# Complexity Analysis for Term Rewriting by Integer Transition Systems*

M. Naaf[1], F. Frohn[1], M. Brockschmidt[2], C. Fuhs[3], and J. Giesl[1]

[1] LuFG Informatik 2, RWTH Aachen University, Germany
[2] Microsoft Research, Cambridge, UK
[3] Birkbeck, University of London, UK

**Abstract.** We present a new method to infer upper bounds on the innermost runtime complexity of term rewrite systems (TRSs), which benefits from recent advances on complexity analysis of integer transition systems (ITSs). To this end, we develop a transformation from TRSs to a generalized notion of ITSs with (possibly non-tail) recursion. To analyze their complexity, we introduce a modular technique which allows us to use existing tools for standard ITSs in order to infer complexity bounds for our generalized ITSs. The key idea of our technique is a summarization method that allows us to analyze components of the transition system independently. We implemented our contributions in the tool AProVE, and our experiments show that one can now infer bounds for significantly more TRSs than with previous state-of-the-art tools for term rewriting.

## 1 Introduction

There are many techniques for automatic complexity analysis of programs with integer (or natural) numbers, e.g., [1,2,4,11,13,14,16–18,23,26–28,34]. On the other hand, several techniques analyze complexity of *term rewrite systems* (TRSs), e.g., [7, 8, 12, 19, 20, 24, 29, 32, 36]. TRSs are a classical model for equational reasoning and evaluation with user-defined data structures and recursion [9].

Although the approaches for complexity analysis of term rewriting support modularity, they usually cannot completely remove rules from the TRS after having analyzed them. In contrast, approaches for integer programs may regard small program parts independently and combine the results for these parts to obtain a result for the overall program. In this work, we show how to obtain such a form of modularity also for complexity analysis of TRSs.

After recapitulating TRSs and their complexity in Sect. 2, in Sect. 3 we introduce a transformation from TRSs into a variant of integer transition systems (ITSs) called *recursive natural transition systems* (RNTSs). In contrast to standard ITSs, RNTSs allow arbitrary recursion, and the variables only range over the natural numbers. We show that the innermost runtime complexity of the original TRS is bounded by the complexity of the resulting RNTS, i.e., one can now use any complexity tool for RNTSs to infer complexity bounds for TRSs.

* Supported by DFG grant GI 274/6-1 and the Air Force Research Laboratory (AFRL).

Unfortunately, many existing techniques and tools for standard ITSs do not support the non-tail recursive calls that can occur in RNTSs. Therefore, in Sect. 4 we develop an approach to infer complexity bounds for RNTSs which can use arbitrary complexity tools for standard ITSs as a back-end. The approach from Sect. 4 is completely modular, as it repeatedly finds bounds for parts of the RNTS and combines them. In this way, our technique benefits from all advances of any ITS tools, irrespective of whether they support non-tail recursion (e.g., CoFloCo [16,17]) or not (e.g., KoAT [13]). As demonstrated by our implementation in AProVE [22], our contributions allow us to derive complexity bounds for many TRSs where state-of-the-art tools fail, cf. Sect. 5. All proofs can be found in [5].

## 2 Complexity of Term Rewriting

We assume basic knowledge of term rewriting [9] and recapitulate innermost (relative) term rewriting and its runtime complexity.

**Definition 1 (Term Rewriting [8, 9]).** *We denote the set of* terms *over a finite signature $\Sigma$ and the variables $\mathcal{V}$ by $\mathcal{T}(\Sigma, \mathcal{V})$. The* size *$|t|$ of a term $t$ is defined as $|x| = 1$ 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 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 $s \to_{\mathcal{R}} t$ iff there is a rule $\ell \to r \in \mathcal{R}$, a position $\pi \in \mathcal{P}os(s)$, and a substitution $\sigma$ such that $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$. Here, $\ell\sigma$ is called the* redex *of the rewrite step.*

*For two TRSs $\mathcal{R}$ and $\mathcal{S}$, $\mathcal{R}/\mathcal{S}$ is a* relative TRS*, and its* rewrite relation *$\to_{\mathcal{R}/\mathcal{S}}$ is $\to_{\mathcal{S}}^* \circ \to_{\mathcal{R}} \circ \to_{\mathcal{S}}^*$, i.e., it allows rewriting with $\mathcal{S}$ before and after each $\mathcal{R}$-step. We define the* innermost *rewrite relation as $s \xrightarrow{\text{i}}_{\mathcal{R}/\mathcal{S}} t$ iff $s \to_{\mathcal{S}}^* s' \to_{\mathcal{R}} s'' \to_{\mathcal{S}}^* t$ for some terms $s', s''$, where the proper subterms of the redexes of each step with $\to_{\mathcal{S}}$ or $\to_{\mathcal{R}}$ are in normal form w.r.t. $\mathcal{R} \cup \mathcal{S}$. We write $\xrightarrow{\text{i}}_{\mathcal{R}}$ instead of $\xrightarrow{\text{i}}_{\mathcal{R}/\varnothing}$.*

*$\Sigma_d^{\mathcal{R}\cup\mathcal{S}} = \{\text{root}(\ell) \mid \ell \to r \in \mathcal{R}\cup\mathcal{S}\}$ and $\Sigma_c^{\mathcal{R}\cup\mathcal{S}} = \Sigma \setminus \Sigma_d^{\mathcal{R}\cup\mathcal{S}}$ are the* defined *(resp.* constructor*) symbols of $\mathcal{R}/\mathcal{S}$. A term $f(t_1, \ldots, t_k)$ is* basic *iff $f \in \Sigma_d^{\mathcal{R}\cup\mathcal{S}}$ and $t_1, \ldots, t_k \in \mathcal{T}(\Sigma_c^{\mathcal{R}\cup\mathcal{S}}, \mathcal{V})$. $\mathcal{R}/\mathcal{S}$ is a* constructor system *iff $\ell$ is basic for all $\ell \to r \in \mathcal{R}\cup\mathcal{S}$.*

In this paper, we will restrict ourselves to the analysis of constructor systems.

*Example 2. The following rules implement the* insertion sort *algorithm.*

$$\mathsf{isort}(\mathsf{nil}, ys) \to ys \quad (1)$$
$$\mathsf{isort}(\mathsf{cons}(x, xs), ys) \to \mathsf{isort}(xs, \mathsf{ins}(x, ys)) \quad (2)$$
$$\mathsf{ins}(x, \mathsf{nil}) \to \mathsf{cons}(x, \mathsf{nil}) \quad (3)$$
$$\mathsf{ins}(x, \mathsf{cons}(y, ys)) \to \mathsf{if}(\mathsf{gt}(x, y), x, \mathsf{cons}(y, ys)) \quad (4)$$
$$\mathsf{if}(\mathsf{true}, x, \mathsf{cons}(y, ys)) \to \mathsf{cons}(y, \mathsf{ins}(x, ys)) \quad (5)$$
$$\mathsf{if}(\mathsf{false}, x, \mathsf{cons}(y, ys)) \to \mathsf{cons}(x, \mathsf{cons}(y, ys)) \quad (6)$$
$$\mathsf{gt}(0, y) \to \mathsf{false} \quad (7)$$
$$\mathsf{gt}(\mathsf{s}(x), 0) \to \mathsf{true} \quad (8)$$
$$\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{gt}(x, y) \quad (9)$$

*Relative rules are useful to model built-in operations in programming languages since applications of these rules are disregarded for the complexity of a TRS. For example, the translation from RAML programs [27] to term rewriting*

*in [8] uses relative rules to model the semantics of comparisons and similar operations on RAML's primitive data types. Thus, we decompose the rules above into a relative TRS* $\mathcal{R}/\mathcal{S}$ *with* $\mathcal{R} = \{(1), \ldots, (6)\}$ *and* $\mathcal{S} = \{(7), (8), (9)\}$.[4]

*In our example, we have* $\Sigma_d^{\mathcal{R}\cup\mathcal{S}} = \{\mathsf{isort}, \mathsf{ins}, \mathsf{if}, \mathsf{gt}\}$ *and* $\Sigma_c^{\mathcal{R}\cup\mathcal{S}} = \{\mathsf{cons}, \mathsf{nil}, \mathsf{s}, \mathsf{0}, \mathsf{true}, \mathsf{false}\}$. *Since all left-hand sides are basic,* $\mathcal{R}/\mathcal{S}$ *is a constructor system. An example rewrite sequence to sort the list* $[2, 0]$ *is*

$$\begin{aligned} t = \mathsf{isort}(\mathsf{cons}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{cons}(\mathsf{0}, \mathsf{nil})), \mathsf{nil}) &\xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{isort}(\mathsf{cons}(\mathsf{0}, \mathsf{nil}), \mathsf{ins}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{nil})) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \\ \mathsf{isort}(\mathsf{cons}(\mathsf{0}, \mathsf{nil}), \mathsf{cons}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{nil})) &\xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{isort}(\mathsf{nil}, \mathsf{ins}(\mathsf{0}, \mathsf{cons}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{nil}))) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \\ \mathsf{isort}(\mathsf{nil}, \mathsf{if}(\mathsf{gt}(\mathsf{0}, \mathsf{s}(\mathsf{s}(\mathsf{0}))), \ldots, \ldots)) &\xrightarrow{\mathrm{i}}_{\mathcal{S}} \mathsf{isort}(\mathsf{nil}, \mathsf{if}(\mathsf{false}, \ldots, \ldots)) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \\ \mathsf{isort}(\mathsf{nil}, \mathsf{cons}(\mathsf{0}, \mathsf{cons}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{nil}))) &\xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{cons}(\mathsf{0}, \mathsf{cons}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{nil})) \end{aligned}$$

Note that ordinary TRSs are a special case of relative TRSs (where $\mathcal{S} = \varnothing$). We usually just write "TRSs" to denote "relative TRSs". We now define the *runtime complexity* of a TRS $\mathcal{R}/\mathcal{S}$. In Def. 3, $\omega$ is the smallest infinite ordinal, i.e., $\omega > e$ holds for all $e \in \mathbb{N}$, and for any $M \subseteq \mathbb{N} \cup \{\omega\}$, $\sup M$ is the least upper bound of $M$, where $\sup \varnothing = 0$.

**Definition 3 (Innermost Runtime Complexity [24,25,32,36]).** *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.,* $\mathrm{dh}(t, \to) = \sup\{e \mid \exists\, t' \in \mathcal{T}(\Sigma, \mathcal{V}).\ t \to^e t'\}$. *If* $t$ *starts an infinite* $\to$*-sequence, this yields* $\mathrm{dh}(t, \to) = \omega$. *The* innermost runtime complexity *function* $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}$ *maps any* $n \in \mathbb{N}$ *to the length of the longest sequence of* $\xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}$*-steps starting with a basic term whose size is at most* $n$*, i.e.,* $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}(n) = \sup\{\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}) \mid t \text{ is basic}, |t| \leq n\}$.

