Noname manuscript No.
(will be inserted by the editor)

Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs

Lars Noschinski · Fabian Emmes · Jürgen Giesl

the date of receipt and acceptance should be inserted later

Abstract We present a modular framework to analyze the innermost runtime complexity of term rewrite systems automatically. Our method is based on the dependency pair framework for termination analysis. In contrast to previous work, we developed a *direct* adaptation of successful termination techniques from the dependency pair framework in order to use them for complexity analysis. By extensive experimental results, we demonstrate the power of our method compared to existing techniques.

1 Introduction

In practice, one is often not only interested in analyzing the termination of programs, but one also wants to check whether algorithms terminate in *reasonable* (e.g., polynomial) *time*. While termination of term rewrite systems (TRSs) is well studied, only recently first results were obtained which adapt termination techniques in order to obtain polynomial complexity bounds automatically, e.g., [2–5,7,10,17–19,22–24,26,29,30]. Here, [3,17–19] consider the *dependency pair (DP) method* [1,11,12,16], which is one of the most popular termination techniques for TRSs. Moreover, [30] introduces a similar modular approach for complexity analysis based on relative rewriting. There is also a related area of *implicit computational complexity* which aims at characterizing complexity classes, e.g., using type systems [21], bottom-up logic programs [15], and also using termination techniques like dependency pairs (e.g., [23]).

Techniques for automated termination analysis of term rewriting are very powerful and have been successfully used to analyze termination of programs in many different languages (e.g., Java [8, 28], Haskell [13], and Prolog [14]). Hence, by adapting these termination techniques, the ultimate goal is to obtain approaches which can also analyze the complexity of programs automatically.

Supported by the DFG grant GI 274/5-3.

Lars Noschinski
Institut für Informatik, TU Munich, Germany

Fabian Emmes · Jürgen Giesl
LuFG Informatik 2, RWTH Aachen University, Germany

In this paper, we present a fresh adaptation of the DP framework for *innermost runtime complexity analysis* [17]. We use a different notion of "dependency pairs" for complexity analysis than previous works [3, 17, 19]. This allows us to adapt the termination techniques ("processors") of the DP framework in a much more direct way when using them for complexity analysis. On the other hand, our approach is restricted to the *innermost* evaluation strategy, whereas [3, 17, 19] also analyze runtime complexity of full rewriting. Like [30], our method is modular (i.e., we can determine the complexity of a TRS by determining the complexity of certain sub-problems and by returning the maximum of these complexities). But in contrast to [30], which allows to investigate *derivational complexity* [20], we focus on innermost runtime complexity. In this way, we can inherit the modularity aspects of the DP framework and benefit from its transformation techniques, which increases power significantly.

A preliminary version of this paper appeared in [27]. The current paper extends [27] substantially, e.g., by including proofs for all theorems, by two new processors for complexity analysis (Thm. 32 and 34) and experiments to justify their significance, by detailed information on the impact of the different processors in Sect. 6, and by several additional remarks throughout the paper.

After introducing preliminaries in Sect. 2, in Sect. 3 we adapt the concept of *dependency pairs* from termination analysis to so-called *dependency tuples* for complexity analysis. While the *DP framework* for termination works on *DP problems*, we now work on *DT problems* (Sect. 4). Sect. 5 adapts the processors of the DP framework in order to analyze the complexity of DT problems. We implemented our contributions in the termination analyzer AProVE. Due to the results of this paper, AProVE was the most powerful tool for innermost runtime complexity analysis in the *International Termination Competition* in 2010 – 2012. In Sect. 6 we report on extensive experiments where we compared our technique empirically with previous approaches.

2 Runtime Complexity of Term Rewriting

See e.g. [6] for the basics of term rewriting, where we only consider finite rewrite systems. Let $\mathcal{T}(\Sigma,\mathcal{V})$ be the set of all terms over a (finite) signature Σ and a set of variables $\mathcal{V}$. We just write $\mathcal{T}$ if Σ and $\mathcal{V}$ are clear from the context. The *arity* of a function symbol $f \in \Sigma$ is denoted by $\mathrm{ar}(f)$ and the size of a term is $|x| = 1$ for $x \in \mathcal{V}$ and $|f(t_1,\ldots,t_n)| = 1 + |t_1| + \ldots + |t_n|$. The *derivation height* of a term t w.r.t. a relation $\to$ is the length of the longest sequence of $\to$-steps starting with t, i.e., $\mathrm{dh}(t,\to) = \sup\{\, n \mid \exists t' \in \mathcal{T}, t \to^n t' \,\}$, cf. [20].

Here, for any set $M \subseteq \mathbb{N} \cup \{\omega\}$, "$\sup M$" is the least upper bound of M and $\sup \varnothing = 0$. Note that since we restricted ourselves to finite TRSs, the rewrite relation is finitely branching and thus, the set $\{\, n \mid \exists t' \in \mathcal{T}, t \to^n t' \,\}$ is infinite iff t starts an infinitely long sequence of $\to$-steps. Thus, $\mathrm{dh}(t,\to) = \omega$ iff t is non-terminating w.r.t. the relation $\to$ (i.e., iff there is an infinite reduction $t \to t_1 \to t_2 \to \ldots$). Note that in the literature, dh is usually left undefined for terms with non-terminating derivations. However, we extended it in order to treat terminating and non-terminating terms in a uniform way.

Example 1 As an example, consider the TRS $\mathcal{R} = \{\mathsf{dbl}(0) \to 0,\ \mathsf{dbl}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{dbl}(x)))\}$. *Then* $\mathrm{dh}(\mathsf{dbl}(\mathsf{s}^n(0)), \to_{\mathcal{R}}) = n+1$, *but* $\mathrm{dh}(\mathsf{dbl}^n(\mathsf{s}(0)), \to_{\mathcal{R}}) = 2^n + n - 1$.

For a TRS $\mathcal{R}$ with *defined symbols* $\Sigma_d(\mathcal{R}) = \{\, \mathrm{root}(\ell) \mid \ell \to r \in \mathcal{R} \,\}$, a term $f(t_1,\ldots,t_n)$ is *basic* if $f \in \Sigma_d(\mathcal{R})$ and $t_1,\ldots,t_n$ do not contain symbols from $\Sigma_d(\mathcal{R})$. If $\mathcal{R}$ is clear from the context, we just write Σ_d instead of $\Sigma_d(\mathcal{R})$. So for $\mathcal{R}$ in Ex. 1, the basic terms are $\mathsf{dbl}(\mathsf{s}^n(0))$

and $\mathsf{dbl}(\mathsf{s}^n(x))$ for $n \in \mathbb{N}$, $x \in \mathcal{V}$. The *innermost runtime complexity function* $\mathrm{irc}_{\mathcal{R}}$ maps any $n \in \mathbb{N}$ to the length of a longest sequence of $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$-steps starting with a basic term t with $|t| \leq n$. Here, "$\xrightarrow{\mathrm{i}}_{\mathcal{R}}$" is the innermost rewrite relation and $\mathcal{T}_B$ is the set of all basic terms.

Definition 2 **($\mathrm{irc}_{\mathcal{R}}$ [17])** For a TRS $\mathcal{R}$, its *innermost runtime complexity function* $\mathrm{irc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ is $\mathrm{irc}_{\mathcal{R}}(n) = \sup\{\, \mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}_B, |t| \leq n \,\}$.

If one only considers evaluations of basic terms, the (runtime) complexity of the TRS $\mathcal{R}$ in Ex. 1 is linear ($\mathrm{irc}_{\mathcal{R}}(n) = n - 1$ for $n \geq 2$). But if one also permits evaluations starting with $\mathsf{dbl}^n(\mathsf{s}(\mathsf{0}))$, the complexity of the dbl-TRS is exponential.

When analyzing the complexity of *programs*, one is typically interested in evaluations where a defined function like dbl is applied to data objects (i.e., terms without defined symbols). Therefore, *runtime complexity* corresponds to the usual notion of "complexity" for programs [4, 5]. Note that most termination techniques which transform programs to TRSs (e.g., [8, 13, 14, 28]) result in rewrite systems where one only regards innermost rewrite sequences. This also holds for techniques to analyze termination of languages like Haskell by term rewriting [13], although Haskell has a lazy (outermost) evaluation strategy. Thus, we restrict ourselves to innermost rewriting, since this suffices to analyze TRSs resulting from the transformation of programs. So for any TRS $\mathcal{R}$, we want to determine the *asymptotic complexity* of the function $\mathrm{irc}_{\mathcal{R}}$.

Definition 3 **(Asymptotic Complexities)** Let $\mathfrak{C} = \{Pol_0, Pol_1, Pol_2, \ldots, ?\}$ with the order $Pol_0 \sqsubset Pol_1 \sqsubset Pol_2 \sqsubset \ldots \sqsubset ?$. Let $\sqsubseteq$ be the reflexive closure of $\sqsubset$. For any function $f : \mathbb{N} \to \mathbb{N} \cup \{\omega\}$ we define its *complexity* $\iota(f) \in \mathfrak{C}$ as follows: $\iota(f) = Pol_k$ if k is the smallest number with $f(n) \in \mathrm{O}(n^k)$ and $\iota(f) = ?$ if there is no such k. For any TRS $\mathcal{R}$, we define its *complexity* $\iota_{\mathcal{R}}$ as $\iota(\mathrm{irc}_{\mathcal{R}})$.

So the TRS $\mathcal{R}$ in Ex. 1 has linear complexity, i.e., $\iota_{\mathcal{R}} = Pol_1$. As another example, consider the following TRS $\mathcal{R}$ where "m" stands for "minus".

Example 4

$$\begin{array}{lll}
\mathsf{m}(x,y) \to \mathsf{if}(\mathsf{gt}(x,y),x,y) & \mathsf{gt}(\mathsf{0},k) \to \mathsf{false} & \mathsf{p}(\mathsf{0}) \to \mathsf{0} \\
\mathsf{if}(\mathsf{true},x,y) \to \mathsf{s}(\mathsf{m}(\mathsf{p}(x),y)) & \mathsf{gt}(\mathsf{s}(n),\mathsf{0}) \to \mathsf{true} & \mathsf{p}(\mathsf{s}(n)) \to n \\
\mathsf{if}(\mathsf{false},x,y) \to \mathsf{0} & \mathsf{gt}(\mathsf{s}(n),\mathsf{s}(k)) \to \mathsf{gt}(n,k) &
\end{array}$$

Here, $\iota_{\mathcal{R}} = Pol_2$ (e.g., $\mathsf{m}(\mathsf{s}^n(\mathsf{0}),\mathsf{s}^k(\mathsf{0}))$ starts evaluations of quadratic length).

In Def. 3, we restricted ourselves to polynomial complexity classes, because the underlying techniques that we use to generate suitable well-founded orders automatically result in polynomial bounds. However, the approach could also be used for other complexity classes (then the order $\sqsubset$ would have to be extended accordingly).

3 Dependency Tuples

In the DP method, for every $f \in \Sigma_d$ one introduces a fresh symbol $f^\sharp$ with $\mathrm{ar}(f) = \mathrm{ar}(f^\sharp)$. For a term $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma_d$ we define $t^\sharp = f^\sharp(t_1, \ldots, t_n)$ and let $\mathcal{T}^\sharp = \{\, t^\sharp \mid t \in \mathcal{T}, \mathrm{root}(t) \in \Sigma_d \,\}$. Let $Pos(t)$ contain all positions of t and let $Pos_d(t) = \{\, \pi \mid \pi \in Pos(t), \mathrm{root}(t|_\pi) \in \Sigma_d \,\}$. Then for every rule $\ell \to r$ with $Pos_d(r) = \{\pi_1, \ldots, \pi_n\}$, its *dependency pairs* are $\ell^\sharp \to r|_{\pi_1}^\sharp, \ldots, \ell^\sharp \to r|_{\pi_n}^\sharp$.

While DPs are used for termination, for complexity we have to regard *all* defined functions in a right-hand side *at once*. The reason is that in order to estimate the derivation height of a term corresponding to the left-hand side of a rule $\ell \to r$, we have to consider the rewrite steps originating from r. Here, all subterms of r with defined root symbol may possibly be reduced and can thus contribute to the overall derivation height. Thus, we extend the concept of *weak dependency pairs* [17–19] and only build a single *dependency tuple* $\ell^\sharp \to [r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_n}]$ for each $\ell \to r$. To avoid an extra treatment of tuples, for every $n \geq 0$, we introduce a fresh *compound symbol* $\textsc{Com}_n$ of arity n and use $\ell^\sharp \to \textsc{Com}_n(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_n})$.

Definition 5 (Dependency Tuple) A *dependency tuple* is a rule of the form $s^\sharp \to \textsc{Com}_n(t_1^\sharp, \ldots, t_n^\sharp)$ for $s^\sharp, t_1^\sharp, \ldots, t_n^\sharp \in \mathcal{T}^\sharp$. Let $\ell \to r$ be a rule with $Pos_d(r) = \{\pi_1, \ldots, \pi_n\}$. Then $DT(\ell \to r)$ is defined to be $\ell^\sharp \to \textsc{Com}_n(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_n})$. To make $DT(\ell \to r)$ unique, we use a total order $<$ on positions where $\pi_1 < \ldots < \pi_n$. For a TRS $\mathcal{R}$, let $DT(\mathcal{R}) = \{DT(\ell \to r) \mid \ell \to r \in \mathcal{R}\}$.

Example 6 For the TRS $\mathcal{R}$ from Ex. 4, $DT(\mathcal{R})$ is the following set of rules.

$$\mathsf{m}^\sharp(x,y) \to \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y), \mathsf{gt}^\sharp(x,y)) \quad (1) \qquad \mathsf{p}^\sharp(0) \to \textsc{Com}_0 \quad (4)$$

$$\mathsf{if}^\sharp(\mathsf{true},x,y) \to \textsc{Com}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y), \mathsf{p}^\sharp(x)) \quad (2) \qquad \mathsf{p}^\sharp(\mathsf{s}(n)) \to \textsc{Com}_0 \quad (5)$$

$$\mathsf{if}^\sharp(\mathsf{false},x,y) \to \textsc{Com}_0 \quad (3) \qquad \mathsf{gt}^\sharp(0,k) \to \textsc{Com}_0 \quad (6)$$

$$\mathsf{gt}^\sharp(\mathsf{s}(n),0) \to \textsc{Com}_0 \quad (7)$$

$$\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \textsc{Com}_1(\mathsf{gt}^\sharp(n,k)) \quad (8)$$

For termination, one analyzes *chains* of DPs, which correspond to sequences of function calls that can occur in reductions. Since DTs represent *several* DPs, we now obtain *chain trees*.

Definition 7 (Chain Tree) Let $\mathcal{D}$ be a set of DTs and $\mathcal{R}$ be a TRS. Let T be a (possibly infinite) tree whose nodes are labeled with both a DT from $\mathcal{D}$ and a substitution. Let the root node be labeled with $(s^\sharp \to \textsc{Com}_n(\ldots) \mid \sigma)$. Then T is a *$(\mathcal{D},\mathcal{R})$-chain tree for $s^\sharp\sigma$* if the following holds for all nodes of T: If a node is labeled with $(u^\sharp \to \textsc{Com}_m(v_1^\sharp, \ldots, v_m^\sharp) \mid \mu)$, then $u^\sharp\mu$ is in normal form w.r.t. $\mathcal{R}$. Moreover, if this node has the children $(p_1^\sharp \to \textsc{Com}_{m_1}(\ldots) \mid \tau_1), \ldots, (p_k^\sharp \to \textsc{Com}_{m_k}(\ldots) \mid \tau_k)$, then there are pairwise different[1] $i_1, \ldots, i_k \in \{1, \ldots, m\}$ with $v_{i_j}^\sharp\mu \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} p_j^\sharp\tau_j$ for all $j \in \{1, \ldots, k\}$. A path in the chain tree is called a *chain*.

Example 8 For the TRS $\mathcal{R}$ from Ex. 4 and its DTs from Ex. 6, the tree in Fig. 1 is a $(DT(\mathcal{R}),\mathcal{R})$-chain tree for $\mathsf{m}^\sharp(\mathsf{s}(0),0)$. Here, we use substitutions with $\sigma(x) = \mathsf{s}(0)$ and $\sigma(y) = 0$, $\tau(x) = \tau(y) = 0$, and $\mu(n) = \mu(k) = 0$.

Note that the *chains* in Def. 7 correspond to "*innermost chains*" in the DP framework [1, 11, 12]. When considering *full* instead of innermost rewriting, the DP framework uses a different notion of chains where, e.g., $u^\sharp\mu$ would not have to be in normal form. However, in contrast to other techniques for complexity analysis with dependency pairs [3, 17–19], our approach is inherently restricted to the innermost rewrite strategy.

For any term $s^\sharp \in \mathcal{T}^\sharp$, we now define its *complexity* as the maximal number of nodes in any chain tree for $s^\sharp$. However, sometimes we do not want to count *all* DTs in the chain tree, but only the DTs from some subset $\mathcal{S}$. This will be crucial to adapt termination techniques for complexity, cf. Sect. 5.2 and 5.4.

[1] One could also allow chain trees where $i_1, \ldots, i_k$ do not have to be pairwise different. But Thm. 12 shows that for complexity analysis, it is already sufficient to consider just chain trees with pairwise different $i_1, \ldots, i_k$.

