)}(s) = \max\{n \mid \exists t\ s \to^n t\}$. We call a term $t = f(t_1,\dots,t_n)$ *constructor-based* if all its arguments t_i are values, i.e., $t_i \in \mathcal{T}(\mathcal{C},\mathcal{V})$ for all $1 \leqslant i \leqslant n$. The set $\mathcal{T}_{\mathrm{b}}$ collects all constructor-based terms. The *runtime complexity* (with respect to $\mathcal{R}$) is defined as follows: $\mathsf{Rc}_{\mathcal{R}}(m) = \max\{\mathsf{DI}_{(\mathcal{R},\xrightarrow{\mathsf{i}})}(t) \mid t = f(t_1,\dots,t_n) \in \mathcal{T}_{\mathrm{b}} \text{ and } \sum_i^n \mathsf{size}(t_i) \leqslant m\}$.

A proper order is a transitive and irreflexive relation. The *reflexive closure* of a proper order $\succ$ is denoted as $\succ^=$. A proper order $\succ$ is *well-founded* if there is no infinite decreasing sequence $t_1 \succ t_2 \succ t_3 \cdots$. A well-founded proper order that is also a rewrite relation is called a *reduction order*. We say a reduction order $\succ$ and a TRS $\mathcal{R}$ are *compatible* if $\mathcal{R} \subseteq \succ$. It is well-known that a TRS is terminating if and only if there exists a compatible reduction order.

3 Main Result

In the sequel $\mathcal{R}$ denotes a constructor TRS over a (possible variadic) signature $\mathcal{F}$. Let $>$ denote a precedence on $\mathcal{F}$ such that for all $f \in \mathcal{D}$ we have for all $c \in \mathcal{C}$: $f > c$. (Recall that $\mathcal{F}$ contains at least one constant.) We assume that $\mathcal{R}$ is completely defined, i.e., ground normal forms and ground values coincide.[3]

For each n-ary function symbol $f \in \mathcal{D}$ of fixed arity, we suppose the existence of a mapping safe that associates with f a (possibly empty) list $\{i_1,\dots,i_m\}$ with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. For a mapping safe and a term $t = f(t_1,\dots,t_n)$, $\mathsf{safe}(f)$ denotes the *safe argument positions* of t. The argument positions of t *not* included in $\mathsf{safe}(f)$ are called *normal* and are denoted by $\mathsf{nrm}(f)$. The mapping safe (nrm) is referred to as *safe* (*normal*) mapping. We generalise safe (normal) mappings to constructor symbols and variadic function symbols as follows: For each function symbol $f \in \mathcal{C}$, we fix $\mathsf{safe}(f) = \{1,\dots,\mathsf{ar}(f)\}$ and for each variadic function symbol $f \in \mathcal{D}$ we assert $\mathsf{safe}(f) = \varnothing$. The *normalised signature* $\mathcal{F}^{\mathsf{n}}$ contains a function symbol f^{n} for each $f \in \mathcal{F}$. If f is of fixed-arity and $\mathsf{nrm}(f) = \{i_1,\dots,i_p\}$, then $\mathsf{ar}(f) = p$. The normalised signature $\mathcal{C}^{\mathsf{n}}$ is defined accordingly.

Definition 3. *Let $>$ be a precedence and safe a safe mapping. We define $>_{\mathsf{pop}}$ inductively as follows: $s = f(s_1,\dots,s_n) >_{\mathsf{pop}} t$ if one of the following alternatives holds:*

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

[3] The assumption that $\mathcal{R}$ is completely defined arises naturally in the context of implicit characterisation of complexity classes. We follow this convention to some extent, but show that this restriction is not necessary.

We write $s >_{\mathsf{pop}} t\ \langle i \rangle$ if $s >_{\mathsf{pop}} t$ follows by application of clause (i) in Definition 3. A similar notation will be used for the orders defined below.

Definition 4. *Let $>$ be a precedence and* safe *a safe mapping. We define the* polynomial path order $>_{\mathsf{pop}*}$ *(*POP* *for short) inductively as follows: $s = f(s_1, \ldots, s_n) >_{\mathsf{pop}*} t$ if one of the following alternatives holds:*

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

Here $(>_{\mathsf{pop}*})_{\mathsf{mul}}$ denotes the multiset extension of $>_{\mathsf{pop}*}$ and recall that for variadic function symbols, the set of safe arguments is empty.

Example 5. Consider the following TRS $\mathcal{R}_{\mathsf{insert}}$ (This is a simplification of an example from [17].)

$$\begin{aligned}
\mathsf{if}(\mathsf{true}, x, y) &\to x & x \geqslant 0 &\to \mathsf{true}\\
\mathsf{if}(\mathsf{false}, x, y) &\to y & 0 \geqslant \mathsf{s}(x) &\to \mathsf{false}\\
\mathsf{ins}(x, \mathsf{nil}) &\to \mathsf{cons}(x, \mathsf{nil}) & \mathsf{s}(x) \geqslant \mathsf{s}(y) &\to x \geqslant y\\
\mathsf{ins}(x, \mathsf{cons}(y, ys)) &\to \mathsf{if}(y \geqslant x, \mathsf{cons}(x, \mathsf{cons}(y, ys)), \mathsf{cons}(y, \mathsf{ins}(x, ys)))
\end{aligned}$$

We represent lists with the help of the constructors nil and cons. To show compatibility with POP*, we assume a precedence $\succ$ that fulfills $\mathsf{ins} \succ \mathsf{if}$, $\mathsf{ins} \succ \geqslant$, $\mathsf{ins} \succ \mathsf{cons}$, $0 \succ \mathsf{true}$, and $0 \succ \mathsf{false}$. Further we define a safe mapping safe as follows:

$$\begin{aligned}
\mathsf{safe}(\mathsf{s}) &= \{1\} & \mathsf{safe}(\mathsf{if}) &= \{1, 2, 3\} & \mathsf{safe}(\mathsf{ins}) &= \varnothing\\
\mathsf{safe}(\mathsf{cons}) &= \{1, 2\} & \mathsf{safe}(\geqslant) &= \{2\}
\end{aligned}$$

It is straightforward to verify that the induced polynomial path order $\succ_{\mathsf{pop}*}$ is compatible with $\mathcal{R}_{\mathsf{insert}}$.

