RWTH Aachen

Department of Computer Science
Technical Report

Analyzing Runtime Complexity via Innermost Runtime Complexity

Florian Frohn, Jürgen Giesl

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2017-2

RWTH Aachen · Department of Computer Science · March 2017

Analyzing Runtime Complexity via Innermost Runtime Complexity*

Florian Frohn and Jürgen Giesl

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

Abstract
There exist powerful techniques to infer upper bounds on the *innermost* runtime complexity of term rewrite systems (TRSs), i.e., on the lengths of rewrite sequences that follow an innermost evaluation strategy. However, the techniques to analyze the (full) runtime complexity of TRSs are substantially weaker. In this paper, we present a sufficient criterion to ensure that the runtime complexity of a TRS coincides with its innermost runtime complexity. This criterion can easily be checked automatically and it allows us to use all techniques and tools for innermost runtime complexity in order to analyze (full) runtime complexity. By extensive experiments with an implementation of our results in the tool AProVE, we show that this improves the state of the art of automated complexity analysis significantly.

1 Introduction

Runtime complexity (rc) and *innermost runtime complexity* (irc) are well-established notions for term rewrite systems (TRSs) which measure the worst-case lengths of rewrite sequences that start with *basic terms*. While rc considers arbitrary rewrite sequences, irc requires an *innermost* (eager) evaluation strategy. So the innermost runtime complexity of a TRS $\mathcal{R}$ is always less than or equal to $\mathcal{R}$'s runtime complexity. A *basic term* is of the form $f(t_1, \ldots, t_k)$, where f is a *defined function symbol* (i.e., f corresponds to an algorithm that can be evaluated) and $t_1, \ldots, t_k$ are *constructor terms* (i.e., they represent data). Hence, (innermost) runtime complexity corresponds to the intuitive notion of complexity for programs. More precisely, *innermost* runtime complexity corresponds to the complexity of call-by-value functional programs, whereas rc considers evaluations under *any* strategy. Moreover, by a suitable transformation, rc can also be used to over-approximate the complexity of conditional TRSs, cf. [18].

While complexity analysis of term rewriting is well studied (e.g., [3, 4, 5, 6, 7, 9, 10, 12, 14, 15, 16, 17, 18, 19, 20, 21, 25]), the results of the annual *Termination Competition* [23] show that automatic techniques to infer upper bounds for rc are still substantially weaker than corresponding techniques for irc. 899 examples were analyzed for both rc and irc at the Termination

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

Competitions 2015 and 2016.[1] For 235 of them, a super-polynomial lower bound on rc was inferred. Hence, no upper bounds can be obtained for these examples since the participating tools only compute polynomial upper bounds. For the remaining 664 examples, a polynomial upper bound on irc was proven for 357 TRSs (53.8%) by at least one tool at one of the competitions. In contrast, a polynomial upper bound on rc was inferred for just 218 examples (32.8%).[2]

These numbers indicate that current techniques for complexity analysis of TRSs are much better in analyzing irc than rc, or that irc is significantly easier to handle than rc. In both cases, it is worthwhile to identify (decidable) classes of TRSs where full and innermost runtime complexity coincide. In this paper, we provide a criterion for rc = irc which is easy to automate. It builds upon an important result from [24] that a relaxation of innermost rewriting called *non-dup generalized innermost rewriting* ("*ndg* rewriting") does not yield longer evaluation sequences than innermost rewriting itself. Our main contribution is a criterion to automatically identify classes of TRSs where *all* rewrite sequences starting with basic terms are ndg, which then implies rc = irc. For these classes of TRSs, our results allow us to apply all existing (and all *future*) techniques and results specific to irc (e.g., [3, 4, 5, 21]) to analyze rc directly.

After introducing the needed preliminaries and comparing with related work in Sect. 2, we recall "ndg rewriting" in Sect. 3 and show that it is undecidable whether all rewrite sequences of a TRS are ndg. Hence, we develop a sufficient criterion for this property in Sect. 4 which is easy to check automatically. In Sect. 5 we extend our approach in order to handle TRSs with overlapping (non-overlay) rules. We implemented our contributions in the tool AProVE [12], resulting in a significant improvement of the state of the art in the automated analysis of rc (cf. Sect. 6). App. A contains those proofs that were omitted from the main part of the paper.

2 Preliminaries and Related Work

See, e.g., [8] for the basics of rewriting. $\mathcal{T}(\Sigma,\mathcal{V})$ is the set of all terms over the signature Σ and the variables $\mathcal{V}$. We write $\mathcal{T}$ instead of $\mathcal{T}(\Sigma,\mathcal{V})$ if Σ and $\mathcal{V}$ are irrelevant or clear from the context. For $t \in \mathcal{T}$, $\mathcal{V}(t)$ is the set of all variables in t. The outermost function symbol of a term $t \in \mathcal{T} \setminus \mathcal{V}$ is $\mathrm{root}(t)$. A *TRS* is a finite set of rules $\ell \to r$ where $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$. Given a TRS $\mathcal{R}$ over Σ, Σ_d is the set of its *defined symbols* $\{\mathrm{root}(\ell) \mid \ell \to r \in \mathcal{R}\}$. In contrast, $\Sigma_c = \Sigma \setminus \Sigma_d$ are the *constructors* of $\mathcal{R}$. $\mathcal{T}_B(\Sigma,\mathcal{V})$ resp. $\mathcal{T}_B$ is the set of all *basic* terms over Σ and $\mathcal{V}$. A term $f(t_1,\dots,t_k)$ is basic if $f \in \Sigma_d$ and $t_1,\dots,t_k \in \mathcal{T}(\Sigma_c,\mathcal{V})$. $\mathcal{R}$ is called a *constructor system* if ℓ is basic for each $\ell \to r \in \mathcal{R}$. For $x \in \mathcal{V}$ and $t \in \mathcal{T}$, $\#_x(t)$ denotes the number of occurrences of x in t. A rule $\ell \to r$ is *duplicating* if $\#_x(\ell) < \#_x(r)$ holds for some $x \in \mathcal{V}$, and a TRS is *duplicating* if it contains at least one duplicating rule. A term t is *linear* if $\#_x(t) = 1$ for all $x \in \mathcal{V}(t)$. $\mathcal{R}$ is *left-linear* if ℓ is linear for each $\ell \to r \in \mathcal{R}$.

Example 1. *Consider the TRS* $\mathcal{R}_{\mathsf{times}}$ *where* $\mathsf{s}(\mathsf{0})$ *represents 1,* $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ *represents 2, etc.*

$$\begin{array}{rcll} \mathsf{plus}(\mathsf{0},y) & \to & y & (1) \\ \mathsf{plus}(\mathsf{s}(x),y) & \to & \mathsf{s}(\mathsf{plus}(x,y)) & (2) \end{array} \qquad \begin{array}{rcll} \mathsf{times}(x,\mathsf{0}) & \to & \mathsf{0} & (3) \\ \mathsf{times}(x,\mathsf{s}(y)) & \to & \mathsf{plus}(\mathsf{times}(x,y),x) & (4) \end{array}$$

We have $\Sigma_d = \{\mathsf{plus},\mathsf{times}\}$ *and* $\Sigma_c = \{\mathsf{0},\mathsf{s}\}$. *Rule (4) is duplicating as* $\#_x(\mathsf{times}(x,\mathsf{s}(y))) = 1$

[1] We consider examples as equal if they have the same name. Note that the results of the Termination Competitions 2015 and 2016 are orthogonal. On the one hand, the participating tools improved from 2015 to 2016, but on the other hand, the timeout per example was reduced from 300 s in 2015 to just 30 s in 2016. Hence, in the numbers above, we consider the best results of both competitions to represent the state of the art.

[2] Here, we ignore upper bounds on rc proven by our tool AProVE [12] in 2016. The reason is that at the Termination Competition 2016, AProVE used a preliminary version of the new techniques presented in the current paper and we want to compare with the state of the art *before* the introduction of these techniques. Before 2016, AProVE was not able to infer any upper bounds on rc.

and $\#_x(\mathsf{plus}(\mathsf{times}(x,y),x)) = 2$*. As the left-hand sides of the rules are basic,* $\mathcal{R}_{\mathsf{times}}$ *is a constructor system. Since no variable occurs twice in any left-hand side,* $\mathcal{R}_{\mathsf{times}}$ *is also left-linear.*

Positions are finite sequences of natural numbers, i.e., $\mathrm{pos} = \mathbb{N}^*$. For $\pi, \tau \in \mathrm{pos}$, π is *below* τ ($\pi \geq \tau$) if τ is a (not necessarily proper) prefix of π, i.e., $\pi = \tau.\tau'$ for some $\tau' \in \mathrm{pos}$. We write $\pi \| \tau$ (π and τ are *parallel*) if neither $\pi \geq \tau$ nor $\tau \geq \pi$. The empty position is denoted by ε. The set of all positions of $t \in \mathcal{T}$ is $\mathrm{pos}(t)$. We write $t|_\pi$ to refer to t's subterm at position π where $t|_\varepsilon = t$ and $f(t_1, \dots, t_k)|_{i.\pi} = t_i|_\pi$, and $t[s]_\pi$ is the term that results from replacing $t|_\pi$ with s in t. We say that s is a *subterm* of t if $t|_\pi = s$ for some position π. We write $t \trianglerighteq_\pi s$ in this case, where we omit π if the position is irrelevant. If $\pi \neq \varepsilon$, then s is a *proper* subterm of t.

$\mathcal{R}$ is an *overlay system* if whenever there is a position π such that ℓ and (a variable-renamed version of) $\ell'|_\pi$ with $\ell'|_\pi \notin \mathcal{V}$ unify for two rules $\ell \to r, \ell' \to r' \in \mathcal{R}$, then $\pi = \varepsilon$. Obviously, every constructor system is also an overlay system.

A *context* C is a term with a unique position $\pi \neq \varepsilon$ where $C|_\pi = \Box$. Here, $\Box$ is a special constant and we assume $\Box \notin \Sigma$ for all signatures Σ. We write $C[t]$ as an abbreviation for $C[t]_\pi$. A context $f(t_1, \dots, t_k)$ is *basic* if $f \in \Sigma_d$ and $t_1, \dots, t_k \in \mathcal{T}(\Sigma_c \cup \{\Box\}, \mathcal{V})$.

A *substitution* $\sigma : \mathcal{V} \to \mathcal{T}$ is a function where $\sigma(x) \neq x$ holds for just finitely many $x \in \mathcal{V}$. Hence, we can represent substitutions as finite sets $\sigma = \{x_1/t_1, \dots, x_k/t_k\}$, meaning that $\sigma(x_i) = t_i$ for $1 \leq i \leq k$ and $\sigma(x) = x$ for $x \in \mathcal{V} \setminus \{x_1, \dots, x_k\}$. We lift substitutions to terms as usual and write $t\sigma$ instead of $\sigma(t)$. A *variable renaming* is an injective substitution $\sigma : \mathcal{V} \to \mathcal{V}$.

We write $s \to_{\ell \to r, \pi} t$ if s can be *reduced* (or *evaluated*) to t by applying the rule $\ell \to r$ at position π (i.e., if $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$ for some substitution σ), and $s \to_{\mathcal{R},\pi} t$ if $s \to_{\ell \to r,\pi} t$ holds for some $\ell \to r \in \mathcal{R}$. We omit the subscripts $\ell \to r$, $\mathcal{R}$, or π if they are irrelevant. For any $m \in \mathbb{N}$, $s \to^m t$ means that there exist $t_0, \dots, t_m$ with $s = t_0 \to t_1 \to \dots \to t_m = t$. A term t is a *redex* if there is an $\ell \to r \in \mathcal{R}$ such that ℓ matches t. A redex is called *innermost* if none of its proper subterms is a redex. We write $s \xrightarrow{\mathsf{i}}_\pi t$ for innermost rewrite steps, i.e., if $s|_\pi$ is an innermost redex. A term is a *normal form* if none of its subterms is a redex.

For a binary relation $\to$ on terms, we define the *derivation height* of a term t to be the length of the longest $\to$-sequence starting with t. Here, for any $M \subseteq \mathbb{N} \cup \{\omega\}$, $\sup M$ is the least upper bound of M, where $\sup \varnothing = 0$.

Definition 2 (Derivation Height [16, 21])**.** *We define the* derivation height $\mathrm{dh} : \mathcal{T} \times 2^{\mathcal{T} \times \mathcal{T}} \to \mathbb{N} \cup \{\omega\}$ *as* $\mathrm{dh}(t, \to) = \sup\{m \mid \exists t' \in \mathcal{T}.\ t \to^m t'\}$.

The *innermost runtime complexity* of a TRS maps any $n \in \mathbb{N}$ to the length of the longest $\xrightarrow{\mathsf{i}}$-sequence starting with a basic term t with $|t| \leq n$. It corresponds to the usual notion of "worst-case complexity" for programs with an eager evaluation strategy. In contrast, the *runtime complexity* of a TRS does not impose any restrictions on the evaluation strategy. Here, the *size* of a term is $|x| = 1$ for $x \in \mathcal{V}$ and $|f(t_1, \dots, t_k)| = 1 + |t_1| + \dots + |t_k|$.

Definition 3 ((Innermost) Runtime Complexity $\mathrm{rc}_\mathcal{R}$, $\mathrm{irc}_\mathcal{R}$ [14, 21])**.** *For a TRS* $\mathcal{R}$*, the* runtime complexity $\mathrm{rc}_\mathcal{R} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ *and* innermost runtime complexity $\mathrm{irc}_\mathcal{R} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ *are defined as* $\mathrm{rc}_\mathcal{R}(n) = \sup\{\mathrm{dh}(t, \to_\mathcal{R}) \mid t \in \mathcal{T}_B, |t| \leq n\}$ *and* $\mathrm{irc}_\mathcal{R}(n) = \sup\{\mathrm{dh}(t, \xrightarrow{\mathsf{i}}_\mathcal{R}) \mid t \in \mathcal{T}_B, |t| \leq n\}$.

