Derivational Complexity Analysis Revisited

dissertation

by

Andreas Schnabl

submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements for the degree of Doctor rerum naturalium

advisor: Assoz.-Prof. Dr. Georg Moser

Innsbruck, 12 December 2011

dissertation

Derivational Complexity Analysis Revisited

Andreas Schnabl (0215074)
andreas.schnabl@uibk.ac.at

12 December 2011

advisor: Assoz.-Prof. Dr. Georg Moser

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum	Unterschrift

Abstract

This thesis is concerned with *derivational complexity analysis* of *term rewrite systems*. Term rewriting is a simple, but Turing-complete model of computation which resembles first-order functional programming. Derivational complexity is a natural measure of time complexity for term rewrite systems: it relates the maximum number of computational steps in term rewriting (*rewrite steps*) with the size of the initial term. In the literature, there exist many results certifying upper bounds on the derivational complexity of a term rewrite system whose termination can be proved by certain techniques. In this thesis, we expand upon these results by the following contributions.

First, we determine the hardness of deciding whether a certain class of functions is an upper bound on the derivational complexity of a given term rewrite system. For that, we employ the *arithmetical hierarchy*, a device for classifying the degree of undecidability of decision problems. Although term rewriting has some significant differences from other models of computation, it turns out that the hardness of this decision problem is the same as for the corresponding decision problem for Turing Machines.

Second, we consider variants of existing termination proof techniques which have been defined specifically for derivational complexity analysis: *context dependent interpretations* and restrictions of *matrix interpretations*. We state complexity results corresponding to these techniques. Moreover, despite their seeming difference, we present an equivalence between important subsets of them.

Third, we consider the currently most important technique for proving termination of term rewrite systems: the *dependency pair framework*. It is a framework for combining many small partial proof arguments (which are formalised as *DP processors*) into a complete termination proof of the considered term rewrite system. Moreover, the possibility to define new DP processors makes the dependency pair framework easily extensible. While, as stated before, complexity results exist for many (direct) termination proof techniques, no results existed for any DP processors in the dependency pair framework before. We consider the most important DP processors and give derivational complexity bounds term rewrite systems whose termination can be proved by various combinations of these DP processors. Moreover, we provide examples which show that all these complexity bounds are essentially optimal.

Finally, we give a brief overview over the software tool $\mathsf{T_CT}$, which is an automatic prover for upper complexity bounds for term rewrite systems. The focus of $\mathsf{T_CT}$ is polynomial bounds. In particular, those techniques mentioned in this thesis, from which a polynomial upper bound on the derivational complexity of the considered term rewrite system can be concluded, have been implemented in $\mathsf{T_CT}$.

Acknowledgments

I would like to express my gratitude to my supervisor Georg Moser for his advice and guidance throughout my studies. He taught me scientific writing and introduced me to the research field of derivational complexity analysis. His stimulus ensured that I was really getting forward in the right direction.

I want to give my special thanks to my long-time office mate and $\mathsf{T_CT}$-codeveloper Martin Avanzini. We had numerous enlightening discussions about programming $\mathsf{T_CT}$, each other's theoretical work, and science in general.

Another special mentioning goes to Martin Korp, Christian Sternagel, and Harald Zankl, for providing the source code of the termination prover $\mathsf{T_TT_2}$, upon which the first incarnation of $\mathsf{T_CT}$was based. I gratefully mention Christian and Harald again for the development of the CL LaTeX template which I used for writing this thesis.

Further, I thank my colleagues from the Computational Logic research group, namely Martin Avanzini, Simon Bailey, Clemens Ballarin, Bertram Felgenhauer, Martina Ingenhaeff, Martin Korp, Aart Middeldorp, Georg Moser, Friedrich Neurauter, Christian Sternagel, René Thiemann, Sarah Winkler, and Harald Zankl, for providing a very pleasant group environment and work atmosphere over the past four years.

I would also like to thank my parents Irmgard and Ferdinand for their continued support over the years. Without them, it would have been much harder to focus on my studies.

Finally, I gratefully acknowledge the financial support by FWF (Austrian Science Fund) via the projects P20133-N15 and I608-N18, and by the University of Innsbruck via grant 2010/2/MIP15 of the office of the vice rector for research.

Contents

Preface

Own Contributions

Since this thesis is embedded into an existing and active research area, it goes without saying that it is impossible to strive for its self-containment without including anything other than my own work. Giving all required foundations of the treated topic and describing related work is necessary to make this thesis more complete. Moreover, in order to improve quality of the results, and make them available to the international community in a timelier manner, most of the actual scientific contributions of this thesis have already been published in conference proceedings and journal papers. These previous publications are [84, 82, 83, 98]. The paper [82] is a journal paper partly based on a previous conference paper [80], and [83] is a conference paper built upon a previous workshop contribution [81]. On the other hand, [84, 98] are principally independent conference papers. In the following, I provide a general breakdown of what parts of this thesis have been known previously, what parts are new contributions, and which of these new contributions have already been published in one of the just mentioned papers. More specific pointers to used sources are given at the respective places in the text of the thesis.

Chapter 2: Since this chapter recalls the fundamentals of the topics covered throughout the rest of the thesis, it draws almost exclusively from pre-existing knowledge (which can mainly be found in the references given at the beginning of each section in this chapter). Exceptions to this rule, where my definitions slightly digress from standard notions in the literature I am aware of, are the notion of $\textsc{LifeTime}_M$ in Definition 2.11 ([98]), the notion of simple progeny in Definition 2.25 (similar to the notion of progenies used in [82]), the operator Ξ from Definition 2.36 ([98]), the notion of induced complexity given in Definitions 2.37 and 2.51 (used less formally in [82]), the measure of DP complexity as shown in Definition 2.47 (used in a similar way in [80]), and the notion whether a DP processor completely solves a DP problem, see Definitions 2.49, 2.58, and 2.70. Moreover, the small observations made in Theorems 2.52, 2.53, and 2.56 (which was given in [82] first), and Example 2.54, and the slight modifications to the usable rules refinement exhibited throughout Section 2.4.2 (to the extent they differ from the treatment of usable rules given by Hirokawa and Middeldorp in [43]) are new.

