Constant Runtime Complexity of Term Rewriting is Semi-Decidable

Florian Frohn, Jürgen Giesl

LuFG Informatik 2, RWTH Aachen University, Germany

Abstract

We prove that it is semi-decidable whether the runtime complexity of a term rewrite system is constant. Our semi-decision procedure exploits that constant runtime complexity is equivalent to termination of a restricted form of narrowing, which can be examined by considering finitely many start terms. We implemented our semi-decision procedure in the tool AProVE to show its efficiency and its success for systems where state-of-the-art complexity analysis tools fail.

Keywords: Computational Complexity, Decidability, Formal Methods, Program Correctness, Term Rewriting

1. Introduction

There are many techniques to infer upper bounds on the runtime complexity of term rewrite systems (TRSs) or closely related formalisms. As "runtime complexity" corresponds to the usual notion of program complexity, such techniques can be used to analyze the complexity of programs in real-world languages via suitable transformations [6, 8, 14]. Usually, complexity bounds are inferred to provide guarantees on a program's resource usage. But *constant* bounds are also important for detecting bugs, as constant-time algorithms cannot fully traverse their in- or output if it exceeds a certain size. Thus, if there is a constant bound for an algorithm which is supposed to traverse arbitrarily large data, then the algorithm is incorrect. To find such bugs in real programs, one would have to combine our results with corresponding transformations from programs to TRSs.

In this paper we prove that it is semi-decidable if the runtime complexity of a TRS is constant. A similar result is known for Turing Machines [12].[1] Note that in general there is no complexity-preserving transformation from one language to another, i.e., semi-decidability of constant bounds for one language does not imply semi-decidability for other Turing-complete languages (like term rewriting). After introducing preliminaries in Sect. 2, we present our semi-decision procedure in Sect. 3. Sect. 4 discusses related work and shows the efficiency

[1] However, Turing Machines and TRSs are inherently different. For example, Turing Machines only read a constant part of the input in constant time, whereas TRSs can copy the whole input in constant time using duplicating rules like $\mathsf{f}(x) \to \mathsf{g}(x, x)$. Similarly, TRSs can compare the whole input in constant time using non-left-linear rules like $\mathsf{f}(x, x) \to \mathsf{g}(x)$, etc.

of our procedure by evaluating our implementation in the tool AProVE [9].

2. Preliminaries

We recapitulate the main notions for TRSs [4]. $\mathcal{T}(\Sigma, \mathcal{V})$ is the set of *terms* over a finite signature Σ and the variables $\mathcal{V}$, where $\mathcal{V}(t)$ is the set of variables occurring in t and $\mathrm{root}(t)$ is the *root symbol* of a term $t \notin \mathcal{V}$. The *positions* $\mathrm{pos}(t) \subset \mathbb{N}^*$ are $\{\varepsilon\}$ if $t \in \mathcal{V}$ and $\{\varepsilon\} \cup \bigcup_{i=1}^{k} \{i.\pi \mid \pi \in \mathrm{pos}(t_i)\}$ if $t = f(t_1, \ldots, t_k)$. The subterm of t at position $\pi \in \mathrm{pos}(t)$ is $t|_\pi = t$ if $\pi = \varepsilon$ and $t|_\pi = t_i|_{\pi'}$ if $\pi = i.\pi'$ and $t = f(t_1, \ldots, t_k)$. The term that results from replacing $t|_\pi$ with $s \in \mathcal{T}(\Sigma, \mathcal{V})$ is $t[s]_\pi$. The *size* of a term is $|x| = 1$ if $x \in \mathcal{V}$ and $|f(t_1, \ldots, t_k)| = 1 + \sum_{i=1}^{k} |t_i|$. A *TRS* $\mathcal{R}$ is a finite set of rules $\{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$ with $\ell_i, r_i \in \mathcal{T}(\Sigma, \mathcal{V})$, $\ell_i \notin \mathcal{V}$, and $\mathcal{V}(r_i) \subseteq \mathcal{V}(\ell_i)$ for all $1 \leq i \leq n$. The *rewrite relation* is defined as $s \to_\mathcal{R} t$ if there are $\pi \in \mathrm{pos}(s)$, $\ell \to r \in \mathcal{R}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$. Here, $\ell\sigma$ is the *redex* of the rewrite step. For two terms s and t, $s \to_\mathcal{R}^n t$ stands for a rewrite sequence $s = s_0 \to_\mathcal{R} s_1 \to_\mathcal{R} \cdots \to_\mathcal{R} s_{n-1} \to_\mathcal{R} s_n = t$ for some terms $s_1, \ldots, s_{n-1}$. The *defined* (resp. *constructor*) symbols of $\mathcal{R}$ are $\Sigma_d(\mathcal{R}) = \{\mathrm{root}(\ell) \mid \ell \to r \in \mathcal{R}\}$ and $\Sigma_c(\mathcal{R}) = \Sigma \setminus \Sigma_d(\mathcal{R})$. A term $f(t_1, \ldots, t_k)$ is *basic* if $f \in \Sigma_d(\mathcal{R})$ and $t_1, \ldots, t_k$ are *constructor terms* (i.e., $t_1, \ldots, t_k \in \mathcal{T}(\Sigma_c(\mathcal{R}), \mathcal{V})$).

Example 1. The following TRS $\mathcal{R}$ is a variation of the example `SK90/4.51` from the *Termination Problems Data Base* (TPDB) [18] where two rules which are not reachable from basic terms were removed for the sake of clarity.

$$\mathsf{f}(\mathsf{a}) \to \mathsf{g}(\mathsf{h}(\mathsf{a})) \qquad\qquad \mathsf{h}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{h}(\mathsf{f}(x)))$$

We have $\Sigma_d(\mathcal{R}) = \{\mathsf{f}, \mathsf{h}\}$, $\Sigma_c(\mathcal{R}) = \{\mathsf{g}, \mathsf{a}\}$, and $x \in \mathcal{V}$. An example rewrite sequence (where the underlined subterms are the redexes) is

$$\underline{\mathsf{h}(\mathsf{g}(\mathsf{a}))} \to_\mathcal{R} \mathsf{g}(\mathsf{h}(\underline{\mathsf{f}(\mathsf{a})})) \to_\mathcal{R} \mathsf{g}(\underline{\mathsf{h}(\mathsf{g}(\mathsf{h}(\mathsf{a}))))} \to_\mathcal{R} \mathsf{g}(\mathsf{g}(\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a})))))).$$

We now define the *runtime complexity* of a TRS $\mathcal{R}$. In the following definition, ω is the smallest infinite ordinal and hence, $\omega > n$ holds for all $n \in \mathbb{N}$. For any $M \subseteq \mathbb{N} \cup \{\omega\}$, $\sup M$ is the least upper bound of M.