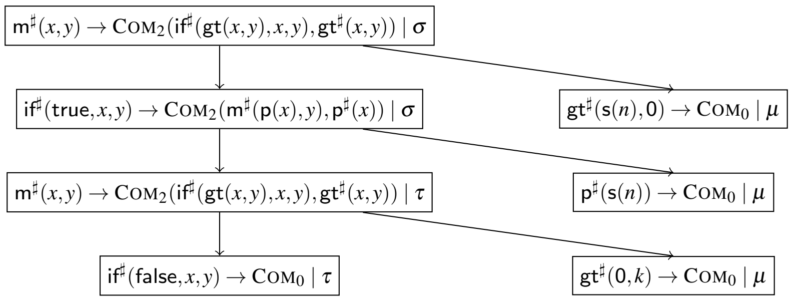

m♯(x,y) → Com₂(if♯(gt(x,y),x,y), gt♯(x,y)) | σ
if♯(true,x,y) → Com₂(m♯(p(x),y), p♯(x)) | σ
gt♯(s(n),0) → Com₀ | μ
m♯(x,y) → Com₂(if♯(gt(x,y),x,y), gt♯(x,y)) | τ
p♯(s(n)) → Com₀ | μ
if♯(false,x,y) → Com₀ | τ
gt♯(0,k) → Com₀ | μ

Fig. 1 Chain tree for the TRS from Ex. 4

Definition 9 (Complexity of Terms, $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}$, $|T|_{\mathcal{S}}$) Let $\mathcal{D}$ be a set of dependency tuples, $\mathcal{S} \subseteq \mathcal{D}$, $\mathcal{R}$ a TRS, and $s^\sharp \in \mathcal{T}^\sharp$. For a chain tree T, $|T|_{\mathcal{S}} \in \mathbb{N} \cup \{\omega\}$ denotes the number of nodes in T that are labeled with a DT from $\mathcal{S}$. We define $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp) = \sup\{|T|_{\mathcal{S}} \mid T$ is a $(\mathcal{D},\mathcal{R})$-chain tree for $s^\sharp\}$. In other words, $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp)$ is the maximal number of nodes from $\mathcal{S}$ occurring in any $(\mathcal{D},\mathcal{R})$-chain tree for $s^\sharp$. For terms $s^\sharp$ without a $(\mathcal{D},\mathcal{R})$-chain tree, we define $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp) = 0$.

Example 10 *For $\mathcal{R}$ from Ex. 4, we have $Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(\mathsf{m}^\sharp(\mathsf{s}(0),0)) = 7$, since the maximal tree for $\mathsf{m}^\sharp(\mathsf{s}(0),0)$ in Fig. 1 has 7 nodes. In contrast, if $\mathcal{S}$ is $DT(\mathcal{R})$ without the $\mathsf{gt}^\sharp$-DTs (6) – (8), then $Cplx_{\langle DT(\mathcal{R}),\mathcal{S},\mathcal{R}\rangle}(\mathsf{m}^\sharp(\mathsf{s}(0),0)) = 5$.*

Dependency tuples can be used to approximate the derivation heights of terms. The reason is that every actual reduction corresponds to a chain tree. However, the converse does not hold, i.e., there exist chain trees that do not correspond to an actual reduction.

Example 11 *To see this, consider the non-confluent TRS $\mathcal{R}$*

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{g}(x)) \quad (9) \qquad \mathsf{g}(x) \to x \quad (10) \qquad \mathsf{g}(x) \to \mathsf{a}(\mathsf{f}(x)) \quad (11)$$

with the DTs

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_2(\mathsf{f}^\sharp(\mathsf{g}(x)), \mathsf{g}^\sharp(x)) \quad (12) \qquad \mathsf{g}^\sharp(x) \to \textsc{Com}_1(\mathsf{f}^\sharp(x)) \quad (13)$$

Chain trees do not take into account that the subterms $\mathsf{g}(x)$ and $\mathsf{g}^\sharp(x)$ in the right-hand side of (12) have to be evaluated in the same way. Thus, for the substitution σ with $\sigma(x) = \mathsf{s}(x)$, there is a chain tree with the root $((12) \mid \sigma)$ and the children $((12) \mid id)$ and $((13) \mid \sigma)$, where id is the identical substitution. Here the step from $((12) \mid \sigma)$ to $((12) \mid id)$ corresponds to a reduction of the subterm $g(s(x))$ with rule (10), whereas the step from $((12) \mid \sigma)$ to $((13) \mid \sigma)$ corresponds to a reduction of $g(s(x))$ with rule (11). Note that $((13) \mid \sigma)$ then again has a child $((12) \mid id)$. Thus, for this example the size of chain trees can be exponential (i.e., we have $Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(\mathsf{f}^\sharp(\mathsf{s}^n(0))) = 2^{n+1} - 2$), although the runtime complexity of $\mathcal{R}$ is linear.

Thm. 12 proves that $Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$ is indeed an upper bound for t's derivation height w.r.t. $\xrightarrow{i}_{\mathcal{R}}$, provided that t is in argument normal form. Here, a term $t = f(t_1,\ldots,t_n)$ is in *argument normal form* iff all t_i are normal forms w.r.t. $\mathcal{R}$. Thus, all basic terms are in argument normal form, but in addition, a term $f(t_1,\ldots,t_n)$ in argument normal form may also have defined symbols in the t_i, as long as these subterms cannot be reduced further. This generalized form of basic terms is needed for the proof of Thm. 12.

Theorem 12 **(*Cplx* bounds Derivation Height)** *Let $\mathcal{R}$ be a TRS. Let $t = f(t_1,\ldots,t_n) \in \mathcal{T}$ be in argument normal form. Then we have* $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$. *If $\mathcal{R}$ is confluent, we have* $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$.

Proof We first consider the case where $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \omega$. As t is in argument normal form, there is a $\ell_1 \to r_1 \in \mathcal{R}$ and a substitution σ_1 such that $t = \ell_1\sigma_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}} r_1\sigma_1$ and $\mathrm{dh}(r_1\sigma_1, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \omega$. Thus, there exists a minimal subterm $r_1\sigma_1|_{\pi_1}$ of $r_1\sigma_1$ such that $\mathrm{dh}(r_1\sigma_1|_{\pi_1}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \omega$ and all proper subterms of $r_1\sigma_1|_{\pi_1}$ are innermost terminating. Since σ_1 instantiates all variables with normal forms, we have $\pi_1 \in \mathit{Pos}_d(r_1)$, i.e., $r_1\sigma_1|_{\pi_1} = r_1|_{\pi_1}\sigma_1$. In the infinite innermost reduction of $r_1|_{\pi_1}\sigma_1$, again all arguments are normalized first, leading to a term t' with $\mathrm{dh}(t', \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \omega$. As t' is in argument normal form, there is again a rule $\ell_2 \to r_2 \in \mathcal{R}$ and a substitution σ_2 such that $t' = \ell_2\sigma_2 \xrightarrow{\mathrm{i}}_{\mathcal{R}} r_2\sigma_2$ and $\mathrm{dh}(r_2\sigma_2, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \omega$. Continuing in this way, one obtains an infinite chain

$$(\ell_1^\sharp \to \mathrm{COM}_{n_1}(\ldots, r_1|^\sharp_{\pi_1}, \ldots) \mid \sigma_1), \quad (\ell_2^\sharp \to \mathrm{COM}_{n_2}(\ldots, r_2|^\sharp_{\pi_2}, \ldots) \mid \sigma_2), \quad \ldots$$

So there is an infinite chain tree for $\ell_1^\sharp\sigma_1 = t^\sharp$ and hence, $\mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp) = \omega$.

For the case that $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$ is finite, we proceed by induction on $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$. If $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$ is 0, then t is in $\mathcal{R}$-normal form. Thus, $t^\sharp$ is in normal form w.r.t. $DT(\mathcal{R}) \cup \mathcal{R}$ and $\mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp) = 0$.

Otherwise, as t is in argument normal form, there exists a rule $\ell \to r \in \mathcal{R}$ and a substitution σ such that $t = \ell\sigma \xrightarrow{\mathrm{i}}_{\mathcal{R}} r\sigma = u$ and

$$\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = 1 + \mathrm{dh}(u, \xrightarrow{\mathrm{i}}_{\mathcal{R}}). \tag{14}$$

For a term s, we denote by $s{\Downarrow}$ a maximal argument normal form of s, i.e., $s{\Downarrow}$ is an argument normal form such that $s \xrightarrow{\mathrm{i},>\varepsilon}{}^*_{\mathcal{R}} s{\Downarrow}$ and such that for all argument normal forms s' with $s \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} s'$, we have $\mathrm{dh}(s', \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \mathrm{dh}(s{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$. Here, "$\xrightarrow{\mathrm{i},>\varepsilon}{}^*_{\mathcal{R}}$" denotes innermost reductions below the root position. By induction on u, one can easily show that

$$\mathrm{dh}(u, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \Sigma_{\pi\in \mathit{Pos}_d(u)}\, \mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \tag{15}$$

holds (with "=" instead of "$\leq$" if $\mathcal{R}$ is confluent). As σ instantiates all variables by normal forms, $u|_\pi = r\sigma|_\pi$ is in normal form for all $\pi \in \mathit{Pos}_d(u) \setminus \mathit{Pos}_d(r)$. For such π, the fact that $u|_\pi$ is in normal form implies $u|_\pi{\Downarrow} = u|_\pi$ and $\mathrm{dh}(u|_\pi, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = 0$. Hence, from (15) we obtain

$$\begin{aligned} \mathrm{dh}(u, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) &\leq \Sigma_{\pi\in \mathit{Pos}_d(u)}\, \mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \\ &= \Sigma_{\pi\in \mathit{Pos}_d(u)\setminus \mathit{Pos}_d(r)}\, \mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \;+\; \Sigma_{\pi\in \mathit{Pos}_d(r)}\, \mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \\ &= \Sigma_{\pi\in \mathit{Pos}_d(u)\setminus \mathit{Pos}_d(r)}\, \mathrm{dh}(u|_\pi, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \;+\; \Sigma_{\pi\in \mathit{Pos}_d(r)}\, \mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \\ &= \Sigma_{\pi\in \mathit{Pos}_d(r)}\, \mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}). \end{aligned} \tag{16}$$

Note that $\mathrm{dh}(u|_\pi{\Downarrow}, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \mathrm{dh}(u, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) < \mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$ for all $\pi \in \mathit{Pos}_d(r)$. So from the induction hypothesis, (14), and (16) we obtain

$$\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = 1 + \mathrm{dh}(u, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq 1 + \Sigma_{\pi\in \mathit{Pos}_d(r)} \mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(u|_\pi{\Downarrow}^\sharp). \tag{17}$$

Let $\mathit{Pos}_d(r) = \{\pi_1, \ldots, \pi_n\}$. Then there exists a chain tree for $t^\sharp$ where the root node is $(\ell^\sharp \to \mathrm{COM}_n(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_n}) \mid \sigma)$ and where the children of the root node are (maximal) chain trees for $u|_{\pi_1}{\Downarrow}^\sharp, \ldots, u|_{\pi_n}{\Downarrow}^\sharp$. The reason is that $r|_{\pi_i}\sigma = u|_{\pi_i}$ and hence $r|^\sharp_{\pi_i}\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} u|_{\pi_i}{\Downarrow}^\sharp$ for all $i \in \{1, \ldots, n\}$. So, together with (17) we have

$$\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$$

with "=" instead of "$\leq$" for confluent $\mathcal{R}$. □

Compared to the *weak DPs* of [17–19], DTs have the advantage that they allow a more *direct* adaptation of termination techniques ("DP processors") for complexity analysis. While weak DPs also use compound symbols, they only consider the *topmost* defined function symbols in right-hand sides of rules. Hence, [17–19] does not use DP concepts when defined functions occur nested on right-hand sides (as in the m- and the first if-rule) and thus, it cannot fully benefit from the advantages of the DP technique. Instead, [17–19] has to impose several restrictions which are not needed in our approach (cf. the discussion in Sect. 5.2 after Thm. 26). In contrast, the termination techniques of the DP framework can be directly extended in order to work on DTs (i.e., in order to analyze $Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$ for all basic terms t of a certain size). Using Thm. 12, this yields an upper bound for the complexity $\iota_\mathcal{R}$ of the TRS $\mathcal{R}$, cf. Thm. 16.

On the other hand, weak DPs have the advantage that they can also be used to analyze the runtime complexity of *full* rewriting (whereas DTs are restricted to *innermost* rewriting) and DTs may also lead to less precise results when analyzing non-confluent TRSs. As shown in Ex. 11, there exist non-confluent TRSs where $Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$ is exponentially larger than $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_\mathcal{R})$ (in contrast to [17–19], where the step from TRSs to weak DPs does not change the complexity). However, our main interest is in TRSs resulting from "typical" *programs*, which are confluent and use an innermost evaluation strategy. Here, the step from TRSs to DTs does not "lose" anything (i.e., one has equality in Thm. 12).

4 DT Problems

Our goal is to find out automatically how large $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp)$ could be for basic terms t of size n. To this end, we will repeatedly replace the triple $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ by "simpler" triples $\langle\mathcal{D}',\mathcal{S}',\mathcal{R}'\rangle$ and examine $Cplx_{\langle\mathcal{D}',\mathcal{S}',\mathcal{R}'\rangle}(t^\sharp)$ instead.

This is similar to the DP framework where termination problems are represented by so-called DP problems (consisting of a set of DPs and a set of rules) and where DP problems are transformed into "simpler" DP problems repeatedly. For complexity analysis, we consider "DT problems" instead of "DP problems" (our "DT problems" are similar to the "complexity problems" of [30]). As before, the set $\mathcal{S}$ in a DT problem $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ denotes those DTs that should be counted for complexity.

Definition 13 (DT Problem) Let $\mathcal{R}$ be a TRS, $\mathcal{D}$ a set of DTs, $\mathcal{S}\subseteq\mathcal{D}$. Then $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ is a *DT problem* and $\mathcal{R}$'s *canonical DT problem* is $\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle$.

Thm. 12 showed the connection between the derivation height of a term and the maximal number of nodes in a chain tree. This leads to the definition of the *complexity of a DT problem* $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$. It is defined as the asymptotic complexity of the function $\mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}$ which maps any number n to the maximal number of $\mathcal{S}$-nodes in any $(\mathcal{D},\mathcal{R})$-chain tree for $t^\sharp$, where t is a basic term of at most size n.

Definition 14 (Complexity of DT Problems) For a DT problem $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$, its *complexity function* is $\mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) = \sup\{\, Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \mid t\in\mathcal{T}_B, |t|\le n\,\}$. We define the *complexity* $\iota_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}$ of the DT problem as $\iota(\mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle})$.

Note that obviously, $\mathcal{S}_1\subseteq\mathcal{S}_2$ implies $\iota_{\langle\mathcal{D},\mathcal{S}_1,\mathcal{R}\rangle}\sqsubseteq\iota_{\langle\mathcal{D},\mathcal{S}_2,\mathcal{R}\rangle}$.

Example 15 Consider $\mathcal{R}$ from Ex. 4 and let $\mathcal{D}=DT(\mathcal{R})=\{(1),\ldots,(8)\}$. For $t\in\mathcal{T}_B$ with $|t|=n$, the maximal chain tree for $t^\sharp$ has approximately n^2 nodes, i.e., $\mathrm{irc}_{\langle\mathcal{D},\mathcal{D},\mathcal{R}\rangle}(n)\in\mathrm{O}(n^2)$. Thus, $\langle\mathcal{D},\mathcal{D},\mathcal{R}\rangle$'s complexity is $\iota_{\langle\mathcal{D},\mathcal{D},\mathcal{R}\rangle}=Pol_2$.

Thm. 16 shows that to analyze the complexity of a TRS $\mathcal{R}$, it suffices to analyze the complexity of its canonical DT problem.

Theorem 16 (Upper bound for TRSs via Canonical DT Problems) *Let $\mathcal{R}$ be a TRS and let $\langle \mathcal{D}, \mathcal{D}, \mathcal{R} \rangle$ be the corresponding canonical DT problem. Then we have $\iota_{\mathcal{R}} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}$ and if $\mathcal{R}$ is confluent, we have $\iota_{\mathcal{R}} = \iota_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}$.*

Proof For any $n \in \mathbb{N}$, $\mathrm{irc}_{\mathcal{R}}(n) = \sup\{\, \mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}_B, |t| \le n \,\} \le \sup\{\, \mathit{Cplx}_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}(t^\sharp) \mid t \in \mathcal{T}_B, |t| \le n \,\} = \mathrm{irc}_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}(n)$ by Thm. 12, with equality if $\mathcal{R}$ is confluent. Thus, $\iota_{\mathcal{R}} = \iota(\mathrm{irc}_{\mathcal{R}}) \sqsubseteq \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}) = \iota_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}$ and if $\mathcal{R}$ is confluent, we even have $\iota_{\mathcal{R}} = \iota_{\langle \mathcal{D},\mathcal{D},\mathcal{R} \rangle}$. $\square$

Now we can introduce our notion of processors which is analogous to the "DP processors" for termination [11, 12] (and related to the "complexity problem processors" in [30]). A DT processor transforms a DT problem P to a pair (c, P') of an asymptotic complexity $c \in \mathfrak{C}$ and a DT problem P', such that P's complexity is bounded by the maximum of c and of the complexity of P'.