Chapter 3: The content of this chapter is a contribution of this thesis. It has previously been published jointly by myself and Simonsen in [98].

Chapter 4: This chapter recapitulates known results about termination proof techniques and associated complexity results, and therefore mostly draws from pre-existing knowledge found in the literature. Exceptions to this rule are the complexity considerations for top-bounds (Theorem 4.56), top-DP-bounds (Theorem 4.58), and match-DP-bounds (Example 4.59), which are completely new.

Chapter 5: The material presented in this chapter mostly consists of new contributions, which have been published jointly by Moser, myself, and Waldmann in [84]. Exceptions to this rule are the following items, which are already known in the literature: parts of the foundations of context dependent interpretations (generalised from Hofbauer's version [49]) which are introduced in the first part of Section 5.3 in order to be built upon (essentially Definition 5.9, Theorem 5.10, a slightly less general version of Lemma 5.16, and the notions of Δ-linear and Δ-restricted interpretations introduced by Definition 5.13) have been included in my Master's thesis [96] and a subsequent conference paper [79]. Moreover, Section 5.4 is entirely devoted to the description of related work by Middeldorp, Moser, Neurauter, Waldmann, and Zankl [86, 110, 76]. Finally, the experiments described in Section 5.5 were completely rerun with the newest versions of the respective tools and testbeds, hence this section is only partially based on the corresponding section in [84].

Chapter 6: This chapter wholly consists of new contributions, most of which have been published jointly by Moser and myself. The material presented in Sections 6.2 and 6.3 appeared in [83], while Sections 6.4, 6.5, 6.6, 6.8, and 6.9 were essentially published in [82]. A result similar to the main result of Section 6.7 was contained in [82], as well, but I have completely revised its proof, and hence improved that result. The experimental comparison described in Section 6.10 is completely new.

Chapter 7: The material presented in this chapter is completely new, and has not been published before. The software described in this chapter has been developed jointly by Avanzini, Moser, and myself.

I would like to note that the publications listed at the beginning of this section do not form an exhaustive listing of the papers (co-)authored by me during my PhD studies. The content of the remaining papers has not (or only partially, where explicitly mentioned) been used for this thesis. These papers are [79, 7, 97]. The first of these papers [79] has not been included in this thesis because it was based on the results produced by my Master's thesis. The software tool described in the second paper [7] is not being developed anymore. Moreover functionality-wise, it has been mostly subsumed by $\mathsf{T_CT}$, which is described in Chapter 7 and used for the experiments conducted for Section 5.5. The final of these papers [97] is a description of `cdiprover3`, which is used for a part of the experiments described in Section 5.5. However, since its functionality related to polynomial interpretations is subsumed by $\mathsf{T_CT}$, context dependent interpretations are already treated in Section 5.3, and `cdiprover3` is not being developed anymore, either, I decided not to include the material of that paper into this thesis.

Coauthorship statements

As mentioned above, most new scientific contributions made by this thesis have previously appeared in papers coauthored by me [84, 82, 83, 98]. In the following, I clarify my contributions to each of these papers in their published versions.

- The paper [84] was joint work with Georg Moser and Johannes Waldmann. Moser had introduced me to the topic of derivational complexity and suggested the topic of the paper. Waldmann had the idea for one of the paper's two main theoretical contributions (proving polynomial complexity bounds using triangular matrix interpretations). I had the idea for the other theoretical contribution (the relationship between context dependent and triangular matrix interpretations), filled out the formal proof details for the main theorems, implemented the techniques, and conducted the experiments. In the manuscript, I was responsible for the general description of one of the main topics (context dependent interpretations) and all the proofs. Waldmann filled in the general remarks about the other main topic (matrix interpretations), summarised the relationship between various subclasses of matrix and context dependent interpretations, and gave some general context for matrix interpretations. Finally, Moser fleshed out everything else (description of experiments and the paper's general context, and parts of the description of matrix and context dependent interpretations), and did the final editing, and a lot of reformulation and restructuring throughout most of the paper.

- The paper [82] was joint work with Georg Moser. I had the proof ideas for most of the theoretical contributions (the upper complexity bounds and examples illustrating their optimality for the basic dependency pair method, its restriction to string rewriting, dependency graphs, and usable rules), while some of them, and several corrections in others, were the result of joint discussion. I subsequently filled out the proofs in the paper. Moser suggested the topic of the conference paper [80] underlying this article. He added a crucial ingredient in one of the proofs (using fast growing functions to provide an upper bound on the derivational complexity of the simulating TRS for dependency graphs) and its technical details, collaborated with me in working out some of the other proofs, and contributed many simplifications, reformulations and clarifications to most other technical proof parts.

- The paper [83] was joint work with Georg Moser. I had the idea for the theoretical contribution of the paper, and constructed the proof of all technical theorems. Several technical corrections were the result of joint discussion. Moser contributed many suggestions for improving the structure and presentation of the proof, added the general descriptions putting the theoretical relevance of the result into context, and the final editing of the paper.

- The paper [98] was joint work with Jakob Grue Simonsen. The idea for writing this paper came from Simonsen. The main theoretical contributions of the paper were the result of joint discussion, with Simonsen mainly providing proof concepts on the general level and pointers to relevant literature in computability theory, and me transferring the general ideas to term rewriting and filling in the technical details on that level. In the manuscript, I mostly contributed the material related to the core complexity notions for rewriting (derivational complexity, implicit complexity, and runtime complexity), while Simonsen wrote down the results related to other notions of complexity (minimal complexity and complexity under strategies), which have specific relevance outside of term rewriting, and he provided most of the explanations giving the general context of the paper. I did the final editing of the paper.

Chapter 1

Introduction

What more do we know if we have proved a theorem by restricted means other than if we merely know the theorem is true?

