Inferring Lower Bounds for Runtime Complexity*

Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder

LuFG Informatik 2, RWTH Aachen University, Germany
{florian.frohn,giesl,hensel,cornelius,stroeder}@informatik.rwth-aachen.de

Abstract

We present the first approach to deduce lower bounds for innermost runtime complexity of term rewrite systems (TRSs) automatically. Inferring lower runtime bounds is useful to detect bugs and to complement existing techniques that compute upper complexity bounds. The key idea of our approach is to generate suitable families of rewrite sequences of a TRS and to find a relation between the length of such a rewrite sequence and the size of the first term in the sequence. We implemented our approach in the tool AProVE and evaluated it by extensive experiments.

1998 ACM Subject Classification F.1.3 - Complexity Measures and Classes, F.4.2 - Grammars and Other Rewriting Systems, I.2.3 Deduction and Theorem Proving

Keywords and phrases Term Rewriting, Runtime Complexity, Lower Bounds, Induction

Digital Object Identifier 10.4230/LIPIcs.RTA.2015.x

1 Introduction

There exist numerous methods to infer *upper bounds* for the runtime complexity of TRSs [3, 12, 14, 17, 21]. We present the first automatic technique to infer *lower bounds* for the innermost[1] runtime complexity of TRSs. *Runtime complexity* [12] refers to the "worst" cases in terms of evaluation length and our goal is to find lower bounds for these cases. While upper complexity bounds help to prove the absence of bugs that worsen the performance of programs, lower bounds can be used to *find* such bugs. Moreover, in combination with methods to deduce upper bounds, our approach can prove *tight* complexity results. In addition to *asymptotic* lower bounds, in many cases our technique can even compute *concrete* bounds.

As an example, consider the following TRS $\mathcal{R}_{\mathsf{qs}}$ for *quicksort*. The auxiliary function $\mathsf{low}(x, xs)$ returns those elements from the list xs that are smaller than x (and high works analogously). To ease readability, we use infix notation for the function symbols $\leq$ and $\mathsf{++}$.

▶ **Example 1** (TRS $\mathcal{R}_{\mathsf{qs}}$ for Quicksort).

$$\begin{aligned}
\mathsf{qs}(\mathsf{nil}) &\to \mathsf{nil} && (1)\\
\mathsf{qs}(\mathsf{cons}(x, xs)) &\to \mathsf{qs}(\mathsf{low}(x, xs)) \mathbin{\mathsf{++}} \mathsf{cons}(x, \mathsf{qs}(\mathsf{high}(x, xs))) && (2)\\
\mathsf{low}(x, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{low}(x, \mathsf{cons}(y, ys)) &\to \mathsf{ifLow}(x \leq y, x, \mathsf{cons}(y, ys))\\
\mathsf{ifLow}(\mathsf{true}, x, \mathsf{cons}(y, ys)) &\to \mathsf{low}(x, ys)\\
\mathsf{ifLow}(\mathsf{false}, x, \mathsf{cons}(y, ys)) &\to \mathsf{cons}(y, \mathsf{low}(x, ys))\\
\mathsf{high}(x, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{high}(x, \mathsf{cons}(y, ys)) &\to \mathsf{ifHigh}(x \leq y, x, \mathsf{cons}(y, ys))
\end{aligned}$$

$$\begin{aligned}
\mathsf{zero} \leq x &\to \mathsf{true}\\
\mathsf{succ}(x) \leq \mathsf{zero} &\to \mathsf{false}\\
\mathsf{succ}(x) \leq \mathsf{succ}(y) &\to x \leq y
\end{aligned}$$

* Supported by the DFG grant GI 274/6-1.

[1] We consider *innermost* rewriting, since TRSs resulting from the translation of programs usually have to be evaluated with an innermost strategy (e.g., [10, 18]). Obviously, lower bounds for innermost reductions are also lower bounds for full reductions (i.e., our approach can also be used for full rewriting).

$$\begin{array}{lcl@{\qquad}lcl} \mathsf{ifHigh}(\mathsf{true}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{cons}(y, \mathsf{high}(x, ys)) & \mathsf{nil} \mathbin{+\!\!+} ys & \to & ys \\ \mathsf{ifHigh}(\mathsf{false}, x, \mathsf{cons}(y, ys)) & \to & \mathsf{high}(x, ys) & \mathsf{cons}(x, xs) \mathbin{+\!\!+} ys & \to & \mathsf{cons}(x, xs \mathbin{+\!\!+} ys) \end{array} \tag{3}$$

For any $n \in \mathbb{N}$, let $\gamma_{\mathbf{List}}(n)$ be the term $\overbrace{\mathsf{cons}(\mathsf{zero}, \dots, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}) \dots)}^{n \text{ times}}$, i.e., the list of length n where all elements have the value zero (we also use the notation "$\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil})$"). To find lower bounds, we automatically generate *rewrite lemmas* that describe families of rewrite sequences. For example, our technique infers the following rewrite lemma automatically.

$$\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \quad \overset{\mathrm{i}}{\to}^{3n^2+2n+1} \quad \gamma_{\mathbf{List}}(n) \tag{4}$$

This rewrite lemma means that for each $n \in \mathbb{N}$, there is an innermost rewrite sequence of length $3n^2 + 2n + 1$ that reduces $\mathsf{qs}(\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil}))$ to $\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil})$. From this rewrite lemma, our technique then concludes that the innermost runtime of $\mathcal{R}_{\mathsf{qs}}$ is at least quadratic.

While most methods to infer upper bounds are adaptions of termination techniques, the approach in this paper is related to our technique to prove non-termination of TRSs [7]. Both techniques generate "meta-rules" representing infinitely many rewrite sequences. However, the *rewrite lemmas* in the current paper are more general than the meta-rules in [7], as they can be parameterized by *several* variables $n_1, \dots, n_m$ of type $\mathbb{N}$.

In Sect. 2 we show how to automatically speculate conjectures that may result in suitable rewrite lemmas. Sect. 3 explains how these conjectures can be verified automatically by induction. From these induction proofs, one can deduce information on the lengths of the rewrite sequences represented by a rewrite lemma, cf. Sect. 4. Thus, the use of induction to infer lower runtime bounds represents a novel application for automated inductive theorem proving. This complements our earlier work on using inductive theorem proving for termination analysis [9]. Finally, Sect. 5 shows how rewrite lemmas can be used to infer lower bounds for the innermost runtime complexity of a TRS.

Sect. 6 discusses an improvement of our approach by pre-processing the TRS before the analysis and Sect. 7 extends our approach to handle rewrite lemmas with arbitrary unknown right-hand sides. We implemented our technique in the tool AProVE [11] and demonstrate its power by an extensive experimental evaluation in Sect. 8. All proofs can be found in [8].

2 Speculating Conjectures

We now show how to speculate conjectures (whose validity must be proved afterwards in Sect. 3). See, e.g., [5] for the basics of rewriting, where we only consider finite TRSs. $\mathcal{T}(\Sigma, \mathcal{V})$ is the set of all terms over a (finite) signature Σ and a set of variables $\mathcal{V}$ and $\mathcal{T}(\Sigma) = \mathcal{T}(\Sigma, \varnothing)$ is the set of ground terms. The *arity* of a symbol $f \in \Sigma$ is denoted by $\mathrm{ar}_\Sigma(f)$. As usual, the *defined symbols* of a TRS $\mathcal{R}$ are $\Sigma_{def}(\mathcal{R}) = \{ \mathrm{root}(\ell) \mid \ell \to r \in \mathcal{R} \}$ and the *constructors* $\Sigma_{con}(\mathcal{R})$ are all other function symbols in $\mathcal{R}$. Thus, $\Sigma_{def}(\mathcal{R}_{\mathsf{qs}}) = \{\mathsf{qs}, \mathsf{low}, \mathsf{ifLow}, \mathsf{high}, \mathsf{ifHigh}, \mathsf{+\!\!+}, \leq\}$ and $\Sigma_{con}(\mathcal{R}_{\mathsf{qs}}) = \{\mathsf{nil}, \mathsf{cons}, \mathsf{zero}, \mathsf{succ}, \mathsf{true}, \mathsf{false}\}$.

Our approach is based on rewrite lemmas containing *generator functions* such as $\gamma_{\mathbf{List}}$ for types like **List**. Hence, in the first step of our approach we compute suitable types for the TRS $\mathcal{R}$ to be analyzed. While ordinary TRSs are defined over untyped signatures Σ, Def. 2 shows how to extend such signatures by (monomorphic) types (see, e.g., [9, 14, 22]).

▶ **Definition 2** (Typing). Let Σ be an (untyped) signature. A many-sorted signature Σ' is a *typed variant* of Σ if it contains the same function symbols as Σ, with the same arities. So $f \in \Sigma$ with $\mathrm{ar}_\Sigma(f) = k$ iff $f \in \Sigma'$ where f's type has the form $\tau_1 \times \ldots \times \tau_k \to \tau$. Similarly, a typed variant $\mathcal{V}'$ of the set of variables $\mathcal{V}$ contains the same variables as $\mathcal{V}$, but now every variable has a type τ. We always assume that for every type τ, $\mathcal{V}'$ contains infinitely many

variables of type τ. Given Σ' and $\mathcal{V}'$, $t \in \mathcal{T}(\Sigma, \mathcal{V})$ is a *well-typed* term of type τ iff

- $t \in \mathcal{V}'$ is a variable of type τ or
- $t = f(t_1, \ldots, t_k)$ with $k \geq 0$, where each t_i is a well-typed term of type τ_i, and where $f \in \Sigma'$ has the type $\tau_1 \times \ldots \times \tau_k \to \tau$.

We only permit typed variants Σ' where there exist well-typed ground terms of types $\tau_1, \ldots, \tau_k$ over Σ', whenever some $f \in \Sigma'$ has type $\tau_1 \times \ldots \times \tau_k \to \tau$.[2]

A TRS $\mathcal{R}$ over Σ and $\mathcal{V}$ is *well typed* w.r.t. Σ' and $\mathcal{V}'$ iff for all $\ell \to r \in \mathcal{R}$, we have that ℓ and r are well typed and that they have the same type.[3]

For any TRS $\mathcal{R}$, one can use a standard type inference algorithm to compute a typed variant Σ' such that $\mathcal{R}$ is well typed. Of course, a trivial solution is to use a many-sorted signature with just one sort (then every term and every TRS are trivially well typed). But to make our approach more powerful, it is advantageous to use the most general typed variant where $\mathcal{R}$ is well typed. Here, the set of terms is decomposed into as many types as possible. Then fewer terms are well typed and more useful rewrite lemmas can be generated.