*Example 4. The rewrite sequence for* $t$ *in Ex. 2 is maximal, and thus,* $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}) = 6$. *So the* $\xrightarrow{\mathrm{i}}_{\mathcal{S}}$*-step does not contribute to* $t$*'s derivation height. As* $|t| = 9$*, this implies* $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}(9) \geq 6$. *We will show how our new approach proves* $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}(n) \in \mathcal{O}(n^2)$ *automatically.*

## 3 From TRSs to Recursive Natural Transition Systems

We now reduce complexity analysis of TRSs to complexity analysis of *recursive natural transition systems* (RNTSs). In contrast to term rewriting, RNTSs offer built-in support for arithmetic, but disallow pattern matching. To analyze TRSs, it suffices to regard RNTSs where all variables range over $\mathbb{N}$. We use the signature $\Sigma_{\mathsf{exp}} = \{+, \cdot\} \cup \mathbb{N}$ for arithmetic expressions and $\Sigma_{\mathsf{fml}} = \Sigma_{\mathsf{exp}} \cup \{\mathsf{true}, \mathsf{false}, <, \wedge\}$ for arithmetic formulas ("constraints"). We will also use relations like $=$ in constraints, but these are just syntactic sugar. To extend the rewrite relation with semantics for these symbols, let $[\![.]\!]$ evaluate all arithmetic and Boolean expressions in a term. So for example, $[\![\mathsf{gt}(1 + 2, 5 + y)]\!] = \mathsf{gt}(3, 5 + y)$ and

[4] In this way, the complexity of gt is 0, whereas comparisons have complexity 1 with the slightly more complicated encoding from [8]. Since this difference does not affect the asymptotic complexity of Ex. 2, we use the simpler encoding for the sake of readability.

$[\![3 > 5 \wedge \mathsf{true}]\!] = \mathsf{false}$. We allow substitutions with infinite domains and call $\sigma$ a *natural substitution* iff $\sigma(x) \in \mathbb{N}$ for all $x \in \mathcal{V}$.

**Definition 5 (Recursive Natural Transition System).** *An* RNTS *over a finite signature $\Sigma$ with $\Sigma \cap \Sigma_{\mathsf{fml}} = \varnothing$ is a set of rules $\mathcal{P} = \{\ell_1 \xrightarrow{w_1} r_1\ [\varphi_1], \ldots, \ell_n \xrightarrow{w_n} r_n\ [\varphi_n]\}$ with $\ell_i = f(x_1, \ldots, x_k)$ for $f \in \Sigma$ and pairwise different variables $x_1, \ldots, x_k$, $r_i \in \mathcal{T}(\Sigma \uplus \Sigma_{\mathsf{exp}}, \mathcal{V})$, constraints $\varphi_i \in \mathcal{T}(\Sigma_{\mathsf{fml}}, \mathcal{V})$, and weights $w_i \in \mathcal{T}(\Sigma_{\mathsf{exp}}, \mathcal{V})$. An RNTS $\mathcal{P}$ induces a rewrite relation $\xrightarrow{m}_{\mathcal{P}}$ on ground terms from $\mathcal{T}(\Sigma \uplus \Sigma_{\mathsf{exp}}, \varnothing)$, where $s \xrightarrow{m}_{\mathcal{P}} t$ iff there are $\ell \xrightarrow{w} r\ [\varphi] \in \mathcal{P}$, $\pi \in \mathcal{P}os(s)$, and a natural substitution $\sigma$ such that $s|_\pi = \ell\sigma$, $[\![\varphi\sigma]\!] = \mathsf{true}$, $m = [\![w\sigma]\!] \in \mathbb{N}$, and $t = [\![s[r\sigma]_\pi]\!]$. We sometimes just write $s \to_{\mathcal{P}} t$ instead of $s \xrightarrow{m}_{\mathcal{P}} t$. Again, let $\Sigma_d^{\mathcal{P}} = \{\mathrm{root}(\ell) \mid \ell \xrightarrow{w} r\ [\varphi] \in \mathcal{P}\}$ and $\Sigma_c^{\mathcal{P}} = \Sigma \setminus \Sigma_d^{\mathcal{P}}$.*

*A term $f(n_1, \ldots, n_k)$ with $f \in \Sigma$ and $n_1, \ldots, n_k \in \mathbb{N}$ is* nat-basic*, and its* size *is $\|f(n_1, \ldots, n_k)\| = 1 + n_1 + \ldots + n_k$. To consider weights for derivation heights, we define $\mathrm{dhw}(t, \to_{\mathcal{P}})$ to be the maximum weight of any $\to_{\mathcal{P}}$-sequence starting with $t$, i.e., $\mathrm{dhw}(t_0, \to_{\mathcal{P}}) = \sup\{\sum_{i=1}^{e} m_i \mid \exists t_1, \ldots, t_e \in \mathcal{T}(\Sigma \uplus \Sigma_{\mathsf{exp}}, \varnothing).\ t_0 \xrightarrow{m_1}_{\mathcal{P}} \ldots \xrightarrow{m_e}_{\mathcal{P}} t_e\}$. Then $\mathrm{irc}_{\mathcal{P}}$ maps $n \in \mathbb{N}$ to the maximum weight of any $\to_{\mathcal{P}}$-sequence starting with a nat-basic term whose size is at most $n$, i.e., $\mathrm{irc}_{\mathcal{P}}(n) = \sup\{\mathrm{dhw}(t, \to_{\mathcal{P}}) \mid t$ is nat-basic, $\|t\| \le n\}$.*

Note that the rewrite relation for RNTSs is "innermost" by construction, as rules do not contain symbols from $\Sigma$ below the root in left-hand sides, and they are only applicable if all variables are instantiated by numbers.

The crucial idea of our approach is to model the behavior of a TRS by a corresponding RNTS which results from abstracting constructor terms to their size. Thus, we use the following transformation $\wr\cdot\wr$ from TRSs to RNTSs.

**Definition 6 (Abstraction $\wr\cdot\wr$ from TRSs to RNTSs).** *For a TRS $\mathcal{R}/\mathcal{S}$, the* size abstraction *$\wr t\wr$ of a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ is defined as follows:*

$$\begin{array}{rcll} \wr x\wr & = & x & \text{for } x \in \mathcal{V} \\ \wr f(t_1, \ldots, t_k)\wr & = & 1 + \wr t_1\wr + \ldots + \wr t_k\wr & \text{if } f \in \Sigma_c^{\mathcal{R}\cup\mathcal{S}} \\ \wr f(t_1, \ldots, t_k)\wr & = & f(\wr t_1\wr, \ldots, \wr t_k\wr) & \text{if } f \in \Sigma_d^{\mathcal{R}\cup\mathcal{S}} \end{array}$$

*We lift $\wr\cdot\wr$ to rules with basic left-hand sides. For $\ell = f(t_1, \ldots, t_k)$ with $t_1, \ldots, t_k \in \mathcal{T}(\Sigma_c^{\mathcal{R}\cup\mathcal{S}}, \mathcal{V})$ and $w \in \mathcal{T}(\Sigma_{\mathsf{exp}}, \mathcal{V})$, we define*

$$\wr \ell \to r \wr_w \quad = \quad f(x_1, \ldots, x_k) \xrightarrow{w} \wr r\wr \quad \left[\bigwedge\nolimits_{i=1}^{k} x_i = \wr t_i\wr \ \wedge\ \bigwedge\nolimits_{x \in \mathcal{V}(\ell)} x \ge 1\right]$$

*for pairwise different fresh variables $x_1, \ldots, x_k$. For a constructor system $\mathcal{R}/\mathcal{S}$, we define the RNTS $\wr\mathcal{R}/\mathcal{S}\wr = \{\wr \ell \to r \wr_1 \mid \ell \to r \in \mathcal{R}\} \cup \{\wr \ell \to r \wr_0 \mid \ell \to r \in \mathcal{S}\}$.*

*Example 7. For the TRS $\mathcal{R}/\mathcal{S}$ from Ex. 2, $\wr\mathcal{R}/\mathcal{S}\wr$ corresponds to the following RNTS.*

$$\begin{array}{rcll}
\mathsf{isort}(xs, ys) & \xrightarrow{1} & ys & [xs \ = 1 \wedge \ldots] \quad (1') \\
\mathsf{isort}(xs', ys) & \xrightarrow{1} & \mathsf{isort}(xs, \mathsf{ins}(x, ys)) & [xs' = 1 + x + xs \wedge \ldots] \quad (2') \\
\mathsf{ins}(x, ys) & \xrightarrow{1} & 2 + x & [ys \ = 1 \wedge \ldots] \quad (3') \\
\mathsf{ins}(x, ys') & \xrightarrow{1} & \mathsf{if}(\mathsf{gt}(x, y), x, ys') & [ys' = 1 + y + ys \wedge \ldots] \quad (4') \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + y + \mathsf{ins}(x, ys) & [b = 1 \wedge ys' = 1 + y + ys \wedge \ldots] \quad (5') \\
\mathsf{if}(b, x, ys') & \xrightarrow{1} & 1 + x + ys' & [b = 1 \wedge ys' = 1 + y + ys \wedge \ldots] \quad (6') \\
\mathsf{gt}(x, y) & \xrightarrow{0} & 1 & [x \ = 1 \wedge \ldots] \quad (7') \\
\mathsf{gt}(x', y) & \xrightarrow{0} & 1 & [x' = 1 + x \wedge y = 1 \wedge \ldots] \quad (8') \\
\mathsf{gt}(x', y') & \xrightarrow{0} & \mathsf{gt}(x, y) & [x' = 1 + x \wedge y' = 1 + y \wedge \ldots] \quad (9')
\end{array}$$

*In these rules, "$\wedge\ldots$" stands for the constraint that all variables have to be instantiated with values $\geq 1$. Note that we make use of fresh variables like $x$ and $xs$ on the right-hand side of $(2')$ to simulate matching of constructor terms. Using this RNTS, the rewrite steps in Ex. 2 can be simulated as follows.*

$$\begin{array}{rcl}
t' = \mathsf{isort}(7,1) & \xrightarrow{1} & \mathsf{isort}(3, \mathsf{ins}(3,1)) \xrightarrow{1} \mathsf{isort}(3,5) \\
\xrightarrow{1} \mathsf{isort}(1, \mathsf{ins}(1,5)) & \xrightarrow{1} & \mathsf{isort}(1, \mathsf{if}(\mathsf{gt}(1,3),1,5)) \xrightarrow{0} \mathsf{isort}(1, \mathsf{if}(1,1,5)) \\
\xrightarrow{1} \mathsf{isort}(1,7) & \xrightarrow{1} & 7
\end{array}$$

