Automated Complexity Analysis Based on Context-Sensitive Rewriting*

Nao Hirokawa[1] and Georg Moser[2]

[1] School of Information Science, JAIST, Japan
hirokawa@jaist.ac.jp

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

Abstract. In this paper we present a simple technique for analysing the runtime complexity of rewrite systems. In complexity analysis many techniques are based on reduction orders. We show how the monotonicity condition for orders can be weakened by using the notion of context-sensitive rewriting. The presented technique is very easy to implement, even in a modular setting, and has been integrated in the Tyrolean Complexity Tool. We provide ample experimental data for assessing the viability of our method.

Keywords: term rewriting, complexity analysis, context-sensitive rewriting, automation

1 Introduction

This paper is concerned with runtime complexity analysis of term rewrite systems. In recent years the field of complexity analysis of rewrite systems has been dramatically revived. Nowadays this area provides a wide range of different techniques to analyse the time complexity of rewrite systems, fully automatically. Techniques range from *direct methods*, like *polynomial interpretations*, *matrix interpretations* of *polynomial path orders* (e.g. [1–3]) to *transformation techniques*, like adaptions of the *dependency pair method* [4, 5] or modular techniques [6, 7]. See [8] for an overview of complexity analysis methods for term rewrite systems. Furthermore the connection between (runtime) complexity analysis and *implicit computational complexity* [9] is by now well-understood, cf. [10]. Despite this wealth of results, very simple examples cannot be handled and in particular the enormous power of today's termination provers for rewrite systems is still far beyond the ability of today's complexity analysers. Modern termination provers provide termination or non-termination certificates for upto 90 % of the problems *Termination Problem Database* (*TPDB* for short), while with respect to

* This research is partly supported by JSPS KAKENHI Grant Number 25730004 and FWF (Austrian Science Fund) project I 963-N15.

automated polynomial runtime complexity analysis of rewrite systems, we see a success rate of 38 %.[3]

Consider Example 1 below. The example encodes division in a natural way. It can be analysed with the techniques mentioned above, but the optimal linear bound on the runtime complexity is not attainable.

Example 1. Consider the following TRS $\mathcal{R}_{div}$[4]

$$\begin{array}{rrcl}
1\colon & x - \mathsf{0} \to x & 3\colon & \mathsf{0} \div \mathsf{s}(y) \to \mathsf{0} \\
2\colon & \mathsf{s}(x) - \mathsf{s}(y) \to x - y & 4\colon & \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y)) \ .
\end{array}$$

The example also clarifies a difference between the *derivational* and the *runtime complexity* of rewrite systems. The *derivational complexity function* with respect to a terminating TRS relates the maximal derivation height to the size of the initial term, cf. [12, 13]. On the other hand the *runtime complexity function* with respect to a terminating TRS restricts the derivational complexity function so that only basic terms are considered as starting terms. Here basic terms refer to terms that contain a defined symbol only at root. This terminology was suggested in [4]. Related notions have been studied in [1, 14]. It is easy to see that the derivational complexity with respect to $\mathcal{R}_{div}$ bounded from below by an exponential function, while the runtime complexity is linear. Essentially this is due to the fact that, in the computation of division, no contraction is ever required below the second argument. Furthermore, dependency on the first argument is linear.

An inspection of the motivating example in the context of runtime complexity analysis reveals that direct methods are not applicable as the monotonicity constraints are too restrictive. While monotonicity is no longer an issue for transformation techniques, neither the *weak dependency pair method* [4] nor the *dependency tuple method* [5] can deduce the linear (innermost) runtime complexity, as essential constraints cannot be met. In this paper we extend the applicability of direct techniques for complexity results by showing how the monotonicity constraints can be significantly weakened through the employ of *usable replacement maps*, which govern those argument position actually used in rewriting. As usable replacement maps are not computable in general, we provide sufficiently expressive approximations of usable replacement maps. More generally, we show how notions from context-sensitive rewriting can be made applicable in the context of complexity analysis.

This paper is structured as follows. In the next section we cover basics. In Section 3 we define usable replacement maps. In Section 4 we provide experimental data that verifies that the proposed technique makes a difference in practice. Furthermore, in Section 5 we present related work and conclude in Section 6.

[3] We base the comparison on last year's run of TERMCOMP, where we consider the categories *TRS Standard* and *Runtime Complexity – Innermost Rewriting.* Note that for termination YES and NO answers have been counted.

[4] This is Example 3.1 in Arts and Giesl's collection of TRSs [11].

2 Runtime Complexity Analysis Based on Matrix Interpretations

We assume familiarity with term rewriting [13, 15] but briefly review basic concepts and notations from term rewriting, context-sensitive rewriting and recall matrix interpretations. In particular we will adapt triangular matrix interpretations for runtime complexity analysis.

Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature, such that $\mathcal{F}$ contains at least one constant. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$. The *set of positions* $\mathcal{P}\mathsf{os}(t)$ of a term t is defined as usual. We write $\mathcal{P}\mathsf{os}_{\mathcal{G}}(t) \subseteq \mathcal{P}\mathsf{os}(t)$ for the set of positions of subterms whose root symbol is contained in $\mathcal{G} \subseteq \mathcal{F}$. The subterm of t at position p is denoted as $t|_p$, and $t[u]_p$ denotes the term that is obtained from t by replacing the subterm at p by u. The subterm relation is denoted as $\trianglelefteq$. $\mathcal{V}\mathsf{ar}(t)$ denotes the set of variables occurring in a term t. The *size* $|t|$ of a term is defined as the number of symbols in t.

A *term rewrite system* (*TRS*) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a *finite* set of rewrite rules $l \to r$. In the sequel, $\mathcal{R}$ always denotes a TRS. The rewrite relation is denoted as $\to_{\mathcal{R}}$ and we use the standard notations for its transitive and reflexive closure. 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 performed to rewrite s to t, we write $s \to^n t$. With $\mathsf{NF}(\mathcal{R})$ we denote the set of all normal forms of a term rewrite system $\mathcal{R}$. 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 $\mathcal{R}$. *Defined symbols* of $\mathcal{R}$ are symbols appearing at root in left-hand sides of $\mathcal{R}$. The set of defined function symbols is denoted as $\mathcal{D}$, while the *constructor symbols* $\mathcal{F} \setminus \mathcal{D}$ are collected in $\mathcal{C}$. We call a term $t = f(t_1,\ldots,t_n)$ *basic* or *constructor based* if $f \in \mathcal{D}$ and $t_i \in \mathcal{T}(\mathcal{C},\mathcal{V})$ for all $1 \leqslant i \leqslant n$. The set of all basic terms are denoted by $\mathcal{T}_{\mathsf{b}}$. We call a TRS *(innermost) terminating* if no infinite (innermost) rewrite sequence exists.