Georg Kreisel

In this thesis, we consider *term rewriting*, a model of computation. On one hand, term rewriting is very simple and intuitive. First-order *terms* are used as the data structure in this model of computation, and a program is given by a set of *rules* (a *term rewrite system*), where each rule is essentially an oriented equation over terms. Then a step of computation (a *rewrite step*) in this model is a replacement of an occurrence of the left-hand of such a rule in a term by its right-hand side. Note that this way of computation is in general nondeterministic. A term which no longer allows the application of any rewrite rules (a *normal form*) is regarded as the result of such a computation. On the other hand, term rewrite systems have the same computational power as Turing Machines, the classical universal model of computation, so any computable function can be computed by a term rewrite system (if the *Church-Turing thesis*, which asserts the same for Turing Machines, is true). Hence, they are an adequate mathematical abstraction of programs in any contemporary programming language. In particular, term rewriting shares a lot of conceptual similarity with (first-order) functional and declarative programming.

An important property of programs in any model of computation is *correctness*, and one significant component of correctness is *termination*, i.e. the assertion that the program under consideration always produces a result in a finite amount of time, regardless of the input it is given. For Turing Machines, the problem whether a given Machine satisfies termination is called the *uniform halting problem*, which is well-known to be undecidable in general. The same is true for the problem of deciding termination of a given term rewrite system. Even the exact degree of undecidability of these two problems (which is measured by the *arithmetic hierarchy*) is known to be the same. Still, termination of term rewriting has been a very active research field in the last decades, producing many partial decision procedures for this problem. One of these techniques, which we want to highlight, is the *dependency pair framework* [35, 36, 103]. Unlike most other techniques, which try to prove or disprove termination in a single step, the dependency pair framework allows the modular combination of many small proof arguments; these "small proof arguments" are essentially formalised as *DP processors.* The existence of these techniques lead to the development of powerful tools which prove or disprove termination of term rewrite systems automatically. Since 2004, there exists an annual international termination competition with the purpose of stimulating the development of such tools. This competition has initially been hosted in Paris[1], and since 2008 in Innsbruck[2]. We mention the following automatic termination provers, which have participated in the standard term rewriting or *string rewriting* (which is essentially term rewriting with a restriction on the structure of terms) category in the termination competition during the last two years: AProVE [34], C*i*ME [15], Matchbox [108], MU-TERM [69], $\mathsf{T_TT_2}$ [64], and VMTL [95]. Most of these tools implement some variant of the dependency pair framework, and the most successful of them crucially depend on it.

Another property of programs in any model of computation is their *complexity.* Since for many programs, not only termination, but also termination within a reasonable time frame, and with a reasonable usage of resources, is desirable, the complexity of a program is also a very important property. Note that in general, the term "complexity" is ambiguous, since various measures of complexity exist. On a very general level, programs with *polynomially* bounded *time complexity* are deemed to be *feasible* [17]. A very natural measure of time complexity for term rewriting, which has been suggested by Hofbauer and Lautemann in [50], is the length of the longest sequence of rewrite steps starting from any term, dependent on that term's size. This is formalised by the notion of *derivational complexity.* Any complexity measure akin to worst case time complexity (such as derivational complexity) can be viewed as an extension of the property of termination: while termination only asserts that any computation by the program under consideration must finish at some point, the complexity measure expresses how soon this point must occur. Equal to termination, it is also undecidable whether the derivational complexity of a given term rewrite system is contained in a certain class of functions (for many important classes of functions). However, the degree of undecidability of each of these decision problems is different from the degree of undecidability of termination (it is neither strictly higher nor strictly lower, but incomparable, see Section 3.4 for details). It should also be noted that it is a priori not clear whether counting rewrite steps is an *invariant cost model*, i.e. whether the length of a sequence of rewrite steps is polynomially related to the length of an equivalent computation performed on a Turing Machine. However, this has been answered in the positive by Dal Lago and Martini [19], and by Avanzini and Moser [6].

Other complexity measures for term rewriting than derivational complexity have appeared in the literature, as well. For instance, it was suggested by Cichon and Lescanne in [14, 68] to only consider those terms as starting terms which represent a function call with values as arguments. Moreover, [14] proposed as a specific complexity measure the further restriction of values to natural numbers represented by the function symbols $\mathsf{0}$ and s (successor). In the vein of the above mentioned restriction on starting terms, Hirokawa and Moser later formally

[1]See `http://www.lri.fr/~marche/termination-competition/`

[2]See `http://termcomp.uibk.ac.at/`

defined the notion of *runtime complexity* [44]. We would also like to mention the concept of *implicit computational complexity* (see [9] for an overview of the research field around this concept, for instance), which does not consider the complexity of some specific algorithm using some specific computational device, but rather the complexity of the considered function itself, which essentially amounts to the worst case time complexity of the best algorithm computing the function. Finally, another notion, which measures *average case* time complexity (in contrast to the notions mentioned up to here, which are *worst case* time complexity measures) can be found in [13].

The main theme of this thesis is the analysis of upper bounds on the derivational complexity of term rewrite systems. Specifically, we focus on obtaining derivational complexity bounds by using termination proofs. For many known termination proof techniques (in particular, for most direct termination proof techniques), it is known that provability of termination of a term rewrite system via that technique implies some upper bound on the derivational complexity of that term rewrite system. We seek to expand this knowledge. In this regard, we put a particular emphasis on the dependency pair framework, but we also consider other termination proof techniques. Doing that is useful in several ways. First, it yields a way to automatically prove complexity bounds for term rewrite systems. Second, it reveals an aspect of the inherent power of termination proof techniques. Third, it may be useful to obtain more information about the inner workings of the analysed proof techniques.

Concerning the first of these points, the immediately obtained way to automatically prove complexity bounds for term rewrite systems works as follows: using the same techniques as automatic termination provers (or often, subsets or restrictions thereof, which yield tighter complexity bounds than the related full technique), termination of a given system is shown. The upper bound on its derivational complexity then follows immediately from the termination proof in conjunction with a general complexity result for the applied technique. Lifted from term rewrite systems to "real world programming languages", this immediately yields an opportunity for practical applicability whenever the inferred complexity bounds are low enough: since for many programs, feasibility is a desirable property (or even an expected property, so infeasibility would indicate a programming error; or, for instance in real-time systems, feasibility is even a strictly necessary property), the automatic confirmation of (feasible) complexity bounds is an important step in the verification of a program.

