

On Implementing Modular Complexity Analysis

Harald Zankl and Martin Korp*

Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria

Abstract

We recall the recent approach by (Zankl and Korp, 2010) to prove upper bounds on the (derivational) complexity of term rewrite systems modularly. In this note we show that this approach is suitable to tighten bounds after they have been established. The idea is to replace proof steps with a large bound by (new) proofs that yield smaller bounds. An evaluation of the approach shows the benefits.

1 Introduction

Term rewriting is a Turing-complete model of computation. The objects are modeled as terms and computation is mimicked by reduction steps. For terminating rewrite systems Hofbauer and Lautemann [4] consider the length of derivations as a measurement for the complexity of rewrite systems. The resulting notion of *derivational complexity* relates the length of a rewrite sequence to the size of its starting term. Thereby it is, e.g., a suitable metric for the complexity of deciding the word problem for a given confluent and terminating rewrite system (since the decision procedure rewrites terms to normal form). To show (feasible) upper bounds on the derivational complexity of a rewrite system currently few techniques are known. Match-bounds [2] and arctic matrix interpretations [5] induce linear upper bounds on the derivational complexity and triangular matrix interpretations [7] imply at most polynomially long derivations (the dimension of the matrices yields the degree of the polynomial). Recently in [10] a new approach has been introduced that allows to prove (upper) bounds on the (derivational) complexity of rewrite systems modularly. The idea is based on relative rewriting. In this note we present a method that enables this modular setting to infer tighter bounds.

2 Modular Complexity Analysis

We assume familiarity with (relative) term rewriting [1, 3, 9]. Let \mathcal{F} be a signature and \mathcal{V} a disjoint set of variables. By $\mathcal{T}(\mathcal{F}, \mathcal{V})$ we denote the set of terms over \mathcal{F} and \mathcal{V} . The size of a term t is denoted $|t|$. A *rewrite rule* is a pair of terms (l, r) , written $l \rightarrow r$, such that l is not a variable and all variables in r are contained in l . A *term rewrite system* (TRS for short) is a set of rewrite rules. For complexity analysis we assume TRSs to be finite and terminating. A TRS \mathcal{R} is said to be *duplicating* if there exists a rewrite rule $l \rightarrow r \in \mathcal{R}$ and a variable x that occurs more often in r than in l . A *rewrite relation* is a binary relation on terms that is closed under contexts and substitutions. For a TRS \mathcal{R} we define $\rightarrow_{\mathcal{R}}$ to be the smallest rewrite relation that contains \mathcal{R} . As usual \rightarrow^* denotes the reflexive and transitive closure of \rightarrow and \rightarrow^n the n -th iterate of \rightarrow . A *relative TRS* \mathcal{R}/\mathcal{S} is a pair of TRSs \mathcal{R} and \mathcal{S} with the induced rewrite relation $\rightarrow_{\mathcal{R}/\mathcal{S}} = \rightarrow_{\mathcal{S}}^* \cdot \rightarrow_{\mathcal{R}} \cdot \rightarrow_{\mathcal{S}}^*$. In the sequel we will sometimes identify a TRS \mathcal{R} with the relative TRS \mathcal{R}/\emptyset and vice versa.

The *derivation height* of a term t with respect to a relation \rightarrow is defined as follows: $\text{dh}(t, \rightarrow) = \sup \{m \mid \exists u \ t \rightarrow^m u\}$. The *derivational complexity* of a relation \rightarrow computes the

* This research is supported by FWF (Austrian Science Fund) project P22467.

maximal derivation height of all terms up to size n and is defined as $\text{dc}(n, \rightarrow) = \sup \{\text{dh}(t, \rightarrow) \mid t \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \text{ and } |t| \leq n\}$. Sometimes we say that a TRS \mathcal{R} (relative TRS \mathcal{R}/\mathcal{S}) has linear, quadratic, etc. or polynomial derivational complexity if $\text{dc}(n, \rightarrow_{\mathcal{R}})$ ($\text{dc}(n, \rightarrow_{\mathcal{R}/\mathcal{S}}$) can be bounded by a linear, quadratic, etc. function or polynomial in n . For functions $f, g: \mathbb{N} \rightarrow \mathbb{N}$ we write $f(n) = \mathcal{O}(g(n))$ if there are constants $M, N \in \mathbb{N}$ such that $f(n) \leq M \cdot g(n) + N$ for all $n \in \mathbb{N}$. Finally, we use **zero** to denote the constant zero function, i.e., $\text{zero}: \mathbb{N} \rightarrow \mathbb{N}$ with $\text{zero}(n) = 0$.