*For the nat-basic term $t'$, we have $\|t'\| = 1 + 7 + 1 = 9$. So the above sequence proves* $\mathrm{dhw}(t', \to_{\mathcal{P}}) \geq 6$ *and hence,* $\mathrm{irc}_{\mathcal{P}}(9) \geq 6$. *Note that unlike Ex. 2, here rewriting nat-basic terms is non-deterministic as, e.g., we also have* $\mathsf{isort}(7,1) \xrightarrow{1} \mathsf{isort}(2, \mathsf{ins}(4,1))$. *The reason is that $\wr\cdot\wr$ is a* blind abstraction *[10], which abstracts several different terms to the same number.*

$[\![\wr\cdot\wr]\!]$ maps basic ground terms to nat-basic terms, e.g., $[\![\wr\mathsf{ins}(\mathsf{s}(\mathsf{0}), \mathsf{nil})\wr]\!] = [\![\mathsf{ins}(1{+}1, 1)]\!] = \mathsf{ins}(2,1)$. We now show that under certain conditions, $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}) \leq \mathrm{dhw}([\![\wr t\wr]\!], \to_{\wr\mathcal{R}/\mathcal{S}\wr})$ holds for all ground terms $t$, i.e., rewrite sequences of a TRS $\mathcal{R}/\mathcal{S}$ can be simulated in the RNTS $\wr\mathcal{R}/\mathcal{S}\wr$ resulting from its transformation. We would like to conclude that in these cases, we also have $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}(n) \leq \mathrm{irc}_{\wr\mathcal{R}/\mathcal{S}\wr}(n)$. However, irc considers arbitrary (basic) terms, but the above connection between the derivation heights of $t$ and $[\![\wr t\wr]\!]$ only holds for *ground* terms $t$. For *full* rewriting, we clearly have $\mathrm{dh}(t, \to_{\mathcal{R}}) \leq \mathrm{dh}(t\sigma, \to_{\mathcal{R}})$ for any substitution $\sigma$. However, this does not hold for *innermost* rewriting. For example, $\mathsf{f}(\mathsf{g}(x))$ has an infinite innermost reduction with the TRS $\{\mathsf{f}(\mathsf{g}(x)) \to \mathsf{f}(\mathsf{g}(x)), \mathsf{g}(\mathsf{a}) \to \mathsf{a}\}$, but $\mathsf{f}(\mathsf{g}(\mathsf{a}))$ is innermost terminating. Nevertheless, we show in Thm. 9 that for *constructor systems* $\mathcal{R}$, $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \mathrm{dh}(t\sigma, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$ holds for any ground substitution $\sigma$.

However, for *relative* rewriting with constructor systems $\mathcal{R}$ and $\mathcal{S}$, $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}) \leq \mathrm{dh}(t\sigma, \xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}})$ does not necessarily hold if $\mathcal{S}$ is not innermost terminating. To see this, consider $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{f}(x)\}$ and $\mathcal{S} = \{\mathsf{g}(\mathsf{a}) \to \mathsf{g}(\mathsf{a})\}$. Now $\mathsf{f}(\mathsf{g}(x))$ has an infinite reduction w.r.t. $\xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}$ since $\mathsf{g}(x)$ is a normal form w.r.t. $\mathcal{R} \cup \mathcal{S}$. However, its instance $\mathsf{f}(\mathsf{g}(\mathsf{a}))$ has the derivation height 0 w.r.t. $\xrightarrow{\mathrm{i}}_{\mathcal{R}/\mathcal{S}}$, as $\mathsf{g}(\mathsf{a})$ is not innermost terminating w.r.t. $\mathcal{S}$ and no rule of $\mathcal{R}$ can ever be applied. To solve this problem, we extend the TRS $\mathcal{S}$ by a *terminating variant* $\mathcal{N}$.

**Definition 8 (Terminating Variant).** *A TRS $\mathcal{N}$ is a* terminating variant *of $\mathcal{S}$ iff $\xrightarrow{i}_{\mathcal{N}}$ terminates and every $\mathcal{N}$-normal form is also an $\mathcal{S}$-normal form.*

So if one can prove innermost termination of $\mathcal{S}$, then one can use $\mathcal{S}$ as a terminating variant of itself. For instance in Ex. 2, termination of $\mathcal{S} = \{(7), (8), (9)\}$ can easily be shown automatically by standard tools like AProVE [22]. Otherwise, one can for instance use a terminating variant $\{f(x_1, \ldots, x_k) \to t_f \mid f \in \Sigma_d^{\mathcal{S}}\}$ where for each $f$, we pick some constructor ground term $t_f \in \mathcal{T}(\Sigma_c^{\mathcal{R} \cup \mathcal{S}}, \varnothing)$. Now one can prove that for innermost (relative) rewriting, the derivation height of a term does not decrease when it is instantiated by a ground substitution.

**Theorem 9 (Soundness of Instantiation and Terminating Variants).** *Let $\mathcal{R}$, $\mathcal{S}$ be constructor systems and $\mathcal{N}$ be a terminating variant of $\mathcal{S}$. Then $\mathrm{dh}(t, \xrightarrow{i}_{\mathcal{R}/\mathcal{S}}) \leq \mathrm{dh}(t\sigma, \xrightarrow{i}_{\mathcal{R}/(\mathcal{S} \cup \mathcal{N})})$ holds for any term $t$ where $t\sigma$ is ground.*

However, the restriction to ground terms $t$ still does not ensure $\mathrm{dh}(t, \xrightarrow{i}_{\mathcal{R}/\mathcal{S}}) \leq \mathrm{dhw}(\llbracket \wr t \wr \rrbracket, \to_{\wr \mathcal{R}/\mathcal{S} \wr})$. The problem is that $\xrightarrow{i}_{\mathcal{R}/\mathcal{S}}$ can rewrite a term $t$ at position $\pi$ also if there is a defined symbol below $t|_\pi$ as long as no rule can be applied to that subterm. So for Ex. 2, we have $\mathsf{isort}(\mathsf{nil}, \mathsf{if}(\mathsf{true}, \mathsf{0}, \mathsf{nil})) \xrightarrow{i}_{\mathcal{R}} \mathsf{if}(\mathsf{true}, \mathsf{0}, \mathsf{nil})$, but $\wr \mathcal{R}/\mathcal{S} \wr$ cannot rewrite $\llbracket \wr \mathsf{isort}(\mathsf{nil}, \mathsf{if}(\mathsf{true}, \mathsf{0}, \mathsf{nil})) \wr \rrbracket = \mathsf{isort}(1, \mathsf{if}(1, 1, 1))$ since the $\mathsf{if}$-rules of $\wr \mathcal{R}/\mathcal{S} \wr$ may be applied only if the third argument is $\geq 3$, and the variables in the $\mathsf{isort}$-rule may be instantiated only by numbers (not by normal forms like $\mathsf{if}(1, 1, 1)$). This problem can be solved by requiring that $\mathcal{R}/\mathcal{S}$ is *completely defined*, i.e., that $\mathcal{R} \cup \mathcal{S}$ can rewrite every basic ground term. However, this is too restrictive as we, e.g., would like $\mathsf{gt}(\mathsf{true}, \mathsf{false})$ to be in normal form. Fortunately, (innermost) runtime complexity is *persistent* w.r.t. type introduction [6]. Thus, we only need to ensure that every *well-typed* basic ground term can be rewritten.

**Definition 10 (Typed TRSs (cf. e.g. [21, 37])).** *In a many-sorted (first-order monomorphic) signature $\Sigma$ over the set of types $Ty$, every symbol $f \in \Sigma$ has a type of the form $\tau_1 \times \ldots \times \tau_k \to \tau$ with $\tau_1, \ldots, \tau_k, \tau \in Ty$. Moreover, every variable has a type from $Ty$, and we assume that $\mathcal{V}$ contains infinitely many variables of every type in $Ty$. We call $t \in \mathcal{T}(\Sigma, \mathcal{V})$ a* well-typed *term of type $\tau$ iff either $t \in \mathcal{V}$ is a variable of type $\tau$ or $t = f(t_1, \ldots, t_k)$ where $f$ has the type $\tau_1 \times \ldots \times \tau_k \to \tau$ and each $t_i$ is a well-typed term of type $\tau_i$.*

*A rewrite rule $\ell \to r$ is well typed iff $\ell$ and $r$ are well-typed terms of the same type. A TRS $\mathcal{R}/\mathcal{S}$ is well typed iff all rules of $\mathcal{R} \cup \mathcal{S}$ are well typed. (W.l.o.g., here one may rename the variables in every rule. Then it is not a problem if the variable $x$ is used with type $\tau_1$ in one rule and with type $\tau_2$ in another rule.)*

*Example 11. For any TRS $\mathcal{R}/\mathcal{S}$, standard algorithms can compute a type assignment to make $\mathcal{R}/\mathcal{S}$ well typed (and to decompose the terms into as many types as possible). For the TRS from Ex. 2 we obtain the following type assignment. Note that for this type assignment the TRS is* not *completely defined since $\mathsf{if}(\mathsf{true}, \mathsf{0}, \mathsf{nil})$ is a well-typed basic ground term in normal form w.r.t. $\mathcal{R} \cup \mathcal{S}$.*

| | | |
|---|---|---|
| $\mathsf{isort} :: \mathbf{List} \times \mathbf{List} \to \mathbf{List}$ | $\mathsf{0} :: \mathbf{Nat}$ | $\mathsf{gt} :: \mathbf{Nat} \times \mathbf{Nat} \to \mathbf{Bool}$ |
| $\mathsf{ins} :: \mathbf{Nat} \times \mathbf{List} \to \mathbf{List}$ | $\mathsf{s} :: \mathbf{Nat} \to \mathbf{Nat}$ | $\mathsf{true}, \mathsf{false} :: \mathbf{Bool}$ |
| $\mathsf{if} :: \mathbf{Bool} \times \mathbf{Nat} \times \mathbf{List} \to \mathbf{List}$ | $\mathsf{nil} :: \mathbf{List}$ | $\mathsf{cons} :: \mathbf{Nat} \times \mathbf{List} \to \mathbf{List}$ |

**Definition 12 (Completely Defined).** *A well-typed TRS $\mathcal{R}/\mathcal{S}$ over a many-sorted signature with types $Ty$ is* completely defined *iff there is at least one constant for each $\tau \in Ty$ and no well-typed basic ground term in $\mathcal{R} \cup \mathcal{S}$-normal form.*

For completely defined TRSs, the transformation from TRSs to RNTSs is sound.