The second of the above mentioned points, the revelation of the inherent power of termination proof techniques, works on a more fundamental level. Upper bounds on the derivational complexity implied by the applicability of some termination proof technique immediately reveal a limit of the power of that technique: the technique can not be sufficient to prove termination of any system whose derivational complexity is beyond that bound. Some techniques even completely *characterise* a complexity class. So, termination proof techniques which characterise a larger complexity class are in a sense more powerful than techniques characterising only a smaller class of functions. This has been a motivation of Hofbauer and Lautemann to introduce derivational complexity [50]. Previous work in this direction has been done by Dershowitz and

Okada [24]. Note that still, this measure of strength does not entail a strict hierarchy of termination proof techniques. Hence, in many cases term rewrite systems exist whose termination can be proved using the "weaker" method, but not by the "stronger" one.

Finally, as mentioned above, the proof that some technique is a sound complexity proof technique might reveal more information about the inner workings of that technique than the proof of its soundness as a termination proof technique. This is particularly true for techniques whose soundness as a termination proof technique is shown via indirect, nonconstructive means, while soundness as a complexity proof technique is typically shown constructively. A prime example of this is the basic dependency pair method, whose soundness as a termination proof technique is usually shown in the literature via the nonexistence of *minimal infinite chains* (compare Theorem 2.44 below). The only information this yields about how termination in a term rewrite system comes about is the absurdity of its nontermination. In contrast, a proof of its soundness as a complexity proof technique would typically use more constructive means (it has to give a concrete complexity bound), which yield more information about the technique itself.

As examples, we mention some direct termination proof techniques for which complexity results are known. *Polynomial interpretations* [65, 66] are one of the oldest techniques for proving termination of term rewrite systems. They work by assigning natural numbers to terms in such a way that rewrite steps imply a decrease of the assigned number. The provability of termination of a term rewrite system by a polynomial interpretation implies a double exponential upper bound on its derivational complexity [50]. Another, recently introduced, well-known termination proof method is the technique of *matrix interpretations* [52, 27]. Matrix interpretations work on a similar principle as polynomial interpretations, but use vectors of natural numbers instead of natural numbers. It has been shown that in general, termination proofs by matrix interpretations imply an exponential upper bound on the derivational complexity of the considered term rewrite system [52, 27]. Finally, we mention *context dependent interpretations* [49], a generalisation of polynomial interpretations which has been developed with the intent of obtaining better complexity bounds. In [96, 79], two subclasses of context dependent interpretations have been identified (Δ-*restricted interpretations* and Δ-*linear interpretations*) which are suitable for automated search, and whose applicability guarantees a quadratic or exponential complexity bound, respectively.

In order to meet our goal of increasing the understanding about upper bounds on the derivational complexity of term rewrite systems, we take the following fourfold approach. First, we do a classification of the main problem considered in this thesis: given a term rewrite system, how difficult is it to decide whether its derivational complexity is contained within some fixed class of functions? In particular, how difficult is this problem compared to similar problems such as deciding termination of a given term rewrite system? The answers to these questions allow us to get a general understanding how the complexity analysis of particular term rewrite systems should be approached. Second, we consider variants of established termination proof techniques which were introduced with the specific intent of obtaining tighter upper bounds on the derivational complexity of term rewrite systems. Third, we give upper bounds on the derivational complexity of term rewrite systems whose termination can be shown by well-known proof techniques. Here we pay specific attention to the dependency pair framework. We analyse several ways of applying it to prove termination of term rewrite systems. Finally, we give an overview of $\mathsf{T_CT}$, an automatic complexity prover for term rewriting by Martin Avanzini, Georg Moser, and the author of this thesis. In $\mathsf{T_CT}$, the above mentioned restrictions of matrix interpretations together with various other techniques, which can be used to obtain upper bounds on the derivational complexity of term rewrite systems, are implemented.

More concretely, the main contributions of this thesis are the following:

1. We investigate the degree of undecidability of the problem of checking whether a certain class of functions is an upper bound on the derivational complexity of a given term rewrite system. We also perform this analysis for several extensions and variants of this decision problem, such as considering runtime complexity or implicit complexity instead of derivational complexity, restricting the upper complexity bound to a single concrete function, restricting to a particular strategy of choosing the next applied rewrite step (for terms which allow more than a single rewrite step), or considering the length of the shortest sequence of rewrite steps to a normal form (rather than the longest one). It turns out that each of these decision problems is as hard as its counterpart for Turing Machines. Although it is not obvious that the results about time complexity for Turing Machines can be transferred to derivational complexity for term rewriting, we put together all links which are necessary to show that this is indeed the case.

2. We identify a subclass of matrix interpretations called *triangular matrix interpretations*, which is a syntactic restriction of the class of matrix interpretations. If termination of a term rewrite system is provable by such a matrix interpretation, then its derivational complexity is bounded by a polynomial. Moreover, we establish a tight correspondence between a subclass of context dependent interpretations (which is a slight generalisation of the two subclasses introduced in [96, 79]) and a restriction of triangular matrix interpretations: whenever termination of a term rewrite system can be proved by one of the two methods, it can be proved by the other one, as well.

3. We establish upper bounds on the derivational complexity of term rewrite systems whose termination can be proved by various incarnations of the dependency pair framework. On one hand, we investigate the full dependency pair framework based on a selected set of DP processors; we chose this set such that it consists of the most important and fundamental DP processors. On the other hand, we also investigate the most basic version of the *dependency pair method*, as introduced in [2], possibly enhanced by some of the refinements described in [2, 36, 43] (*argument filterings*, *dependency graphs*, and *usable rules*). For all complexity bounds shown

for these methods, we also present examples witnessing that the bounds are essentially optimal.

This thesis is structured as follows (excluding this introduction). In Chapter 2, we recall well-known basics about term rewriting and other, related areas, which we will use in this thesis. This is also where we take the opportunity to fix the notation we employ for these notions. Item 1 of the above mentioned contributions is laid out in Chapter 3. Chapter 4 begins to cover the topic of complexity bounds inferred from the applicability of termination proof techniques. Here, we recall some classical termination proof techniques whose potential as complexity proof techniques has already been analysed in the literature. We use this chapter to recall and collect these results, focusing on direct proof techniques. Item 2 of the contributions, the investigation of the above mentioned restricted proof techniques, is covered by Chapter 5. In Chapter 6, we treat Item 3 of the contributions. Chapter 7 is devoted to an overview of the complexity prover TCT. Finally, we give a brief summary of the thesis in Chapter 8.

