A Combination Framework for Complexity*

Martin Avanzini[1] and Georg Moser[1]

1 **Institute of Computer Science,**
University of Innsbruck, Austria
`{martin.avanzini,georg.moser}@uibk.ac.at`

Abstract

In this paper we present a combination framework for the automated polynomial complexity analysis of term rewrite systems. The framework covers both *derivational* and *runtime complexity* analysis, and is employed as theoretical foundation in the automated complexity tool TCT. We present generalisations of powerful complexity techniques, notably a generalisation of *complexity pairs* and *(weak) dependency pairs*. Finally, we also present a novel technique, called *dependency graph decomposition*, that in the dependency pair setting greatly increases modularity.

1998 ACM Subject Classification F.1.3, F.3.2, F.4.1, F.4.2

Keywords and phrases Program Analysis, Term Rewriting, Complexity Analysis, Automation

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

In order to measure the complexity of a term rewrite system (TRS for short) it is natural to look at the maximal length of derivation sequences—the *derivation length*—as suggested by Hofbauer and Lautemann in [15]. The resulting notion of complexity is called *derivational complexity*. Hirokawa and the second author introduced in [12] a variation, called *runtime complexity*, that only takes *basic* or *constructor-based* terms as start terms into account. The restriction to basic terms allows one to accurately express the complexity of a program through the runtime complexity of a TRS. Noteworthy both notions constitute an *invariant cost model* for rewrite systems [10, 4].

The body of research in the field of complexity analysis of rewrite systems provides a wide range of different techniques to analyse the time complexity of rewrite systems, fully automatically. Techniques range from *direct methods*, like *polynomial path orders* [3, 5] and other suitable restrictions of termination orders [9, 20], to *transformation techniques*, maybe most prominently adaptions of the *dependency pair method* [12, 14, 21], *semantic labeling* over finite carriers [2], methods to combine base techniques [24] and the *weight gap principle* [12, 24]. (See [19] for an overview of complexity analysis methods for term rewrite systems.) In particular the dependency pair method for complexity analysis allows for a wealth of techniques originally intended for termination analysis. We mention *(safe) reduction pairs* [12, 14], *various rule transformations* [21], and *usable rules* [12, 14]. Some very effective methods have been introduced specifically for complexity analysis in the context of dependency pairs. For instance, *path analysis* [12, 13, 14] decomposes the analysed rewrite relation into simpler ones, by treating paths through the *dependency graph* independently. *Knowledge propagation* [21] is another complexity technique relying on dependency graph analysis, which allows one to propagate bounds for specific rules along the dependency graph.

* This work was partially supported by FWF (Austrian Science Fund) project I-603-N18.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Besides these, various minor simplifications are implemented in tools, mostly relying on dependency graph analysis. With this paper, we provide following contributions.

1. We propose a uniform *combination framework for complexity analysis*, that is capable of expressing the majority of the rewriting based complexity techniques in a unified way. Such a framework is essential for the development of a modern complexity analyser for term rewrite systems. The implementation of our complexity analyser TCT [7], the *Tyrolean Complexity Tool*, closely follows the formalisation proposed in this work. Noteworthy, TCT is currently the only tool that participates in all four complexity sub-divisions of the annual *termination competition*.[1]
2. A majority of the cited techniques were introduced in restricted or incompatible contexts. For instance, in [24] the derivational complexity of relative TRSs is considered. Conversely, neither [12, 14] nor [21] treat relative systems, and restrict their attention to basic start terms. Where non-obvious, we generalise these techniques to our setting. Noteworthy, our notion of *$\mathcal{P}$-monotone complexity pair* generalises complexity pairs from [24] for derivational complexity, *μ-monotone complexity pairs* for runtime complexity analysis [14], and *safe reduction pairs* studied in [12, 21] that work on dependency pairs.[2] We also generalise the two different forms of dependency pairs for complexity analysis introduced in [12] and [21]. This for instance allows our tool TCT to employ these powerful techniques on a TRS $\mathcal{R}$ relative to some theory expressed as a TRS $\mathcal{S}$.
3. We introduce a novel proof technique for runtime-complexity analysis called *dependency graph decomposition*. Resulting sub-problems are syntactically of a simpler form, and the analysis of these sub-problems is often easier. Importantly, the sub-problems are usually also computationally simpler in the sense that their complexity is strictly smaller than the one of the input problem. If the complexity of the two generated sub-problems is bounded by a function in $\mathsf{O}(f)$ and $\mathsf{O}(g)$ respectively, then the complexity of the input is bounded by $\mathsf{O}(f \cdot g)$. Experiments conducted with TCT indicate that this estimation is often asymptotically precise.[3]

This paper is structured as follows. In the next section we cover some basics. Our *combination framework* is then introduced in Section 3. In Section 4 we introduce *$\mathcal{P}$-monotone complexity pairs*. In Section 5 we introduce *dependency pairs for complexity analysis*, and reprove soundness of weak dependency pairs and dependency tuples. In Section 6 we introduce *dependency graph decomposition*, and conclude in Section 7.

Due to space limitations some proofs are only sketched, or have been completely omitted. The reader is kindly referred to the technical report [6], where proofs are given in full detail.

2 Preliminaries

Let R be a binary relation. The transitive closure of R is denoted by R^+ and its transitive and reflexive closure by R^*. For $n \in \mathbb{N}$ we denote by R^n the *n-fold composition of R*. The binary relation R is *well-founded* (on a set A) if there exists no infinite chain $a_0, a_1, \dots$ with $a_i \mathrel{R} a_{i+1}$ for all $i \in \mathbb{N}$ ($a_0 \in A$). The relation R is *finitely branching* if for all elements a, the set $\{b \mid a \mathrel{R} b\}$ is finite. A *preorder* is a reflexive and transitive binary relation.

[1] `http://www.termination-portal.org/wiki/Termination_Competition/`.
[2] In [21] safe reductions pairs are called COM-*monotone reduction pairs*.
[3] Detailed experimental evidence is provided online under `http://cl-informatik.uibk.ac.at/software/tct/experiments/tct2`.

We assume familiarity with rewriting [8] and just fix notations. We denote by $\mathcal{V}$ a countably infinite set of *variables* and by $\mathcal{F}$ a *signature*. The signature $\mathcal{F}$ and variables $\mathcal{V}$ are fixed throughout the paper, the set of terms over $\mathcal{F}$ and $\mathcal{V}$ is written as $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Throughout the following, we suppose a partitioning of $\mathcal{F}$ into *constructors* $\mathcal{C}$ and *defined symbols* $\mathcal{D}$. The set of *basic terms* $f(s_1, \ldots, s_n)$, where $f \in \mathcal{D}$ and arguments s_i $(i = 1, \ldots, n)$ contain only variables or constructors, is denoted by $\mathcal{T}_\mathsf{b}$. Terms are denoted by $s, t, \ldots$, possibly followed by subscripts. We use $s|_p$ to refer to the *subterm* of s at position p. We denote by $|t|$ the *size* of t, i.e., the number of occurrences of symbols in t. A *rewrite relation* $\to$ is a binary relation on terms closed under contexts and stable under substitutions. We use $\mathcal{R}, \mathcal{S}, \mathcal{Q}, \mathcal{W}$ to refer to *term rewrite systems* (*TRSs* for short). We denote by $\mathrm{NF}(\mathcal{R})$ the *normal forms* of $\mathcal{R}$, and abusing notation we extend this notion to binary relations $\to$ on terms in the obvious way. For a set of terms $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, we define $\to(T) := \{t \mid \exists s \in T.\ s \to t\}$.

For two TRSs $\mathcal{Q}$ and $\mathcal{R}$, we define $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ if there exists a context C, substitution σ, and rule $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$ such that $s = C[f(l_1\sigma, \ldots, l_n\sigma)]$, $t = C[r\sigma]$ and all arguments $l_i\sigma$ $(i = 1, \ldots, n)$ are $\mathcal{Q}$ normal forms. If $\mathcal{Q} = \varnothing$, we sometimes drop $\mathcal{Q}$ and write $\to_{\mathcal{R}}$ instead of $\xrightarrow{\varnothing}_{\mathcal{R}}$. Note that $\to_{\mathcal{R}}$ corresponds to the usual definition of rewrite relation of $\mathcal{R}$. The innermost rewrite relation of a TRS $\mathcal{R}$ is given by $\xrightarrow{\mathcal{R}}_{\mathcal{R}}$. We extend $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to a relative setting and define for TRSs $\mathcal{R}$ and $\mathcal{S}$ the relation $\xrightarrow{\mathcal{Q}}_{\mathcal{R}/\mathcal{S}} := \xrightarrow{\mathcal{Q}}^*_{\mathcal{S}} \cdot \xrightarrow{\mathcal{Q}}_{\mathcal{R}} \cdot \xrightarrow{\mathcal{Q}}^*_{\mathcal{S}}$, and call $\xrightarrow{\mathcal{Q}}_{\mathcal{R}/\mathcal{S}}$ the *$\mathcal{Q}$-restricted rewrite relation of $\mathcal{R}$ modulo $\mathcal{S}$*.

To compare partial functions we use *Kleene equality*: two partial functions $f, g \colon \mathbb{N} \to \mathbb{N}$ are equal, in notation $f \simeq g$, if for all $n \in \mathbb{N}$ either $f(n)$ and $g(n)$ are defined and $f(n) = g(n)$, or both $f(n)$ and $g(n)$ are undefined. The *derivation height* of a term t with respect to a binary relation $\to$ on terms is given by $\mathsf{dh}(t, \to) \simeq \max\{n \mid \exists t_1, \ldots, t_n.\ t \to t_1 \to \cdots \to t_n\}$. We emphasise that $\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{R}/\mathcal{S}})$, if defined, binds the number of $\mathcal{R}$ steps in all $\xrightarrow{\mathcal{Q}}_{\mathcal{R}\cup\mathcal{S}}$ derivations starting from t. We emphasise that our techniques always imply that $\mathsf{dh}(t, \to)$ is well-defined. Let T be a set of terms, and define $\mathsf{cp}(n, T, \to) := \max\{\mathsf{dh}(t, \to) \mid \exists t \in T,\ |t| \leqslant n\}$. The *derivational complexity* of a TRS $\mathcal{R}$ is given by $\mathrm{dc}_{\mathcal{R}}(n) := \mathsf{cp}(n, \mathcal{T}(\mathcal{F}, \mathcal{V}), \to_{\mathcal{R}})$ for all $n \in \mathbb{N}$, the *runtime complexity* takes only basic terms as starting terms T into account: $\mathrm{rc}_{\mathcal{R}}(n) := \mathsf{cp}(n, \mathcal{T}_\mathsf{b}, \to_{\mathcal{R}})$ for all $n \in \mathbb{N}$. By exchanging $\to_{\mathcal{R}}$ with $\xrightarrow{\mathcal{R}}_{\mathcal{R}}$ we obtain the notions of *innermost derivational* or *runtime complexity* respectively.