**Theorem 13 (Soundness of Abstraction $\wr\cdot\wr$).** *Let $\mathcal{R}/\mathcal{S}$ be a well-typed, completely defined constructor system. Then* $\mathrm{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}/\mathcal{S}}) \leq \mathrm{dhw}([\![\wr t \wr]\!], \to_{\wr \mathcal{R}/\mathcal{S} \wr})$ *holds for all well-typed ground terms $t$. Let $\mathcal{N}$ be a terminating variant of $\mathcal{S}$ such that $\mathcal{R}/(\mathcal{S} \cup \mathcal{N})$ is also well typed. If $\mathcal{R}/(\mathcal{S} \cup \mathcal{N})$ is completely defined, then we have* $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}(n) \leq \mathrm{irc}_{\wr \mathcal{R}/(\mathcal{S} \cup \mathcal{N}) \wr}(n)$ *for all $n \in \mathbb{N}$.*

As every TRS $\mathcal{R}/\mathcal{S}$ is well typed w.r.t. *some* type assignment (e.g., the one with just a single type), the only additional restriction in Thm. 13 is that the TRS has to be completely defined. This can always be achieved by extending $\mathcal{S}$ by a suitable terminating variant $\mathcal{N}$ of $\mathcal{S}$ automatically. Based on standard algorithms to detect well-typed basic ground terms $f(\ldots)$ in $(\mathcal{R} \cup \mathcal{S})$-normal form [30, 31], we add the rules $f(x_1, \ldots, x_k) \to t_f$ to $\mathcal{N}$, where again for each $f$, we choose some constructor ground term $t_f \in \mathcal{T}(\Sigma_c^{\mathcal{R} \cup \mathcal{S}}, \varnothing)$. As shown by Thm. 9, we have $\mathrm{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}/\mathcal{S}}) \leq \mathrm{dh}(t\sigma, \xrightarrow{\mathsf{i}}_{\mathcal{R}/(\mathcal{S} \cup \mathcal{N})})$ for *any* terminating variant $\mathcal{N}$, i.e., adding such rules never decreases the derivation height. So even if $\mathcal{R}/\mathcal{S}$ is not completely defined and just $\mathcal{R}/(\mathcal{S} \cup \mathcal{N})$ is completely defined, we still have $\mathrm{irc}_{\mathcal{R}/\mathcal{S}}(n) \leq \mathrm{irc}_{\mathcal{R}/(\mathcal{S} \cup \mathcal{N})}(n) \leq \mathrm{irc}_{\wr \mathcal{R}/(\mathcal{S} \cup \mathcal{N}) \wr}(n)$.

*Example 14. To make the TRS of Ex. 2 completely defined, we add rules for all defined symbols in basic ground normal forms. In this example, the only such symbol is* $\mathsf{if}$*. Hence, for instance we add* $\mathsf{if}(b, x, xs) \to \mathsf{nil}$ *to $\mathcal{S}$. The resulting TRS* $\mathcal{S} \cup \{\mathsf{if}(b, x, xs) \to \mathsf{nil}\}$ *is clearly a terminating variant of $\mathcal{S}$. Hence, to analyze complexity of the insertion sort TRS, we now extend the RNTS of Ex. 7 by*

$$\wr \mathsf{if}(b, x, xs) \to \mathsf{nil} \wr_0 \quad = \quad \mathsf{if}(b, x, xs) \xrightarrow{0} 1 \;\; [b \geq 1 \wedge x \geq 1 \wedge xs \geq 1] \tag{10}$$

## 4 Analyzing the Complexity of RNTSs

Thm. 13 allows us to reduce complexity analysis of term rewriting to the analysis of RNTSs. Our RNTSs are related to *integer transition systems* (ITSs), a formalism often used to abstract programs. The main difference is that RNTSs can model procedure calls by nested function symbols $f(\ldots g(\ldots) \ldots)$ on the right-hand side of rules, whereas ITSs may allow right-hand sides like $f(\ldots) + g(\ldots)$, but no nesting of $f, g \in \Sigma$. So ITSs cannot pass the result of one function as a parameter to another function. Note that in contrast to the usual definition of ITSs, in our setting reductions can begin with any (nat-basic) terms instead of dedicated start terms, and it suffices to regard natural instead of integer numbers. (An extension to recursive transition systems on integers would be possible by measuring the size of integers by their absolute value, as in [13].)

**Definition 15 (ITS).** *An RNTS $\mathcal{P}$ over the signature $\Sigma$ is an* ITS *iff symbols from $\Sigma$ occur only at parallel positions in right-hand sides of $\mathcal{P}$. Here, $\pi$ and $\pi'$ are* parallel *iff $\pi$ is not a prefix of $\pi'$ and $\pi'$ is not a prefix of $\pi$.*

Upper runtime complexity bounds for an ITS $\mathcal{P}$ can, for example, be inferred by generating ranking functions which decrease with each application of a rule from $\mathcal{P}$. Then, the ranking functions are multiplied with the weight of the rules.

However, many analysis techniques for ITSs (e.g., [1,4,13,34]) cannot handle the RNTSs generated from standard TRSs. Thus, we now introduce a new modular approach that allows us to apply existing tools for ITSs to analyze RNTSs. Our approach builds upon the idea of alternating between *runtime* and *size* analysis [13]. The key insight is to *summarize* procedures by approximating their runtime and the size of their result, and then to eliminate them from the program. In this way, our analysis decomposes the "call graph" of the RNTS into "blocks" of mutually recursive functions and exports each block of mutually recursive functions into a separate ITS. Thus, in each analysis step it suffices to analyze just an ITS instead of an RNTS. We use weakly monotonic runtime and size bounds from $\mathcal{T}(\Sigma_{\mathsf{exp}}, \mathcal{V})$ to compose them easily when analyzing nested terms.

**Definition 16 (Runtime and Size Bounds).** *For any terms $t_1, \dots, t_k$, let $\{x_1/t_1, \dots, x_k/t_k\}$ be the substitution $\sigma$ with $x_i\sigma = t_i$ for $1 \le i \le k$ and $y\sigma = y$ for $y \in \mathcal{V} \setminus \{x_1, \dots, x_k\}$. Then $\mathsf{rt} : \Sigma \to \mathcal{T}(\Sigma_{\mathsf{exp}}, \mathcal{V}) \cup \{\omega\}$ is a* runtime bound *for an RNTS $\mathcal{P}$ iff we have $\mathrm{dhw}(f(n_1, \dots, n_k), \to_{\mathcal{P}}) \le [\![\mathsf{rt}(f)\,\{x_1/n_1, \dots, x_k/n_k\}]\!]$ for all $n_1, \dots, n_k \in \mathbb{N}$ and all $f \in \Sigma$. Similarly, $\mathsf{sz} : \Sigma \to \mathcal{T}(\Sigma_{\mathsf{exp}}, \mathcal{V}) \cup \{\omega\}$ is a* size bound *for $\mathcal{P}$ iff $n \le [\![\mathsf{sz}(f)\,\{x_1/n_1, \dots, x_k/n_k\}]\!]$ for all $n_1, \dots, n_k \in \mathbb{N}$, all $f \in \Sigma$, and all $n \in \mathbb{N}$ with $f(n_1, \dots, n_k) \to_{\mathcal{P}}^* n$.*

*Example 17. For the RNTS $\{(1'), \dots, (9'), (10)\}$ from Ex. 14, any function $\mathsf{rt}$ with $\mathsf{rt}(\mathsf{isort}) \ge \lfloor \frac{x_1 - 1}{2} \rfloor \cdot x_2 + 1$, $\mathsf{rt}(\mathsf{ins}) \ge x_2$, $\mathsf{rt}(\mathsf{if}) \ge x_3 - 1$, and $\mathsf{rt}(\mathsf{gt}) \ge 0$ is a runtime bound (recall that the $\mathsf{gt}$-rules have weight 0). Similarly, any $\mathsf{sz}$ with $\mathsf{sz}(\mathsf{isort}) \ge x_1 + x_2 - 1$, $\mathsf{sz}(\mathsf{ins}) \ge x_1 + x_2 + 1$, $\mathsf{sz}(\mathsf{if}) \ge x_2 + x_3 + 1$, $\mathsf{sz}(\mathsf{gt}) \ge 1$ is a size bound.*

A runtime bound clearly gives rise to an upper bound on the runtime complexity.

**Theorem 18 ($\mathsf{rt}$ and $\mathrm{irc}$).** *Let $\mathsf{rt}$ be a runtime bound for an RNTS $\mathcal{P}$. Then for all $n \in \mathbb{N}$, we have $\mathrm{irc}_{\mathcal{P}}(n) \le \sup\{[\![\mathsf{rt}(f)\,\{x_1/n_1, \dots, x_k/n_k\}]\!] \mid f \in \Sigma, n_1, ..., n_k \in \mathbb{N}, \sum_{i=1}^{k} n_i < n\}$. So in particular, $\mathrm{irc}_{\mathcal{P}}(n) \in \mathcal{O}(\sum_{f \in \Sigma} [\![\mathsf{rt}(f)\,\{x_1/n, \dots, x_k/n\}]\!])$.*

Thus, a suitable runtime bound $\mathsf{rt}$ for the RNTS $\{(1'), \dots, (9'), (10)\}$ yields $\mathrm{irc}(n) \in \mathcal{O}(n^2)$, cf. Ex. 17. In Sect. 4.2 we present a new technique to infer runtime and size bounds $\mathsf{rt}$ and $\mathsf{sz}$ automatically with existing complexity tools for ITSs. As these tools usually return only runtime bounds, Sect. 4.1 shows how they can also be used to generate size bounds.

### 4.1 Size Bounds as Runtime Bounds

We first present a transformation for a large class of ITSs that lets us obtain size bounds from any method that can infer runtime bounds. The transformation extends each function symbol from $\Sigma$ by an additional accumulator argument. Then terms that are multiplied with the result of a function are collected in the accumulator. Terms that are added to the result are moved to the weight of the rule.

**Theorem 19 (ITS Size Bounds).** *Let $\mathcal{P}$ be an ITS whose rules are of the form $\ell \xrightarrow{w} u + v \cdot r\,[\varphi]$ or $\ell \xrightarrow{w} u\,[\varphi]$ with $u, v \in \mathcal{T}(\Sigma_{\mathsf{exp}}, \mathcal{V})$ and $\mathrm{root}(r) \in \Sigma$. Let $\mathcal{P}_{\mathsf{size}} =$*

$$\begin{array}{rl} \{f'(x_1,...,x_k,z) \xrightarrow{u\cdot z} g'(t_1,...,t_n, v\cdot z)\,[\varphi] \mid f(x_1,...,x_k) \xrightarrow{w} u + v\cdot g(t_1,...,t_n)\,[\varphi] \in \mathcal{P}\} \\ \cup \quad \{f'(x_1,...,x_k,z) \xrightarrow{u\cdot z} 0\ [\varphi] \mid f(x_1,...,x_k) \xrightarrow{w} u\ [\varphi] \in \mathcal{P}\} \end{array}$$

*for a fresh variable $z \in \mathcal{V}$. Let $\mathsf{rt}$ be a runtime bound for $\mathcal{P}_{\mathsf{size}}$. Then $\mathsf{sz}$ with $\mathsf{sz}(f) = \mathsf{rt}(f')\{x_{k+1}/1\}$ for any $f \in \Sigma$ is a size bound for $\mathcal{P}$.*

Thm. 19 can be generalized to right-hand sides like $f(x)+2\cdot g(y)$ with $f, g \in \Sigma$, cf. [5]. However, it is not applicable if the results of function calls are multiplied on right-hand sides (e.g., $f(x) \cdot g(y)$) and our technique fails in such cases.