Clearly, $\mathrm{rc}_\mathcal{R}(n) \geq \mathrm{irc}_\mathcal{R}(n)$ holds for any $n \in \mathbb{N}$. So an upper bound for $\mathrm{rc}_\mathcal{R}$ is also an upper bound for $\mathrm{irc}_\mathcal{R}$ and a lower bound for $\mathrm{irc}_\mathcal{R}$ is also a lower bound for $\mathrm{rc}_\mathcal{R}$. In this paper we investigate for which classes of TRSs $\mathcal{R}$ we have $\mathrm{rc}_\mathcal{R} = \mathrm{irc}_\mathcal{R}$. This allows us to use techniques that infer upper bounds for $\mathrm{irc}_\mathcal{R}$ in order to obtain upper bounds for $\mathrm{rc}_\mathcal{R}$. Similarly, it allows us to apply techniques for the generation of lower bounds for $\mathrm{rc}_\mathcal{R}$ in order to get lower bounds for $\mathrm{irc}_\mathcal{R}$. For the TRS of Ex. 1, our technique will indeed determine $\mathrm{rc}_{\mathcal{R}_{\mathsf{times}}} = \mathrm{irc}_{\mathcal{R}_{\mathsf{times}}}$. Thus, it suffices to prove $\mathrm{irc}_{\mathcal{R}_{\mathsf{times}}}(n) \in \mathcal{O}(n^3)$ in order to infer $\mathrm{rc}_{\mathcal{R}_{\mathsf{times}}}(n) \in \mathcal{O}(n^3)$.

To our knowledge, the most closely related work to ours is [6, 13, 15, 22, 24]. In [13], sufficient criteria are presented such that full and innermost termination coincide. The least restrictive criterion in [13] requires the TRS to be a locally confluent overlay system. Our technique is also particularly well suited for overlay systems, but we also discuss non-overlay systems in Sect. 5. Moreover, instead of local confluence we require that one may only use instantiations which keep certain subterms of the rules in normal form. So compared to [13], both the properties of interest (termination vs. complexity) as well as the identified sufficient criteria are very different. Ex. 4 shows that the conditions required by [13] are not sufficient to ensure rc = irc.

Example 4. *Consider the following TRS $\mathcal{R}$:*

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathsf{f}(\mathsf{0}, y) & \to & y & \mathsf{g}(x) & \to & \mathsf{f}(x, \mathsf{a}) \\ \mathsf{f}(\mathsf{s}(x), y) & \to & \mathsf{f}(x, \mathsf{node}(y, y)) & \mathsf{a} & \to & \mathsf{b} \end{array}$$

$\mathcal{R}$ is non-overlapping and thus a locally confluent overlay system. Hence, termination and innermost termination of $\mathcal{R}$ coincide by [13]. Any basic term of size n only leads to innermost rewrite sequences of length $\mathcal{O}(n)$. In contrast, arbitrary rewrite sequences can have exponential length. For example, the basic term $\mathsf{g}(\mathsf{s}^n(\mathsf{0}))$ of size $n+2$ is first reduced to $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{a})$. Instead of evaluating the subterm a, one could now apply the second f-rule repeatedly and obtain a complete binary tree of height n whose (exponentially many) leaves are a's. Finally, these leaves can all be reduced to b in 2^n rewrite steps. Thus, we have $\mathrm{irc}(n) \in \Theta(n)$ and $\mathrm{rc}(n) \in \Theta(2^n)$.

In [22], the authors identify criteria which ensure that all normal forms of a term w.r.t. full rewriting are also reachable via innermost rewriting. This turns out to be the case for right-linear terminating overlay systems. As mentioned before, our technique is also particularly well suited for overlay systems, but we neither require termination nor right-linearity. In fact, non-right-linear rules are common in many TRSs like $\mathcal{R}_{\mathsf{times}}$ from Ex. 1 which implement natural algorithms. The following example illustrates that the property that every normal form is reachable via innermost rewriting is not sufficient for rc = irc.

Example 5. *Consider the TRS with the rules $\mathsf{c} \to \mathsf{f}(\mathsf{a})$, $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a})$, and $\mathsf{a} \to \mathsf{b}$. Clearly, all normal forms are reachable via innermost rewriting as the only possible non-innermost rewrite steps have the form $\mathsf{f}^n(\mathsf{a}) \to \mathsf{f}^n(\mathsf{a})$. However, we have $\mathrm{irc}(n) \in \Theta(1)$ but $\mathrm{rc}(n) \in \Theta(\omega)$ due to the non-terminating rewrite sequence $\mathsf{c} \to \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \ldots$ that starts with the basic term c.*

However, rc = irc indeed holds for right-linear overlay systems, which is a special case of the criterion introduced in this paper.

In [15], it is shown that for non-duplicating overlay systems, whenever a term t has a reduction to a normal form then t also starts an innermost reduction of the same length. Thus, this implies rc = irc for non-duplicating terminating overlay systems, which can be used to improve automated complexity analysis [6]. In contrast, our approach does not require termination and it allows us to infer rc = irc for many TRSs that are duplicating or no overlay systems.

In [24], non-dup-generalized innermost rewriting is introduced as an extension of innermost rewriting. More precisely, ndg rewriting allows non-innermost rewrite steps as long as all proper subterms of left-hand sides with defined root symbol and all variables that occur more than once on right-hand sides of rules are instantiated to normal forms. Our work is based on [24] which states that ndg rewriting is not more costly than innermost rewriting. The use case in [24] is to implement rewriting more efficiently by allowing certain non-innermost steps while guaranteeing that the applied evaluation strategy is not worse than innermost rewriting. In contrast, our goal is automated complexity analysis. To this end, we introduce a novel technique to prove that *all* rewrite sequences starting with basic terms are ndg for a given TRS. For such TRSs, the runtime complexity for full and innermost rewriting coincides.

3 Non-Dup-Generalized Innermost Rewriting

In this section, we recall the definition of *ndg rewriting* from [24]. As mentioned, "ndg" requires that variables occurring multiple times in right-hand sides of rules may only be instantiated by normal forms (we call such rewrite steps *spare*). So the main difference to full rewriting is that ndg rewriting does not allow rewrite steps that duplicate redexes. Moreover, proper subterms of left-hand sides with defined root may only be instantiated to normal forms. In Sect. 4, we show how to automatically prove that *every* rewrite sequence starting with a basic term is ndg.

Definition 6 (Spare and ndg Rewriting [24])**.** *Let $s \to_{\ell\to r,\pi} t$ and let σ be the matcher with $\ell\sigma = s|_\pi$. The rewrite step $s \to_{\ell\to r,\pi} t$ is* spare *if $x\sigma$ is a normal form for all variables x with $\#_x(r) > 1$. It is* non-dup-generalized innermost (ndg)*, denoted $s \hookrightarrow_{\ell\to r,\pi} t$, if it is spare and $\ell|_\tau\sigma$ is a normal form for all $\tau \in \mathrm{pos}(\ell) \setminus \{\varepsilon\}$ with $\mathrm{root}(\ell|_\tau) \in \Sigma_d$. A TRS $\mathcal{R}$ is spare resp. ndg if every $\to_\mathcal{R}$-sequence starting with a basic term only consists of spare resp. ndg rewrite steps.*

Example 7. *For $\mathcal{R}_{\mathsf{times}}$, the rewrite step $\mathsf{times}(x, \mathsf{s}(\mathsf{plus}(\mathsf{0}, z))) \hookrightarrow \mathsf{plus}(\mathsf{times}(x, \mathsf{plus}(\mathsf{0}, z)), x)$ is ndg, but it is not an innermost step due to the redex $\mathsf{plus}(\mathsf{0}, z)$. In contrast, $\mathsf{times}(\mathsf{plus}(\mathsf{0}, z), \mathsf{s}(y))$ $\to \mathsf{plus}(\mathsf{times}(\mathsf{plus}(\mathsf{0}, z), y), \mathsf{plus}(\mathsf{0}, x))$ is not ndg or spare, as the redex $\mathsf{plus}(\mathsf{0}, z)$ is duplicated.*

Cor. 8 states two straightforward observations: innermost rewrite steps are ndg, since an innermost redex has no redexes as proper subterms. Moreover, spareness and ndg are the same for overlay systems, where no proper non-variable subterm of a left-hand side unifies with a redex.

Corollary 8 (Innermost Rewriting, Spareness, and ndg)**.**

(a) *Every innermost rewrite step is an ndg rewrite step, i.e., $\xrightarrow{\mathrm{i}} \subseteq \hookrightarrow$.*

(b) *Every spare overlay system is ndg.*

The following examples show that, in general, rc and irc do not coincide if $\mathcal{R}$ is not ndg.

Example 9. *The TRS from Ex. 5 is spare, but not ndg due to the rewrite sequence $\mathsf{c} \to \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \dots$ where the subterm a below the root of the left-hand side is not in normal form. As mentioned in Ex. 5, we have $\mathrm{irc}(n) \in \Theta(1)$ but $\mathrm{rc}(n) \in \Theta(\omega)$.*

The TRS in Ex. 4 is not spare, because the sequence $\mathsf{g}(\mathsf{s}^n(\mathsf{0})) \to \mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{a}) \to \mathsf{f}(\mathsf{s}^{n-1}(\mathsf{0}), \mathsf{node}(\mathsf{a}, \mathsf{a})) \to \dots$ duplicates redexes. Here, we have $\mathrm{irc}(n) \in \Theta(n)$ but $\mathrm{rc}(n) \in \Theta(2^n)$.

The next TRS is a non-left-linear, but non-duplicating overlay system. It shows why for spareness it is not enough if $x\sigma$ is a normal form whenever $\#_x(\ell) < \#_x(r)$ (i.e., if x is duplicated):

$$\mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \to \mathsf{f}(\mathsf{h}, \mathsf{h}) \qquad \mathsf{f}(x, x) \to \mathsf{g}(x, x) \qquad \mathsf{h} \to \mathsf{0} \qquad \mathsf{h} \to \mathsf{s}(\mathsf{0})$$

We have $\mathrm{rc}(n) \in \Theta(\omega)$ due to the non-terminating rewrite sequence $\mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \to \mathsf{f}(\mathsf{h}, \mathsf{h}) \to \mathsf{g}(\mathsf{h}, \mathsf{h}) \to^2 \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \to \dots$ However, $\mathrm{irc}(n) \in \Theta(1)$ holds, as we have, e.g., $\mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \xrightarrow{\mathrm{i}} \mathsf{f}(\mathsf{h}, \mathsf{h}) \xrightarrow{\mathrm{i}}{}^2 \mathsf{f}(\mathsf{0}, \mathsf{0}) \xrightarrow{\mathrm{i}} \mathsf{g}(\mathsf{0}, \mathsf{0})$. All other $\xrightarrow{\mathrm{i}}$-sequences that start with basic terms have at most length 4, too. Note that if in Def. 6 spareness only required $x\sigma$ to be a normal form for variables x that are duplicated, then this TRS would trivially be spare although $\mathrm{rc} \neq \mathrm{irc}$. But with our definition of spareness in Def. 6 the TRS is not spare, since the variable x which occurs twice in the right-hand side $\mathsf{g}(x, x)$ is instantiated by the redex h in the above reduction.

Our technique relies on the following important result of [24], which states that for any ndg rewrite sequence starting in a term s, s starts an innermost rewrite sequence of the same length.

Theorem 10 (Length of ndg Rewriting [24, Lemma 8])**.** *If $s \hookrightarrow^n t$ then $s \xrightarrow{\mathrm{i}}{}^n u$ for some $u \in \mathcal{T}$.*

Cor. 11 follows from Thm. 10, because if $\mathcal{R}$ is ndg, then $s \to_{\mathcal{R}}^n t$ implies $s \hookrightarrow_{\mathcal{R}}^n t$ for basic terms s.

Corollary 11 (rc = irc)**.** *Let $\mathcal{R}$ be a TRS which is ndg. Then* $\mathrm{rc}_{\mathcal{R}} = \mathrm{irc}_{\mathcal{R}}$.

According to Cor. 11, innermost and full runtime complexity coincide for TRSs that are ndg. Unfortunately, the question whether a TRS is spare resp. ndg is undecidable.

Theorem 12 (Spareness is Undecidable)**.** *It is undecidable whether a TRS is spare (resp. ndg).*

Proof. The proof relies on an encoding of Turing machines to left-linear basic TRSs where each configuration of the Turing machine is represented by a ground term (i.e., it relies on the Turing completeness of left-linear basic TRSs). We call a TRS basic if $\ell, r \in \mathcal{T}_B$ for all $\ell \to r \in \mathcal{R}$.

Let $\mathcal{M} = (Q, \Gamma, \delta)$ be a Turing machine where Q is the set of states, Γ is the tape alphabet with $Q \cap \Gamma = \varnothing$, and $\Box \in \Gamma$ is the blank symbol. A configuration of the Turing machine has the form (q, w, a, w') with $q \in Q$, $w, w' \in \Gamma^\omega \setminus \Gamma^*$, and $a \in \Gamma$, where w and w' both consist of a finite word followed by infinitely many occurrences of $\Box$. The configuration means that q is the current state, the symbol at the current position of the tape is a, the symbols right of the current position are described by the infinite word w', and the symbols left of it are described by the infinite word w. To ease the formulation, if $w = b\,.\,\overline{w}$ then this means that b is the symbol directly left of the current position, i.e., w is the word obtained when reading the symbols on the tape from right to left. The transition function $\delta : (Q \times \Gamma) \to (Q \times \Gamma \times \{L, R\})$ induces a transition relation $\to_{\mathcal{M}}$ on configurations where $(q_1, w_1, a_1, w_1') \to_{\mathcal{M}} (q_2, w_2, a_2, w_2')$ if either

- $w_1 = a_2\,.\,w_2, w_2' = b\,.\,w_1'$, and $\delta(q_1, a_1) = (q_2, b, L)$ or
- $w_2 = b\,.\,w_1, w_1' = a_2\,.\,w_2'$, and $\delta(q_1, a_1) = (q_2, b, R)$.