A replacement map μ is a function with $\mu(f) \subseteq \{1,\ldots,n\}$ for all n-ary functions with $n \geqslant 1$ [16]. The set $\mathcal{P}\mathsf{os}_\mu(t)$ of *μ-replacing positions* in t is defined as follows:

$$\mathcal{P}\mathsf{os}_\mu(t) := \begin{cases} \{\epsilon\} & \text{if } t \text{ is a variable ,} \\ \{\epsilon\} \cup \{ip \mid i \in \mu(f) \text{ and } p \in \mathcal{P}\mathsf{os}_\mu(t_i)\} & \text{if } t = f(t_1,\ldots,t_n) \text{ .} \end{cases}$$

A *μ-step* $s \xrightarrow{\mu} t$ is a rewrite step $s \to t$ whose rewrite position is in $\mathcal{P}\mathsf{os}_\mu(s)$. The set of all non-$\mu$-replacing positions in t is denoted by $\overline{\mathcal{P}\mathsf{os}_\mu}(t)$; namely, $\overline{\mathcal{P}\mathsf{os}_\mu}(t) := \mathcal{P}\mathsf{os}(t) \setminus \mathcal{P}\mathsf{os}_\mu(t)$.

A *monotone $\mathcal{F}$-algebra* is a pair $(\mathcal{A},\succ)$ where $\mathcal{A}$ is an $\mathcal{F}$-algebra and $\succ$ is a proper order such that for every function symbol $f \in \mathcal{F}$, $f_{\mathcal{A}}$ is strictly monotone in all coordinates with respect to $\succ$. A (monotone) $\mathcal{F}$-algebra $(\mathcal{A},\succ)$ is called *well-founded* if $\succ$ is well-founded. Any monotone $\mathcal{F}$-algebra $(\mathcal{A},R)$ induces a binary relation $R_{\mathcal{A}}$ on terms: define $s \mathrel{R_{\mathcal{A}}} t$ if $[\alpha]_{\mathcal{A}}(s) \mathrel{R} [\alpha]_{\mathcal{A}}(t)$ for all assignments α. We say $\mathcal{A}$ is *compatible* with a TRS $\mathcal{R}$ if $\mathcal{R} \subseteq R_{\mathcal{A}}$. Let μ denote a replacement map. Then we call a well-founded algebra $(\mathcal{A}, \succ)$ *μ-monotone* if for every function symbol $f \in \mathcal{F}$, $f_{\mathcal{A}}$ is strictly monotone *on* $\mu(f)$, i.e. $f_{\mathcal{A}}$ is strictly monotone with respect to every argument position in $\mu(f)$. Similarly a relation R is called μ-monotone if it is strictly monotone on $\mu(f)$ for all $f \in \mathcal{F}$. Let $\mathcal{R}$ be a TRS compatible with a μ-monotone relation R. Then clearly any μ-step $s \xrightarrow{\mu} t$ implies $s \mathrel{R} t$.

We recall the concept of *matrix interpretations* on natural numbers (see [17] but compare also [18]). Let $\mathcal{F}$ denote a signature. We fix a dimension $d \in \mathbb{N}$ and use the set $\mathbb{N}^d$ as the carrier of an algebra $\mathcal{A}$, together with the following extension of the natural order $>$ on $\mathbb{N}$: $(x_1, x_2, \ldots, x_d) > (y_1, y_2, \ldots, y_d) :\Longleftrightarrow x_1 > y_1 \wedge x_2 \geqslant y_2 \wedge \ldots \wedge x_d \geqslant y_d$. Let μ be a replacement map. For each n-ary function symbol f, we choose as an interpretation a linear function of the following shape:

$$f_{\mathcal{A}} \colon (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \mapsto F_1 \boldsymbol{v}_1 + \cdots + F_n \boldsymbol{v}_n + \boldsymbol{f} \ ,$$

where $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ are (column) vectors of variables, $F_1, \ldots, F_n$ are matrices (each of size $d \times d$), and $\boldsymbol{f}$ is a vector over $\mathbb{N}$. Moreover, suppose for any $i \in \mu(f)$ the top left entry $(F_i)_{1,1}$ is positive. Then it is easy to see that the algebra $\mathcal{A}$ forms a μ-monotone well-founded algebra Let $\mathcal{A}$ be a matrix interpretation, let α_0 denote the assignment mapping any variable to $\mathbf{0}$, i.e. $\alpha_0(x) = \mathbf{0}$ for all $x \in \mathcal{V}$, and let t be a term. In the following we write $[t]$, $[t]_j$ as an abbreviation for $[\alpha_0]_{\mathcal{A}}(t)$, or $([\alpha_0]_{\mathcal{A}}(t))_j$ $(1 \leqslant j \leqslant d)$, respectively, if the algebra $\mathcal{A}$ is clear from the context.

The *derivation height* of a term s with respect to a well-founded, finitely branching relation $\to$ is defined as: $\mathsf{dh}(s, \to) = \max\{n \mid \exists t\ s \to^n t\}$.

Definition 2. *We define the* runtime complexity function $\mathsf{rc}_{\mathcal{R}}(n)$ *and the* innermost runtime complexity function $\mathsf{rc}^{\mathsf{i}}_{\mathcal{R}}(n)$ *as follows:*

$$\mathsf{rc}_{\mathcal{R}}(n) := \max\{\mathsf{dh}(t, \to_{\mathcal{R}}) \mid t \textit{ is basic and } |t| \leqslant n\}$$
$$\mathsf{rc}^{\mathsf{i}}_{\mathcal{R}}(n) := \max\{\mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid t \textit{ is basic and } |t| \leqslant n\} \ .$$

We may say the (innermost) runtime complexity of $\mathcal{R}$ is *linear*, *quadratic*, or *polynomial* if there exists a (linear, quadratic) polynomial $p(n)$ such that $\mathsf{rc}^{(\mathsf{i})}_{\mathcal{R}}(n) \leqslant p(n)$ for sufficiently large n.

Note that $\mathsf{dh}(t, \succ)$ is undefined, if the relation $\succ$ is not well-founded or not finitely branching. In fact compatibility of a constructor TRS with the polynomial path order $>_{\mathsf{pop}*}$ ([3]) induces polynomial innermost runtime complexity, whereas $\mathsf{f}(x) >_{\mathsf{pop}*} \mathsf{g}^n(x) >_{\mathsf{pop}*} \cdots >_{\mathsf{pop}*} \cdots >_{\mathsf{pop}*} \mathsf{g}^2(x) >_{\mathsf{pop}*} \mathsf{g}(x) >_{\mathsf{pop}*} x$ holds for all $n \in \mathbb{N}$, when precedence $\mathsf{f} > \mathsf{g}$ is used. Hence $\mathsf{dh}(t, >_{\mathsf{pop}*})$ is undefined, while the order $>_{\mathsf{pop}*}$ can still be employed in complexity analysis. Let R be a binary relation over terms, let $\succ$ be a proper order on terms, and let G denote a mapping associating a term with a natural number. Then $\succ$ is G-*collapsible on R* if $\mathsf{G}(s) > \mathsf{G}(t)$, whenever $s \mathrel{R} t$ and $s \succ t$ holds. An order $\succ$ is *collapsible (on R)*, if there is a mapping G such that $\succ$ is G-collapsible (on R).