Next we recall the recent approach for modular complexity analysis via relative rewriting [10].

Definition 1. A *complexity pair* (\succ, \succeq) consists of two finitely branching rewrite relations \succ and \succeq that are *compatible*, i.e., $\succeq \cdot \succ \subseteq \succ$ and $\succ \cdot \succeq \subseteq \succ$. We call a relative TRS \mathcal{R}/\mathcal{S} *compatible* with a complexity pair (\succ, \succeq) if $\mathcal{R} \subseteq \succ$ and $\mathcal{S} \subseteq \succeq$.

The next lemma states that we can use complexity pairs to measure derivational complexity.

Lemma 2. *Let \mathcal{R}/\mathcal{S} be a relative TRS compatible with a complexity pair (\succ, \succeq) . Then for any term t we have $\text{dh}(t, \succ) \geq \text{dh}(t, \rightarrow_{\mathcal{R}/\mathcal{S}})$.*

Because we are especially interested in feasible upper bounds we state the next corollary.

Corollary 3. *Let \mathcal{R}/\mathcal{S} be a terminating relative TRS compatible with a complexity pair (\succ, \succeq) . If the derivational complexity of \succ is linear, quadratic, etc. or polynomial then the derivational complexity of \mathcal{R}/\mathcal{S} is linear, quadratic, etc. or polynomial.*

This corollary allows to investigate the complexity of (compatible) complexity pairs instead of the complexity of the underlying relative TRS. The next theorem is the key observation from [10].

Theorem 4. *Let $(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{S}$ be a terminating relative TRS. Then the equality*

$$\text{dc}(n, \rightarrow_{(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{S}}) = \mathcal{O}(\text{dc}(n, \rightarrow_{\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})}) + \text{dc}(n, \rightarrow_{\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})}))$$

holds.

Theorem 4 allows to split a relative TRS $(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{S}$ into *smaller* components $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ and $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$ and evaluate the complexities of these components (e.g., by different complexity pairs) independently. Note that this approach is not restricted to relative rewriting. To estimate the complexity of a (non-relative) TRS \mathcal{R} just consider the relative TRS \mathcal{R}/\emptyset . Proofs in the modular setting can be viewed as trees, as shown in the next example.

Example 5. Consider the complexity proof in Figure 1b on page 46, for the abstract TRS $\mathcal{R} = \{1, 2, 3, 4, 5\}$ consisting of five rewrite rules. The root node of the tree is the TRS of interest and the other nodes are relative rewrite systems and represent intermediate complexity problems. The edges indicate the (derivational) complexity of the proof steps. It is possible to apply Theorem 4 explicitly to split a problem into two (or more) problems as demonstrated in the second node. Such situations do not effect the complexity of the given problem which justifies the labels $\mathcal{O}(1)$. The remaining proof steps rely on an implicit application of Theorem 4 and measure the complexity of the rewrite rules that are moved from the first into the second component (relative to the remaining rules). For instance in the proof shown in Figure 1b there is an edge from $\{1, 3, 5\}/\{2, 4\}$ to $\{1\}/\{2, 3, 4, 5\}$ labeled $\mathcal{O}(n^3)$, stating that the (derivational) complexity of $\{3, 5\}/\{1, 2, 4\}$ is at most cubic. This step is sound because Theorem 4 states that

computing an upper bound on $\{1\}/\{2, 3, 4, 5\}$ and $\{3, 5\}/\{1, 2, 4\}$ suffices to get an upper bound on $\{1, 3, 5\}/\{2, 4\}$. Since the leaves in the tree give rise to constant complexity, the complexity of the original problem can be overestimated by summing up the complexities annotated to the edges; yielding a cubic upper bound for the proof in Figure 1b.

In the next section we revisit this example and show how existing bounds can be tightened.

3 Implementation

First we present the procedure for obtaining *some* complexity proof. Afterwards Section 3.2 is concerned with tightening the bounds starting from an existing complexity proof.