For any Turing machine $\mathcal{M} = (Q, \Gamma, \delta)$, we define the TRS $\mathcal{R}_{\mathcal{M}}$ by adapting our previous related encoding from [10]. In $\mathcal{R}_{\mathcal{M}}$ there is a function symbol cf of arity 4, all symbols from Γ become function symbols of arity 1, and $Q \cup \{\underline{a} \mid a \in \Gamma\}$ are constants.

$$
\begin{array}{rcll}
\mathcal{R}_{\mathcal{M}} & = & \{\mathsf{cf}(q_1, a_2(xs), \underline{a_1}, ys) \to \mathsf{cf}(q_2, xs, \underline{a_2}, b(ys)) \mid a_2 \in \Gamma, \delta(q_1, a_1) = (q_2, b, L)\} & \cup \\
& & \{\mathsf{cf}(q_1, xs, \underline{a_1}, a_2(ys)) \to \mathsf{cf}(q_2, b(xs), \underline{a_2}, ys) \mid a_2 \in \Gamma, \delta(q_1, a_1) = (q_2, b, R)\} & \cup \\
& & \{\mathsf{cf}(q_1, \underline{\Box}, \underline{a_1}, ys) \to \mathsf{cf}(q_2, \underline{\Box}, \underline{\Box}, b(ys)) \mid \delta(q_1, a_1) = (q_2, b, L)\} & \cup \\
& & \{\mathsf{cf}(q_1, xs, \underline{a_1}, \underline{\Box}) \to \mathsf{cf}(q_2, b(xs), \underline{\Box}, \underline{\Box}) \mid \delta(q_1, a_1) = (q_2, b, R)\} &
\end{array}
$$

Obviously, $\mathcal{R}_{\mathcal{M}}$ is basic and left-linear. A configuration (q, w, a, w') of the Turing machine can now be encoded as the ground term $(q, w, a, w')_{\mathcal{T}} = \mathsf{cf}(q, w_{\mathcal{T}}, \underline{a}, w'_{\mathcal{T}})$ where $v_{\mathcal{T}} = \underline{\Box}$ if $v_{\mathcal{T}} = \Box.\Box \ldots$ (i.e., if $v_{\mathcal{T}}$ is the infinite word consisting only of $\Box$) and otherwise, $v_{\mathcal{T}} = a(v'_{\mathcal{T}})$ where $v = a.v'$. Now we can prove that $\mathcal{R}_{\mathcal{M}}$ indeed simulates the Turing machine $\mathcal{M}$:

$$(q_1, w_1, a_1, w_1') \to_{\mathcal{M}} (q_2, w_2, a_2, w_2') \qquad \text{iff} \qquad (q_1, w_1, a_1, w_1')_{\mathcal{T}} \to_{\mathcal{R}_{\mathcal{M}}} (q_2, w_2, a_2, w_2')_{\mathcal{T}} \quad (1)$$

In the following, we write $f_1 f_2 \ldots f_n c$ to denote terms of the form $f_1(f_2(\ldots f_n(c) \ldots))$ to ease readability. For the "only if" direction of (1), we just regard the case $\delta(q_1, a_1) = (q_2, b, L)$. The case $\delta(q_1, a_1) = (q_2, b, R)$ is analogous. Hence, $w_1 = a_2.w_2$ and $w_2' = b.w_1'$. Let $w_2 = b_1.b_2 \ldots b_n.\Box.\Box \ldots$ and $w_1' = c_1.c_2 \ldots c_m.\Box.\Box \ldots$ Note that w_2 and w_1' have to be of this form, as for every configuration of a Turing machine the tape just contains finitely many non-blank symbols.

First consider $w_1 \neq \Box.\Box \ldots$ Then we have $(q_1, w_1, a_1, w_1')_{\mathcal{T}} = \mathsf{cf}(q_1, a_2 b_1 b_2 \ldots b_n \underline{\Box}, \underline{a_1}, (w_1')_{\mathcal{T}})$. By definition, $\mathsf{cf}(q_1, a_2(xs), \underline{a_1}, ys) \to \mathsf{cf}(q_2, xs, \underline{a_2}, b(ys)) \in \mathcal{R}_{\mathcal{M}}$. Hence, we get $(q_1, w_1, a_1, w_1')_{\mathcal{T}} = \mathsf{cf}(q_1, a_2 b_1 b_2 \ldots b_n \underline{\Box}, \underline{a_1}, (w_1')_{\mathcal{T}}) \to_{\mathcal{R}_{\mathcal{M}}} \mathsf{cf}(q_2, b_1 b_2 \ldots b_n \underline{\Box}, \underline{a_2}, b(w_1')_{\mathcal{T}}) = (q_2, w_2, a_2, w_2')_{\mathcal{T}}$.

Now consider $w_1 = \Box.\Box\ldots$ Thus, $(q_1, w_1, a_1, w_1')_{\mathcal{T}} = \mathsf{cf}(q_1, \underline{\Box}, \underline{a_1}, (w_1')_{\mathcal{T}})$. By definition, $\mathsf{cf}(q_1, \underline{\Box}, \underline{a_1}, ys) \to \mathsf{cf}(q_2, \underline{\Box}, \underline{\Box}, b(ys)) \in \mathcal{R}_{\mathcal{M}}$. Note that $w_1 = a_2.w_2$ implies $a_2 = \Box$ and $w_2 = \Box.\Box\ldots$ Hence, $(q_1, w_1, a_1, w_1')_{\mathcal{T}} = \mathsf{cf}(q_1, \underline{\Box}, \underline{a_1}, (w_1')_{\mathcal{T}}) \to_{\mathcal{R}_{\mathcal{M}}} \mathsf{cf}(q_2, \underline{\Box}, \underline{\Box}, b(w_1')_{\mathcal{T}}) = (q_2, w_2, a_2, w_2')_{\mathcal{T}}$.

For the "if" direction of (1), first consider the case that a rule $\mathsf{cf}(q_1, a_2(xs), \underline{a_1}, ys) \to \mathsf{cf}(q_2, xs, \underline{a_2}, b(ys))$ is applied to rewrite $(q_1, w_1, a_1, w_1')_{\mathcal{T}}$ to $(q_2, w_2, a_2, w_2')_{\mathcal{T}}$. The case that a rule of the form $\mathsf{cf}(q_1, xs, \underline{a_1}, a_2(ys)) \to \mathsf{cf}(q_2, b(xs), \underline{a_2}, ys)$ is applied is analogous. Then we get $w_1 = a_2.w_2$ and $w_2' = b.w_1'$. Moreover, we have $\delta(q_1, a_1) = (q_2, b, L)$ by the definition of $\mathcal{R}_{\mathcal{M}}$. Hence, we get $(q_1, w_1, a_1, w_1') = (q_1, a_2.w_2, a_1, w_1') \to_{\mathcal{M}} (q_2, w_2, a_2, b.w_1') = (q_2, w_2, a_2, w_2')$.

Now consider the case that a rule $\mathsf{cf}(q_1, \underline{\Box}, \underline{a_1}, ys) \to \mathsf{cf}(q_2, \underline{\Box}, \underline{\Box}, b(ys))$ is applied to rewrite $(q_1, w_1, a_1, w_1')_{\mathcal{T}}$ to $(q_2, w_2, a_2, w_2')_{\mathcal{T}}$. The case that a rule of the form $\mathsf{cf}(q_1, xs, \underline{a_1}, \underline{\Box}) \to \mathsf{cf}(q_2, b(xs), \underline{\Box}, \underline{\Box})$ is applied is analogous. Then we get $w_1 = w_2 = \Box.\Box\ldots$, $a_2 = \Box$, and $w_2' = b.w_1'$. Moreover, $\delta(q_1, a_1) = (q_2, b, L)$ by the definition of $\mathcal{R}_{\mathcal{M}}$. Hence, $(q_1, w_1, a_1, w_1') = (q_1, \Box.\Box\ldots, a_1, w_1') \to_{\mathcal{M}} (q_2, \Box.\Box\ldots, \Box, b.w_1') = (q_2, w_2, a_2, w_2')$, which finishes the proof of (1).

By (1), undecidability of termination for Turing machines implies undecidability of normalization of basic ground terms w.r.t. left-linear basic TRSs like $\mathcal{R}_{\mathcal{M}}$, as $(q, w, a, w')_{\mathcal{T}}$ is a basic ground term. The reason is that for any Turing machine $\mathcal{M}$, we have:

	$\mathcal{M}$ is terminating w.r.t. the start configuration (q, w, a, w')	
iff	$\mathcal{R}_{\mathcal{M}}$ is terminating on $(q, w, a, w')_{\mathcal{T}}$	by (1)
iff	$\mathcal{R}_{\mathcal{M}}$ is normalizing on $(q, w, a, w')_{\mathcal{T}}$	see below (†)

For the step (†), note that termination and normalization of $\mathcal{R}_{\mathcal{M}}$ on basic ground terms are equivalent as $\mathcal{R}_{\mathcal{M}}$ is basic and non-overlapping.

Now we can prove that spareness of TRSs is undecidable. To this end, let $\mathcal{R}$ be a left-linear basic TRS over the signature Σ. As $\mathcal{R}$ is basic, every rewrite sequence that starts with a basic term only leads to basic terms. Hence, $\mathcal{R}$ is spare.

Given a basic ground term $f(t_1, \ldots, t_k) \in \mathcal{T}_B(\Sigma, \varnothing)$, we define a constructor system $\mathcal{R}'$ over the signature $\Sigma' = \Sigma \uplus \{\mathsf{a}, \mathsf{c}, \mathsf{g}, \mathsf{h}, \mathsf{inf}\}$ such that normalization of $f(t_1, \ldots, t_k)$ w.r.t. $\mathcal{R}$ can be checked by checking spareness of $\mathcal{R}'$ instead. Since we have shown that normalization of basic ground terms w.r.t. left-linear basic TRSs is undecidable, in this way one can prove that spareness is also undecidable. As a constructor system is spare iff it is ndg by Cor. 8 (b), this also shows that it is undecidable whether a TRS is ndg.

The construction of $\mathcal{R}'$ works as follows: All rules of $\mathcal{R}$ are also included in $\mathcal{R}'$. Moreover, for each defined function symbol $e \in \Sigma_d$, we add rules $e(\ldots) \to \mathsf{a}$ to $\mathcal{R}'$ such that for any $p_1, \ldots, p_m \in \mathcal{T}(\Sigma_c, \varnothing)$, $e(p_1, \ldots, p_m)$ can be reduced to a iff $e(p_1, \ldots, p_m)$ is in $\to_{\mathcal{R}}$-normal form. Note that this is easily possible, as $\mathcal{R}$ is a left-linear constructor system and we only consider basic ground terms $e(p_1, \ldots, p_m)$. So the new rules $e(\ldots) \to \mathsf{a}$ need to cover all constructor ground terms that are not matched by the left-hand sides of the other e-rules of $\mathcal{R}$. Furthermore, we add the rules $\mathsf{g} \to \mathsf{h}(\mathsf{inf}, f(t_1, \ldots, t_k))$, $\mathsf{h}(x, \mathsf{a}) \to \mathsf{c}(x, x)$, and $\mathsf{inf} \to \mathsf{inf}$. By construction, $\mathcal{R}'$ is not spare iff $f(t_1, \ldots, t_k)$ is normalizing w.r.t. $\mathcal{R}$. To see this, recall that spareness of $\mathcal{R}'$ means that all rewrite sequences starting with basic terms are spare. Clearly, only rules from $\mathcal{R}$ are applicable to basic terms whose root is from Σ_d and thus, all these rewrite sequences are spare. Hence, we now consider all basic terms with root g, h, or inf.

- For the basic term g we have $\mathsf{g} \hookrightarrow \mathsf{h}(\mathsf{inf}, f(t_1, \ldots, t_k))$. If $f(t_1, \ldots, t_k)$ is not normalizing, then by construction, we can never evaluate h and hence the resulting rewrite sequence is spare. If $f(t_1, \ldots, t_k)$ is normalizing, then let t be a normal form of $f(t_1, \ldots, t_k)$. Note that as $\mathcal{R}$ is a basic TRS, t is also a basic term. Hence, we get $\mathsf{h}(\mathsf{inf}, f(t_1, \ldots, t_k)) \hookrightarrow^* \mathsf{h}(\mathsf{inf}, t) \hookrightarrow \mathsf{h}(\mathsf{inf}, \mathsf{a}) \to \mathsf{c}(\mathsf{inf}, \mathsf{inf})$. Note that the last step is not spare.

- Each basic term of the form $\mathsf{h}(s, s')$ is either a normal form (if $s' \neq \mathsf{a}$) or just enables the spare rewrite step $\mathsf{h}(s, \mathsf{a}) \hookrightarrow \mathsf{c}(s, s)$.

- If we start with the basic term inf, the only possible reduction is $\mathsf{inf} \hookrightarrow \mathsf{inf} \hookrightarrow \ldots$

Hence, a semi-decision procedure for spareness yields a semi-decision procedure for non-normalization of arbitrary basic ground terms for basic left-linear TRSs. □

On the other hand, the question whether a TRS is *not* spare resp. *not* ndg is semi-decidable. A semi-decision procedure is obtained by enumerating all rewrite sequences starting with basic terms and checking whether these sequences contain non-spare resp. non-ndg steps. In fact, spareness and ndg are even undecidable for left-linear constructor systems (which correspond to first-order functional programs), as the TRS $\mathcal{R}'$ constructed in the proof of Thm. 12 is a left-linear constructor system. However, there are some obvious sufficient syntactic criteria for spareness.

Lemma 13 (Right-Linear TRSs are Spare)**.** *Every right-linear TRS is spare. Hence, every right-linear overlay system is ndg.*

By Lemma 13, ndg is a generalization of the criterion presented in [22], as mentioned in Sect. 2.

Lemma 14 (TRSs Without Nested Defined Symbols in Right-Hand Sides are ndg)**.** *If there is no rule* $\ell \to r \in \mathcal{R}$ *with* $\mathrm{root}(r|_\pi), \mathrm{root}(r|_{\pi.\tau}) \in \Sigma_d$ *where* $\pi, \pi.\tau \in \mathrm{pos}(r)$ *and* $\tau \neq \varepsilon$*, then* $\mathcal{R}$ *is ndg.*

Proof. Let $t_0 \to t_1 \to \ldots$ be a rewrite sequence where t_0 is basic. As there is no rule where defined symbols are nested on the right-hand side, defined symbols are not nested in any t_i, $i \in \mathbb{N}$. Hence, $t_0 \to t_1 \to \ldots$ is an innermost and thus ndg rewrite sequence by Cor. 8 (a). □

We will present a much more powerful sufficient criterion for spareness in Sect. 4 which is still easy to automate. For spareness, this criterion subsumes Lemma 13 and 14. According to Cor. 8 (b), it can be used to prove that overlay systems are ndg. Hence, for checking ndg, the criterion of Sect. 4 subsumes Lemma 13, but not Lemma 14, which is also applicable to non-overlay systems. In Sect. 5, we will introduce a technique to check whether a spare non-overlay system is ndg. The combination of the techniques introduced in Sect. 4 and 5 then also subsumes Lemma 14.

4 Approximating Spareness

According to Cor. 11, innermost and full runtime complexity coincide for ndg TRSs. We presented two simple syntactic sufficient criteria which ensure that a TRS is ndg in Lemma 13 and 14, but these criteria are still far too restrictive. Hence, we now introduce a much more powerful technique which allows us to prove spareness in many cases. So for overlay systems, due to Cor. 8(b) this technique can be used to prove that the system is ndg.