3 The Combination Framework

At the heart of our framework lies the notion of *complexity processor*, or simply *processor*. A complexity processor dictates how to transform the analysed input *problem* into sub-problems (if any), and how to relate the complexity of the obtained sub-problems to the complexity of the input problem. In our framework, such a processor is modeled as a set of inference rules

$$\frac{\vdash \mathcal{P}_1 \colon f_1 \quad \cdots \quad \vdash \mathcal{P}_n \colon f_n}{\vdash \mathcal{P} \colon f}\ ,$$

over judgements of the form $\vdash \mathcal{P} \colon f$. Here $\mathcal{P}$ denotes a *complexity problem* (*problem* for short) and $f \colon \mathbb{N} \to \mathbb{N}$ a *bounding function*. The validity of a judgement $\vdash \mathcal{P} \colon f$ is given when the function f binds the complexity of the problem $\mathcal{P}$ asymptotically.

Conceptually, a complexity problem $\mathcal{P}$ consists of a set of *starting terms* $\mathcal{T}$ together with a relation $\xrightarrow{\mathcal{Q}}_{\mathcal{S}\cup\mathcal{W}}$ for TRSs $\mathcal{S}, \mathcal{W}$ and $\mathcal{Q}$. The complexity function $\mathsf{cp}_{\mathcal{P}} \colon \mathbb{N} \to \mathbb{N}$ of $\mathcal{P}$ accounts for the number of applications of rules from $\mathcal{S}$ in derivations starting from terms $t \in \mathcal{T}$, measured in the size of t.

▶ **Definition 3.1** (Complexity Problem, Complexity Function).

1. A *complexity problem* $\mathcal{P}$ (*problem* for short) is a quadruple $\langle \mathcal{S}, \mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$, in notation $\langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$, where $\mathcal{S}, \mathcal{W}, \mathcal{Q}$ are TRSs and $\mathcal{T} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ a set of terms.
2. The *complexity (function)* $\mathsf{cp}_{\mathcal{P}} \colon \mathbb{N} \to \mathbb{N}$ of $\mathcal{P}$ is defined as the partial function

$$\mathsf{cp}_{\mathcal{P}}(n) := \mathsf{cp}(n, \mathcal{T}, \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}) .$$

In the sequel $\mathcal{P}$, possibly followed by subscripts, always denotes a complexity problem. Consider a problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$. We call $\mathcal{S}$ and $\mathcal{W}$ the *strict* and *weak component* of $\mathcal{P}$ respectively. The set $\mathcal{T}$ is called the set of *starting terms* of $\mathcal{P}$. We sometimes write $l \to r \in \mathcal{P}$ for $l \to r \in \mathcal{S} \cup \mathcal{W}$, and we denote by $\to_{\mathcal{P}}$ the rewrite relation $\xrightarrow{\mathcal{Q}}_{\mathcal{S} \cup \mathcal{W}}$. A derivation $t \to_{\mathcal{P}} t_1 \to_{\mathcal{P}} \cdots$ is also called a $\mathcal{P}$-*derivation* (*starting from* t). Observe that the derivational complexity of a TRS $\mathcal{R}$ corresponds to the complexity function of $\langle \mathcal{R}/\varnothing, \varnothing, \mathcal{T}(\mathcal{F}, \mathcal{V}) \rangle$. By exchanging the set of starting terms to basic terms we can express the *runtime complexity* of a TRS $\mathcal{R}$. If the starting terms are all basic terms, we call such a problem also a *runtime complexity problem.* Likewise, we can treat innermost rewriting by using $\mathcal{Q} = \mathcal{R}$. For the case $\mathrm{NF}(\mathcal{Q}) \subseteq \mathrm{NF}(\mathcal{S} \cup \mathcal{W})$, that is when $\to_{\mathcal{P}}$ is included in the innermost rewrite relation of $\mathcal{R} \cup \mathcal{S}$, we also call $\mathcal{P}$ an *innermost complexity problem.*

▶ **Example 3.2.** Consider the rewrite system $\mathcal{R}_\times$ given by the four rules

$$1\colon 0 + y \to y \qquad 2\colon \mathsf{s}(x) + y \to \mathsf{s}(x + y) \qquad 3\colon 0 \times y \to 0 \qquad 4\colon \mathsf{s}(x) \times y \to (x \times y) + y \ ,$$

and let $\mathcal{T}_\mathsf{b}$ denote basic terms with defined symbols $+, \times$ and constructors $\mathsf{s}, 0$. Then $\mathcal{P}_\times := \langle \mathcal{R}_\times/\varnothing, \mathcal{R}_\times, \mathcal{T}_\mathsf{b} \rangle$ is an innermost runtime complexity problem, in particular the complexity of $\mathcal{P}$ equals the innermost runtime complexity of $\mathcal{R}_\times$.

Note that even if $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ is terminating, the complexity function is not necessarily defined on all inputs. For a counter example, consider the problem $\mathcal{P}_1 := \langle \mathcal{S}_1/\mathcal{W}_1, \varnothing, \{\mathsf{f}(\bot)\} \rangle$ where $\mathcal{S}_1 := \{\mathsf{g}(\mathsf{s}(x)) \to \mathsf{g}(x)\}$ and $\mathcal{W}_1 := \{\mathsf{f}(x) \to \mathsf{f}(\mathsf{s}(x)),\ \mathsf{f}(x) \to \mathsf{g}(x)\}$. Note that for all $n \in \mathbb{N}$, maximal $\to_{\mathcal{P}_1}$ derivations are of the form

$$\mathsf{f}(\bot) \to^*_{\mathcal{W}_1} \mathsf{f}(\mathsf{s}^n(\bot)) \to_{\mathcal{W}_1} \mathsf{g}(\mathsf{s}^n(\bot)) \to^n_{\mathcal{S}_1} \mathsf{g}(\bot) .$$

Hence $\mathsf{f}(\bot) \to^n_{\mathcal{S}_1/\mathcal{W}_1} \mathsf{g}(\bot)$ holds for all $n \in \mathbb{N}$. Whereas $\to_{\mathcal{S}_1/\mathcal{W}_1}$ is well-founded, the above family of derivations shows that $\mathsf{cp}_{\mathcal{P}_1}(m) \simeq \mathsf{dh}(\mathsf{f}(\bot), \to_{\mathcal{S}_1/\mathcal{W}_1})$ is undefined for $m \geqslant 2$. If $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ is well-founded and *finitely branching* then $\mathsf{cp}_{\mathcal{P}}$ is defined on all inputs, by König's Lemma. This condition is sufficient but not necessary. The complexity function of the problem $\mathcal{P}_2 := \langle \mathcal{S}_2/\mathcal{W}_1, \varnothing, \{\mathsf{f}(\bot)\} \rangle$, where $\mathcal{S}_2 := \{\mathsf{g}(x) \to x\}$, is constant but $\mathsf{f}(\bot) \to_{\mathcal{S}_2/\mathcal{W}_1} \mathsf{s}^n(\bot)$ for all $n \in \mathbb{N}$, i.e, $\to_{\mathcal{S}_2/\mathcal{W}_1}$ is not finitely branching. In this work we do not presuppose that the complexity function is defined on all inputs, instead, this will be determined by our methods.

▶ **Definition 3.3** (Judgement, Processor, Proof)**.**

1. A *(complexity) judgment* is a statement $\vdash \mathcal{P} \colon f$ where $\mathcal{P}$ is a complexity problem and $f \colon \mathbb{N} \to \mathbb{N}$. The judgment is *valid* if $\mathsf{cp}_{\mathcal{P}}$ is defined on all inputs, and $\mathsf{cp}_{\mathcal{P}} \in \mathsf{O}(f)$.
2. A *complexity processor* Proc (*processor* for short) is an inference rule

$$\frac{\vdash \mathcal{P}_1 \colon f_1 \quad \cdots \quad \vdash \mathcal{P}_n \colon f_n}{\vdash \mathcal{P} \colon f}\ \mathsf{Proc}\ ,$$

over complexity judgements. The problems $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are called the *sub-problems generated by* Proc *on* $\mathcal{P}$. The processor Proc is *sound* if $\vdash \mathcal{P} \colon f$ is valid whenever the statements $\vdash \mathcal{P}_1 \colon f_1, \ldots, \vdash \mathcal{P}_n \colon f_n$ are valid. The processor is *complete* if the inverse direction holds.

3. Let empty denote the axiom $\vdash \langle \varnothing/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle \colon f$ for all TRSs $\mathcal{W}$ and $\mathcal{Q}$, set of terms $\mathcal{T}$ and $f \colon \mathbb{N} \to \mathbb{N}$. A *complexity proof* (*proof* for short) of a judgement $\vdash \mathcal{P} \colon f$ is a deduction using sound processors from the axiom empty and *assumptions* $\vdash \mathcal{P}_1 \colon f_1, \dots, \vdash \mathcal{P}_n \colon f_n$, in notation $\mathcal{P}_1 \colon f_1, \dots, \mathcal{P}_n \colon f_n \vdash \mathcal{P} \colon f$.

We say that a complexity proof is *closed* if its set of assumptions is empty, otherwise it is *open*. We follow the usual convention and annotate side conditions as premises to inference rules. As stated in the next lemma, soundness of a processor guarantees our formal system is correct. Completeness ensures that a deduction gives asymptotically tight bounds.