Definition 2 (Runtime Complexity [10, 11, 15]). The *derivation height* of a term t w.r.t. a relation $\to$ is the length of the longest sequence of $\to$-steps starting with t, i.e., $\mathrm{dh}(t, \to) = \sup\{n \in \mathbb{N} \mid t' \in \mathcal{T}(\Sigma, \mathcal{V}), t \to^n t'\}$. Thus, $\mathrm{dh}(t, \to) = \omega$ if t starts an infinite $\to$-sequence. The *runtime complexity* function $\mathrm{rc}_\mathcal{R}$ maps any $m \in \mathbb{N}$ to the length of the longest $\to_\mathcal{R}$-sequence starting with a basic term whose size is at most m, i.e., $\mathrm{rc}_\mathcal{R}(m) = \sup\{\mathrm{dh}(t, \to_\mathcal{R}) \mid t \text{ is basic}, |t| \leq m\}$.

Example 3. There is no longer $\to_\mathcal{R}$-sequence for $\mathsf{h}(\mathsf{g}(\mathsf{a}))$ than the one in Ex. 1, i.e., $\mathrm{dh}(\mathsf{h}(\mathsf{g}(\mathsf{a})), \to_\mathcal{R}) = 3$. So $|\mathsf{h}(\mathsf{g}(\mathsf{a}))| = 3$ implies $\mathrm{rc}_\mathcal{R}(3) \geq 3$. Our new approach proves $\mathrm{rc}_\mathcal{R}(m) \in \mathcal{O}(1)$ automatically, i.e., $\mathcal{R}$ has constant runtime complexity.

So our goal is to check whether there is an $n \in \mathbb{N}$ such that all evaluations of basic terms take at most n steps. Our semi-decision procedure is based on *narrowing*, which is similar to rewriting, but uses unification instead of matching.

Definition 4 (Narrowing). A substitution σ is a *unifier* of $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$ if $s\sigma = t\sigma$, and σ is the *most general unifier* (mgu) if every unifier has the form $\sigma \circ \theta$ for some substitution θ. A term s *narrows* to t ($s \overset{\sigma}{\underset{\pi}{\rightsquigarrow}}_{\mathcal{R}} t$) if there is a position $\pi \in \mathrm{pos}(s)$ with $s|_\pi \notin \mathcal{V}$, a (variable-renamed) rule $\ell \to r \in \mathcal{R}$ with $\sigma = \mathrm{mgu}(s|_\pi, \ell)$, and $t = s[r]_\pi\sigma$. We omit π or σ if they are irrelevant and write $s \overset{\sigma_1 \circ \cdots \circ \sigma_n}{\rightsquigarrow}{}^n_{\mathcal{R}} t$ or $s \rightsquigarrow^n_{\mathcal{R}} t$ if we have $s \overset{\sigma_1}{\rightsquigarrow}_{\mathcal{R}} \cdots \overset{\sigma_n}{\rightsquigarrow}_{\mathcal{R}} t$. A finite narrowing sequence $t_0 \overset{\sigma_1}{\rightsquigarrow}_{\mathcal{R}} \cdots \overset{\sigma_n}{\rightsquigarrow}_{\mathcal{R}} t_n$ is *constructor based* if $t_0\,\sigma_1 \cdots \sigma_n$ is a basic term. An infinite narrowing sequence is *constructor based* if all its finite prefixes are constructor based.

Example 5. $\mathcal{R}$ from Ex. 1 has the constructor-based narrowing sequence

$$\underline{\mathsf{h}(x)} \overset{\{x/\mathsf{g}(x')\}}{\underset{\varepsilon}{\rightsquigarrow}}_{\mathcal{R}} \mathsf{g}(\mathsf{h}(\underline{\mathsf{f}(x')})) \overset{\{x'/\mathsf{a}\}}{\underset{1.1}{\rightsquigarrow}}_{\mathcal{R}} \mathsf{g}(\underline{\mathsf{h}(\mathsf{g}(\mathsf{h}(\mathsf{a})))}) \overset{\varnothing}{\underset{1}{\rightsquigarrow}}_{\mathcal{R}} \mathsf{g}(\mathsf{g}(\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a})))))).$$

3. Constant Bounds for Runtime Complexity of Term Rewriting

For our semi-decision procedure, we will show that the runtime complexity of a TRS $\mathcal{R}$ is constant iff $\mathcal{R}$ has no infinite constructor-based narrowing sequence.

Example 6. To see why constructor-based narrowing terminates for $\mathcal{R}$ of Ex. 1, first consider sequences starting with basic terms of the form $\mathsf{h}(t)$. If $t \in \mathcal{V}$, then narrowing $\mathsf{h}(t)$ terminates after three steps, cf. Ex. 5. The same holds if $t = \mathsf{g}(t')$ with $t' \in \mathcal{V}$ or $t' = \mathsf{a}$. For other constructor terms t', narrowing $\mathsf{h}(\mathsf{g}(t'))$ terminates after one step. Finally, if $t \notin \mathcal{V}$ and $\mathrm{root}(t) \neq \mathsf{g}$, then $\mathsf{h}(t)$ is a normal form w.r.t. $\rightsquigarrow_{\mathcal{R}}$. Now we consider basic start terms $\mathsf{f}(t)$. If $t \in \mathcal{V}$ or $t = \mathsf{a}$, then narrowing $\mathsf{f}(t)$ terminates after one step: $\mathsf{f}(t) \rightsquigarrow_{\mathcal{R}} \mathsf{g}(\mathsf{h}(\mathsf{a}))$. If $t \neq \mathsf{a}$ is a non-variable constructor term, then $\mathsf{f}(t)$ is a normal form w.r.t. $\rightsquigarrow_{\mathcal{R}}$. This covers all constructor-based narrowing sequences, i.e., $\mathcal{R}$'s runtime complexity is constant.