*Example 20. To get a size bound for $\mathcal{P}^{\mathsf{gt}} = \{(7'), (8'), (9')\}$, we construct $\mathcal{P}^{\mathsf{gt}}_{\mathsf{size}}$:*

$$\begin{array}{ll} \mathsf{gt}'(x,y,z) \xrightarrow{z} 0\ [x = 1 \wedge \ldots] & \mathsf{gt}'(x',y,z) \xrightarrow{z} 0\ [x' = 1 + x \wedge y = 1 \wedge \ldots] \\ \mathsf{gt}'(x',y',z) \xrightarrow{0} \mathsf{gt}'(x,y,z)\,[x' = 1 + x \wedge y' = 1 + y \wedge \ldots] & \end{array}$$

*Existing ITS tools can compute a runtime bound like $\mathsf{rt}(\mathsf{gt}') = x_3$ for $\mathcal{P}^{\mathsf{gt}}_{\mathsf{size}}$. Hence, by Thm. 19 we obtain the size bound $\mathsf{sz}$ for $\mathcal{P}^{\mathsf{gt}}$ with $\mathsf{sz}(\mathsf{gt}) = \mathsf{rt}(\mathsf{gt}')\{x_3/1\} = 1$.*

### 4.2 Complexity Bounds for Recursive Programs

Now we show how complexity tools for ITSs can be used to infer runtime and size bounds for RNTSs. We first define a *call-graph* relation $\sqsupset$ to determine in which order we analyze symbols of $\Sigma$. Essentially, $f \sqsupset g$ holds iff $f(\ldots)$ rewrites to a term containing $g$.

**Definition 21 ($\sqsupset$).** *For an RNTS $\mathcal{P}$, the* call-graph relation $\sqsupset$ *is the transitive closure of $\{(\mathrm{root}(\ell), g) \mid \ell \xrightarrow{w} r[\varphi] \in \mathcal{P}, g \in \Sigma \text{ occurs in } r\}$. An RNTS has* nested recursion *iff it has a rule $\ell \xrightarrow{w} r\,[\varphi]$ with $\mathrm{root}(r|_\pi) \sqsupset \mathrm{root}(\ell)$ and $\mathrm{root}(r|_{\pi'}) \sqsupset \mathrm{root}(\ell)$ for positions $\pi < \pi'$. As usual, $\pi < \pi'$ means that $\pi$ is a proper prefix of $\pi'$ (i.e., that $\pi'$ is* strictly below *$\pi$). A symbol $f \in \Sigma^{\mathcal{P}}_d$ is a* bottom symbol *iff $f \sqsupset g$ implies $g \sqsupset f$ for all $g \in \Sigma^{\mathcal{P}}_d$. The sub-RNTS of $\mathcal{P}$* induced *by $f$ is $\mathcal{P}^f = \{\ell \xrightarrow{w} r[\varphi] \in \mathcal{P} \mid f \sqsupseteq \mathrm{root}(\ell)\}$, where $\sqsupseteq$ is the reflexive closure of $\sqsupset$.*

*Example 22. For the RNTS $\mathcal{P}$ from Ex. 14 and 17, we have $\mathsf{isort} \sqsupset \mathsf{ins} \sqsupset \mathsf{if} \sqsupset \mathsf{ins} \sqsupset \mathsf{gt}$. The only bottom symbol is $\mathsf{gt}$. It induces the sub-RNTS $\mathcal{P}^{\mathsf{gt}} = \{(7'), (8'), (9')\}$, $\mathsf{ins}$ induces $\{(3'), \ldots, (9'), (10)\}$, and $\mathsf{isort}$ induces the full RNTS of Ex. 14.*

Our approach cannot handle programs like $f(\ldots) \to f(\ldots f(\ldots)\ldots)$ with nested recursion, but such programs rarely occur in practice. To compute bounds for an RNTS $\mathcal{P}$ without nested recursion, we start with the trivial bounds $\mathsf{rt}(f) = \mathsf{sz}(f) = \omega$ for all $f \in \Sigma^{\mathcal{P}}_d$. In each step, we analyze the sub-RNTS $\mathcal{P}^f$ induced by a bottom symbol $f$ and refine $\mathsf{rt}$ and $\mathsf{sz}$ for all defined symbols of $\mathcal{P}^f$. Afterwards we remove the rules $\mathcal{P}^f$ from $\mathcal{P}$ and continue with the next bottom symbol. By this removal of rules, the former defined symbol $f$ becomes a constructor, and former non-bottom symbols are turned into bottom symbols.

**Algorithm 1** Computing Runtime and Size Bounds for RNTSs

1 Let $\mathsf{rt}(f) := \mathsf{sz}(f) := \omega$ for each $f \in \Sigma_d^{\mathcal{P}}$ and $\mathsf{rt}(f) := \mathsf{sz}(f) := 0$ for each $f \in \Sigma_c^{\mathcal{P}}$.
2 If $\mathcal{P}$ has nested recursion, then return $\mathsf{rt}$ and $\mathsf{sz}$.
3 While $\mathcal{P}$ is not empty:
  3.1 Choose a bottom symbol $f$ of $\mathcal{P}$ and let $\mathcal{P}^f$ be the sub-RNTS induced by $f$.
  3.2 Construct $\mathcal{P}^f_{\mathsf{sz}}$ according to Thm. 27 and $(\mathcal{P}^f_{\mathsf{sz}})_{\mathsf{size}}$ according to Thm. 19 (resp. its generalization) if possible, otherwise return $\mathsf{rt}$ and $\mathsf{sz}$.
  3.3 Compute a runtime bound for $(\mathcal{P}^f_{\mathsf{sz}})_{\mathsf{size}}$ using existing ITS tools and let $\mathsf{sz}_f$ be this bound (cf. Thm. 19).
  3.4 For each $g \in \Sigma_d^{\mathcal{P}^f}$, let $\mathsf{sz}(g) := \mathsf{sz}_f(g)$.
  3.5 Construct $\mathcal{P}^f_{\mathsf{rt},\mathsf{sz}}$ according to Thm. 27.
  3.6 Compute a runtime bound $\mathsf{rt}_f$ for $\mathcal{P}^f_{\mathsf{rt},\mathsf{sz}}$ using existing ITS tools.
  3.7 For each $g \in \Sigma_d^{\mathcal{P}^f}$, let $\mathsf{rt}(g) := \mathsf{rt}_f(g)$.
  3.8 Let $\mathcal{P} := \mathcal{P} \setminus \mathcal{P}^f$.
4 Return $\mathsf{rt}$ and $\mathsf{sz}$.

To analyze the RNTS $\mathcal{P}^f$, Thm. 27 will transform $\mathcal{P}^f$ into two ITSs $\mathcal{P}^f_{\mathsf{sz}}$ and $\mathcal{P}^f_{\mathsf{rt},\mathsf{sz}}$ by abstracting away calls to functions which we already analyzed. Then existing tools for ITSs can be used to compute a size resp. runtime bound for $\mathcal{P}^f_{\mathsf{sz}}$ resp. $\mathcal{P}^f_{\mathsf{rt},\mathsf{sz}}$. Our overall algorithm to infer bounds for RNTSs is summarized in Alg. 1. It clearly terminates, as every loop iteration eliminates a defined symbol (since Step 3.8 removes all rules for the currently analyzed symbol $f$).

When computing bounds for a bottom symbol $f \in \Sigma_d^{\mathcal{P}}$, we already know (weakly monotonic) size and runtime bounds for all constructors $g \in \Sigma_c^{\mathcal{P}}$. Hence to transform RNTSs into ITSs, *outer* calls of constructors $g$ in terms $g(\dots f(\dots)\dots)$ can be replaced by $\mathsf{sz}(g)$. In Def. 23, while $\mathsf{sz}(t)$ replaces *all* calls to procedures $g \in \Sigma$ in $t$ by their size bound, the *outer abstraction* $\mathfrak{a}^{\circ}_{\mathsf{sz}}(t)$ only replaces constructors $g \in \Sigma_c^{\mathcal{P}}$ by their size bound $\mathsf{sz}(g)$, provided that they do not occur below defined symbols $f \in \Sigma_d^{\mathcal{P}}$.

**Definition 23 (Outer Abstraction).** *Let $\mathcal{P}$ be an RNTS with the size bound $\mathsf{sz}$. We lift $\mathsf{sz}$ to terms by defining $\mathsf{sz}(x) = x$ for $x \in \mathcal{V}$ and*

$$\mathsf{sz}(g(s_1,\dots,s_n)) = \begin{cases} \mathsf{sz}(g)\,\{x_j/\mathsf{sz}(s_j) \mid 1 \le j \le n\} & \text{if } g \in \Sigma \\ g(\mathsf{sz}(s_1),\dots,\mathsf{sz}(s_n)) & \text{if } g \in \Sigma_{\mathsf{exp}} \end{cases}$$

*The* outer abstraction *of a term is defined as $\mathfrak{a}^{\circ}_{\mathsf{sz}}(x) = x$ for $x \in \mathcal{V}$ and*

$$\mathfrak{a}^{\circ}_{\mathsf{sz}}(g(s_1,\dots,s_n)) = \begin{cases} \mathsf{sz}(g)\,\{x_j/\mathfrak{a}^{\circ}_{\mathsf{sz}}(s_j) \mid 1 \le j \le n\} & \text{if } g \in \Sigma_c^{\mathcal{P}} \\ g(\mathfrak{a}^{\circ}_{\mathsf{sz}}(s_1),\dots,\mathfrak{a}^{\circ}_{\mathsf{sz}}(s_n)) & \text{if } g \in \Sigma_{\mathsf{exp}} \\ g(s_1,\dots,s_n) & \text{if } g \in \Sigma_d^{\mathcal{P}} \end{cases}$$

*Example 24. Consider the following variant $\mathcal{R}^{\times}$ of* `AG01/#3.16.xml` *from the TPDB*[5] *and its RNTS-counterpart $\wr\mathcal{R}^{\times}\wr$:*