The idea of our technique is to over-approximate all non-innermost redexes which are reachable from basic terms by a finite set of contexts *Def* where the inner redex is below □. Similarly, for all rules $\ell \to r$ with $\#_x(r) > 1$ we over-approximate the redexes $\ell\sigma$ which are reachable from basic terms by a finite set of contexts *Dup* where □ stands for the "duplicated" subterm $x\sigma$. Then, we check if there are contexts in *Def* and *Dup* that "overlap". If this is not the case, then the analyzed TRS is spare.

To formalize the notions of "overlap" and "over-approximation" we introduce an instance relation on contexts. Then, two contexts overlap if they have a common instance and a context D over-approximates all contexts that are instances of D. The intuition behind the instance relation is that D is "more general" than C if C results from D by instantiating variables and replacing $\Box$ by a term containing $\Box$. Then, we can use a context D to represent all terms $C[t]$ where C is an instance of D and t has a certain property (like "may be duplicated" or "may contain redexes").

Definition 15 (Instance $\sqsubseteq$)**.** *Given two contexts $C[\Box]_\pi, D[\Box]_\tau$ we call C an* instance *of D ($C \sqsubseteq D$) iff $\pi \geq \tau$ and $D[x]$ matches C, where x is a fresh variable.*

In other words, $C \sqsubseteq D$ holds iff there is a context C' and a substitution σ with $C = D\sigma[C']$.

Example 16. *The context $\mathsf{plus}(\mathsf{s}(\Box), \mathsf{s}(y))$ is an instance of $\mathsf{plus}(\Box, y)$, as we have $1.1 \geq 1$ (where 1.1 and 1 are the positions of $\Box$ in $\mathsf{plus}(\mathsf{s}(\Box), \mathsf{s}(y))$ and $\mathsf{plus}(\Box, y)$, respectively) and $\mathsf{plus}(\Box, y)[x] = \mathsf{plus}(x, y)$ matches $\mathsf{plus}(\mathsf{s}(\Box), \mathsf{s}(y))$.*

The following corollary states some useful observations on the instance relation.

Corollary 17 (Properties of $\sqsubseteq$)**.**

(a) *The instance relation is transitive, i.e., $C \sqsubseteq D$ and $D \sqsubseteq E$ implies $C \sqsubseteq E$.*

(b) *For any context C and any substitution σ we have $C\sigma \sqsubseteq C$.*

(c) *For any term t with positions $\pi, \tau \in \mathrm{pos}(t)$, $\pi \geq \tau$ implies $t[\Box]_\pi \sqsubseteq t[\Box]_\tau$.*

Two contexts C and D *overlap* if there is a context which is an instance of both C and D. In other words, C and D overlap if there exist terms that are represented both by C and by D.

Definition 18 (Overlapping Contexts)**.** *Given two contexts C, D we say that C and D* overlap *if there is a context E such that $E \sqsubseteq C$ and $E \sqsubseteq D$.*

Example 19. *The contexts $\mathsf{plus}(\Box, \mathsf{s}(y))$ and $\mathsf{plus}(\mathsf{s}(\Box), y)$ overlap, as $\mathsf{plus}(\mathsf{s}(\Box), \mathsf{s}(y))$ is an instance of both of them.*

Note that for any two contexts C and D, it is decidable whether they overlap: One has to check whether the positions of $\Box$ in C and D are not parallel and whether $C[x]$ and a variable-renamed version of $D[y]$ have a most general unifier σ.

A context is *duplicating* if it results from a rule $\ell \to r$ where a variable x occurs more than once in r and if $\ell\sigma$ appears in a rewrite sequence that starts with a basic term. To turn $\ell\sigma$ into a context, one replaces a subterm of $x\sigma$ by $\Box$.

Definition 20 (Duplicating Context)**.** *Given a TRS $\mathcal{R}$, we call a context C* duplicating *if there is a term $s \in \mathcal{T}_B$, a substitution σ, and a rewrite sequence $s \to^* t \trianglerighteq \ell\sigma$ where ℓ is the left-hand side of a rule $\ell \to r \in \mathcal{R}$ such that $\ell|_\pi = x \in \mathcal{V}$, $\#_x(r) > 1$, and $C = \ell\sigma[\Box]_{\pi.\tau}$ for some $\pi.\tau \in \mathrm{pos}(\ell\sigma)$.*

Example 21. *Reconsider the TRS $\mathcal{R}_{\mathsf{times}}$ from Ex. 1. Rule (4) is the only rule where a variable occurs more than once on the right-hand side. Its left-hand side is $\mathsf{times}(x, \mathsf{s}(y))$. If one starts rewriting with a basic term, one can only reach instantiations of the form $\mathsf{times}(t_1, \mathsf{s}(t_2))$ with $t_1, t_2 \in \mathcal{T}(\Sigma_c, \mathcal{V})$. As the variable x of the left-hand side is duplicated, the duplicating contexts of $\mathcal{R}_{\mathsf{times}}$ are $\mathsf{times}(\mathsf{s}^n(\Box), \mathsf{s}(t_2))$ where $t_2 \in \mathcal{T}(\Sigma_c, \mathcal{V})$ and $n \in \mathbb{N}$. So in other words, spareness of $\mathcal{R}_{\mathsf{times}}$ can only be violated if a basic term can be rewritten to a term containing an instance of $\mathsf{times}(\mathsf{s}^n(\Box), \mathsf{s}(t_2))$, where $\Box$ is replaced by a term that is not a normal form.*

As the following theorem shows, it is undecidable whether a context is duplicating.

Theorem 22 (Duplicating Contexts are Undecidable)**.** *It is undecidable if a context is duplicating.*

The proof is similar to the one of Thm. 12, i.e., for any left-linear basic TRS $\mathcal{R}$ and any basic ground term t we define a left-linear constructor system $\mathcal{R}'$ such that t is normalizing w.r.t. $\mathcal{R}$ iff a certain context is duplicating w.r.t. $\mathcal{R}'$. Hence, whether a context is duplicating is even undecidable for left-linear constructor systems. However, the duplicating contexts of a TRS can easily be over-approximated by a finite set of contexts *Dup* such that every duplicating context is an instance of an element of *Dup*. In this approximation, we do not take the requirement into account that a duplicating context must be reachable by a rewrite sequence that starts with a basic term. Moreover, we disregard possible instantiations of left-hand sides and only consider contexts where $\Box$ is at the position of a duplicated variable.

Definition 23 ($Dup_{\mathcal{R}}$)**.** *Given a TRS $\mathcal{R}$, we define $Dup_{\mathcal{R}} = \{C \mid C[x] \to r \in \mathcal{R}, \#_x(r) > 1\}$. We omit the index $\mathcal{R}$ if it is clear from the context.*

Example 24. *We have $Dup_{\mathcal{R}_{\mathsf{times}}} = \{\mathsf{times}(\Box, \mathsf{s}(y))\}$, as $\mathsf{times}(x, \mathsf{s}(y)) \to \mathsf{plus}(\mathsf{times}(x, y), x)$ is $\mathcal{R}_{\mathsf{times}}$'s only rule with a non-linear right-hand side and 1 is the only position of x on the left-hand side. Note that all duplicating contexts $\mathsf{times}(\mathsf{s}^n(\Box), \mathsf{s}(t_2))$ are instances of $\mathsf{times}(\Box, \mathsf{s}(y))$.*

The following lemma states that *Dup* indeed over-approximates all duplicating contexts.

Lemma 25 (*Dup* Over-Approximates Duplicating Contexts)**.** *If C is duplicating, then there is a $D \in Dup$ such that $C \sqsubseteq D$.*

Proof. If C is duplicating, then there is a rule $\ell \to r \in \mathcal{R}$ and a rewrite sequence $s \to^* t \trianglerighteq \ell\sigma = C[p]_{\pi.\tau}$ where $\ell|_\pi = x$ is a variable that occurs more than once on the right-hand side. Then we have $\ell[\Box]_\pi \in Dup$ and $C \sqsubseteq \ell[\Box]_\pi$. To see this, note that $\ell[x]_\pi = \ell$ matches $C[p]_{\pi.\tau} = \ell\sigma$. So if x' is a fresh variable, then $\ell[x']_\pi$ also matches $C[\Box]_{\pi.\tau} = C$. Moreover, we have $\pi.\tau \geq \pi$. $\Box$

Defined contexts characterize those contexts with defined root that can be reached by rewriting basic terms, where a redex may occur at the position of $\Box$.

Definition 26 (Defined Context)**.** *Given a TRS $\mathcal{R}$, we call a context C* defined *if there is a term $s \in \mathcal{T}_B$ and a rewrite sequence $s \to^* t \trianglerighteq C[p]$ where $\mathrm{root}(C) \in \Sigma_d$ and p is a redex.*

Example 27. *Reconsider the TRS $\mathcal{R}_{\mathsf{times}}$. For any $t \in \mathcal{T}(\Sigma_c, \mathcal{V})$, the context $\mathsf{plus}(\Box, t)$ is defined due to the rewrite sequence $\mathsf{times}(t, \mathsf{s}(0)) \to \mathsf{plus}(\mathsf{times}(t, 0), t)$. Further defined contexts are, e.g., $\mathsf{plus}(\mathsf{plus}(\Box, t), t)$, $\mathsf{plus}(\mathsf{plus}(\mathsf{plus}(\Box, t), t), t)$, ... and $\mathsf{plus}(\mathsf{s}(\Box), \mathsf{s}(t))$, $\mathsf{plus}(\mathsf{s}(\mathsf{s}(\Box)), \mathsf{s}(\mathsf{s}(t)))$, ...*

Thm. 28 states that our notions of Def. 20 and 26 are indeed suitable to determine spareness.

Theorem 28 (No Defined and Duplicating Context $\Longrightarrow$ Spare)**.** *If no defined context is duplicating, then $\mathcal{R}$ is spare. If $\mathcal{R}$ is left-linear and spare, then no defined context is duplicating.*

Proof. If $\mathcal{R}$ is not spare, then there is a $s \in \mathcal{T}_B$ and a sequence $s \to^* t \to_{\ell \to r, \mu} u$ where all but the last step are spare, i.e., there are positions π, τ such that $t|_{\mu.\pi.\tau}$ is a redex, $\ell|_\pi = x \in \mathcal{V}$, and $\#_x(r) > 1$. Thus, $t|_\mu[\Box]_{\pi.\tau}$ is defined and duplicating. See App. A for the converse direction. $\Box$

So while the absence of contexts that are both defined and duplicating always implies spareness, the following example shows that the converse only holds for left-linear TRSs.

Example 29. *Consider the TRS with the rules $\mathsf{f}(x, x) \to \mathsf{g}(x, x)$, $\mathsf{b} \to \mathsf{f}(\mathsf{c}, \mathsf{a})$, and $\mathsf{c} \to \mathsf{f}(\mathsf{a}, \mathsf{a})$. Since the basic terms b, c, or $\mathsf{f}(t, t)$ for $t \in \mathcal{T}(\Sigma_c, \mathcal{V})$ only start rewrite sequences with spare steps, the TRS is spare. However, the context $\mathsf{f}(\Box, \mathsf{a})$ is both defined (due to the rewrite sequence $\mathsf{b} \to \mathsf{f}(\mathsf{c}, \mathsf{a})$) and duplicating (due to $\mathsf{c} \to \mathsf{f}(\mathsf{a}, \mathsf{a})$).*

As in the proof of Thm. 12 one can show that definedness of contexts is undecidable, too.

Theorem 30 (Defined Contexts are Undecidable)**.** *It is undecidable if a context is defined.*

Our aim is to use Thm. 28 to prove spareness of TRSs. Thus, we have to show that there is no context that is defined and duplicating. While these properties are both undecidable by Thm. 22 and 30, we can approximate duplicating contexts by *Dup* due to Lemma 25. Hence, we now have to find a similar over-approximation for defined contexts. Here, a problem is that a defined context may have *several* inner redexes (i.e., redexes can also occur on positions parallel to $\Box$).

Example 31. *Consider a TRS containing the rule $\mathsf{f}(x) \to \mathsf{h}(\mathsf{e}, \mathsf{g}(x))$, where h, e, and g are defined symbols (and thus e is a redex). To compute all defined contexts, we have to consider all terms t with $\mathsf{g}(s) \to^* t$ for some $s \in \mathcal{T}(\Sigma_c, \mathcal{V})$, as each of these terms gives rise to a rewrite sequence $\mathsf{f}(s) \to \mathsf{h}(\mathsf{e}, \mathsf{g}(s)) \to^* \mathsf{h}(\mathsf{e}, t)$, i.e., $\mathsf{h}(\Box, t)$ is a defined context for all these terms t.*

To avoid reasoning about all terms t reachable from instances of some term $\mathsf{g}(x)$ as in Ex. 31, we abstract inner defined symbols to variables in order to approximate the set of all defined contexts (e.g., the context $\mathsf{h}(\Box, \mathsf{g}(x))$ with the defined symbols h and g is abstracted to $\mathsf{h}(\Box, x_1)$). However, inner defined symbols above $\Box$ are abstracted to $\Box$ (e.g., the context $\mathsf{g}(\mathsf{g}(\Box))$ is abstracted to $\mathsf{g}(\Box)$ and $\mathsf{h}(\mathsf{e}, \mathsf{g}(\Box))$ is abstracted to $\mathsf{h}(x_1, \Box)$). Moreover, we also abstract from variables that occur multiple times in a term. To this end, we replace all occurrences of variables by pairwise different variables. The reason is that equal subterms may be reduced differently, i.e., equality of subterms is not preserved by rewriting. Thus, Def. 32 introduces the *abstraction* of a context C, which results from replacing all its topmost proper subterms that have a defined root (but do not contain $\Box$) or that are variables by pairwise different variables.