Lemma 3. *Let R be a finitely branching and well-founded relation. Further, let $\succ$ be a G-collapsible order with $R \subseteq \succ$. Then $\mathsf{dh}(t, R) \leqslant \mathsf{G}(t)$ holds for all terms t.*

If a TRS $\mathcal{R}$ and a μ-monotone matrix interpretation $\mathcal{A}$ are compatible, $\mathsf{G}(t)$ can be given by $[t]_1$. In order to estimate derivational or runtime complexity, one needs to associate $[t]_1$ to $|t|$. For this sake we define degrees of matrix interpretations.

Definition 4. *A matrix interpretation is of* (basic) degree k *if there is a constant c such that $[t]_i \leqslant c \cdot |t|^k$ for all (basic) terms t and i, respectively.*

An *upper triangular complexity matrix* is a matrix M in $\mathbb{N}^{d \times d}$ such that we have $M_{j,k} = 0$ for all $1 \leqslant k < j \leqslant d$, and $M_{j,j} \leqslant 1$ for all $1 \leqslant j \leqslant d$. We say that a ($\mu$-)monotone well-founded algebra $\mathcal{A}$ is a *triangular matrix interpretation* (*TMI* for short) if $\mathcal{A}$ is a matrix interpretation (over $\mathbb{N}$) and all matrices employed are of upper triangular complexity form. The following result can be easily distilled from the literature, cf. [19, 20].

Theorem 5. *Let $\mathcal{A}$ be a TMI and let M denote the component-wise maximum of all matrices occurring in $\mathcal{A}$. Further, let k denote the number of ones occurring along the diagonal of M. Then, $\succ_{\mathcal{A}}$ is $\mathsf{O}(n^k)$-collapsible.*

In order to cope with runtime complexity, a similar idea to restricted polynomial interpretations (see [1]) can be integrated to triangular matrix interpretations. We call $\mathcal{A}$ a *restricted matrix interpretation* (*RMI* for short) if $\mathcal{A}$ is a matrix interpretation, but for each constructor symbol $f \in \mathcal{C}$, the interpretation $f_{\mathcal{A}}$ of f employs upper triangular complexity matrices, only. The following theorem is obtained from the combination of existing results [1, 2, 19].

Theorem 6. *Let $\mathcal{A}$ be an RMI and let t be a basic term. Further, let M denote the component-wise maximum of all matrices used for the interpretation of constructor symbols, and let k denote the number of ones occurring along the diagonal of M. Then $\mathcal{A}$ is of basic degree k. Furthermore, if M is the unit matrix then $\mathcal{A}$ is of basic degree* 1.

It is not difficult to see that Theorem 6 also holds for RMIs based on *lower* triangular complexity matrices. We refrain from given the formal details, but rather exemplify the definition below.

Example 7. Consider the TRS $\mathcal{R}_{sum}$[5]

$$\begin{array}{llll} 1: & \mathsf{sum}(0) \to 0 & 3: & \mathsf{sum1}(0) \to 0 \\ 2: & \mathsf{sum}(\mathsf{s}(x)) \to \mathsf{sum}(x) + \mathsf{s}(x) & 4: & \mathsf{sum1}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{sum1}(x) + (x + x)) \end{array}$$

[5] The TRS is Example 2.17 in Steinbach and Kühler's collection of TRSs [21].

where, sum and $\mathsf{sum1}$ are defined symbols, and $\mathsf{0}$, s, and $+$ are constructor symbols. Consider the 2-dimensional RMI $\mathcal{A}$ (based on lower triangular complexity matrices) with

$$\mathsf{0}_{\mathcal{A}} = \begin{pmatrix}0\\1\end{pmatrix} \qquad \mathsf{sum}_{\mathcal{A}}(\boldsymbol{x}) = \begin{pmatrix}1&2\\1&3\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}0\\1\end{pmatrix}$$

$$\mathsf{s}_{\mathcal{A}}(\boldsymbol{x}) = \begin{pmatrix}1&0\\1&1\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}2\\2\end{pmatrix} \qquad \mathsf{sum1}_{\mathcal{A}}(\boldsymbol{x}) = \begin{pmatrix}1&2\\0&3\end{pmatrix}\boldsymbol{x}$$

$$+_{\mathcal{A}}(\boldsymbol{x},\boldsymbol{y}) = \begin{pmatrix}1&0\\0&0\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}1&0\\0&0\end{pmatrix}\boldsymbol{y} \; .$$

The rules in $\mathcal{R}_{sum}$ are interpreted and ordered as follows.

$$1\colon \quad \begin{pmatrix}2\\4\end{pmatrix} > \begin{pmatrix}0\\1\end{pmatrix} \qquad 3\colon \quad \begin{pmatrix}2\\3\end{pmatrix} > \begin{pmatrix}0\\1\end{pmatrix}$$

$$2\colon \begin{pmatrix}3&2\\4&3\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}6\\9\end{pmatrix} > \begin{pmatrix}2&2\\0&0\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}2\\0\end{pmatrix} \quad 4\colon \begin{pmatrix}3&2\\3&3\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}6\\6\end{pmatrix} > \begin{pmatrix}3&2\\3&2\end{pmatrix}\boldsymbol{x} + \begin{pmatrix}2\\2\end{pmatrix} \; .$$

Therefore, $\mathcal{R}_{sum} \subseteq >_{\mathcal{A}}$ holds. By an application of Theorem 6 we conclude that the runtime complexity is *quadratic*. As we see later, there is a tighter bound.

3 Usable Replacement Maps

Unfortunately, there is no RMI compatible with the TRS of our running example (Example 1). The reason is that the monotonicity requirement of matrix interpretations is too severe for complexity analysis. Inspired by the idea of Fernández [22], we show how context-sensitive rewriting is used in complexity analysis. Here we briefly explain our idea. Let $\mathbf{n}$ denote the numeral $s^n(\mathsf{0})$. Consider the derivation from $\mathbf{4} \div \mathbf{2}$:

$$\underline{\mathbf{4} \div \mathbf{2}} \to \mathsf{s}(\underline{(\mathbf{3} - \mathbf{1})} \div \mathbf{2}) \to \mathsf{s}((\underline{\mathbf{2} - \mathsf{0}}) \div \mathbf{2}) \to \mathsf{s}(\underline{\mathbf{2} \div \mathbf{2}}) \to \cdots$$

where redexes are underlined. Observe that e.g. any second argument of $\div$ is never rewritten. More precisely, any derivation from a basic term consists of only μ-steps with the replacement map μ: $\mu(\mathsf{s}) = \mu(\div) = \{1\}$ and $\mu(-) = \varnothing$.