3.1 Establishing Bounds

To estimate the complexity of a TRS \mathcal{R} we first transform \mathcal{R} into the relative TRS \mathcal{R}/\emptyset . Obviously $\text{dc}(n, \rightarrow_{\mathcal{R}}) = \text{dc}(n, \rightarrow_{\mathcal{R}/\emptyset})$. If the input already is a relative TRS this step is omitted. Afterwards for a relative TRS \mathcal{R}/\mathcal{S} , we try to establish a bound on the complexity of $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ and continue with the relative TRS $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$. Here \mathcal{R}_1 and \mathcal{R}_2 form a decomposition of \mathcal{R} . This process is repeated until the remaining problem equals $\emptyset/(\mathcal{R} \cup \mathcal{S})$. Finally the complexity of \mathcal{R}/\mathcal{S} is obtained by summing up all intermediate bounds. In order to establish a maximal number of complexity proofs we run all techniques that can be derived from Corollary 3 (cf. also Theorem 7) in parallel and the first technique that can shift some rules is used to achieve progress.

Note that the procedure sketched above contains an implicit application of Theorem 4, i.e., some method immediately proves a bound for $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ and leaves $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$ as open proof obligation. In contrast to an explicit application of Theorem 4, here the method that establishes the bound on $\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S})$ can select the decomposition of \mathcal{R} into \mathcal{R}_1 and \mathcal{R}_2 which is beneficial for performance. As an immediate consequence, proof trees degenerate to lists (see Figure 1a on page 46).

In the following we describe the presented approach more formal and refer to it as the *complexity framework*. In this context we call a relative TRS \mathcal{R}/\mathcal{S} *complexity problem* (CP problem for short). To operate on CP problems so called *complexity processors* are used.

Definition 6. A *complexity processor* (CP processor for short) is a function that takes a CP problem \mathcal{R}/\mathcal{S} as input and returns a set of pairs $\bigcup_{1 \leq i \leq m} \{(\mathcal{R}_i/\mathcal{S}_i, f_i)\}$ as output. Here $\mathcal{R}_i/\mathcal{S}_i$ is a complexity problem and $f_i: \mathbb{N} \rightarrow \mathbb{N}$ for each $1 \leq i \leq m$. A complexity processor is *sound* if the equality $\text{dc}(n, \mathcal{R}/\mathcal{S}) = \mathcal{O}(f_1(n) + \dots + f_m(n) + \text{dc}(n, \rightarrow_{\mathcal{R}_1/\mathcal{S}_1}) + \dots + \text{dc}(n, \rightarrow_{\mathcal{R}_m/\mathcal{S}_m}))$ holds.

Next we list some CP processors. The first one is based on complexity pairs. In [10] powerful methods are presented that yield complexity pairs and allow to implement this processor.

Theorem 7. *The CP processor that maps the CP problem \mathcal{R}/\mathcal{S} to $\{(\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S}), f)\}$ if $\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S})$ is compatible with a complexity pair (\succ, \succeq) and to $\{(\mathcal{R}/\mathcal{S}, \text{zero})\}$ otherwise is sound. Here $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ and $f(n) = \text{dc}(n, \succ)$.*

The next CP processor is not implemented explicitly (cf. the discussion at the beginning of the section) but very suitable to tighten existing bounds (see Section 3.2).

Theorem 8. *The CP processor $(\mathcal{R}_1 \cup \mathcal{R}_2)/\mathcal{S} \mapsto \{(\mathcal{R}_1/(\mathcal{R}_2 \cup \mathcal{S}), \text{zero}), (\mathcal{R}_2/(\mathcal{R}_1 \cup \mathcal{S}), \text{zero})\}$ is sound.*

Further CP processors are presented in [10]. They are not essential for our purposes here. Finally, the main theorem states that the complexity framework is suitable for complexity analysis. We say that P is a complexity proof for a relative TRS \mathcal{R}/\mathcal{S} if all leaves in the proof are of the shape $\emptyset/(\mathcal{R} \cup \mathcal{S})$.