The main contributions listed above are independent of each other, so they can be read in any order. However, all of the chapters (not considering introduction and conclusion) depend on Chapter 2. Moreover, Chapters 5, 6, and 7 all assume knowledge of the termination proof techniques and complexity results presented in Chapter 4, and Chapter 7 additionally depends on Sections 5.2 and 5.4. So more or less, the interdependence of chapters in this thesis can be expressed by the following graph:

$$\begin{tikzcd}
 & 2 \arrow[dl] \arrow[dr] & & \\
3 & & 4 \arrow[dl] \arrow[dr] & \\
 & 5 \arrow[d] & & 6 \\
 & 7 & &
\end{tikzcd}$$

Chapter 2

Preliminaries

If in other sciences we should arrive at certainty without doubt and truth without error, it behooves us to place the foundations of knowledge in mathematics.

Roger Bacon

2.1 Orders and Relations

We recall and fix some well-known notions and notations for orders and relations, which are needed for this thesis.

Definition 2.1. Let $\to$ be a binary relation over some domain M. Then we call $\to$ *transitive* if for all $x, y, z \in M$ such that $x \to y$ and $y \to z$, we have $x \to z$. The relation $\to$ is *reflexive* if $x \to x$ for all $x \in M$, and it is *irreflexive* if $x \to x$ for no $x \in M$. It is *antisymmetric* if for all $x, y \in M$, $x \to y$ and $y \to x$ implies $x = y$. We call $\to$ *total* if for all $x, y \in M$, we have $x \to y$ or $y \to x$.

Definition 2.2. A *preorder* is a transitive and reflexive binary relation. We call a transitive, antisymmetric, and reflexive binary relation a *partial order*, and a transitive and irreflexive relation a *proper order*. A binary relation is a *total order* if it is transitive, antisymmetric, reflexive, and total.

Definition 2.3. Let $\to$ be a binary relation over some domain M. Then the *transitive closure* of $\to$, often denoted as $\to^+$, is the smallest transitive binary relation over M such that $\to \subseteq \to^+$. The *reflexive closure* of $\to$, often denoted as $\to^=$, is the smallest reflexive binary relation over M such that $\to \subseteq \to^=$. We write $\to^*$ for the *reflexive and transitive closure* of $\to$, which is the smallest reflexive and transitive binary relation such that $\to \subseteq \to^*$. For $n \in \mathbb{N}$, we often denote the *n-fold composition* of $\to$ as $\to^n$, i.e., for $x, y \in M$, $x \to^0 y$ if and only if $x = y$, and $x \to^{n+1} y$ if and only if there exists some $z \in M$ such that $x \to^n z$ and $z \to y$.

It is easy to see that for any binary relation $\to$, we have $\to^+ = \bigcup_{n \geqslant 1} \to^n$, $\to^= = \to^0 \cup \to^1$, and $\to^* = \bigcup_{n \in \mathbb{N}} \to^n$.

Definition 2.4. Let $\to_\alpha$ and $\to_\beta$ be binary relations over some domain M. Then the *composition of* $\to_\alpha$ *and* $\to_\beta$, denoted as $\to_\alpha \cdot \to_\beta$, is defined such that for all $x, y \in M$, we have $x \to_\alpha \cdot \to_\beta y$ if and only if there exists some $z \in M$ such that $x \to_\alpha z$ and $z \to_\beta y$.

The relation $\to_\alpha$ *relative to* $\to_\beta$ (also called $\to_\alpha$ *modulo* $\to_\beta$), is denoted as $\to_\alpha / \to_\beta$. It is defined by $\to_\alpha / \to_\beta = \to_\beta^* \cdot \to_\alpha \cdot \to_\beta^*$.

We often write $x \to_\alpha y \to_\beta z$ in order to indicate that $x \to_\alpha y$ and $y \to_\beta z$.

Definition 2.5. A binary relation $\to$ over a domain M is called *well-founded* if there exists not infinite sequence $t_0, t_1, \ldots$ of elements of M such that $t_i \to t_{i+1}$ for all $n \in \mathbb{N}$. It is *finitely branching* if for all $s \in M$, the set $\{t \mid s \to t\}$ is finite.

Definition 2.6. Let M be a set. Then a *multiset* X over M is a mapping associating a natural number $X(m)$ (called the *multiplicity* of m) with every $m \in M$. We say that some $m \in M$ is an *element of* X (denoted as $m \in X$) if $X(m) > 0$. A multiset X is a *subset* of a multiset Y (we write $X \subseteq Y$ in that case) if for all $m \in M$, we have $X(m) \leqslant Y(m)$. The *union* of two multisets X and Y (written as $X \cup Y$) is the multiset Z such that $Z(m) = X(m) + Y(m)$ for all $m \in M$. The *difference* of two multisets X and Y (written as $X \setminus Y$) is the multiset Z such that $Z(m) = \max\{X(m) - Y(m), 0\}$ for all $m \in M$.

Sometimes we explicitly write down multisets in a set-style notation, e.g. $M = \{\{x_1, \ldots, x_n\}\}$. Then $M(m)$ is defined to be the number of elements x_i of the given sequence such that $m = x_i$.

Definition 2.7. The *multiset extension* of a proper order $>$, denoted as $>^{\text{mul}}$, is defined to be the following order over multisets over M: given two multisets X and Y over M, we have $X >^{\text{mul}} Y$ if and only if there exists a multiset Z such that $Z \subseteq X$, $Z \subseteq Y$, and there exists some $x \in X \setminus Z$ such that $x > y$ for all $y \in Y \setminus Z$.

The *lexicographic extension* of a or proper order $>$, denoted as $>^{\text{lex}}$, is the following order over tuples of length n over M: we have $(x_1, \ldots, x_n) >^{\text{lex}} (y_1, \ldots, y_n)$ if and only if there exists some $1 \leqslant i \leqslant n$ such that $x_j = y_j$ for all $1 \leqslant j < i$, and $x_i > y_i$.