In contrast, if we change the second rule to $\mathsf{h}(\mathsf{g}(x)) \to \mathsf{g}(\mathsf{h}(x))$, then the runtime complexity becomes linear and constructor-based narrowing becomes non-terminating: $\underline{\mathsf{h}(x)} \overset{\{x/\mathsf{g}(x')\}}{\rightsquigarrow}_{\mathcal{R}} \mathsf{g}(\underline{\mathsf{h}(x')}) \overset{\{x'/\mathsf{g}(x'')\}}{\rightsquigarrow}_{\mathcal{R}} \mathsf{g}(\mathsf{g}(\underline{\mathsf{h}(x'')})) \rightsquigarrow_{\mathcal{R}} \cdots$

Unfortunately, the reasoning in Ex. 6 is hard to automate, since it explicitly considers all sequences that start with any of the infinitely many basic terms. For automation, we show that constructor-based narrowing sequences can be "generalized" such that one only has to regard finitely many start terms. Then a semi-decision procedure for termination of constructor-based narrowing is obtained by enumerating only those sequences that begin with these start terms.

We first define a partial ordering $\gtrsim$ that clarifies which narrowing sequences are more general than others, and prove the equivalence between constant runtime and termination of constructor-based narrowing afterwards.

Definition 7 (Ordering Narrowing Sequences). Let $\mathcal{R}$ be a TRS and let $s_0 \overset{\sigma_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} s_1 \overset{\sigma_2}{\underset{\pi_2}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\sigma_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} s_n$ and $t_0 \overset{\theta_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} t_1 \overset{\theta_2}{\underset{\pi_2}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\theta_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} t_n$ be narrowing sequences of the same length n that use the same narrowing positions. We have $(s_0 \rightsquigarrow^n_{\mathcal{R}} s_n) \gtrsim (t_0 \rightsquigarrow^n_{\mathcal{R}} t_n)$ if there is a substitution η such that $s_i\,\sigma_{i+1} \cdots \sigma_n\,\eta = t_i\,\theta_{i+1} \cdots \theta_n$ for all $0 \leq i < n$ and $s_n\,\eta = t_n$.

Example 8. For the TRS $\mathcal{R}$ of Ex. 1, we have

$$\begin{array}{llllll}
\underline{\mathsf{h}(x)} & \overset{\{x/\mathsf{g}(x')\}}{\underset{\varepsilon}{\rightsquigarrow}}_{\mathcal{R}} & \mathsf{g}(\mathsf{h}(\underline{\mathsf{f}(x')})) & \overset{\{x'/\mathsf{a}\}}{\underset{1.1}{\rightsquigarrow}}_{\mathcal{R}} & \mathsf{g}(\underline{\mathsf{h}(\mathsf{g}(\mathsf{h}(\mathsf{a})))}) & \overset{\varnothing}{\underset{1}{\rightsquigarrow}}_{\mathcal{R}} \mathsf{g}(\mathsf{g}(\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a})))))\text{,} \\
\underline{\mathsf{h}(\mathsf{g}(x'))} & \overset{\varnothing}{\underset{\varepsilon}{\rightsquigarrow}}_{\mathcal{R}} & \mathsf{g}(\mathsf{h}(\underline{\mathsf{f}(x')})) & \overset{\{x'/\mathsf{a}\}}{\underset{1.1}{\rightsquigarrow}}_{\mathcal{R}} & \mathsf{g}(\underline{\mathsf{h}(\mathsf{g}(\mathsf{h}(\mathsf{a})))}) & \overset{\varnothing}{\underset{1}{\rightsquigarrow}}_{\mathcal{R}} \mathsf{g}(\mathsf{g}(\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a})))))\text{, and} \\
\underline{\mathsf{h}(\mathsf{g}(\mathsf{a}))} & \overset{\varnothing}{\underset{\varepsilon}{\rightsquigarrow}}_{\mathcal{R}} & \mathsf{g}(\mathsf{h}(\underline{\mathsf{f}(\mathsf{a})})) & \overset{\varnothing}{\underset{1.1}{\rightsquigarrow}}_{\mathcal{R}} & \mathsf{g}(\underline{\mathsf{h}(\mathsf{g}(\mathsf{h}(\mathsf{a})))}) & \overset{\varnothing}{\underset{1}{\rightsquigarrow}}_{\mathcal{R}} \mathsf{g}(\mathsf{g}(\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a})))))\text{.}
\end{array}$$

Thus, $(\mathsf{h}(x) \rightsquigarrow^3_{\mathcal{R}} t) \gtrsim (\mathsf{h}(\mathsf{g}(x')) \rightsquigarrow^3_{\mathcal{R}} t) \gtrsim (\mathsf{h}(\mathsf{g}(\mathsf{a})) \rightsquigarrow^3_{\mathcal{R}} t)$ for $t = \mathsf{g}(\mathsf{g}(\mathsf{h}(\mathsf{f}(\mathsf{h}(\mathsf{a})))))$, i.e., the sequence $\mathsf{h}(x) \rightsquigarrow^3_{\mathcal{R}} t$ is most general. Indeed, all the sequences in Ex. 6 are specializations (w.r.t. $\gtrsim$) of sequences that start with $\mathsf{h}(x)$ or $\mathsf{f}(x)$.

The following theorem shows that every constructor-based narrowing sequence is a specialization of a narrowing sequence starting with a basic term of the form $f(x_1, \ldots, x_k)$. Thus, when reasoning about termination of constructor-based narrowing, it suffices to consider only the "most general" sequences that start with such basic terms. This is the foundation of our semi-decision procedure, as there are just finitely many such terms (up to variable renaming).