An easy inductive argument shows that if $s \in \mathsf{Val}(\mathcal{R})$ and $s >_{\mathsf{pop}*} t$, then $t \in \mathsf{Val}(\mathcal{R})$. Note that $>_{\mathsf{pop}*}$ is *not* a reduction order. Although $>_{\mathsf{pop}*}$ is a well-founded proper order that is closed under substitutions, the order is not closed under contexts due to the restrictive definition of clause 4 in the above definition. However we still have the following theorem, which follows as the multiset path order extends $>_{\mathsf{pop}*}$.

Theorem 6. *Every TRS $\mathcal{R}$ that is compatible with $>_{\mathsf{pop}*}$ for some well-founded precedence $>$ is terminating.*

As *normal* and *safe* arguments are distinguisable, we strengthen the notion of runtime complexity as follows: $\mathsf{Rc}^{\mathsf{n}}_{\mathcal{R}}(m) = \max\{\mathsf{DI}_{(\mathcal{R},\xrightarrow{\mathsf{i}})}(t) \mid t = f(t_1,\dots,t_n) \in \mathcal{T}_{\mathrm{b}} \text{ and } \sum_{i\in\mathsf{nrm}(f)} \mathsf{size}(t_i) \leqslant m\}$. This function is called the *normal* runtime complexity.

Main Theorem. *Let $\mathcal{R}$ be a finite, completely defined constructor TRS. Assume further $\mathcal{R}$ is compatible with $>_{\mathsf{pop}^*}$, i.e., $\mathcal{R} \subseteq >_{\mathsf{pop}^*}$. Then the induced (normal) runtime complexity is polynomial.*

Assume $\mathcal{R}$ is a finite, constructor TRS that is not completely defined; i.e., at least one defined function symbol occurs in a ground normal form. To obtain a completely defined TRS it suffices to add suitable rules, thus we arrive at the following corollary, see [3] for the proof.

Corollary 7. *Let $\mathcal{R}$ be a finite, constructor TRS. Assume further $\mathcal{R}$ is compatible with $>_{\mathsf{pop}^*}$, i.e., $\mathcal{R} \subseteq >_{\mathsf{pop}^*}$. Then the induced (normal) runtime complexity is polynomial.*

Definition 8. *The* predicative *rewrite relation $s \xrightarrow{\mathsf{p}} t$ is defined as follows: $s \xrightarrow{\mathsf{p}} t$ if $s \to t$ by contracting safe argument positions first, i.e., if there exist a rewrite rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s = C[l\sigma]$, $t = C[r\sigma]$ and all safe argument position of $l\sigma$ are in normal form.*

Clearly predicative rewriting is a generalisation of innermost rewriting. Essentially following the pattern of the proof of the theorem, we arrive at the following corollary.

Corollary 9. *Let $\mathcal{R}$ be a finite constructor TRS. Assume further $\mathcal{R}$ is compatible with $>_{\mathsf{pop}^*}$, i.e., $\mathcal{R} \subseteq >_{\mathsf{pop}^*}$. Then for all $f \in \mathcal{F}$ of arity n, with $\mathsf{nrm}(f) = \{i_1,\dots,i_p\}$ and for all values $s_1,\dots,s_n$: $\mathsf{DI}_{(\mathcal{R},\xrightarrow{\mathsf{p}})}(f(s_1,\dots,s_n))$ is bounded by a polynomial in the sum of the sizes of the normal argument terms $s_{i_1},\dots,s_{i_p}$.*

Remark 10. Beckmann and Weiermann observed in [5] that general rewriting is too powerful to serve as a suitable computation model to characterise the class of polytime computable functions as a TRS. Their notion of a *feasible* rewrite system is reflected adequately in the notion of predicative rewriting.

4 Polynomial Path Order on Sequences

In this section we extend definitions and results originally presented in [1]. The main aim is to define a *polynomial path order* $\blacktriangleright$ *on sequences of terms* such that $\blacktriangleright$ induces polynomial derivation length with respect to a compatible TRS $\mathcal{R}$.

Let $\odot \notin \mathcal{F}^{\mathsf{n}}$ be a variadic function symbol. We extend the normalised signature $\mathcal{F}^{\mathsf{n}}$ by $\odot$ and define $\mathcal{S}\mathrm{eq}(\mathcal{F}^{\mathsf{n}},\mathcal{V}) = \mathcal{T}(\mathcal{F}^{\mathsf{n}} \cup \{\odot\},\mathcal{V})$. Elements of $\mathcal{S}\mathrm{eq}(\mathcal{F}^{\mathsf{n}},\mathcal{V})$ are sometimes referred to as *sequences*. Instead of $\odot(s_1,\dots,s_n)$, we usually write $(s_1 \cdots s_n)$ and denote the empty sequence $()$ as $\varnothing$. Let $a = (a_1 \dots a_n)$ and $b = (b_1 \dots b_m)$ be elements of $\mathcal{S}\mathrm{eq}(\mathcal{F}^{\mathsf{n}},\mathcal{V})$. For $a \neq \varnothing$ and $b \neq \varnothing$ define $a \mathbin{@} b = (a_1 \dots a_n\ b_1 \dots b_m)$. If $a = \varnothing$ $(b = \varnothing)$ we set $a \mathbin{@} b = b$ $(a \mathbin{@} b = a)$.

Let $>$ denote the precedence on $\mathcal{F}^{\mathsf{n}}$ induced by the total precedence $>$ on $\mathcal{F}$. Buchholz [9] was the first to observe that finite term rewrite systems compatible with recursive path orders $\succ$ are even compatible to *finite* approximations of $\succ$. This observation carries over to polynomial path orders. The following definitions generalise the *path order on* **FP** (*POP* for short) as defined in [1]. To keep this exposition short, we only state the definition of *approximations* of the *polynomial path order* $\blacktriangleright$ *on sequences*. The general definitions for $\succcurlyeq$ and $\blacktriangleright$ is obtained by dropping the restrictions on depth and width, cf. [3]. Note that $\blacktriangleright$ can be conceived as the *limit* of the finite approximations $\blacktriangleright_k$. We use the convention that $f \in \mathcal{F}^{\mathsf{n}}$, i.e., $s = f(s_1, \dots, s_n)$ implicitly indicates that $f \neq \odot$.