Definition 32 (Abstraction of Contexts)**.** *For a context C, let $\widetilde{C} = C[\Box]_\tau$ where τ is the topmost position of C with $\tau \neq \varepsilon$, $C|_\tau \trianglerighteq \Box$, and $\mathrm{root}(C|_\tau) \in \Sigma_d \cup \{\Box\}$.*[3] *Let $\Pi_d = \{\pi \mid \mathrm{root}(\widetilde{C}|_\pi) \in \Sigma_d, \pi \neq \varepsilon\}$ contain all positions of $\widetilde{C}$'s proper subterms with defined root and let $\Pi_{\mathcal{V}} = \{\pi \mid \widetilde{C}|_\pi \in \mathcal{V}\}$ contain all positions of variables in $\widetilde{C}$. Moreover, let Π consist of the topmost positions of $\Pi_d \cup \Pi_{\mathcal{V}}$, i.e., Π is the smallest subset of $\Pi_d \cup \Pi_{\mathcal{V}}$ such that for each $\tau \in \Pi_d \cup \Pi_{\mathcal{V}}$ there is a $\pi \in \Pi$ with $\pi \leq \tau$. Finally, let $\pi_1, \ldots, \pi_n$ be Π's elements in lexicographic order. Then we call $\lfloor C \rfloor = \widetilde{C}[x_1]_{\pi_1} \ldots [x_n]_{\pi_n}$ the* abstraction *of C, where $x_1, \ldots, x_n \in \mathcal{V}$ are pairwise different.*

Example 33. *Recall the rule $\mathsf{f}(x) \to \mathsf{h}(\mathsf{e}, \mathsf{g}(x))$ from Ex. 31. To approximate the defined contexts induced by this rule we first replace one topmost proper subterm of the right-hand side with defined root by $\Box$ and then we take the abstraction of the resulting context. In this way, we obtain the contexts $\lfloor \mathsf{h}(\Box, \mathsf{g}(x)) \rfloor = \mathsf{h}(\Box, x_1)$ and $\lfloor \mathsf{h}(\mathsf{e}, \Box) \rfloor = \mathsf{h}(x_1, \Box)$.*

Lemma 34 states that every context C is an instance of its abstraction $\lfloor C \rfloor$. Moreover, if C is an instance of D, then $\lfloor C \rfloor$ is also an instance of D, provided that D is a linear basic context.

Lemma 34 (Properties of $\lfloor\ \rfloor$)**.**

(a) *For any context C, we have $C \sqsubseteq \lfloor C \rfloor$.*

(b) *For any context C and any linear basic context D, $C \sqsubseteq D$ implies $\lfloor C \rfloor \sqsubseteq D$.*

(c) *For any context C, any $\pi \in \mathrm{pos}(C)$ with $C|_\pi \not\trianglerighteq \Box$, and any term t with $\mathrm{root}(t) \in \Sigma_d$, we have $\lfloor C \rfloor \sqsubseteq \lfloor C[t]_\pi \rfloor$.*

Proof. For (a), we first show $C \sqsubseteq \widetilde{C}$. We have $\widetilde{C}|_\tau = \Box$. The position of $\Box$ in C is indeed below τ since $C|_\tau \trianglerighteq \Box$. Moreover $\widetilde{C}[x]_\tau$ matches C for a fresh variable x, since $\widetilde{C}[x]_\tau = C[x]_\tau$.

[3] Note that τ is unique since $\Box$ only occurs once in C.

Now we show that $\widetilde{C} \sqsubseteq \lfloor C \rfloor$. For all $\pi \in \Pi_d$, $\widetilde{C}|_\pi$ does not contain $\Box$. Thus, $\Box$ is at the same position in $\widetilde{C}$ and $\lfloor C \rfloor$. Moreover, by instantiating every x_i by $\widetilde{C}|_{\pi_i}$, $\lfloor C \rfloor$ matches $\widetilde{C}$. Hence, the claim follows from transitivity of $\sqsubseteq$ (Cor. 17 (a)).

For (b), let $C \sqsubseteq D$. We first show that this implies $\widetilde{C} \sqsubseteq D$. Let $C|_\pi = \Box$ and $D|_\mu = \Box$. Then $C \sqsubseteq D$ implies $\pi \geq \mu$. Moreover, $\widetilde{C}|_\tau = \Box$ with $\pi \geq \tau$. We also obtain $\tau \geq \mu$, because otherwise we have $\mu > \tau$, but then $D[x]_\mu$ would not match C, since $\mathrm{root}(C|_\tau) \in \Sigma_d$ and $\mathrm{root}(D[x]_\mu|_\tau) = \mathrm{root}(D|_\tau) \notin \Sigma_d$ as D is basic. Let σ be the matcher with $D[x]_\mu\sigma = C$. By defining $x\sigma' = \widetilde{C}|_\mu$ and $y\sigma' = y\sigma$ for all variables $y \neq x$, we get $D[x]_\mu\sigma' = D\sigma[\widetilde{C}|_\mu]_\mu = C[\widetilde{C}|_\mu]_\mu = \widetilde{C}$.

To show $\lfloor C \rfloor \sqsubseteq D$, note again that $\Box$ is at the same position τ in $\widetilde{C}$ and $\lfloor C \rfloor$, i.e., for $D|_\mu = \Box$ we have $\tau \geq \mu$. Moreover, as $D[x]_\mu$ matches $\widetilde{C}$ and D is basic, we must have $D|_{\pi_i} \in \mathcal{V}$ for all $1 \leq i \leq n$. The variables $D|_{\pi_i}$ are pairwise different, since D is linear. Hence, by instantiating every variable $D|_{\pi_i}$ by x_i, $D[x]_\mu$ matches $\lfloor C \rfloor$.

For (c), since $C|_\pi \not\trianglerighteq \Box$, the position of $\Box$ is the same in $\lfloor C \rfloor$ and $\lfloor C[t]_\pi \rfloor$. Moreover up to variable renaming, their only difference is that there could be a position above or equal to π where $\lfloor C[t]_\pi \rfloor$ has a fresh variable (since $\mathrm{root}(t) \in \Sigma_d$). Hence $\lfloor C[t]_\pi \rfloor$ matches $\lfloor C \rfloor$. $\Box$

To approximate the set of all defined contexts, we iteratively compute a set *Def* such that each defined context is an instance of an element of *Def*. For every rule $\ell \to r$ where ℓ is basic, every subterm $C[p]$ of r leads to a defined context $C\sigma$ if $\mathrm{root}(C) \in \Sigma_d$ and $p\sigma$ reduces to a redex. Moreover, given a rule $\ell \to r$ with $\ell|_\pi = x \in \mathcal{V}$ and a defined context D, consider the case that D overlaps with the context $\ell[\Box]_\pi$. So D represents terms which have a redex below the position of $\Box$ and $\ell[\Box]_\pi$ also represents some of these terms. Then by the application of the rule $\ell \to r$, the inner defined symbol represented by $\Box$ is copied to all occurrences of x in r. If one of these occurrences is below a defined symbol, then we again obtain a defined context.

Example 35. *Consider the following TRS:*

$$\mathsf{f}(w, x, y, z) \to \mathsf{g}(\mathsf{h}) \quad (5) \qquad \mathsf{g}(\mathsf{s}(x)) \to \mathsf{f}(x, x, x, \mathsf{h}) \quad (6) \qquad \mathsf{h} \to \mathsf{s}(\mathsf{h}) \quad (7)$$

The context $\mathsf{g}(\Box)$ is defined due to (5). By replacing the variable x in the left-hand side of (6) with $\Box$, we obtain the context $\mathsf{g}(\mathsf{s}(\Box))$. As $\mathsf{g}(\Box)$ and $\mathsf{g}(\mathsf{s}(\Box))$ overlap, the defined symbol below $\Box$ in $\mathsf{g}(\Box)$ can be copied to all occurrences of x in the right-hand side of (6). Hence, instances of $\mathsf{f}(\Box, x, x, \mathsf{h})$, $\mathsf{f}(x, \Box, x, \mathsf{h})$, and $\mathsf{f}(x, x, \Box, \mathsf{h})$ might be defined. To avoid reasoning about the terms reachable from h, we replace it with a variable. Finally, we abstract from the multiple occurrences of x by replacing them with different variables. Hence, we add $\lfloor \mathsf{f}(\Box, x, x, \mathsf{h}) \rfloor = \mathsf{f}(\Box, x_1, x_2, x_3)$, $\lfloor \mathsf{f}(x, \Box, x, \mathsf{h}) \rfloor = \mathsf{f}(x_1, \Box, x_2, x_3)$, and $\lfloor \mathsf{f}(x, x, \Box, \mathsf{h}) \rfloor = \mathsf{f}(x_1, x_2, \Box, x_3)$ to Def.

In Rule (5) of Ex. 35, we obtained a defined context by replacing the subterm h of the right-hand side with $\Box$. In general, we have to consider all instances of subterms which reduce to a redex. For the sake of simplicity, we over-approximate the set of such subterms by considering all subterms p of right-hand sides with $\mathrm{root}(p) \in \Sigma_d$. Then, we obtain an over-approximation of all defined contexts by iterating the construction illustrated in Ex. 35.

Definition 36 ($\mathit{Def}_\mathcal{R}$)**.** *Given a TRS $\mathcal{R}$, we define $\mathit{Def}_\mathcal{R}$ to be the smallest set such that:*

(a) *If $\ell \to r \in \mathcal{R}$, $r \trianglerighteq C[p]$, and $\mathrm{root}(C), \mathrm{root}(p) \in \Sigma_d$, then $\lfloor C \rfloor \in \mathit{Def}_\mathcal{R}$.*

(b) *If $D \in \mathit{Def}_\mathcal{R}$, $\ell[x]_\pi \to r \in \mathcal{R}$ with $r \trianglerighteq C[x]$ and $\mathrm{root}(C) \in \Sigma_d$, and D and $\ell[\Box]_\pi$ overlap, then $\lfloor C \rfloor \in \mathit{Def}_\mathcal{R}$.*

We omit the index $\mathcal{R}$ if it is clear from the context.

Lemma 37 shows that *Def* is finite (and hence computable), as it only contains abstractions of contexts that result from replacing subterms of right-hand sides of rules with $\Box$.

Lemma 37 (Finiteness of *Def*)**.** *For each TRS $\mathcal{R}$, $Def_{\mathcal{R}}$ is finite.*

Example 38. *For the TRS $\mathcal{R}_{\mathsf{times}}$ of Ex. 1, by Def. 36 (a) we have $\lfloor \mathsf{plus}(\Box, x) \rfloor = \mathsf{plus}(\Box, x_1) \in Def_{\mathcal{R}_{\mathsf{times}}}$ due to the right-hand side of Rule (4). This context overlaps with the context $\mathsf{plus}(\mathsf{s}(\Box), y)$ obtained from the left-hand side of Rule (2) by replacing the variable x by $\Box$. Since the corresponding right-hand side is $\mathsf{s}(\mathsf{plus}(x, y))$, this implies $\lfloor \mathsf{plus}(\Box, y) \rfloor \in Def_{\mathcal{R}_{\mathsf{times}}}$. As $\lfloor \mathsf{plus}(\Box, y) \rfloor = \mathsf{plus}(\Box, x_1)$, we therefore obtain $Def_{\mathcal{R}_{\mathsf{times}}} = \{\mathsf{plus}(\Box, x_1)\}$.*

Lemma 39 shows that the approximation of Def. 36 is indeed correct.

Lemma 39 (*Def* Over-Approximates Defined Contexts)**.** *If C is defined, then there is a $D \in Def$ such that $C \sqsubseteq D$.*

Proof. We use induction on n to prove that if there is a rewrite sequence $s \to^n t \trianglerighteq_\pi C[p]_\tau$ where s is basic, $C|_\tau = \Box$, and $\mathrm{root}(C), \mathrm{root}(p) \in \Sigma_d$, then there is a $D \in Def$ with $C \sqsubseteq D$.

<u>**Induction Base $(n = 0)$.**</u> As s is basic, s cannot have a subterm $C[p]$ such that $\mathrm{root}(C), \mathrm{root}(p) \in \Sigma_d$. Hence, our claim trivially holds.

<u>**Induction Step $(n > 0)$.**</u> Here, we have $s \to^{n-1} s' \to_{\ell \to r, \mu} t \trianglerighteq_\pi C[p]_\tau$ for some rule $\ell \to r$.

***<u>Case 1:</u>* μ *and* π *are parallel positions, i.e.,* $\mu \| \pi$.** Then we also have $s \to^{n-1} s' \trianglerighteq_\pi C[p]_\tau$ and hence our claim follows from the induction hypothesis.

***<u>Case 2:</u>* μ *is below* π*, but parallel to* $\pi.\tau$*, i.e.,* $\mu = \pi.\iota$ *and* $\iota \| \tau$.** We get $s \to^{n-1} s' \trianglerighteq_\pi C[\ell\sigma]_\iota[p]_\tau$. By the induction hypothesis, there is a $D \in Def$ such that $C[\ell\sigma]_\iota[\Box]_\tau \sqsubseteq D$. By construction, each $D \in Def$ is basic and linear. Hence by Lemma 34 (b), we get $\lfloor C[\ell\sigma]_\iota[\Box]_\tau \rfloor \sqsubseteq D$. Moreover, we have $C \sqsubseteq \lfloor C \rfloor$ by Lemma 34 (a) and $\lfloor C \rfloor = \lfloor C[r\sigma]_\iota[\Box]_\tau \rfloor \sqsubseteq \lfloor C[\ell\sigma]_\iota[\Box]_\tau \rfloor$ by Lemma 34 (c), since $\mathrm{root}(\ell\sigma) \in \Sigma_d$. Hence, $C \sqsubseteq D$ follows by transitivity of $\sqsubseteq$ (Cor. 17 (a)).

***<u>Case 3:</u>* μ *is below* $\pi.\tau$ $(\mu \geq \pi.\tau)$.** Here, $s \to^{n-1} s' \trianglerighteq_\pi C[q]_\tau$ with $\mathrm{root}(q) \in \Sigma_d$, as $q = \ell\sigma$ if $\mu = \pi.\tau$ and $\mathrm{root}(q) = \mathrm{root}(p)$ if $\mu > \pi.\tau$. So our claim follows from the induction hypothesis.

***<u>Case 4:</u>* μ *is strictly below* π*, but strictly above* $\pi.\tau$ $(\pi.\tau > \mu > \pi)$.** Then there is a position ν with $\pi.\nu = \mu$. We get $s \to^{n-1} s' \trianglerighteq_\pi C[\ell\sigma]_\nu$ where $\mathrm{root}(\ell\sigma) \in \Sigma_d$. The induction hypothesis implies that there is a $D \in Def$ such that $C[\Box]_\nu \sqsubseteq D$. Clearly, we have $\nu < \tau$ and hence, $C = C[\Box]_\tau \sqsubseteq C[\Box]_\nu$ by Cor. 17 (c). Hence, $C \sqsubseteq D$ follows from transitivity of $\sqsubseteq$ (Cor. 17 (a)).