Recall that $\mathcal{P}\mathsf{os}_\mu(t)$ denotes the set of μ-replacing positions in t and $\overline{\mathcal{P}\mathsf{os}}_\mu(t) = \mathcal{P}\mathsf{os}(t) \setminus \mathcal{P}\mathsf{os}_\mu(t)$. Further, a term t is a *μ-replacing term* with respect to a TRS $\mathcal{R}$ if $p \in \overline{\mathcal{P}\mathsf{os}}_\mu(t)$ implies $t|_p \in \mathsf{NF}(\mathcal{R})$. The set of all μ-replacing terms is denoted by $\mathcal{T}(\mu)$. Below $[L](\to^*)$ denotes the set $\{t \mid s \to^* t$ for some $s \in L\}$.

The above observation is cast in the following definition, borrowed from [23]. Usable replacement maps satisfy a desired property for runtime complexity analysis, as detailed in this section.

Definition 8. *Let $\to$ denote a binary relation. A replacement map μ is called a* usable replacement map *with respect to $\to$ and the set of starting terms $\mathcal{T}_\mathsf{b}$ if $[\mathcal{T}_\mathsf{b}](\to^*) \subseteq \mathcal{T}(\mu)$.*

The main result of this section is the definition of suitable *approximations* of usable replacement maps. For that we adapt the cap-function ICAP suitably, cf. [24]. Let μ be a replacement map. Clearly the function μ is representable as set of ordered pairs (f, i). Below we often confuse the notation of μ as a function or as a set.

Definition 9. *Let $\mathcal{R}$ be a TRS and let μ be a replacement map. We define the operator $\Upsilon^{\mathcal{R}}$ as follows:*

$$\Upsilon^{\mathcal{R}}(\mu) := \{(f, i) \mid l \to C[f(r_1, \dots, r_n)] \in \mathcal{R} \text{ and } \mathsf{CAP}^l_\mu(r_i) \neq r_i\} \ .$$

Here $\mathsf{CAP}^s_\mu(t)$ is inductively defined on t as follows:

$$\mathsf{CAP}^s_\mu(t) = \begin{cases} t & \text{if } t = s|_p \text{ for some } p \in \overline{\mathcal{P}\mathsf{os}}_\mu(s) \ , \\ u & \text{if } t = f(t_1, \dots, t_n) \text{ and } u \text{ and } l \text{ unify for no } l \to r \in \mathcal{R} \ , \\ y & \text{otherwise} \ , \end{cases}$$

where, $u = f(\mathsf{CAP}^s_\mu(t_1), \dots, \mathsf{CAP}^s_\mu(t_n))$, y is a fresh variable, and we assume that $\mathcal{V}\mathsf{ar}(l) \cap \mathcal{V}\mathsf{ar}(u) = \varnothing$ holds.

We define the *approximated* innermost usable replacement map $\mu^{\mathcal{R}}_{\mathsf{i}}$ as follows $\mu^{\mathcal{R}}_{\mathsf{i}} := \Upsilon^{\mathcal{R}}(\varnothing)$ and let the *approximated* usable replacement map $\mu^{\mathcal{R}}_{\mathsf{f}}$ denote the least fixed point of $\Upsilon^{\mathcal{R}}$. The existence of $\Upsilon^{\mathcal{R}}$ follows from the monotonicity of $\Upsilon^{\mathcal{R}}$. If $\mathcal{R}$ is clear from context, we simple write μ_{i}, μ_{f}, and Υ, respectively. In the remainder of the section we establish that μ_{i} and μ_{f} constitute usable replacement maps for $\xrightarrow{\mathsf{i}}$ and $\to$ respectively. Suppose $s \in \mathcal{T}(\mu)$: observe that the function $\mathsf{CAP}^s_\mu(t)$ replaces a subterm u of t by a fresh variable if $u\sigma$ is a redex for some $s\sigma \in \mathcal{T}(\mu)$. This is exemplified below.

Example 10. Consider the TRS $\mathcal{R}_{div}$. Let $l \to r$ be rule 4, namely, $l = \mathsf{s}(x) \div \mathsf{s}(y)$ and $r = \mathsf{s}((x - y) \div \mathsf{s}(y))$. Suppose $\mu(f) = \varnothing$ for all functions f and let w and z be fresh variables. The next table summarises $\mathsf{CAP}^l_\mu(t)$ for each proper subterm t in r. To see the computation process, we also indicate the term u in Definition 9.

t	x	y	$x - y$	$\mathsf{s}(y)$	$(x - y) \div \mathsf{s}(y)$
$\mathsf{CAP}^l_\mu(t)$	x	y	w	$\mathsf{s}(y)$	z
u	$-$	$-$	$x - y$	$\mathsf{s}(y)$	$w \div \mathsf{s}(y)$

By underlining proper subterms t in r such that $\mathsf{CAP}^l_\mu(t) \neq t$, we have

$$\mathsf{s}(\underline{\underline{(x - y)} \div \mathsf{s}(y)})$$

which indicates $(\mathsf{s}, 1), (\div, 1) \in \Upsilon(\mu)$.

The next lemma clarifies the rôle played by the cap function $\mathsf{CAP}^s_\mu(t)$.

Lemma 11. *Let s and t be terms, and σ a substitution such that $s\sigma \in \mathcal{T}(\mu)$ and $\mathsf{CAP}^s_\mu(t) = t$. Then $t\sigma \in \mathsf{NF}(\mathcal{R})$.*

Proof. We use induction on t. Suppose $s\sigma \in \mathcal{T}(\mu)$ and $\mathsf{CAP}^s_\mu(t) = t$. If $t = s|_p$ for some $p \in \overline{\mathcal{P}\mathsf{os}}_\mu(s)$ then $t\sigma = (s\sigma)|_p \in \mathsf{NF}$ follows by definition of $\mathcal{T}(\mu)$.

We can assume that $t = f(t_1, \dots, t_n)$. Assume otherwise that $t = x \in \mathcal{V}$, then $\mathsf{CAP}^s_\mu(x) = x$ entails that $x\sigma$ occurs at a non-μ-replacing position in $s\sigma$. Hence $x\sigma \in \mathsf{NF}$ follows from $s\sigma \in \mathcal{T}(\mu)$. Moreover, by assumption we have:

1. $\mathsf{CAP}^s_\mu(t_i) = t_i$ for each i, and
2. there is no rule $l \to r \in \mathcal{R}$ such that t and l unify.