2.2 Computability and Recursion Theory

2.2.1 Turing Machines

In this section, we introduce the essentials of Turing Machines, see [93], for instance. Moreover, this section serves the purpose of fixing the specific formalisation of Turing Machines used throughout this thesis.

Definition 2.8. A *(deterministic single-tape) Turing Machine* is defined to be a triple (Q, Σ, δ), where

- Q is a finite set of *states* containing at least three distinct states q_s (the *starting state*), q_a (the *accept state*), and q_r (the *reject state*),

- Σ is a finite set of *tape symbols* containing at least two distinct symbols $\Box$ (the *blank symbol*) and $\vdash$ (the *left end marker*), and
- δ is a function from $Q \setminus \{q_a, q_r\} \times \Sigma$ to $Q \times \Sigma \times \{L, R\}$ and is called the *transition function*. It must be defined such that for all $q \in Q \setminus \{q_a, q_r\}$, we have $\delta(q, \vdash) = (q', \vdash, R)$ for some $q' \in Q$ (here L represents a move to the left, and R a move to the right).

A *(deterministic dual-tape) Turing Machine* is a triple (Q, Σ, δ), identical to a single-tape Turing Machine, except that δ is a function from $Q \setminus \{q_a, q_r\} \times \Sigma \times \Sigma$ to $Q \times \Sigma \times \{L, R\} \times \Sigma \times \{L, R\}$. We assume that for all $q \in Q \setminus \{q_a, q_r\}$ and $b \in \Sigma$, we have $\delta(q, \vdash, b) = (q', \vdash, R, b', x')$ and $\delta(q, b, \vdash) = (q'', b'', x'', \vdash, R)$ for some $q', q'' \in Q$, $b', b'' \in \Sigma$, and $x', x'' \in \{L, R\}$.

Definition 2.9. A *configuration* of a single-tape Turing Machine is a triple (q, w, i), where

- $q \in Q$ (q denotes the current state),
- $w = \vdash w'$ with $w' \in \Sigma^*$ (w denotes the current content of the tape, apart from infinitely many $\Box$ symbols to the right of w), and
- $i \in \{1 \dots |w|\}$ (i is the current position of the scanner).

A configuration of a dual-tape Turing Machine is a quintuple (q, w, i, v, j), where

- q, w, and i are defined as for single-tape Turing Machines,
- $v = \vdash v'$ with $v' \in \Sigma^*$ (v denotes the current content of the second tape, except for infinitely many $\Box$ symbols to the right of v), and
- $j \in \{1 \dots |v|\}$ (j is the current position of the scanner on the second tape).

A *start configuration* of a single-tape Turing Machine is a configuration (q, w, i) such that $q = q_s$, $w = \vdash w'$ for some *input word* w', and $i = 1$. A start configuration of a dual-tape Machine is a configuration (q, w, i, v, j) such that $q = q_s$, $w = \vdash w'$, $i = 1$, $v = \vdash$, and $j = 1$. The *size* of a configuration $\alpha = (q, w, i)$ of a single-tape Machine, denoted as $|\alpha|$, is the length of w (also denoted as $|w|$). For dual tape Machine configurations $\alpha = (q, w, i, v, j)$, we set $|\alpha| = |w| + |v|$.

Definition 2.10. We recall the following standard semantic notions of Turing Machines, which we will use liberally throughout the thesis. Let $M = (Q, \Sigma, \delta)$ be a (single-tape or dual-tape) Turing Machine.

1. The single-tape Machine M *moves in a single step* from a configuration (q, w, i) to a configuration (q', w', i') if $w = w_1 \dots w_{|w|}$, $w' = w'_1 \dots w'_{|w'|}$, $\delta(q, w_i) = (q', w'_i, x)$, $w_j = w'_j$ for all $j \neq i$, and either $x = L$ and $i' = i - 1$, or $x = R$ and $i' = i + 1$.

2. The dual-tape Machine M *moves in a single step* from a configuration (q, w, i, v, j) to a configuration (q', w', i', v', j') if $w = w_1 \dots w_{|w|}$, $v = v_1 \dots v_{|v|}$, $w' = w'_1 \dots w'_{|w'|}$, $v' = v'_1 \dots v'_{|v'|}$, $\delta(q, w_i, v_j) = (q', w'_i, x, v'_j, y)$, $w_k = w'_k$ for all $k \neq i$, $v_l = v'_l$ for all $l \neq j$, either $x = L$ and $i' = i - 1$, or $x = R$ and $i' = i + 1$, and either $y = L$ and $j' = j - 1$, or $y = R$ and $j' = j + 1$.

3. The Machine M *moves (in n steps)* from a configuration α_0 to a configuration α_n if there exist configurations $\alpha_1, \dots, \alpha_{n-1}$ such that M moves from α_{i-1} to α_i in a single step for all $1 \leqslant i \leqslant n$.

4. The single-tape (respectively dual-tape) Machine M *halts on configuration* α if it moves from α to a configuration (q, w, i) (respectively (q, w, i, v, j)) such that $q \in \{q_a, q_r\}$.

5. The Machine M *halts on input word w* if it halts on the start configuration for the input word w. It *halts (universally)* if it halts on all words over Σ.

6. The single-tape (respectively dual-tape) Machine M *accepts* an input word x if it moves from the start configuration for x to a configuration (q, w, i) (respectively (q, w, i, v, j)) such that $q = q_a$. It *rejects* x if it moves from the start configuration for x to a configuration (q, w, i) (respectively (q, w, i, v, j)) such that $q = q_r$. Given a language $\mathcal{L}$, the Machine M *accepts exactly* $\mathcal{L}$ if for every input word x, M accepts x if and only if $x \in \mathcal{L}$.

Finally, given a (single-tape or dual-tape) Turing Machine M, we define the functions $\textsc{Time}_M$ and $\textsc{LifeTime}_M$ gauging two measures of time complexity. For the second function, we chose the name $\textsc{LifeTime}_M$ to reflect the close relationship to the Turing Machine *mortality problem* (see [41], for instance), which asks whether a given Turing Machine halts on all configurations.