***<u>Case 5:</u>* π *is below* μ*, i.e.,* $\mu \leq \pi$.** Then there is a position ν such that $\mu.\nu = \pi$. Thus, we have $s \to^{n-1} s' \to_{\ell \to r, \mu} s'[r\sigma]_\mu \trianglerighteq r\sigma[C[p]_\tau]_\nu$.

<u>Case 5.1:</u> $\nu.\tau \in \mathrm{pos}(r)$ and $r|_{\nu.\tau} \notin \mathcal{V}$. Then $\mathrm{root}(C) = \mathrm{root}(r|_\nu)$ and $\mathrm{root}(p) = \mathrm{root}(r|_{\nu.\tau})$. Hence, we obtain $\lfloor r|_\nu[\Box]_\tau \rfloor \in Def$ by Def. 36 (a). Moreover, $C \sqsubseteq r|_\nu[\Box]_\tau$ holds as C also has $\Box$ at the position τ, and as $r|_\nu\,\sigma = C[p]_\tau$ and thus, $r|_\nu[x]_\tau\,\sigma' = C$ if $x\sigma' = \Box$ and $y\sigma' = y\sigma$ for all variables $y \neq x$. By Lemma 34 (a) and transitivity of $\sqsubseteq$ (Cor. 17 (a)), we get $C \sqsubseteq \lfloor r|_\nu[\Box]_\tau \rfloor$.

<u>Case 5.2:</u> $\nu \in \mathrm{pos}(r)$, $r|_\nu \notin \mathcal{V}$, and ($\nu.\tau \notin \mathrm{pos}(r)$ or $r|_{\nu.\tau} \in \mathcal{V}$). In this case, the root of C is "above" and p is "below" some variable x of r in $r\sigma$, cf. Fig. 1. So $\tau = \xi.\iota$ such that $\xi \neq \varepsilon$, $r|_{\nu.\xi} = x \in \mathcal{V}$, and $x\sigma|_\iota = r\sigma|_{\nu.\xi.\iota} = r\sigma|_{\nu.\tau} = p$. As $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$, there is also some $\pi \in \mathrm{pos}(\ell)$ with $\ell|_\pi = x$. Note that $\ell\sigma|_{\pi.\iota} = x\sigma|_\iota = p$.

As $\ell \notin \mathcal{V}$, we have $\pi \neq \varepsilon$. Since $\mathrm{root}(\ell)$, $\mathrm{root}(p) \in \Sigma_d$ and $s \to^{n-1} s' \trianglerighteq \ell\sigma = \ell\sigma[p]_{\pi.\iota}$, there is a $D \in \mathit{Def}$ such that $\ell\sigma[\Box]_{\pi.\iota} \sqsubseteq D$ by the induction hypothesis. Moreover, we have $\mathrm{root}(r|_\nu) \in \Sigma_d$ as ν is the position of C in $r\sigma$ and as $r|_\nu \notin \mathcal{V}$. We obtain $\lfloor r|_\nu[\Box]_\xi \rfloor \in \mathit{Def}$ by Def. 36 (b), since the following holds:

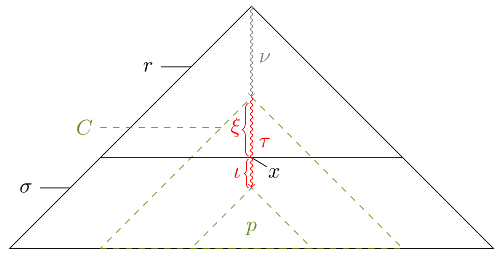

Figure 1: Case 5.2

- $D \in \mathit{Def}$
- $\ell \to r = \ell[x]_\pi \to r \in \mathcal{R}$ with $r = r[x]_{\nu.\xi} \trianglerighteq r|_\nu[x]_\xi$
- $\mathrm{root}(r|_\nu) \in \Sigma_d$
- D and $\ell[\Box]_\pi$ overlap as $\ell\sigma[\Box]_{\pi.\iota} \sqsubseteq D$ and $\ell\sigma[\Box]_{\pi.\iota} \sqsubseteq \ell[\Box]_\pi$; the latter holds due to Cor. 17 (b), (c), and (a)

We now prove $C \sqsubseteq r|_\nu[\Box]_\xi$. Then, $C \sqsubseteq \lfloor r|_\nu[\Box]_\xi \rfloor$ follows by Lemma 34 (a) and transitivity of $\sqsubseteq$ (Cor. 17 (a)). Note that $\Box$ is at position τ in C and at position ξ in $r|_\nu[\Box]_\xi$ with $\xi \leq \tau = \xi.\iota$. Moreover, let x' be a fresh variable where $x'\sigma = x\sigma[\Box]_\iota$. Then $r|_\nu[x']_\xi\,\sigma = r\sigma|_\nu[x'\sigma]_\xi = r\sigma|_\nu[x\sigma[\Box]_\iota]_\xi = (r\sigma[x\sigma]_{\nu\xi}[\Box]_{\nu\xi\iota})|_\nu = r\sigma[\Box]_{\nu\tau}|_\nu = r\sigma|_\nu[\Box]_\tau = C[p]_\tau[\Box]_\tau = C$.

Case 5.3: $\nu \notin \mathrm{pos}(r)$ *or* $r|_\nu \in \mathcal{V}$. Then there is an $x \in \mathcal{V}(r)$ with $x\sigma \trianglerighteq C[p]$. As we also have $x \in \mathcal{V}(\ell)$, we obtain $s' \trianglerighteq \ell\sigma \trianglerighteq x\sigma \trianglerighteq C[p]$, i.e., the claim follows by the induction hypothesis. ☐

This leads to our main theorem: If the contexts in *Def* do not overlap with the contexts in *Dup*, then $\mathcal{R}$ is spare. So if $\mathcal{R}$ is an overlay system then rc and irc coincide by Cor. 8 (b) and 11.

Theorem 40 (Approximating Spareness by *Def* and *Dup*)**.** *If there is no $D \in \mathit{Def}_\mathcal{R}$ which overlaps with some $C \in \mathit{Dup}_\mathcal{R}$, then $\mathcal{R}$ is spare.*

Proof. Assume that $\mathcal{R}$ is not spare. By Thm. 28, then there is a defined context E that is duplicating. By Lemma 25 and 39 there are $C \in \mathit{Dup}_\mathcal{R}$ and $D \in \mathit{Def}_\mathcal{R}$ with $E \sqsubseteq C$ and $E \sqsubseteq D$. Hence, C and D overlap which contradicts the prerequisite of the theorem. ☐

The criterion of Thm. 40 can easily be automated since $\mathit{Def}_\mathcal{R}$ and $\mathit{Dup}_\mathcal{R}$ are computable finite sets of contexts and for any two contexts, it is decidable whether they overlap.

Example 41. *We have $\mathit{Dup}_{\mathcal{R}_{\mathsf{times}}} = \{\mathsf{times}(\Box, \mathsf{s}(y))\}$ and $\mathit{Def}_{\mathcal{R}_{\mathsf{times}}} = \{\mathsf{plus}(\Box, x_1)\}$, cf. Ex. 24 and 38. Clearly, $\mathit{Dup}_{\mathcal{R}_{\mathsf{times}}}$ and $\mathit{Def}_{\mathcal{R}_{\mathsf{times}}}$ do not overlap. As $\mathcal{R}_{\mathsf{times}}$ is an overlay system, this implies $\mathrm{rc}_{\mathcal{R}_{\mathsf{times}}} = \mathrm{irc}_{\mathcal{R}_{\mathsf{times}}}$. Current complexity analysis tools are able to prove the tight upper bound $\mathrm{irc}_{\mathcal{R}_{\mathsf{times}}}(n) \in \mathcal{O}(n^3)$ automatically. However, they could not infer any polynomial upper bound for $\mathrm{rc}_{\mathcal{R}_{\mathsf{times}}}$ so far. Using our new technique from this section, our tool* AProVE *can now prove $\mathrm{rc}_{\mathcal{R}_{\mathsf{times}}}(n) \in \mathcal{O}(n^3)$ automatically by showing $\mathrm{irc}_{\mathcal{R}_{\mathsf{times}}}(n) \in \mathcal{O}(n^3)$ and $\mathrm{rc}_{\mathcal{R}_{\mathsf{times}}} = \mathrm{irc}_{\mathcal{R}_{\mathsf{times}}}$.*

Note that for spareness, Thm. 40 clearly subsumes Lemma 13 and 14. Lemma 13 is subsumed as $\mathit{Dup}_\mathcal{R} = \varnothing$ if $\mathcal{R}$ is right-linear. Lemma 14 is subsumed since $\mathit{Def}_\mathcal{R} = \varnothing$ if $\mathcal{R}$ has no rules where defined symbols are nested on right-hand sides. Thus, in both cases Thm. 40 implies spareness.

5 Handling Non-Overlay Systems

For overlay systems, our criterion for spareness in Thm. 40 implies the desired ndg property as well. We now introduce a technique to prove ndg also for non-overlay systems. It tries to identify rules with non-basic left-hand sides that are not reachable from basic terms. These rules can be removed from the TRS without affecting its runtime complexity. If this removal is possible for *all* rules with non-basic left-hand sides, then we obtain a constructor system and hence, an overlay system, where spareness and ndg are equivalent. Of course, the technique of this section could also be used in other applications concerned with reachability analysis for TRSs.

To detect these removable rules, we analyze which functions are "called" by other functions, similar to the computation of "usable rules" in the dependency pair technique [2, 11]. However, in contrast to usable rules, we have to infer information on the possible nesting of defined symbols, i.e., we determine whether some function symbol g can possibly occur in the i-th argument of some function symbol f. To this end, for each defined symbol f and each argument position i of f, we compute a set $\Sigma_d|_{f,i}$ which over-approximates those defined symbols that may occur below the i-th argument of f in rewrite sequences starting with basic terms. Thus, we can remove all rules where some defined symbol $g \notin \Sigma_d|_{f,i}$ is below the i-th argument of f on the left-hand side. In the following, let $\Sigma_d(t)$ denote the set of all defined symbols occurring in a term t.

Example 42. *Consider the following TRS $\mathcal{R}$.*

$$
\begin{array}{rcllrcll}
\mathsf{inc}(x) & \to & \mathsf{s}(x) & (8) & \mathsf{times}(x,\mathsf{0}) & \to & \mathsf{0} & (11)\\
\mathsf{plus}(\mathsf{0},y) & \to & y & (9) & \mathsf{times}(x,\mathsf{s}(y)) & \to & \mathsf{plus}(\mathsf{times}(x,y),x) & (12)\\
\mathsf{plus}(\mathsf{s}(x),y) & \to & \mathsf{plus}(x,\mathsf{inc}(y)) & (10) & \mathsf{plus}(x,\mathsf{plus}(y,z)) & \to & \mathsf{plus}(\mathsf{plus}(x,y),z) & (13)
\end{array}
$$

Since the left-hand side of (10) is basic, inc *can occur below the second argument of* plus *in rewrite sequences starting with basic terms, i.e.,* $\mathsf{inc} \in \Sigma_d|_{\mathsf{plus},2}$. *Similarly, as the left-hand side of (12) is basic,* times *can occur below* plus*'s first argument, i.e.,* $\mathsf{times} \in \Sigma_d|_{\mathsf{plus},1}$. *So in general, we include $g \in \Sigma_d|_{f,i}$ if a rule $\ell \to r$ can be applied in rewrite sequences starting with basic terms, where $\mathrm{root}(r|_\pi) = f$ and $g \in \Sigma_d(r|_{\pi.i})$ for some π. This leads to Condition (a) in Thm. 43 below.*

As times *may rewrite to a term containing* plus *or* inc *due to Rules (12) and (10),* plus *and* inc *may also occur below the first argument of* plus*, i.e.,* $\{\mathsf{plus},\mathsf{inc}\} \subseteq \Sigma_d|_{\mathsf{plus},1}$. *Thus, in general we include $g \in \Sigma_d|_{f,i}$ if there is a rule $\ell \to r$ that is applicable in rewrite sequences starting with basic terms, where $\mathrm{root}(\ell) \in \Sigma_d|_{f,i}$ and $g \in \Sigma_d(r)$, cf. Condition (c) in Thm. 43.*

Since inc *may occur below the second argument of* plus*, the variable y may match a term containing* inc *in Rule (10). Hence,* inc *may also occur below the only argument of* inc*, i.e.,* $\mathsf{inc} \in \Sigma_d|_{\mathsf{inc},1}$. *So in general, we also include $g \in \Sigma_d|_{f,i}$ if a rule $h(t_1,\dots,t_n) \to r$ is applicable in rewrite sequences starting with basic terms, where $\mathrm{root}(r|_\pi) = f$, $y \in \mathcal{V}(r|_{\pi.i})$ for some π, $y \in \mathcal{V}(t_j)$, and $g \in \Sigma_d|_{h,j}$ for some $j \in \{1,\dots,n\}$. This leads to Condition (b) in Thm. 43.*

In our example, no other defined symbols can be nested in rewrite sequences starting with basic terms. As $\mathsf{plus} \notin \Sigma_d|_{\mathsf{plus},2}$*, Rule (13) can never be applied in such sequences and hence, it may be removed from the TRS. Then, our tool* AProVE *can prove the tight bound $\mathrm{rc}_{\mathcal{R}\setminus\{(13)\}}(n) \in \mathcal{O}(n^3)$ using the technique from Sect. 4, which was not possible with existing tools so far.*

Theorem 43 (Removing Non-Reachable Rules for rc)**.** *For each $f \in \Sigma_d$ and each $i \in \{1,\dots,\mathrm{arity}(f)\}$, let $\Sigma_d|_{f,i}$ be the smallest set such that $g \in \Sigma_d|_{f,i}$ if there is some rule $\ell = h(t_1,\dots,t_n) \to r \in \mathcal{R}$ where $\Sigma_d(t_j) \subseteq \Sigma_d|_{h,j}$ for all $j \in \{1,\dots,n\}$ and at least one of the following conditions (a) – (c) holds:*

(a) $\mathrm{root}(r|_\pi) = f$ *and* $g \in \Sigma_d(r|_{\pi.i})$ *for some position* $\pi \in \mathrm{pos}(r)$

(b) $\mathrm{root}(r|_\pi) = f$, $y \in \mathcal{V}(r|_{\pi.i})$, $y \in \mathcal{V}(t_j)$, *and* $g \in \Sigma_d|_{h,j}$ *for some* π *and some* $j \in \{1,\dots,n\}$

(c) $\mathrm{root}(\ell) \in \Sigma_d|_{f,i}$ *and* $g \in \Sigma_d(r)$

If $\ell \to r \in \mathcal{R}$, $\pi \in \mathrm{pos}(\ell)$, $\mathrm{root}(\ell|_\pi) = f \in \Sigma_d$, and $g \in \Sigma_d(\ell|_{\pi.i}) \setminus \Sigma_d|_{f,i}$, then $\mathrm{rc}_{\mathcal{R}} = \mathrm{rc}_{\mathcal{R}\setminus\{\ell\to r\}}$.