Theorem 9 (Most General Narrowing Sequences)**.** *For a TRS $\mathcal{R}$, let $t_0 \overset{\theta_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\theta_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} t_n$ be a constructor-based narrowing sequence where $\mathrm{root}(t_0) = f$. Then there exists a narrowing sequence $f(x_1, \ldots, x_k) = s_0 \overset{\sigma_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\sigma_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} s_n$ for pairwise different variables $x_1, \ldots, x_k$ such that $(s_0 \rightsquigarrow^n_{\mathcal{R}} s_n) \gtrsim (t_0 \rightsquigarrow^n_{\mathcal{R}} t_n)$.*

Proof. We prove for all n: If $t_0 \overset{\theta_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\theta_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} t_n$ is constructor based and $\mathrm{root}(t_0) = f$, then there is a narrowing sequence $f(x_1, ..., x_k) = s_0 \overset{\sigma_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\sigma_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} s_n$ with

$$(s_0 \overset{\sigma_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\sigma_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} s_n) \gtrsim (t_0 \overset{\theta_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\theta_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} t_n) \text{ and} \tag{1}$$

$$\mathrm{root}(t_i|_\tau) \in \Sigma_d(\mathcal{R}) \text{ implies } \mathrm{root}(t_i|_\tau) = \mathrm{root}(s_i|_\tau) \text{ for all } 0 \le i \le n \text{ and } \tau \in \mathrm{pos}(t_i). \tag{2}$$

We use induction on n. If $n = 0$, then there is clearly a substitution η with $s_0\,\eta = f(x_1, \ldots, x_k)\,\eta = t_0$ since $\mathrm{root}(t_0) = f$. Moreover as t_0 is basic, its only defined symbol is at the root position ε. We have $\mathrm{root}(t_0|_\varepsilon) = f = \mathrm{root}(s_0|_\varepsilon)$.

In the induction step, we may assume that $t_0 \overset{\theta_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\theta_{n+1}}{\underset{\pi_{n+1}}{\rightsquigarrow}}_{\mathcal{R}} t_{n+1}$ is constructor based. Let $\ell \to r \in \mathcal{R}$ be the (variable-renamed) rule used for $t_n \overset{\theta_{n+1}}{\underset{\pi_{n+1}}{\rightsquigarrow}}_{\mathcal{R}} t_{n+1}$, i.e., $t_n|_{\pi_{n+1}} \notin \mathcal{V}$, $\theta_{n+1} = \mathrm{mgu}(t_n|_{\pi_{n+1}}, \ell)$, $t_{n+1} = t_n[r]_{\pi_{n+1}}\,\theta_{n+1}$. By the induction hypothesis, (1) and (2) hold. By (1), there is a substitution η with

$$s_i\,\sigma_{i+1} \cdots \sigma_n\,\eta = t_i\,\theta_{i+1} \cdots \theta_n \text{ for all } 0 \le i < n \quad \text{and} \quad s_n\,\eta = t_n. \tag{3}$$

Like in the step $t_n \overset{\theta_{n+1}}{\underset{\pi_{n+1}}{\rightsquigarrow}}_{\mathcal{R}} t_{n+1}$, we now want to use the same rule $\ell \to r$ on the same position π_{n+1} to narrow s_n. Since $t_n|_{\pi_{n+1}} \notin \mathcal{V}$ and $t_n|_{\pi_{n+1}}$ unifies with ℓ, we have $\mathrm{root}(t_n|_{\pi_{n+1}}) \in \Sigma_d(\mathcal{R})$. Thus, (2) implies $\mathrm{root}(t_n|_{\pi_{n+1}}) = \mathrm{root}(s_n|_{\pi_{n+1}})$ and hence, $s_n|_{\pi_{n+1}} \notin \mathcal{V}$. Moreover, as w.l.o.g. s_n is variable-disjoint from ℓ, $s_n|_{\pi_{n+1}}$ unifies with ℓ because (3) implies $s_n\,\eta = t_n$ and thus,

$$s_n|_{\pi_{n+1}}\,\eta\,\theta_{n+1} \;=\; s_n\,\eta\,\theta_{n+1}|_{\pi_{n+1}} \;=\; t_n\,\theta_{n+1}|_{\pi_{n+1}} \;=\; t_n|_{\pi_{n+1}}\,\theta_{n+1} \;=\; \ell\,\theta_{n+1},$$

as θ_{n+1} unifies $t_n|_{\pi_{n+1}}$ and ℓ. Let σ_{n+1} be the mgu of $s_n|_{\pi_{n+1}}$ and ℓ. Then we have $s_n \overset{\sigma_{n+1}}{\underset{\pi_{n+1}}{\rightsquigarrow}}_{\mathcal{R}} s_n[r]_{\pi_{n+1}}\,\sigma_{n+1} = s_{n+1}$. It remains to show that

$$(s_0 \overset{\sigma_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\sigma_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} s_n \overset{\sigma_{n+1}}{\underset{\pi_{n+1}}{\rightsquigarrow}}_{\mathcal{R}} s_{n+1}) \gtrsim (t_0 \overset{\theta_1}{\underset{\pi_1}{\rightsquigarrow}}_{\mathcal{R}} \cdots \overset{\theta_n}{\underset{\pi_n}{\rightsquigarrow}}_{\mathcal{R}} t_n \overset{\theta_{n+1}}{\underset{\pi_{n+1}}{\rightsquigarrow}}_{\mathcal{R}} t_{n+1}) \text{ and} \tag{4}$$

$$\mathrm{root}(t_i|_\tau) \in \Sigma_d(\mathcal{R}) \text{ implies } \mathrm{root}(t_i|_\tau) = \mathrm{root}(s_i|_\tau) \text{ for all } 0 \le i \le n+1 \text{ and } \tau \in \mathrm{pos}(t_i). \tag{5}$$

To prove (5), due to (2) we only have to regard the case where $i = n + 1$.