Definition 11. *Let $k, l \geqslant 1$ and let $>$ be a precedence. We define $\succcurlyeq_k^l$ inductively as follows: $s \succcurlyeq_k^l t$ for $s = f(s_1, \dots, s_n)$ or $s = (s_1 \cdots s_n)$ if one of the following alternatives holds:*

1. *s_i $(\succcurlyeq^{=})_k^l$ t for some $i \in \{1, \dots, n\}$,*
2. *$s = f(s_1, \dots, s_n)$ such that of the following two possibilities holds:*
   - *$t = g(t_1, \dots, t_m)$ with $f > g$ or*
   - *$t = (t_1 \cdots t_m)$,*

   and $s \succcurlyeq_k^{l-1} t_i$ for all $1 \leqslant i \leqslant m$, and $m < k + \mathsf{wd}(s)$, or
3. *$s = (s_1 \cdots s_n)$, $t = (t_1 \cdots t_m)$ and the following properties hold:*
   - *$[t_1, \dots, t_m] = N_1 \uplus \cdots \uplus N_n$,*
   - *there exists $i \in \{1, \dots, n\}$ such that $[s_i] \neq N_i$,*
   - *for all $1 \leqslant i \leqslant n$ such that $[s_i] \neq N_i$ we have $s_i \succcurlyeq_k^l r$ for all $r \in N_i$*
   - *$m < k + \mathsf{wd}(s)$.*

We write $\succcurlyeq_k$ to abbreviate $\succcurlyeq_k^k$.

Definition 12. *Let $k, l \geqslant 1$ and let $>$ be a precedence. We define the* approximation of the polynomial path order $\blacktriangleright_k^l$ on sequences *inductively as follows: $s \blacktriangleright_k^l t$ for $s = f(s_1, \dots, s_n)$ or $s = (s_1 \cdots s_n)$ if one of the following alternatives holds:*

1. *$s \succcurlyeq_k^l t$,*
2. *s_i $(\blacktriangleright^{=})_k^l$ t for some $i \in \{1, \dots, n\}$,*
3. *$s = f(s_1, \dots, s_n)$, $t = (t_1 \cdots t_m)$, and the following properties hold:*
   - *$s \blacktriangleright_k^{l-1} t_{i_0}$ for some $i_0 \in \{1, \dots, n\}$,*
   - *$s \succcurlyeq_k^{l-1} t_i$ for all $i \neq i_0$, and*
   - *$m < k + \mathsf{wd}(s)$,*
4. *$s = f(s_1, \dots, s_n)$, $t = f(t_1, \dots, t_m)$ with $(s_1 \cdots s_n) \blacktriangleright_k^l (t_1 \cdots t_m)$, or*
5. *$s = (s_1 \cdots s_n)$, $t = (t_1 \cdots t_m)$ and the following properties hold:*
   - *$[t_1, \dots, t_m] = N_1 \uplus \cdots \uplus N_n$,*
   - *there exists $i \in \{1, \dots, n\}$ such that $[s_i] \neq N_i$,*
   - *for all $1 \leqslant i \leqslant n$ such that $[s_i] \neq N_i$: $s_i \succcurlyeq_k^l r$ for all $r \in N_i$, and*
   - *$m < k + \mathsf{wd}(s)$.*

We write $\blacktriangleright_k$ to abbreviate $\blacktriangleright_k^k$.

Note that $\varnothing$ is the minimal element of $\succcurlyeq_k$ and $\blacktriangleright_k$ and that $\blacktriangleright$ is a reduction order. The following lemmas are direct consequences of the definitions.

Lemma 13.

1. *If $s \blacktriangleright_k t$ and $k < l$, then $s \blacktriangleright_l t$.*
2. *If $s \blacktriangleright_k t$, then $C[s] \blacktriangleright_k C[t]$, where $C[\Box]$ denotes a context over $\mathcal{S}\mathrm{eq}(\mathcal{F}^{\mathsf{n}}, \mathcal{V})$.*

Lemma 14. *If $s \blacktriangleright_k^l t$, then $\mathsf{dp}(t) \leqslant \mathsf{dp}(s) + l$ and $\mathsf{wd}(t) \leqslant k + \mathsf{wd}(s)$. Moreover, if $s \blacktriangleright_k^l t$, then $\|t\| \leqslant \|s\| + k + l$.*

By Lemma 14, there exists a (uniform) constant c such that $\|t\| \leqslant \|s\| + c$, whenever $s \blacktriangleright_k t$. And thus if we have a $\blacktriangleright_k$-descending sequence $s = t_0 \blacktriangleright_k t_1 \blacktriangleright_k \cdots \blacktriangleright_k t_\ell$ we conclude that $\|t_i\| \leqslant ci + \|s\|$ for all $i \geqslant 1$.

Definition 15. *We define*

$$\mathsf{G}_k(s) := \max\{\ell \in \mathbb{N} \mid \exists(t_0, \ldots, t_\ell) \colon s = t_0 \blacktriangleright_k t_1 \blacktriangleright_k \cdots \blacktriangleright_k t_\ell\}$$
$$\mathsf{F}_{k,p}(m) := \max\{\mathsf{G}_k(f(t_1, \ldots, t_n)) \colon \mathsf{rk}(f) = p \wedge \sum_i \mathsf{G}_k(t_i) \leqslant m\}$$

In the definition of $\mathsf{F}_{k,p}$, we assume $f \in \mathcal{F}^{\mathsf{n}}$.

A direct consequence of Definition 15 is that $\mathsf{G}_k((t_1 \cdots t_n)) = n + \sum_{i=1}^n \mathsf{G}_k(t_i)$ holds. The following lemma is generalisation of a similar lemma in [1] and the proof given in [1] can be easily adapted.

Lemma 16. *We define $d_{k,0} := k + 1$ and $d_{k,p+1} := (d_{k,p})^k + 1$. Then for all k, p there exists a constant c (depending only on k and p) such that for all m: $\mathsf{F}_{k,p}(m) \leqslant c(m+2)^{d_{k,p}}$.*

As a consequence of Lemma 16 we obtain that $\mathsf{F}_{k,p}(m)$ is asymptotically bounded by $m^{d_{k,p}}$ for large enough m. The following lemma follows by a standard inductive argument.