Theorem 9. *Let \mathcal{R}/\mathcal{S} be a relative TRS, P a complexity proof for \mathcal{R}/\mathcal{S} , and f_1, \dots, f_m the complexities occurring in P . If all CP processors in P are sound then $\text{dc}(n, \rightarrow_{\mathcal{R}/\mathcal{S}}) = \mathcal{O}(f_1(n) + \dots + f_m(n))$.*

3.2 Tightening Bounds

In contrast to termination, which is a plain YES/NO question, complexity corresponds to an optimization problem. Hence the tools should try to establish as tight bounds as possible. In the direct setting (where all base methods are used stand-alone) all methods can be executed in parallel and after a fixed amount of time the tightest bound is reported. In the modular setting this simple idea does not work because two problems emerge. The first problem is that the tool does not know how much time it may spend on a single proof step. If it spends too much then it may not finish the proof within the global time limit and if it spends too little then it can miss a low bound. The second problem is that in the modular setting (according to Theorem 4) separate criteria may make statements about the complexity of different rules. The question is then to identify the *better* bound. The following idea overcomes both problems: First we establish *some* complexity proof according to the procedure described at the beginning of Section 3.1 to obtain a bound for as many systems as possible. Afterwards we *optimize* this bound. The next example shows how the latter works.

Example 10. In Figure 1, three complexity proofs for the TRS \mathcal{R} of Example 5 are given. We show how the proof in Figure 1b can be optimized to the one in Figure 1c on an abstract level (note that a similar reasoning has been applied to obtain the proof shown in Figure 1b). In Figure 1b one part in the proof, highlighted by a solid box, is overestimated by a cubic upper bound. Hence the complexity of the whole system is at most cubic. We remark that this proof step estimates the complexity of $\{3, 5\}/\{1, 2, 4\}$. Now assume that the cubic bound is not optimal, i.e., there exists a proof (that may be longer and harder to find) that induces a quadratic upper bound on the complexity of $\{3, 5\}/\{1, 2, 4\}$. Then the proof can be optimized by splitting $\{1, 3, 5\}/\{2, 4\}$ into the problems $\{1\}/\{2, 3, 4, 5\}$ and $\{3, 5\}/\{1, 2, 4\}$ as illustrated in Figure 1c. After that, the proof of $\{1\}/\{2, 3, 4, 5\}$ is reused in the optimized proof (cf. the dashed boxes in Figure 1b and Figure 1c) whereas the original proof of $\{3, 5\}/\{1, 2, 4\}$ is replaced by the new one, as indicated by the solid box in Figure 1c. Now, the proof in Figure 1c establishes a quadratic upper bound on the complexity of \mathcal{R} .

As the previous example demonstrates the basic idea is to replace single proof steps by new proofs that induce tighter bounds. This procedure is repeated until either the global time limit is reached or none of the bounds can be tightened further. Note that the transformation is sound by Theorem 9.

4 Experimental Results

The technique described in the preceding section is implemented in the complexity analyzer \mathcal{GT} [10] which is developed by the authors. Below we report on the experiments we performed on the 1172 non-duplicating TRSs in version 7.0.2 of the termination problems database without strategy or theory annotation. Note that duplicating systems induce at least exponential