Due to 2) $l\sigma$ is not reducible at the root, and the induction hypothesis yields $t_i\sigma \in \mathsf{NF}$ because of 1). Therefore, we obtain $t\sigma \in \mathsf{NF}$. □

For a smooth inductive proof of the key lemma, Lemma 14, we develop an alternative characterisation of the set of μ-replacing terms $\mathcal{T}(\mu)$.

Definition 12. *The set* $\{(f, i) \mid f(t_1, \dots, t_n) \trianglelefteq t$ *and* $t_i \notin \mathsf{NF}(\mathcal{R})\}$ *is denoted by* $\upsilon(t)$.

The next lemma shows that the set of μ-replacing terms $\mathcal{T}(\mu)$ can be characterised through the above definition.

Lemma 13. $\mathcal{T}(\mu) = \{t \mid \upsilon(t) \subseteq \mu\}$.

Proof. For the inclusion from left to right, let $t \in \mathcal{T}(\mu)$ and let $(f, i) \in \upsilon(t)$. We show $(f, i) \in \mu$. By Definition 12 there is a position $p \in \mathcal{P}\mathsf{os}(t)$ with $t|_p = f(t_1, \dots, t_n)$ and $t|_{pi} \notin \mathsf{NF}$. Thus $pi \in \mathcal{P}\mathsf{os}_\mu(t)$ and $i \in \mathcal{P}\mathsf{os}_\mu(t|_p)$. Hence $(f, i) \in \mu$ is concluded.

Next we consider the opposite direction $\{t \mid \upsilon(t) \subseteq \mu\} \subseteq \mathcal{T}(\mu)$. Let t be a minimal counter-example such that $\upsilon(t) \subseteq \mu$ and $t \notin \mathcal{T}(\mu)$. One can write $t = f(t_1, \dots, t_n)$. Then, there exists a position $p \in \overline{\mathcal{P}\mathsf{os}}_\mu(t)$ such that $t|_p \notin \mathsf{NF}$. Because $\epsilon \notin \overline{\mathcal{P}\mathsf{os}}_\mu(t)$ by definition, $p = iq$ with $i \in \mathbb{N}$. As $iq \in \overline{\mathcal{P}\mathsf{os}}_\mu(t)$ one of $(f, i) \notin \mu$ or $q \in \overline{\mathcal{P}\mathsf{os}}_\mu(t|_i)$ must hold. Consider the first alternative. Then by Definition 12, $(f, i) \in \upsilon(t) \subseteq \mu$ and we obtain a contradiction. Now, consider the second alternative. Note that $t|_{iq} \notin \mathsf{NF}$ implies $t|_i \notin \mathsf{NF}$. In conjunction with $q \in \overline{\mathcal{P}\mathsf{os}}_\mu(t|_i)$ this yields that t_i is counter-example which is smaller than t. This contradicts the definition of t. □

The next lemma about the operator Υ is a key for the main theorem. Note that every subterm of a μ-replacing term is a μ-replacing term.

Lemma 14. *If* $l \to r \in \mathcal{R}$ *and* $l\sigma \in \mathcal{T}(\mu)$ *then* $r\sigma \in \mathcal{T}(\mu \cup \Upsilon(\mu))$.

Proof. Let $l \to r \in \mathcal{R}$ and suppose $l\sigma \in \mathcal{T}(\mu)$. By Lemma 13 we have

$$\mathcal{T}(\mu) = \{t \mid \upsilon(t) \subseteq \mu\} \qquad \mathcal{T}(\mu \cup \Upsilon(\mu)) = \{t \mid \upsilon(t) \subseteq \mu \cup \Upsilon(\mu)\} \ .$$

Hence it is sufficient to show $\upsilon(r\sigma) \subseteq \mu \cup \Upsilon(\mu)$. Let $(f, i) \in \upsilon(r\sigma)$. There is $p \in \mathcal{P}\mathsf{os}(r\sigma)$ with $r\sigma|_p = f(t_1, \dots, t_n)$ and $t_i \notin \mathsf{NF}$. If p is below some variable position of r, $r\sigma|_p$ is a subterm of $l\sigma$, and thus $\upsilon(r\sigma|_p) \subseteq \upsilon(l\sigma) \subseteq \mu$. Otherwise, p is a non-variable position of r. We may write $r|_p = f(r_1, \dots, r_n)$ and $r_i\sigma = t_i \notin \mathsf{NF}$. Due to Lemma 11 we obtain $\mathsf{CAP}^l_\mu(r_i) \neq r_i$. Therefore, $(f, i) \in \Upsilon(\mu)$. □

Lemma 15. *For the approximated usable replacement maps $\mathcal{T}(\mu_{\mathsf{i}})$ and $\mathcal{T}(\mu_{\mathsf{f}})$, the following implications hold:*

1. *If $s \in \mathcal{T}(\mu_{\mathsf{i}})$ and $s \xrightarrow{\mathrm{i}} t$ then $t \in \mathcal{T}(\mu_{\mathsf{i}})$.*
2. *If $s \in \mathcal{T}(\mu_{\mathsf{f}})$ and $s \to t$ then $t \in \mathcal{T}(\mu_{\mathsf{f}})$.*

Proof. We show property 1). Suppose $s \in \mathcal{T}(\mu_{\mathsf{i}})$ and $s \xrightarrow{\mathrm{i}} t$ is a rewrite step at p. Due to the definition of innermost rewriting, we have $s|_p \in \mathcal{T}(\varnothing)$. Hence, $t|_p \in \mathcal{T}(\mu_{\mathsf{i}})$ is obtained by Lemma 14. Because $s \in \mathcal{T}(\mu_{\mathsf{i}})$ we have $p \in \mathcal{P}\mathsf{os}_{\mu_{\mathsf{i}}}(s)$. Hence due to $t|_p \in \mathcal{T}(\mu_{\mathsf{i}})$ we conclude $t = s[t|_p]_p \in \mathcal{T}(\mu_{\mathsf{i}})$ due to the above remark. The proof of 2) proceeds along the same pattern and is left to the reader. $\square$

We arrive at the main result of this section.

Theorem 16. *The inclusions $[\mathcal{T}(\varnothing)](\xrightarrow{\mathrm{i}}{}^{*}_{\mathcal{R}}) \subseteq \mathcal{T}(\mu_{\mathsf{i}})$ and $[\mathcal{T}(\varnothing)](\to^{*}_{\mathcal{R}}) \subseteq \mathcal{T}(\mu_{\mathsf{f}})$ hold. In particular μ_{i} and μ_{f} constitute usable replacement maps for $\xrightarrow{\mathrm{i}}$ and $\to$, respectively.*

Proof. We focus on the second part of the theorem, where we have to prove that $t \in \mathcal{T}(\mu_{\mathsf{f}})$, whenever there exists $s \in \mathcal{T}(\varnothing)$ such that $s \to^{*}_{\mathcal{R}} t$. As $\mathcal{T}(\varnothing) \subseteq \mathcal{T}(\mu_{\mathsf{f}})$ this follows directly from Lemma 15.