To make $\mathcal{R}_{\mathsf{qs}}$ from Ex. 1 well typed, we obtain a typed variant of its signature with the types **Nats**, **Bool**, and **List**. Here, the function symbols have the following types:

$$\begin{aligned}
\mathsf{nil} &: \mathbf{List} & \mathsf{qs} &: \mathbf{List} \to \mathbf{List} \\
\mathsf{cons} &: \mathbf{Nats} \times \mathbf{List} \to \mathbf{List} & ++ &: \mathbf{List} \times \mathbf{List} \to \mathbf{List} \\
\mathsf{zero} &: \mathbf{Nats} & \leq &: \mathbf{Nats} \times \mathbf{Nats} \to \mathbf{Bool} \\
\mathsf{succ} &: \mathbf{Nats} \to \mathbf{Nats} & \mathsf{low}, \mathsf{high} &: \mathbf{Nats} \times \mathbf{List} \to \mathbf{List} \\
\mathsf{true}, \mathsf{false} &: \mathbf{Bool} & \mathsf{ifLow}, \mathsf{ifHigh} &: \mathbf{Bool} \times \mathbf{Nats} \times \mathbf{List} \to \mathbf{List}
\end{aligned}$$

A type τ *depends* on a type τ' (denoted $\tau \sqsupseteq_{dep} \tau'$) iff $\tau = \tau'$ or if there is a $c \in \Sigma'_{con}(\mathcal{R})$ of type $\tau_1 \times \ldots \times \tau_k \to \tau$ where $\tau_i \sqsupseteq_{dep} \tau'$ for some $1 \leq i \leq k$. To ease the presentation, we do not allow mutually recursive types (i.e., if $\tau \sqsupseteq_{dep} \tau'$ and $\tau' \sqsupseteq_{dep} \tau$, then $\tau' = \tau$). To speculate conjectures, we now introduce generator functions γ_τ. For any $n \in \mathbb{N}$, $\gamma_\tau(n)$ is a term from $\mathcal{T}(\Sigma'_{con}(\mathcal{R}))$ where a recursive constructor of type τ is nested n times. A constructor $c : \tau_1 \times \ldots \times \tau_k \to \tau$ is *recursive* iff $\tau_i = \tau$ for some $1 \leq i \leq k$. So for the type **Nats** above, we have $\gamma_{\mathbf{Nats}}(0) = \mathsf{zero}$ and $\gamma_{\mathbf{Nats}}(n+1) = \mathsf{succ}(\gamma_{\mathbf{Nats}}(n))$. If a constructor has a non-recursive argument of type τ', then γ_τ instantiates this argument by $\gamma_{\tau'}(0)$. So for **List**, we get $\gamma_{\mathbf{List}}(0) = \mathsf{nil}$ and $\gamma_{\mathbf{List}}(n+1) = \mathsf{cons}(\mathsf{zero}, \gamma_{\mathbf{List}}(n))$. If a constructor has several recursive arguments, then several generator functions are possible. So for a type **Tree** with the constructors $\mathsf{leaf} : \mathbf{Tree}$ and $\mathsf{node} : \mathbf{Tree} \times \mathbf{Tree} \to \mathbf{Tree}$, we have $\gamma_{\mathbf{Tree}}(0) = \mathsf{leaf}$, but either $\gamma_{\mathbf{Tree}}(n+1) = \mathsf{node}(\gamma_{\mathbf{Tree}}(n), \mathsf{leaf})$ or $\gamma_{\mathbf{Tree}}(n+1) = \mathsf{node}(\mathsf{leaf}, \gamma_{\mathbf{Tree}}(n))$. Similarly, if a type has several non-recursive or recursive constructors, then several different generator functions can be constructed by considering all combinations of non-recursive and recursive constructors.

To ease the presentation, we only consider generator functions for *simply structured* types τ. Such types have exactly two constructors $c, d \in \Sigma'_{con}(\mathcal{R})$, where c is not recursive, d has exactly one argument of type τ, and each argument type $\tau' \neq \tau$ of c or d is simply structured, too. The presented approach can easily be extended to more complex types by applying suitable heuristics to choose one of the possible generator functions.

▶ **Definition 3** (Generator Functions and Equations)**.** Let $\mathcal{R}$ be a TRS that is well typed w.r.t. Σ' and $\mathcal{V}'$. We extend the set of types by a fresh type $\mathbb{N}$. For every type $\tau \neq \mathbb{N}$, let γ_τ be a fresh *generator function symbol* of type $\mathbb{N} \to \tau$. The set $\mathcal{G}_\mathcal{R}$ consists of the following *generator*

[2] This is not a restriction, as one can simply add new constants to Σ and Σ'.

[3] 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 τ_1 in one rule and with type τ_2 in another rule.

equations for every simply structured type τ with the constructors $c : \tau_1 \times \ldots \times \tau_k \to \tau$ and $d : \rho_1 \times \ldots \times \rho_b \to \tau$, where $\rho_j = \tau$. We write $\mathcal{G}$ instead of $\mathcal{G}_\mathcal{R}$ if $\mathcal{R}$ is clear from the context.

$$\begin{array}{rcl} \gamma_\tau(0) & = & c(\gamma_{\tau_1}(0), \ldots, \gamma_{\tau_k}(0)) \\ \gamma_\tau(n+1) & = & d(\gamma_{\rho_1}(0), \ldots, \gamma_{\rho_{j-1}}(0), \gamma_\tau(n), \gamma_{\rho_{j+1}}(0), \ldots, \gamma_{\rho_b}(0)) \end{array}$$

We extend $\sqsupseteq_{dep}$ to $\Sigma_{def}(\mathcal{R})$ by defining $f \sqsupseteq_{dep} h$ iff $f = h$ or if there is a rule $f(\ldots) \to r$ and a symbol g in t with $g \sqsupseteq_{dep} h$. When speculating conjectures, we take the dependencies between defined symbols into account. If $f \sqsupseteq_{dep} g$ and $g \not\sqsupseteq_{dep} f$, then we first generate a rewrite lemma for g. This lemma can be used when generating a lemma for f afterwards.

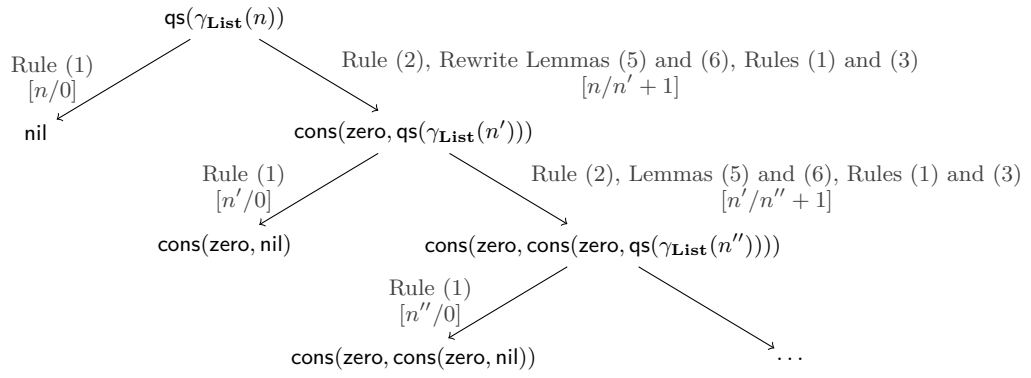
For $f \in \Sigma'_{def}(\mathcal{R})$ of type $\tau_1 \times \ldots \times \tau_k \to \tau$ with simply structured types $\tau_1, \ldots, \tau_k$, our goal is to speculate a conjecture of the form $f(\gamma_{\tau_1}(s_1), \ldots, \gamma_{\tau_k}(s_k)) \stackrel{\mathrm{i}}{\to}^* t$, where the $s_1, \ldots, s_k$ are polynomials over variables $n_1, \ldots, n_m$ of type $\mathbb{N}$. Moreover, t is a term built from Σ, arithmetic expressions, generator functions, and $n_1, \ldots, n_m$. As usual, a rewrite step is *innermost* (denoted $s \stackrel{\mathrm{i}}{\to}_\mathcal{R} t$ where we omit the index $\mathcal{R}$ if it is clear from the context) if the reduced subterm of s does not have redexes as proper subterms. From the speculated conjecture, we afterwards infer a rewrite lemma $f(\gamma_{\tau_1}(s_1), \ldots, \gamma_{\tau_k}(s_k)) \stackrel{\mathrm{i}}{\to}^{rt(n_1,\ldots,n_m)} t$, where $rt : \mathbb{N}^m \to \mathbb{N}$ describes the *runtime* of the lemma. To speculate a conjecture, we first generate *sample conjectures* that describe the effect of applying f to specific arguments. To this end, we narrow $f(\gamma_{\tau_1}(n_1), \ldots, \gamma_{\tau_k}(n_k))$ where $n_1, \ldots, n_k \in \mathcal{V}$ using the rules of the TRS and the lemmas we have proven so far, taking also the generator equations and integer arithmetic into account.

For any proven rewrite lemma $s \stackrel{\mathrm{i}}{\to}^{rt(\ldots)} t$, let the set $\mathcal{L}$ contain the rule $s \to t$. Moreover, let $\mathcal{A}$ be the infinite set of all valid equalities in the theory of $\mathbb{N}$ with addition and multiplication. Then s *narrows* to t ("$s \rightsquigarrow_{(\mathcal{R} \cup \mathcal{L})/(\mathcal{G} \cup \mathcal{A})} t$" or just "$s \rightsquigarrow t$" if $\mathcal{R}$, $\mathcal{L}$, $\mathcal{G}$ are clear from the context) iff there exist a term s', a substitution σ that maps variables of type $\mathbb{N}$ to arithmetic expressions, a position π, and a variable-renamed rule $\ell \to r \in \mathcal{R} \cup \mathcal{L}$ such that $s\sigma \equiv_{\mathcal{G} \cup \mathcal{A}} s'\sigma$, $s'|_\pi \sigma = \ell\sigma$, and $s'[r]_\pi \sigma = t$. Although checking $s\sigma \equiv_{\mathcal{G} \cup \mathcal{A}} s'\sigma$ (i.e., $\mathcal{G} \cup \mathcal{A} \models s\sigma = s'\sigma$) is undecidable in general, the required narrowing can usually be performed automatically using SMT solvers.

▶ **Example 4** (Narrowing). In Ex. 1 we have $\mathsf{qs} \sqsupseteq_{dep} \mathsf{low}$ and $\mathsf{qs} \sqsupseteq_{dep} \mathsf{high}$. If the lemmas

$$\mathsf{low}(\gamma_{\mathbf{Nats}}(0), \gamma_{\mathbf{List}}(n)) \stackrel{\mathrm{i}}{\to}^{3n+1} \gamma_{\mathbf{List}}(0) \quad (5) \qquad \mathsf{high}(\gamma_{\mathbf{Nats}}(0), \gamma_{\mathbf{List}}(n)) \stackrel{\mathrm{i}}{\to}^{3n+1} \gamma_{\mathbf{List}}(n) \quad (6)$$

were already proved, then the following narrowing tree can be generated to find sample conjectures for qs. The arrows are annotated with the rules and the substitutions used for variables of type $\mathbb{N}$. To save space, some arrows correspond to *several* narrowing steps.



The goal is to get representative rewrite sequences, but not to cover all reductions. So we stop constructing the tree after some steps and choose suitable narrowings heuristically.

After constructing a narrowing tree for f, we collect *sample points* (t, σ, d). Here, t results from a $\rightsquigarrow$-normal form q reached in a path of the tree by normalizing q w.r.t. the generator equations $\mathcal{G}$ applied from right to left. So terms from $\mathcal{T}(\Sigma, \mathcal{V})$ are rewritten to generator symbols with arithmetic expressions as arguments. Moreover, σ is the substitution for variables of type $\mathbb{N}$, and d is the number of applications of recursive f-rules on the path (the *recursion depth*). A rule $f(\ldots) \to r$ is *recursive* iff r contains a symbol g with $g \sqsupseteq_{dep} f$.

▶ **Example 5** (Sample Points). In Ex. 4, we obtain the following set of sample points:[4]

$$S \;=\; \{\, (\gamma_{\mathbf{List}}(0), [n/0], 0), \quad (\gamma_{\mathbf{List}}(1), [n/1], 1), \quad (\gamma_{\mathbf{List}}(2), [n/2], 2)\,\} \tag{7}$$

The sequence from $\mathsf{qs}(\gamma_{\mathbf{List}}(n))$ to nil does not use recursive qs-rules. So its recursion depth is 0 and the $\rightsquigarrow$-normal form nil rewrites to $\gamma_{\mathbf{List}}(0)$ when applying $\mathcal{G}$ from right to left. The sequence from $\mathsf{qs}(\gamma_{\mathbf{List}}(n))$ to $\mathsf{cons}(\mathsf{zero}, \mathsf{nil})$ (resp. $\mathsf{cons}(\mathsf{zero}, \mathsf{cons}(\mathsf{zero}, \mathsf{nil}))$) uses the recursive qs-rule (2) once (resp. twice), i.e., it has recursion depth 1 (resp. 2). Moreover, these $\rightsquigarrow$-normal forms rewrite to $\gamma_{\mathbf{List}}(1)$ (resp. $\gamma_{\mathbf{List}}(2)$) when using $\mathcal{G}$ from right to left.

A sample point (t, σ, d) for a narrowing tree with the root $s = f(\ldots)$ represents the *sample conjecture* $s\sigma \xrightarrow{\mathrm{i}}{}^{*} t$, which stands for a reduction with d applications of recursive f-rules. So for $s = \mathsf{qs}(\gamma_{\mathbf{List}}(n))$, the sample points in (7) represent the sample conjectures $\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(0)$, $\mathsf{qs}(\gamma_{\mathbf{List}}(1)) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(1)$, $\mathsf{qs}(\gamma_{\mathbf{List}}(2)) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(2)$. Now the goal is to speculate a general conjecture from these sample conjectures (whose validity must be proved afterwards).