Lemma 17. *For all k, there exists a constant c such that for $s \in \mathcal{T}(\mathcal{C}^{\mathsf{n}} \cup \{\odot\}, \mathcal{V})$: $\mathsf{G}_k(s) \leqslant c \cdot \mathsf{size}(s)^2$.*

We arrive at the main theorem of this section.

Theorem 18. *For all $f \in \mathcal{F}^{\mathsf{n}}$ of arity n, for all $s_1, \ldots, s_n \in \mathcal{T}(\mathcal{C}^{\mathsf{n}} \cup \{\odot\})$, and for all k: $\mathsf{G}_k(f(s_1, \ldots, s_n))$ is bounded by a polynomial in the sum of the sizes of $s_1, \ldots, s_n$. The polynomial depends only on k and the rank of f.*

Proof. Let $f \in \mathcal{F}^{\mathsf{n}}$ and let $s_1, \ldots, s_n \in \mathcal{T}(\mathcal{C}^{\mathsf{n}} \cup \{\odot\})$. By Lemma 16 there exists $c_1 \in \mathbb{N}$ depending on k and $\mathsf{rk}(f)$ such that

$$\mathsf{G}_k(f(s_1, \ldots, s_n)) \leqslant m^{c_1} \tag{1}$$

if $\sum_i \mathsf{G}_k(s_i) \leqslant m$ and m is large enough. By Lemma 17, there exists a constant c_2 (depending on the rank of the function symbols in s_i) such that $\mathsf{G}_k(s_i) \leqslant c_2 \cdot \mathsf{size}(s_i)^2$. Replacing m in (1) by $c_2 \cdot (\sum_i \mathsf{size}(s_i))^2$ and setting $c = c_2^{c_1}$ yields:

$$\mathsf{G}_k(f(s_1, \ldots, s_n)) \leqslant [c_2 \cdot (\sum_i \mathsf{size}(s_i))^2]^{c_1} = c \cdot (\sum_i \mathsf{size}(s_i))^{2c_1}$$

□

5 Predicative Interpretation

The purpose of this section is to prove our main theorem. Let $\mathcal{R}$ denote a completely defined, constructor TRS. We embed the order $>_{\mathsf{pop}*}$ into $\blacktriangleright_k$ such that k depends only on $\mathcal{R}$. This becomes possible if we represent the information on normal and safe arguments underlying the definition of $>_{\mathsf{pop}*}$ explicitly by interpreting the signature $\mathcal{F}$ in the normalised signature $\mathcal{F}^{\mathsf{n}}$.

Definition 19. *The* maximal size *of the safe values of a term t is defined as follows:*

$$\mathsf{sv}(t) = \begin{cases} \|t\| & t \in \mathsf{Val}(\mathcal{R}) \\ \max\{\mathsf{sv}(t_i) \mid i \in \mathsf{safe}(f)\} & t = f(t_1, \dots, t_n) \text{ and } t \notin \mathsf{Val}(\mathcal{R}) \end{cases}$$

We represent $\mathsf{sv}(t)$ unary. Let s denote a fresh constant that is minimal in the precedence $>$ on $\mathcal{F}^{\mathsf{n}}$. We define $\mathsf{SV}(t) = \mathsf{U}(\mathsf{sv}(t))$, where $\mathsf{U}\colon \mathbb{N} \to \mathcal{T}(\{\mathsf{s}, \odot\})$ denotes the representation of n as a sequence $(\mathsf{s} \ \cdots \ \mathsf{s})$ with n occurrences of the constant s. As a direct consequence of the definition, we have: $s \rhd t$ implies $\mathsf{SV}(s) \blacktriangleright_k \mathsf{SV}(t)$ for any k.

Definition 20. *Let* safe *denote a safe mapping. A* predicative interpretation *(with respect to $\mathcal{T}(\mathcal{F}, \mathcal{V})$) is a pair (S, N) of mappings $\mathsf{S}\colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}^{\mathsf{n}}, \mathcal{V})$ and $\mathsf{N}\colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}^{\mathsf{n}}, \mathcal{V})$, defined as follows:*

$$\mathsf{S}(t) = \begin{cases} \varnothing & \text{if } t \in \mathsf{Val}(\mathcal{R}) \\ (f^{\mathsf{n}}(\mathsf{N}(s_{j_1}), \dots, \mathsf{N}(s_{j_p}))\ \mathsf{S}(s_{i_1})\ \dots\ \mathsf{S}(s_{i_q})) & \text{if } t \notin \mathsf{Val}(\mathcal{R}) \end{cases}$$
$$\mathsf{N}(t) = (\mathsf{S}(t)) \mathbin{@} \mathsf{SV}(t)$$

In the definition of S, we assume $t = f(s_1, \dots, s_n)$, $\mathsf{nrm}(f) = \{j_1, \dots, j_p\}$ and $\mathsf{safe}(f) = \{i_1, \dots, i_q\}$. (Recall that $\mathsf{safe}(f) \cup \mathsf{nrm}(f) = \{1, \dots, n\}$.)

Note that $\mathsf{N}(s) \succcurlyeq_k \mathsf{S}(s)$ (and thus $\mathsf{N}(s) \blacktriangleright_k \mathsf{S}(s)$) holds for any k. We arrive at the two main lemmas of this section.

Lemma 21. *Let $f(l_1, \dots, l_n) \to r \in \mathcal{R}$, let $\sigma\colon \mathcal{V} \to \mathsf{Val}(\mathcal{R})$ be a substitution and let $k = 2 \cdot \max\{\mathsf{size}(r) \mid l \to r \in \mathcal{R}\}$. If $f(l_1, \dots, l_n) >_{\mathsf{pop}} r$ then $f^{\mathsf{n}}(\mathsf{N}(l_{i_1}\sigma), \dots, \mathsf{N}(l_{i_p}\sigma)) \succcurlyeq_k \mathsf{Q}(r\sigma)$ for $\mathsf{Q} \in \{\mathsf{S}, \mathsf{N}\}$, where $\mathsf{nrm}(f) = \{i_1, \dots, i_p\}$.*

Proof. We sketch the proof plan: Instead of showing the lemma directly, one shows the following stronger property for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ where s is either a value or of form $f(s_1, \dots, s_n)$ such that $s_i\sigma \in \mathsf{Val}(\mathcal{R})$ for all $1 \leqslant i \leqslant n$.