Definition 2.11. Let M be a (single-tape or dual-tape) Turing Machine. Then we define the mappings $\textsc{Time}_M, \textsc{LifeTime}_M : \mathbb{N} \to \mathbb{N}$ as follows:

$$\begin{aligned}\textsc{Time}_M(n) &= \max\{m \mid M \text{ runs } m \text{ steps on input word } x \wedge |x| \leqslant n\}\\ \textsc{LifeTime}_M(n) &= \max\{m \mid M \text{ runs } m \text{ steps on configuration } \alpha \wedge |\alpha| \leqslant n\}\end{aligned}$$

If there exists any input word of length at most n (respectively, a configuration α with $|\alpha| \leqslant n$) on which M does not halt, then $\textsc{Time}_M(n)$ (respectively, $\textsc{LifeTime}_M(n)$) is undefined.

2.2.2 The Arithmetical Hierarchy

We now recapitulate the arithmetical hierarchy, see [93, 28], for instance.

Definition 2.12. Let $n \in \mathbb{N}$. A set (or, equivalently, a language, or a decision problem) $A \subseteq \mathbb{N}$ *is in* Σ^0_n if there exists an $(n + 1)$-ary decidable predicate[1]

[1]The predicate may be chosen to be primitive recursive without changing the notions defined.

$P(x_1, \ldots, x_n, x_{n+1})$ such that A is exactly the subset of $\mathbb{N}$ for which the unary predicate

$$\exists x_1 \forall x_2 \ldots Q x_n \; P(x_1, \ldots, x_n, x_{n+1})$$

holds, where Q is either $\exists$ or $\forall$ depending on whether n is odd or even.

The set A *is in* Π^0_n if there exists an $(n+1)$-ary decidable predicate P such that A is exactly the subset of $\mathbb{N}$ for which

$$\forall x_1 \exists x_2 \ldots Q x_n \; P(x_1, \ldots, x_n, x_{n+1})$$

holds, where Q is $\forall$ or $\exists$, depending on whether n is odd or even.

For decision problems over countable domains other than $\mathbb{N}$ (such as tuples of natural numbers, Turing Machines, terms, or TRSs, for instance), we often tacitly assume the existence of a Gödel numbering, which makes the decision problem equivalent to one over $\mathbb{N}$. Thus, the definitions of this section naturally extend to decision problems over any such domain.

Note that Δ^0_0 is exactly the set of decidable languages, and Σ^0_1 is the set of recursively enumerable languages.

Definition 2.13. Let $n \in \mathbb{N}$. A set $A \subseteq \mathbb{N}$ is Σ^0_n-*hard* (respectively Π^0_n-*hard*) if for every set B in Σ^0_n (respectively, for every set B in Π^0_n), there exists a computable function f such that $x \in B$ if and only if $f(x) \in A$ for all $x \in \mathbb{N}$.

Definition 2.14. Let $n \in \mathbb{N}$. A set $A \subseteq \mathbb{N}$ is Σ^0_n-*complete* (respectively Π^0_n-*complete*) if it is both contained in Σ^0_n (respectively in Π^0_n) and Σ^0_n-hard (respectively Π^0_n-hard).

Proposition 2.15 ([93, Theorem 13-VIII])**.** *The following decision problem* Fin *is* Σ^0_2*-complete:*

Instance: A (single-tape) Turing Machine M

Question: Is the number of inputs on which M halts finite?

2.2.3 Primitive and Multiple Recursion

We recall some essentials of recursion theory, compare [89, 94]. We call the following functions over $\mathbb{N}$ *initial functions*: the zero functions $z_n(x_1, \ldots, x_n) = 0$ of all arities, the unary successor function $s(x) = x + 1$, and all projection functions $\pi^i_n(x_1, \ldots, x_n) = x_i$ for $1 \leqslant i \leqslant n$. A class $\mathcal{C}$ of functions over $\mathbb{N}$ is *closed under composition* if for all $f \colon \mathbb{N}^m \to \mathbb{N}$ and $g_1, \ldots, g_m \colon \mathbb{N}^n \to \mathbb{N}$ in $\mathcal{C}$, the function $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$ is in $\mathcal{C}$, as well. It is *closed under primitive recursion* if for all $f \colon \mathbb{N}^n \to \mathbb{N}$ and $g \colon \mathbb{N}^{n+2} \to \mathbb{N}$, the function h defined by $h(0, x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ and $h(y+1, x_1, \ldots, x_n) = f(y, h(y, x_1, \ldots, x_n), x_1, \ldots, x_n)$ is contained in $\mathcal{C}$, as well. The *k-ary Ackermann function* A_k for $k \geqslant 2$ is defined recursively as follows:

$$\begin{aligned}
A_k(0, \ldots, 0, x_k) &= x_k + 1 \\
A_k(x_1, \ldots, x_{k-1} + 1, 0) &= A_k(x_1, \ldots, x_{k-1}, 1) \\
A_k(x_1, \ldots, x_{k-1} + 1, x_k + 1) &= A_k(x_1, \ldots, x_{k-1}, A_k(x_1, \ldots, x_{k-1} + 1, x_k)) \\
A_k(x_1, \ldots, x_i + 1, 0, \ldots, 0, x_k) &= A_k(x_1, \ldots, x_i, x_k, 0, \ldots, 0, x_k)
\end{aligned}$$

Here, the last equation is a schema instantiated for all $1 \leqslant i \leqslant k-2$. The set of *primitive recursive functions* is the smallest set of functions over $\mathbb{N}$ which contains all initial functions and is closed under composition and primitive recursion. The set of *multiply recursive functions* is the smallest set of functions over $\mathbb{N}$ which contains all initial functions and k-ary Ackermann functions, and is closed under composition and primitive recursion.

The k-ary Ackermann functions form a hierarchy which completely captures the set of multiply recursive functions in the following sense:

Theorem 2.16 ([94], Chapter 1)**.** *For every multiply recursive function f, there exists a k such that $A_k(n) \notin \mathsf{O}(f(n))$.*

2.3 Term Rewrite Systems

In the following sections, we introduce the notions of term rewriting used throughout this thesis. Their purpose is to keep the thesis self-contained rather than to give a complete treatment of term rewriting. More general introductions to term rewriting (including the concepts defined in this section, unless mentioned otherwise) can be found in [8, 102], for instance.