Thus, let $\mathrm{root}(t_{n+1}|_\tau) = \mathrm{root}(t_n[r]_{\pi_{n+1}}\,\theta_{n+1}\,|_\tau) \in \Sigma_d(\mathcal{R})$. We first regard the case where $\tau \in \mathrm{pos}(t_n[r]_{\pi_{n+1}})$ and $\mathrm{root}(t_n[r]_{\pi_{n+1}}\,|_\tau) \in \Sigma_d(\mathcal{R})$. Then we have

$$\begin{array}{ll}
\mathrm{root}(t_n[r]_{\pi_{n+1}}\,\theta_{n+1}\,|_\tau) & \\
= \mathrm{root}(t_n[r]_{\pi_{n+1}}|_\tau) & \text{as } \tau \in \mathrm{pos}(t_n[r]_{\pi_{n+1}}) \text{ and } \mathrm{root}(t_n[r]_{\pi_{n+1}}|_\tau) \in \Sigma_d(\mathcal{R}) \\
= \mathrm{root}(s_n[r]_{\pi_{n+1}}|_\tau) & \text{by the induction hypothesis (2)} \\
= \mathrm{root}(s_n[r]_{\pi_{n+1}}\,\sigma_{n+1}\,|_\tau) & \text{as } \mathrm{root}(s_n[r]_{\pi_{n+1}}|_\tau) \in \Sigma_d(\mathcal{R}), \text{ i.e., } s_n[r]_{\pi_{n+1}}|_\tau \notin \mathcal{V} \\
= \mathrm{root}(s_{n+1}|_\tau) & \text{as } s_{n+1} = s_n[r]_{\pi_{n+1}}\,\sigma_{n+1}.
\end{array}$$

Otherwise, since $t_0\,\theta_1 \cdots \theta_{n+1}$ is basic, θ_{n+1} instantiates $\mathcal{V}(t_0\,\theta_1 \cdots \theta_n)$ by constructor terms. As $t_0\,\theta_1 \cdots \theta_n \to_{\mathcal{R}}^n t_n$, we have $\mathcal{V}(t_n) \subseteq \mathcal{V}(t_0\,\theta_1 \cdots \theta_n)$. Hence, $\tau \geq \pi_{n+1}$ (i.e., τ is below or equal to π_{n+1}), since θ_{n+1} only instantiates $\mathcal{V}(t_n)$ by constructor terms. Thus, there are positions τ_r and τ' such that $\tau = \pi_{n+1}.\tau_r.\tau'$ and $r|_{\tau_r} \in \mathcal{V}$. Since $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$, there is also a position τ_ℓ with $\ell|_{\tau_\ell} = r|_{\tau_r}$. Thus, for $\mathrm{root}(t_{n+1}|_\tau) = \mathrm{root}(t_n[r]_{\pi_{n+1}}\,\theta_{n+1}|_\tau) \in \Sigma_d(\mathcal{R})$ we now get

$$\begin{array}{ll}
\mathrm{root}(t_n[r]_{\pi_{n+1}}\,\theta_{n+1}\,|_\tau) & \\
= \mathrm{root}(t_n[r]_{\pi_{n+1}}\,\theta_{n+1}\,|_{\pi_{n+1}.\tau_r.\tau'}) & \text{as } \tau = \pi_{n+1}.\tau_r.\tau' \\
= \mathrm{root}(t_n[\ell]_{\pi_{n+1}}\,\theta_{n+1}\,|_{\pi_{n+1}.\tau_\ell.\tau'}) & \text{as } r|_{\tau_r} = \ell|_{\tau_\ell} \\
= \mathrm{root}(t_n\,\theta_{n+1}\,|_{\pi_{n+1}.\tau_\ell.\tau'}) & \text{as } \theta_{n+1} = \mathrm{mgu}(t_n|_{\pi_{n+1}}, \ell) \\
= \mathrm{root}(t_n|_{\pi_{n+1}.\tau_\ell.\tau'}) & \text{as } \theta_{n+1} \text{ instantiates } \mathcal{V}(t_n) \text{ by constructor terms} \\
= \mathrm{root}(s_n|_{\pi_{n+1}.\tau_\ell.\tau'}) & \text{by the induction hypothesis (2)} \\
= \mathrm{root}(s_n\,\sigma_{n+1}\,|_{\pi_{n+1}.\tau_\ell.\tau'}) & \text{as } \mathrm{root}(s_n|_{\pi_{n+1}.\tau_\ell.\tau'}) \in \Sigma_d(\mathcal{R}) \\
= \mathrm{root}(s_n[\ell]_{\pi_{n+1}}\,\sigma_{n+1}\,|_{\pi_{n+1}.\tau_\ell.\tau'}) & \text{as } \sigma_{n+1} = \mathrm{mgu}(s_n|_{\pi_{n+1}}, \ell) \\
= \mathrm{root}(s_n[r]_{\pi_{n+1}}\,\sigma_{n+1}\,|_{\pi_{n+1}.\tau_r.\tau'}) & \text{as } r|_{\tau_r} = \ell|_{\tau_\ell} \\
= \mathrm{root}(s_{n+1}|_{\pi_{n+1}.\tau_r.\tau'}) & \text{as } s_{n+1} = s_n[r]_{\pi_{n+1}}\,\sigma_{n+1} \\
= \mathrm{root}(s_{n+1}|_\tau) & \text{as } \tau = \pi_{n+1}.\tau_r.\tau'.
\end{array}$$

To prove (4), we have to show that there is a substitution η' such that

$$s_i\,\sigma_{i+1} \cdots \sigma_{n+1}\,\eta' = t_i\,\theta_{i+1} \cdots \theta_{n+1} \quad \text{for all } 0 \leq i \leq n+1.$$

By (3), we have $s_n\,\eta = t_n$ and moreover, we had $\theta_{n+1} = \mathrm{mgu}(t_n|_{\pi_{n+1}}, \ell)$. Hence, we obtain $s_n\,\eta\,\theta_{n+1} = t_n\,\theta_{n+1} = t_n[\ell]_{\pi_{n+1}}\,\theta_{n+1}$. W.l.o.g., s_n, t_n, and ℓ are variable disjoint and thus, a unifier of s_n, t_n, and $t_n[\ell]_{\pi_{n+1}}$ is

$$\mu \;=\; (\eta \circ \theta_{n+1})|_{\mathcal{V}\setminus(\mathcal{V}(t_n)\cup\mathcal{V}(\ell))} \;\cup\; \theta_{n+1}|_{\mathcal{V}(t_n)\cup\mathcal{V}(\ell)}. \tag{6}$$

Here, for any $V \subseteq \mathcal{V}$ and any substitution σ, $\sigma|_V$ denotes the restriction of σ to V, i.e., we have $x\sigma|_V = x\sigma$ if $x \in V$ and $x\sigma|_V = x$, otherwise.