▶ **Lemma 3.4.** *If there exists a closed complexity proof $\vdash \mathcal{P} \colon f$, then the judgement $\vdash \mathcal{P} \colon f$ is valid.*

4 Suiting Reduction Orders to Complexity

Maybe the most obvious tools for complexity analysis in rewriting are *reduction orders*, in particular *interpretations*. Consequently these have been used quite early for complexity analysis. For instance, in [9] *polynomial interpretations* are used in a direct setting in order to estimate the runtime complexity analysis of a TRS. On the other hand in [24] *complexity pairs*, that constitute of a reduction order and a corresponding preorder, are employed to estimate the derivational complexity in a relative setting. Relaxing monotonicity requirements on complexity pairs gives rise to a notion of *reduction pair*, so called *safe reduction pairs* [12], that can be used to estimate the runtime complexity of dependency pair problems, cf. [14, 21]. In the following, we introduce *$\mathcal{P}$-monotone complexity pairs*, that give a unified account of the orders given in [9, 24, 14, 21].

We fix a complexity problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$. Consider a proper order $\succ$ on terms, and let $\mathsf{G} \colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathbb{N}$. Suppose that $\mathsf{G}(s) > \mathsf{G}(t)$ holds whenever $s \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t$ and $s \succ t$ holds, for all terms s reachable from $t \in \mathcal{T}$ with a $\mathcal{P}$-derivation ($s \in \to^*_{\mathcal{P}}(\mathcal{T})$). Then $\succ$ is called G-*collapsible* (on $\mathcal{P}$). If in addition $\mathsf{G}(t)$ is asymptotically bounded by a function $f \colon \mathbb{N} \to \mathbb{N}$ in the size of t for all start terms $t \in \mathcal{T}$, i.e., $\mathsf{G}(t) \in \mathsf{O}(f(|t|))$ for $t \in \mathcal{T}$, we say that $\succ$ *induces* the complexity f on $\mathcal{P}$. In particular polynomial and matrix interpretations [8, 16] are collapsible, and also *recursive path order* [8] are. All these termination techniques have been suitable tamed so that the induced complexity is a polynomial [9, 18, 3].

Consider an order $\succ$ that induces the complexity f on $\mathcal{P}$. If this order includes the relation $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$, the judgement $\vdash \mathcal{P} \colon f$ is valid. To check the inclusion, as in [24] we consider a pair of orders $(\succsim, \succ)$ where the preorder $\succsim$ and the order $\succ$ are compatible in the sense that $\succsim \cdot \succ \cdot \succsim \subseteq \succ$ holds. It is obvious that when both orders are monotone and stable under substitutions, the assertions $\mathcal{W} \subseteq \succsim$ and $\mathcal{S} \subseteq \succ$ imply $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} \subseteq \succ$ as desired. Guided by the observation that monotonicity is required only on argument positions that can be rewritten in reductions of starting terms, Hirokawa and the second author [14] propose the use of *μ-monotone orders* for runtime complexity analysis. Initially introduced [25] for termination analysis of *context sensitive rewrite systems* [17], the parameter μ denotes a *replacement map*, i.e., a map that assigns to every n-ary function symbol $f \in \mathcal{F}$ a subset of its argument positions: $\mu(f) \subseteq \{1, \dots, n\}$. In the realm of context sensitive rewriting this map governs under which argument positions a rewrite step is allowed, here μ is used to designate which arguments are *usable* for a set of rules $\mathcal{R}$ in $\mathcal{P}$-derivations, i.e., can be rewritten by a rule $l \to r \in \mathcal{R}$ in $\mathcal{P}$-derivations starting from $t \in \mathcal{T}$.

Denote by $\mathcal{P}\mathrm{os}_\mu(t)$ the *μ-replacing positions* in t, defined as $\mathcal{P}\mathrm{os}_\mu(t) := \{\epsilon\}$ if t is a variable, and $\mathcal{P}\mathrm{os}_\mu(t) := \{\epsilon\} \cup \{i{\cdot}p \mid i \in \mu(f) \text{ and } p \in \mathcal{P}\mathrm{os}_\mu(t_i)\}$ for $t = f(t_1, \dots, t_n)$. For a binary relation $\to$ on terms we denote by $\mathcal{T}_\mu(\to)$ the set of terms t where subterms at non-μ-replacing positions are in normal form: $t \in \mathcal{T}_\mu(\to)$ if for all positions p in t, if $p \notin \mathcal{P}\mathrm{os}_\mu(t)$ then $t|_p \in \mathrm{NF}(\to)$.

▶ **Definition 4.1.** Let $\mathcal{P}$ be a complexity problem with starting terms $\mathcal{T}$ and let $\mathcal{R}$ denote a set of rewrite rules. A replacement map μ is called a *usable replacement map* for $\mathcal{R}$ in $\mathcal{P}$, if $\to^*_{\mathcal{P}}(\mathcal{T}) \subseteq \mathcal{T}_\mu(\xrightarrow{\mathcal{Q}}_{\mathcal{R}})$.

Put otherwise, μ denotes a usable replacement map for $\mathcal{R}$ in $\mathcal{P}$ if for any rewrite step $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ with $s \in \to^*_{\mathcal{P}}(\mathcal{T})$ the rewrite position p is μ-replacing. It is undecidable to determine if μ is a usable replacement map for rules $\mathcal{R}$ in $\mathcal{P}$. Exploiting that for runtime complexity starting terms are basic, in [14] good approximations for full and innermost rewriting are given.

▶ **Example 4.2** (Example 3.2 continued)**.** Consider the $\mathcal{P}_\times$-derivation

$$\underline{\mathbf{2} \times \mathbf{1}} \to_{\mathcal{P}_\times} (\underline{\mathbf{1} \times \mathbf{1}})+\mathbf{1} \to_{\mathcal{P}_\times} ((\underline{\mathbf{0} \times \mathbf{1}})+\mathbf{1})+\mathbf{1} \to_{\mathcal{P}_\times} (\underline{\mathbf{0} + \mathbf{1}})+\mathbf{1} \to_{\mathcal{P}_\times} \mathsf{s}(\underline{\mathbf{0} + \mathbf{0}})+\mathbf{1} \to_{\mathcal{P}_\times} \mathbf{1}+\mathbf{1} \ ,$$

where redexes are underlined. Here, and also in consecutive examples, we use the notation $\mathbf{n}$ for the numeral $\mathsf{s}(\cdots(\mathsf{s}(0))\cdots)$ with $n \in \mathbb{N}$ occurrences of the constructor s. Observe that if multiplication occurs in a context, then only under the first argument position of addition. This holds even for all $\mathcal{P}_\times$-derivations of basic terms. The map $\mu_\times$, defined by $\mu_\times(+) = \{1\}$ and $\mu_\times(\times) = \mu_\times(\mathsf{s}) = \varnothing$, thus constitutes a usable replacement map for the multiplication rules $\{3, 4\}$ in $\mathcal{P}_\times$. Since the argument position of s is not usable in $\mu_\times$, the last step witnesses that $\mu_\times$ does not designate a usable replacement map for the addition rules $\{1, 2\}$.

We say that an order $\succ$ is *μ-monotone* if it is monotone on μ positions, in the sense that for all function symbols f, if $i \in \mu(f)$ and $s_i \succ t_i$ then $f(s_1, \dots, s_i, \dots, s_n) \succ f(s_1, \dots, t_i, \dots, s_n)$ holds. The next intermediate lemma follows by a standard induction on the rewrite position, and is central to the definition of *$\mathcal{P}$-monotone complexity pair* defined below.

▶ **Lemma 4.3.** *Let μ be a usable replacement map for $\mathcal{R}$ in $\mathcal{P}$, and let $\succ$ denote a μ-monotone order that is stable under substitutions. If $\mathcal{R} \subseteq \succ$ holds, i.e., rewrite rules in $\mathcal{R}$ are oriented from left to right, then $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ implies $s \succ t$ for all terms $s \in \to^*_{\mathcal{P}}(\mathcal{T})$.*

▶ **Definition 4.4** (Complexity Pair, $\mathcal{P}$-monotone)**.**
1. A *complexity pair* is a pair $(\succsim, \succ)$, such that $\succsim$ is a stable preorder and $\succ$ a stable order with $\succsim \cdot \succ \cdot \succsim \subseteq \succ$.
2. Suppose $\succsim$ is $\mu_{\mathcal{W}}$-monotone for a usable replacement map of $\mathcal{W}$ in $\mathcal{P}$, and likewise $\succ$ is $\mu_{\mathcal{S}}$-monotone for a usable replacement map of $\mathcal{S}$ in $\mathcal{P}$. Then $(\succsim, \succ)$ is called *$\mathcal{P}$-monotone*.

▶ **Lemma 4.5.** *Consider a $\mathcal{P}$-monotone complexity pair $(\succsim, \succ)$ such that the order $\succ$ is G-collapsible on $\mathcal{P}$. Further, suppose that $(\succsim, \succ)$ is compatible with $\mathcal{P}$ in the sense that $\mathcal{W} \subseteq \succsim$ and $\mathcal{S} \subseteq \succ$ hold. Then $s \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t$ implies $s \succ t$ for all terms $s \in \to^*_{\mathcal{P}}(\mathcal{T})$. In particular, $\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}) \leqslant \mathsf{G}(t)$ for all $t \in \mathcal{T}$.*

Proof. Consider a $\mathcal{Q}$-restricted relative step $s \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{W}} \cdot \xrightarrow{\mathcal{Q}}_{\mathcal{S}} \cdot \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{W}} t$ for $s \in \to^*_{\mathcal{P}}(\mathcal{T})$. Using the assumptions on $(\succsim, \succ)$ and the inclusions $\mathcal{W} \subseteq \succsim$ and $\mathcal{S} \subseteq \succ$ to satisfy the assumptions of Lemma 4.3, we obtain $s \succsim^* \cdot \succ \cdot \succsim^* t$. Hence $s \succ t$ follows by transitivity of $\succsim$ and the inclusion $\succsim \cdot \succ \cdot \succsim \subseteq \succ$.

As a consequence, every rewrite sequence $t = t_0 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} \cdots$ for $t \in \mathcal{T}$ translates to $\mathsf{G}(t) = \mathsf{G}(t_0) > \mathsf{G}(t_1) > \cdots$, thus $\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}})$ is defined and bounded by $\mathsf{G}(t)$. ◀

As immediate consequence of this lemma, we obtain our first processor.