In general, we search for a maximal subset of sample conjectures that are suitable for generalization. More precisely, if s is the root of the narrowing tree, then we take a maximal subset S_{max} of sample points such that for all $(t, \sigma, d), (t', \sigma', d') \in S_{max}$, the sample conjectures $s\sigma \xrightarrow{\mathrm{i}}{}^{*} t$ and $s\sigma' \xrightarrow{\mathrm{i}}{}^{*} t'$ are identical up to the occurring natural numbers and the variable names. For instance, $\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(0)$, $\mathsf{qs}(\gamma_{\mathbf{List}}(1)) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(1)$, and $\mathsf{qs}(\gamma_{\mathbf{List}}(2)) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(2)$ are indeed identical up to the numbers in these sample conjectures. To obtain a general conjecture, we replace all numbers in the sample conjectures by polynomials. So in our example, we want to speculate a conjecture of the form $\mathsf{qs}(\gamma_{\mathbf{List}}(pol^{left})) \xrightarrow{\mathrm{i}}{}^{*} \gamma_{\mathbf{List}}(pol^{right})$. Here, pol^{left} and pol^{right} are polynomials in one variable n (the *induction variable* of the conjecture) that stands for the recursion depth. This facilitates a proof of the resulting conjecture by induction on n.

So in general, in any sample conjecture $s\sigma \xrightarrow{\mathrm{i}}{}^{*} t$ that correspond to a sample point $(t, \sigma, d) \in S_{max}$, we replace the natural numbers in $s\sigma$ and t by polynomials. For any term q, let $\mathrm{pos}(q)$ be the set of its positions and $\Pi^{q}_{\mathbb{N}} = \{\pi \in \mathrm{pos}(q) \mid q|_\pi \in \mathbb{N}\}$. Then for each $\pi \in \Pi^{s\sigma}_{\mathbb{N}}$ (resp. $\pi \in \Pi^{t}_{\mathbb{N}}$) with $(t, \sigma, d) \in S_{max}$, we search for a polynomial pol^{left}_{π} (resp. pol^{right}_{π}). To this end, for every sample point $(t, \sigma, d) \in S_{max}$, we generate the constraints

$$\text{“}pol^{left}_{\pi}(d) \;=\; s\sigma|_\pi\text{”} \text{ for every } \pi \in \Pi^{s\sigma}_{\mathbb{N}} \quad \text{and} \quad \text{“}pol^{right}_{\pi}(d) \;=\; t|_\pi\text{”} \text{ for every } \pi \in \Pi^{t}_{\mathbb{N}}. \tag{8}$$

Here, pol^{left}_{π} and pol^{right}_{π} are polynomials with abstract coefficients. So if one searches for polynomials of degree e, then the polynomials have the form $c_0 + c_1 \cdot n + c_2 \cdot n^2 + \ldots + c_e \cdot n^e$ and the constraints in (8) are linear diophantine equations over the unknown coefficients $c_i \in \mathbb{N}$.[5] These equations can easily be solved automatically. Finally, the desired generalized

[4] We always simplify arithmetic expressions in terms and substitutions, e.g., the substitution $[n/0+1]$ in the second sample point is simplified to $[n/1]$.

[5] Note that in the constraints (8), n is instantiated by an actual number d. Thus, if $pol^{left}_{\pi} = c_0 + c_1 \cdot n + c_2 \cdot n^2 + \ldots + c_e \cdot n^e$, then $pol^{left}_{\pi}(d)$ is a *linear* polynomial over the unknowns $c_0, \ldots, c_e$.

speculated conjecture is obtained from $s\sigma \xrightarrow{i}{}^* t$ by replacing $s\sigma|_\pi$ with pol_π^{left} for every $\pi \in \Pi_{\mathbb{N}}^{s\sigma}$ and by replacing $t|_\pi$ with pol_π^{right} for every $\pi \in \Pi_{\mathbb{N}}^{t}$.

▶ **Example 6** (Speculating Conjectures). In Ex. 4, we narrowed $s = \mathsf{qs}(\gamma_{\mathbf{List}}(n))$ and S_{max} is the set S in (7). For each $(t, \sigma, d) \in S_{max}$, we have $\Pi_{\mathbb{N}}^{s\sigma} = \{1.1\}$ and $\Pi_{\mathbb{N}}^{t} = \{1\}$. So from the sample conjecture $\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \xrightarrow{i}{}^* \gamma_{\mathbf{List}}(0)$, where the recursion depth is $d=0$, we obtain the constraints $pol_{1.1}^{left}(d) = pol_{1.1}^{left}(0) = \mathsf{qs}(\gamma_{\mathbf{List}}(0))|_{1.1} = 0$ and $pol_1^{right}(d) = pol_1^{right}(0) = \gamma_{\mathbf{List}}(0)|_1 = 0$. Similarly, from the two other sample conjectures we get $pol_{1.1}^{left}(1) = pol_1^{right}(1) = 1$ and $pol_{1.1}^{left}(2) = pol_1^{right}(2) = 2$. When using $pol_{1.1}^{left} = c_0 + c_1 \cdot n + c_2 \cdot n^2$ and $pol_1^{right} = d_0 + d_1 \cdot n + d_2 \cdot n^2$ with the abstract coefficients $c_0, \ldots, c_2, d_0, \ldots, d_2$, the solution $c_0 = c_2 = d_0 = d_2 = 0$, $c_1 = d_1 = 1$ (i.e., $pol_{1.1}^{left} = n$ and $pol_1^{right} = n$) is easily found automatically. So the resulting conjecture is $\mathsf{qs}(\gamma_{\mathbf{List}}(pol_{1.1}^{left})) \xrightarrow{i}{}^* \gamma_{\mathbf{List}}(pol_1^{right})$, i.e., $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{i}{}^* \gamma_{\mathbf{List}}(n)$.

If S_{max} contains sample points with e different recursion depths, then we generate polynomials of at most degree $e-1$ satisfying the constraints (8) (these polynomials are determined uniquely). Ex. 7 shows how to speculate conjectures with *several* variables.

▶ **Example 7** (Conjecture With Several Variables). The following TRS combines half and plus.

$$\mathsf{hp}(\mathsf{zero}, y) \to y \qquad \mathsf{hp}(\mathsf{succ}(\mathsf{succ}(x)), y) \to \mathsf{succ}(\mathsf{hp}(x, y))$$

Narrowing $s = \mathsf{hp}(\gamma_{\mathbf{Nats}}(n_1), \gamma_{\mathbf{Nats}}(n_2))$ yields the sample points $(\gamma_{\mathbf{Nats}}(n_2), [n_1/0], 0)$, $(\gamma_{\mathbf{Nats}}(n_2+1), [n_1/2], 1)$, $(\gamma_{\mathbf{Nats}}(n_2+2), [n_1/4], 2)$, and $(\gamma_{\mathbf{Nats}}(n_2+3), [n_1/6], 3)$. For the last three sample points (t, σ, d), the only number in $s\sigma$ is at position 1.1 and the polynomial $pol_{1.1}^{left} = 2 \cdot n$ satisfies the constraint $pol_{1.1}^{left}(d) = s\sigma|_{1.1}$. Moreover, the only number in t is at position 1.2 and the polynomial $pol_{1.2}^{right} = n$ satisfies $pol_{1.2}^{right} = t|_{1.2}$. Thus, we speculate the conjecture $\mathsf{hp}(\gamma_{\mathbf{Nats}}(2 \cdot n), \gamma_{\mathbf{Nats}}(n_2)) \xrightarrow{i}{}^* \gamma_{\mathbf{Nats}}(n_2 + n)$ with the induction variable n.

3 Proving Rewrite Lemmas

If the proof of a speculated conjecture succeeds, then we have found a *rewrite lemma*.

▶ **Definition 8** (Rewrite Lemmas). Let $\mathcal{R}$ be a TRS that is well typed w.r.t. Σ' and $\mathcal{V}'$. For any term q, let $q{\downarrow_{\mathcal{G}/\mathcal{A}}}$ be q's normal form w.r.t. $\mathcal{G}_\mathcal{R}$, where the generator equations are applied from left to right and $\mathcal{A}$-equivalent (sub)terms are considered to be equal. Moreover, let $s \xrightarrow{i}{}^* t$ be a conjecture with $\mathcal{V}(s) = \{n_1, \ldots, n_m\} \neq \varnothing$, where $\overline{n} = (n_1, \ldots, n_m)$ are pairwise different variables of type $\mathbb{N}$, s is well typed, $\mathrm{root}(s) \in \Sigma_{def}(\mathcal{R})$, and s has no defined symbol from $\Sigma_{def}(\mathcal{R})$ below the root. Let $rt : \mathbb{N}^m \to \mathbb{N}$. Then $s \to_\mathcal{R}^{rt(\overline{n})} t$ is a *rewrite lemma* for $\mathcal{R}$ iff $s\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}} \xrightarrow{i}{}_\mathcal{R}^{rt(\overline{n}\sigma)} t\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}}$ for all $\sigma : \mathcal{V}(s) \to \mathbb{N}$, i.e., $s\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}}$ can be reduced to $t\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}}$ in exactly $rt(n_1\sigma, \ldots, n_m\sigma)$ innermost $\mathcal{R}$-steps. We omit the index $\mathcal{R}$ if it is clear from the context.

So the conjecture $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{i}{}^* \gamma_{\mathbf{List}}(n)$ gives rise to a rewrite lemma, since $\sigma(n) = b \in \mathbb{N}$ implies $\mathsf{qs}(\gamma_{\mathbf{List}}(b)){\downarrow_{\mathcal{G}/\mathcal{A}}} = \mathsf{qs}(\mathsf{cons}^b(\mathsf{zero}, \mathsf{nil})) \xrightarrow{i}{}^{3b^2+2b+1} \mathsf{cons}^b(\mathsf{zero}, \mathsf{nil}) = \gamma_{\mathbf{List}}(b){\downarrow_{\mathcal{G}/\mathcal{A}}}$.

To prove rewrite lemmas, essentially we use rewriting with $\xrightarrow{i}{}_{(\mathcal{R}\cup\mathcal{L})/(\mathcal{G}\cup\mathcal{A})}$.[6] However, this would allow us to prove lemmas that do not correspond to *innermost* rewriting with $\mathcal{R}$, if $\mathcal{R}$ contains rules with overlapping left-hand sides. Consider $\mathcal{R} = \{\mathsf{g}(\mathsf{zero}) \to \mathsf{zero}, \mathsf{f}(\mathsf{g}(x)) \to \mathsf{zero}\}$. We have $\mathsf{f}(\mathsf{g}(\gamma_{\mathbf{Nats}}(n))) \xrightarrow{i}{}_{(\mathcal{R}\cup\mathcal{L})/(\mathcal{G}\cup\mathcal{A})} \mathsf{zero}$, but for the instantiation $[n/0]$, this would not be an innermost reduction. To avoid this, we use the following relation $\xrightarrow{i}{}_\mathcal{R} \subseteq \xrightarrow{i}{}_{(\mathcal{R}\cup\mathcal{L})/(\mathcal{G}\cup\mathcal{A})}$: We have $s \xrightarrow{i}{}_\mathcal{R} t$ iff there exist a term s', a substitution σ, a position π, and a

[6] Here, we define $\xrightarrow{i}{}_{(\mathcal{R}\cup\mathcal{L})/(\mathcal{G}\cup\mathcal{A})}$ to be the relation $\equiv_{\mathcal{G}\cup\mathcal{A}} \circ (\xrightarrow{i}{}_\mathcal{R} \cup \to_\mathcal{L}) \circ \equiv_{\mathcal{G}\cup\mathcal{A}}$. An adaption of our approach to runtime complexity of full rewriting is obtained by considering $\to_{(\mathcal{R}\cup\mathcal{L})/(\mathcal{G}\cup\mathcal{A})}$ instead.

rule $\ell \to r \in \mathcal{R} \cup \mathcal{L}$ such that $s \equiv_{\mathcal{G}\cup\mathcal{A}} s'$, $s'|_\pi = \ell\sigma$ and $s'[r\sigma]_\pi \equiv_{\mathcal{G}\cup\mathcal{A}} t$. Moreover, if $\ell \to r \in \mathcal{R}$, then there must not be any proper non-variable subterm q of $\ell\sigma$, a (variable-renamed) rule $\ell' \to r' \in \mathcal{R}$, and a substitution σ' such that $\ell'\sigma' \equiv_{\mathcal{G}\cup\mathcal{A}} q\sigma'$. Now $\mathsf{f}(\mathsf{g}(\gamma_{\mathbf{Nats}}(n))) \not\xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{zero}$, because the subterm $\mathsf{g}(\gamma_{\mathbf{Nats}}(n))$ unifies with the left-hand side $\mathsf{g}(\mathsf{zero})$ modulo $\mathcal{G} \cup \mathcal{A}$.