Proof. By induction on m, we prove that if there is a rewrite sequence $q_0 \to q_1 \to \ldots \to q_{m-1} \to q_m \trianglerighteq f(s_1,\dots,s_k)$ where $q_0 \in \mathcal{T}_B$, $f \in \Sigma_d$, and $g \in \Sigma_d(s_i)$, then $g \in \Sigma_d|_{f,i}$.

In the induction base, we have $m = 1$. Thus $g \in \Sigma_d|_{f,i}$ follows by (a) as q_0 is basic. In the induction step ($m > 1$), let $h(t_1, \dots, t_n) \to r$ be the rule applied in the rewrite step $q_{m-1} \to q_m$, i.e., $q_{m-1}|_\pi = h(t_1, \dots, t_n)\sigma$ and $q_m = q_{m-1}[r\sigma]_\pi$. As $q_{m-1} \trianglerighteq h(t_1\sigma, \dots, t_n\sigma)$, the induction hypothesis implies $\Sigma_d(t_j) \subseteq \Sigma_d(t_j\sigma) \subseteq \Sigma_d|_{h,j}$ for all $j \in \{1, \dots, n\}$. Thus, the rule $h(t_1, \dots, t_n) \to r$ satisfies the requirements of Thm. 43. Let $\iota \in \text{pos}(q_m)$ with $q_m|_\iota = f(s_1, \dots, s_k)$.

Case 1: $\iota \| \pi$. Then $q_{m-1} \trianglerighteq f(s_1, \dots, s_k)$ and $g \in \Sigma_d(s_i) \subseteq \Sigma_d|_{f,i}$ by the induction hypothesis.

Case 2: $\iota \geq \pi$, i.e., there is a position τ such that $\iota = \pi.\tau$.

***Case 2.1:* $\tau \notin \text{pos}(r)$ *or* $r|_\tau \in \mathcal{V}$.** Then again we have $q_{m-1} \trianglerighteq f(s_1, \dots, s_k)$ and thus, $g \in \Sigma_d|_{f,i}$ follows from the induction hypothesis.

***Case 2.2:* $\tau \in \text{pos}(r)$ *and* $r|_\tau \notin \mathcal{V}$.** Thus, $\text{root}(r|_\tau) = f$ and hence $\tau.i \in \text{pos}(r)$. Let $\xi \in \text{pos}(s_i)$ with $\text{root}(s_i|_\xi) = g$.

Case 2.2.1: $\tau.i.\xi \in \text{pos}(r)$ and $r|_{\tau.i.\xi} \notin \mathcal{V}$. Then $r|_{\tau.i.\xi}\,\sigma = r\sigma|_{\tau.i.\xi} = f(s_1, \dots, s_k)|_{i.\xi} = s_i|_\xi$ implies $\text{root}(r|_{\tau.i.\xi}) = g$. Hence, we have $g \in \Sigma_d|_{f,i}$ by (a).

Case 2.2.2: $\tau.i.\xi \notin \text{pos}(r)$ or $r|_{\tau.i.\xi} \in \mathcal{V}$. Then there is a prefix ξ' of ξ such that $r|_{\tau.i.\xi'} = y \in \mathcal{V}$ and $g \in \Sigma_d(y\sigma)$. Hence, we also have $y \in \mathcal{V}(t_j)$ for some $j \in \{1, \dots, n\}$, and thus $g \in \Sigma_d(t_j\sigma)$. Therefore, $q_{m-1} \trianglerighteq h(t_1\sigma, \dots, t_n\sigma)$ implies $g \in \Sigma_d|_{h,j}$ by the induction hypothesis. So we have $\text{root}(r|_\tau) = f$, $y \in \mathcal{V}(r|_{\tau.i})$, $y \in \mathcal{V}(t_j)$, and $g \in \Sigma_d|_{h,j}$, which implies $g \in \Sigma_d|_{f,i}$ by (b).

Case 3: $\iota < \pi$, i.e., there is a $\tau \neq \varepsilon$ with $\pi = \iota.\tau$. Again, let $\xi \in \text{pos}(s_i)$ with $\text{root}(s_i|_\xi) = g$.

***Case 3.1:* $\tau \| i.\xi$ *or* $\tau > i.\xi$** Then $q_{m-1} \trianglerighteq f(s'_1, \dots, s'_k)$ where $g \in \Sigma_d(s'_i)$. Hence, we get $g \in \Sigma_d|_{f,i}$ by the induction hypothesis.

***Case 3.2:* $\tau = i.\tau'$ *with* $\tau' \leq \xi$.** Then we have $\text{root}(q_{m-1}|_\iota) = f$ and $\text{root}(q_{m-1}|_{\iota.i.\tau'}) = h$. Hence, we have $h \in \Sigma_d|_{f,i}$ by the induction hypothesis. Moreover, $g \in \Sigma_d(s_i|_\xi) \subseteq \Sigma_d(s_i|_{\tau'}) = \Sigma_d(r\sigma)$. Hence, we have $g \in \Sigma_d(r)$ or there is a variable $y \in \mathcal{V}(r)$ such that $g \in \Sigma_d(y\sigma)$.

Case 3.2.1: $g \in \Sigma_d(r)$. As we have $h \in \Sigma_d|_{f,i}$, we get $g \in \Sigma_d|_{f,i}$ by (c).

Case 3.2.2: there is a variable $y \in \mathcal{V}(r)$ such that $g \in \Sigma_d(y\sigma)$. As we also have $y \in \mathcal{V}(h(t_1, ..., t_n))$, we get $g \in \Sigma_d(q_{m-1}|_\pi)$. Since $\text{root}(q_{m-1}|_\iota) = f$ and $\iota.i.\tau' = \iota.\tau = \pi$, we obtain $g \in \Sigma_d|_{f,i}$ by the induction hypothesis. $\square$

Note that together with our criterion for checking spareness (Thm. 40), Thm. 43 subsumes the simple criterion for ndg in Lemma 14, because all sets $\Sigma_d|_{f,i}$ are empty for TRSs without nested defined symbols in right-hand sides. So for such TRSs, Thm. 43 allows us to remove all rules with nested defined symbols in the left-hand side and thus, spareness implies ndg.

We also considered re-using our approximation *Def* from Sect. 4 in order to obtain more precise information on possible nestings of defined symbols than the information provided by the sets $\Sigma_d|_{f,i}$. For each $C \in \mathit{Def}$ one could compute which defined symbols can occur below $\Box$. Then in Ex. 42 we would find out that inc might occur below $\Box$ in the context $\mathsf{plus}(x_1, \Box) \in \mathit{Def}_\mathcal{R}$. Thus, if we had $\mathsf{plus}(x_1, \mathsf{s}(\Box)) \in \mathit{Def}_\mathcal{R}$, we could express that certain defined symbols may only occur below plus's second argument if the root of plus's second argument is s. However, a prototypical implementation of this approach did not improve the results, such that we discarded it.

6 Experiments and Conclusion

We presented a novel technique to prove upper bounds on the runtime complexity of TRSs. Our technique is based on the observation that rc and irc coincide if the TRS is ndg. So for this class of

TRSs, all techniques to analyze irc can be used to analyze rc. A TRS is ndg if in all rewrite sequences that start with basic terms, (i) the variables that occur more than once on right-hand sides and (ii) proper subterms of left-hand sides with defined root symbols are instantiated to normal forms, cf. Sect. 3. We showed that, (i) – called spareness – is already undecidable. Hence, we developed an approximation which can prove spareness of TRSs in many cases, cf. Sect. 4. Here, the idea is to over-approximate the contexts of nested defined symbols as well as the contexts of terms that are duplicated. If both approximations do not overlap, then the TRS is spare. As (ii) trivially holds for overlay systems, this technique can prove that overlay systems are ndg. To handle non-overlay systems, we introduced a technique to identify rules with non-basic left-hand sides that are not reachable from basic terms, cf. Sect. 5. By removing these rules before analyzing runtime complexity, many TRSs can be transformed into overlay systems.

We implemented our technique in the tool AProVE. To evaluate its power, we analyzed all examples of the category "*Runtime Complexity – Full Rewriting*" of the *Termination Problems Data Base* 10.4. This collection of examples was used at the *Termination Competition 2016*. Besides our tool AProVE, we also tested with TcT [7], since AProVE and TcT were the most powerful tools for analyzing rc at the Termination Competition 2016. There, AProVE applied a preliminary version of the technique from this paper (which was the reason why AProVE won this category). While both AProVE and TcT also support the inference of lower bounds, in the following experiments both tools were configured to just prove upper bounds unless stated otherwise. (While our results also allow to apply techniques for lower bounds on rc to infer lower bounds on irc, further experiments showed that existing techniques do not benefit from this approach.)

We omitted 60 non-standard TRSs with extra variables on right-hand sides from our experiments. Moreover, in the following Tables 1 – 3 we omitted 235 examples where AProVE can prove a super-polynomial lower bound on rc. The reason is that AProVE and TcT just support polynomial upper bounds. In all tables, "$>$ *poly*" refers to (possibly infinite) super-polynomial bounds.

Table 1 compares TcT with AProVE where we used a timeout of 60 s for each tool on each example. The entries below the diagonal correspond to examples where AProVE's results are better than TcT's (e.g., there are 65 examples where TcT could not prove any polynomial upper bound on rc whereas AProVE now infers a linear bound). Similarly, the entries on the diagonal denote examples where both tools obtain the same result, and the entries above the diagonal are examples where TcT is better than AProVE. Thus, AProVE yields better results in 118 cases and TcT yields better results in 43 cases. Among the 270 TRSs where AProVE infers a polynomial upper bound are 120 non-constructor systems where the technique of Sect. 5 removes rules with non-basic left-hand sides, i.e., the improvement of Sect. 5 increases the performance of AProVE significantly. The average runtime of AProVE on each example was 11.4 s and the average runtime of TcT was 40.3 s. However, comparisons of the performance of different complexity analysis tools should be treated with caution. The reason is that we deal with an optimization problem and, in general, it is not possible to check if the current solution is optimal. Hence, it is often a good strategy to try to improve the current result until the specified timeout expires, even though AProVE does not take this approach.

		AProVE					
	$\mathrm{rc}_{\mathcal{R}}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^5)$	$>$ *poly*
TcT	$\mathcal{O}(1)$	21	1	–	–	–	–
	$\mathcal{O}(n)$	18	110	9	–	–	24
	$\mathcal{O}(n^2)$	–	7	8	2	–	4
	$\mathcal{O}(n^3)$	–	–	1	1	–	2
	$\mathcal{O}(n^5)$	–	–	–	–	–	1
	$>$ *poly*	7	65	17	3	–	363

Table 1: TcT vs. AProVE

According to Table 1, AProVE is now the most powerful tool for upper bounds on rc. However, this is only due to the results of the current paper. To demonstrate this, we show that TcT can outperform AProVE again by integrating the technique from this paper. To this end, Table 2 compares AProVE with "TcT preproc", i.e., with the results obtained by applying AProVE's implementation of the technique from Sect. 4 and 5 and analyzing the resulting TRS with TcT afterwards. Here, we used a timeout of 60 s for TcT *after* preprocessing the TRS with AProVE. "TcT preproc" can prove upper bounds in 299 cases, while AProVE only succeeds in 270 cases. Hence, combining TcT with the technique of the current paper results in the most powerful tool for upper bounds on rc. However, the results of the tools are orthogonal: There are 20 examples where AProVE obtains better bounds and 53 examples where "TcT preproc" is better.

		AProVE					
	$\mathrm{rc}_{\mathcal{R}}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^5)$	$>$ *poly*
TcT preproc	$\mathcal{O}(1)$	39	1	–	–	–	–
	$\mathcal{O}(n)$	5	174	12	–	–	28
	$\mathcal{O}(n^2)$	–	5	18	2	–	3
	$\mathcal{O}(n^3)$	–	–	1	4	–	6
	$\mathcal{O}(n^5)$	–	–	–	–	–	1
	$>$ *poly*	2	3	4	–	–	356

Table 2: TcT preproc vs. AProVE

Table 3 compares the results of all the different settings used to prove upper bounds with AProVE and TcT. Moreover, the "union" of AProVE and "TcT preproc" is presented separately ("AProVE & TcT"). Here, we used the best bound obtained by AProVE or "TcT preproc" for each example. The entries in the row "$\mathcal{O}(n^k)$ or less" of Table 3 mean that the corresponding tool proved at least the upper bound $\mathcal{O}(n^k)$, but maybe even a smaller upper bound (so the entry 229 in the second row, second column, means that AProVE proved constant or linear upper bounds in 229 cases). At the Termination Competition 2015, TcT was the only tool for upper bounds on rc and hence it represents the former state of the art for this task. Thus, the setting "AProVE & TcT" shows how the state of the art has improved by the technique presented in this paper. Compared to TcT, "AProVE & TcT" proves 99 additional upper bounds.

$\mathrm{rc}_{\mathcal{R}}(n)$	TcT	AProVE	TcT preproc	AProVE & TcT
$\mathcal{O}(1)$	22	46	40	47
$\mathcal{O}(n)$ or less	183	229	259	269
$\mathcal{O}(n^2)$ or less	204	264	287	297
$\mathcal{O}(n^3)$ or less	208	270	298	307
$\mathcal{O}(n^5)$ or less	209	270	299	308

Table 3: comparing different settings for upper bounds

Finally, Table 4 compares "AProVE & TcT" with the lower bounds proved by AProVE. To infer the lower bounds, we used a timeout of 300 s. The reason for the larger timeout is that "AProVE lower" does not "compete" with the upper bounds proved by AProVE and TcT. In contrast, better lower bounds can emphasize the quality of the obtained upper bounds by showing that these upper bounds are optimal (if the lower and upper bounds coincide). This allows us to estimate the quality of the inferred upper bounds. Indeed, the upper bounds are tight in all but 35 cases. In 33 of these cases, the lower and upper bounds just differ by a factor of n. This comparison clearly shows that the quality of the bounds found by AProVE and TcT is very good.