Definition 17 (Processor, $\oplus$) A *DT processor* Proc is a function $\textsc{Proc}(P) = (c, P')$ mapping any DT problem P to a complexity $c \in \mathfrak{C}$ and a DT problem P'. A processor is *sound* if $\iota_P \sqsubseteq c \oplus \iota_{P'}$. Here, "$\oplus$" is the "maximum" function on $\mathfrak{C}$, i.e., for any $c, d \in \mathfrak{C}$, we define $c \oplus d = d$ if $c \sqsubseteq d$ and $c \oplus d = c$ otherwise.

The following lemma about the connection between $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R} \rangle}$ and the operation $\oplus$ will be useful in the proofs later on.

Lemma 18 (Connection between $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R} \rangle}$ and $\oplus$)

(a) Let f and g be functions from $\mathbb{N}$ to $\mathbb{N} \cup \{\omega\}$. Then $\iota(f) \oplus \iota(g) = \iota(f+g)$.
(b) For any $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{D}$, we have $\iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle} \oplus \iota_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle} = \iota_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}$.

Proof For (a), $\iota(g) \sqsubset \iota(f)$ implies $\iota(f+g) = \iota(f)$ and $\iota(f) \sqsubseteq \iota(g)$ implies $\iota(f+g) = \iota(g)$.

For (b), let $t^\sharp \in \mathcal{T}^\sharp$ and let m be the maximal number of nodes from $\mathcal{S}_1 \cup \mathcal{S}_2$ occurring in any $(\mathcal{D},\mathcal{R})$-chain tree for $t^\sharp$, i.e., $\mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}(t^\sharp) = m$. Similarly, let m_1 and m_2 be the maximal numbers of nodes from $\mathcal{S}_1$ resp. $\mathcal{S}_2$ occurring in $(\mathcal{D},\mathcal{R})$-chain trees for $t^\sharp$, i.e., $\mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}(t^\sharp) = m_1$ and $\mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}(t^\sharp) = m_2$. When extending "$\le$" and "$+$" to $\mathbb{N} \cup \{\omega\}$, we have $\sup\{m_1, m_2\} \le m \le m_1 + m_2$, i.e., $\sup\{\mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}(t^\sharp), \mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}(t^\sharp)\} \le \mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}(t^\sharp) \le \mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}(t^\sharp) + \mathit{Cplx}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}(t^\sharp)$. So on the one hand, we have $\sup\{\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}(n), \mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}(n)\} \le \mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}(n)$ for all $n \in \mathbb{N}$ which means $\iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle} \oplus \iota_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle} = \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}) \oplus \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}) \sqsubseteq \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}) = \iota_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}$. On the other hand, we have $\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}(n) \le \mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}(n) + \mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}(n)$ for all $n \in \mathbb{N}$ which means $\iota_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle} = \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1 \cup \mathcal{S}_2,\mathcal{R} \rangle}) \sqsubseteq \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle} + \mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}) = \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle}) \oplus \iota(\mathrm{irc}_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}) = \iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R} \rangle} \oplus \iota_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R} \rangle}$ by (a). $\square$

To analyze the complexity $\iota_{\mathcal{R}}$ of a TRS $\mathcal{R}$, we start with the canonical DT problem $P_0 = \langle DT(\mathcal{R}), DT(\mathcal{R}), \mathcal{R} \rangle$. Then we apply a sound processor to P_0 which yields a result (c_1, P_1). Afterwards, we apply another (possibly different) sound processor to P_1 which yields (c_2, P_2), etc. This is repeated until we obtain a *solved* DT problem (whose complexity is obviously Pol_0).

Definition 19 (Proof Chain, Solved DT Problem) We call a DT problem $P = \langle \mathcal{D},\mathcal{S},\mathcal{R} \rangle$ *solved* if $\mathcal{S} = \varnothing$. A *proof chain* is a finite sequence $P_0 \overset{c_1}{\rightsquigarrow} P_1 \overset{c_2}{\rightsquigarrow} \ldots \overset{c_k}{\rightsquigarrow} P_k$ ending with a solved DT problem P_k, such that for all $0 \le i < k$ there exists a sound processor $\textsc{Proc}_i$ with $\textsc{Proc}_i(P_i) = (c_{i+1}, P_{i+1})$.

By Def. 17 and 19, for every P_i in a proof chain, $c_{i+1} \oplus \ldots \oplus c_k$ is an upper bound for its complexity ι_{P_i}. Here, the empty sum (for $i = k$) is defined as Pol_0.

Theorem 20 (Approximating Complexity by Proof Chain) *Let* $P_0 \overset{c_1}{\rightsquigarrow} P_1 \overset{c_2}{\rightsquigarrow} \ldots \overset{c_k}{\rightsquigarrow} P_k$ *be a proof chain. Then* $\iota_{P_0} \sqsubseteq c_1 \oplus \ldots \oplus c_k$.

Proof The theorem can easily be proved by induction on k. □

Thm. 16 and 20 now imply that our approach for complexity analysis is correct.

Corollary 21 (Correctness of Approach) *If* P_0 *is the canonical DT problem for a TRS* $\mathcal{R}$ *and* $P_0 \overset{c_1}{\rightsquigarrow} \ldots \overset{c_k}{\rightsquigarrow} P_k$ *is a proof chain, then* $\iota_{\mathcal{R}} \sqsubseteq c_1 \oplus \ldots \oplus c_k$.

Of course, one could also define DT processors that transform a DT problem P into a complexity c and a *set* $\{P'_1, \ldots, P'_n\}$ such that $\iota_P \sqsubseteq c \oplus \iota_{P'_1} \oplus \ldots \oplus \iota_{P'_n}$. Then instead of a proof chain one would obtain a proof tree and Cor. 21 would have to be adapted accordingly.

5 DT Processors

In this section, we present several processors to simplify DT problems automatically. To this end, we adapt processors of the DP framework for termination.

The *usable rules processor* (Sect. 5.1) simplifies a problem $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ by deleting rules from $\mathcal{R}$. The *reduction pair processor* (Sect. 5.2) removes DTs from $\mathcal{S}$, based on term orders. In Sect. 5.3 we introduce the *dependency graph*, on which the *leaf removal*, *rhs simplification*, *unreachable DT removal*, and *knowledge propagation* (Sect. 5.4) processors are based. Finally, Sect. 5.5 adapts processors based on transformations like *narrowing*.

5.1 Usable Rules Processor

As in termination analysis, we can restrict ourselves to those rewrite rules that can be used to reduce right-hand sides of DTs (when instantiating their variables with normal forms). This leads to the notion of *usable rules*.

Definition 22 (Usable Rules $\mathcal{U}_{\mathcal{R}}$ [1]) For a TRS $\mathcal{R}$ and any symbol f, let $\mathrm{Rls}_{\mathcal{R}}(f) = \{\ell \to r \mid \mathrm{root}(\ell) = f\}$. For any term t, $\mathcal{U}_{\mathcal{R}}(t)$ is the smallest set with

- $\mathcal{U}_{\mathcal{R}}(x) = \varnothing$ if $x \in \mathcal{V}$ and
- $\mathcal{U}_{\mathcal{R}}(f(t_1, \ldots, t_n)) = \mathrm{Rls}_{\mathcal{R}}(f) \cup \bigcup_{\ell \to r \in \mathrm{Rls}_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}}(r) \cup \bigcup_{1 \le i \le n} \mathcal{U}_{\mathcal{R}}(t_i)$

For any set $\mathcal{D}$ of DTs, we define $\mathcal{U}_{\mathcal{R}}(\mathcal{D}) = \bigcup_{s \to t \in \mathcal{D}} \mathcal{U}_{\mathcal{R}}(t)$.

So for $\mathcal{R}$ and $DT(\mathcal{R})$ in Ex. 4 and 6, $\mathcal{U}_{\mathcal{R}}(DT(\mathcal{R}))$ contains just the gt- and the p-rules. The following processor removes non-usable rules from DT problems.

Theorem 23 (Usable Rules Processor) *Let* $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ *be a DT problem. Then the following processor is sound:* $\textsc{Proc}(\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle) = (Pol_0, \langle \mathcal{D}, \mathcal{S}, \mathcal{U}_{\mathcal{R}}(\mathcal{D}) \rangle)$.

Proof We have to prove $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \sqsubseteq Pol_0 \oplus \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{U}_{\mathcal{R}}(\mathcal{D}) \rangle}$. This is equivalent to $\iota(\mathrm{irc}_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}) \sqsubseteq \iota(\mathrm{irc}_{\langle \mathcal{D}, \mathcal{S}, \mathcal{U}_{\mathcal{R}}(\mathcal{D}) \rangle})$. This holds, since for all $\mathcal{S} \subseteq \mathcal{D}$, we have $\mathrm{irc}_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} = \mathrm{irc}_{\langle \mathcal{D}, \mathcal{S}, \mathcal{U}_{\mathcal{R}}(\mathcal{D}) \rangle}$. The reason is that in a chain tree, variables are always instantiated with normal forms. □

So when applying the usable rules processor on the canonical DT problem $\langle \mathcal{D}, \mathcal{D}, \mathcal{R} \rangle$ of $\mathcal{R}$ from Ex. 4, we obtain $\langle \mathcal{D}, \mathcal{D}, \mathcal{R}_1 \rangle$ where $\mathcal{R}_1$ are the gt- and p-rules.

The idea of applying *usable rules* also for complexity analysis is due to [17], which introduced a technique similar to Thm. 23. While Def. 22 is the most basic definition of *usable rules*, the processor of Thm. 23 can also be used with more sophisticated definitions of "usable rules" (e.g., as in [12]).

5.2 Reduction Pair Processor

Using orders is one of the most important methods for termination or complexity analysis. In the most basic approach, one tries to find a well-founded order $\succ$ such that every reduction step (strictly) decreases w.r.t. $\succ$. This proves termination and most reduction orders also imply some complexity bound, cf. e.g. [7, 20]. However, *direct* applications of orders have two main drawbacks: The obtained bounds are often far too high to be useful and there are many TRSs that cannot be oriented strictly with standard orders amenable to automation, cf. [30].

Therefore, the *reduction pair processor* of the DP framework only requires a strict decrease (w.r.t. $\succ$) for at least one DP, while for all other DPs and rules, a weak decrease (w.r.t. $\succsim$) suffices. Making the rules weakly decreasing ensures that one has a weak decrease when going from one dependency pair to the next in a chain. Thus, the strictly decreasing DPs can only occur finitely often in chains and can therefore be deleted. Afterwards one can use other orders (or termination techniques) to solve the remaining DP problem. To adapt the reduction pair processor for complexity analysis, we have to restrict ourselves to COM-*monotonic* orders. (In [17] "COM-monotonic" is called "safe".)

Definition 24 (Reduction Pair) A *reduction pair* $(\succsim, \succ)$ consists of a stable monotonic quasi-order $\succsim$ and a stable well-founded order $\succ$ which are compatible (i.e., $\succsim \circ \succ \circ \succsim \subseteq \succ$). An order $\succ$ is COM-*monotonic* iff $\textsc{Com}_n(s_1^\sharp, \ldots, s_i^\sharp, \ldots, s_n^\sharp) \succ \textsc{Com}_n(s_1^\sharp, \ldots, t^\sharp, \ldots, s_n^\sharp)$ for all $n \in \mathbb{N}$, all $1 \le i \le n$, and all $s_1^\sharp, \ldots, s_n^\sharp, t^\sharp \in \mathcal{T}^\sharp$ with $s_i^\sharp \succ t^\sharp$. A reduction pair $(\succsim, \succ)$ is COM-monotonic iff $\succ$ is COM-monotonic. For a reduction pair $(\succsim, \succ)$ and a set $\mathcal{D}$ of DTs, we write $\mathcal{D}_\succ = \mathcal{D} \cap \succ$ and $\mathcal{D}_\succsim = \mathcal{D} \cap \succsim$.

For a DT problem $(\mathcal{D}, \mathcal{S}, \mathcal{R})$, we orient $\mathcal{D} \cup \mathcal{R}$ by $\succsim$ or $\succ$. But in contrast to the reduction pair processor for termination, if a DT is oriented strictly, we may not remove it from $\mathcal{D}$, *but only from* $\mathcal{S}$. So the DT is not counted anymore for complexity, but it may still be used in reductions. We will improve this later in Sect. 5.4.

Example 25 This TRS $\mathcal{R}$ shows why DTs may not be removed from $\mathcal{D}$. (An alternative such example is shown in [9, Ex. 11].)

$$\mathsf{f}(\mathsf{0}) \to \mathsf{0} \qquad \mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{id}(x)) \qquad \mathsf{id}(\mathsf{0}) \to \mathsf{0} \qquad \mathsf{id}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{id}(x))$$

Let $\mathcal{D} = DT(\mathcal{R}) = \{\mathsf{f}^\sharp(\mathsf{0}) \to \textsc{Com}_0,\ \mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_2(\mathsf{f}^\sharp(\mathsf{id}(x)), \mathsf{id}^\sharp(x)),\ \mathsf{id}^\sharp(\mathsf{0}) \to \textsc{Com}_0,\ \mathsf{id}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_1(\mathsf{id}^\sharp(x))\}$, where $\mathcal{U}_\mathcal{R}(\mathcal{D})$ are just the id*-rules. For the DT problem $\langle \mathcal{D}, \mathcal{S}, \mathcal{U}_\mathcal{R}(\mathcal{D}) \rangle$ with $\mathcal{S} = \mathcal{D}$, there is a linear* polynomial interpretation $[\cdot]$ *that orients the first two DTs strictly and the remaining DTs and usable rules weakly: $[\mathsf{0}] = 0, [\mathsf{s}](x) = x + 1, [\mathsf{id}](x) = x, [\mathsf{f}^\sharp](x) = x + 1, [\mathsf{id}^\sharp](x) = 0, [\textsc{Com}_0] = 0, [\textsc{Com}_1](x) = x, [\textsc{Com}_2](x, y) = x + y$. If one removed the first two DTs from $\mathcal{D}$, there would be another linear polynomial interpretation* *that orients the remaining DTs strictly (e.g., by $[\mathsf{id}^\sharp](x) = x+1$). Then, one would falsely conclude that the whole TRS has linear runtime complexity.*

Hence, the first two DTs should only be removed from $\mathcal{S}$, not from $\mathcal{D}$. This results in $\langle \mathcal{D}, \mathcal{S}', \mathcal{U}_\mathcal{R}(\mathcal{D})\rangle$ where $\mathcal{S}'$ consists of the last two DTs. These DTs can occur quadratically often in reductions with $\mathcal{D} \cup \mathcal{U}_\mathcal{R}(\mathcal{D})$. Hence, when trying to orient $\mathcal{S}'$ strictly and the remaining DTs and usable rules weakly, we have to use a quadratic polynomial interpretation (e.g., $[0] = 0, [\mathsf{s}](x) = x+2, [\mathsf{id}](x) = x, [\mathsf{f}^\sharp](x) = x^2, [\mathsf{id}^\sharp](x) = x+1, [\textsc{Com}_0] = 0, [\textsc{Com}_1](x) = x, [\textsc{Com}_2](x,y) = x+y$). Thus, now we (correctly) conclude that the TRS has quadratic runtime complexity (indeed, $\mathrm{dh}(\mathsf{f}(\mathsf{s}^n(0)), \xrightarrow{\mathrm{i}}_\mathcal{R}) = \frac{(n+1)\cdot(n+2)}{2}$).

So when applying the reduction pair processor to $\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle$, we get $(c, \langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_\succ, \mathcal{R}\rangle)$. Here, c is an upper bound for the number of $\mathcal{D}_\succ$-steps in innermost reductions with $\mathcal{D} \cup \mathcal{R}$.

Theorem 26 (Reduction Pair Processor) *Let $P = \langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle$ be a DT problem and $(\succsim, \succ)$ be a* Com*-monotonic reduction pair. Let $\mathcal{D} \subseteq {\succsim} \cup {\succ}$, $\mathcal{R} \subseteq {\succsim}$, and $c \sqsupseteq \iota(\mathrm{irc}_\succ)$ for the function $\mathrm{irc}_\succ(n) = \sup\{\,\mathrm{dh}(t^\sharp, \succ) \mid t \in \mathcal{T}_B, |t| \le n\}$. Then the following processor is sound:* $\textsc{Proc}(\,\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle\,) = (c,\ \langle \mathcal{D},\ \mathcal{S} \setminus \mathcal{D}_\succ,\ \mathcal{R}\rangle)$.