When proving a conjecture $s \xrightarrow{\mathrm{i}}{}^* t$ by induction, in the step case we try to reduce $s[n/n+1]$ to $t[n/n+1]$, where one may use the rule IH: $s \to t$ as induction hypothesis. Here, the variables in IH may not be instantiated. The reason for not allowing instantiations of the non-induction variables from $\mathcal{V}(s) \setminus \{n\}$ is that such induction proofs are particularly suitable for inferring runtimes of rewrite lemmas, cf. Sect. 4.

Thus, for any rule IH: $\ell \to r$, let $s \mapsto_{\mathrm{IH}} t$ iff there exist a term s' and a position π such that $s \equiv_{\mathcal{G}\cup\mathcal{A}} s'$, $s'|_\pi = \ell$ and $s'[r]_\pi \equiv_{\mathcal{G}\cup\mathcal{A}} t$. Let $\xrightarrow{\mathrm{i}}_{(\mathcal{R},\mathrm{IH})} = \xrightarrow{\mathrm{i}}_{\mathcal{R}} \cup \mapsto_{\mathrm{IH}}$. Moreover, $\xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}}$ (resp. $\xrightarrow{\mathrm{i}}{}^*_{(\mathcal{R},\mathrm{IH})}$) denotes the transitive-reflexive closure of $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ (resp. $\xrightarrow{\mathrm{i}}_{(\mathcal{R},\mathrm{IH})}$), where in addition $s \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} s'$ and $s \xrightarrow{\mathrm{i}}{}^*_{(\mathcal{R},\mathrm{IH})} s'$ also hold if $s \equiv_{\mathcal{G}\cup\mathcal{A}} s'$. Thm. 9 shows which rewrite sequences are needed to prove a conjecture $s \xrightarrow{\mathrm{i}}{}^* t$ by induction on its induction variable n.

▶ **Theorem 9** (Proving Rewrite Lemmas). *Let $\mathcal{R}$, s, t be as in Def. 8, $n \in \mathcal{V}(s) = \{n_1, \ldots, n_m\}$, and $\overline{n} = (n_1, \ldots, n_m)$. If $s[n/0] \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t[n/0]$ and $s[n/n+1] \xrightarrow{\mathrm{i}}{}^*_{(\mathcal{R},\mathrm{IH})} t[n/n+1]$, where* IH *is the rule $s \to t$, then there is an $\mathit{rt} : \mathbb{N}^m \to \mathbb{N}$ such that $s \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(\overline{n})} t$ is a rewrite lemma for $\mathcal{R}$.*

▶ **Example 10** (Proof of Rewrite Lemma). Assume that we have already proved the rewrite lemmas (5) and (6). To prove the conjecture $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{\mathrm{i}}{}^* \gamma_{\mathbf{List}}(n)$, in the induction base we show $\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \gamma_{\mathbf{List}}(0)$ and in the induction step, we obtain $\mathsf{qs}(\gamma_{\mathbf{List}}(n+1)) \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} \mathsf{nil} \mathbin{+\!\!+} \mathsf{cons}(\mathsf{zero}, \mathsf{qs}(\gamma_{\mathbf{List}}(n))) \mapsto_{\mathrm{IH}} \mathsf{nil} \mathbin{+\!\!+} \mathsf{cons}(\mathsf{zero}, \gamma_{\mathbf{List}}(n)) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \gamma_{\mathbf{List}}(n+1)$. Thus, there is a rewrite lemma $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(n)} \gamma_{\mathbf{List}}(n)$. Sect. 4 will clarify how to find the function rt.

4 Inferring Bounds for Rewrite Lemmas

Now we show how to infer the function rt for a rewrite lemma $s \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(\overline{n})} t$ from its proof. If $n \in \overline{n}$ was the induction variable and the induction hypothesis was applied ih times in the induction step, then we get the following recurrence equations for rt where $\widetilde{n}$ is $\overline{n}$ without the variable n:

$$\mathit{rt}(\overline{n}[n/0]) = \mathit{ib}(\widetilde{n}) \qquad \text{and} \qquad \mathit{rt}(\overline{n}[n/n+1]) = \mathit{ih} \cdot \mathit{rt}(\overline{n}) + \mathit{is}(\overline{n}) \tag{9}$$

Here, $\mathit{ib}(\widetilde{n})$ is the length of the reduction $s[n/0]{\downarrow_{\mathcal{G}/\mathcal{A}}} \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t[n/0]{\downarrow_{\mathcal{G}/\mathcal{A}}}$, which must exist due to the induction base. The addend $\mathit{is}(\overline{n})$ is the length of $s[n/n+1]{\downarrow_{\mathcal{G}/\mathcal{A}}} \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t[n/n+1]{\downarrow_{\mathcal{G}/\mathcal{A}}}$, but without those subsequences that are covered by the induction hypothesis IH. Since the non-induction variables were not instantiated in IH, $\mathit{rt}(\overline{n})$ is the runtime for each application of IH. To compute ib and is, for each previous rewrite lemma $s' \xrightarrow{\mathrm{i}}{}^{\mathit{rt}'(\overline{n}')} t'$ that was used in the proof of $s \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(\overline{n})} t$, we assume that rt' is known. Thus, rt' can be used to infer the number of rewrite steps represented by that previous lemma. To avoid treating rules and rewrite lemmas separately, in Def. 11 we regard each rule $s \to t \in \mathcal{R}$ as a rewrite lemma $s \xrightarrow{\mathrm{i}}{}^1 t$.

▶ **Definition 11** ($\mathit{ih}, \mathit{ib}, \mathit{is}$). Let $s \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(\overline{n})} t$ be a rewrite lemma with an induction proof as in Thm. 9. More precisely, let $u_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}} \ldots \xrightarrow{\mathrm{i}}_{\mathcal{R}} u_{b+1}$ be the rewrite sequence $s[n/0] \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t[n/0]$ for the induction base and let $v_1 \xrightarrow{\mathrm{i}}_{(\mathcal{R},\mathrm{IH})} \ldots \xrightarrow{\mathrm{i}}_{(\mathcal{R},\mathrm{IH})} v_{k+1}$ be the rewrite sequence $s[n+1] \xrightarrow{\mathrm{i}}{}^*_{(\mathcal{R},\mathrm{IH})} t[n/n+1]$ for the induction step, where IH: $s \to t$ is applied ih times. For $j \in \{1, \ldots, b\}$, let $\ell_j \xrightarrow{\mathrm{i}}{}^{\mathit{rt}_j(\overline{y}_j)} r_j$ and σ_j be the rewrite lemma and substitution used to reduce u_j to u_{j+1}. Similarly for $j \in \{1, \ldots, k\}$, let $p_j \to^{\mathit{rt}'_j(\overline{z}_j)} q_j$ and θ_j be the lemma and substitution used to reduce v_j to v_{j+1}. Then we define:

$$\mathit{ib}(\widetilde{n}) \;=\; \sum\nolimits_{j\in\{1,\ldots,b\}} \mathit{rt}_j(\overline{y}_j\sigma_j) \qquad \text{and} \qquad \mathit{is}(\overline{n}) \;=\; \sum\nolimits_{j\in\{1,\ldots,k\},\ p_j\to q_j\neq \mathrm{IH}} \mathit{rt}'_j(\overline{z}_j\theta_j)$$

By solving the recurrence equations (9), we can now compute rt explicitly.

▶ **Theorem 12** (Explicit Runtime of Rewrite Lemmas). *Let $s \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(\overline{n})} t$ be a rewrite lemma, where ih, ib, and is are as in Def. 11. Then we obtain $\mathit{rt}(\overline{n}) = \mathit{ih}^n \cdot \mathit{ib}(\widetilde{n}) + \sum_{i=0}^{n-1} \mathit{ih}^{n-1-i} \cdot \mathit{is}(\overline{n}[n/i])$.*

▶ **Example 13** (Computing rt). Reconsider $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(n)} \gamma_{\mathbf{List}}(n)$ from Ex. 10. The proof of the induction base is $\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \equiv_{\mathcal{G}} \mathsf{qs}(\mathsf{nil}) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{qs}}} \mathsf{nil} \equiv_{\mathcal{G}} \gamma_{\mathbf{List}}(0)$. Hence, $\mathit{ib} = \mathit{rt}_1 = 1$. The proof of the induction step is as follows. Here, we use that the runtime of both previously proved lemmas (5) and (6) is $3n+1$. Note that the non-overlap condition required by the relation $\xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{qs}}}$ is clearly satisfied in all steps with $\xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{qs}}}$ in the proof.

$$\begin{array}{rcll|l}
\mathsf{qs}(\gamma_{\mathbf{List}}(n+1)) & \equiv_{\mathcal{G}} & \mathsf{qs}(\mathsf{cons}(\gamma_{\mathbf{Nats}}(0), \gamma_{\mathbf{List}}(n))) & \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{qs}}} & \mathit{rt}'_1 = 1 \\
\mathsf{qs}(\mathsf{low}(\gamma_{\mathbf{Nats}}(0), \gamma_{\mathbf{List}}(n))) \;\mathsf{++}\; \mathsf{cons}(\gamma_{\mathbf{Nats}}(0), \mathsf{qs}(\mathsf{high}(\ldots))) & & & \to_{\mathcal{L}} & \mathit{rt}'_2(n) = 3n+1 \\
\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \;\mathsf{++}\; \mathsf{cons}(\gamma_{\mathbf{Nats}}(0), \mathsf{qs}(\mathsf{high}(\gamma_{\mathbf{Nats}}(0), \gamma_{\mathbf{List}}(n)))) & & & \to_{\mathcal{L}} & \mathit{rt}'_3(n) = 3n+1 \\
\mathsf{qs}(\gamma_{\mathbf{List}}(0)) \;\mathsf{++}\; \mathsf{cons}(\gamma_{\mathbf{Nats}}(0), \mathsf{qs}(\gamma_{\mathbf{List}}(n))) & & & \equiv_{\mathcal{G}} & \\
\mathsf{qs}(\mathsf{nil}) \;\mathsf{++}\; \mathsf{cons}(\mathsf{zero}, \mathsf{qs}(\gamma_{\mathbf{List}}(n))) & & & \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{qs}}} & \mathit{rt}'_4 = 1 \\
\mathsf{nil} \;\mathsf{++}\; \mathsf{cons}(\mathsf{zero}, \mathsf{qs}(\gamma_{\mathbf{List}}(n))) & & & \mapsto_{\mathrm{IH}} & \mathit{rt}'_5(n) = \mathit{rt}(n) \\
\mathsf{nil} \;\mathsf{++}\; \mathsf{cons}(\mathsf{zero}, \gamma_{\mathbf{List}}(n)) & & & \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\mathsf{qs}}} & \mathit{rt}'_6 = 1 \\
\mathsf{cons}(\mathsf{zero}, \gamma_{\mathbf{List}}(n)) & \equiv_{\mathcal{G}} & \gamma_{\mathbf{List}}(n+1) & &
\end{array}$$

$$\begin{array}{rcccl}
\text{Hence, } \mathit{is}(n) & = & \sum_{j\in\{1,\ldots,6\},\ p_j\to q_j\neq \mathrm{IH}} \mathit{rt}'_j(\overline{z}_j\theta_j) & = & \mathit{rt}'_1 + \mathit{rt}'_2(n) + \mathit{rt}'_3(n) + \mathit{rt}'_4 + \mathit{rt}'_6 \\
& = & 1 + (3n+1) + (3n+1) + 1 + 1 & = & 6n+5.
\end{array}$$

In our example, we have $\mathit{ih} = 1$. So Thm. 12 implies $\mathit{rt}(n) = \mathit{ib} + \sum_{i=0}^{n-1} \mathit{is}(i) = 1 + \sum_{i=0}^{n-1}(6i+5)$ $= 3n^2 + 2n + 1$. Thus, we get the rewrite lemma (4): $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{\mathrm{i}}{}^{3n^2+2n+1} \gamma_{\mathbf{List}}(n)$.