As μ is a unifier of $s_n|_{\pi_{n+1}}$ and ℓ, by the definition of $\sigma_{n+1} = \mathrm{mgu}(s_n|_{\pi_{n+1}}, \ell)$ there must be a substitution η' such that $\sigma_{n+1} \circ \eta' = \mu$. Thus, we get

$$\begin{array}{lll}
s_{n+1}\,\eta' = s_n[r]_{\pi_{n+1}}\,\sigma_{n+1}\,\eta' & & \text{as } s_{n+1} = s_n[r]_{\pi_{n+1}}\,\sigma_{n+1} \\
\quad = s_n[r]_{\pi_{n+1}}\,\mu & & \text{as } \sigma_{n+1} \circ \eta' = \mu \\
\quad = s_n\,\mu\,[r\,\mu]_{\pi_{n+1}} & & \\
\quad = s_n\,\eta\,\theta_{n+1}\,[r\,\theta_{n+1}]_{\pi_{n+1}} & & \text{by (6) since } \mathcal{V}(r) \subseteq \mathcal{V}(\ell) \\
\quad = s_n\,\eta\,[r]_{\pi_{n+1}}\,\theta_{n+1} & & \\
\quad = t_n[r]_{\pi_{n+1}}\,\theta_{n+1} & & \text{as } s_n\,\eta = t_n \text{ by the induction hypothesis (3)} \\
\quad = t_{n+1} & & \text{as } t_{n+1} = t_n[r]_{\pi_{n+1}}\,\theta_{n+1}.
\end{array}$$

For all $0 \leq i \leq n$ we have

$$\begin{aligned} s_i\,\sigma_{i+1}\cdots\sigma_{n+1}\,\eta' &= s_i\,\sigma_{i+1}\cdots\sigma_n\,\mu && \text{as } \sigma_{n+1}\circ\eta' = \mu \\ &= s_i\,\sigma_{i+1}\cdots\sigma_n\,\eta\,\theta_{n+1} && \text{by (6)} \\ &= t_i\,\theta_{i+1}\cdots\theta_n\theta_{n+1} && \text{as } s_i\,\sigma_{i+1}\cdots\sigma_n\,\eta = t_i\,\theta_{i+1}\cdots\theta_n \\ & && \text{by the induction hypothesis (3). } \square \end{aligned}$$

The following theorem shows that constant runtime complexity is indeed equivalent to termination of constructor-based narrowing.

Theorem 10 (Constant Runtime and Constructor-Based Narrowing)**.** *We have* $\mathrm{rc}_\mathcal{R}(m) \in \mathcal{O}(1)$ *iff there is no infinite constructor-based narrowing sequence.*

Proof. For the "if" direction, let $\mathrm{rc}_\mathcal{R}(m) \notin \mathcal{O}(1)$. Then for each $n \in \mathbb{N}$ there is a rewrite sequence of length n starting with a basic term $f(\ldots)$. Since $\Sigma_d(\mathcal{R})$ is finite, there is an $f \in \Sigma_d(\mathcal{R})$ with rewrite sequences $t_1 \to_\mathcal{R}^{n_1} q_1, t_2 \to_\mathcal{R}^{n_2} q_2, \ldots$, where $n_1 < n_2 < \cdots$, $\mathrm{root}(t_1) = \mathrm{root}(t_2) = \cdots = f$, and $t_1, t_2, \ldots$ are basic. Every rewrite sequence is also a narrowing sequence (where the narrowing substitutions just instantiate variables in rules, but not in narrowed terms). Thus, $t_i \overset{\theta_i}{\rightsquigarrow}_\mathcal{R}^{n_i} q_i$ where $t_i\theta_i = t_i$ for all $i \geq 1$. By Lemma 9, $f(x_1, \ldots, x_k)$ starts narrowing sequences $f(x_1, \ldots, x_k) \overset{\sigma_1}{\rightsquigarrow}_\mathcal{R}^{n_1} s_1$, $f(x_1, \ldots, x_k) \overset{\sigma_2}{\rightsquigarrow}_\mathcal{R}^{n_2} s_2$, …, where $(f(x_1, \ldots, x_k) \overset{\sigma_i}{\rightsquigarrow}_\mathcal{R}^{n_i} s_i) \gtrsim (t_i \overset{\theta_i}{\rightsquigarrow}_\mathcal{R}^{n_i} q_i)$ and thus, $f(x_1, \ldots, x_k)\sigma_i$ matches $t_i\theta_i = t_i$ for all $i \geq 1$. Hence, all $f(x_1, \ldots, x_k)\sigma_i$ are basic, i.e., all narrowing sequences $f(x_1, \ldots, x_k) \overset{\sigma_i}{\rightsquigarrow}_\mathcal{R}^{n_i} s_i$ are constructor based. Consider the tree of all constructor-based narrowing sequences (up to variable renaming) starting in the root node $f(x_1, \ldots, x_k)$. Since $f(x_1, \ldots, x_k) \overset{\sigma_i}{\rightsquigarrow}_\mathcal{R}^{n_i} s_i$ for infinitely many $n_1 < n_2 < \cdots$, the tree has infinitely many nodes. As $\mathcal{R}$ is finite, $\rightsquigarrow_\mathcal{R}$ is finitely branching. Thus, by König's Lemma the tree has an infinite path, i.e., there is an infinite narrowing sequence starting with $f(x_1, \ldots, x_k)$ whose finite prefixes are all constructor based. Hence, the infinite narrowing sequence is also constructor based.