(†) Let $\ell = \|t\|$, if $f \in \mathcal{D}$, then $s >_{\mathsf{pop}} t$ implies $\mathsf{Q}(s\sigma) \succcurlyeq_{2\ell} f^{\mathsf{n}}(\mathsf{N}(s_1\sigma), \dots, \mathsf{N}(s_p\sigma)) \succcurlyeq_{2\ell} \mathsf{Q}(t\sigma)$; otherwise $\mathsf{N}(s\sigma) \succcurlyeq_{2\ell} \mathsf{N}(t\sigma)$ holds.

Here we suppose $\mathsf{safe}(f) = \{p+1, \dots, n\}$. To show (†) one proceeds by induction on $>_{\mathsf{pop}}$. See [3] for the complete proof. □

Lemma 22. *Let $l \to r \in \mathcal{R}$, let $\sigma\colon \mathcal{V} \to \mathsf{Val}(\mathcal{R})$ be a substitution, and let $k = 2 \cdot \max\{\mathsf{size}(r) \mid l \to r \in \mathcal{R}\}$. If $l >_{\mathsf{pop}*} r$ then $\mathsf{Q}(l\sigma) \blacktriangleright_k \mathsf{Q}(r\sigma)$ for $\mathsf{Q} \in \{\mathsf{S}, \mathsf{N}\}$.*

Proof. Similar to the proof of Lemma 21 one shows the following property for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ where s is either a value or of form $f(s_1, \dots, s_n)$ such that $s_i\sigma \in \mathsf{Val}(\mathcal{R})$ for all $1 \leqslant i \leqslant n$.

(‡) Let $\ell = \|t\|$. If $f \in \mathcal{D}$, then $s = f(s_1, \dots, s_n) >_{\mathsf{pop}*} t$ implies (i) $f^{\mathsf{n}}(\mathsf{N}(s_1\sigma), \dots, \mathsf{N}(s_p\sigma)) \blacktriangleright_{2\ell} \mathsf{S}(t\sigma)$ and (ii) $(f^{\mathsf{n}}(\mathsf{N}(s_1\sigma), \dots, \mathsf{N}(s_p\sigma))) \mathbin{@} \mathsf{SV}(s\sigma) \blacktriangleright_{2\ell} \mathsf{N}(t\sigma)$. Otherwise if $f \in \mathcal{C}$ then $\mathsf{N}(s\sigma) \blacktriangleright_{2\ell} \mathsf{N}(t\sigma)$ holds.

Here we suppose $\mathsf{safe}(f) = \{p+1, \dots, n\}$. To show (‡) one proceeds by induction on $>_{\mathsf{pop}*}$. See [3] for the complete proof. □

From Lemmata 21 and 22 the main lemma of this section follows.

Lemma 23. *Let s and t be terms such that $s \xrightarrow{\mathsf{i}} t$ and let $k = 2 \cdot \max\{\mathsf{size}(r) \mid l \to r \in \mathcal{R}\}$. Then $\mathsf{Q}(s) \blacktriangleright_k \mathsf{Q}(t)$ for $\mathsf{Q} \in \{\mathsf{S}, \mathsf{N}\}$.*

Main Theorem. *Let $\mathcal{R}$ be a finite, completely defined constructor TRS. Assume further $\mathcal{R}$ is compatible with $>_{\mathsf{pop}*}$. Then the induced (normal) runtime complexity is polynomial.*

Proof. Let $t = f(t_1, \dots, t_n)$ be term in $\mathcal{T}_{\mathrm{b}}$ and without loss of generality let $\mathsf{safe}(f) = \{p+1, \dots, n\}$. We set $k = 2 \cdot \max\{\mathsf{size}(r) \mid l \to r \in \mathcal{R}\}$. By Lemma 23 any innermost rewrite steps $t \xrightarrow{\mathsf{i}} u$ induces $\mathsf{S}(t) \blacktriangleright_k \mathsf{S}(u)$. Thus we obtain:

$$\begin{aligned}\mathsf{DI}_{(\mathcal{R}, \xrightarrow{\mathsf{i}})}(f(t_1, \dots, t_n)) &= \max\{\ell \mid \exists u\ t \xrightarrow{\mathsf{i}}{}^{\ell} u\} \\ &\leqslant \max\{\ell \mid \exists\, (s'_1, \dots, s'_\ell)\colon \mathsf{S}(t) \blacktriangleright_k s'_1 \blacktriangleright_k \cdots \blacktriangleright_k s'_\ell\} \\ &\leqslant \mathsf{G}_k(\mathsf{S}(f(t_1, \dots, t_n)))\end{aligned}$$

Next notice that $\mathsf{S}(f(t_1, \dots, t_n)) = (f^{\mathsf{n}}(\mathsf{N}(t_1), \dots, \mathsf{N}(t_p)) \varnothing \dots \varnothing)$. By Theorem 18 and the observation following Definition 15 we see that

$$\mathsf{G}_k((f^{\mathsf{n}}(\mathsf{N}(t_1), \dots, \mathsf{N}(t_p)) \varnothing \dots \varnothing)) \leqslant n + 1 + \mathsf{G}_k(f^{\mathsf{n}}(\mathsf{N}(t_1), \dots, \mathsf{N}(t_p)))$$

Employing Lemma 16, we see (for a fixed f) that $n+1+\mathsf{G}_k(f^{\mathsf{n}}(\mathsf{N}(t_1), \dots, \mathsf{N}(t_p)))$ is asymptotically bounded by a polynomial in the sum of the sizes of the arguments $\mathsf{N}(t_1), \dots, \mathsf{N}(t_p)$. By definition $\mathsf{size}(\mathsf{N}(t_i)) = \|t_i\| \leqslant \mathsf{size}(t_i)$ for all $1 \leqslant i \leqslant p$.

Hence for each term $t \in \mathcal{T}_{\mathrm{b}}$, $\mathsf{DI}_{(\mathcal{R}, \xrightarrow{\mathsf{i}})}(t)$ is bounded by a polynomial in the sum of the sizes of the normal argument terms of t. In particular, as the signature $\mathcal{F}$ is finite, the normal runtime complexity function is polynomial. □

Remark 24. In the above theorem we assume a constructor TRS. It is not difficult to see that this restriction is not necessary. (Essentially one replaces the application of Lemmata 21 and 22 by the application of the properties (†) and (‡) respectively.) However, the restriction that the arguments of f are in normal form is necessary. Hence we prefer the given formulation of the theorem.