[5] *Termination Problems Data Base*, the collection of examples used at the annual *Termination and Complexity Competition*, see `http://termination-portal.org`.

| $\mathcal{R}^\times$ : | $\wr\mathcal{R}^\times\int$ : | | |
|---|---|---|---|
| $\mathsf{f}_+(\mathsf{0},y) \to y$ | $\mathsf{f}_+(x,y) \xrightarrow{1} y$ | $[x = 1 \wedge \ldots]$ | (11) |
| $\mathsf{f}_+(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{f}_+(x,y))$ | $\mathsf{f}_+(x',y) \xrightarrow{1} 1+\mathsf{f}_+(x,y)$ | $[x' = x+1 \wedge \ldots]$ | (12) |
| $\mathsf{f}_\times(\mathsf{0},y) \to \mathsf{0}$ | $\mathsf{f}_\times(x,y) \xrightarrow{1} 1$ | $[x = 1 \wedge \ldots]$ | (13) |
| $\mathsf{f}_\times(\mathsf{s}(x),y) \to \mathsf{f}_+(\mathsf{f}_\times(x,y),y)$ | $\mathsf{f}_\times(x',y) \xrightarrow{1} \mathsf{f}_+(\mathsf{f}_\times(x,y),y)$ | $[x' = x+1 \wedge \ldots]$ | (14) |

*Assume that we already analyzed its only bottom symbol* $\mathsf{f}_+$ *and obtained* $\mathsf{sz}(\mathsf{f}_+) = x_1 + x_2$ *and* $\mathsf{rt}(\mathsf{f}_+) = x_1$. *Afterwards,* (11) *and* (12) *were removed. Now Def. 23 is used to transform the sub-RNTS* $\{(13),(14)\}$ *induced by* $\mathsf{f}_\times$ *into an ITS. The only rule of* $\wr\mathcal{R}^\times\int$ *that violates the restriction of ITSs is (14). Thus, let* $(14')$ *result from* (14) *by replacing its right-hand side by* $\mathfrak{a}^{\mathsf{o}}_{\mathsf{sz}}(\mathsf{f}_+(\mathsf{f}_\times(x,y),y)) = \mathsf{sz}(\mathsf{f}_+)\{x_1/\mathsf{f}_\times(x,y), x_2/y\} = \mathsf{f}_\times(x,y)+y$. *Now* $\{(13),(14')\}$ *is an ITS, and together with Thm. 19, existing ITS tools can generate a size bound like* $\mathsf{sz}(\mathsf{f}_\times) = x_1 \cdot x_2$.

To finish the transformation of RNTSs to ITSs, we now handle terms like $f(\ldots g(\ldots)\ldots)$ where $f \in \Sigma_d^{\mathcal{P}}$ is the bottom symbol we are analyzing and we have an *inner* call of a constructor $g \in \Sigma_c^{\mathcal{P}}$. We would like to replace $g$ by $\mathsf{sz}(g)$ again. However, $f$ might behave non-monotonically (i.e., $f$ might need *less* runtime on *greater* arguments). Therefore, we replace all inner calls $g(\ldots)$ of constructors by fresh variables $x$. The size bound of the replaced call $g(\ldots)$ is an upper bound for the value of $x$, but $x$ can also take smaller values.

**Definition 25 (Inner Abstraction).** *Let* $\mathcal{P}$ *be an RNTS with size bound* $\mathsf{sz}$, $t$ *be a term, and* $\mathcal{P}os_c^{top} \subseteq \mathcal{P}os(t)$ *be the topmost positions of* $\Sigma_c^{\mathcal{P}}$*-symbols below* $\Sigma_d^{\mathcal{P}}$*-symbols in* $t$. *Thus,* $\mu \in \mathcal{P}os_c^{top}$ *iff* $\mathrm{root}(t|_\mu) \in \Sigma_c^{\mathcal{P}}$, *there exists a* $\pi < \mu$ *with* $\mathrm{root}(t|_\pi) \in \Sigma_d^{\mathcal{P}}$, *and* $\mathrm{root}(t|_{\pi'}) \in \Sigma_{\mathsf{exp}}$ *for all* $\pi < \pi' < \mu$. *For* $\mathcal{P}os_c^{top} = \{\mu_1, \ldots, \mu_k\}$, $t$*'s* inner abstraction *is* $\mathfrak{a}^{\mathsf{i}}(t) = t[x_1]_{\mu_1} \ldots [x_k]_{\mu_k}$ *where* $x_1, \ldots, x_k$ *are pairwise different fresh variables, and its* condition *is* $\psi^{\mathsf{i}}_{\mathsf{sz}}(t) = \bigwedge_{1 \le i \le k} x_i \le \mathsf{sz}(t|_{\mu_i})$.

*Example 26. For the RNTS of Ex. 14 and 17, we start with analyzing* $\mathcal{P}^{\mathsf{gt}}$ *which yields* $\mathsf{sz}(\mathsf{gt}) = 1$ *and* $\mathsf{rt}(\mathsf{gt}) = 0$, *cf. Ex. 20. After removing the* gt*-rules, the new bottom symbols are* ins *and* if. *The right-hand side of Rule* $(4')$ *contains a call of* gt below *the symbol* if. *With the size bound* $\mathsf{sz}(\mathsf{gt}) = 1$, *the inner abstraction of this right-hand side is* $\mathfrak{a}^{\mathsf{i}}(\mathsf{if}(\mathsf{gt}(x,y),x,ys')) = \mathsf{if}(x_1,x,ys')$, *and the corresponding condition* $\psi^{\mathsf{i}}_{\mathsf{sz}}(\mathsf{if}(\mathsf{gt}(x,y),x,ys'))$ *is* $x_1 \le 1$, *since* $\mathsf{sz}(\mathsf{gt}(x,y)) = 1$.

Thm. 27 states how to transform RNTSs into ITSs in order to compute runtime and size bounds. Suppose that we have already analyzed the function symbols $g_1, \ldots, g_m$, that $f$ becomes a new bottom symbol if the rules for $g_1, \ldots, g_m$ are removed, that $\mathcal{Q}$ is the sub-RNTS induced by $f$, and that $\mathcal{P}$ results from $\mathcal{Q}$ by deleting the rules for $g_1, \ldots, g_m$. Thus, if $g_i$ occurs in $\mathcal{P}$, then $g_i \in \Sigma_c^{\mathcal{P}}$.

So in our leading example, we have $g_1 = \mathsf{gt}$ (i.e., all gt-rules were analyzed and removed). Thus, ins is a new bottom symbol. If we want to analyze it by Thm. 27, then $\mathcal{Q}$ contains all ins-, if-, and gt-rules and $\mathcal{P}$ just contains all ins- and if-rules.

Since we restricted ourselves to RNTSs $\mathcal{Q}$ without nested recursion, $\mathcal{P}$ has no nested defined symbols. To infer a *size* bound for the bottom symbol $f$ of $\mathcal{P}$, we abstract away inner occurrences of $g_i$ by $\mathfrak{a}^{\mathsf{i}}$ (e.g., gt on the right-hand side

of Rule (4′) in our example), and we abstract away outer occurrences of $g_i$ by $\mathfrak{a}^{\mathsf{o}}_{\mathsf{sz}}$. So every right-hand side $r$ is replaced by $\mathfrak{a}^{\mathsf{o}}_{\mathsf{sz}}(\mathfrak{a}^{\mathsf{i}}(r))$ and we add the condition $\psi^{\mathsf{i}}_{\mathsf{sz}}(r)$ which restricts the values of the fresh variables introduced by $\mathfrak{a}^{\mathsf{i}}$.

To infer *runtime* bounds, inner occurrences of $g_i$ are also abstracted by $\mathfrak{a}^{\mathsf{i}}$, and outer occurrences of $g_i$ are simply removed. So every right-hand side $r$ is replaced by $\sum_{\pi \in \mathcal{P}os_d(r)} \mathfrak{a}^{\mathsf{i}}(r|_\pi)$, where $\mathcal{P}os_d(r) = \{\pi \in \mathcal{P}os(r) \mid \mathrm{root}(r|_\pi) \in \Sigma^{\mathcal{P}}_d\}$. However, we have to take into account how many computation steps would be required in the procedures $g_i$ that were called in $r$. Therefore, we compute the *cost* of all calls of $g_i$ in a rule's right-hand side and add it to the weight of the rule. To estimate the cost of a call $g_i(s_1, \ldots, s_n)$, we "apply" $\mathsf{rt}(g_i)$ to the size bounds of $s_1, \ldots, s_n$ and add the costs for evaluating $s_1, \ldots, s_n$.

**Theorem 27 (Transformation of RNTSs to ITSs).** *Let $\mathcal{Q}$ be an RNTS with size and runtime bounds $\mathsf{sz}$ and $\mathsf{rt}$ and let $\mathcal{P} = \mathcal{Q} \setminus (\mathcal{Q}^{g_1} \cup \ldots \cup \mathcal{Q}^{g_m})$, where $g_1, \ldots, g_m \in \Sigma$ and $\mathcal{Q}^{g_i}$ is the sub-RNTS of $\mathcal{Q}$ induced by $g_i$. We define*

$$\mathcal{P}_{\mathsf{sz}} \;=\; \{\, \ell \xrightarrow{w} \mathfrak{a}^{\mathsf{o}}_{\mathsf{sz}}(\mathfrak{a}^{\mathsf{i}}(r))\ \big[\varphi \wedge \psi^{\mathsf{i}}_{\mathsf{sz}}(r)\big] \;\mid\; \ell \xrightarrow{w} r\,[\varphi] \in \mathcal{P} \,\}$$

*Let $\mathsf{sz}'$ be a size bound for $\mathcal{P}_{\mathsf{sz}}$ where $\mathsf{sz}'(f) = \mathsf{sz}(f)$ for all $f \in \Sigma \setminus \Sigma^{\mathcal{P}}_d$. If $\mathcal{P}$ does not have nested defined symbols, then $\mathsf{sz}'$ is a size bound for $\mathcal{Q}$.*

*To obtain a runtime bound for $\mathcal{Q}$, we define an RNTS $\mathcal{P}_{\mathsf{rt},\mathsf{sz}'}$. To this end, we define the* cost *of a term as $\mathfrak{c}_{\mathsf{rt},\mathsf{sz}'}(x) = 0$ for $x \in \mathcal{V}$ and*

$$\mathfrak{c}_{\mathsf{rt},\mathsf{sz}'}(g(s_1, \ldots, s_n)) = \begin{cases} \sum_{1 \le j \le n} \mathfrak{c}_{\mathsf{rt},\mathsf{sz}'}(s_j) + \mathsf{rt}(g)\,\{x_j/\mathsf{sz}'(s_j) \mid 1 \le j \le n\} & \text{if } g \in \Sigma^{\mathcal{P}}_c \\ \sum_{1 \le j \le n} \mathfrak{c}_{\mathsf{rt},\mathsf{sz}'}(s_j) & \text{otherwise} \end{cases}$$

*Now $\mathcal{P}_{\mathsf{rt},\mathsf{sz}'} = \{\ell \xrightarrow{w+\mathfrak{c}_{\mathsf{rt},\mathsf{sz}'}(r)} \sum_{\pi \in \mathcal{P}os_d(r)} \mathfrak{a}^{\mathsf{i}}(r|_\pi)\ \big[\varphi \wedge \psi^{\mathsf{i}}_{\mathsf{sz}'}(r)\big] \mid \ell \xrightarrow{w} r\,[\varphi] \in \mathcal{P}\}$. Then every runtime bound $\mathsf{rt}'$ for $\mathcal{P}_{\mathsf{rt},\mathsf{sz}'}$ with $\mathsf{rt}'(f) = \mathsf{rt}(f)$ for all $f \in \Sigma \setminus \Sigma^{\mathcal{P}}_d$ is a runtime bound for $\mathcal{Q}$. Here, all occurrences of $\omega$ in $\mathcal{P}_{\mathsf{sz}}$ or $\mathcal{P}_{\mathsf{rt},\mathsf{sz}'}$ are replaced by pairwise different fresh variables.*

If $\mathcal{P}$ does not have nested defined symbols, then $\mathcal{P}_{\mathsf{sz}}$ and $\mathcal{P}_{\mathsf{rt},\mathsf{sz}'}$ are ITSs and thus, they can be analyzed by existing ITS tools.