		AProVE lower					
	$\mathrm{rc}_{\mathcal{R}}(n)$	$\Omega(1)$	$\Omega(n)$	$\Omega(n^2)$	$\Omega(n^3)$	$\Omega(n^{>3})$	$>$ *poly*
AProVE & TcT	$\mathcal{O}(1)$	47	–	–	–	–	–
	$\mathcal{O}(n)$	10	212	–	–	–	–
	$\mathcal{O}(n^2)$	–	21	7	–	–	–
	$\mathcal{O}(n^3)$	–	1	2	7	–	–
	$\mathcal{O}(n^{>3})$	–	1	–	–	–	–
	$>$ *poly*	6	298	47	4	1	235

Table 4: AProVE & TcT vs. AProVE lower

See [1] for further information about the evaluation. Moreover, [1] offers a custom web-interface which can be used to access our implementation of the presented technique.

Acknowledgments. We are grateful to the reviewers for their suggestions and comments.

References

[1] AProVE: https://aprove-developers.github.io/complexityFullRewriting/.

[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. Dependency pairs and polynomial path orders. In *Proc. RTA '09*, LNCS 5595, pages 48–62, 2009.

[4] M. Avanzini and G. Moser. Complexity analysis by graph rewriting. In *Proc. FLOPS '10*, LNCS 6009, pages 257–271, 2010.

[5] M. Avanzini and G. Moser. Polynomial path orders. *Logical Methods in Computer Science*, 9(4), 2013.

[6] M. Avanzini and G. Moser. A combination framework for complexity. *Information and Computation*, 248:22–55, 2016.

[7] M. Avanzini, G. Moser, and M. Schaper. TcT: Tyrolean complexity tool. In *Proc. TACAS '16*, LNCS 9636, pages 407–423, 2016.

[8] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[9] G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *Journal of Functional Programming*, 11(1):33–53, 2001.

[10] F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder. Lower bounds for runtime complexity of term rewriting. *Journal of Automated Reasoning*, 2017. To appear. DOI: 10.1007/s10817-016-9397-x.

[11] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

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

[13] B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24(1/2):2–23, 1995.

[14] N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR '08*, LNAI 5195, pages 364–379, 2008.

[15] N. Hirokawa, A. Middeldorp, and H. Zankl. Uncurrying for innermost termination and derivational complexity. *Journal of Automated Reasoning*, 50(3):279–315, 2013.

[16] D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. RTA '89*, LNCS 355, pages 167–177, 1989.

[17] M. Hofmann and G. Moser. Amortised resource analysis and typed polynomial interpretations. In *Proc. RTA-TLCA '14*, LNCS 8560, pages 272–286, 2014.

[18] C. Kop, A. Middeldorp, and T. Sternagel. Complexity of conditional term rewriting. *Logical Methods in Computer Science*, 13(1), 2017.

[19] A. Middeldorp, G. Moser, F. Neurauter, J. Waldmann, and H. Zankl. Joint spectral radius theory for automated complexity analysis of rewrite systems. In *Proc. CAI '11*, LNCS 6742, pages 1–20, 2011.

[20] G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3), 2011.

[21] 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.

[22] M. Sakai, K. Okamoto, and T. Sakabe. Innermost reductions find all normal forms on right-linear terminating overlay TRSs. In *WRS '03*, 2003.

[23] Termination Competition: http://termination-portal.org/wiki/Termination_Competition.

[24] J. van de Pol and H. Zantema. Generalized innermost rewriting. In *Proc. RTA '05*, LNCS 3467, pages 2–16, 2005.

[25] H. Zankl and M. Korp. Modular complexity analysis for term rewriting. *Logical Methods in Computer Science*, 10(1), 2014.

A Remaining Proofs

Lemma 13 (Right-Linear TRSs are Spare)**.** *Every right-linear TRS is spare. Hence, every right-linear overlay system is ndg.*

Proof. This is an immediate consequence of Def. 6 and Cor. 8 (b). ☐

Theorem 22 (Duplicating Contexts are Undecidable)**.** *It is undecidable if a context is duplicating.*

Proof. Let $\mathcal{R}$ be a left-linear basic TRS. We adapt the construction from the proof of Thm. 12 by introducing an additional argument for h and by modifying the g- and h-rules to

$$\mathsf{g} \to \mathsf{h}(\mathsf{inf}, \mathsf{inf}, f(t_1, \dots, t_k)) \qquad\qquad \mathsf{h}(x, y, \mathsf{a}) \to \mathsf{c}(x, x)$$

Then the context $\mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})$ is duplicating w.r.t. $\mathcal{R}'$ iff $\mathsf{f}(t_1, \dots, t_k)$ is normalizing w.r.t. $\mathcal{R}$. To see this, we consider all rewrite sequences starting with basic terms. Clearly, there is no term q such that $\mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})[q]$ is reachable from a basic term whose root is from Σ. Hence, we now regard all basic terms with root g, h, or inf.

- For the basic term g we have $\mathsf{g} \to \mathsf{h}(\mathsf{inf}, \mathsf{inf}, f(t_1, \dots, t_k))$. If $f(t_1, \dots, t_k)$ is not normalizing, then by construction, the rewrite sequence can never reach a term containing $\mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})[q]$ for any term q. If $f(t_1, \dots, t_k)$ is normalizing, then let t be a normal form of $f(t_1, \dots, t_k)$. Since $\mathcal{R}$ is a basic TRS, t is again a basic term. Hence, we get $\mathsf{h}(\mathsf{inf}, \mathsf{inf}, f(t_1, \dots, t_k)) \to^* \mathsf{h}(\mathsf{inf}, \mathsf{inf}, t) \to \mathsf{h}(\mathsf{inf}, \mathsf{inf}, \mathsf{a}) = \mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})[\mathsf{inf}]$, i.e., $\mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})$ is duplicating.

- Each basic term of the form $\mathsf{h}(s, s', s'')$, is either a normal form (if $s'' \neq \mathsf{a}$) or just enables the rewrite step $\mathsf{h}(s, s', \mathsf{a}) \to \mathsf{c}(s, s)$. Hence, $\mathsf{h}(\Box, s', \mathsf{a})$ is duplicating. However, as $\mathsf{h}(s, s', \mathsf{a})$ is basic, we have $s' \neq \mathsf{inf}$ (i.e., this does not mean that $\mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})$ is duplicating). Since s does not contain defined symbols, $\mathsf{c}(s, s)$ is a normal form, i.e., no further terms with root h are reachable from $\mathsf{h}(s, s', \mathsf{a})$.

- If we start with the basic term inf, the only possible reduction is $\mathsf{inf} \to \mathsf{inf} \to \dots$

Hence, $\mathsf{h}(\Box, \mathsf{inf}, \mathsf{a})$ is duplicating iff $f(t_1, \dots, t_k)$ is normalizing. Since normalization of basic ground terms w.r.t. left-linear basic TRSs is undecidable (cf. the proof of Thm. 12), this implies that it is also undecidable whether a context is duplicating. ☐

Theorem 28 (No Defined and Duplicating Context $\Longrightarrow$ Spare)**.** *If no defined context is duplicating, then $\mathcal{R}$ is spare. If $\mathcal{R}$ is left-linear and spare, then no defined context is duplicating.*

Proof. In Sect. 4 we already proved that if $\mathcal{R}$ is not spare, then there is a context that is both defined and duplicating. Now assume that $\mathcal{R}$ is left-linear and there is a context C which is defined and duplicating. Then there is a rewrite sequence $s \to^* t \trianglerighteq C[p]$ where s is basic and p is a redex by the definition of defined contexts. Moreover, there is a rule $\ell \to r$, a variable x with $\#_x(r) > 1$, a substitution σ, and positions π and τ such that $C = \ell\sigma[\Box]_{\pi.\tau}$ and $\ell|_\pi = x \in \mathcal{V}$ by the definition of duplicating contexts. As ℓ is linear, this implies $C[p] = \ell\theta$ where θ is the substitution with $x\theta = x\sigma[p]_\tau$ and $y\theta = y\sigma$ for all variables $y \neq x$. Hence we get $s \to^* t \trianglerighteq C[p] = \ell\theta \to_{\ell\to r} r\theta$ where $x\theta$ is not a normal form, i.e., where the last rewrite step is not spare. Thus, $\mathcal{R}$ is not spare. ☐

Theorem 30 (Defined Contexts are Undecidable)**.** *It is undecidable if a context is defined.*

Proof. We use the same construction as in the proof of Thm. 12. Here, the context $\mathsf{h}(\Box, \mathsf{a})$ is defined w.r.t. $\mathcal{R}'$ iff $f(t_1, \dots, t_k)$ is normalizing w.r.t. $\mathcal{R}$. ☐

Lemma 37 (Finiteness of Def)**.** *For each TRS $\mathcal{R}$, $Def_{\mathcal{R}}$ is finite.*

Proof. We have $Def_{\mathcal{R}} \subseteq \{\lfloor C \rfloor \mid \ell \to r \in \mathcal{R}, r \trianglerighteq C[p]\}$, which is finite. $\square$

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

http://aib.informatik.rwth-aachen.de/

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2013-01 * Fachgruppe Informatik: Annual Report 2013
2013-02 Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssysteme in Klein- und mittelständischen Unternehmen
2013-03 Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM
2013-04 Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries
2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013
2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation
2013-07 André Stollenwerk: Ein modellbasiertes Sicherheitskonzept für die extrakorporale Lungenunterstützung
2013-08 Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika Ábrahám: On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers
2013-10 Joost-Pieter Katoen, Thomas Noll, Thomas Santen, Dirk Seifert, and Hao Wu: Performance Analysis of Computing Servers using Stochastic Petri Nets and Markov Automata
2013-12 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl: Alternating Runtime and Size Complexity Analysis of Integer Programs
2013-13 Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle: SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators
2013-14 Jörg Brauer: Automatic Abstraction for Bit-Vectors using Decision Procedures
2013-16 Carsten Otto: Java Program Analysis by Symbolic Execution
2013-19 Florian Schmidt, David Orlea, and Klaus Wehrle: Support for error tolerance in the Real-Time Transport Protocol

2013-20 Jacob Palczynski: Time-Continuous Behaviour Comparison Based on Abstract Models

2014-01 * Fachgruppe Informatik: Annual Report 2014

2014-02 Daniel Merschen: Integration und Analyse von Artefakten in der modellbasierten Entwicklung eingebetteter Software

2014-03 Uwe Naumann, Klaus Leppkes, and Johannes Lotz: dco/c++ User Guide

2014-04 Namit Chaturvedi: Languages of Infinite Traces and Deterministic Asynchronous Automata

2014-05 Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp: Automated Termination Analysis for Programs with Pointer Arithmetic

2014-06 Esther Horbert, Germán Martín García, Simone Frintrop, and Bastian Leibe: Sequence Level Salient Object Proposals for Generic Object Detection in Video

2014-07 Niloofar Safiran, Johannes Lotz, and Uwe Naumann: Algorithmic Differentiation of Numerical Methods: Second-Order Tangent and Adjoint Solvers for Systems of Parametrized Nonlinear Equations

2014-08 Christina Jansen, Florian Göbe, and Thomas Noll: Generating Inductive Predicates for Symbolic Execution of Pointer-Manipulating Programs

2014-09 Thomas Ströder and Terrance Swift (Editors): Proceedings of the International Joint Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments 2014

2014-14 Florian Schmidt, Matteo Ceriotti, Niklas Hauser, and Klaus Wehrle: HotBox: Testing Temperature Effects in Sensor Networks

2014-15 Dominique Gückel: Synthesis of State Space Generators for Model Checking Microcontroller Code

2014-16 Hongfei Fu: Verifying Probabilistic Systems: New Algorithms and Complexity Results

2015-01 * Fachgruppe Informatik: Annual Report 2015

2015-02 Dominik Franke: Testing Life Cycle-related Properties of Mobile Applications

2015-05 Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder: Inferring Lower Bounds for Runtime Complexity

2015-06 Thomas Ströder and Wolfgang Thomas (Editors): Proceedings of the Young Researchers' Conference "Frontiers of Formal Methods"

2015-07 Hilal Diab: Experimental Validation and Mathematical Analysis of Cooperative Vehicles in a Platoon

2015-08 Mathias Pelka, Jó Agila Bitsch, Horst Hellbrück, and Klaus Wehrle (Editors): Proceedings of the 1st KuVS Expert Talk on Localization

2015-09 Xin Chen: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models

2015-11 Stefan Wüller, Marián Kühnel, and Ulrike Meyer: Information Hiding in the Public RSA Modulus

2015-12 Christoph Matheja, Christina Jansen, and Thomas Noll: Tree-like Grammars and Separation Logic

2015-13 Andreas Polzer: Ansatz zur variantenreichen und modellbasierten Entwicklung von eingebetteten Systemen unter Berücksichtigung regelungs- und softwaretechnischer Anforderungen

2015-14 Niloofar Safiran and Uwe Naumann: Symbolic vs. Algorithmic Differentiation of GSL Integration Routines

2016-01 * Fachgruppe Informatik: Annual Report 2016

2016-02 Ibtissem Ben Makhlouf: Comparative Evaluation and Improvement of Computational Approaches to Reachability Analysis of Linear Hybrid Systems

2016-03 Florian Frohn, Matthias Naaf, Jera Hensel, Marc Brockschmidt, and Jürgen Giesl: Lower Runtime Bounds for Integer Programs

2016-04 Jera Hensel, Jürgen Giesl, Florian Frohn, and Thomas Ströder: Proving Termination of Programs with Bitvector Arithmetic by Symbolic Execution

2016-05 Mathias Pelka, Grigori Goronzy, Jó Agila Bitsch, Horst Hellbrück, and Klaus Wehrle (Editors): Proceedings of the 2nd KuVS Expert Talk on Localization

2016-06 Martin Henze, René Hummen, Roman Matzutt, Klaus Wehrle: The SensorCloud Protocol: Securely Outsourcing Sensor Data to the Cloud

2016-07 Sebastian Biallas: Verification of Programmable Logic Controller Code using Model Checking and Static Analysis

2016-09 Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, Peter Schneider-Kamp, and Cornelius Aschermann: Automatically Proving Termination and Memory Safety for Programs with Pointer Arithmetic

2016-10 Stefan Wüller, Ulrike Meyer, and Susanne Wetzel: Towards Privacy-Preserving Multi-Party Bartering

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.