Note that $\mathcal{T}(\varnothing)$ is the set of all argument normalised terms. Therefore, $\mathcal{T}_{\mathsf{b}} \subseteq \mathcal{T}(\varnothing)$. Hence the second half of the theorem follows. $\square$

Given a TRS $\mathcal{R}$ we write $\xrightarrow{\mu_{\mathsf{i}}}$ for the μ_{i}-step relation of $\mathcal{R}$, and $\xrightarrow{\mu_{\mathsf{f}}}$ for the μ_{f}-step relation. The following corollary to Theorem 16 is immediate.

Corollary 17. *We have $\mathsf{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leqslant \mathsf{dh}(t, \xrightarrow{\mu_{\mathsf{i}}})$ and $\mathsf{dh}(t, \to_{\mathcal{R}}) = \mathsf{dh}(t, \xrightarrow{\mu_{\mathsf{f}}})$ for all terminating terms $t \in \mathcal{T}_{\mathsf{b}}$.*

An advantage of the use of context-sensitive rewriting is that the compatibility requirement of monotone algebra in termination or complexity analysis is relaxed to μ-monotone algebra. We illustrate its use in the next examples.

Example 18. Recall the TRS $\mathcal{R}_{div}$ given in Example 1 above. The usable replacement maps are as follows:

$$\mu_{\mathsf{i}}(-) = \varnothing \quad \mu_{\mathsf{i}}(\mathsf{s}) = \mu_{\mathsf{i}}(\div) = \{1\} \qquad \mu_{\mathsf{f}}(\mathsf{s}) = \mu_{\mathsf{f}}(-) = \mu_{\mathsf{f}}(\div) = \{1\} \ .$$

Consider the 1-dimensional RMI $\mathcal{A}$ (i.e. linear polynomial interpretations) with $\mathsf{0}_{\mathcal{A}} = 1$, $\mathsf{s}_{\mathcal{A}}(x) = x + 2$, $-_{\mathcal{A}}(x, y) = x + 1$, and $\div_{\mathcal{A}}(x, y) = 3x$; $\mathcal{A}$ is strictly μ_{i}-monotone and μ_{f}-monotone. The rules in $\mathcal{R}_{div}$ are interpreted and ordered as follows.

$$\begin{array}{rlrl} 1\colon & x + 1 > x & 3\colon & 3 > 1 \\ 2\colon & x + 3 > x + 1 & 4\colon & 3x + 6 > 3x + 5 \ . \end{array}$$

Therefore, $\mathcal{R}_{div} \subseteq >_{\mathcal{A}}$ holds. Applying Theorem 6 in the context of usable replacement maps, we conclude that the (innermost) runtime complexity is *linear*, which is optimal.

Example 19. Recall the TRS $\mathcal{R}_{sum}$ of Example 7. The usable replacement map for full rewriting is as follows:

$$\mu_{\mathsf{f}}(\mathsf{sum}) = \mu_{\mathsf{f}}(\mathsf{sum1}) = \varnothing \qquad\qquad \mu_{\mathsf{f}}(\mathsf{s}) = \mu_{\mathsf{f}}(+) = \{1\} \ .$$

Consider the 1-dimensional RMI $\mathcal{A}$ with

$$\mathsf{0}_{\mathcal{A}} = 2 \quad \mathsf{s}_{\mathcal{A}}(x) = x + 2 \quad +_{\mathcal{A}}(x, y) = x \quad \mathsf{sum}_{\mathcal{A}}(x) = 2x \quad \mathsf{sum1}(x, y) = 2x \ .$$

which is strictly μ_{f}-monotone. The rules in $\mathcal{R}_{div}$ are interpreted and ordered as follows.

$$\begin{array}{llll} 1\colon & 4 > 2 & 3\colon & 4 > 2 \\ 2\colon & 2x + 4 > 2x & 4\colon & 2x + 4 > 2x + 2 \ . \end{array}$$

Therefore, $\mathcal{R}_{div} \subseteq >_{\mathcal{A}}$ holds. By an application of Theorem 6 we conclude that the runtime complexity is *linear*, which is optimal.

We cast the observations in the example into another corollary to Theorem 16.

Corollary 20. *Let $\mathcal{R}$ be a TRS and let $\mathcal{A}$ be a d-degree μ_{i}-monotone (or μ_{f}-monotone) RMI compatible with $\mathcal{R}$. Then the (innermost) runtime complexity function $\mathsf{rc}^{(\mathsf{i})}_{\mathcal{R}}$ with respect to $\mathcal{R}$ is bounded by a d-degree polynomial.*

Proof. It suffices to consider the case for full rewriting. Let s, t be terms such that $s \to_{\mathcal{R}} t$. By the theorem, we have $s \xrightarrow{\mu_{\mathsf{f}}} t$. Furthermore, by assumption $\mathcal{R} \subseteq \succ_{\mathcal{A}}$ and for any $f \in \mathcal{F}$, $f_{\mathcal{A}}$ is strictly monotone on all $\mu_{\mathsf{f}}(f)$. Thus $s \succ_{\mathcal{A}} t$ follows. Finally, the corollary follows by application of Theorem 6. ☐

4 Experiments

The usable replacement map method has been incorporated into the *Tyrolean Complexity Tool* $\mathsf{T_CT}$ [25]. We note that the established method can easily combined with existing modular frameworks and the implementation in $\mathsf{T_CT}$ makes (essential) use of this. In this section we present an experimental evaluation of the technique based on version 8.0.6 of the *Termination Problems Database* (*TPDB* for short). We consider TRSs without theory annotation, where the runtime complexity analysis is non-trivial, that is the set of basic terms is infinite. This testbed comprises 1249 TRSs.

All experiments were conducted on a machine that is identical to the official competition server (8 AMD Opteron® 885 dual-core processors with 2.8GHz, 8x8 GB memory). As timeout we use 60 seconds. The complete experimental data can be found at `http://cl-informatik.uibk.ac.at/software/tct/experiments/rtatlca14`, where also the testbed employed is detailed.