For the "only if" direction, let $\mathrm{rc}_\mathcal{R}(m) \in \mathcal{O}(1)$, i.e., there is an $n \in \mathbb{N}$ such that $\mathrm{rc}_\mathcal{R}(m) \leq n$ holds for all $m \in \mathbb{N}$. Assume that there is an infinite constructor-based narrowing sequence $t_0 \overset{\sigma_1}{\rightsquigarrow}_\mathcal{R} t_1 \overset{\sigma_2}{\rightsquigarrow}_\mathcal{R} \cdots$ Then for each $n \in \mathbb{N}$, we have $t_0\,\sigma_1 \cdots \sigma_{n+1} \to_\mathcal{R}^{n+1} t_{n+1}$. As $t_0\,\sigma_1 \cdots \sigma_{n+1}$ is basic, this is a contradiction to $\mathrm{rc}_\mathcal{R}(m) \leq n$ when choosing $m = |t_0\,\sigma_1 \cdots \sigma_{n+1}|$. $\square$

By Thm. 9 and 10, $\mathrm{rc}_\mathcal{R}(m) \in \mathcal{O}(1)$ holds iff there is no infinite constructor-based narrowing sequence that starts with a term of the form $f(x_1, \ldots, x_k)$ where $f \in \Sigma_d(\mathcal{R})$ and $x_1, \ldots, x_k$ are pairwise different variables. This yields our main result: It is semi-decidable whether a TRS has constant runtime.

Procedure 1 Semi-Decision Procedure for $\mathrm{rc}_\mathcal{R}(m) \in \mathcal{O}(1)$

1. For each $f \in \Sigma_d(\mathcal{R})$
   1.1. Set $n := 0$.
   1.2. Set $n := n + 1$.
   1.3. Iterate over all narrowing sequences $f(x_1,\ldots,x_k) \overset{\sigma}{\rightsquigarrow}_\mathcal{R}^{n} s$ (up to variable renaming). If there is a sequence where $f(x_1,\ldots,x_k)\sigma$ is basic, then go back to Step 1.2.
2. Return "$\mathrm{rc}_\mathcal{R}(m) \in \mathcal{O}(1)$".

Corollary 11 (Constant Runtime of Rewriting is Semi-Decidable)**.** *Proc. 1 is a semi-decision procedure for* $\mathrm{rc}_{\mathcal{R}}(m) \in \mathcal{O}(1)$.

To implement Proc. 1 efficiently, one builds up the trees of all constructor-based narrowing sequences for all terms $f(x_1, \ldots, x_k)$, and returns "$\mathrm{rc}_{\mathcal{R}}(m) \in \mathcal{O}(1)$" if constructing these trees terminates (i.e., if all these trees are finite).

Example 12. Reconsider the variation of `SK90/4.51` from the TPDB in Ex. 1. Until 2016, no tool could prove that the runtime of `SK90/4.51` is constant at the annual *Termination and Complexity Competition* (*TermComp*) [17].[2] Since *TermComp 2016*, AProVE can infer a constant upper bound via the new semi-decision procedure in Proc. 1. Thus, it enumerates all constructor-based narrowing sequences starting with $\mathsf{h}(x)$ and $\mathsf{f}(x)$ before returning "$\mathrm{rc}_{\mathcal{R}}(m) \in \mathcal{O}(1)$".

One can adapt Proc. 1 to *innermost* rewriting by only considering *innermost narrowing sequences* $f(x_1, \ldots, x_k) \overset{\sigma}{\rightsquigarrow}{}^{n}_{\mathcal{R}} s$ in Step 1.3. A narrowing sequence $t \overset{\theta}{\rightsquigarrow}{}^{n}_{\mathcal{R}} q$ is *innermost* if $t\theta \to^{n}_{\mathcal{R}} q$ is an *innermost* rewrite sequence. This adaption is sound because whenever there is a constructor-based innermost narrowing sequence $t \overset{\theta}{\rightsquigarrow}{}^{n}_{\mathcal{R}} q$, then there is a narrowing sequence $(f(x_1, \ldots, x_k) \overset{\sigma}{\rightsquigarrow}{}^{n}_{\mathcal{R}} s) \gtrsim (t \overset{\theta}{\rightsquigarrow}{}^{n}_{\mathcal{R}} q)$ due to Lemma 9. Note that $f(x_1, \ldots, x_k) \overset{\sigma}{\rightsquigarrow}{}^{n}_{\mathcal{R}} s$ is also an *innermost* sequence because the steps are performed at the same positions as in $t \overset{\theta}{\rightsquigarrow}{}^{n}_{\mathcal{R}} q$.

Corollary 13 (Constant Runtime of Innermost Rewriting is Semi-Decidable)**.** *It is semi-decidable if the innermost runtime complexity of a TRS is constant.*

4. Conclusion, Related Work, and Experiments

We proved that constant runtime is semi-decidable for TRSs and implemented our semi-decision procedure in the tool AProVE. This implementation complements AProVE's other techniques to analyze the runtime complexity of TRSs.

Related Work. There are many approaches to analyze the runtime complexity of TRSs (e.g., [10, 15, 19]), but they do not focus on *constant* upper bounds.[3] While some approaches apply narrowing for case analyses, we use narrowing as a stand-alone technique for runtime complexity analysis.

In [7] we investigated *lower bounds* and showed that $\mathrm{rc}_{\mathcal{R}}(m) \in \Omega(m)$ is not semi-decidable. Here, we also used a connection between runtime bounds and narrowing. The proof in [7] shows that $\mathrm{rc}_{\mathcal{R}}(m) \notin \mathcal{O}(1)$ is not semi-decidable either. So with Cor. 11, $\mathrm{rc}_{\mathcal{R}}(m) \in \mathcal{O}(1)$ is semi-decidable, but not decidable.

Our technique can also be used to prove that a TRS has the *finite variant*

[2] Without Proc. 1, the leading tools for complexity of term rewriting at *TermComp* (AProVE and TcT [3]) also fail to prove a constant upper bound for the variant of `SK90/4.51` in Ex. 1.