▶ **Theorem 4.6** (Complexity Pair Processor). *Let $(\succsim, \succ)$ be a $\mathcal{P}$-monotone complexity pair such that $\succ$ induces the complexity f on $\mathcal{P}$. The following processor is sound:*

$$\frac{\mathcal{S} \subseteq \succ \quad \mathcal{W} \subseteq \succsim}{\vdash \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle \colon f} \;\mathsf{CP}\;.$$

When the set of starting terms is unrestricted only the full replacement map is usable for rules of $\mathcal{P}$. In this case, our notion of complexity pairs collapses to the one given by Zankl and Korp [24]. We emphasise that in contrast to [14], our notion of complexity pair is parameterised in separate replacements for $\succsim$ and $\succ$. By this separation we can restate *(safe) reduction pairs* originally proposed in [12], employed in the dependency pair setting below, as instances of complexity pairs (cf. Lemma 5.9).

A variation of the complexity pair processor, that iteratively orients disjoint subsets of $\mathcal{S}$, occurred first in [24]. The following processor constitutes a straight forward generalisation of [24, Theorem 4.4] to our setting.

▶ **Theorem 4.7** (Decompose Processor [24]). *The following processor is sound:*

$$\frac{\vdash \langle \mathcal{S}_1/\mathcal{S}_2 \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle \colon f \qquad \vdash \langle \mathcal{S}_2/\mathcal{S}_1 \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle \colon g}{\vdash \langle \mathcal{S}_1 \cup \mathcal{S}_2/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle \colon f + g} \;\mathsf{decompose}\;.$$

Proof. The lemma follows as $\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}_1 \cup \mathcal{S}_2/\mathcal{W}}) \leqslant \mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}_1/\mathcal{S}_2 \cup \mathcal{W}}) + \mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}_2/\mathcal{S}_1 \cup \mathcal{W}})$. ◀

In combination with for instance complexity pairs, the decompose processor allows as in [24] *shifting* of rules from the strict to the weak component. This is demonstrated in the following proof, that was automatically found by our complexity prover $\mathsf{T_CT}$.

▶ **Example 4.8** (Examples 3.2 and 4.2 continued). Consider the linear polynomial interpretation $\mathcal{A}$ over $\mathbb{N}$ such that $0_{\mathcal{A}} = 0$, $\mathsf{s}_{\mathcal{A}}(x) = x + 1$, $x +_{\mathcal{A}} y = x + y$ and $x \times_{\mathcal{A}} y = x \cdot y + x^2$. Let $\mathcal{P}_4 := \langle \{4\}/\{1, 2, 3\}, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}\rangle$ denote the problem that accounts for the rules $4\colon\; \mathsf{s}(x) \times y \to (x \times y) + y$ in $\mathcal{P}_\times$. The induced order $>_{\mathcal{A}}$ together with its reflexive closure $\geqslant_{\mathcal{A}}$ forms a $\mathcal{P}_4$-monotone complexity pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ that induces quadratic complexity on $\mathcal{P}_4$. The following depicts a complexity proof $\langle \{1, 2, 3\}/\{4\}, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}\rangle \colon g \vdash \mathcal{P}_\times \colon n^2 + g$.

$$\frac{\dfrac{\{4\} \subseteq >_{\mathcal{A}} \quad \{1,2,3\} \subseteq \geqslant_{\mathcal{A}}}{\vdash \langle \{4\}/\{1,2,3\}, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}\rangle \colon n^2} \;\mathsf{CP} \qquad \vdash \langle \{1,2,3\}/\{4\}, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}\rangle \colon g}{\vdash \mathcal{P}_\times \colon n^2 + g} \;\mathsf{decompose}\;.$$

The above complexity proof can now be completed iteratively, on the simpler problem $\langle \{1, 2, 3\}/\{4\}, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}\rangle$. Since the complexity of $\mathcal{P}_\times$ is cubic, one has to use a technique beyond quadratic polynomial interpretations here. We remark that the decompose processor finds applications beyond its combination with complexity pairs, for instance $\mathsf{T_CT}$ uses this processor to separation independent components by analysing the *dependency graph* [14].

5 Dependency Pair Processors

The introduction of *dependency pairs* (*DPs* for short) [1], and its formalisation in the *dependency pair framework* [23], drastically increased power and modularity in termination provers. It is well established that the DP method is unsuitable for complexity analysis. The induced complexity is simply too high [22], in the sense that the complexity of $\mathcal{R}$ is not suitably reflected in its canonical DP problem. Hirokawa and the second author [12] recover this deficiency with the introduction of *weak dependency pairs*. Crucially, weak dependency pairs group different function calls in right-hand sides, using *compound symbols*.

In this section, we first introduce a notion of *dependency pair complexity problem* (*DP problem* for short), a specific instance of a complexity problem. In Theorem 5.8 and Theorem 5.12 we then introduce the *weak dependency pair* and *dependency tuples* processors, that construct from a runtime complexity problem its canonical DP problem. We emphasise that both processors are conceptually *not* new, weak dependency pairs were introduced in [12], and dependency tuples in [21]. Here, we establish a simulation that also accounts for relative rewrite steps, consequently our processors provide a generalisations of [12, 21].

Consider a signature $\mathcal{F}$ that is partitioned into defined symbols $\mathcal{D}$ and constructors $\mathcal{C}$. Let $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ be a term. For $t = f(t_1,\dots,t_n)$ and $f \in \mathcal{D}$, we set $t^\sharp = f^\sharp(t_1,\dots,t_n)$ where $f^\sharp$ is a new n-ary function symbol called *dependency pair symbol*. For t not of this shape, we set $t^\sharp = t$. The least extension of the signature $\mathcal{F}$ containing all such dependency pair symbols is denoted by $\mathcal{F}^\sharp$. For a set $T \subseteq \mathcal{T}(\mathcal{F},\mathcal{V})$, we denote by $T^\sharp$ the set of marked terms $T^\sharp = \{t^\sharp \mid t \in T\}$. Let $\mathcal{C}_{\mathsf{com}} = \{\mathsf{c}_0,\mathsf{c}_1,\dots\}$ be a countable infinite set of fresh *compound symbols*, where we suppose $\mathsf{ar}(\mathsf{c}_n) = n$. Compound symbols are used to group calls in *dependency pairs for complexity* (*dependency pairs* or *DPs* for short). We define $\textsc{com}(t_1,\dots,t_n) := \mathsf{c}_n(t_1,\dots,t_n)$ where $\mathsf{c}_n \in \mathcal{C}_{\mathsf{com}}$ for $n \neq 1$, for $n = 1$ we set $\textsc{com}(t) := t$.

▶ **Definition 5.1** (Dependency Pair, Dependency Pair Complexity Problem).

1. A *dependency pair* (*DP* for short) is a rewrite rule $l^\sharp \to \textsc{com}(r_1^\sharp,\dots,r_n^\sharp)$ where $l, r_1,\dots,r_n \in \mathcal{T}(\mathcal{F},\mathcal{V})$ and l is not a variable.
2. Let $\mathcal{S}$ and $\mathcal{W}$ be two TRSs, and let $\mathcal{S}^\sharp$ and $\mathcal{W}^\sharp$ be two sets of dependency pairs. A complexity problem $\langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ with $\mathcal{T}^\sharp \subseteq \mathcal{T}_{\mathsf{b}}^\sharp$ is called a *dependency pair complexity problem* (or simply *DP problem*).

We keep the convention that $\mathcal{R}, \mathcal{S}, \mathcal{W}, \dots$ are TRSs over $\mathcal{T}(\mathcal{F},\mathcal{V})$, and the marked version $\mathcal{R}^\sharp, \mathcal{S}^\sharp, \mathcal{W}^\sharp, \dots$ always denote sets of dependency pairs.

▶ **Example 5.2** (Example 3.2 continued). Denote by $\mathcal{S}_\times^\sharp$ the dependency pairs

$$5\colon\ \mathsf{s}(x) \times^\sharp y \to \mathsf{c}_2((x \times y) +^\sharp y, x \times^\sharp y) \qquad 6\colon\ \mathsf{s}(x) +^\sharp y \to x +^\sharp y\ ,$$

and $\mathcal{T}_{\mathsf{b}}^\sharp$ the set of (marked) basic terms with defined symbols $+^\sharp, \times^\sharp$ and constructors $\mathsf{s}, 0$. Then $\mathcal{P}_\times^\sharp := \langle \mathcal{S}_\times^\sharp / \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_{\mathsf{b}}^\sharp\rangle$, where $\mathcal{R}_\times$ are the rules for addition and multiplication depicted in Example 3.2, is a DP problem. We anticipate that the DP problem $\mathcal{P}_\times^\sharp$ reflects the complexity of our multiplication problem $\mathcal{P}_\times$, compare Theorem 5.12 below.

For the remainder of this section, we fix a DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$. We call an n-holed context C a *compound context* if it contains only compound symbols. Consider the $\mathcal{P}_\times^\sharp$ derivation

$$\begin{aligned} D\colon\ \underline{\mathbf{2} \times^\sharp \mathbf{1}} &\to_{\mathcal{P}_\times^\sharp} \mathsf{c}_2((\underline{\mathbf{1} \times \mathbf{1}}) +^\sharp \mathbf{1}, \mathbf{1} \times^\sharp \mathbf{1}) \\ &\to_{\mathcal{P}_\times^\sharp}^* \mathsf{c}_2(\underline{\mathbf{1} +^\sharp \mathbf{1}}, \underline{\mathbf{1} \times^\sharp \mathbf{1}}) \\ &\to_{\mathcal{P}_\times^\sharp}^2 \mathsf{c}_2(\mathbf{0} +^\sharp \mathbf{1}, \mathsf{c}_2((\mathbf{0} \times \mathbf{1}) +^\sharp \mathbf{1}, \mathbf{0} \times^\sharp \mathbf{1}))\ . \end{aligned}$$

Observe that any term in the above sequence can be written as $C[t_1,\dots,t_n]$ where C is a maximal compound context, and $t_1,\dots,t_n$ are marked terms without compound symbols. For instance, the last term in this sequence is given as $C[\mathbf{0} \times^\sharp \mathbf{1}, (\mathbf{0} \times \mathbf{1}) +^\sharp \mathbf{1}, \mathbf{0} \times^\sharp \mathbf{1}]$ for

$C := \mathsf{c}_2(\Box, \mathsf{c}_2(\Box, \Box))$. This holds even in general, with the exception that $t_1, \dots, t_n$ are not necessarily marked. Note that such an unmarked term t_i ($i \in \{1, \dots, n\}$) can only result from the application of a collapsing rule $l^\sharp \to x$ for x a variable, which is permitted by our formulation of dependency pair. We capture this observation with the set $\mathcal{T}^\sharp_\rightharpoonup$, defined as the least extension of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}^\sharp(\mathcal{F}, \mathcal{V})$ that is closed under compound contexts. Then the following observation holds.