Proof To prove soundness, we have to show that $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle} \sqsubseteq c \oplus \iota_{\langle \mathcal{D}, \mathcal{S}\setminus\mathcal{D}_\succ, \mathcal{R}\rangle}$ holds. It suffices to show $\iota_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle} \sqsubseteq \iota(\mathrm{irc}_\succ)$, since then $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{S}\cup\mathcal{D}_\succ, \mathcal{R}\rangle} = \iota_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle} \oplus \iota_{\langle \mathcal{D}, \mathcal{S}\setminus\mathcal{D}_\succ, \mathcal{R}\rangle} \sqsubseteq \iota(\mathrm{irc}_\succ) \oplus \iota_{\langle \mathcal{D}, \mathcal{S}\setminus\mathcal{D}_\succ, \mathcal{R}\rangle} \sqsubseteq c \oplus \iota_{\langle \mathcal{D}, \mathcal{S}\setminus\mathcal{D}_\succ, \mathcal{R}\rangle}$ by Lemma 18 (b).

To prove $\iota_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle} \sqsubseteq \iota(\mathrm{irc}_\succ)$, let $s \in \mathcal{T}_B$ be a basic term and consider an arbitrary innermost $(\mathcal{D} \cup \mathcal{R})$-reduction sequence starting with $s^\sharp$. All terms in such a reduction sequence are of the form $C[t_1^\sharp, \ldots, t_n^\sharp]$ for a context C consisting only of compound symbols and $t_1^\sharp, \ldots, t_n^\sharp \in \mathcal{T}^\sharp$.

As $\succ$ is Com-monotonic, all $\mathcal{D}$-steps in such a reduction sequence take place on monotonic positions. So if $u \xrightarrow{\mathrm{i}}_{\mathcal{D}_\succ} v$ is a rewrite step in an innermost $(\mathcal{D} \cup \mathcal{R})$-reduction of $s^\sharp$, then $u \succ v$. On the other hand, $\succsim$ is monotonic, too. Hence, $u \to_{\mathcal{D}_\succsim \cup \mathcal{R}} v$ implies $u \succsim v$. Now let

$$s^\sharp \;=\; s_0 \;\xrightarrow{\mathrm{i}}_{\nu_0}\; t_0 \;\xrightarrow{\mathrm{i}}{}^*_\mathcal{R}\; s_1 \;\xrightarrow{\mathrm{i}}_{\nu_1}\; t_1 \;\xrightarrow{\mathrm{i}}{}^*_\mathcal{R}\; s_2 \;\ldots$$

be a (finite or infinite) innermost $(\mathcal{D} \cup \mathcal{R})$-reduction, where $\nu_i \in \mathcal{D}$ for all i. Then

$$s^\sharp = s_0 \cdot\!\succ_0 t_0 \succsim s_1 \cdot\!\succ_1 t_1 \succsim s_2 \;\ldots$$

holds. Here "$\cdot\!\succ_i$" is "$\succ$" if $\nu_i \in \mathcal{D}_\succ$ and "$\succsim$" otherwise. Let $n_1 < n_2 < \ldots$ be the indexes where $\cdot\!\succ_{n_j} = \succ$. For each n_j we have $s_{n_j} \succ t_{n_j}$. As ${\succsim} \circ {\succ} \circ {\succsim} \subseteq {\succ}$, we obtain $s^\sharp \succ t_{n_1} \succ t_{n_2} \succ \ldots$ and therefore $\mathrm{dh}(s^\sharp, \succ) > \mathrm{dh}(t_{n_1}, \succ) > \mathrm{dh}(t_{n_2}, \succ) > \ldots$ or $\mathrm{dh}(s^\sharp, \succ) = \omega$. Hence, $\mathrm{irc}_\succ(|s^\sharp|)$ is an upper bound on the number of $\mathcal{D}_\succ$-steps in any innermost $(\mathcal{D} \cup \mathcal{R})$-reduction of $s^\sharp$.

Moreover, $\mathit{Cplx}_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle}(s^\sharp)$ is the maximal number of $\mathcal{S}$-steps in any innermost $(\mathcal{D} \cup \mathcal{R})$-reduction of $s^\sharp$. Hence, $\mathit{Cplx}_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle}(s^\sharp) \le \mathrm{irc}_\succ(|s^\sharp|)$ for all $s \in \mathcal{T}_B$. This implies $\mathrm{irc}_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle}(n) \le \mathrm{irc}_\succ(n)$ for all n and hence, $\iota_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle} = \iota(\mathrm{irc}_{\langle \mathcal{D}, \mathcal{D}_\succ, \mathcal{R}\rangle}) \sqsubseteq \iota(\mathrm{irc}_\succ)$. □

Note that our reduction pair processor is much closer to the original processor of the DP framework than [17]. In [17, Cor. 18], all (weak) DPs and the usable rules have to be oriented strictly in one go with the same order and this order has to be monotonic on *all* symbols. (In [19], the authors weaken this to monotonicity on only those positions below which rewriting can take place.) In [17, Thm. 3], again all (weak) DPs and the usable rules have to be oriented strictly in one go, but with two different orders. However, here one is restricted to non-duplicating TRSs. The idea of not removing strictly oriented rules but only avoiding to count them for complexity is also used in [30], i.e., here we integrate an approach of [30] as a processor. However, [30] treats derivational complexity instead of (innermost) runtime complexity, and it operates directly on TRSs and not on DPs or DTs. Therefore, [30] has to impose stronger restrictions (it requires $\succ$ to be monotonic on *all* symbols) and it cannot use other DP- resp. DT-based processors.

As noted by [25], the condition "$c \sqsupseteq \iota(\mathrm{irc}_{\succ})$" for the function $\mathrm{irc}_{\succ}(n) = \sup\{\,\mathrm{dh}(t^\sharp,\succ) \mid t \in \mathcal{T}_B, |t| \le n\}$ in Thm. 26 can be weakened by replacing $\mathrm{dh}(t^\sharp,\succ)$ with $\mathrm{dh}(t^\sharp, \succ \cap \overset{\mathrm{i}}{\to}_{\mathcal{D}/\mathcal{R}})$, where $\to_{\mathcal{D}/\mathcal{R}} = \to_{\mathcal{R}}^* \circ \to_{\mathcal{D}} \circ \to_{\mathcal{R}}^*$ and $\overset{\mathrm{i}}{\to}_{\mathcal{D}/\mathcal{R}}$ is the restriction of $\to_{\mathcal{D}/\mathcal{R}}$ where in each rewrite step with $\to_{\mathcal{R}}$ or $\to_{\mathcal{D}}$, the arguments of the redex must be in $(\mathcal{D} \cup \mathcal{R})$-normal form, cf. [3]. Such a weakening is required to use reduction pairs based on path orders where a term $t^\sharp$ may start $\succ$-decreasing sequences of arbitrary (finite) length.

To automate Thm. 26, we need reduction pairs $(\succsim, \succ)$ where an upper bound c for $\iota(\mathrm{irc}_{\succ})$ is easy to compute. This holds for reduction pairs based on *polynomial interpretations* with coefficients from $\mathbb{N}$ (which are well suited for automation). For COM-monotonicity, we restrict ourselves to *complexity polynomial interpretations (CPIs)* $[\cdot]$ where $[\mathrm{COM}_n](x_1,\ldots,x_n) = x_1 + \ldots + x_n$ for all $n \in \mathbb{N}$. This is the "smallest" polynomial which is monotonic in $x_1,\ldots,x_n$. As COM_n only occurs on right-hand sides of inequalities, $[\mathrm{COM}_n]$ should be as small as possible.

Moreover, a *CPI* interprets constructors $f \in \Sigma \setminus \Sigma_d$ by polynomials $[f](x_1,\ldots,x_n) = a_1x_1 + \ldots + a_nx_n + b$ where $b \in \mathbb{N}$ and $a_i \in \{0,1\}$. This ensures that the mapping from constructor ground terms $t \in \mathcal{T}(\Sigma \setminus \Sigma_d, \varnothing)$ to their interpretations is in $\mathrm{O}(|t|)$, cf. [7, 20]. Note that the interpretations in Ex. 25 were *CPIs*.

Thm. 27 shows how such interpretations can be used for the processor of Thm. 26. Here, as an upper bound c for $\iota(\mathrm{irc}_{\succ})$, one can simply take Pol_m, where m is the maximal degree of the polynomials in the interpretation.

Theorem 27 (Reduction Pair Processor with Polynomial Interpretations) *Let $P = \langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle$ be a DT problem and let $\succsim$ and $\succ$ be induced by a CPI $[\cdot]$. Let $m \in \mathbb{N}$ be the maximal degree of all polynomials $[f^\sharp]$, for all $f^\sharp$ with $f \in \Sigma_d$. Let $\mathcal{D} \subseteq \succsim \cup \succ$ and $\mathcal{R} \subseteq \succsim$. Then the following processor is sound:* $\mathrm{PROC}(\,\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle\,) = (Pol_m,\ \langle \mathcal{D},\ \mathcal{S} \setminus \mathcal{D}_{\succ},\ \mathcal{R}\rangle)$.

Proof CPIs are obviously COM-monotonic. Hence, it remains to prove that $Pol_m \sqsupseteq \iota(\mathrm{irc}_{\succ})$ holds. Let $[\cdot]_0$ be a variant of the polynomial interpretation which maps every variable to 0. Then we have $\mathrm{dh}(t,\succ) \le [t]_0$ for all terms t. Thus,

$$\mathrm{irc}_{\succ}(n) \le \sup\{[t^\sharp]_0 \mid t \in \mathcal{T}_B \text{ and } |t| \le n\}. \tag{18}$$

Let b_{max} be the maximum of all $[f](0,\ldots,0)$, for all constructors $f \in \Sigma \setminus \Sigma_d$. Then for every term s containing only constructors and variables, we obtain $[s]_0 \le b_{max} \cdot |s|$, where $|s|$ is again the size of s. Hence, there exists a number $k \in \mathbb{N}$ such that for all $t \in \mathcal{T}_B$ we have

$$[t^\sharp]_0 \le k \cdot [f^\sharp](|t|,\ldots,|t|), \quad \text{where } f^\sharp = \mathrm{root}(t^\sharp). \tag{19}$$

To see this, note that for $t = f(t_1,\ldots,t_n) \in \mathcal{T}_B$ we have

$$\begin{aligned}
[t^\sharp]_0 &= [f^\sharp]([t_1]_0,\ldots,[t_n]_0) \\
&\le [f^\sharp](b_{max}\cdot|t_1|,\ldots,b_{max}\cdot|t_n|) \\
&\le [f^\sharp](b_{max}\cdot|t|,\ldots,b_{max}\cdot|t|) \\
&\le b_{max}^m \cdot [f^\sharp](|t|,\ldots,|t|), \quad \text{since the degree of } [f^\sharp] \text{ is at most } m
\end{aligned}$$

Hence,

$$\begin{aligned}\mathrm{irc}_{\succ}(n) &\leq \sup\{[t^\sharp]_0 \mid t \in \mathcal{T}_B \text{ and } |t| \leq n\} && \text{by (18)}\\ &\leq k \cdot [f^\sharp](n,\dots,n) && \text{by (19).}\end{aligned}$$

Since the polynomials $[f^\sharp]$ have at most degree m, we have $\iota(\mathrm{irc}_{\succ}) \sqsubseteq Pol_m$. □

Example 28 This TRS [1] illustrates Thm. 27, where $\mathsf{q}(x,y,y)$ *computes* $\lfloor \frac{x}{y} \rfloor$.

$$\mathsf{q}(\mathsf{0},\mathsf{s}(y),\mathsf{s}(z)) \to \mathsf{0} \qquad \mathsf{q}(\mathsf{s}(x),\mathsf{s}(y),z) \to \mathsf{q}(x,y,z) \qquad \mathsf{q}(x,\mathsf{0},\mathsf{s}(z)) \to \mathsf{s}(\mathsf{q}(x,\mathsf{s}(z),\mathsf{s}(z)))$$

The dependency tuples $\mathcal{D}$ *of this TRS are*

$$\mathsf{q}^\sharp(\mathsf{0},\mathsf{s}(y),\mathsf{s}(z)) \to \textsc{Com}_0 \quad (20) \qquad \mathsf{q}^\sharp(\mathsf{s}(x),\mathsf{s}(y),z) \to \textsc{Com}_1(\mathsf{q}^\sharp(x,y,z)) \quad (21)$$

$$\mathsf{q}^\sharp(x,\mathsf{0},\mathsf{s}(z)) \to \textsc{Com}_1(\mathsf{q}^\sharp(x,\mathsf{s}(z),\mathsf{s}(z))) \quad (22)$$

As the usable rules are empty, Thm. 23 transforms the canonical DT problem to $\langle \mathcal{D},\mathcal{D},\varnothing\rangle$. *Consider the CPI* $[\mathsf{0}] = 0$, $[\mathsf{s}](x) = x+1$, $[\mathsf{q}^\sharp](x,y,z) = x+1$, $[\textsc{Com}_0] = 0$, $[\textsc{Com}_1](x) = x$. *With the corresponding reduction pair, the DTs* (20) *and* (21) *are strictly decreasing and* (22) *is weakly decreasing. Moreover, the degree of* $[\mathsf{q}^\sharp]$ *is 1. Hence, the reduction pair processor returns* $(Pol_1, \langle \mathcal{D}, \{(22)\}, \varnothing\rangle)$. *However, no reduction pair based on CPIs orients* (22) *strictly and both* (20) *and* (21) *weakly. So for the moment we cannot simplify this problem further.*

Apart from polynomial interpretations, our reduction pair processor could of course also use matrix interpretations [9, 22, 24, 26, 29], polynomial path orders (POP* [3]), etc. For POP*, we would extend $\mathfrak{C}$ by a complexity Pol_* for polytime computability, where $Pol_n \sqsubset Pol_* \sqsubset\, ?$ for all $n \in \mathbb{N}$.

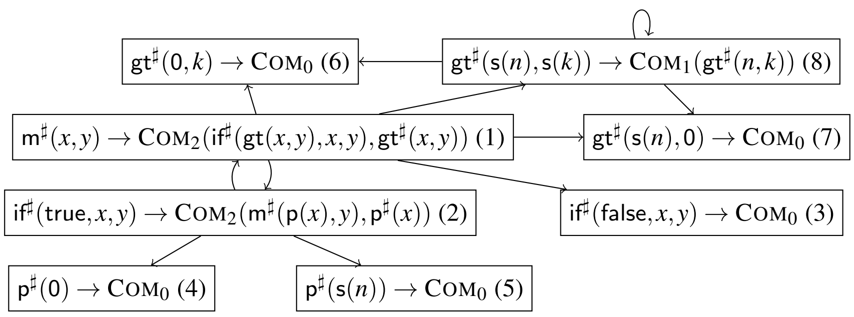
5.3 Dependency Graph Processors

As in the DP framework for termination, it is useful to have a finite representation of (a superset of) all possible chain trees.

Definition 29 (Dependency Graph) Let $\mathcal{D}$ be a set of DTs and $\mathcal{R}$ a TRS. The $(\mathcal{D},\mathcal{R})$-*dependency graph* is the directed graph whose nodes are the DTs in $\mathcal{D}$ and there is an edge from $s \to t$ to $u \to v$ in the dependency graph iff there is a chain tree with an edge from a node $(s \to t \mid \sigma_1)$ to a node $(u \to v \mid \sigma_2)$.

Every $(\mathcal{D},\mathcal{R})$-chain corresponds to a path in the $(\mathcal{D},\mathcal{R})$-dependency graph. While dependency graphs are not computable in general, there are several techniques to compute over-approximations of dependency graphs for termination, cf. e.g. [1]. These techniques can also be applied for $(\mathcal{D},\mathcal{R})$-dependency graphs.

Example 30 For the TRS $\mathcal{R}$ *from Ex. 4, we obtain the following* $(\mathcal{D},\mathcal{R}_1)$*-dependency graph, where* $\mathcal{D} = DT(\mathcal{R})$ *and* $\mathcal{R}_1$ *are the* gt*- and* p*-rules.*


$\mathsf{gt}^\sharp(0,k) \to \mathrm{C{\scriptsize OM}}_0$ (6)
$\mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \mathrm{C{\scriptsize OM}}_1(\mathsf{gt}^\sharp(n,k))$ (8)
$\mathsf{m}^\sharp(x,y) \to \mathrm{C{\scriptsize OM}}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y),\mathsf{gt}^\sharp(x,y))$ (1)
$\mathsf{gt}^\sharp(\mathsf{s}(n),0) \to \mathrm{C{\scriptsize OM}}_0$ (7)
$\mathsf{if}^\sharp(\mathsf{true},x,y) \to \mathrm{C{\scriptsize OM}}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y),\mathsf{p}^\sharp(x))$ (2)
$\mathsf{if}^\sharp(\mathsf{false},x,y) \to \mathrm{C{\scriptsize OM}}_0$ (3)
$\mathsf{p}^\sharp(0) \to \mathrm{C{\scriptsize OM}}_0$ (4)
$\mathsf{p}^\sharp(\mathsf{s}(n)) \to \mathrm{C{\scriptsize OM}}_0$ (5)

For termination analysis, one can regard strongly connected components of the graph separately and ignore nodes that are not on cycles. Such a strong form of modularity is not possible for complexity analysis: If one regards the DTs $\mathcal{D}' = \{(1),(2)\}$ and $\mathcal{D}'' = \{(8)\}$ on the two SCCs of the graph separately, then both resulting DT problems $\langle \mathcal{D}',\mathcal{D}',\mathcal{R}_1\rangle$ and $\langle \mathcal{D}'',\mathcal{D}'',\mathcal{R}_1\rangle$ have linear complexity. However, this allows no conclusions on the complexity of $\langle \mathcal{D},\mathcal{D},\mathcal{R}_1\rangle$ (which is quadratic). Nevertheless, it is often possible to remove DTs $s \to t$ that are leaves *(i.e., $s \to t$ has no successors in the dependency graph). This yields $\langle \mathcal{D}_1,\mathcal{D}_1,\mathcal{R}_1\rangle$, where $\mathcal{D}_1 = \{(1),(2),(8)\}$.*

Theorem 31 (Leaf Removal Processor) *Let $\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle$ be a DT problem and let $s \to t \in \mathcal{D}$ be a leaf in the $(\mathcal{D},\mathcal{R})$-dependency graph. By $Pre(s \to t) \subseteq \mathcal{D}$ we denote the predecessors of $s \to t$, i.e., $Pre(s \to t)$ consists of all DTs $u \to v$ where there is an edge from $u \to v$ to $s \to t$ in the $(\mathcal{D},\mathcal{R})$-dependency graph. If $s \to t \notin \mathcal{S}$ or $Pre(s \to t) \subseteq \mathcal{S}$, then the following processor is sound:* $\mathrm{P{\scriptsize ROC}}(\,\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle\,) = (Pol_0, \langle \mathcal{D} \setminus \{s \to t\}, \mathcal{S} \setminus \{s \to t\}, \mathcal{R}\rangle)$.