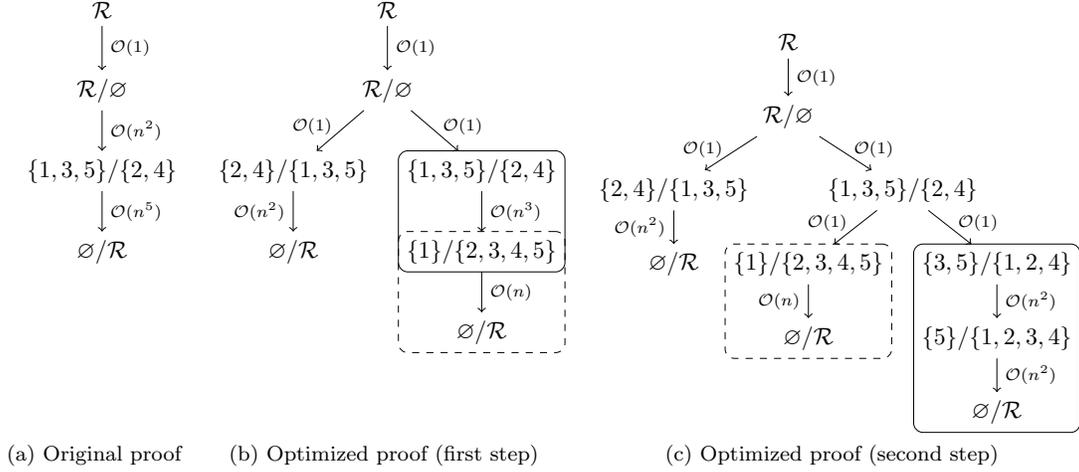


Figure 1: Tightening bounds

Table 1: Derivational complexity of 1172 TRSs

	$\mathcal{O}(n^k)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	time
modular	334/334	208/221	228/321	261/329	2.6/10.6
$\mathfrak{G}\mathfrak{T}$	328/328	216/219	310/317	319/324	4.2/11.5

derivational complexity. All tests have been performed on a server equipped with eight dual-core AMD Opteron[®] processors 885 running at a clock rate of 2.6 GHz and on 64 GB of main memory. If the tool did not report an answer within 60 seconds, its execution was aborted.

Our results are summarized in Table 1. As base methods we use the match-bounds technique [6, 10] as well as polynomially bounded matrix interpretations [5, 7, 8] of dimensions one to five. The row **modular** refers to the implementation as in [10] where all base methods are run in parallel and started upon program execution. For reference we also list the data for the 2010 competition version of $\mathfrak{G}\mathfrak{T}$, which was the winner in the derivational complexity category. In the table the columns $\mathcal{O}(n)$, $\mathcal{O}(n^2)$, etc. give the number of TRSs whose derivational complexity could be shown to be at most linear, quadratic, etc. We also list the total number of TRSs for which a polynomial bound could be established (cf. column $\mathcal{O}(n^k)$) and the average time (in seconds) needed for finding a bound. The numbers before (after) the slash correspond to the setting which does not (does) make use of tightening bounds (cf. Section 3.2).

The results in the table indicate that refining bounds is beneficial, especially if all criteria are started synchronously, which is essential to maximize the total number of upper bounds. The 2010 version of $\mathfrak{G}\mathfrak{T}$ did not use tightening of bounds. To maximize the number of low bounds $\mathfrak{G}\mathfrak{T}$ executes criteria that yield larger complexity bounds slightly delayed. This explains why for $\mathfrak{G}\mathfrak{T}$ tightening bounds increases the global performance less compared to **modular**. On the contrary, $\mathfrak{G}\mathfrak{T}$ misses some proofs compared to **modular** since (costly) criteria are not executed for up to 60 seconds.

For further comparison with other tools we refer the reader to the international termination competition (<http://termcomp.uibk.ac.at>). Since 2008, when the complexity categories have been installed in the termination competition, $\mathfrak{G}\mathfrak{T}$ won the division for derivational complexity

every year.

References

- [1] Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
- [2] Geser, A., Hofbauer, D., Waldmann, J., Zantema, H.: On tree automata that certify termination of left-linear term rewriting systems. *I&C* 205(4), 512–534 (2007)
- [3] Geser, A.: Relative termination. PhD thesis, Universität Passau, Germany (1990). Available as: Report 91-03, Ulmer Informatik-Berichte, Universität Ulm, 1991
- [4] Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations (preliminary version). In: RTA 1989. LNCS, vol. 355, pp. 167–177 (1989)
- [5] Koprowski, A., Waldmann, J.: Max/plus tree automata for termination of term rewriting. *AC* 19(2), 357–392 (2009)
- [6] Korp, M., Middeldorp, A.: Match-bounds revisited. *I&C* 207(11), 1259–1283 (2009)
- [7] Moser, G., Schnabl, A., Waldmann, J.: Complexity analysis of term rewriting based on matrix and context dependent interpretations. In: FSTTCS 2008. LIPIcs, vol. 2, pp. 304–315 (2008)
- [8] Neurauter, F., Zankl, H., Middeldorp, A.: Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In: LPAR 17. LNCS ARCoSS, vol. 6397, pp. 550–564 (2010)
- [9] TeReSe: Term Rewriting Systems. vol. 55 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2003)
- [10] Zankl, H., Korp, M.: Modular complexity analysis via relative complexity. In: RTA 2010. LIPIcs, vol. 6, pp. 385–400 (2010)