6 Experimental Data

To prove compatibility of a given TRS $\mathcal{R}$ with recursive path orders we have to find a *precedence* $>$ such that the induced order is compatible with $\mathcal{R}$. When we want to orient $\mathcal{R}$ by a polynomial path order $>_{\mathsf{pop}*}$ we additionally require a suitable *safe mapping*. To automate this search we encode the constraint $s >_{\mathsf{pop}*} t$ into a propositional formula:

$$\tau(s >_{\mathsf{pop}*} t) = \tau_1(s >_{\mathsf{pop}*} t) \vee \tau_2(s >_{\mathsf{pop}*} t) \vee \tau_3(s >_{\mathsf{pop}*} t) \vee \tau_4(s >_{\mathsf{pop}*} t)$$

Here $\tau_i(\cdot)$ is designed to encode clause (i) from Definition 4. Based on such an encoding, compatibility of a TRS with $>_{\mathsf{pop}*}$ becomes expressible as the satisfiability of the formula $\big(\bigwedge_{l\to r\in\mathcal{R}} \tau(l >_{\mathsf{pop}*} r)\big) \wedge P \wedge S$. Here the subformula P is satisfiable if and only if all the variables $>_{f,g}$ (defined below) encode a strict precedence, see [26] for a suitable definition of P. The subformula S is used to cover the additional conditions imposed on safe mappings defined in the beginning of Section 3.

We only describe cases (2)–(4), the encoding for case (1)—the comparison using the weaker order $>_{\mathsf{pop}}$—can be easily derived in a similar fashion. If $s = f(s_1,\dots,s_n)$ we set $\tau_2(s >_{\mathsf{pop}*} t) = \bigvee_i s_i >^{=}_{\mathsf{pop}*} t$, otherwise $\tau_2(s >_{\mathsf{pop}*} t) = \bot$. For case (3) we introduce for every function symbol f and argument position i of f the (propositional) variables $\beta_{f,i}$, such that $\beta_{f,i} = \text{true}$ represents the assertion $i \in \mathsf{safe}(f)$. Moreover, for all function symbols f, g we introduce variables $>_{f,g}$ such that truth of $>_{f,g}$ expresses that $f > g$ holds. If $s = f(s_1,\dots,s_n)$ and $t = g(t_1,\dots,t_m)$ for $f \in \mathcal{D}$ with $f \neq g$, we define $\tau_3(s >_{\mathsf{pop}*} t)$ as:

$$>_{f,g} \;\wedge\; \bigvee_{i_0=1}^{m} (\tau(s >_{\mathsf{pop}*} t_i) \;\wedge\; \beta_{g,i_0} \;\wedge \bigwedge_{i=1,i\neq i_0}^{m} (\tau(s >_{\mathsf{pop}} t_i) \vee (\beta_{g,i} \wedge (s \rhd t_i)))$$

(For s, t of different shape, we set $\tau_3(s >_{\mathsf{pop}*} t) = \bot$.) To deal with case (4) we follow [19]. The main idea is to describe a multiset comparison in terms of *multiset covers*. Formally, a multiset cover is a pair of mappings $\gamma\colon \{1,\dots,m\} \to \{1,\dots,n\}$ and $\varepsilon\colon \{1,\dots,n\} \to \{\text{true},\text{false}\}$ such that for all i, j ($1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$): if $\varepsilon(i) = \text{true}$ then the set $\{j \mid \gamma(j) = i\}$ is a singleton. It is easy to see that $[s_1,\dots,s_n]\ (\succ^{=})_{\mathsf{mul}}\ [t_1,\dots,t_m]$ if there exists a multiset cover (γ,ε) such that for each j there exists an i with $\gamma(j) = i$ and $\varepsilon(i) = \text{true}$ implies $s_i = t_j$, while $\varepsilon(i) = \text{false}$ implies $s_i \succ t_j$. Similarly we obtain $[s_1,\dots,s_n] \succ_{\mathsf{mul}} [t_1,\dots,t_m]$ if $[s_1,\dots,s_n]\ (\succ^{=})_{\mathsf{mul}}\ [t_1,\dots,t_m]$ and $\varepsilon(i) = \text{false}$ for some $i \in \{1,\dots,n\}$.

This definition allows an easy encoding of multiset comparisons and based on it, clause (4) of Definition 4 becomes representable (for terms $s = f(s_1,\dots,s_n)$ and $t = f(t_1,\dots,t_m)$) as the conjunction of the following two conditions together with the assumption that there exists a suitable multiset cover (γ,ε):

- whenever $\gamma(j) = i$ then the indicated argument positions i and j, are either both normal or both safe,

– at least one cover is strict ($\varepsilon(i) = \text{false}$) for some normal argument position i of f.

We introduce variables $\gamma_{i,j}$ and ε_i, where $\gamma_{i,j} = \text{true}$ represents $\gamma(j) = i$ and $\varepsilon_i = \text{true}$ denotes $\varepsilon(i) = \text{true}$ ($1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$). Summing up, we set $\tau_4(s\ (>_{\mathsf{pop}*})_{\mathsf{mul}}\ t)$ ($s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_m)$) equal to:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \Big(\gamma_{i,j} \to \big(\varepsilon_i \to \tau(s_i = t_j)\big) \wedge \big(\neg\varepsilon_i \to \tau(s_i \succ t_j)\big) \wedge \big(\beta_{f,i} \leftrightarrow \beta_{f,j}\big)\Big)$$
$$\wedge \bigwedge_{j=1}^{m} \mathrm{one}(\gamma_{1,j}, \ldots, \gamma_{n,j}) \wedge \bigwedge_{i=1}^{n} \big(\varepsilon_i \to \mathrm{one}(\gamma_{i,1}, \ldots, \gamma_{i,m})\big) \wedge \bigvee_{i=1}^{n} \big(\neg\beta_{f,i} \wedge \neg\varepsilon_i\big)$$

Here $\mathrm{one}(\alpha_1, \ldots, \alpha_n)$ is satisfiable if and only if exactly one of the variables $\alpha_1, \ldots, \alpha_n$ is true. And if s, t do not have the assumed form, we set $\tau_4(s\ (>_{\mathsf{pop}*})_{\mathsf{mul}}\ t) = \bot$.