▶ **Lemma 5.3.** *For every TRS $\mathcal{R}$ and DPs $\mathcal{R}^\sharp$, we have $\to^*_{\mathcal{R}^\sharp \cup \mathcal{R}}(\mathcal{T}^\sharp_\rightharpoonup) \subseteq \mathcal{T}^\sharp_\rightharpoonup$. In particular, $\to^*_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp) \subseteq \mathcal{T}^\sharp_\rightharpoonup$ follows.*

Proof. Let $s = C[s_1, \dots, s_n] \in \mathcal{T}^\sharp_\rightharpoonup$ where C is a maximal compound context. Suppose $s \to_{\mathcal{R}^\sharp \cup \mathcal{R}} t$. Since C contains only compound symbols, it follows that $t = C[s_1, \dots, t_i, \dots, s_n]$ where $s_i \to_{\mathcal{R}^\sharp \cup \mathcal{R}} t_i$ for some $i \in \{1, \dots, n\}$, where again $t_i \in \mathcal{T}^\sharp_\rightharpoonup$. Consequently, $t \in \mathcal{T}^\sharp_\rightharpoonup$ and the first half of the lemma follows by inductive reasoning. From this the second half of the lemma follows, using that $\mathcal{T}^\sharp \subseteq \mathcal{T}^\sharp_\rightharpoonup$ and taking $\mathcal{R}^\sharp := \mathcal{S}^\sharp \cup \mathcal{W}^\sharp$ and $\mathcal{R} := \mathcal{S} \cup \mathcal{W}$. ◀

Consider a term $t = C[t_1, \dots, t_n] \in \mathcal{T}^\sharp_\rightharpoonup$ for a maximal compound context C. Any reduction of t consists of *independent sub-derivations* of t_i ($i = 1, \dots, n$), which are possibly interleaved. To avoid reasoning up to permutations of rewrite steps, we introduce a notion of *derivation tree* that disregards the order of parallel steps under compound contexts.

A (directed) *hypergraph* over *labels* $\mathcal{L}$ is a triple $G = (N, E, \mathsf{lab})$ where N is a set of *nodes*, $E \subseteq N \times \mathcal{P}(N)$ a set of *edges*, and $\mathsf{lab} \colon N \cup E \to \mathcal{L}$ a *labeling function*. For $e = \langle u, \{v_1, \dots, v_n\} \rangle \in E$ we call the node u the *source*, and nodes $v_1, \dots, v_n$ the *targets* of e. We keep the convenience that every node is the source of at most one edge. We denote by $\rightharpoonup_G$ the *successor relation* in G, i.e., $u \rightharpoonup_G v$ if there exists an edge $e = \langle u, \{v_1, \dots, v_n\} \rangle \in E$ with $v \in \{v_1, \dots, v_n\}$. We set $u \xrightharpoonup{\mathcal{K}}_G v$ for labels $\mathcal{K} \subseteq \mathcal{L}$ if additionally $\mathsf{lab}(e) \in \mathcal{K}$ holds, and abbreviate $\xrightharpoonup{\{l\}}_G$ by $\xrightharpoonup{l}_G$. If there exists a *path* $u = w_1 \rightharpoonup_G \cdots \rightharpoonup_G w_n = v$ we say that v is *reachable* from u in G. We call G a *hypertree* (*tree* for short) if there exists a unique node $u \in N$, the *root* of G, such that every $v \in N$ is reachable from u by a unique path.

▶ **Definition 5.4.** Let $t \in \mathcal{T}^\sharp(\mathcal{F}, \mathcal{V}) \cup \mathcal{T}(\mathcal{F}, \mathcal{V})$. The set of $\mathcal{P}^\sharp$ *derivation trees of* t, in notation $\mathsf{DTree}_{\mathcal{P}^\sharp}(t)$, is defined as the least set of labeled hypertrees such that:

1. $T \in \mathsf{DTree}_{\mathcal{P}^\sharp}(t)$ where T consists of a unique node labeled by t.
2. Suppose $t \xrightarrow{\mathcal{Q}}_{\{l \to r\}} \textsc{com}(t_1, \dots, t_n)$ for $l \to r \in \mathcal{P}^\sharp$ and let $T_i \in \mathsf{DTree}_{\mathcal{P}^\sharp}(t_i)$ for $i = 1, \dots, n$. Then $T \in \mathsf{DTree}_{\mathcal{P}^\sharp}(t)$, where T is a tree with children T_i ($i = 1, \dots, n$), the root of T is labeled by t, and the edge from the root of T to its children is labeled by $l \to r$.

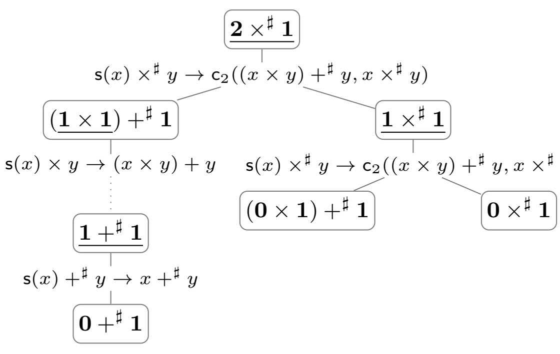

Figure 1 $\mathcal{P}^\sharp_\times$ derivation tree of $\mathbf{2} \times^\sharp \mathbf{1}$.

Figure 1 depicts a derivation tree T of $\mathcal{P}^\sharp_\times$ (cf. Example 5.2) that *corresponds* to the derivation D given below Example 5.2, in the sense that every edge $e = \langle u, \{v_1, \dots, v_n\} \rangle$ in T labeled by rule $l \to r$ corresponds to a rewrite step $t \xrightarrow{\mathcal{Q}}_{l \to r} \textsc{com}(t_1, \dots, t_n)$ in D, with t and $t_1, \dots, t_n$ precisely the label of source u and targets $v_1, \dots, v_n$ respectively. We also say that $l \to r$ was *applied* at node u in T. This correspondence leads to the following characterisation of the complexity function of DP problems $\mathcal{P}^\sharp$. Let $|T|_{\mathcal{R}^\sharp \cup \mathcal{R}}$ denote the number of applications of a rule $l \to r$ in the derivation tree T, i.e., the number of edges in T labeled by a rule $l \to r \in \mathcal{R}^\sharp \cup \mathcal{R}$.

▶ **Lemma 5.5.** *For every $t \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\sharp(\mathcal{F},\mathcal{V})$, we have*

$$\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}}) \simeq \max\{|T|_{\mathcal{S}^\sharp \cup \mathcal{S}} \mid T \text{ is a } \mathcal{P}^\sharp\text{-derivation tree of } t\} \ .$$

In particular $\mathsf{cp}_{\mathcal{P}^\sharp}(n) \simeq \max\{|T|_{\mathcal{S}^\sharp \cup \mathcal{S}} \mid T$ *is a* $\mathcal{P}^\sharp$*-derivation tree of* $t \in \mathcal{T}$ *with* $|t| \leqslant n\}$ *holds.*

5.1 Weak Dependency Pairs and Dependency Tuples

▶ **Definition 5.6** (Weak Dependency Pairs [12])**.** Let $\mathcal{R}$ denote a TRS such that the defined symbols of $\mathcal{R}$, i.e., roots of left-hand sides, are included in $\mathcal{D}$. Consider a rule $l \to C[r_1,\dots,r_n]$ in $\mathcal{R}$, where C is a maximal context containing only constructors. The dependency pair $l^\sharp \to \textsc{com}(r_1^\sharp,\dots,r_n^\sharp)$ is called a *weak dependency pair* of $\mathcal{R}$, in notation $\mathsf{WDP}(l \to r)$. We denote by $\mathsf{WDP}(\mathcal{R}) := \{\mathsf{WDP}(l \to r) \mid l \to r \in \mathcal{R}\}$ the set of all weak dependency pairs of $\mathcal{R}$.

In [12] it has been shown that for any term $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$, $\mathsf{dh}(t, \to_{\mathcal{R}}) = \mathsf{dh}(t^\sharp, \to_{\mathsf{WDP}(\mathcal{R}) \cup \mathcal{R}})$. We extend this result to our setting, where the following lemma serves as a preparatory step.

▶ **Lemma 5.7.** *Let $\mathcal{R}$ and $\mathcal{Q}$ be two TRSs, such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$. Then every derivation*

$$t = t_0 \xrightarrow{\mathcal{Q}}_{\mathcal{R}_1} t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R}_2} t_2 \xrightarrow{\mathcal{Q}}_{\mathcal{R}_3} \cdots \ ,$$

for basic term t and $\mathcal{R}_i \subseteq \mathcal{R}$ $(i \geqslant 1)$ is simulated step-wise by a derivation

$$t^\sharp = s_0 \xrightarrow{\mathcal{Q}}_{\mathsf{WDP}(\mathcal{R}_1) \cup \mathcal{R}_1} s_1 \xrightarrow{\mathcal{Q}}_{\mathsf{WDP}(\mathcal{R}_2) \cup \mathcal{R}_2} s_2 \xrightarrow{\mathcal{Q}}_{\mathsf{WDP}(\mathcal{R}_3) \cup \mathcal{R}_3} \cdots \ ,$$

and vice versa.