[3] An exception is the tool Oops [16] which was used to detect TRSs with constant bounds at *TermComp 2010*. However, it only handles TRSs where there are only finitely many basic terms (as all constructors or all defined symbols are constants) or where there is no rule whose left-hand side is basic. Clearly, our technique succeeds in both cases and hence subsumes Oops.

property, which is, e.g., of interest for equational unification. The reason is that constant runtime complexity implies the finite variant property [5, Section 9].

In [2, 13] a semi-decision procedure for termination of *basic narrowing* is discussed, which is similar to Proc. 1. While termination of basic and constructor-based narrowing coincides for *left-linear constructor systems* where all left-hand sides are basic terms that do not contain the same variable twice, in general the two notions differ.[4] Moreover, [2, 13] does not discuss the relationship between narrowing and runtime complexity (i.e., it contains no result like Thm. 10).

Experiments. We evaluated our technique on all 959 examples from the category "Runtime Complexity – Full Rewriting" and all 1022 examples from the category "Runtime Complexity – Innermost Rewriting" of the TPDB 10.4 [18], the example collection used at *TermComp 2016* [17]. For these categories, the TPDB 10.5 used at *TermComp 2017* is a subset of the TPDB 10.4, as all non-left-linear and all non-constructor systems were removed from the innermost rewriting category. We used a timeout of 60 s per example and compared the performance of our semi-decision procedure with TcT [3] and various versions of AProVE [9] where we disabled the new technique from this paper. These are the most powerful complexity tools for TRSs at *TermComp* since many years.

For full rewriting, our procedure infers a constant upper bound for 57 examples. For 6 of them, neither AProVE nor TcT can show $\mathrm{rc}_{\mathcal{R}}(m) \in \mathcal{O}(1)$. The average runtime of our procedure on successfully analyzed examples was 1.8 s. For all but 5 TRSs where our procedure fails, AProVE can disprove constant complexity (by inferring a non-constant *lower* bound for $\mathrm{rc}_{\mathcal{R}}(m)$). A manual analysis [1] reveals that the remaining 5 TRSs have non-constant complexity, too.

For innermost rewriting, the numbers are almost identical. Here, our technique succeeds in 58 cases, including an example whose runtime is constant w.r.t. innermost, but not w.r.t. full rewriting. So our procedure indeed detects all (standard) TRSs with constant runtime in these categories of the TPDB. On the other hand, it fails for one non-constant example with *relative* rules, where it is clearly still sound, but no longer a semi-decision procedure.

See [1] for a discussion of our experiments including a detailed comparison between our new semi-decision procedure and existing techniques for the inference of constant bounds, as well as a web interface to access our implementation.

Acknowledgement. We are grateful to Carsten Fuhs for many helpful comments. The work was supported by the DFG grant GI 274/6-1.

References

[1] `https://aprove-developers.github.io/trs-constant-bounds/`.

[4] Basic narrowing disallows narrowing steps that reduce subterms which were introduced by preceding narrowing substitutions. Thus, basic narrowing does not terminate for $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}), \mathsf{a} \to \mathsf{b}\}$ due to the infinite basic narrowing sequence $\mathsf{f}(\mathsf{a}) \rightsquigarrow_{\mathcal{R}} \mathsf{f}(\mathsf{a}) \rightsquigarrow_{\mathcal{R}} \cdots$ In contrast, all constructor-based narrowing sequences w.r.t. $\mathcal{R}$ have at most length 1.

[2] M. Alpuente, S. Escobar, J. Iborra, Termination of narrowing revisited, TCS 410 (46) (2009) 4608–4625.

[3] M. Avanzini, G. Moser, M. Schaper, TcT: Tyrolean complexity tool, in: TACAS'16, LNCS 9636, 2016, pp. 407–423.

[4] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge, 1998.

[5] C. Bouchard, K. A. Gero, C. Lynch, P. Narendran, On forward closure and the finite variant property, in: FroCoS'13, LNCS 8152, 2013, pp. 327–342.

[6] F. Frohn, J. Giesl, Complexity analysis for Java with AProVE, in: iFM'17, LNCS 10510, 2017, pp. 85–101.

[7] F. Frohn, J. Giesl, J. Hensel, C. Aschermann, T. Ströder, Lower bounds for runtime complexity of term rewriting, JAR 59 (1) (2017) 121–163.

[8] J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, C. Fuhs, Symbolic evaluation graphs and term rewriting: A general methodology for analyzing logic programs, in: PPDP'12, 2012, pp. 1–12.

[9] J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, R. Thiemann, Analyzing program termination and complexity automatically with AProVE, JAR 58 (1) (2017) 3–31.

[10] N. Hirokawa, G. Moser, Automated complexity analysis based on the dependency pair method, in: IJCAR'08, LNCS 5195, 2008, pp. 364–379.

[11] D. Hofbauer, C. Lautemann, Termination proofs and the length of derivations, in: RTA'89, LNCS 355, 1989, pp. 167–177.

[12] C. E. Hughes, S. M. Selkow, The finite power property for context-free languages, TCS 15 (1981) 111–114.

[13] J. M. Hullot, Canonical forms and unification, in: CADE'80, LNCS 87, 1980, pp. 318–334.

[14] G. Moser, M. Schaper, From Jinja Bytecode to term rewriting: A complexity reflecting transformation, `cbr.uibk.ac.at/publications/ic16.pdf`.

[15] L. Noschinski, F. Emmes, J. Giesl, Analyzing innermost runtime complexity of term rewriting by dependency pairs, JAR 51 (1) (2013) 27–56.

[16] Oops, `http://www.termination-portal.org/wiki/Tools:Oops`.

[17] *TermComp*, `http://termination-portal.org/wiki/Termination_Competition`

[18] TPDB, `http://termination-portal.org/wiki/TPDB`.

[19] H. Zankl, M. Korp, Modular complexity analysis for term rewriting, LMCS 10 (1:19) (2014) 1–33.