We compare the polynomial path order POP* to a restricted class of polynomial interpretations (SMC for short) [7] and to LMPO [17]. SMC refers to simple-mixed polynomial interpretations where constructor symbols are interpreted by a strongly linear (also called *additive*) polynomial [7]. Defined symbols on the other hand are interpreted by simple-mixed polynomials [10]. Since POP* and LMPO are in essence syntactic restrictions of MPO we also provide a comparison to MPO. POP* is implemented using the previously described propositional encoding; while the implementation of SMC rests on a propositional encoding of the techniques described in [10]. To check satisfiability we employ MiniSat.[4] LMPO and MPO are implemented using an extension of the constraint solving technique described in [11], which allows us to compare different implementation techniques at the same time.

As testbed we use those TRSs from the termination problem data base version 4.0 that can be shown terminating with at least one of the tools that participated in the termination competition 2007.[5] We use three different testbeds: **T** collects the 957 terminating TRSs from TPDB, **TC** collects the 449 TRSs from the TPDB that are also constructor systems, and **TCO** collects the 236 TRSs that are terminating, constructor based and orthogonal.[6] The results of our comparisons are given in Table 1. The tests presented below were conducted on a small complexity analyser running single-threaded on a 2.1 GHz Intel Core 2 Duo with 1 GB of memory. For each system we used a timeout of 30 seconds.

Some comments: What is noteworthy is the good performance of POP*as a direct *termination* method in comparison to MPO. It is well-known that MPO implies primitive recursive derivation length, cf. [12]. In contrast to this POP*

[4] Available online at `http://minisat.se`.

[5] These 957 systems can be found online: `http://www.lri.fr/~marche/termination-competition/2007/webform.cgi?command=trs&file=trs-standard.db&timelimit=120`

[6] The main reason for this delineation is that in related work [7,17] *confluent* constructor TRS are considered.

Table 1. Experimental results

		POP*	LMPO	SMC	MPO
T	Yes	65	74	156	106
	Maybe	892	812	395	847
	Timeout (30 sec.)	0	71	406	4
TC	Yes	41	54	83	65
	Maybe	408	372	271	381
	Timeout (30 sec.)	0	23	95	3
TCO	Yes	19	25	38	29
	Maybe	217	201	147	207
	Timeout (30 sec.)	0	10	51	0
Average yes time (milliseconds)		15	14	1353	10

implies polynomial runtime complexity and is thus a much weaker order. Still more than half of the TRSs compatible with MPO are also compatible with POP*. On the other hand the comparison between POP* and LMPO is quite favourable for our approach. Compatibility with LMPO tells us that the given TRS is (in principle) polytime computable, while compatibility with POP* tells additionally that the runtime of a straightforward implementation (using an innermost strategy) is polytime computable. Hence compatibility with POP* provides us with a theoretical stronger result, while the difference on the experimental data appears negligible.

The good performance of SMC in strength is a clear indication that currently (restrictions of) semantic termination techniques (like polynomial interpretations) are of some interest in automatically estimating the runtime complexity of TRSs. This may be surprising, as for additive polynomial interpretations it is (almost) trivial to check that the induced upper bound on the derivation height is polynomial. However, the significant increase in the time necessary to find an additive polynomial interpretation, as indicated in Table 1, clearly shows the limits of semantic methods for large examples.

7 An Application: Complexity of Scheme Programs

In recent work together with Hirokawa and Middeldorp (see [2]) we study the runtime complexity of (a subset of) Scheme programs by a translation into so-called *S-expression rewrite systems* (*SRS* for short). By designing the translation to be complexity preserving, the complexity of the initial Scheme program can be estimated by analysing the complexity of the resulting SRS. Here we indicate how our main theorem is applicable to (a subset of) S-expression rewrite systems, cf. [24].

Definition 25. *Let $\mathcal{K}$ be a set of constants, $\mathcal{V}$ be a set of variables such that $\mathcal{V} \cap \mathcal{K} = \varnothing$, and $\circ \notin \mathcal{K} \cup \mathcal{V}$ a variadic function symbol. We define the set $\mathcal{S}(\mathcal{K}, \mathcal{V})$ of* S-expressions *built from $\mathcal{K}$ and $\mathcal{V}$ as $\mathcal{T}(\mathcal{K} \cup \{\circ\}, \mathcal{V})$. We write $(s_1 \cdots s_n)$ instead of $\circ(s_1, \ldots, s_n)$. An* S-expression rewrite system *(*SRS *for short) is a TRS with the property that the left- and right-hand sides of all rewrite rules are S-expressions.*

Let $\mathcal{S}$ be an SRS over $\mathcal{S}(\mathcal{K}, \mathcal{V})$ and let $\mathcal{K} = \mathcal{D} \cup \mathcal{C}$ such that $\mathcal{D} \cap \mathcal{C} = \varnothing$. We call the elements of $\mathcal{C}$ *constructor* constants and the elements of $\mathcal{D}$ *defined* constants. We momentarily redefine the notion of *value* in the context of SRSs. The set of *values* $\mathsf{Val}(\mathcal{S})$ of $\mathcal{S}$ with respect to $\mathcal{C}$ is inductively defined as follows:

1. if $v \in \mathcal{K}$ then $v \in \mathsf{Val}(\mathcal{S})$,
2. if $v_1, \ldots, v_n \in \mathsf{Val}(\mathcal{S})$ and $\mathsf{c} \in \mathcal{C}$ then $(\mathsf{c}\ v_1 \ldots v_n) \in \mathsf{Val}(\mathcal{S})$.

Observe that (defined) constants are values, this reflects that in Scheme procedures are values, cf. [21] and allows for a representation of higher-order programs. Scheme programs are conceivable as SRSs allowing conditional if expressions in conjunction with an eager, i.e., innermost rewrite strategy. Thus we can delineate a class of SRSs that easily accommodate a suitably large subset of Scheme programs.