Table 1 summarises the experimental results of the use of usable replacement maps for full and innermost runtime complexity analysis. The tests employ one-

	full		*innermost*	
result	RMI(1-3),(−)	RMI(1-3), (+)	RMI(1-3), (−)	RMI(1-3), (+)
$\mathsf{O}(n)$	103	134	104	140
$\mathsf{O}(n^2)$	174	209	174	226
$\mathsf{O}(n^k)$	183	225	183	247
timeout (60s)	113	109	113	117

Table 1. Experimental results I (one- to three-dimensional RMIs)

to three-dimensional RMIs.[6] The tests clearly indicate the power of the established technique. This power is not only in the absolute number of examples, but more importantly in the precision of the analysis. We note that our tests only make use of the simplest notion of RMIs for runtime complexity analysis cf. [26]. This is a rather mundane method, more sophisticated methods have been reported in [2, 19]. However, to assess the power of the established technique this restriction is insignificant.

In Table 2 we present the overall power obtained for the automated runtime complexity analysis. Here we test the current version of $\mathsf{T_CT}$ using a strategy that avoids the new method, in contrast to its standard strategy. It is to be expected that the effect of the proposed technique is smaller than in Table 1. This is due to the presence of transformation techniques, like the *weak dependency pair method* [4] or the *dependency tuple method* [5] and the use of a modular framework [7]. While the usable argument method is still effective for weak dependency pairs, as it may lighten the *weight gap constraint*, the dependency tuple method allows to remove all monotonicity constraints. Note that the dependency tuple method is only applicable for innermost runtime complexity.

Despite these theoretical facts, the method has a significant impact on full and innermost rewriting. While the overall effect is a lot smaller than for direct techniques, we emphasise that the method allows to win some examples that can be handled with linear rather than with quadratic complexity. The alert reader may wonder, why there is a positive effect at all in the innermost case, as transformation techniques do not require monotonicity. This is due to the case that even for innermost runtime complexity analysis, direct methods are currently not superseded by transformational techniques.

5 Related Work

Usable replacement maps Usable replacement maps were originally introduced by Fernández [22] for proving termination of *innermost rewriting*. This notion

[6] Note that matrix interpretations, that is the test "RMI(1-3)", cannot discern between innermost versus full rewriting. Hence the practical differences noted in the table are coincidental.

	full		*innermost*	
result	$\mathsf{T_CT}$ (−)	$\mathsf{T_CT}$ (+)	$\mathsf{T_CT}$ (−)	$\mathsf{T_CT}$ (+)
$\mathsf{O}(1)$	92	100	326	325
$\mathsf{O}(n)$	408	421	500	508
$\mathsf{O}(n^2)$	423	428	554	555
$\mathsf{O}(n^3)$	424	429	564	565
$\mathsf{O}(n^k)$	426	431	568	569
timeout (60s)	714	711	617	615

Table 2. Experimental results II (overall effect)

is already applicable for analysing innermost runtime complexity. More precisely, we link Theorem 16 to Fernández' work. In [22] an application of context-sensitive rewriting for innermost termination has been established.

Proposition 21 ([22])**.** *A TRS $\mathcal{R}$ is innermost terminating if $\xrightarrow{\mu_\mathsf{i}}$ is terminating.*

Proof. We show the contraposition. If $\mathcal{R}$ is not innermost terminating, there is an infinite sequence $t_0 \xrightarrow{\mathsf{i}} t_1 \xrightarrow{\mathsf{i}} t_2 \xrightarrow{\mathsf{i}} \cdots$, where $t_0 \in \mathcal{T}(\varnothing)$. From Theorem 16 and Lemma 15 we obtain $t_0 \xrightarrow{\mu_\mathsf{i}} t_1 \xrightarrow{\mu_\mathsf{i}} t_2 \xrightarrow{\mu_\mathsf{i}} \cdots$. Hence, $\xrightarrow{\mu_\mathsf{i}}$ is not terminating. □

We lifted Fernández' notion to full rewriting, exploiting the cap function ICAP in [24]. Realisation of a fixed point calculation of usable replacement maps is considered as a primary result of this paper. Note that Proposition 21 does not generalise to full termination, even if one replaces the innermost replacement map μ_i, by the replacement map μ_f.

Example 22. Consider the famous Toyama's example $\mathcal{R}$

$$\mathsf{f}(\mathsf{a},\mathsf{b},x) \to \mathsf{f}(x,x,x) \qquad \mathsf{g}(x,y) \to x \qquad \mathsf{g}(x,y) \to y \ .$$

The replacement map μ_f is empty. Thus, the algebra $\mathcal{A}$ over $\mathbb{N}$

$$\mathsf{f}_\mathcal{A}(x,y,z) = \max\{x-y,0\} \qquad \mathsf{g}_\mathcal{A}(x,y) = x+y+1 \qquad \mathsf{a}_\mathcal{A} = 1 \qquad \mathsf{b}_\mathcal{A} = 0 \ .$$

is μ_f-monotone and we have $\mathcal{R} \subseteq >_\mathcal{A}$. However, we should not conclude termination of $\mathcal{R}$, because $\mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}(\mathsf{a},\mathsf{b}))$ is non-terminating.

Cap functions In [22] usable replacement maps for innermost rewriting are defined as $\{(f,i) \mid l \to C[f(r_1,\ldots,r_n)] \in \mathcal{R}$ and $r_i|_p \ntriangleleft l$, for some $p \in \mathcal{P}\mathsf{os}_\mathcal{D}(r)\}$. This and our definition μ_i coincide if the following cap function is used during the computation of $\Upsilon^\mathcal{R}$:

$$\mathsf{CAP}^s_\mu(t) = \begin{cases} t & \text{if } t \lhd s \ , \\ u & \text{if } t = f(t_1,\ldots,t_n) \text{ and } f \in \mathcal{C} \ , \\ y & \text{otherwise} \ , \end{cases}$$

where, $u = f(\mathsf{CAP}^s_\mu(t_1), \ldots, \mathsf{CAP}^s_\mu(t_n))$ and y is a fresh variable. In the light of this reformulation one can (easily) verify that the usable replacement map μ_{i} in Section 3 is always a subset of the above set.

There exists a cap function for context-sensitive rewriting, introduced by Alarcón et al. [27] for termination analysis. Their definition is the following:

$$\mathsf{CAP}^s_\mu(t) = \begin{cases} t & \text{if } t \text{ is a variable },\\ u & \text{if } t = f(t_1, \ldots, t_n) \text{ and } u \text{ and } l \text{ unify for no } l \to r \in \mathcal{R} ,\\ y & \text{otherwise },\end{cases}$$

where, y is a fresh variable and $u = f(u_1, \ldots, u_n)$. Each u_i stands for $\mathsf{CAP}^s_\mu(t_i)$ if $i \in \mu(f)$, and t_i otherwise. This definition cannot be used for calculation of usable replacement maps: It is designed for exploiting a given replacement map μ to *ignore* potential rewrite positions, while our cap function is aimed at *detecting* potential reducible positions to build a usable replacement map.