▶ **Theorem 5.8** (Weak Dependency Pair Processor)**.** *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ such that all defined symbols in $\mathcal{S} \cup \mathcal{W}$ occur in $\mathcal{D}$. The following processor is sound and complete.*

$$\frac{\vdash \langle \mathsf{WDP}(\mathcal{S}) \cup \mathcal{S}/\mathsf{WDP}(\mathcal{W}) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle \colon f}{\vdash \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle \colon f} \ \textsf{Weak Dependency Pairs}$$

Proof. Set $\mathcal{P} := \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ and $\mathcal{P}^\sharp := \langle \mathsf{WDP}(\mathcal{S}) \cup \mathcal{S}/\mathsf{WDP}(\mathcal{W}) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$. Suppose first $\mathsf{cp}_{\mathcal{P}^\sharp} \in \mathsf{O}(f(n))$. Lemma 5.7 shows that every $\to_{\mathcal{P}}$ reduction of $t \in \mathcal{T}$ is simulated by a corresponding $\to_{\mathcal{P}^\sharp}$ reduction starting from $t^\sharp \in \mathcal{T}^\sharp$. Observe that every $\xrightarrow{\mathcal{Q}}_{\mathcal{S}}$ step in the considered derivation is simulated by a $\xrightarrow{\mathcal{Q}}_{\mathsf{WDP}(\mathcal{S}) \cup \mathcal{S}}$ step. We thus obtain $\mathsf{cp}_{\mathcal{P}} \in \mathsf{O}(f(n))$. This proves soundness, completeness is obtained dual. ◀

Unlike for termination analysis, one has to account also for rewrite rules beside dependency pairs. In contrast, DP problems of the form $\langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ are usually easier to analyse, as rules that need to be accounted for, viz those appearing in the strict component, can only be applied in compound contexts. Some processors tailored for DP problems are even sound only in this setting [6]. Notably, in this setting the complexity pair processor requires that the strict order is monotone only on argument positions of compound symbols:

▶ **Lemma 5.9.** *Let μ denote a usable replacement map for dependency pairs $\mathcal{R}^\sharp$ in $\mathcal{P}^\sharp$. Then $\mu_{\textsc{com}}$ is a usable replacement map for $\mathcal{R}^\sharp$ in $\mathcal{P}^\sharp$, where $\mu_{\textsc{com}}$ denotes the restriction of μ to compound symbols in the following sense: $\mu_{\textsc{com}}(\mathsf{c}_n) := \mu(\mathsf{c}_n)$ for all $\mathsf{c}_n \in \mathcal{C}_{\mathsf{com}}$, and otherwise $\mu_{\textsc{com}}(f) := \varnothing$ for $f \in \mathcal{F}^\sharp$.*

Proof. For a proof by contradiction, suppose μ_{COM} is not a usable replacement map for $\mathcal{R}^\sharp$ in $\mathcal{P}$. Thus there exists $s \in \rightarrow^*_{\mathcal{P}}(\mathcal{T})$ and position $p \in \mathcal{P}\mathrm{os}(s)$ such that $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}^\sharp,p} t$ for some term t, but $p \notin \mathcal{P}\mathrm{os}_{\mu_{\text{COM}}}(s)$. Since $s \in \mathcal{T}^\sharp_{\rightarrow}$ by Lemma 5.3, symbols above position p in s are compound symbols, and so $p \notin \mathcal{P}\mathrm{os}_{\mu}(s)$ by definition of μ_{COM}. This contradicts however that μ is a usable replacement map for $\mathcal{R}^\sharp$ in $\mathcal{P}$. ◀

We remark that using Lemma 5.9 together with Theorem 4.6, our notion of $\mathcal{P}$-monotone complexity pair generalises *safe reduction pairs* from [12], that constitute of a rewrite preorder $\gtrsim$ compatible with a total order $\succ$ that is stable under substitutions. Here *safe* means that $\succ$ is monotone on compound contexts. It also generalises the notion of *μ-monotone complexity pair* from [14], that is parameterised by a single replacement map μ for all rules in $\mathcal{P}$.

In [12], the *weight gap principle* is introduced, with the objective to move the strict rules $\mathcal{S}$ into the weak component, in order to obtain a DP problem of the form $\langle \mathcal{S}^\sharp / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$, after the weak dependency pair transformation. *Dependency tuples* introduced in [21] avoid the problem altogether. A complexity problem is directly translated into this form, at the expense of completeness and a more complicated set of dependency pairs.

▶ **Definition 5.10** (Dependency Tuples [21])**.** Let $\mathcal{R}$ denote a TRS such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$. For a rewrite rule $l \rightarrow r \in \mathcal{R}$, let $r_1, \ldots, r_n$ denote all subterms of the right-hand side whose root symbol is in $\mathcal{D}$. The dependency pair $l^\sharp \rightarrow \text{COM}(r_1^\sharp, \ldots, r_n^\sharp)$ is called a *dependency tuple* of $\mathcal{R}$, in notation $\mathsf{DT}(l \rightarrow r)$. We denote by $\mathsf{DT}(\mathcal{R}) := \{\mathsf{DT}(l \rightarrow r) \mid l \rightarrow r \in \mathcal{R}\}$, the set of all dependency tuples of $\mathcal{R}$.

The central theorem of [21] states that dependency tuples are sound for runtime complexity analysis. We extend this result to a relative setting.

▶ **Lemma 5.11.** *Let $\mathcal{R}$ and $\mathcal{Q}$ be two TRSs, such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$, and such that $\mathrm{NF}(\mathcal{Q}) \subseteq \mathrm{NF}(\mathcal{R})$. Then every derivation*

$$t = t_0 \xrightarrow{\mathcal{Q}}_{\mathcal{R}_1} t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{R}_2} t_2 \xrightarrow{\mathcal{Q}}_{\mathcal{R}_3} \cdots ,$$

for basic term t and $\mathcal{R}_i \subseteq \mathcal{R}$ $(i \geqslant 1)$ is simulated step-wise by a derivation

$$t^\sharp = s_0 \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(\mathcal{R}_1) \cup \mathcal{R}_1} s_1 \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(\mathcal{R}_2) \cup \mathcal{R}_2} s_2 \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(\mathcal{R}_3) \cup \mathcal{R}_3} \cdots .$$

▶ **Theorem 5.12** (Dependency Tuple Processor)**.** *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ be an* innermost *complexity problem such that all defined symbols in $\mathcal{S} \cup \mathcal{W}$ occur in $\mathcal{D}$. The following processor is sound.*

$$\frac{\vdash \langle \mathsf{DT}(\mathcal{S}) / \mathsf{DT}(\mathcal{W}) \cup \mathcal{S} \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon f}{\vdash \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle \colon f} \ \text{Dependency Tuples}$$

Proof. The theorem follows by reasoning identical to Theorem 5.8, using Lemma 5.11. ◀

The problem $\mathcal{P}^\sharp_\times$ depicted in Example 5.2 is obtained from the runtime complexity problem $\mathcal{P}_\times$ of Example 3.2 using the above processor. For the sake of presentation we omitted the trivial dependency pairs $\mathsf{7}\colon 0 +^\sharp y \rightarrow \mathsf{c}_0$ and $\mathsf{8}\colon 0 \times^\sharp y \rightarrow \mathsf{c}_0$. That this omission is inessential has already been observed in [21], see also the technical report [6] on how this simplification can be formalised in our setting.

6 Dependency Graph Decomposition

In this section we focus on a novel technique that we call *dependency graph decomposition* (*DG decomposition* for short). Our work on this processor is motivated by the fact that we were not aware of a single method that translates a complexity problem into computationally simpler sub-problems, in the sense that any proof is of the form $\mathcal{P}_1\colon f_1, \dots, \mathcal{P}_n\colon f_n \vdash \mathcal{P}\colon f$ with $f \in \mathsf{O}(f_i)$ for some $i \in \{1, \dots, n\}$. This implies that the maximal bound one can prove is essentially determined by the strength of the employed base techniques, viz complexity pairs. In our experience however, a complexity prover is seldom able to synthesise a suitable and precise complexity pair that induces a complexity bound beyond a cubic polynomial.

We adapt the notion of dependency graph [1] to complexity problems.

▶ **Definition 6.1** (Dependency Graph)**.** Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ denote a DP problem. The nodes of the *dependency graph* (*DG* for short) $\mathcal{G}$ of $\mathcal{P}^\sharp$ are the dependency pairs from $\mathcal{S}^\sharp \cup \mathcal{W}^\sharp$, and there is an arrow labeled by $i \in \mathbb{N}$ from $s^\sharp \to \textsc{com}(t_1^\sharp, \dots, t_n^\sharp)$ to $u^\sharp \to \textsc{com}(v_1^\sharp, \dots, v_m^\sharp)$ if for some substitutions $\sigma, \tau\colon \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$, $t_i^\sharp\sigma \xrightarrow{\mathcal{Q}}{}^{*}_{\mathcal{S}\cup\mathcal{W}} u^\sharp\tau$.

Figure 2 depicts the dependency graph of our running example $\mathcal{P}_\times^\sharp$, where nodes (5) and (6) refer to the DPs given in Example 5.2. The dependency graph $\mathcal{G}$ indicates in which order dependency pairs can occur in a derivation tree of $\mathcal{P}^\sharp$. To make this intuition precise, we adapt the notion of *DP chain* known from termination analysis to derivation trees. Recall that for a derivation tree T, $\rightharpoonup_T$ denotes the successor relation, and $\xrightarrow{\mathcal{R}}_T$ its restriction to edges labeled by $l \to r \in \mathcal{R}$.

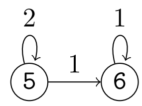

Figure 2 DG of $\mathcal{P}_\times^\sharp$.

▶ **Definition 6.2** (Dependency Pair Chain)**.** Let T be a derivation tree, and consider a path

$$u_1 \xrightharpoonup{\{l_1 \to r_1\}}_T \cdot \xrightharpoonup{\mathcal{S}\cup\mathcal{W}}{}^{*}_T u_2 \xrightharpoonup{\{l_2 \to r_2\}}_T \cdot \xrightharpoonup{\mathcal{S}\cup\mathcal{W}}{}^{*}_T \cdots ,$$

for a sequence of dependency pairs $C\colon l_1 \to r_1, l_2 \to r_2, \dots$. The sequence C is called a *dependency pair chain (in T)*, or *DP chain* for brevity.

▶ **Lemma 6.3.** *Every chain in a $\mathcal{P}^\sharp$ derivation tree is a path in the dependency graph of $\mathcal{P}^\sharp$.*

Dependency graph decomposition seeks to analyse *recursive definitions*, as reflected by *cycles* in the DG, separately. This method is thus closely connected to *cycle analysis* as introduced for termination in [11], that allows the decomposition of the input into separate cycles with respect to the DG.