Definition 26. *$\mathcal{S}$ is called a* constructor *if, for every $l \to r \in \mathcal{S}$, $l = (l_0 \cdots l_n)$ with $l_0 \in \mathcal{D}$ and $l_i \in \mathsf{Val}(\mathcal{S})$ for all $i \in \{1, \ldots, n\}$. (Here the set of values $\mathsf{Val}(\mathcal{S})$ is defined with respect to $\mathcal{C}$.)*

Corollary 27. *Let $>$ denote a precedence on $\mathcal{K}$ such that for all $f \in \mathcal{D}$ we have for all $c \in \mathcal{C}$: $f > c$ and let $>_{\mathsf{pop}*}$ denote the induced POP*. Let $\mathcal{S}$ be a constructor SRS compatible with $>_{\mathsf{pop}*}$. Then for all $f \in \mathcal{D}$ of arity n and for all values $s_1, \ldots, s_n$: $\mathsf{DI}_{(\mathcal{S}, \xrightarrow{\mathsf{i}})}((\mathsf{f}\ s_1\ \ldots\ s_n))$ is bounded by a polynomial in the sum of the sizes of the arguments $s_1, \ldots, s_n$.*

Proof. It is important to note that the set of S-expressions $\mathcal{S}(\mathcal{K}, \mathcal{V})$ equals $\mathcal{T}(\mathcal{K} \cup \{\circ\}, \mathcal{V})$, i.e., SRSs are *first-order* rewrite systems, whose single defined symbol is the variadic function symbol $\circ$.

Hence Theorem 27 follows almost immediately from Corollary 7. However the fact that according to the above definition values may contain defined symbol need to be taken into account. For that is suffices to redefine Definitions 19 and 20 in the natural way. It is not difficult to argue that suitable adaption of Lemmata 21 and 22 to SRSs are provable. □

8 Conclusion

In this paper we have introduced a restriction of the multiset path order, called *polynomial path order* (*POP** for short). Our main result states that POP* induces polynomial runtime complexity. In Section 6 we have provided evidence that our approach performs well in comparison to related methods. In Section 7

the necessary theory to apply our main theorem in the context of (higher-order) functional languages with eager evaluations has been developed. In related work (together with Hirokawa and Middeldorp), studying the termination behaviour and the runtime complexity of (a subclass of higher-order) Scheme programs, this basis has proven quite useful, cf. [2].

In concluding we also want to mention that as an easy corollary to our main theorem we obtain that POP* also characterises the polytime computable functions. To be precise the polytime computable functions are exactly the functions computable by an orthogonal constructor TRS (based on a simple signature) compatible with POP*. (Here *simple* signature means that the size of any constructor term depends linearly on its depth, an equivalent restriction is necessary in [17].) See [3] for details.

In future work we will strengthen the applicability of our method. The experimental evidence presented in Section 6 shows that compatibility of rewrite systems with POP* can be easily and quickly tested. However, the strength of the method seems to be improvable. One possible field of future work is to extend POP* to quasi-precedences. The theoretical changes necessary to accomodate quasi-precedences seem to be manageable. Another natural extension is to combine POP* with the transformation technique of semantic labeling, cf. [27]. It is easy to see that semantic labeling (in the basic form) does not affect the derivation length. Furthermore for *finite* models the main theorem remains directly applicable.

References

1. T. Arai and G. Moser. Proofs of termination of rewrite systems for polytime functions. In *Proc. 25th FSTTCS*, number 3821 in LNCS, pages 529–540, 2005.
2. M. Avanzini, N. Hirokawa, A. Middeldorp, and G. Moser. Towards an automatic runtime complexity analysis of Scheme programs by rewriting. Submitted for publication, December 2007.[7]
3. M. Avanzini and G. Moser. Complexity analysis by rewriting. Draft, October 2007.[7]
4. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
5. A. Beckmann and A. Weiermann. A term rewriting characterization of the polytime functions and related complexity classes. *Archive*, 36:11–30, 1996.
6. S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Comput. Complexity*, 2(2):97–110, 1992.
7. G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *JFP*, 11(1):33–53, 2001.
8. G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-intepretations and small space bounds. In *Proc. 16th RTA*, number 3467 in LNCS, pages 150–164, 2005.
9. W. Buchholz. Proof-theoretic analysis of termination proofs. *APAL*, 75:57–65, 1995.

[7] Available online at `http://cl-informatik.uibk.ac.at/~georg/list.publications.html`.

10. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *JAR*, 34(4):325–363, 2005.
11. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proc. 14th RTA*, number 2706 in LNCS, pages 311–32, 2003.
12. D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *TCS*, 105(1):129–140, 1992.
13. D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. 3rd RTA*, number 355 in LNCS, pages 167–177, 1989.
14. M. Hofmann. Linear types and non-size increasing polynomial time compuations. In *Proc. 14th LICS*, pages 464–473, 1999.
15. I. Lepper. Derivation lengths and order types of Knuth-Bendix orders. *TCS*, 269:433–450, 2001.
16. S. Lucas and R. Peña. Termination and complexity bounds for SAFE programs. In *Proc. 7th PROLE*, pages 233–242, 2007.
17. J. Marion. Analysing the implicit complexity of programs. *IC*, 183:2–18, 2003.
18. G. Moser. Derivational complexity of Knuth Bendix orders revisited. In *Proc. 13th LPAR*, number 4246 in LNCS, pages 75–89, 2006.
19. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. 6th FroCos*, number 4720 in LNCS, pages 267–282, 2007.
20. H. Schwichtenberg. An arithmetic for polynomial-time computation. *TCS*, 357(1):202–214, 2006.
21. M. Sperber, R. K. Dybvig, M. Flatt, and A. v. Stratten et al. Revised^6 report on the algorithmic language Scheme. Available online at `www.r6rs.org`., 2007.
22. J. Steinbach and U. Kühler. Check your ordering - termination proofs and open problems. Technical Report SEKI-Report SR-90-25, University of Kaiserslautern, 1990.
23. Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
24. Y. Toyama. Termination of S-expression rewriting systems: Lexicographic path ordering for higher-order terms. In *Proc. 15th RTA*, number 3091 in LNCS, pages 40–54, 2004.
25. A. Weiermann. Termination proofs for term rewriting systems with lexicographic path ordering imply multiply recursive derivation lengths. *TCS*, 139:355–362, 1995.
26. H. Zankl and A. Middeldorp. Satisfying KBO constraints. In *Proc. of 18th RTA*, number 4533 in LNCS, pages 389–403, 2007.
27. H. Zantema. Termination of term rewriting by semantic labelling. *FI*, 24:89–105, 1995.