To compute asymptotic bounds for the complexity of a TRS afterwards, we have to infer asymptotic bounds for the runtime of rewrite lemmas. Based on Thm. 12, such bounds can be automatically obtained from the induction proofs of the lemmas. To ease the formulation of bounds for $\mathit{rt} : \mathbb{N}^m \to \mathbb{N}$, we define the unary function $\mathit{rt}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ as $\mathit{rt}_{\mathbb{N}}(n) = \mathit{rt}(n, \ldots, n)$.

If the induction hypothesis was not used in the proof of a rewrite lemma (i.e., $\mathit{ih} = 0$), then we have $\mathit{rt}(\overline{n}[n/0]) = \mathit{ib}(\widetilde{n})$ and $\mathit{rt}(\overline{n}[n/n+1]) = \mathit{is}(\overline{n})$. Thus, if ib and is are polynomials of degree d_{ib} and d_{is}, respectively, then we obtain $\mathit{rt}_{\mathbb{N}}(n) \in \Omega(n^{\max\{d_{\mathit{ib}}, d_{\mathit{is}}\}})$.

If $\mathit{ih} = 1$, then Thm. 12 implies $\mathit{rt}(\overline{n}) = \mathit{ib}(\widetilde{n}) + \sum_{i=0}^{n-1} \mathit{is}(\overline{n}[n/i])$. Again, let ib and is be polynomials of degree d_{ib} and d_{is}, respectively. Then $\mathit{is}(\overline{n}) = t_0 + t_1 n + t_2 n^2 + \ldots + t_{d_{\mathit{is}}} n^{d_{\mathit{is}}}$, where the t_j are polynomials of degree at most $d_{\mathit{is}} - j$ containing variables from $\widetilde{n}$. Hence, $\mathit{rt}(\overline{n}) =$

$$\mathit{ib}(\widetilde{n}) + \sum_{i=0}^{n-1} (t_0 + t_1 i + t_2 i^2 + \ldots + t_{d_{\mathit{is}}} i^{d_{\mathit{is}}}) = \mathit{ib}(\widetilde{n}) + t_0 \cdot \sum_{i=0}^{n-1} i^0 + t_1 \cdot \sum_{i=0}^{n-1} i^1 + t_2 \cdot \sum_{i=0}^{n-1} i^2 + \ldots + t_{d_{\mathit{is}}} \cdot \sum_{i=0}^{n-1} i^{d_{\mathit{is}}}.$$

By Faulhaber's formula [15], for any $e \in \mathbb{N}$, $\sum_{i=0}^{n-1} i^e$ is a polynomial over the variable n of degree $e+1$. For example if $e = 1$, then $\sum_{i=0}^{n-1} i^1 = \frac{n\cdot(n-1)}{2}$ has degree 2. By taking also the degree d_{ib} of ib into account, rt has degree $\max\{d_{\mathit{ib}}, d_{\mathit{is}} + 1\}$, i.e., $\mathit{rt}_{\mathbb{N}}(n) \in \Omega(n^{\max\{d_{\mathit{ib}}, d_{\mathit{is}}+1\}})$.

Finally we consider the case where the induction hypothesis was used several times, i.e., $\mathit{ih} > 1$. By construction we always have $\mathit{is}(\overline{n}) \geq 1$ (since the induction step cannot only consist of applying the induction hypothesis). Thus, Thm. 12 implies $\mathit{rt}(\overline{n}) \geq \sum_{i=0}^{n-1} \mathit{ih}^{n-1-i} = \sum_{j=0}^{n-1} \mathit{ih}^j = \frac{\mathit{ih}^n - 1}{\mathit{ih} - 1}$. So $\mathit{rt}_{\mathbb{N}}(n) \in \Omega(\mathit{ih}^n)$, i.e., the runtime of the rewrite lemma is exponential.

▶ **Theorem 14** (Asymptotic Runtime of Rewrite Lemmas). *Let $s \xrightarrow{\mathrm{i}}{}^{\mathit{rt}(\overline{n})} t$ be a rewrite lemma with ih, ib, and is as in Def. 11. Moreover, let ib and is be polynomials of degree d_{ib} and d_{is}.*

- *If* $ih = 0$, *then* $rt_{\mathbb{N}}(n) \in \Omega(n^{\max\{d_{ib}, d_{is}\}})$.
- *If* $ih = 1$, *then* $rt_{\mathbb{N}}(n) \in \Omega(n^{\max\{d_{ib}, d_{is}+1\}})$.
- *If* $ih > 1$, *then* $rt_{\mathbb{N}}(n) \in \Omega(ih^n)$.

▶ **Example 15** (Exponential Runtime)**.** Consider the TRS $\mathcal{R}_{exp}$ with the rules $\mathsf{f}(\mathsf{succ}(x), \mathsf{succ}(x)) \to \mathsf{f}(\mathsf{f}(x,x), \mathsf{f}(x,x))$ and $\mathsf{f}(\mathsf{zero}, \mathsf{zero}) \to \mathsf{zero}$. Our approach speculates and proves the rewrite lemma $\mathsf{f}(\gamma_{\mathbf{Nats}}(n), \gamma_{\mathbf{Nats}}(n)) \xrightarrow{\mathrm{i}}{}^{rt(n)} \mathsf{zero}$. For the induction base, we have $\mathsf{f}(\gamma_{\mathbf{Nats}}(0), \gamma_{\mathbf{Nats}}(0)) \equiv_{\mathcal{G}} \mathsf{f}(\mathsf{zero}, \mathsf{zero}) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{exp}} \mathsf{zero}$ and thus $ib = 1$. The induction step is proved as follows:

$$\begin{array}{lll|l} \mathsf{f}(\gamma_{\mathbf{Nats}}(n+1), \gamma_{\mathbf{Nats}}(n+1)) & \equiv_{\mathcal{G}} & \mathsf{f}(\mathsf{succ}(\gamma_{\mathbf{Nats}}(n)), \mathsf{succ}(\gamma_{\mathbf{Nats}}(n))) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{exp}} & rt'_1 = 1 \\ & & \mathsf{f}(\mathsf{f}(\gamma_{\mathbf{Nats}}(n), \gamma_{\mathbf{Nats}}(n)), \mathsf{f}(\gamma_{\mathbf{Nats}}(n), \gamma_{\mathbf{Nats}}(n))) \mapsto^{2}_{\mathrm{IH}} & \\ & & \mathsf{f}(\mathsf{zero}, \mathsf{zero}) \xrightarrow{\mathrm{i}}_{\mathcal{R}_{exp}} & rt'_4 = 1 \\ & & \mathsf{zero} & \end{array}$$

Thus, $ih = 2$ and $is(n)$ is the constant 2 for all $n \in \mathbb{N}$. Hence, by Thm. 14 we have $rt(n) \in \Omega(2^n)$. Indeed, Thm. 12 implies $rt(n) = 2^n + \sum_{i=0}^{n-1} 2^{n-1-i} \cdot 2 = 2^{n+1} + 2^n - 2$.

5 Inferring Bounds for TRSs

We now use rewrite lemmas to infer lower bounds for the *innermost runtime complexity* $\mathrm{irc}_{\mathcal{R}}$ of a TRS $\mathcal{R}$. To define $\mathrm{irc}_{\mathcal{R}}$, the *derivation height* of a term t w.r.t. a relation $\to$ is the length of the longest $\to$-sequence starting with t, i.e., $\mathrm{dh}(t, \to) = \sup\{\, m \mid \exists t' \in \mathcal{T}(\Sigma, \mathcal{V}),\ t \to^m t' \,\}$, cf. [13]. Here, for any $M \subseteq \mathbb{N} \cup \{\omega\}$, $\sup M$ is the least upper bound of M and $\sup \varnothing = 0$. Since we only regard finite TRSs, $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \omega$ iff t starts an infinite sequence of $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-steps. So as in [17], dh treats terminating and non-terminating terms in a uniform way.

When analyzing the complexity of *programs*, one is interested in evaluations of *basic terms* $f(t_1, \ldots, t_k)$ where a defined symbol $f \in \Sigma_{def}(\mathcal{R})$ is applied to data objects $t_1, \ldots, t_k \in \mathcal{T}(\Sigma_{con}(\mathcal{R}), \mathcal{V})$. The *innermost runtime complexity function* $\mathrm{irc}_{\mathcal{R}}$ corresponds to the usual notion of "complexity" for programs. It maps any $n \in \mathbb{N}$ to the length of the longest sequence of $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-steps starting with a basic term t with $|t| \leq n$. Here, the *size* of a term is $|x| = 1$ for $x \in \mathcal{V}$ and $|f(t_1, \ldots, t_k)| = 1 + |t_1| + \ldots + |t_k|$, and $\mathcal{T}_B$ is the set of all basic terms.

▶ **Definition 16** (Innermost Runtime Complexity $\mathrm{irc}_{\mathcal{R}}$ [12])**.** For a TRS $\mathcal{R}$, its *innermost runtime complexity function* $\mathrm{irc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ is $\mathrm{irc}_{\mathcal{R}}(n) = \sup\{\, \mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}_B, |t| \leq n \,\}$.

In Sect. 4 we computed the length $rt(\overline{n})$ of the rewrite sequences represented by a rewrite lemma $s \xrightarrow{\mathrm{i}}{}^{rt(\overline{n})} t$, where $\mathcal{V}(s) = \overline{n}$. However, $\mathrm{irc}_{\mathcal{R}}$ is defined w.r.t. the size of the start term of a rewrite sequence. Thus, to obtain a lower bound for $\mathrm{irc}_{\mathcal{R}}$ from $rt(\overline{n})$, for any $\sigma : \mathcal{V}(s) \to \mathbb{N}$ one has to take the relation between $\overline{n}\sigma$ and the size of the start term $s\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}}$ into account. Note that our approach in Sect. 2 only speculates lemmas where s has the form $f(\gamma_{\tau_1}(s_1), \ldots, \gamma_{\tau_k}(s_k))$. Here, $f \in \Sigma_{def}(\mathcal{R})$, $s_1, \ldots, s_k$ are polynomials over $\overline{n}$, and $\tau_1, \ldots, \tau_k$ are simply structured types. For any τ_i, let $d_{\tau_i} : \rho_1 \times \cdots \times \rho_b \to \tau$ be τ_i's recursive constructor. Then for any $n \in \mathbb{N}$, Def. 3 implies $|\gamma_{\tau_i}(n){\downarrow_{\mathcal{G}/\mathcal{A}}}| = sz_{\tau_i}(n)$ for $sz_{\tau_i} : \mathbb{N} \to \mathbb{N}$ with

$$sz_{\tau_i}(n) = |\gamma_{\tau_i}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}| + n \cdot \left(1 + |\gamma_{\rho_1}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}| + \cdots + |\gamma_{\rho_b}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}| - |\gamma_{\tau_i}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}|\right).$$

The reason is that $\gamma_{\tau_i}(n){\downarrow_{\mathcal{G}/\mathcal{A}}}$ contains n occurrences of d_{τ_i} and of each $\gamma_{\rho_1}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}, \ldots, \gamma_{\rho_b}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}$ except $\gamma_{\tau_i}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}$, and just one occurrence of $\gamma_{\tau_i}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}$. For instance, $|\gamma_{\mathbf{Nats}}(n){\downarrow_{\mathcal{G}/\mathcal{A}}}|$ is $sz_{\mathbf{Nats}}(n) = |\gamma_{\mathbf{Nats}}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}| + n \cdot (1 + |\gamma_{\mathbf{Nats}}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}| - |\gamma_{\mathbf{Nats}}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}|) = |\mathsf{zero}| + n = 1 + n$ and $|\gamma_{\mathbf{List}}(n){\downarrow_{\mathcal{G}/\mathcal{A}}}|$ is $sz_{\mathbf{List}}(n) = |\gamma_{\mathbf{List}}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}| + n \cdot (1 + |\gamma_{\mathbf{Nats}}(0){\downarrow_{\mathcal{G}/\mathcal{A}}}|) = |\mathsf{nil}| + n \cdot (1 + |\mathsf{zero}|) = 1 + n \cdot 2$. Consequently, the size of $s{\downarrow_{\mathcal{G}/\mathcal{A}}} = f(\gamma_{\tau_1}(s_1), \ldots, \gamma_{\tau_k}(s_k)){\downarrow_{\mathcal{G}/\mathcal{A}}}$ with $\mathcal{V}(s) = \overline{n}$ is given by the following function $sz : \mathbb{N}^m \to \mathbb{N}$:

$$sz(\overline{n}) = 1 + sz_{\tau_1}(s_1) + \cdots + sz_{\tau_k}(s_k)$$

For instance, the term $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \downarrow_{\mathcal{G}/\mathcal{A}} = \mathsf{qs}(\mathsf{cons}^n(\mathsf{zero}, \mathsf{nil}))$ has the size $sz(n) = 1 + sz_{\mathbf{List}}(n) = 2n + 2$. Since $|\gamma_\tau(0)\downarrow_{\mathcal{G}/\mathcal{A}}|$ is a constant for each type τ, sz is a polynomial whose degree is given by the maximal degree of the polynomials $s_1, \ldots, s_k$.