▶ **Example 6.4** (Example 5.2 continued)**.** Reconsider the problem $\mathcal{P}_\times^\sharp = \langle \{5,6\}/\mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle$ given in Example 5.2. A decomposition into cycles amounts to an inference

$$\frac{\vdash \langle \{5\}/\mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle\colon f \qquad \vdash \langle \{6\}/\mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle\colon g}{\vdash \langle \{5,6\}/\mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle\colon c_{f,g}} ,$$

for cycles (5) and (6), compare Figure 2. This inference is sound for termination analysis [11]. Notice that for f and g we can substitute linear functions, whereas the overall complexity of $\mathcal{P}_\times^\sharp$ is cubic. To see that this bound holds, consider a maximal reduction of $t^\sharp \in \mathcal{T}^\sharp$ for the more involved case $t^\sharp = \mathbf{m} \times^\sharp \mathbf{n}$. Then the i^{th} application of

$$5\colon\ \mathsf{s}(x) \times^\sharp y \to \mathsf{c}_2((x \times y) +^\sharp y, x \times^\sharp y) ,$$

in this derivation triggers an independent sub-derivation starting from $t_i^\sharp = \mathbf{m_i} +^\sharp \mathbf{n}$, where $m_i := (m - i) * n$. It is not difficult to verify that the number of applications of (6) in a sub-derivation of $t_i^\sharp$ is bounded by m_i, thus bounded by a *quadratic* polynomial in the size of $t^\sharp$. As there are at most as many such sub-derivations as there are applications of the DP (5), viz linearly many in the size of $t^\sharp$, we obtain an overall *cubic* bound.

Dependency graph decomposition can infer this bound automatically, using similar reasoning. Taking the call-structure between cycles into account is crucial for such an analysis:

▶ **Example 6.5.** Let $\mathcal{P}^\sharp_{\mathsf{exp}} := \langle \mathcal{R}^\sharp_{\mathsf{exp}}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}} \rangle$ where dependency pairs $\mathcal{R}^\sharp_{\mathsf{exp}}$ are

$$\mathtt{9}\colon \mathsf{d}^\sharp(\mathsf{s}(x)) \to \mathsf{d}^\sharp(x) \qquad \mathtt{10}\colon \mathsf{e}^\sharp(\mathsf{s}(x)) \to \mathsf{c}_2(\mathsf{d}^\sharp(\mathsf{e}(x)), \mathsf{e}^\sharp(x)) \ ,$$

and the rewrite system $\mathcal{R}_{\mathsf{exp}}$ is given by the four rules

$$\mathtt{11}\colon \mathsf{d}(0) \to 0 \qquad \mathtt{12}\colon \mathsf{d}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{d}(x))) \qquad \mathtt{13}\colon \mathsf{e}(0) \to 0 \qquad \mathtt{14}\colon \mathsf{e}(\mathsf{s}(x)) \to \mathsf{d}(\mathsf{e}(x)) \ ,$$

that compute exponentiation on numerals. The DG of $\mathcal{P}^\sharp_{\mathsf{exp}}$ consists of two cycles, (9) and (10) respectively. While the complexity of $\langle \{\mathtt{9}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}} \rangle$ and $\langle \{\mathtt{10}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}^\sharp_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}} \rangle$ is again linear, the complexity of $\mathcal{P}^\sharp_{\mathsf{exp}}$ is exponential.

In contrast to a full decomposition into all cycles, DG decomposition produces a pair of sub-problems, obtained by separating the dependency graph between maximal cycles. Iterated application then extends to a separate analysis of all cycles. Call a set of DPs $\mathcal{R}^\sharp$ *forward closed* in $\mathcal{P}^\sharp$, if it is closed under successors with respect to the DG of $\mathcal{P}^\sharp$, i.e., if there is an edge from $s \to t \in \mathcal{R}^\sharp$ to $u \to v$ then also $u \to v \in \mathcal{R}^\sharp$. Throughout the following, we fix a complexity problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp_u \cup \mathcal{S}^\sharp_l \cup \mathcal{S}/\mathcal{W}^\sharp_u \cup \mathcal{W}^\sharp_l \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ whose strict and weak dependency pairs are partitioned such that $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ is *forward closed* in $\mathcal{P}^\sharp$.

As $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ is forward closed in $\mathcal{P}^\sharp$, DPs from $\mathcal{S}^\sharp_u \cup \mathcal{W}^\sharp_u$ can trigger applications of DPs from $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ but not vice versa, compare Lemma 6.3. To formalise this observation, consider a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$. Then the forward closed set $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ induces a separation of T into two (possibly empty) layers, demarcated by topmost applications of DPs from $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$: the *lower layer* constitutes of the (maximal) subtrees $T_1, \dots, T_m$ of T with a dependency pair $l \to r \in \mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ applied at the root, by forward closure these are $\langle \mathcal{S}^\sharp_u \cup \mathcal{S}/\mathcal{W}^\sharp_u \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ derivation trees of some terms $t_i^\sharp$ $(i = 1, \dots, m)$ in T; the *upper layer* consists of the derivation tree $T_\uparrow$ obtained from T by removing the sub-trees $T_1, \dots, T_m$. Compare Figure 3 that illustrates this separation. The DG decomposition processor uses the DPs $\mathsf{sep}(\mathcal{S}^\sharp_u \cup \mathcal{W}^\sharp_u)$, defined as follows, to extend the derivation trees T_i of $t_i^\sharp$ to derivation trees of $t^\sharp \in \mathcal{T}^\sharp$.

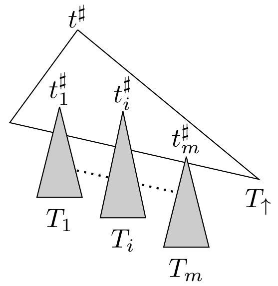

Figure 3 Separation of derivation tree T in upper and lower layer.

▶ **Definition 6.6.** For a set of DPs $\mathcal{R}^\sharp$ we define

$$\mathsf{sep}(\mathcal{R}^\sharp) := \{ l \to r_i \mid l \to \textsc{com}(r_1, \dots, r_i, \dots, r_k) \in \mathcal{R}^\sharp \} \ .$$

▶ **Example 6.7** (Example 6.4 continued)**.** Consider the complexity problem $\mathcal{P}^\sharp_\times$ from Example 5.2, where $\{\mathtt{6}\colon \mathsf{s}(x) +^\sharp y \to x +^\sharp y\}$ constitutes a forward closed set of DPs with respect to the DG drawn in Figure 2.

Let T denote a $\mathcal{P}^\sharp_\times$ derivation tree of $t^\sharp := \mathbf{m} \times^\sharp \mathbf{n}$ $(m, n \in \mathbb{N})$. For $m_i := (m - i) \cdot n$ $(i = 1, \dots, m)$, the nodes labeled by $t_i^\sharp := \mathbf{m_i} +^\sharp \mathbf{n}$ demarcate upper and lower layer in T, compare the derivation tree depicted in Figure 1. Consider the DPs $\mathsf{sep}(\{\mathtt{5}\})$ given by

$$\mathtt{5a}\colon \mathsf{s}(x) \times^\sharp y \to (x \times y) +^\sharp y \qquad \mathtt{5b}\colon \mathsf{s}(x) \times^\sharp y \to x \times^\sharp y \ .$$

Let T_i $(i = 1, \dots, m)$ denote the sub-trees rooted at the nodes labeled by $t_i^\sharp$ that constitute the lower layer in T. In combination with rewrite rules $\mathcal{R}_\times$, the DPs (5a) and (5b) generate exactly the terms $t_i^\sharp$ from $t^\sharp$. As a consequence, the complexity problem $\langle\{6\}/\{5a, 5b\} \cup \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_b^\sharp\rangle$ accounts for applications of $\{6\}$ in the sub-trees T_i. In other words, it accounts for applications of DPs in the sub-derivations of $t_i^\sharp$ as investigated in Example 6.4. In correspondence to Example 6.4, it is not difficult to verify that $\vdash \langle\{6\}/\{5a, 5b\} \cup \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_b^\sharp\rangle\colon n^2$ is valid.

It is also not difficult to verify that $\vdash \langle\{5\}/\mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_b^\sharp\rangle\colon n$ holds, and this linear bound can be used to bind the applications of the remaining DP (5) in the upper layer $T_\uparrow$ of T, thus in T. As this also bind the number of sub-trees $T_1, \dots, T_m$ that constitute lower layer, we overall get a cubic bound on applications on DPs in T in the size of $t^\sharp$, i.e., $|T|_{\{5,6\}} \in \mathsf{O}(|t^\sharp|^3)$.

The previous complexity proof is an instance of DG decomposition as introduced below. The next two lemmas, used in the soundness proof of the DG decomposition processor, formalise the crucial proof steps employed in Example 6.7. The first observation is simple.

▶ **Lemma 6.8.** *Let $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ be a forward closed set of DPs in $\mathcal{P}^\sharp$, and let T be a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$. Consider the maximal sub-trees $T_1, \dots, T_m$ of T such that $l \to r \in \mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ is applied at the root, and let $T_\uparrow$ be obtained from T by removing $T_1, \dots, T_m$. Then*
1. *$T_\uparrow$ is a $\langle\mathcal{S}_u^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ derivation tree of $t^\sharp$;*
2. *for all $i = 1, \dots, m$, there exists a $\langle\mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp\rangle$ derivation trees of $t^\sharp$, that contains T_i as sub-tree.*

Denote by $\mathsf{Pre}_\mathcal{G}(l \to r)$ *direct predecessors* of the dependency pair $l \to r$ in the DG $\mathcal{G}$ of $\mathcal{P}^\sharp$, extended to sets of DPs by $\mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp) := \cup_{l \to r \in \mathcal{R}^\sharp} \mathsf{Pre}_\mathcal{G}(l \to r)$.