Proof Let T be an arbitrary $(\mathcal{D},\mathcal{R})$-chain tree and let T' result from removing all leaves marked with $s \to t$. Since $s \to t$ is a leaf in the dependency graph, it cannot occur in inner nodes of the chain tree T. Hence, T' is a $(\mathcal{D} \setminus \{s \to t\},\mathcal{R})$-chain tree. If $s \to t \notin \mathcal{S}$, we obviously have $|T|_{\mathcal{S}} = |T'|_{\mathcal{S}\setminus\{s\to t\}}$. Hence, $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} = \iota_{\langle \mathcal{D}\setminus\{s\to t\},\mathcal{S}\setminus\{s\to t\},\mathcal{R}\rangle}$ and thus, the processor is sound.

Otherwise, if $Pre(s \to t) \subseteq \mathcal{S}$, let k be the maximal index of the compound symbols $\mathrm{C{\scriptsize OM}}_k$ occurring in $Pre(s \to t)$. Then T can have at most $1 + k \cdot |T|_{Pre(s\to t)}$ nodes labeled with $s \to t$ and hence $|T|_{\mathcal{S}} \le |T|_{\mathcal{S}\setminus\{s\to t\}} + 1 + k \cdot |T|_{Pre(s\to t)} \le 1 + (k+1) \cdot |T'|_{\mathcal{S}\setminus\{s\to t\}}$. So for any $u \in \mathcal{T}_B$ we have $Cplx_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}(u^\sharp) \le 1 + (k+1) \cdot Cplx_{\langle \mathcal{D}\setminus\{s\to t\},\mathcal{S}\setminus\{s\to t\},\mathcal{R}\rangle}(u^\sharp)$ and hence $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} = \iota_{\langle \mathcal{D}\setminus\{s\to t\},\mathcal{S}\setminus\{s\to t\},\mathcal{R}\rangle}$, which proves the soundness of the processor. $\square$

Note that a similar argument can also be used to remove whole SCCs without a successor. But since this is only possible if none of the DTs in the SCC is in $\mathcal{S}$, this is rarely useful in practice.

While the above processor only removes DTs that are leaves of the dependency graph, the following processor can also be used to simplify non-leaf DTs by removing subterms from their right-hand sides. More precisely, if $s \to \mathrm{C{\scriptsize OM}}_n(t_1,\ldots,t_n)$ is a DT where t_i never gives rise to edges in chain trees, then t_i can be removed from the right-hand side $\mathrm{C{\scriptsize OM}}_n(t_1, \ldots, t_n)$. In the following processor, for any set $\mathcal{S}$ of DTs, let $\mathcal{S}[s\to t \,/\, s\to t']$ denote the result of replacing $s \to t$ by $s \to t'$. So if $s\to t \in \mathcal{S}$, then $\mathcal{S}[s\to t \,/\, s\to t'] = (\mathcal{S} \setminus \{s\to t\}) \cup \{s\to t'\}$ and otherwise, $\mathcal{S}[s\to t \,/\, s\to t'] = \mathcal{S}$.

Theorem 32 (Rhs Simplification Processor) *Let $\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle$ be a DT problem and let $s \to t \in \mathcal{D}$ with $t = \mathrm{C{\scriptsize OM}}_n(t_1,\ldots,t_n)$. Let $I = \{i_1,\ldots,i_{n'}\}$ with $1 \le i_1 < \ldots < i_{n'} \le n$, such that*

I contains all i where $t_i\sigma \overset{\mathrm{i}}{\to}{}^*_{\mathcal{R}} u\tau$ holds for some $u \to v \in \mathcal{D}$ and some substitutions σ and τ where $s\sigma$ and $u\tau$ are in normal form w.r.t. $\mathcal{R}$. Let $t' = \textsc{Com}_{n'}(t_{i_1},\ldots,t_{i_{n'}})$, $\mathcal{D}' = \mathcal{D}[s \to t \;/\; s \to t']$, and $\mathcal{S}' = \mathcal{S}[s \to t \;/\; s \to t']$. Then the processor with $\textsc{Proc}(\,\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle\,) = (Pol_0, \langle \mathcal{D}',\mathcal{S}',\mathcal{R}\rangle)$ is sound.

Proof Consider an arbitrary $(\mathcal{D},\mathcal{R})$-chain tree T. We show that replacing every occurrence of $s \to t$ by $s \to t'$ in T yields a $(\mathcal{D}',\mathcal{R})$-chain tree T'. As $|T|_{\mathcal{S}} \le |T'|_{\mathcal{S}'}$, we have $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D}',\mathcal{S}',\mathcal{R}\rangle}$ and hence, the processor is sound.

Let $(s \to \textsc{Com}_n(t_1,\ldots,t_n) \mid \sigma)$ be a node in T with children $(u_1 \to v_1 \mid \tau_1)$, ..., $(u_k \to v_k \mid \tau_k)$. Then there exist pairwise different $i_1,\ldots,i_k \in \{1,\ldots,n\}$ with $t_{i_j}\sigma \overset{\mathrm{i}}{\to}{}^*_{\mathcal{R}} u_j\tau_j$ for all $j \in \{1,\ldots,k\}$. We have $i_1,\ldots,i_k \in I$ and hence, replacing $s \to \textsc{Com}_n(t_1,\ldots,t_n)$ by $s \to t'$ yields a $(\mathcal{D}',\mathcal{R})$-chain tree. $\square$

Example 33 To illustrate the leaf removal and rhs simplification processors, consider the DT problem $\langle\{(23),(24)\},\{(23),(24)\},\varnothing\rangle$ with the following dependency graph.

$$\begin{tikzcd}
\boxed{\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_3(\mathsf{f}^\sharp(x),\mathsf{g}^\sharp(x),\mathsf{h}^\sharp(x))\ (23)} \arrow[loop above] \arrow[r] & \boxed{\mathsf{g}^\sharp(x) \to \textsc{Com}_1(\mathsf{h}^\sharp(x))\ (24)}
\end{tikzcd}$$

As $Pre((24)) = \{(23)\} \subseteq \mathcal{S}$, the leaf removal processor removes (24) and returns $\langle\{(23)\}, \{(23)\},\varnothing\rangle$. The rhs simplification processor then removes the right-hand sides $\mathsf{g}^\sharp(x)$ and $\mathsf{h}^\sharp(x)$ from (23), which results in the DT problem $\langle\{(25)\},\{(25)\},\varnothing\rangle$ with

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_1(\mathsf{f}^\sharp(x)). \tag{25}$$

Note that the restrictions on $\mathcal{S}$ for the leaf removal processor are necessary for the soundness.[2] *Consider the variant $P = \langle\{(23),(24)\},\{(24)\},\varnothing\rangle$ of the DT problem above, which has a complexity of $\iota_P = Pol_1$. Here, neither $(24) \notin \mathcal{S}$ nor $Pre((24)) \subseteq \mathcal{S}$ holds. If we removed (24) nevertheless, we would end up with $\langle\{(23)\},\varnothing,\varnothing\rangle$, which has the complexity Pol_0.*

In [18, 19], instead of using techniques like the rhs simplification processor, the authors exploit dependency graphs by applying *path detection* (or *path analysis*) to simplify the complexity analysis. We also experimented with an additional processor performing path detection, but in our framework, this did not improve the analysis. The reason is that while path detection would allow the removal of certain DTs from $\mathcal{D}$, in most cases it does not harm to keep these DTs in $\mathcal{D}$. This is because our reduction pair processor can usually easily orient these DTs weakly by interpreting their tuple symbols $f^\sharp$ by constants $[f^\sharp]$.

Finally, the following processor removes DTs that cannot be "reached" by evaluations starting with basic terms. As a simple example consider a TRS with the rules $\mathsf{f}(\mathsf{s}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{g}(x))$ and $\mathsf{g}(\mathsf{a}) \to \mathsf{a}$. The DT $\mathsf{f}^\sharp(\mathsf{s}(\mathsf{g}(x))) \to \textsc{Com}_2(\mathsf{f}^\sharp(\mathsf{g}(x)),\mathsf{g}^\sharp(x))$ resulting from the first rule cannot occur in any chain tree for a term $s^\sharp$ with $s \in \mathcal{T}_B$, since its left-hand side contains the defined symbol g. Thus, the DT can be removed from the canonical DT problem. In this way, one can detect that this TRS has constant (and not linear) innermost runtime complexity.

Theorem 34 (Unreachable DT Removal Processor) *Let $P = \langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle$ be a DT problem. Let $\mathcal{M} \subseteq \mathcal{D}$ be the set of all DTs whose left-hand side does not contain any defined function symbol from Σ_d. Let $\mathcal{N} \subseteq \mathcal{D}$ be the set containing $\mathcal{M}$ and all DTs which are reachable from $\mathcal{M}$ in the $(\mathcal{D},\mathcal{R})$-dependency graph. Then the following processor is sound: $\textsc{Proc}(P) = (Pol_0, \langle \mathcal{N},\mathcal{S}\cap\mathcal{N},\mathcal{R}\rangle)$.*

[2] Thus, Thm. 31 is a corrected version of [27, Thm. 28], where these restrictions were missing.

Proof Let T be a $(\mathcal{D},\mathcal{R})$-chain tree for $s^\sharp$ where $s \in \mathcal{T}_B$. Then the root node of T is labeled with $(u \to v \mid \sigma)$ where $u\sigma = s^\sharp$. As $s^\sharp$ does not contain any defined symbols from Σ_d, we have $u \to v \in \mathcal{M}$.

By the definition of the dependency graph, all DTs in the nodes of T are reachable from $u \to v$ in the $(\mathcal{D},\mathcal{R})$-dependency graph. Hence, all these DTs are contained in $\mathcal{N}$.

Thus, $Cplx_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp) = Cplx_{\langle \mathcal{N},\mathcal{S}\cap\mathcal{N},\mathcal{R})\rangle}(s^\sharp)$, which implies the soundness of Proc. $\square$

5.4 Knowledge Propagation

In the DP framework for termination, the reduction pair processor removes "strictly decreasing" DPs. While this is unsound for complexity analysis (cf. Ex. 25), we now show that by an appropriate *extension* of DT problems, one can obtain a similar processor also for complexity analysis.

Lemma 35 shows that we can estimate the complexity of a DT problem if we know the complexity of all its *predecessors* in the dependency graph. The reason is that in any chain tree, the number of occurrences of a DT in the tree is bounded by the number of occurrences of its predecessors.

Lemma 35 (Complexity Bounded by Predecessors) *Let $\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle$ be a DT problem and $s \to t \in \mathcal{D}$. Then $\iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},Pre(s\to t),\mathcal{R}\rangle}$.*

Proof Let k be the maximal index of the compound symbols $\textsc{Com}_k$ occurring in $Pre(s \to t)$ and let T be a $(\mathcal{D},\mathcal{R})$-chain tree. Thus, we have $|T|_{\{s\to t\}} \le 1 + k \cdot |T|_{Pre(s\to t)}$, which implies $Cplx_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle}(u^\sharp) \le 1 + k \cdot Cplx_{\langle \mathcal{D},Pre(s\to t),\mathcal{R}\rangle}(u^\sharp)$ for any $u \in \mathcal{T}_B$. Hence, $\iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},Pre(s\to t),\mathcal{R}\rangle}$. $\square$

$$\begin{tikzcd}
\boxed{\mathsf{q}^\sharp(\mathsf{s}(x),\mathsf{s}(y),z) \to \textsc{Com}_1(\mathsf{q}^\sharp(x,y,z))\ (21)} \arrow[loop above] \arrow[d, bend right] \\
\boxed{\mathsf{q}^\sharp(x,\mathsf{0},\mathsf{s}(z)) \to \textsc{Com}_1(\mathsf{q}^\sharp(x,\mathsf{s}(z),\mathsf{s}(z)))\ (22)} \arrow[u, bend right]
\end{tikzcd}$$

Example 36 *Consider the TRS from Ex. 28. By usable rules and reduction pairs, we obtained $\langle \mathcal{D}, \{(22)\}, \varnothing\rangle$ for $\mathcal{D} = \{(20), (21), (22)\}$. The leaf removal processor yields $\langle \mathcal{D}', \{(22)\}, \varnothing\rangle$ with $\mathcal{D}' = \{(21), (22)\}$. Consider the the $(\mathcal{D}',\varnothing)$-dependency graph above. We have $\iota_{\langle \mathcal{D}',\{(22)\},\varnothing\rangle} \sqsubseteq \iota_{\langle \mathcal{D}',\{(21)\},\varnothing\rangle}$ by Lemma 35, since (21) is the only predecessor of (22). Thus, the complexity of $\langle \mathcal{D}', \{(22)\},\varnothing\rangle$ does not matter for the overall complexity, if we can guarantee that we have already taken the complexity of $\langle \mathcal{D}', \{(21)\},\varnothing\rangle$ into account.*

Therefore, we now extend the definition of DT problems by a set $\mathcal{K}$ of DTs with "known" complexity, i.e., the complexity of the DTs in $\mathcal{K}$ has already been taken into account. So a processor only needs to estimate the complexity of a set of DTs correctly if their complexity is higher than the complexity of the DTs in $\mathcal{K}$. Otherwise, the processor may return an arbitrary result. To this end, we introduce a "subtraction" operation $\ominus$ on complexities from $\mathfrak{C}$.

Definition 37 (Extended DT Problems, $\ominus$) For $c, d, \in \mathfrak{C}$, let $c \ominus d = c$ if $d \sqsubset c$ and $c \ominus d = Pol_0$ if $c \sqsubseteq d$. Let $\mathcal{R}$ be a TRS, $\mathcal{D}$ a set of DTs, and $\mathcal{S},\mathcal{K} \subseteq \mathcal{D}$. Then $\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle$ is an *extended DT problem* and $\langle DT(\mathcal{R}),DT(\mathcal{R}),\varnothing,\mathcal{R}\rangle$ is the *canonical extended DT problem* for $\mathcal{R}$.[3] We define the complexity of an extended DT problem to be $\gamma_{\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle} =$

[3] Note that we cannot always assume that $\mathcal{K} = \mathcal{D} \setminus \mathcal{S}$, i.e., that we have already taken the complexity of all DTs from $\mathcal{D} \setminus \mathcal{S}$ into account. In particular the narrowing processor of Thm. 43 can yield DT problems $\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle$ where $\mathcal{S} \cup \mathcal{K} \ne \mathcal{D}$.

$\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle}$ and also use γ instead of ι in the soundness condition for processors. So on extended DT problems, a processor with $\textsc{Proc}(P) = (c, P')$ is *sound* if $\gamma_P \sqsubseteq c \oplus \gamma_{P'}$. An extended DT problem $\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle$ is *solved* if $\mathcal{S} = \varnothing$.

So for $\mathcal{K} = \varnothing$, the definition of "complexity" for extended DT problems is equivalent to complexity for ordinary DT problems, i.e., $\gamma_{\langle \mathcal{D},\mathcal{S},\varnothing,\mathcal{R}\rangle} = \iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}$. Cor. 38 states that our approach is still correct for extended DT problems.

Corollary 38 (Correctness) *If P_0 is the canonical extended DT problem for a TRS $\mathcal{R}$ and $P_0 \overset{c_1}{\rightsquigarrow} \dots \overset{c_k}{\rightsquigarrow} P_k$ is a proof chain, then $\iota_{\mathcal{R}} = \gamma_{P_0} \sqsubseteq c_1 \oplus \dots \oplus c_k$.*

The following lemma shows the connection between $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}$ and $\ominus$.

Lemma 39 (Connection between $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}$ and $\ominus$)

(a) For any $c, d, e \in \mathfrak{C}$, we have $c \ominus d \sqsubseteq e$ iff $c \sqsubseteq d \oplus e$.

(b) For any $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{D}$, we have $\iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}_1 \setminus \mathcal{S}_2,\mathcal{R}\rangle}$.