Dependency pairs Weak dependency pairs [4] and dependency tuples [5] are transformational approaches that split a rewrite relation into two relations. This split allows us to weaken the monotonicity condition. Although these approaches exploit dependencies of defined symbols, they do not analyse how variables in rewrite rules or dependency pairs (tuples) are instantiated in rewriting.[7] As a consequence, the transformations do not resolve the problem of variable duplication addressed in the introduction. We emphasise that usable replacement maps and dependency pairs (tuples) are complementary and the combination is beneficial, as seen in Section 4. A similar observation holds for techniques employing modularity.

6 Conclusion

In this paper we have defined the notion of usable replacement maps. It is a straightforward observation that only usable arguments need to be considered for monotonicity conditions. In a nutshell, we have shown how monotonicity conditions for orders can be weakened by using the notion of context-sensitive rewriting.

The presented technique is very easy to implement and has been integrated in the Tyrolean Complexity Tool. Above we have provided ample experimental data for assessing the viability of our method. The positive experimental evaluation, even in the innermost case, is somewhat surprising. One might have assumed that transformation techniques, as for example the dependency tuple method introduced in [5] supersede direct methods and thus refrain us from concerns about monotonicity. Our experiments clearly show that this is not the case. We emphasise that the here proposed method directly extends to modular frameworks and our implementation in TCT makes essential use of this fact.

[7] Approximation techniques for dependency graphs may be considered an exception.

```
minus : (nat, nat) -> nat
minus (m, n) = match m with
               | 0    -> 0
               | S m' -> match n with
                         | 0    -> m
                         | S n' -> minus (m', n');

quot : (nat, nat) -> nat
quot (m, n) = match m with
              | 0    -> 0
              | S m' -> match n with
                        | 0    -> 0
                        | S n' ->
                            (quot (minus (m', n'), n)) + 1;
```

Fig. 1. Division in RaML

Apart from its practical value the proposed technique allows to incorporate features of complexity analysis of functional programs into rewrite systems. We consider a reformulation of our motivating example in an ML-like language, cf. Figure 1. This functional program is subject to the analysis of the RaML-prototype, developed by Hoffmann et al. [28]. The prototype is based on an amortised resource analysis that employs a potential-based type system. Application of the method on the example yields the optimal linear bound on the innermost runtime complexity. Inspection of the complexity proof reveals that the method assigns zero potential to the second argument of minus and div, which is related to the fact that these arguments can be safely ignored in our setting, cf. Example 18. However, the potential-based method depends on the presence of types as detailed in [29]. We emphasise that the usable arguments method allows a similar fine-grained control for the runtime complexity analysis, even without the introduction of types.

Acknowledgments. We would like to thank the anonymous reviewers for their valuable comments that greatly helped in improving the presentation.

References

1. Bonfante, G., Cichon, A., Marion, J.Y., Touzet, H.: Algorithms with polynomial interpretation termination proof. JFP **11**(1) (2001) 33–53
2. Middeldorp, A., Moser, G., Neurauter, F., Waldmann, J., Zankl, H.: Joint spectral radius theory for automated complexity analysis of rewrite systems. In: Proc. 4th CAI. Volume 6742 of LNCS. (2011) 1–20
3. Avanzini, M., Moser, G.: Polynomial path orders. LMCS **9**(4) (2013)
4. Hirokawa, N., Moser, G.: Automated complexity analysis based on the dependency pair method. In: Proc. 4th IJCAR. Number 5195 in LNAI (2008) 364–380
5. Noschinski, L., Emmes, F., Giesl, J.: Analyzing innermost runtime complexity of term rewriting by dependency pairs. JAR **51**(1) (2013) 27–56

6. Zankl, H., Korp, M.: Modular complexity analysis via relative complexity. LMCS **10**(1:19) (2014) 1–33
7. Avanzini, M., Moser, G.: A combination framework for complexity. In: Proc. 24th RTA. Volume 21 of LIPIcs. (2013) 55–70
8. Moser, G.: Proof Theory at Work: Complexity Analysis of Term Rewrite Systems. CoRR **abs/0907.5527** (2009) Habilitation Thesis.
9. Baillot, P., Marion, J.Y., Rocca, S.R.D.: Guest editorial: Special issue on implicit computational complexity. TOCL **10**(4) (2009)
10. Avanzini, M., Moser, G.: Closing the gap between runtime complexity and polytime computability. In: Proc. 21st RTA. Volume 6 of LIPIcs. (2010) 33–48
11. Arts, T., Giesl, J.: A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen (2001)
12. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Proc. 3rd RTA. Number 355 in LNCS (1989) 167–177
13. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
14. Choppy, C., Kaplan, S., Soria, M.: Complexity analysis of term-rewriting systems. TCS **67**(2–3) (1989) 261–282
15. TeReSe: Term Rewriting Systems. Volume 55 of Cambridge Tracks in Theoretical Computer Science. Cambridge University Press (2003)
16. Lucas, S.: Context-sensitive rewriting strategies. IC **178**(1) (2002) 294–343
17. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. JAR **40**(3) (2008) 195–220
18. Hofbauer, D., Waldmann, J.: Termination of string rewriting with matrix interpretations. In: Proc. 17th RTA. Volume 4098 of LNCS. (2006) 328–342
19. Neurauter, F., Zankl, H., Middeldorp, A.: Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In: Proc. the 17th LPAR. Volume 6397 of LNCS (ARCoSS). (2010) 550–564
20. Waldmann, J.: Polynomially bounded matrix interpretations. In: Proc. 21st RTA. Volume 6 of LIPIcs. (2010) 357–372
21. Steinbach, J., Kühler, U.: Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern (1990)
22. Fernández, M.L.: Relaxing monotonicity for innermost termination. Information Processing Letters **93**(1) (2005) 117–123
23. Avanzini, M.: Verifying Polytime Computability Automatically. PhD thesis, University of Innsbruck (2013)
24. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Proc. 5th FROCOS. Volume 3717 of LNAI. (2005) 216–231
25. Avanzini, M., Moser, G.: Tyrolean Complexity Tool: Features and usage. In: Proc. 24th RTA. Volume 21 of LIPIcs. (2013) 71–80
26. Moser, G., Schnabl, A., Waldmann, J.: Complexity analysis of term rewriting based on matrix and context dependent interpretations. In: Proc. 28th FSTTCS. Volume 2 of LIPIcs. (2008) 304–315
27. Alarcón, B., Gutiérrez, R., Lucas, S.: Context-sensitive dependency pairs. IC **208**(8) (2010) 922–968
28. Hoffmann, J., Aehlig, K., Hofmann, M.: Resource aware ML. In: Proc. 24th CAV. Volume 7358 of LNCS. (2012) 781–786
29. Hofmann, M., Moser, G.: Amortised resource analysis and typed polynomial interpretations. In: Proc. of 25th RTA & 12th TLCA. LNCS (ARCoSS) (2014) to appear.