*Example 28. We now finish analyzing the RNTS $\wr\mathcal{R}^{\times}\wr$ after updating $\mathsf{sz}$ as in Ex. 24. The cost of the right-hand side of (14) is $\mathfrak{c}_{\mathsf{rt},\mathsf{sz}}(\mathsf{f}_+(\mathsf{f}_\times(x,y),y)) = \mathsf{rt}(\mathsf{f}_+)\,\{x_1/x{\cdot}y, x_2/y\} = x \cdot y$. So for the sub-RNTS $\mathcal{P} = \{(13),(14)\}$ induced by $\mathsf{f}_\times$, $\mathcal{P}_{\mathsf{rt},\mathsf{sz}}$ is*

$$\mathsf{f}_\times(x,y) \xrightarrow{1} 0\ \ [x = 1 \wedge \ldots] \qquad \mathsf{f}_\times(x',y) \xrightarrow{1+x\cdot y} \mathsf{f}_\times(x,y)\ \ [x' = x+1 \wedge \ldots]$$

*Hence, existing ITS tools like CoFloCo [16, 17] or KoAT [13] yield a bound like $\mathsf{rt}(\mathsf{f}_\times) = x_1^2 \cdot x_2$. So by Thm. 13 and 18 we get $\mathrm{irc}_{\mathcal{R}^\times}(n) \le \mathrm{irc}_{\wr\mathcal{R}^\times\wr}(n) \in \mathcal{O}(n^3)$.*

*Example 29. To finish the analysis of the RNTS from Ex. 14, we continue Ex. 26. After we removed $\mathcal{P}^{\mathsf{gt}}$, the new bottom symbols $\mathsf{ins}$ and $\mathsf{if}$ both induce $\mathcal{P}^{\mathsf{ins}} = \{(3'),\ldots,(6'),(10)\}$. Constructing $\mathcal{P}^{\mathsf{ins}}_{\mathsf{sz}}$ yields the rules $(3'),(5'),(6'),(10)$, and*

$$\mathsf{ins}(x, ys') \;\xrightarrow{1}\; \mathsf{if}(x_1, x, ys')\ \ [\,ys' = 1 + y + ys \,\wedge\, \ldots \,\wedge\, x_1 \le 1\,] \qquad (4'')$$

*Existing tools like* CoFloCo *or* KoAT *compute size bounds like* $1 + x_1 + x_2$ *for* ins *and* $1 + x_2 + x_3$ *for* if *using Thm. 19. After updating* sz*, we construct* $\mathcal{P}^{\mathsf{ins}}_{\mathsf{rt},\mathsf{sz}}$ *which consists of* $(4'')$ *and variants of* $(3'), (5'), (6'), (10)$ *with unchanged weights (as* $\mathsf{c}_{\mathsf{rt},\mathsf{sz}}(\mathsf{gt}(x,y)) = \mathsf{rt}(\mathsf{gt}) = 0$*). ITS tools now infer runtime bounds like* $2 \cdot x_2$ *for* ins *and* $2 \cdot x_3$ *for* if*. After removing* ins *and* if*, we analyze the remaining RNTS* $\mathcal{P}^{\mathsf{isort}} = \{(1'), (2')\}$*. Since the right-hand side of* $(2')$ *contains an inner occurrence of* ins *below* isort*,* $(2')$ *is replaced by*

$$\mathsf{isort}(xs', ys) \xrightarrow{w} \mathsf{isort}(xs, ys') \quad [xs' = 1 + x + xs \wedge ys' \leq 1 + x + ys \wedge \ldots]$$

*where* $w = 1$ *in* $\mathcal{P}^{\mathsf{isort}}_{\mathsf{sz}}$ *and* $w = 1 + \mathsf{rt}(\mathsf{ins})\{x_1/x, x_2/ys\} = 1 + 2 \cdot ys$ *in* $\mathcal{P}^{\mathsf{isort}}_{\mathsf{rt},\mathsf{sz}}$*. Using Thm. 19, one can now infer bounds like* $\mathsf{sz}(\mathsf{isort}) = x_1 + x_2$ *and* $\mathsf{rt}(\mathsf{isort}) = x_1^2 + 2 \cdot x_1 \cdot x_2$*. Hence, by Thm. 18 one can deduce* $\mathrm{irc}(n) \in \mathcal{O}(n^2)$*.*

Based on Thm. 27, we can now show the correctness of our overall analysis.

**Theorem 30 (Alg. 1 is Sound).** *Let* $\mathcal{P}$ *be an RNTS and let* rt *and* sz *be the result of Alg. 1 for* $\mathcal{P}$*. Then* rt *is a runtime bound and* sz *is a size bound for* $\mathcal{P}$*.*

## 5 Related Work, Experiments, and Conclusion

To make techniques for complexity analysis of integer programs also applicable to TRSs, we presented two main contributions: First, we showed in Sect. 3 how TRSs can be abstracted to a variant of integer transition systems (called RNTSs) and presented conditions for the soundness of this abstraction. While abstractions from term-shaped data to numbers are common in program analysis (e.g., for proving termination), soundness of our abstraction for *complexity* of TRSs is not trivial. In [3] a related abstraction technique from first-order functional programs to a formalism corresponding to RNTSs is presented. However, there are important differences between such functional programs and term rewriting: In TRSs, one can also rewrite non-ground terms, whereas functional programming only evaluates ground expressions. Moreover, overlapping rules in TRSs may lead to non-determinism. The most challenging part in Sect. 3 is Thm. 9, i.e., showing that the step from innermost term rewriting to ground innermost rewriting is complexity preserving, even for relative rewriting. Mappings from terms to numbers were also used for complexity analysis of logic programs [15]. However, [15] operates on the logic program level, i.e., it does not translate programs to ITSs and it does not allow the application of ITS-techniques and tools.

Our second contribution (Sect. 4) is an approach to lift any technique for runtime complexity of ITSs to handle (non-nested, but otherwise *arbitrary*) recursion as well. This approach is useful for the analysis of recursive arithmetic programs in general. In particular, by combining our two main contributions we obtain a completely modular approach for the analysis of TRSs. To infer runtime bounds, we also compute size bounds, which may be useful on their own as well.

There exist several approaches that also analyze complexity by inferring both runtime and size bounds. Wegbreit [35] tries to generate closed forms for the exact runtime and size of the result of each analyzed function, whereas we esti-

mate runtime and size by upper bounds. Hence, [35] fails whenever finding such exact closed forms automatically is infeasible. Serrano et al. [33] also compute runtime and size bounds, but in contrast to us they work on logic programs, and their approach is based on abstract interpretation. Our technique in Sect. 4 was inspired by our work on the tool KoAT [13], which composes results of alternating size and runtime complexity analyses for ITSs. In [13] we developed a "bottom-up" technique that corresponds to the approach of Sect. 4.2 when restricting it to ordinary ITSs *without (non-tail) recursion*. But in contrast to Sect. 4.2, KoAT's support for recursion is very limited, as it disregards the return values of "inner" calls. Moreover, [13] does not contain an approach like Thm. 19 in Sect. 4.1 which allows us to obtain size bounds from techniques that compute runtime bounds.

RAML [26–28] reduces the inference of resource annotated types (and hence complexity bounds) for ML programs to linear optimization. Like other techniques for functional programs, it is not directly applicable to TRSs due to the differences between ML and term rewriting.[6] Moreover, RAML has two theoretical boundaries w.r.t. modularity [26]: (A) The number of linear constraints arising from type inference grows exponentially in the size of the program. (B) To achieve context-sensitivity, functions are typed differently for different invocations. In our setting, a blow-up similar to (A) may occur within the used ITS tool, but as the program is analyzed one function at a time, this blow-up is exponential in the size of a single function instead of the whole program. To avoid (B), we analyze each function only once. However, RAML takes amortization effects into account and obtains impressive results in practice. Further leading tools for complexity analysis of programs on integers (resp. naturals) are, e.g., ABC [11], C$^4$B [14], CoFloCo [16,17], LoAT [18], Loopus [34], PUBS [1, 2], Rank [4], and SPEED [23].

Finally, there are numerous techniques for automated complexity analysis of TRSs, e.g., [7, 8, 24, 32, 36]. While they also allow forms of modularity, the modularity of our approach differs substantially due to two reasons:

**(1)** Most previous complexity analysis techniques for TRSs are *top-down* approaches which estimate how often a rule $g(\ldots) \to \ldots$ is applied in reductions that start with terms of a certain size. So the complexity of a rule depends on the context of the whole TRS. This restricts the modularity of these approaches, since one cannot analyze $g$'s complexity without taking the rest of the TRS into account. In contrast, we propose a *bottom-up* approach which analyzes how the complexity of any function $g$ depends on $g$'s inputs. Hence, one can analyze $g$ without taking into account how $g$ is called by other functions $f$.