So the rewrite lemma (4) for qs states that there are terms of size $sz(n) = 2n + 2$ with reductions of length $rt(n) = 3n^2 + 2n + 1$. To determine a lower bound for $\mathrm{irc}_{\mathcal{R}_{\mathsf{qs}}}$, we construct an inverse function sz^{-1} with $(sz \circ sz^{-1})(n) = n$. In our example where $sz(n) = 2n+2$, we have $sz^{-1}(n) = \frac{n-2}{2}$ if n is even. So there are terms of size $sz(sz^{-1}(n)) = n$ with reductions of length $rt(sz^{-1}(n)) = rt(\frac{n-2}{2}) = \frac{3}{4}n^2 - 2n + 2$. Since multivariate polynomials $sz(n_1, \ldots, n_m)$ cannot be inverted, we invert the unary function $sz_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ with $sz_{\mathbb{N}}(n) = sz(n, \ldots, n)$ instead.

Of course, inverting $sz_{\mathbb{N}}$ fails if $sz_{\mathbb{N}}$ is not injective. However, the conjectures speculated in Sect. 2 only contain polynomials with natural coefficients. Then, $sz_{\mathbb{N}}$ is always strictly monotonically increasing. So we only proceed if there is a $sz_{\mathbb{N}}^{-1} : \mathrm{img}(sz_{\mathbb{N}}) \to \mathbb{N}$ where $(sz_{\mathbb{N}} \circ sz_{\mathbb{N}}^{-1})(n) = n$ holds for all $n \in \mathrm{img}(sz_{\mathbb{N}}) = \{n \in \mathbb{N} \mid \exists v \in \mathbb{N}.\ sz_{\mathbb{N}}(v) = n\}$. To extend $sz_{\mathbb{N}}^{-1}$ to a function on $\mathbb{N}$, for any (total) function $h : M \to \mathbb{N}$ with $M \subseteq \mathbb{N}$, we define $\lfloor h \rfloor(n) : \mathbb{N} \to \mathbb{N}$ by:

$$\lfloor h \rfloor(n) = h(\max\{n' \mid n' \in M, n' \leq n\}),\ \text{if } n \geq \min(M) \quad \text{and} \quad \lfloor h \rfloor(n) = 0,\ \text{otherwise}$$

Using this notation, the following theorem states how we can derive lower bounds for $\mathrm{irc}_{\mathcal{R}}$.

▶ **Theorem 17** (Explicit Lower Bounds for $\mathrm{irc}_{\mathcal{R}}$). *Let $s \xrightarrow{\mathrm{i}}^{rt(n_1,\ldots,n_m)} t$ be a rewrite lemma for $\mathcal{R}$, let $sz : \mathbb{N}^m \to \mathbb{N}$ be a function such that $sz(b_1, \ldots, b_m)$ is the size of $s[n_1/b_1, \ldots, n_m/b_m]\downarrow_{\mathcal{G}/\mathcal{A}}$ for all $b_1, \ldots, b_m \in \mathbb{N}$, and let $sz_{\mathbb{N}}$'s inverse function $sz_{\mathbb{N}}^{-1}$ exist. Then $rt_{\mathbb{N}} \circ \lfloor sz_{\mathbb{N}}^{-1} \rfloor$ is a lower bound for $\mathrm{irc}_{\mathcal{R}}$, i.e., $(rt_{\mathbb{N}} \circ \lfloor sz_{\mathbb{N}}^{-1} \rfloor)(n) \leq \mathrm{irc}_{\mathcal{R}}(n)$ holds for all $n \in \mathbb{N}$ with $n \geq \min(\mathrm{img}(sz_{\mathbb{N}}))$.*

So for the rewrite lemma (4) for qs where $sz_{\mathbb{N}}(n) = 2n + 2$, we have $\lfloor sz_{\mathbb{N}}^{-1} \rfloor(n) = \lfloor \frac{n-2}{2} \rfloor \geq \frac{n-3}{2}$ and $\mathrm{irc}_{\mathcal{R}_{\mathsf{qs}}}(n) \geq rt(\lfloor sz_{\mathbb{N}}^{-1} \rfloor(n)) \geq rt(\frac{n-3}{2}) = \frac{3}{4}n^2 - \frac{7}{2}n + \frac{19}{4}$ for all $n \geq 2$.

However, even if $sz_{\mathbb{N}}^{-1}$ exists, finding resp. approximating $sz_{\mathbb{N}}^{-1}$ automatically can be non-trivial in general. Therefore, we now show how to obtain an asymptotic lower bound for $\mathrm{irc}_{\mathcal{R}}$ directly from a rewrite lemma $f(\gamma_{\tau_1}(s_1), \ldots, \gamma_{\tau_k}(s_k)) \xrightarrow{\mathrm{i}}^{rt(\overline{n})} t$ without constructing $sz_{\mathbb{N}}^{-1}$. As mentioned, if e is the maximal degree of the polynomials $s_1, \ldots, s_k$, then sz is also a polynomial of degree e and thus, $sz_{\mathbb{N}}(n) \in \mathcal{O}(n^e)$. Moreover, from the induction proof of the rewrite lemma we obtain an asymptotic lower bound for $rt_{\mathbb{N}}$, cf. Thm. 14. Using these bounds, Lemma 18 can be used to infer an asymptotic lower bound for $\mathrm{irc}_{\mathcal{R}}$ directly.

▶ **Lemma 18** (Asymptotic Bounds for Function Composition). *Let $rt_{\mathbb{N}}, sz_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ where $sz_{\mathbb{N}} \in \mathcal{O}(n^e)$ for some $e \geq 1$ and where $sz_{\mathbb{N}}$ is strictly monotonically increasing.*

- *If $rt_{\mathbb{N}}(n) \in \Omega(n^d)$ with $d \geq 0$, then $(rt_{\mathbb{N}} \circ \lfloor sz_{\mathbb{N}}^{-1} \rfloor)(n) \in \Omega(n^{\frac{d}{e}})$.*
- *If $rt_{\mathbb{N}}(n) \in \Omega(b^n)$ with $b \geq 1$, then $(rt_{\mathbb{N}} \circ \lfloor sz_{\mathbb{N}}^{-1} \rfloor)(n) \in \Omega(b^{\sqrt[e]{n}})$.*

So for the rewrite lemma $\mathsf{qs}(\gamma_{\mathbf{List}}(n)) \xrightarrow{\mathrm{i}}^{rt(n)} \gamma_{\mathbf{List}}(n)$ where $rt_{\mathbb{N}} = rt$ and $sz_{\mathbb{N}} = sz$, we only need the asymptotic bounds $sz(n) \in \mathcal{O}(n)$ and $rt(n) \in \Omega(n^2)$, to conclude $\mathrm{irc}_{\mathcal{R}_{\mathsf{qs}}}(n) \in \Omega(n^{\frac{2}{1}}) = \Omega(n^2)$, i.e., to prove that the quicksort TRS has at least quadratic complexity.

So while Thm. 17 explains how to find concrete lower bounds for $\mathrm{irc}_{\mathcal{R}}$ (if $sz_{\mathbb{N}}$ can be inverted), the following theorem summarizes our results on asymptotic lower bounds for $\mathrm{irc}_{\mathcal{R}}$. To this end, we combine Thm. 14 on the inference of asymptotic bounds for rt with Lemma 18.

▶ **Theorem 19** (Asymptotic Lower Bounds for $\mathrm{irc}_{\mathcal{R}}$). *Let $s \xrightarrow{\mathrm{i}}^{rt(\overline{n})} t$ be a rewrite lemma for $\mathcal{R}$ and let $sz : \mathbb{N}^m \to \mathbb{N}$ be a function such that $sz(b_1, \ldots, b_m)$ is the size of $s[n_1/b_1, \ldots, n_m/b_m]\downarrow_{\mathcal{G}/\mathcal{A}}$ for all $b_1, \ldots, b_m \in \mathbb{N}$, where $sz_{\mathbb{N}}(n) \in \mathcal{O}(n^e)$ for some $e \geq 1$ and $sz_{\mathbb{N}}$ is strictly monotonically increasing. Furthermore, let ih, ib, and is be defined as in Def. 11.*

1. *If $ih = 0$ and ib and is are polynomials of degree d_{ib} and d_{is}, then $\mathrm{irc}_{\mathcal{R}}(n) \in \Omega(n^{\frac{\max\{d_{ib}, d_{is}\}}{e}})$.*

2. *If* $i\!k = 1$ *and* $i\!b$ *and* $i\!s$ *are polynomials of degree* $d_{i\!b}$ *and* $d_{i\!s}$*, then* $\mathrm{irc}_{\mathcal{R}}(n) \in \Omega(n^{\frac{\max\{d_{i\!b}, d_{i\!s}+1\}}{e}})$.
3. *If* $i\!k > 1$*, then* $\mathrm{irc}_{\mathcal{R}}(n) \in \Omega(i\!k^{\sqrt[e]{n}})$.

6 Preprocessing TRSs by Argument Filtering

A drawback of our approach is that generator functions only represent homogeneous data objects (e.g., lists or trees where all elements have the same value zero). To prove lower complexity bounds also in cases where one needs other forms of rewrite lemmas, we use *argument filtering* [2] to remove certain argument positions of function symbols.

▶ **Example 20** (Argument Filtering)**.** Consider the following TRS $\mathcal{R}_{\mathsf{intlist}}$:

$$\mathsf{intlist}(\mathsf{zero}) \to \mathsf{nil} \qquad \mathsf{intlist}(\mathsf{succ}(x)) \to \mathsf{cons}(x, \mathsf{intlist}(x))$$

We have $\mathsf{intlist}(\mathsf{succ}^n(\mathsf{zero})) \xrightarrow{i}{}^{n+1} \mathsf{cons}(\mathsf{succ}^{n-1}(\mathsf{zero}), \ldots \mathsf{cons}(\mathsf{succ}(\mathsf{zero}), \mathsf{cons}(\mathsf{zero}, \mathsf{nil})))$ for all $n \in \mathbb{N}$. However, the inhomogeneous list on the right-hand side cannot be expressed in a rewrite lemma. Filtering away the first argument of cons yields the TRS $(\mathcal{R}_{\mathsf{intlist}})_{\backslash(\mathsf{cons},1)}$:

$$\mathsf{intlist}(\mathsf{zero}) \to \mathsf{nil} \qquad \mathsf{intlist}(\mathsf{succ}(x)) \to \mathsf{cons}(\mathsf{intlist}(x))$$

For this TRS, our approach can speculate and prove the rewrite lemma $\mathsf{intlist}(\gamma_{\mathbf{Nats}}(n)) \xrightarrow{i}{}^{n+1} \gamma_{\mathbf{List}}(n)$, i.e., $\mathsf{intlist}(\mathsf{succ}^n(\mathsf{zero})) \xrightarrow{i}{}^{n+1} \mathsf{cons}^n(\mathsf{nil})$. From this rewrite lemma, one can infer $n - 1 \leq \mathrm{irc}_{(\mathcal{R}_{\mathsf{intlist}})_{\backslash(\mathsf{cons},1)}}(n)$ for all $n \geq 2$ resp. $\mathrm{irc}_{(\mathcal{R}_{\mathsf{intlist}})_{\backslash(\mathsf{cons},1)}}(n) \in \Omega(n)$.