▶ **Lemma 6.9.** *Let $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ be a forward closed set of DPs in $\mathcal{P}^\sharp$, and let T be a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$. Let $T_1, \dots, T_m$ denote the maximal sub-trees of T with $l \to r \in \mathcal{R}^\sharp$ applied at the root. There exists a constant $\Delta \in \mathbb{N}$ depending only on $\mathcal{P}^\sharp$ such that $m \leqslant \max\{1, |T|_{\mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp) \setminus \mathcal{R}^\sharp} \cdot \Delta\}$.*

Proof. Let Δ be the maximal arity of a compound symbol from $\mathcal{P}^\sharp$, and observe that every node in T has at most Δ successors. Denote by $\{u_1, \dots, u_m\}$ the roots of T_i $(i = 1, \dots, m)$. The non-trivial case is $m > 1$. In this case, each path from the root of T to the nodes $u_i \in \{u_1, \dots, u_m\}$ contains at least one node with a DP applied. Let $\{v_1, \dots, v_n\}$ collect such nodes closest to $\{u_1, \dots, u_m\}$. In particular, we can thus associate to every node $u_i \in \{u_1, \dots, u_m\}$ a node $v_{i'} \in \{v_1, \dots, v_n\}$ and DP $l \to r \in \mathcal{P}^\sharp$ such that $v_{i'} \xrightarrow{\{l_i \to r_i\}}_T \cdot \xrightarrow{\mathcal{S} \cup \mathcal{W}}{}^*_T u_i$ holds. As $v_{i'}$ has at most Δ successors and $\xrightarrow{\mathcal{S} \cup \mathcal{W}}_T$ is non-branching, it follows that $m \leqslant \Delta \cdot n$. By Lemma 6.3, for $i = 1, \dots, m$ we see $l_i \to r_i \in \mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp)$. As T_i is maximal, $l_i \to r_i \notin \mathcal{R}^\sharp$. Hence $n \leqslant |T|_{\mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp) \setminus \mathcal{R}^\sharp}$ and the lemma follows. ◀

▶ **Theorem 6.10** (Dependency Graph Decomposition). *Consider a dependency pair problem $\mathcal{P}^\sharp = \langle\mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ such that (i) $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ is forward closed and (ii) $\mathsf{Pre}_\mathcal{G}(\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp) \cap \mathcal{W}_u^\sharp = \varnothing$ for the DG $\mathcal{G}$ of $\mathcal{P}^\sharp$. The following processor is sound.*

$$\frac{\vdash \langle\mathcal{S}_u^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle\colon f \qquad \vdash \langle\mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_l^\sharp \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle\colon g}{\vdash \langle\mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle\colon f * g} \text{DG decomp.},$$

for all bounding functions f and g such that $f(n) \neq 0$ and $g(n) \neq 0$ for all $n \in \mathbb{N}$.

Proof. Consider a $\mathcal{P}^\sharp$ derivation tree of $t^\sharp \in \mathcal{T}^\sharp$. We tacitly employ the characterisation of complexity function given in Lemma 5.5, and estimate $|T|_{\mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}}$ by a function in $\mathsf{O}(f * g)$.

Consider the separation of T as induced by forward closure of $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ into the upper layer $T_\uparrow$, and lower layer consisting of the derivation trees T_i of $t_i^\sharp$ $(i = 1, \ldots, m)$, as in Figure 3. By Lemma 6.8(2) the trees T_i $(i = 1, \ldots, m)$ can be extended to $\langle \mathcal{S}_l^\sharp \cup \mathcal{S} / \mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp \rangle$ derivation tree T_i' of $t^\sharp$. In particular, the complexity of $\langle \mathcal{S}_l^\sharp \cup \mathcal{S} / \mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp \rangle$ binds applications of $\mathcal{S}_l^\sharp \cup \mathcal{S}$ in T_i, i.e., $|T_i|_{\mathcal{S}_l^\sharp \cup \mathcal{S}} = |T_i'|_{\mathcal{S}_l^\sharp \cup \mathcal{S}}$. Hence $|T_i|_{\mathcal{S}_l^\sharp \cup \mathcal{S}} \in \mathsf{O}(g(|t^\sharp|))$ by the second precondition of the processor. Similar, Lemma 6.8(1) and the first precondition of the processor gives $|T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}} \in \mathsf{O}(f(|t^\sharp|))$. By assumption (ii) and Lemma 6.9 we see $m \leqslant \max\{1, |T|_{\mathsf{Pre}_\mathcal{G}(\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp) \setminus (\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp)}\} \leqslant \max\{1, |T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}}\}$. Putting these bounds together we get

$$
\begin{aligned}
|T|_{\mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}} &= |T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}} + \textstyle\sum_{i=1}^m |T_i|_{\mathcal{S}_l^\sharp \cup \mathcal{S}} \\
&\leqslant |T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}} + \max\{1, |T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}}\} \cdot \max_{i=1}^m |T_i|_{\mathcal{S}_l^\sharp \cup \mathcal{S}} \\
&\in \mathsf{O}(f(|t^\sharp|)) + \mathsf{O}(f(|t^\sharp|)) * \mathsf{O}(g(|t^\sharp|)) = \mathsf{O}(f(|t^\sharp|) * f(|t^\sharp|)) \ .
\end{aligned}
$$

◀

▶ **Example 6.11** (Example 6.7 continued). Reconsider the DP problem $\mathcal{P}_\times^\sharp = \langle \mathcal{S}_\times^\sharp / \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle$. According to Theorem 6.10, the following depicts a sound inference:

$$
\frac{\vdash \langle \{\mathsf{5}\} / \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle \colon f \quad \vdash \langle \{\mathsf{6}\} / \{\mathsf{5a}, \mathsf{5b}\} \cup \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle \colon g}{\vdash \langle \mathcal{S}_\times^\sharp / \mathcal{R}_\times, \mathcal{R}_\times, \mathcal{T}_\mathsf{b}^\sharp \rangle \colon f * g} \ .
$$

It is not difficult to find polynomial interpretations that verify that the sub-problems have linear and quadratic complexity respectively. Overall we thus obtain the (tight) bound $\mathsf{O}(n^3)$, which in turn binds the complexity of $\mathcal{P}_\times$ by Theorem 5.12.

7 Conclusion

We have presented a combination framework for automated polynomial complexity analysis of term rewrite systems. The framework is general enough to reason about both runtime and derivational complexity, and to formulate a majority of the techniques available for proving polynomial complexity of rewrite systems. On the other hand, it is concrete enough to serve as a basis for a modular complexity analyser, as demonstrated by our automated complexity analyser TCT which closely implements the discussed framework.

Besides the combination framework we have introduced the notion of $\mathcal{P}$-monotone complexity pair that unifies the different orders used for complexity analysis in the cited literature. Last but not least, we have presented the dependency graph decomposition processor. This processor is easy to implement, and greatly improves modularity. This is underpinned by the experimental evidence given online[4] that highlights the strength of our framework, and in particular of the dependency graph decomposition processor.

References

1 T. Arts and J. Giesl. Termination of Term Rewriting using Dependency Pairs. *TCS*, 236(1–2):133–178, 2000.

2 M. Avanzini. POP* and Semantic Labeling using SAT. In *Proc. of ESSLLI 2008/2009 Student Session*, volume 6211 of *LNCS*, pages 155–166. Springer, 2010.

[4] Available online at `http://cl-informatik.uibk.ac.at/software/tct/experiments/tct2/`.

3 M. Avanzini, N. Eguchi, and G. Moser. New Order-theoretic Characterisation of the Polytime Computable Functions. 2012. Submitted to TCS.

4 M. Avanzini and G. Moser. Closing the Gap Between Runtime Complexity and Polytime Computability. In *Proc. of 21^st RTA*, volume 6 of *LIPIcs*, pages 33–48, 2010.

5 M. Avanzini and G. Moser. Polynomial Path Orders: A Maximal Model. 2012. Submitted to LMCS. Technical Report available at `http://arxiv.org/abs/1209.3793`.

6 M. Avanzini and G. Moser. A Combination Framework for Complexity, Technical Report. *CoRR, cs/CC/1302.0973*, 2013. Available at `http://www.arxiv.org/abs/1302.0973`.

7 M. Avanzini and G. Moser. Tyrolean Complexity Tool: Features and Usage. In *Proc. of 24^th RTA*. LIPIcs, 2013.

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

9 G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with Polynomial Interpretation Termination Proof. *JFP*, 11(1):33–53, 2001.

10 U. Dal Lago and S. Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In *Proc. of 36^th ICALP*, volume 5556 of *LNCS*, pages 163–174. Springer, 2009.

11 J. Giesl, T. Arts, and E. Ohlebusch. Modular Termination Proofs for Rewriting Using Dependency Pairs. *JSC*, 34:21–58, 2002.

12 N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In *Proc. of 4^th IJCAR*, volume 5195 of *LNAI*, pages 364–380, 2008.

13 N. Hirokawa and G. Moser. Complexity, Graphs, and the Dependency Pair Method. In *Proc. of 15^th LPAR*, pages 652–666, 2008.

14 N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. 2012. Submitted to IC, available at `http://arxiv.org/abs/1102.3129`.

15 D. Hofbauer and C. Lautemann. Termination Proofs and the Length of Derivations. In *Proc. of 3^rd RTA*, volume 355 of *LNCS*, pages 167–177. Springer, 1989.

16 D. Hofbauer and J. Waldmann. Termination of String Rewriting with Matrix Interpretations. In *Proc. of 17^th RTA*, volume 4098 of *LNCS*, pages 328–342. Springer, 2011.

17 S. Lucas. Fundamentals of Context-Sensitive Rewriting. In *Proc. of 22^th SOFSEM*, LNCS, pages 405 – 412. Springer, 1995.

18 A. Middeldorp, G. Moser, F. Neurauter, J. Waldmann, and H. Zankl. Joint Spectral Radius Theory for Automated Complexity Analysis of Rewrite Systems. In *Proc. of 4^th CAI*, volume 6742 of *LNCS*, pages 1–20. Springer, 2011.

19 G. Moser. Proof Theory at Work: Complexity Analysis of Term Rewrite Systems. *CoRR*, abs/0907.5527, 2009. Habilitation Thesis.

20 G. Moser, A. Schnabl, and J. Waldmann. Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations. In *Proc. of the 28th FSTTCS*, LIPIcs, pages 304–315, 2008.

21 L. Noschinski, F. Emmes, and J. Giesl. A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems. In *Proc. of 23^rd CADE*, LNAI, pages 422–438. Springer, 2011.

22 A. Schnabl. *Derivational Complexity Analysis Revisited.* PhD thesis, University of Innsbruck, 2012. Available at `http://cl-informatik.uibk.ac.at/research/`.

23 R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting.* PhD thesis, University of Aachen, 2007. Available as Technical Report AIB-2007-17.

24 H. Zankl and M. Korp. Modular Complexity Analysis via Relative Complexity. In *Proc. of 21^st RTA*, volume 6 of *LIPIcs*, pages 385–400, 2010.

25 H. Zantema. Termination of Context-Sensitive Rewriting. In *Proc. of 8^th RTA*, volume 1232 of *LNCS*, pages 172–186. Springer, 1997.