**(2)** In our technique, if a function $g$ has been analyzed, we can replace it by its size bound and do not have to regard $g$'s rules anymore when analyzing a function $f$ that calls $g$. This is possible because we use a fixed abstraction from terms to numbers. In contrast, existing approaches for TRSs cannot remove rules from the original TRS after having oriented them (with a strict order $\succ$), except for special cases. When other parts of the TRS are analyzed afterwards, these previous rules still have to be oriented weakly (with $\succsim$), since existing TRS

[6] See [29] for an adaption of an amortized analysis as in [27] to term rewriting. However, [29] is not automated, and it is restricted to *ground* rewriting with orthogonal rules.

approaches do not have any dedicated size analysis. This makes the existing approaches for TRSs less modular, but also more flexible (since they do not use a fixed abstraction from terms to numbers). In future work, we will try to improve our approach by integrating ideas from [3] which could allow us to infer and to apply multiple norms when abstracting functional programs to RNTSs.

We implemented our contributions in the tool AProVE [22] and evaluated its power on all 922 examples of the category "Runtime Complexity - Innermost Rewriting" of the *Termination and Complexity Competition 2016*.[7] Here, we excluded the 100 examples where AProVE shows $\mathrm{irc}(n) = \omega$.

In our experiments, we consider the previous version of AProVE (AProVE '16), a version using only the techniques from this paper (AProVE RNTS), and AProVE '17 which integrates the techniques from this paper into AProVE's previous approach to analyze irc. In all these versions, AProVE pre-processes the TRS to remove rules with non-basic left-hand sides that are unreachable from basic terms, cf. [19]. AProVE RNTS uses the external tools CoFloCo, KoAT, and PUBS to compute runtime bounds for the ITSs resulting from the technique in Sect. 4. While we restricted ourselves to polynomial arithmetic for simplicity in this paper, KoAT's ability to prove exponential bounds for ITSs also enables AProVE to infer exponential upper bounds for some TRSs. Thus, the capabilities of the backend ITS tool determine which kinds of bounds can be derived by AProVE. We also compare with TcT 3.1.0 [7], since AProVE and TcT were the most powerful complexity tools for TRSs at the *Termination and Complexity Competition 2016*.

Note that while the approach of Sect. 4 allows us to use *any* existing (or future) ITS tools for complexity analysis of RNTSs, CoFloCo can also infer complexity bounds for recursive ITSs directly, i.e., it does not require the technique in Sect. 4. To this end, CoFloCo analyzes program parts independently and uses linear invariants to compose the results. So CoFloCo's approach differs significantly from Sect. 4, which can also infer non-linear size bounds. Thus, the approach of Sect. 4 is especially suitable for examples where non-linear growth of data causes non-linear runtime. For instance, in Ex. 28 the quadratic size bound for $\mathsf{f}_\times$ is crucial to prove a (tight) cubic runtime bound with the technique of Sect. 4. Consequently, CoFloCo's linear invariants are not sufficient and hence it fails for this RNTS. See [5] for a list of 17 examples with non-linear runtime where Sect. 4 was superior to all other considered techniques in our experiments. However, CoFloCo's amortized analysis often results in very precise bounds, i.e., both approaches are orthogonal. Therefore, as an alternative to Sect. 4, AProVE RNTS also uses CoFloCo to analyze the RNTSs obtained from the transformation in Sect. 3 directly.

The table on the right shows the results of our experiments. As suggested in [8], we used a timeout of 300 seconds per

| $\mathrm{irc}_{\mathcal{R}}(n)$ | TcT | AProVE RNTS | AProVE '16 | AProVE & TcT | AProVE '17 |
|---|---|---|---|---|---|
| $\mathcal{O}(1)$ | 47 | 43 | 48 | 53 | 53 |
| $\leq \mathcal{O}(n)$ | 276 | 254 | 320 | 354 | 379 |
| $\leq \mathcal{O}(n^2)$ | 362 | 366 | 425 | 463 | 506 |
| $\leq \mathcal{O}(n^3)$ | 386 | 402 | 439 | 485 | 541 |
| $\leq \mathcal{O}(n^{>3})$ | 393 | 412 | 439 | 491 | 548 |
| $\leq EXP$ | 393 | 422 | 439 | 491 | 553 |

[7] See `http://termination-portal.org/wiki/Termination_Competition/`

example (on an Intel Xeon with 4 cores at 2.33 GHz each and 16 GB of RAM). AProVE & TcT represents the former state of the art, i.e., for each example here we took the best bound found by AProVE '16 or TcT. A row "$\leq \mathcal{O}(n^k)$" means that the corresponding tools proved a bound $\leq \mathcal{O}(n^k)$ (e.g., TcT proved constant or linear upper bounds in 276 cases). Clearly, AProVE '17 is the most powerful tool, i.e., the contributions of this paper significantly improve the state of the art for complexity analysis of TRSs. This also shows that the new technique of this paper is orthogonal to the existing ones. In fact, AProVE RNTS infers better bounds than AProVE & TcT in 127 cases. In 102 of them, AProVE & TcT fails to prove any bound at all. The main reasons for this orthogonality are that on the one hand, our approaches loses precision when abstracting terms to numbers. But on the other hand, our approach allows us to apply arbitrary tools for complexity analysis of ITSs in the back-end and to benefit from their respective strengths. Moreover as mentioned above, the approach of Sect. 4 succeeds on many examples where non-linear growth of data leads to non-linear runtime, which are challenging for existing techniques.

For further details on our experiments including a detailed comparison of AProVE RNTS and prior techniques for TRSs, to access AProVE '17 via a web interface, for improvements to increase the precision of our abstraction from TRSs to RNTSs, and for the proofs of all theorems, we refer to [5].

**Acknowledgments.** We thank A. Flores-Montoya for his help with CoFloCo and the anonymous reviewers for their suggestions and comments.

## References

1. Albert, E., Arenas, P., Genaim, S., Puebla, G.: Closed-form upper bounds in static cost analysis. Journal of Automated Reasoning 46(2), 161–203 (2011)
2. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: Cost analysis of object-oriented bytecode programs. Theoretical Comp. Sc. 413(1), 142–159 (2012)
3. Albert, E., Genaim, S., Gutiérrez, R.: A transformational approach to resource analysis with typed-norms. In: LOPSTR '13. pp. 38–53
4. Alias, C., Darte, A., Feautrier, P., Gonnord, L.: Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In: SAS '10. pp. 117–133
5. AProVE: `https://aprove-developers.github.io/trs_complexity_via_its/`.
6. Avanzini, M., Felgenhauer, B.: Type introduction for runtime complexity analysis. In: WST '14. pp. 1–5, available from `http://www.easychair.org/smart-program/VSL2014/WST-proceedings.pdf`
7. Avanzini, M., Moser, G., Schaper, M.: TcT: Tyrolean complexity tool. In: TACAS '16. pp. 407–423
8. Avanzini, M., Moser, G.: A combination framework for complexity. Information and Computation 248, 22–55 (2016)
9. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge U. Press (1998)
10. Baillot, P., Dal Lago, U., Moyen, J.Y.: On quasi-interpretations, blind abstractions and implicit complexity. Math. Structures in Comp. Sc. 22(4), 549–580 (2012)
11. Blanc, R., Henzinger, T.A., Hottelier, T., Kovács, L.: ABC: Algebraic bound computation for loops. In: LPAR (Dakar) '10. pp. 103–118

12. Bonfante, G., Cichon, A., Marion, J.Y., Touzet, H.: Algorithms with polynomial interpretation termination proof. J. Functional Programming 11(1), 33–53 (2001)
13. Brockschmidt, M., Emmes, F., Falke, S., Fuhs, C., Giesl, J.: Analyzing runtime and size complexity of integer programs. ACM TOPLAS 38(4) (2016)
14. Carbonneaux, Q., Hoffmann, J., Shao, Z.: Compositional certified resource bounds. In: PLDI '15. pp. 467–478
15. Debray, S., Lin, N.: Cost analysis of logic programs. TOPLAS 15(5), 826–875 (1993)
16. Flores-Montoya, A., Hähnle, R.: Resource analysis of complex programs with cost equations. In: APLAS '14. pp. 275–295
17. Flores-Montoya, A.: Upper and lower amortized cost bounds of programs expressed as cost relations. In: FM '16. pp. 254–273
18. Frohn, F., Naaf, M., Hensel, J., Brockschmidt, M., Giesl, J.: Lower runtime bounds for integer programs. In: IJCAR '16. pp. 550–567
19. Frohn, F., Giesl, J.: Analyzing runtime complexity via innermost runtime complexity. In: LPAR '17. pp. 249–268
20. Frohn, F., Giesl, J., Hensel, J., Aschermann, C., Ströder, T.: Lower bounds for runtime complexity of term rewriting. J. Aut. Reasoning 59(1), 121–163 (2017)
21. Fuhs, C., Giesl, J., Parting, M., Schneider-Kamp, P., Swiderski, S.: Proving termination by dep. pairs and inductive theorem proving. JAR 47(2), 133–160 (2011)
22. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. Journal of Automated Reasoning 58, 3–31 (2017)
23. Gulwani, S.: SPEED: Symbolic complexity bound analysis. In: CAV '09. pp. 51–62
24. Hirokawa, N., Moser, G.: Automated complexity analysis based on the dependency pair method. In: IJCAR '08. pp. 364–379
25. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: RTA '89. pp. 167–177
26. Hoffmann, J.: Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis. Ph.D. thesis, Ludwig-Maximilians-University Munich (2011)
27. Hoffmann, J., Aehlig, K., Hofmann, M.: Multivariate amortized resource analysis. ACM Transactions on Programming Languages and Systems 34(3) (2012)
28. Hoffmann, J., Das, A., Weng, S.C.: Towards automatic resource bound analysis for OCaml. In: POPL '17. pp. 359–373
29. Hofmann, M., Moser, G.: Multivariate amortised resource analysis for term rewrite systems. In: TLCA '15. pp. 241–256
30. Kapur, D., Narendran, P., Zhang, H.: On sufficient completeness and related properties of term rewriting systems. Acta Informatica 24, 395–415 (1987)
31. Kounalis, E.: Completeness in data type specifications. EUROCAL '85. pp. 348–362
32. Noschinski, L., Emmes, F., Giesl, J.: Analyzing innermost runtime complexity of term rewriting by dependency pairs. Journal of Aut. Reasoning 51(1), 27–56 (2013)
33. Serrano, A., López-García, P., Hermenegildo, M.: Resource usage analysis of logic programs via abstract interpretation using sized types. Theory and Practice of Logic Programming 14(4-5), 739–754 (2014)
34. Sinn, M., Zuleger, F., Veith, H.: Complexity and resource bound analysis of imperative programs using difference constraints. J. Aut. Reasoning 59(1), 3–45 (2017)
35. Wegbreit, B.: Mechanical program analysis. Comm. ACM 18, 528–539 (1975)
36. Zankl, H., Korp, M.: Modular complexity analysis for term rewriting. Logical Methods in Computer Science 10(1) (2014)
37. Zantema, H.: Termination of term rewriting: Interpretation and type elimination. Journal of Symbolic Computation 17(1), 23–50 (1994)