Def. 21 introduces the concept of argument filtering for terms and TRSs formally.

▶ **Definition 21** (Argument Filtering)**.** Let Σ be a signature with $f \in \Sigma$, $\mathrm{ar}_\Sigma(f) = k$, and let $i \in \{1, \ldots, k\}$. Let $\Sigma_{\backslash(f,i)}$ be like Σ, but with $\mathrm{ar}_{\Sigma_{\backslash(f,i)}}(f) = k - 1$. For any term $t \in \mathcal{T}(\Sigma, \mathcal{V})$, we define the term $t_{\backslash(f,i)} \in \mathcal{T}(\Sigma_{\backslash(f,i)}, \mathcal{V})$ resulting from *filtering* away the i-th argument of f:

$$t_{\backslash(f,i)} = \begin{cases} t, & \text{if } t \text{ is a variable} \\ f(\,(t_1)_{\backslash(f,i)}, \ldots, (t_{i-1})_{\backslash(f,i)}, (t_{i+1})_{\backslash(f,i)}, \ldots, (t_k)_{\backslash(f,i)}\,), & \text{if } t = f(t_1, \ldots, t_k) \\ g(\,(t_1)_{\backslash(f,i)}, \ldots, (t_b)_{\backslash(f,i)}\,), & \text{if } t = g(t_1, \ldots, t_b) \text{ for } g \neq f \end{cases}$$

Let $\mathcal{R}$ be a TRS over Σ. Then we define $\mathcal{R}_{\backslash(f,i)} = \{\ell_{\backslash(f,i)} \to r_{\backslash(f,i)} \mid \ell \to r \in \mathcal{R}\}$.

However, a lower bound for the runtime of $\mathcal{R}_{\backslash(f,i)}$ does not imply a lower bound for $\mathcal{R}$ if the argument that is filtered away influences the control flow of the evaluation. Thus, several conditions have to be imposed to ensure that argument filtering is sound for lower bounds:

(a) Argument filtering must not remove function symbols on left-hand sides of rules. An argument may not be filtered away if it is used for non-trivial pattern matching (i.e., if there is a left-hand side of a rule where the i-th argument of f is not a variable). As an example, consider $\mathcal{R} = \{\mathsf{f}(\mathsf{cons}(\mathsf{true}, xs)) \to \mathsf{f}(\mathsf{cons}(\mathsf{false}, xs))\}$ where $\mathrm{irc}_{\mathcal{R}}(n) \leq 1$ for all n. But if one filters away the first argument of cons, then one obtains the non-terminating rule $\mathsf{f}(\mathsf{cons}(xs)) \to \mathsf{f}(\mathsf{cons}(xs))$, i.e., $\mathrm{irc}_{\mathcal{R}_{\backslash(\mathsf{cons},1)}}(n) = \omega$ for $n \geq 3$.

(b) The TRS must be left-linear. To illustrate this, consider $\mathcal{R} = \{\mathsf{f}(xs, xs) \to \mathsf{f}(\mathsf{cons}(\mathsf{true}, xs), \mathsf{cons}(\mathsf{false}, xs))\}$, where again $\mathrm{irc}_{\mathcal{R}}(n) \leq 1$. But filtering away the first argument of cons yields the non-terminating rule $\mathsf{f}(xs, xs) \to \mathsf{f}(\mathsf{cons}(xs), \mathsf{cons}(xs))$, i.e., $\mathrm{irc}_{\mathcal{R}_{\backslash(\mathsf{cons},1)}}(n) = \omega$ for $n \geq 3$.

(c) Argument filtering must not result in free variables on right-hand sides of rules. The reason is that otherwise, argument filtering might again turn terminating TRSs into non-terminating ones. For instance, consider $\mathcal{R} = \{\mathsf{f}(\mathsf{cons}(x, xs)) \to \mathsf{f}(xs)\}$ where $\mathrm{irc}_{\mathcal{R}}(n) = \lfloor \frac{n}{2} \rfloor - 1$. But if one filters away the second argument of cons, then one gets the rule $\mathsf{f}(\mathsf{cons}(x)) \to \mathsf{f}(xs)$ whose runtime is unbounded, i.e., $\mathrm{irc}_{\mathcal{R}_{\backslash(\mathsf{cons},2)}}(n) = \omega$ for $n \geq 3$.

Thm. 22 states that (a) - (c) are indeed sufficient for the soundness of argument filtering. To infer a lower bound for $\mathrm{irc}_{\mathcal{R}}$ from a bound for $\mathrm{irc}_{\mathcal{R}_{\setminus(f,i)}}$, we have to take into account that filtering changes the size of terms. As an example, consider $\mathcal{R} = \{\mathsf{f}(x) \to \mathsf{a}\}$. Here, we have $\mathrm{irc}_{\mathcal{R}_{\setminus(\mathsf{f},1)}}(1) = 1$ due to the rewrite sequence $\mathsf{f} \xrightarrow{\mathrm{i}}_{\mathcal{R}_{\setminus(\mathsf{f},1)}} \mathsf{a}$. The corresponding rewrite sequence in the original TRS $\mathcal{R}$ is $\mathsf{f}(x) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{a}$. Thus, $\mathrm{irc}_{\mathcal{R}}(2) = 1$, but all terms of size 1 are normal forms of $\mathcal{R}$, i.e., $\mathrm{irc}_{\mathcal{R}}(1) = 0$. So $\mathrm{irc}_{\mathcal{R}_{\setminus(f,i)}}(n) \leq \mathrm{irc}_{\mathcal{R}}(n)$ does not hold in general. Nevertheless, for any rewrite sequence of $\mathcal{R}_{\setminus(f,i)}$ starting with a term t, there is a corresponding rewrite sequence of $\mathcal{R}$ starting with a term[7] s where $|s| \leq 2 \cdot |t|$. Thus, if we have derived a lower bound $p(n)$ for $\mathrm{irc}_{\mathcal{R}_{\setminus(f,i)}}(n)$, we can use $p(\frac{n}{2})$ as a lower bound for $\mathrm{irc}_{\mathcal{R}}(n)$. Hence, in Ex. 20, we obtain $\frac{n}{2} - 1 \leq \mathrm{irc}_{\mathcal{R}_{\mathsf{intlist}}}(n)$ for all $n \geq 4$ resp. $\mathrm{irc}_{\mathcal{R}_{\mathsf{intlist}}}(n) \in \Omega(n)$.

▶ **Theorem 22** (Soundness of Argument Filtering). *Let $f \in \Sigma$, $\mathrm{ar}_{\Sigma}(f) = k$, and $i \in \{1, \ldots, k\}$. Moreover, let $\mathcal{R}$ be a TRS over Σ where the following conditions hold for all rules $\ell \to r \in \mathcal{R}$:*
(a) *If $f(t_1, \ldots, t_k)$ is a subterm of ℓ, then $t_i \in \mathcal{V}$.*
(b) *For any $x \in \mathcal{V}$, there is at most one position $\pi \in \mathrm{pos}(\ell)$ such that $\ell|_\pi = x$.*
(c) *$\mathcal{V}(r_{\setminus(f,i)}) \subseteq \mathcal{V}(\ell_{\setminus(f,i)})$.*
Then for all $n \in \mathbb{N}$, we have $\mathrm{irc}_{\mathcal{R}_{\setminus(f,i)}}(\frac{n}{2}) \leq \mathrm{irc}_{\mathcal{R}}(n)$.

In our implementation, as a heuristic we always perform argument filtering if it is permitted by Thm. 22, except for cases where filtering removes defined function symbols on right-hand sides of rules. As an example, consider $\mathcal{R} = \{\mathsf{a} \to \mathsf{f}(\mathsf{a}, \mathsf{b})\}$ where $\mathrm{irc}_{\mathcal{R}}(n) = \omega$ for $n \geq 1$. If one filters away f's first argument, then one obtains $\mathsf{a} \to \mathsf{f}(\mathsf{b})$ and thus, $\mathrm{irc}_{\mathcal{R}_{\setminus(\mathsf{f},1)}}(n) = 1$ for $n \geq 1$. So here, argument filtering is sound, but it results in significantly worse lower bounds.

7 Indefinite Rewrite Lemmas

Our technique often fails if the analyzed TRS is not completely defined, i.e., if there are normal forms containing defined symbols. As an example, the runtime complexity of $\mathcal{R}_{in} = \{\mathsf{f}(\mathsf{succ}(x)) \to \mathsf{succ}(\mathsf{f}(x))\}$ is linear due to the rewrite sequences $\mathsf{f}(\mathsf{succ}^n(\mathsf{zero})) \xrightarrow{\mathrm{i}}{}^n \mathsf{succ}^n(\mathsf{f}(\mathsf{zero}))$. However, the term $\mathsf{succ}^n(\mathsf{f}(\mathsf{zero}))$ on the right-hand side contains f and thus, it cannot be represented in a rewrite lemma. Therefore, we now also allow *indefinite* conjectures and rewrite lemmas with unknown right-hand sides. Then for our example, we could speculate the indefinite conjecture $\mathsf{f}(\gamma_{\mathbb{N}}(n)) \xrightarrow{\mathrm{i}}{}^* \star$, which gives rise to the indefinite rewrite lemma $\mathsf{f}(\gamma_{\mathbb{N}}(n)) \xrightarrow{\mathrm{i}}{}^n \star$, where $\star$ represents an arbitrary unknown term. To distinguish indefinite conjectures and rewrite lemmas from ordinary ones, we call the latter *definite*.

Recall that when speculating conjectures in Sect. 2, we built a narrowing tree for a term $s = f(\ldots)$ and obtained a sample point (t, σ, d) whenever we reached a normal form t. When speculating indefinite conjectures, we do not narrow in order to reach normal forms, but we create a sample point (σ, d) after each application of a recursive f-rule. Here, σ is again the substitution and d is the recursion depth of the path. Note that while previously proven lemmas $\mathcal{L}$ may be used during narrowing, we do not use previous indefinite rewrite lemmas, since they do not yield any information on the terms resulting from rewriting.

▶ **Example 23** (Speculating Indefinite Conjectures). For $\mathcal{R}_{in}$, we narrow the term $s = \mathsf{f}(\gamma_{\mathbf{Nats}}(x))$. We get $\mathsf{f}(\gamma_{\mathbf{Nats}}(x)) \rightsquigarrow \mathsf{succ}(\mathsf{f}(\gamma_{\mathbf{Nats}}(x')))$ with the substitution $\sigma_1 = [x/x' + 1]$. Since we applied a recursive f-rule once, we construct the sample point $(\sigma_1, 1)$. We continue narrowing and get $\mathsf{succ}(\mathsf{f}(\gamma_{\mathbf{Nats}}(x'))) \rightsquigarrow \mathsf{succ}(\mathsf{succ}(\mathsf{f}(\gamma_{\mathbf{Nats}}(x''))))$ with the substitution $\sigma_2 =$

[7] The term s can be obtained from t by adding a variable as the i-th argument for any f occurring in t.

$[x'/x''+1]$ and recursion depth 2. Since $\sigma_2 \circ \sigma_1$ corresponds to $[x/x''+2]$, this yields the sample point $([x/x''+2], 2)$. Another narrowing step results in the sample point $([x/x'''+3], 3)$.

These sample points represent the sample conjectures $\mathsf{f}(\gamma_{\mathbf{Nats}}(x'+1)) \xrightarrow{i}{}^{*} \star$, $\mathsf{f}(\gamma_{\mathbf{Nats}}(x''+2)) \xrightarrow{i}{}^{*} \star$, $\mathsf{f}(\gamma_{\mathbf{Nats}}(x'''+3)) \xrightarrow{i}{}^{*} \star$ that are identical up to the occurring numbers and variable names. Thus, they are suitable for generalization. As in Sect. 2, we replace the numbers in the sample conjectures by a polynomial pol in one variable n that stands for the recursion depth. This leads to $\mathsf{f}(\gamma_{\mathbf{Nats}}(x+pol)) \xrightarrow{i}{}^{*} \star$ and the constraints $pol(1)=1$, $pol(2)=2$, $pol(3)=3$. A solution is $pol = n$ and thus, we speculate the indefinite conjecture $\mathsf{f}(\gamma_{\mathbf{Nats}}(x+n)) \xrightarrow{i}{}^{*} \star$.