Proof For (a), if $c \sqsubseteq d$, we have both $c \ominus d = Pol_0 \sqsubseteq e$ and $c \sqsubseteq d \sqsubseteq d \oplus e$. Otherwise, let $d \sqsubset c$. If $d \sqsubseteq e$, we have $c \ominus d = c \sqsubseteq e$ iff $c \sqsubseteq d \oplus e = e$. If $e \sqsubset d$, then $d \sqsubset c$ implies that $c \ominus d = c \sqsubseteq e$ is false. Similarly, then $c \sqsubseteq d \oplus e = d$ is also false.

For (b), we obtain $\iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{S}_2\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}_1 \setminus \mathcal{S}_2,\mathcal{R}\rangle}$ iff $\iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}_2,\mathcal{R}\rangle} \oplus \iota_{\langle \mathcal{D},\mathcal{S}_1 \setminus \mathcal{S}_2,\mathcal{R}\rangle}$ by (a). By Lemma 18 (a), this is equivalent to $\iota_{\langle \mathcal{D},\mathcal{S}_1,\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}_2 \cup (\mathcal{S}_1 \setminus \mathcal{S}_2),\mathcal{R}\rangle}$. As $\mathcal{S}_1 \subseteq \mathcal{S}_2 \cup (\mathcal{S}_1 \setminus \mathcal{S}_2)$, this is obviously true. □

Now we introduce a processor which makes use of $\mathcal{K}$. It moves a DT $s \to t$ from $\mathcal{S}$ to $\mathcal{K}$ whenever the complexity of all predecessors of $s \to t$ in the dependency graph has already been taken into account. So in particular, this means that nodes without predecessors (i.e., "roots" of the dependency graph that are not in any cycle) can always be moved from $\mathcal{S}$ to $\mathcal{K}$.

Theorem 40 (Knowledge Propagation Processor) *Let $\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle$ be an extended DT problem, $s \to t \in \mathcal{S}$, and $Pre(s \to t) \subseteq \mathcal{K}$. Then the following processor is sound:* $\textsc{Proc}(\langle \mathcal{D}, \mathcal{S},\mathcal{K},\mathcal{R}\rangle) = (Pol_0, \langle \mathcal{D}, \mathcal{S} \setminus \{s \to t\}, \mathcal{K} \cup \{s \to t\}, \mathcal{R}\rangle)$.

Proof We have to show $\gamma_{\langle \mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle} \sqsubseteq \gamma_{\langle \mathcal{D},\mathcal{S}\setminus\{s\to t\},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle}$. By the definition of γ, this is equivalent to

$$\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}\setminus\{s\to t\},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle}. \tag{26}$$

From Lemma 35, we have $\iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},Pre(\{s\to t\}),\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle}$. Hence, Lemma 18 (b) implies $\iota_{\langle \mathcal{D},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle} = \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle} \oplus \iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} = \iota_{\langle \mathcal{D},\mathcal{K},\mathcal{R}\rangle}$. Thus for (26), it suffices to show

$$\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}\setminus\{s\to t\},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle}. \tag{27}$$

We consider two cases: If $\iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} \sqsubset \iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}$, then $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} = \iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{S}\setminus\{s\to t\},\mathcal{R}\rangle}$ by Lemma 39 (b). Otherwise, $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\{s\to t\},\mathcal{R}\rangle} \sqsubseteq \iota_{\langle \mathcal{D},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle}$ and thus $\iota_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle} \ominus \iota_{\langle \mathcal{D},\mathcal{K}\cup\{s\to t\},\mathcal{R}\rangle} = Pol_0$. In both cases, the required inequality (27) follows. □

Before we can illustrate the knowledge propagation processor, we have to adapt the previous processors to *extended* DT problems. The adaptation of the usable rules, leaf removal, rhs simplification, and unreachable DT removal processors is straightforward. But now the reduction pair processor does not only delete DTs from $\mathcal{S}$, but moves them to $\mathcal{K}$. The reason is that the complexity of these DTs is bounded by the complexity value $c \in \mathfrak{C}$ returned by the processor. (Of course, the special case of the reduction pair processor with polynomial interpretations of Thm. 27 can be adapted analogously.)

Theorem 41 (Processors for Extended DT Problems) *Let $P = \langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle$ be an extended DT problem. Then the following processors are sound.*

- *The usable rules processor:* $\textsc{Proc}(P) = (Pol_0, \langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{U}_{\mathcal{R}}(\mathcal{D}) \rangle)$.
- *The leaf removal processor:* $\textsc{Proc}(P) = (Pol_0, \langle \mathcal{D} \setminus \{s \to t\}, \mathcal{S} \setminus \{s \to t\}, \mathcal{K} \setminus \{s \to t\}, \mathcal{R} \rangle)$ *if $s \to t$ is a leaf in the $(\mathcal{D}, \mathcal{R})$-dependency graph, and $s \to t \notin \mathcal{S}$ or $Pre(s \to t) \subseteq \mathcal{S} \cup \mathcal{K}$.*
- *The rhs simplification processor:* $\textsc{Proc}(P) = (Pol_0, \langle \mathcal{D}', \mathcal{S}', \mathcal{K}[s \to t \,/\, s \to t'], \mathcal{R} \rangle)$ *for $\mathcal{D}'$, $\mathcal{S}'$, $s \to t$, and t' as defined in Thm. 32.*
- *The unreachable DT removal processor:* $\textsc{Proc}(P) = (Pol_0, \langle \mathcal{N}, \mathcal{S} \cap \mathcal{N}, \mathcal{K} \cap \mathcal{N}, \mathcal{R} \rangle)$ *for $\mathcal{N}$ as defined in Thm. 34.*
- *The reduction pair processor:* $\textsc{Proc}(P) = (c,\ \langle \mathcal{D},\ \mathcal{S} \setminus \mathcal{D}_{\succ},\ \mathcal{K} \cup \mathcal{D}_{\succ},\ \mathcal{R} \rangle)$, *if $(\succsim, \succ)$ is a* $\textsc{Com}$*-monotonic reduction pair, $\mathcal{D} \subseteq \succsim \cup \succ$, $\mathcal{R} \subseteq \succsim$, and $c \sqsupseteq \iota(\mathrm{irc}_{\succ})$ for the function* $\mathrm{irc}_{\succ}(n) = \sup\{\, \mathrm{dh}(t^{\sharp}, \succ) \mid t \in \mathcal{T}_B, |t| \le n\}$.

Proof The soundness of the usable rules processor follows as in Thm. 23.

For the leaf removal processor, let $\mathcal{D}' = \mathcal{D} \setminus \{s \to t\}$, $\mathcal{S}' = \mathcal{S} \setminus \{s \to t\}$, and $\mathcal{K}' = \mathcal{K} \setminus \{s \to t\}$. Note that $\iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$ holds. We distinguish two cases.

- $s \to t \notin \mathcal{S}$: We have $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$, hence $\gamma_{\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle} \sqsubseteq \gamma_{\langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle}$ follows by Lemma 35.
- $Pre(s \to t) \subseteq \mathcal{S} \cup \mathcal{K}$: Remember that $\iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', Pre(s \to t), \mathcal{R} \rangle}$ holds. Thus, we have $\iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}' \cup \mathcal{K}', \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle} \oplus \iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle}$ by Lemma 18 (b) and hence, $\iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$ or $\iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle}$. First consider the case $\iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$. Then $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$ and hence, $\gamma_{\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle} \sqsubseteq \gamma_{\langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle}$. Otherwise, we have $\iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \not\sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$, i.e., $\iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle} \sqsubset \iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle}$ and therefore
$$\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle} \oplus \iota_{\langle \mathcal{D}, \{s \to t\}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}.$$
But then $\gamma_{\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle} = Pol_0$ and every processor is correct on such a problem.

The soundness of the rhs simplification and the unreachable DT removal processor follow as in Thm. 32 and 34, respectively.

For the reduction pair processor, we have to show $\gamma_P \sqsubseteq c \oplus \gamma_{\langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_{\succ}, \mathcal{K} \cup \mathcal{D}_{\succ}, \mathcal{R} \rangle}$. If $\gamma_P \sqsubseteq c$, then this is obviously true. Now consider $c \sqsubset \gamma_P$. We have to prove $\gamma_P \sqsubseteq \gamma_{\langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_{\succ}, \mathcal{K} \cup \mathcal{D}_{\succ}, \mathcal{R} \rangle}$, which means $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \ominus \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_{\succ}, \mathcal{R} \rangle} \ominus \iota_{\langle \mathcal{D}, \mathcal{K} \cup \mathcal{D}_{\succ}, \mathcal{R} \rangle}$. For this, it suffices to show (i) $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_{\succ}, \mathcal{R} \rangle}$ and (ii) $\iota_{\langle \mathcal{D}, \mathcal{K} \cup \mathcal{D}_{\succ}, \mathcal{R} \rangle} \sqsubset \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$.

We first prove (i). As $c \sqsubset \gamma_P$ implies $\gamma_P \neq Pol_0$, we have $\gamma_P = \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$ and therefore $c \sqsubset \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$. Moreover, from the proof of Thm. 26 we have $\iota_{\langle \mathcal{D}, \mathcal{D}_{\succ}, \mathcal{R} \rangle} \sqsubseteq \iota(\mathrm{irc}_{\succ}) \sqsubseteq c$. Hence, $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \ominus c \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \ominus \iota_{\langle \mathcal{D}, \mathcal{D}_{\succ}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{S} \setminus \mathcal{D}_{\succ}, \mathcal{R} \rangle}$, using Lemma 39 (b) for the last inequality.

Now we prove (ii). From Lemma 18 (b) and $\iota_{\langle \mathcal{D}, \mathcal{D}_{\succ}, \mathcal{R} \rangle} \sqsubseteq c$ we have
$$\iota_{\langle \mathcal{D}, \mathcal{K} \cup \mathcal{D}_{\succ}, \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle} \oplus \iota_{\langle \mathcal{D}, \mathcal{D}_{\succ}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle} \oplus c. \tag{28}$$
Note that $\gamma_P \neq Pol_0$ implies $\iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle} \sqsubset \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$. Together with $c \sqsubset \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$ this implies $\iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle} \oplus c \sqsubset \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}$ and hence (ii) follows with (28). □

Example 42 Reconsider the TRS $\mathcal{R}$ for division from Ex. 28. Starting with its canonical extended DT problem, we now obtain the following proof chain.

$$\begin{array}{lllll}
& \langle\{(20),(21),(22)\}, & \{(20),(21),(22)\}, & \varnothing, & \mathcal{R}\rangle & \\
\overset{Pol_0}{\rightsquigarrow} & \langle\{(21),(22)\}, & \{(21),(22)\}, & \varnothing, & \mathcal{R}\rangle & \textit{(leaf removal)} \\
\overset{Pol_0}{\rightsquigarrow} & \langle\{(21),(22)\}, & \{(21),(22)\}, & \varnothing, & \varnothing\rangle & \textit{(usable rules)} \\
\overset{Pol_1}{\rightsquigarrow} & \langle\{(21),(22)\}, & \{(22)\}, & \{(21)\}, & \varnothing\rangle & \textit{(reduction pair)} \\
\overset{Pol_0}{\rightsquigarrow} & \langle\{(21),(22)\}, & \varnothing, & \{(21),(22)\}, & \varnothing\rangle & \textit{(knowledge propag.)}
\end{array}$$

For the last step we use $Pre((22)) = \{(21)\}$, cf. Ex. 36. Note that the last DT problem is solved. Thus, $\iota_{\mathcal{R}} \sqsubseteq Pol_0 \oplus Pol_0 \oplus Pol_1 \oplus Pol_0 = Pol_1$, i.e., $\mathcal{R}$ has linear complexity.

5.5 Transformation Processors

To increase power, the DP framework for termination analysis has several processors which *transform* a DP into new ones (by "narrowing", "rewriting", "instantiation", or "forward instantiation") [12]. We now show how to adapt such processors for complexity analysis. For reasons of space, we only present the narrowing processor (the other processors can be adapted in a similar way).

For an extended DT problem $\langle\mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle$, let $s \to t \in \mathcal{D}$ with $t = \text{C}\small{\text{OM}}_n(t_1,\ldots,t_i,\ldots,t_n)$. For each t_i, we now define its *narrowing substitutions* and *narrowing results*. Note that if $s \to t$ is followed by another DT $u \to v$ in a chain, then there is a reduction $t_i\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} u\tau$. The idea of the narrowing processor is to perform the first step of this reduction already directly on the DT $s \to t$. Thus, whenever a subterm $t_i|_\pi \notin \mathcal{V}$ of t_i unifies with the left-hand side of a (variable-renamed) rule $\ell \to r \in \mathcal{R}$ using an mgu μ where $s\mu$ is in $\mathcal{R}$-normal form, then μ is called a *narrowing substitution* of t_i and the corresponding *narrowing result* is $w = t_i[r]_\pi\mu$.[4]

Moreover, if there exists a (variable-renamed) $u \to v \in \mathcal{D}$ where t_i and u have an mgu μ and both $s\mu$ and $u\mu$ are in $\mathcal{R}$-normal form, then the reduction $t_i\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} u\tau$ could also be performed in zero steps. So in this case, μ is an additional *narrowing substitution* of t_i and the corresponding *narrowing result* is $t_i\mu$.

If $\mu_1,\ldots,\mu_d$ are all narrowing substitutions of t_i with the corresponding narrowing results $w_1,\ldots,w_d$, then $s \to t$ can be replaced by $s\mu_j \to \text{C}\small{\text{OM}}_n(t_1\mu_j,\ldots,t_{i-1}\mu_j, w_j, t_{i+1}\mu_j,\ldots,t_n\mu_j)$ for all $1 \le j \le d$.

However, there could be a t_k (with $k \neq i$) which was involved in a chain (i.e., $t_k\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} u\tau$ for some $u \to v \in \mathcal{D}$ and some σ,τ), but this chain is no longer possible when instantiating t_k to $t_k\mu_1,\ldots,t_k\mu_d$. We say that t_k is *captured* by $\mu_1,\ldots,\mu_d$ if for each narrowing substitution ρ of t_k, there is a μ_j that is more general (i.e., $\rho = \mu_j\rho'$ for some substitution ρ'). The narrowing processor has to add another DT $s \to \text{C}\small{\text{OM}}_m(t_{k_1},\ldots,t_{k_m})$ where $t_{k_1},\ldots,t_{k_m}$ are all terms from $t_1,\ldots,t_n$ which are not captured by the narrowing substitutions $\mu_1,\ldots,\mu_d$ of t_i.

This leads to the following processor. Here, we extend the notation introduced in Thm. 32 in order to replace a DT $s \to t$ by a *set* $\mathcal{M}$ of DTs. For any sets $\mathcal{D},\mathcal{M}$ of DTs, $\mathcal{D}[s\to t \,/\, \mathcal{M}]$ denotes the result of replacing $s \to t$ by the DTs in $\mathcal{M}$. So if $s \to t \in \mathcal{D}$, then $\mathcal{D}[s\to t \,/\, \mathcal{M}] = (DT \setminus \{s\to t\}) \cup \mathcal{M}$ and otherwise, $\mathcal{D}[s\to t \,/\, \mathcal{M}] = \mathcal{D}$.

Theorem 43 (Narrowing Processor) *Let $P = \langle\mathcal{D},\mathcal{S},\mathcal{K},\mathcal{R}\rangle$ be an extended DT problem and let $s \to t \in \mathcal{D}$ with $t = \text{C}\small{\text{OM}}_n(t_1,\ldots,t_i,\ldots,t_n)$. Let $\mu_1,\ldots,\mu_d$ be the narrowing substitutions*

[4] This definition of "narrowing substitutions" and "results" is a corrected version of the one in [27].

of t_i with the corresponding narrowing results $w_1, \ldots, w_d$, where $d \geq 0$. Let $t_{k_1}, \ldots, t_{k_m}$ be the terms from $t_1, \ldots, t_n$ that are not captured by $\mu_1, \ldots, \mu_d$, where $k_1, \ldots, k_m$ are pairwise different. We define

$$\begin{aligned} \mathcal{M} = \quad & \{s\mu_j \to \text{COM}_n(t_1\mu_j, \ldots, t_{i-1}\mu_j, w_j, t_{i+1}\mu_j, \ldots, t_n\mu_j) \mid 1 \leq j \leq d\} \\ \cup \; & \{s \quad \to \text{COM}_m(t_{k_1}, \ldots, t_{k_m})\}. \end{aligned}$$

Then the following processor is sound: $\text{PROC}(P) = (Pol_0, \langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle)$*, where* $\mathcal{D}' = \mathcal{D}[s \to t \;/\; \mathcal{M}]$ *and* $\mathcal{S}' = \mathcal{S}[s \to t \;/\; \mathcal{M}]$*.* $\mathcal{K}'$ *results from* $\mathcal{K}$ *by removing* $s \to t$ *and all DTs that are reachable from* $s \to t$ *in the* $(\mathcal{D}, \mathcal{R})$*-dependency graph.*

Proof W.l.o.g. let $\mathcal{M}$ and $\mathcal{D}$ be disjoint (otherwise, we apply a variable renaming on one of them). Given a $(\mathcal{D}, \mathcal{R})$-chain tree T, we construct a $(\mathcal{D}', \mathcal{R})$-chain tree T' by repeatedly replacing every node of the form $(s \to t \mid \sigma)$ by a new node of the form $(s\mu \to t' \mid \sigma')$ with $s\mu \to t' \in \mathcal{M}$. This implies $|T|_{\{s \to t\}} = |T'|_{\mathcal{M}}$ and for any DT $u \to v \notin \{s \to t\} \cup \mathcal{M}$, we have $|T|_{\{u \to v\}} = |T'|_{\{u \to v\}}$. However, we have to show the following two statements in order to ensure that we still obtain a chain tree:

(A) *Relation to predecessor:* If $(s \to t \mid \sigma)$ was the root node of the chain tree for $s\sigma$, then the new node should also be the root node of a chain tree for $s\sigma$, i.e., we need $s\sigma = s\mu\sigma'$. Otherwise, if $(s \to t \mid \sigma)$ had a predecessor $(p \to \text{COM}_k(q_1, \ldots, q_k) \mid \rho)$ with $q_j\rho \xrightarrow{\text{i}}{}^*_{\mathcal{R}} s\sigma$, then the same relation should also hold for the new node $(s\mu \to t' \mid \sigma')$, i.e., we need $q_j\rho \xrightarrow{\text{i}}{}^*_{\mathcal{R}} s\mu\sigma'$. Note that this is obviously fulfilled if $s\sigma = s\mu\sigma'$.