Every indefinite conjecture gives rise to an indefinite rewrite lemma.

▶ **Definition 24** (Indefinite Rewrite Lemmas). Let $\mathcal{R}$, s, rt be as in Def. 8. Then $s \xrightarrow{i}{}^{rt(\overline{n})} \star$ is an *indefinite rewrite lemma* for $\mathcal{R}$ iff for all $\sigma : \mathcal{V}(s) \to \mathbb{N}$ there is a term t such that $s\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}} \xrightarrow{i}{}_{\mathcal{R}}^{rt(\overline{n}\sigma)} t$, i.e., $s\sigma{\downarrow_{\mathcal{G}/\mathcal{A}}}$ starts an innermost $\mathcal{R}$-reduction of at least $rt(n_1\sigma, \ldots, n_m\sigma)$ steps.

In principle, proving indefinite conjectures $s \xrightarrow{i}{}^{*} \star$ is not necessary, since $s \xrightarrow{i}{}^{0} \star$ is always a valid indefinite rewrite lemma. However, to derive useful lower complexity bounds, we need rewrite lemmas $s \xrightarrow{i}{}^{rt(\overline{n})} \star$ for non-trivial functions rt. Thm. 25 shows that the approaches for proving lemmas from Sect. 3 and for deriving bounds from these proofs in Sect. 4 can also be used for indefinite rewrite lemmas. The only adaption needed is that the relation $\xrightarrow{i}{}_{\mathcal{R}}$ may not reduce redexes that contain the symbol $\star$. This restriction is needed due to the innermost evaluation strategy, because $\star$ represents arbitrary terms that are not necessarily in normal form. In this way, all previously proven (definite or indefinite) rewrite lemmas $\mathcal{L}$ can be used in the proof of new (definite or indefinite) rewrite lemmas.

▶ **Theorem 25** (Bounds for Indefinite Rewrite Lemmas). *Let $\xrightarrow{i}{}_{\mathcal{R}}$ and $\xrightarrow{i}{}_{(\mathcal{R},\mathrm{IH})}$ be restricted such that redexes may not contain the symbol $\star$ and let ih, ib, and is be defined as in Def. 11. Here, for an indefinite rewrite lemma $s \xrightarrow{i}{}^{rt(\overline{n})} \star$ with $n \in \mathcal{V}(s)$, we say that any rewrite sequence $s[n/0] = u_1 \xrightarrow{i}{}_{\mathcal{R}} u_2 \xrightarrow{i}{}_{\mathcal{R}} \ldots \xrightarrow{i}{}_{\mathcal{R}} u_{b+1}$ "proves" the induction base and any rewrite sequence $s[n/n+1] = v_1 \xrightarrow{i}{}_{(\mathcal{R},\mathrm{IH})} v_2 \xrightarrow{i}{}_{(\mathcal{R},\mathrm{IH})} \ldots \xrightarrow{i}{}_{(\mathcal{R},\mathrm{IH})} v_{k+1}$ "proves" the induction step, where* IH *is the rule $s \to \star$. Then Thm. 12 and Thm. 14 on explicit and asymptotic runtimes hold for any definite or indefinite rewrite lemma.*

▶ **Example 26** (Complexity of Indefinite Rewrite Lemmas). To continue with Ex. 23, we now infer the runtime for the rewrite lemma $\mathsf{f}(\gamma_{\mathbf{Nats}}(x+n)) \xrightarrow{i}{}^{rt(x,n)} \star$. Since $\mathsf{f}(\gamma_{\mathbf{Nats}}(x+0))$ is already in normal form w.r.t. $\xrightarrow{i}{}_{\mathcal{R}}$, the length of the rewrite sequence in the induction base is $ib(x) = 0$. In the induction step, we obtain $\mathsf{f}(\gamma_{\mathbf{Nats}}(x+n+1)) \xrightarrow{i}{}_{\mathcal{R}} \mathsf{succ}(\mathsf{f}(\gamma_{\mathbf{Nats}}(x+n))) \mapsto_{\mathrm{IH}} \mathsf{succ}(\star)$. Thus, the induction hypothesis is applied $ih = 1$ time and the number of remaining rewrite steps is $is(x,n) = 1$. According to Thm. 12, we have $rt(x,n) = ih^n \cdot ib(x) + \sum_{i=0}^{n-1} ih^{n-1-i} \cdot is(x,i) = 1^n \cdot 0 + \sum_{i=0}^{n-1} 1^{n-1-i} \cdot 1 = n$. Similarly, since $ih = 1$ and both $ib(x) = 0$ and $is(x,n) = 1$ are polynomials of degree 0, Thm. 14 implies $rt_{\mathbb{N}}(n) \in \Omega(n^{\max\{0,0+1\}}) = \Omega(n)$.

8 Experiments and Conclusion

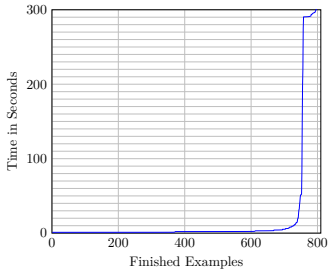
We presented the first approach to infer *lower* bounds for the innermost runtime complexity of TRSs automatically. It is based on speculating rewrite lemmas by narrowing, proving them by induction, and determining the length of the corresponding rewrite sequences from this proof. By taking the size of the start term of the rewrite lemma into account, this yields a lower bound for $\mathrm{irc}_{\mathcal{R}}$. Our approach can be improved by argument filtering and by allowing rewrite lemmas with unknown right-hand sides. In this way the rewrite lemmas do not have to represent rewrite sequences of the original TRS precisely. Future work will be concerned

with considering more general forms of induction proofs and rewrite lemmas.

We implemented our approach in AProVE [11], which uses Z3 [6] to solve arithmetic constraints. While our technique can also infer concrete bounds, currently AProVE only computes asymptotic bounds and provides the lemma that leads to the reported runtime as a witness.

There exist a few results on lower bounds for *derivational complexity* (e.g., [16, 20]) and in the *Termination Competitions*[8] 2009 - 2011, Matchbox [19] proved lower bounds for full derivational complexity where arbitrary rewrite sequences are considered.[9] However, there are no other tools that infer lower bounds for innermost runtime complexity. Hence, we compared our results with the asymptotic *upper* bounds computed by AProVE and TcT [4], the winners in the category *"Runtime Complexity – Innermost Rewriting"* at the *Termination Competition* 2014. We tested with 808 TRSs from this category of the *Termination Problem Data Base* (TPDB 9.0.2) which was used for the Termination Competition 2014. We omitted 118 non-standard TRSs with extra variables on right-hand sides or relative rules. We also disregarded 51 TRSs where AProVE or TcT proved $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(1)$ and 87 examples with $\mathrm{irc}_{\mathcal{R}}(n) \in \Omega(\omega)$ (gray cells in the table below). To identify the latter, we adapted existing innermost non-termination techniques to only consider sequences starting with basic terms. Each tool had a time limit of 300 s for each example. The following table compares the lower bound found by our implementation with the minimum upper bound computed by AProVE or TcT.

$\mathrm{irc}_{\mathcal{R}}(n)$	$\Omega(1)$	$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^3)$	$\Omega(n^{>3})$	$\Omega(2^n)$	$\Omega(3^n)$	$\Omega(\omega)$
$\mathcal{O}(1)$	(51)	–	–	–	–	–	–	–
$\mathcal{O}(n)$	65	201	–	–	–	–	–	–
$\mathcal{O}(n^2)$	5	57	17	–	–	–	–	–
$\mathcal{O}(n^3)$	–	10	3	8	–	–	–	–
$\mathcal{O}(n^{>3})$	3	3	1	–	–	–	–	–
$\mathcal{O}(2^n)$	–	–	–	–	–	–	–	–
$\mathcal{O}(3^n)$	–	–	–	–	–	–	–	–
$\mathcal{O}(\omega)$	78	293	47	6	–	10	1	(87)



The average runtime of our implementation was 22.5 s, but according to the chart above, it was usually much faster. In 694 cases, the analysis finished in 5 seconds. AProVE inferred lower bounds for 657 (81%) of the 808 TRSs. Upper bounds were only obtained for 373 (46%) TRSs, although such bounds exist for at least all 670 TRSs where AProVE shows innermost termination. So although this is the first technique for lower $\mathrm{irc}_{\mathcal{R}}$-bounds, its applicability exceeds the applicability of the techniques for upper bounds which were developed for years. Tight bounds (where the lower and upper bounds are equal) were proven for the 226 TRSs on the diagonal of the table. There are just 74 TRSs where different non-trivial lower and upper bounds were inferred and for 60 of these cases, they just differ by the factor n.

Our approach is particularly powerful for TRSs that implement realistic algorithms, e.g., it shows $\mathrm{irc}_{\mathcal{R}}(n) \in \Omega(n^2)$ for many implementations of classical sorting algorithms like *quicksort*, *maxsort*, *minsort*, and *selection-sort* from the TPDB where neither AProVE nor TcT prove $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$. Detailed experimental results and a web interface for our implementation are available at [1].

Acknowledgments We thank Fabian Emmes for important initial ideas for this paper.

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

[9] For derivational complexity, every non-empty TRS has a trivial linear lower bound. In contrast, proving linear lower bounds for runtime complexity is not trivial. Thus, lower bounds for derivational complexity are in general unsound for runtime complexity. Therefore, an experimental comparison with tools for derivational complexity is not meaningful.

References

1 AProVE: `http://aprove.informatik.rwth-aachen.de/eval/lowerbounds/`.
2 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
3 M. Avanzini and G. Moser. A combination framework for complexity. In *Proc. RTA '13*, LIPIcs 21, pages 55–70, 2013.
4 M. Avanzini and G. Moser. Tyrolean Complexity Tool: Features and usage. In *Proc. RTA '13*, LIPIcs 21, pages 71–80, 2013.
5 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
6 L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS '08*, LNCS 4963, pages 337–340, 2008.
7 F. Emmes, T. Enger, and J. Giesl. Proving non-looping non-termination automatically. In *Proc. IJCAR '12*, LNAI 7364, pages 225–240, 2012.
8 F. Frohn, J. Giesl, F. Emmes, T. Ströder, C. Aschermann, and J. Hensel. Inferring lower bounds for runtime complexity. Technical Report AIB 2015-15, RWTH Aachen, 2015. Available from [1] and from `http://aib.informatik.rwth-aachen.de`.
9 C. Fuhs, J. Giesl, M. Parting, P. Schneider-Kamp, S. Swiderski. Proving termination by dependency pairs and inductive theorem proving. *Journal of Automated Reasoning*, 47(2):133–160, 2011.
10 J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2), 2011.
11 J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Proving termination of programs automatically with AProVE. In *Proc. IJCAR '14*, LNAI 8562, pages 184–191, 2014.
12 N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR '08*, LNAI 5195, pages 364–379, 2008.
13 D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. RTA '89*, LNCS 355, pages 167–177, 1989.
14 M. Hofmann and G. Moser. Amortised resource analysis and typed polynomial interpretations. In *Proc. RTA-TLCA '14*, LNCS 8560, pages 272–286, 2014.
15 D. Knuth. Johann Faulhaber and sums of powers. *Mathematics of Computation*, 61(203):277–294, 1993.
16 G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3), 2011.
17 L. Noschinski, F. Emmes, and J. Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *Journal of Automated Reasoning*, 51(1):27–56, 2013.
18 C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. RTA '10*, LIPIcs 6, pages 259–276, 2010.
19 J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. RTA '04*, LNCS 3091, pages 85–94, 2004.
20 J. Waldmann. Automatic termination. In *Proc. RTA '09*, LNCS 5595, pages 1–16, 2009.
21 H. Zankl and M. Korp. Modular complexity analysis for term rewriting. *Logical Methods in Computer Science*, 10(1), 2014.
22 H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, 1994.