(B) *Relation to successors:* Let $(s \to t \mid \sigma)$ have the children labeled with $(u_1 \to v_1 | \tau_1), \ldots, (u_e \to v_e | \tau_e)$ for $e \geq 0$. Hence, there exist pairwise different indexes $c_1, \ldots, c_e \in \{1, \ldots, n\}$ such that $t_{c_j}\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_j\tau_j$ for all $1 \leq j \leq e$. When replacing $(s \to t \mid \sigma)$ by a new node $(s\mu \to t' \mid \sigma')$ with $s\mu \to t' \in \mathcal{M}$, we have to show that there exist pairwise different indexes $c'_1, \ldots, c'_e$ such that $t'|_{c'_j}\sigma' \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_j\tau_j$ for all $1 \leq j \leq e$. Note that this is obviously fulfilled if for all j we have $t'|_{c'_j}\sigma' = t_{c_j}\sigma$.

We now distinguish three cases. For each of them, we show how to choose the new node $(s\mu \to t' \mid \sigma')$ such that the relations to the predecessor and to the successors in (A) and (B) still hold.

- Case 1: $i \in \{c_1, \ldots, c_e\}$ (i.e., t_i gave rise to a successor of $(s \to t \mid \sigma)$).
  Thus, there is a $1 \leq j_0 \leq e$ with $i = c_{j_0}$. Hence, $t_i\sigma = t_{c_{j_0}}\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_{j_0}\tau_{j_0}$.
  First regard the case where this reduction works in zero steps, i.e., $t_i\sigma = u_{j_0}\tau_{j_0}$. W.l.o.g., we can assume that u_{j_0} is variable-disjoint from t_i. Then t_i unifies with u_{j_0} using some mgu μ where $\sigma = \mu\sigma'$ and $\tau_{j_0} = \mu\tau'_{j_0}$ for some substitutions σ' and τ'_{j_0}. Since $(s \to t \mid \sigma)$ and $(u_{j_0} \to v_{j_0} \mid \tau_{j_0})$ are nodes in a chain tree, both $s\sigma$ and $u_{j_0}\tau_{j_0}$ are in $\mathcal{R}$-normal form. This implies that $s\mu$ and $u_{j_0}\mu$ are also in $\mathcal{R}$-normal form. Hence, t_i has the narrowing substitution μ with corresponding narrowing result $t_i\mu$. Thus, $s\mu \to t\mu \in \mathcal{M}$ and we can replace the node $(s \to t \mid \sigma)$ by $(s\mu \to t\mu \mid \sigma')$. For (A), we have $s\mu\sigma' = s\sigma$. For (B), we let $c'_j = c_j$ for all $1 \leq j \leq e$. Then we obtain $t'|_{c'_j}\sigma' = t'|_{c_j}\sigma' = t|_{c_j}\mu\sigma' = t|_{c_j}\sigma$, which implies (B).
  Otherwise, the reduction $t_i\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_{j_0}\tau_{j_0}$ takes at least one step. Let π be the position of $t_i\sigma$ where the first reduction step takes place and let $\ell \to r \in \mathcal{R}$ be the rule used in this step. We have $\pi \in Pos(t_i)$ and $t_i|_\pi \notin \mathcal{V}$, since the reduction cannot be "in σ". Otherwise, $s\sigma$ would not be an $\mathcal{R}$-normal form, due to $\mathcal{V}(t_i) \subseteq \mathcal{V}(s)$. W.l.o.g., we can assume that ℓ is

variable-disjoint from t_i. Then we can extend σ to the variables of ℓ such that $t_i|_\pi \sigma = \ell\sigma$ and

$$t_i\sigma \ = \ t_i[\ell]_\pi\sigma \xrightarrow{\text{i}}_{\mathcal{R}} \ t_i[r]_\pi\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_{j_0}\tau_{j_0}. \tag{29}$$

Since σ is a unifier of $t_i|_\pi$ and ℓ, they also have an mgu μ with $\sigma = \mu\sigma'$ for some substitution σ'. Moreover, since $s\sigma$ is in $\mathcal{R}$-normal form, $s\mu$ is in $\mathcal{R}$-normal form as well. Hence, μ is a narrowing substitution of t_i and the corresponding narrowing result is $t_i[r]_\pi\mu$.

Let $t' = \textsc{Com}_n(t_1, \ldots, t_{i-1}, t_i[r]_\pi, t_{i+1}, \ldots, t_n)\mu$. Then $s\mu \to t' \in \mathcal{M}$ and we replace the node $(s \to t \mid \sigma)$ by $(s\mu \to t' \mid \sigma')$. It remains to show that (A) and (B) hold.

(A) is satisfied since $\sigma = \mu\sigma'$ and hence, $s\sigma = s\mu\sigma'$. For (B), we let $c'_j = c_j$ for all $1 \leq j \leq e$. For j_0, we now obtain

$$t'|_{c'_{j_0}}\sigma' \ = \ t'|_{c_{j_0}}\sigma' \ = \ t_i[r]_\pi\mu\sigma' \ = \ t_i[r]_\pi\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_{j_0}\tau_{j_0}$$

by (29). For $j \neq j_0$, we have $t'|_{c'_j}\sigma' = t'|_{c_j}\sigma' = t|_{c_j}\mu\sigma' = t|_{c_j}\sigma$, so (B) follows.

- Case 2: $i \notin \{c_1, \ldots, c_e\}$ and none of $t_{c_1}, \ldots, t_{c_e}$ is captured by $\mu_1, \ldots, \mu_d$.

  Hence, $\{c_1, \ldots, c_e\} \subseteq \{k_1, \ldots, k_m\}$. We choose $s\mu \to t'$ to be $s \to \textsc{Com}_m(t_{k_1}, \ldots, t_{k_m})$ (i.e., μ is the identity) and we choose $\sigma' = \sigma$. This implies $s\sigma = s\mu\sigma'$ and thus, (A) holds. Moreover for every c_j, there exists an c'_j with $\textsc{Com}_m(t_{k_1}, \ldots, t_{k_m})|_{c'_j} = t_{c_j}$, since $c_j \in \{k_1, \ldots, k_m\}$. Thus, $t'|_{c'_j}\sigma' = \textsc{Com}_m(t_{k_1}, \ldots, t_{k_m})|_{c'_j}\sigma = t_{c_j}\sigma$, which proves (B).

- Case 3: $i \notin \{c_1, \ldots, c_e\}$ and at least one of $t_{c_1}, \ldots, t_{c_e}$ is captured by $\mu_1, \ldots, \mu_d$.

  Let $1 \leq j_0 \leq e$ such that $t_{c_{j_0}}$ is captured by $\mu_1, \ldots, \mu_d$. We have $t_{c_{j_0}}\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} u_{j_0}\tau_{j_0}$. As in Case 1, this implies that there exists a narrowing substitution μ of $t_{c_{j_0}}$ with $\sigma = \mu\tilde{\sigma}$ for some substitution $\tilde{\sigma}$. Since $t_{c_{j_0}}$ is captured by $\mu_1, \ldots, \mu_d$, there is a $1 \leq j_1 \leq d$ where μ_{j_1} is more general than μ, i.e., $\mu = \mu_{j_1}\overline{\sigma}$ for some substitution $\overline{\sigma}$. We define $\sigma' = \overline{\sigma}\tilde{\sigma}$ which implies $\sigma = \mu_{j_1}\sigma'$. Now we replace $(s \to t \mid \sigma)$ by $(s\mu_{j_1} \to t' \mid \sigma')$ where $t' = \textsc{Com}_n(t_1\mu_{j_1}, \ldots, t_{i-1}\mu_{j_1}, w_{j_1}, t_{i+1}\mu_{j_1}, \ldots, t_n\mu_{j_1})$. Then (A) holds, since $s\sigma = s\mu_{j_1}\sigma'$. For (B), we let $c'_j = c_j$ for all $1 \leq j \leq e$. Since $i \notin \{c_1, \ldots, c_e\}$, we obtain $t'|_{c'_j}\sigma' = t'|_{c_j}\sigma' = t_{c_j}\mu_{j_1}\sigma' = t_{c_j}\sigma$, which implies (B).

Thus, for any $(\mathcal{D}, \mathcal{R})$-chain tree T for a $w^\sharp \in \mathcal{T}^\sharp$ there exists a $(\mathcal{D}', \mathcal{R})$-chain tree T' for $w^\sharp$ where $|T|_{\{s \to t\}} = |T'|_{\mathcal{M}}$ and for any DT $u \to v \notin \{s \to t\} \cup \mathcal{M}$, we have $|T|_{\{u \to v\}} = |T'|_{\{u \to v\}}$. Hence, for any $w^\sharp \in \mathcal{T}^\sharp$ and any $\mathcal{S} \subseteq \mathcal{D}$ with $\mathcal{S}' = \mathcal{S}[s \to t \ / \ \mathcal{M}]$, we have $Cplx_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle}(w^\sharp) \leq Cplx_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}(w^\sharp)$. This implies $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$.

Moreover, if $\mathcal{K} \subseteq \mathcal{D}$ and $\mathcal{K}'$ results from $\mathcal{K}$ by removing $s \to t$ and all DTs that are reachable from $s \to t$ in the $(\mathcal{D}, \mathcal{R})$-dependency graph, then $\mathcal{K}'$ also contains no DT that is contained in $\mathcal{M}$ or reachable from $\mathcal{M}$ in the $(\mathcal{D}', \mathcal{R})$-dependency graph. Hence, for $Cplx_{\langle \mathcal{D}, \mathcal{K}', \mathcal{R} \rangle}(w^\sharp)$ or $Cplx_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle}(w^\sharp)$ it suffices to consider chain trees not containing $s \to t$ or DTs from $\mathcal{M}$. Such chain trees are both $(\mathcal{D}, \mathcal{R})$- and $(\mathcal{D}', \mathcal{R})$-chain trees. Hence, we obtain $Cplx_{\langle \mathcal{D}, \mathcal{K}', \mathcal{R} \rangle}(w^\sharp) = Cplx_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle}(w^\sharp)$ for all $w^\sharp \in \mathcal{T}^\sharp$ and thus, $\iota_{\langle \mathcal{D}, \mathcal{K}', \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle}$. As $\mathcal{K}' \subseteq \mathcal{K}$, we have $\iota_{\langle \mathcal{D}, \mathcal{K}', \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$ and hence $\iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$.

From $\iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle}$ and $\iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$, we obtain $\gamma_P \sqsubseteq \gamma_{\langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle}$, i.e., the narrowing processor is sound. □

Example 44 To illustrate the narrowing processor, consider the following TRS.

$$\mathsf{f}(\mathsf{c}(n,x)) \to \mathsf{c}(\mathsf{f}(\mathsf{g}(\mathsf{c}(n,x))), \mathsf{f}(\mathsf{h}(\mathsf{c}(n,x)))) \qquad \mathsf{g}(\mathsf{c}(\mathsf{0},x)) \to x \qquad \mathsf{h}(\mathsf{c}(\mathsf{1},x)) \to x$$

So f *operates on "lists" of* $\mathsf{0}$*s and* $\mathsf{1}$*s, where* g *removes a leading* $\mathsf{0}$ *and* h *removes a leading* $\mathsf{1}$*. Since* g*'s and* h*'s applicability "exclude" each other, the TRS has linear (and not exponential) complexity. The leaf removal, rhs simplification, and the usable rules processor give the problem* $\langle\, \{(30)\}, \{(30)\}, \varnothing, \{\mathsf{g}(\mathsf{c}(\mathsf{0},x)) \to x,\, \mathsf{h}(\mathsf{c}(\mathsf{1},x)) \to x\}\,\rangle$ *with*

$$\mathsf{f}^\sharp(\mathsf{c}(n,x)) \to \textsc{Com}_2(\mathsf{f}^\sharp(\mathsf{g}(\mathsf{c}(n,x))), \mathsf{f}^\sharp(\mathsf{h}(\mathsf{c}(n,x)))) \tag{30}$$

The only narrowing substitution of $t_1 = \mathsf{f}^\sharp(\mathsf{g}(\mathsf{c}(n,x)))$ *is* $[n/\mathsf{0}]$ *and the corresponding narrowing result is* $\mathsf{f}^\sharp(x)$*. However,* $t_2 = \mathsf{f}^\sharp(\mathsf{h}(\mathsf{c}(n,x)))$ *is not captured by the substitution* $[n/\mathsf{0}]$*, since* $[n/\mathsf{0}]$ *is not more general than* t_2*'s narrowing substitution* $[n/\mathsf{1}]$*. Hence, the DT* (30) *is replaced by the following two new DTs:*

$$\mathsf{f}^\sharp(\mathsf{c}(\mathsf{0},x)) \to \textsc{Com}_2(\mathsf{f}^\sharp(x),\, \mathsf{f}^\sharp(\mathsf{h}(\mathsf{c}(\mathsf{0},x)))) \tag{31}$$

$$\mathsf{f}^\sharp(\mathsf{c}(n,x)) \to \textsc{Com}_1(\mathsf{f}^\sharp(\mathsf{h}(\mathsf{c}(n,x)))) \tag{32}$$

By applying the narrowing processor again, we replace (32) *by* $\mathsf{f}^\sharp(\mathsf{c}(\mathsf{1},x)) \to \textsc{Com}_1(\mathsf{f}^\sharp(x))$ *and* $\mathsf{f}^\sharp(\mathsf{c}(n,x)) \to \textsc{Com}_0$*. One can also simplify (31) further to* $\mathsf{f}^\sharp(\mathsf{c}(\mathsf{0},x)) \to \textsc{Com}_1(\mathsf{f}^\sharp(x))$ *by the rhs simplification processor. Now* $\iota_{\mathcal{R}} \sqsubseteq Pol_1$ *is easy to show by the reduction pair processor.*

Example 45 *Reconsider the TRS of Ex. 4. The canonical extended DT problem is transformed to* $\langle \mathcal{D}_1, \mathcal{D}_1, \varnothing, \mathcal{R}_1\rangle$*, where* $\mathcal{D}_1 = \{(1),(2),(8)\}$ *and* $\mathcal{R}_1$ *are the* gt*- and* p*-rules, cf. Ex. 30. In* $\mathsf{m}^\sharp(x,y) \to \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y), \mathsf{gt}^\sharp(x,y))$ (1)*, one can narrow* $t_1 = \mathsf{if}^\sharp(\mathsf{gt}(x,y),x,y)$*. Its narrowing substitutions are* $[x/\mathsf{0}, y/k]$*,* $[x/\mathsf{s}(n), y/\mathsf{0}]$*, and* $[x/\mathsf{s}(n), y/\mathsf{s}(k)]$*. Note that* $t_2 = \mathsf{gt}^\sharp(x,y)$ *is captured, as its only narrowing substitution is* $[x/\mathsf{s}(n), y/\mathsf{s}(k)]$*. So* (1) *can be replaced by*

$$\mathsf{m}^\sharp(\mathsf{0},k) \to \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{false},\mathsf{0},k), \mathsf{gt}^\sharp(\mathsf{0},k)) \tag{33}$$

$$\mathsf{m}^\sharp(\mathsf{s}(n),\mathsf{0}) \to \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),\mathsf{0}), \mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{0})) \tag{34}$$

$$\mathsf{m}^\sharp(\mathsf{s}(n),\mathsf{s}(k)) \to \textsc{Com}_2(\mathsf{if}^\sharp(\mathsf{gt}(n,k),\mathsf{s}(n),\mathsf{s}(k)), \mathsf{gt}^\sharp(\mathsf{s}(n),\mathsf{s}(k))) \tag{35}$$

$$\mathsf{m}^\sharp(x,y) \to \textsc{Com}_0 \tag{36}$$

The leaf removal processor deletes (33) *and* (36)*, and the rhs simplification processor simplifies* (34) *to*

$$\mathsf{m}^\sharp(\mathsf{s}(n),\mathsf{0}) \to \textsc{Com}_1(\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),\mathsf{0})) \tag{37}$$

This yields $\langle \mathcal{D}_2, \mathcal{D}_2, \varnothing, \mathcal{R}_1\rangle$ *with* $\mathcal{D}_2 = \{(37),(35),(2),(8)\}$*. Now the narrowing processor replaces* $\mathsf{if}^\sharp(\mathsf{true},x,y) \to \textsc{Com}_2(\mathsf{m}^\sharp(\mathsf{p}(x),y), \mathsf{p}^\sharp(x))$ (2) *by*

$$\mathsf{if}^\sharp(\mathsf{true},\mathsf{0},y) \to \textsc{Com}_2(\mathsf{m}^\sharp(\mathsf{0},y), \mathsf{p}^\sharp(\mathsf{0})) \tag{38}$$

$$\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),y) \to \textsc{Com}_2(\mathsf{m}^\sharp(n,y), \mathsf{p}^\sharp(\mathsf{s}(n))) \tag{39}$$

$$\mathsf{if}^\sharp(\mathsf{true},x,y) \to \textsc{Com}_0 \tag{40}$$

The leaf removal processor deletes (38) *and* (40)*, and the rhs simplification processor transforms* (39) *into*

$$\mathsf{if}^\sharp(\mathsf{true},\mathsf{s}(n),y) \to \textsc{Com}_1(\mathsf{m}^\sharp(n,y)) \tag{41}$$

Now the usable rule processor removes the p*-rules from* $\mathcal{R}_1$*. This yields* $\langle \mathcal{D}_3, \mathcal{D}_3, \varnothing, \mathcal{R}_2\rangle$*, where* $\mathcal{D}_3 = \{(37),(35),(41),(8)\}$ *and* $\mathcal{R}_2$ *are the* gt*-rules. By the polynomial interpretation*

$[\mathsf{0}] = [\mathsf{true}] = [\mathsf{false}] = 0$, $[\mathsf{s}](x) = x+2$, $[\mathsf{gt}](x,y) = [\mathsf{gt}^\sharp](x,y) = x$, $[\mathsf{m}^\sharp](x,y) = (x+1)^2$, $[\mathsf{if}^\sharp](x,y,z) = y^2$, all DTs in $\mathcal{D}_3$ are strictly decreasing and all rules in $\mathcal{R}_2$ are weakly decreasing. So the reduction pair processor yields $\langle \mathcal{D}_3, \mathcal{D}_3, \varnothing, \mathcal{R}_2 \rangle \overset{Pol_2}{\leadsto} \langle \mathcal{D}_3, \varnothing, \mathcal{D}_3, \mathcal{R}_2 \rangle$. As this DT problem is solved, we obtain $\iota_{\mathcal{R}} \sqsubseteq Pol_0 \oplus \ldots \oplus Pol_0 \oplus Pol_2 = Pol_2$, i.e., $\mathcal{R}$ has quadratic complexity.

Note that in Thm. 43, we cannot define $\mathcal{K}' = \mathcal{K}[s \to t \;/\; \mathcal{M}]$, because the narrowing step performed on $s \to t$ does not necessarily correspond to an *innermost* reduction. Hence, there can be $(\mathcal{D}', \mathcal{R})$-chains that correspond to non-innermost reductions with $\mathcal{D} \cup \mathcal{R}$. So there may exist terms whose maximal $(\mathcal{D}', \mathcal{R})$-chain tree is larger than their maximal $(\mathcal{D}, \mathcal{R})$-chain tree and thus, $\iota_{\langle \mathcal{D}', \mathcal{K}[s \to t / \mathcal{M}], \mathcal{R} \rangle} \sqsupseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$. But we need $\iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle}$ in order to guarantee the soundness of the processor, i.e., to ensure that $\gamma_{\langle \mathcal{D}, \mathcal{S}, \mathcal{K}, \mathcal{R} \rangle} = \iota_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle} \ominus \iota_{\langle \mathcal{D}, \mathcal{K}, \mathcal{R} \rangle} \sqsubseteq \iota_{\langle \mathcal{D}', \mathcal{S}', \mathcal{R} \rangle} \ominus \iota_{\langle \mathcal{D}', \mathcal{K}', \mathcal{R} \rangle} = \gamma_{\langle \mathcal{D}', \mathcal{S}', \mathcal{K}', \mathcal{R} \rangle}$. For the same reason, applying the narrowing processor can increase the complexity of a DT problem. This is demonstrated in the following example.

Example 46 *Let $\mathcal{R} = \{\mathsf{g}(\mathsf{h}(x)) \to x, \mathsf{h}(\mathsf{s}(x)) \to \mathsf{0}\}$ and let $\mathcal{D}$ be the following set:*

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_3(\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x))), \mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x))), \mathsf{g}^\sharp(x)) \tag{42}$$

$$\mathsf{g}^\sharp(\mathsf{c}(x)) \to \textsc{Com}_1(\mathsf{g}^\sharp(x)) \tag{43}$$

Consider the DT problem $P = \langle \mathcal{D}, \{(43)\}, \{(42)\}, \mathcal{R} \rangle$. This problem has the following dependency graph.

$$\begin{tikzcd}\boxed{(42)} \arrow[r] & \boxed{(43)} \arrow[loop above]\end{tikzcd}$$

Note that there is no loop from (42) to itself, as no instance of $\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x)))$ has an innermost reduction to a term of the form $\mathsf{f}^\sharp(\mathsf{s}(\ldots))$. Therefore, this problem has the complexity $\gamma_P = Pol_1$. By narrowing, we can replace (42) by

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_3(\mathsf{f}^\sharp(x), \mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x))), \mathsf{g}^\sharp(x)) \tag{44}$$

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_0, \tag{45}$$

where (45) is deleted by the leaf removal processor. If we had defined $\mathcal{K}' = \mathcal{K}[s \to t \;/\; \mathcal{M}]$ in Thm. 43, we would result in $P' = \langle \{(44), (43)\}, \{(43)\}, \{(44)\}, \mathcal{R} \rangle$. But P' has the complexity $\gamma_{P'} = Pol_0$ and hence the narrowing processor would not be sound. With the correct definition, the result is $P'' = \langle \{(44), (43)\}, \{(43)\}, \varnothing, \mathcal{R} \rangle$ which still has linear complexity. Narrowing again replaces (44) by

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \textsc{Com}_3(\mathsf{f}^\sharp(x), \mathsf{f}^\sharp(x), \mathsf{g}^\sharp(x)) \tag{46}$$

and (45) (which is again deleted by leaf removal). Now the resulting problem $P''' = \langle \{(46), (43)\}, \{(43)\}, \varnothing, \mathcal{R} \rangle$ has exponential complexity. Thus, the narrowing processor does not always preserve complexity.

		CaT					
		Pol_0	Pol_1	Pol_2	Pol_3	no result	Σ
AProVE	Pol_0	-	199	-	-	144	343
	Pol_1	-	96	10	-	95	201
	Pol_2	-	15	-	-	67	82
	Pol_3	-	-	-	-	13	13
	no result	-	6	2	1	601	610
	Σ	0	316	12	1	920	1249

Fig. 2 AProVE vs. CaT

		TCT					
		Pol_0	Pol_1	Pol_2	Pol_3	no result	Σ
AProVE	Pol_0	1	209	8	-	125	343
	Pol_1	-	128	19	-	54	201
	Pol_2	-	24	17	-	41	82
	Pol_3	-	-	-	-	13	13
	no result	-	29	13	-	568	610
	Σ	1	390	57	0	801	1249

Fig. 3 AProVE vs. TCT

6 Evaluation and Conclusion

We presented a new technique for innermost runtime complexity analysis by adapting the termination techniques of the DP framework. To this end, we introduced several processors to simplify "DT problems", which gives rise to a flexible and modular framework for automated complexity proofs. Thus, recent advances in termination analysis can now also be used for complexity analysis.

To evaluate our contributions, we implemented them in the termination prover AProVE and compared it with the complexity tools CaT 1.5 [30] and TCT [2]. As suggested by the authors of TCT, we used its development version id ccf74e291a. We ran the tools on 1249 TRSs from the category "*Runtime Complexity – Innermost Strategy*" of the *Termination Problem Data Base* used in the full run of the *International Termination Competition* 2012. For more information on the termination competition, see `http://www.termination-portal.org/wiki/Termination_Competition`. We omitted the 60 TRSs which contain rules with extra variables on the right-hand side, since they are trivially non-terminating. As in the competition, each tool had a time limit of 60 seconds for each example. Fig. 2 compares CaT and AProVE. For instance, the first row means that AProVE showed constant complexity for 343 examples.[5] On those examples, CaT proved linear complexity in 199 cases and failed in 144 cases. So in the light gray part of the table, AProVE gave more precise results than CaT. In the medium gray part, both tools obtained equal results. In the dark gray part, CaT was more precise than AProVE. Similarly, Fig. 3 compares TCT and AProVE.

So AProVE showed polynomial innermost runtime for 639 of the 1249 examples (51 %). (Note that the collection also contains many examples whose complexity is not polynomial.) In contrast, CaT resp. TCT proved polynomial innermost runtime for 329 (26 %) resp. 448 (36 %) examples. Even a "combined tool" of CaT and TCT (which always returns the better result of these two tools) would only show polynomial runtime for 501 examples (40 %). Hence, our contributions represent a significant advance. This also confirms the results of the *Termination Competition 2010 – 2012*, where AProVE won the category of innermost

[5] More precisely, the table gives the numbers of TRSs $\mathcal{R}$ where the respective tool proved that $\iota_{\mathcal{R}}$ is *at most* Pol_k.

Processor	No. of Applications
reduction pair processor	660
leaf removal processor	448
unreachable DT removal processor	389
rhs simplification processor	263
narrowing processor	225
knowledge propagation processor	73
match-bounds	31
instantiation processor	13
rewriting processor	9
forward instantiation processor	5

Fig. 4 Impact of different DT processors

runtime complexity analysis. (In contrast to CaT and TCT, AProVE did not participate in any other complexity categories as it cannot analyze derivational or non-innermost runtime complexity.) AProVE also succeeds on Ex. 4, 28, and 44, whereas CaT fails on them and TCT fails on Ex. 4 and 44. (TCT can determine linear complexity for Ex. 28, and Ex. 25 can be analyzed by all three tools.)

Fig. 4 examines the impact of the different DT processors. For each processor, the table states how often it was applied in (successful) proofs (i.e., in proofs where AProVE could infer polynomial innermost runtime within the time limit). Of course, some processors (like the reduction pair processor) were applied several times within the same proof. Here, we used a reduction pair processor with polynomial interpretations which was combined with the usable rule processor. In this way, one can use the refinement of [12] to compute usable rules w.r.t. argument filterings (i.e., when determining the usable rules one does not have to consider those positions which are "filtered away" by the reduction pair). The instantiation, rewriting, and forward instantiation processors are analogous to the narrowing processor (i.e., they adapt the techniques of instantiating, rewriting, and forward instantiating dependency pairs from [12] to complexity analysis). Finally, we also integrated the approach of [10] to use match-bounds for complexity analysis. This approach is particularly useful for string rewrite systems, and AProVE applied it to infer complexity bounds for 31 of the 1249 examples in the collection.

The proof strategy used in AProVE was to first simplify the initial DT problem by the unreachable DT removal, rhs simplification, leaf removal, and knowledge propagation processors. Afterwards, we try to apply the reduction pair processor with polynomial orders and (in parallel, although delayed by 6 seconds) the transformation processors. For the transformation processors, we attempt narrowing first and then try the other transformations in parallel. A successful application of the reduction pair processor is always followed by knowledge propagation, a successful application of a transformation processor is followed by a simplification of the remaining problem. If this strategy does not succeed in 45 seconds, we also try to apply the match-bounds approach in parallel.

The motivation for this strategy is as follows: unreachable DT removal, rhs simplification, leaf removal, and knowledge propagation are fast (no search is needed) and they obviously simplify the problem whenever they are applicable. Hence, they are used as early and as often as possible. In particular, we apply them again after transformation processors, as these processors may change the structure of the dependency graph (and hence lead to new opportunities for simplification). The reduction pair processor needs to search for a suitable reduction order and hence might be slow. However, if applicable, the reduction pair processor always simplifies the problem, so we prefer it to the transformation processors. As shown in Ex. 46, transformation processors do not always preserve the complexity of

		AProVE without rhs simplification and unreachable DT removal processor					
		Pol_0	Pol_1	Pol_2	Pol_3	no result	Σ
full AProVE	Pol_0	131	117	15	1	79	343
	Pol_1	2	171	12	-	16	201
	Pol_2	-	7	60	1	14	82
	Pol_3	-	-	1	7	5	13
	no result	-	-	1	-	609	610
	Σ	133	295	89	9	723	1249

Fig. 5 Impact of the rhs simplification and the unreachable DT removal processor

a DT problem (the transformed problem might have a larger complexity than the original one) and, as for termination analysis, transformation processors are potentially applicable infinitely often, cf. [12].

To evaluate the usefulness of the rhs simplification processor and the unreachable DT processor (which are new compared to the preliminary version of the current paper in [27]), we also tested the full version of AProVE against a variant where we disabled these two processors. As shown in the table of Fig. 5, this variant is substantially weaker. While the full version of AProVE showed polynomial innermost runtime for 639 examples (51 %), this restricted variant only succeeded for 526 TRSs (42 %). In particular, the new processors help to detect more examples with constant complexity. (That there are also a few examples where the restricted variant obtained more precise complexity bounds is due to the heuristics and internal time-limits of AProVE, which determine its strategy to apply the different processors.)

For details on our experiments and to run our implementation in AProVE via a web interface, we refer to `http://aprove.informatik.rwth-aachen.de/eval/Complexity/`.

Acknowledgments. We are grateful to the CaT and the TCT team for their support with the experiments and to G. Moser and H. Zankl for many helpful comments. Moreover, we thank the anonymous referees for many useful remarks and for detecting an error in the previous definition of "narrowing substitutions", cf. Footnote 4.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. M. Avanzini, G. Moser, and A. Schnabl. Automated implicit computational complexity analysis. In *Proc. IJCAR '08*, LNAI 5195, pages 132–138, 2008.
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. Closing the gap between runtime complexity and polytime computability. In *Proc. RTA '10*, LIPIcs 6, pages 33–48, 2010.
5. M. Avanzini and G. Moser. Complexity analysis by graph rewriting. In *Proc. FLOPS '10*, LNCS 6009, pages 257–271, 2010.
6. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
7. 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.
8. M. Brockschmidt, R. Musiol, C. Otto, and J. Giesl. Automated termination proofs for Java programs with cyclic data. In *Proc. CAV '12*, LNCS, pages 105–122, 2012.
9. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.
10. A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Information and Computation*, 205(4):512–534, 2007.
11. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. LPAR '04*, LNAI 3452, pages 301–331, 2005.
12. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.

13. J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM Transactions on Programming Languages and Systems*, 33(2), 2011.
14. J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. Symbolic evaluation graphs and term rewriting – A general methodology for analyzing logic programs. In *Proc. PPDP '12*, pages 1–12. ACM Press, 2012.
15. R. Givan and D. A. McAllester. Polynomial-time computation via local inference relations. *ACM Transactions on Computational Logic*, 3(4):521–541, 2002.
16. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
17. N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR '08*, LNAI 5195, pages 364–379, 2008.
18. N. Hirokawa and G. Moser. Complexity, graphs, and the dependency pair method. In *Proc. LPAR '08*, LNAI 5330, pages 652–666, 2008.
19. N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. *CoRR*, abs/1102.3129, 2011.
20. D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. RTA '89*, LNCS 355, pages 167–177, 1989.
21. M. Hofmann. Linear types and non-size-increasing polynomial time computation. In *Proc. LICS '99*, pages 464–473. IEEE Press, 1999.
22. A. Koprowski and J. Waldmann. Max/plus tree automata for termination of term rewriting. *Acta Cybernetica*, 19(2):357–392, 2009.
23. J.-Y. Marion and R. Péchoux. Characterizations of polynomial complexity classes with a better intensionality. In *Proc. PPDP '08*, pages 79–88. ACM Press, 2008.
24. G. Moser, A. Schnabl, and J. Waldmann. Complexity analysis of term rewriting based on matrix and context dependent interpretations. In *Proc. FSTTCS '08*, LIPIcs 2, pages 304–315, 2008.
25. G. Moser. Personal communication, 2010.
26. F. Neurauter, H. Zankl, and A. Middeldorp. Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In *Proc. LPAR '10*, LNCS 6397, pages 550–564, 2010.
27. L. Noschinski, F. Emmes, and J. Giesl. A dependency pair framework for innermost complexity analysis of term rewrite systems. In *Proc. CADE '11*, LNAI 6803, pages 422–438, 2011.
28. C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. RTA '10*, LIPIcs 6, pp. 259–276, 2010.
29. J. Waldmann. Polynomially bounded matrix interpretations. In *Proc. RTA '10*, LIPIcs 6, pages 357–372, 2010.
30. H. Zankl and M. Korp. Modular complexity analysis via relative complexity. In *Proc. RTA '10*, LIPIcs 6, pages 385–400, 2010.