Terms form the basic data structures on which computations by term rewrite systems are performed. They are formally built up as follows.

Definition 2.17. Let $\mathcal{F}^0, \mathcal{F}^1, \ldots$ be a family of pairwise disjoint sets. Then $\mathcal{F} = \bigcup_{i \in \mathbb{N}} \mathcal{F}^i$ is a *signature*. For each $f \in \mathcal{F}$, we call f a *function symbol*, and we say that the *arity* of f is n such that $f \in \mathcal{F}^n$. A *constant* is a function symbol of arity 0.

Throughout this thesis, unless specifically stated otherwise, we assume signatures to be finite and to contain at least one constant.

Definition 2.18. Let $\mathcal{F}$ be a signature, and $\mathcal{V}$ a countably infinite set disjoint from $\mathcal{F}$, whose elements we call *variables*. The *set of terms built up from $\mathcal{F}$ and $\mathcal{V}$* (denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$) is the smallest set fulfilling the following conditions:

1. For each $x \in \mathcal{V}$, we have $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.
2. For each $f \in \mathcal{F}$, if n is the arity of f and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, then $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

Throughout this thesis, we assume $\mathcal{F}$ to be a signature, and $\mathcal{V}$ a set of variables satisfying the conditions of Definition 2.18. Unless mentioned otherwise, we assume terms to be built up from $\mathcal{F}$ and $\mathcal{V}$. For terms of the shape $f(\ldots(f(t))\ldots)$ where f is repeated n times, we use the abbreviation $f^n(t)$.

Definition 2.19. We now recall some fundamental properties of terms, which will be liberally used in this thesis.

1. The *set of function symbols in a term t*, denoted as $\mathcal{F}\mathsf{un}(t)$, is defined as:

$$\mathcal{F}\mathsf{un}(t) = \begin{cases} \emptyset & \text{if } t \in \mathcal{V} \\ \{f\} \cup \bigcup_{1 \leqslant i \leqslant n} \mathcal{F}\mathsf{un}(t_i) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

2. The *set of variables in a term* t, denoted as $\mathcal{V}\mathsf{ar}(t)$, is:
$$\mathcal{V}\mathsf{ar}(t) = \begin{cases} \{t\} & \text{if } t \in \mathcal{V} \\ \bigcup_{1 \leqslant i \leqslant n} \mathcal{V}\mathsf{ar}(t_i) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

3. The *root symbol* of a term t, (notation: $\mathsf{rt}(t)$), is t if $t \in \mathcal{V}$, and f if $t = f(t_1, \dots, t_n)$.

4. We say that s is a *subterm* of t (writing $s \trianglelefteq t$) if either $s = t$ or $t = f(t_1, \dots, t_n)$ such that $s \trianglelefteq t_i$ for some $1 \leqslant i \leqslant n$. The *strict subterm relation* is defined by $s \lhd t \iff s \trianglelefteq t \wedge s \neq t$. The inverse of the subterm relation is called the *superterm relation.*

5. A term t such that $\mathcal{V}\mathsf{ar}(t) = \emptyset$ is a *ground term.* The set of ground terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is denoted as $\mathcal{T}(\mathcal{F})$.

6. A *position* is a finite sequence of natural numbers. The set of positions of a term t is the following:
$$\mathcal{P}\mathsf{os}(t) = \begin{cases} \{\epsilon\} & \text{if } t \in \mathcal{V} \\ \{\epsilon\} \cup \bigcup\limits_{\substack{1 \leqslant i \leqslant n \\ p \in \mathcal{P}\mathsf{os}(t_i)}} ip & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$
Here we use ϵ to denote the empty sequence. For $p \in \mathcal{P}\mathsf{os}(t)$, we write $t|_p$ to denote the *subterm of* t *at position* p. We have $t|_\epsilon = t$ and, for $t = f(t_1, \dots, t_n)$, $t|_{ip} = t_i|_p$. We use a partial order $\leqslant$ on positions such that $p \leqslant q$ if and only if p is a prefix of q. The strict part of $\leqslant$ is denoted as $<$, i.e. $p < q \iff p \leqslant q \wedge p \neq q$. We use the two abbreviations $\mathcal{P}\mathsf{os}_{\mathcal{F}}(t) = \{p \in \mathcal{P}\mathsf{os}(t) \mid \mathsf{rt}(t) \in \mathcal{F}\}$ and $\mathcal{P}\mathsf{os}_{\mathcal{V}}(t) = \{p \in \mathcal{P}\mathsf{os}(t) \mid \mathsf{rt}(t) \in \mathcal{V}\}$. We call two positions p and q *parallel* (writing $p \parallel q$) if neither $p \leqslant q$ nor $p \geqslant q$ holds.

7. The *size* of a term t, denoted as $|t|$ is the number of function symbol and variable occurrences in t. Formally, it is defined as follows:
$$|t| = \begin{cases} 1 & \text{if } t \in \mathcal{V} \\ 1 + \sum_{1 \leqslant i \leqslant n} |t_i| & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

8. The *depth* of a term t, denoted as $\mathsf{dp}(t)$, is the the length of the longest simple path from the root to any leaf in the parse tree of t. More formally:
$$\mathsf{dp}(t) = \begin{cases} 0 & \text{if } t \in \mathcal{V} \\ \max(\{0\} \cup \{1 + \mathsf{dp}(t_i) \mid 1 \leqslant i \leqslant n\}) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

9. The *number of occurrences* of a variable or function symbol x in a term t, denoted as $|t|_x$, is the following:
$$|t|_x = \begin{cases} 1 & \text{if } t = x \\ 0 & \text{if } t \in \mathcal{V} \text{ and } t \neq x \\ 1 + \sum_{i=1}^n |t_i|_x & \text{if } t = f(t_1, \dots, t_n) \text{ and } f = x \\ \sum_{i=1}^n |t_i|_x & \text{if } t = f(t_1, \dots, t_n) \text{ and } f \neq x \end{cases}$$

Two further important concepts, which are implicitly applied during each step in term rewriting, are *contexts* and *substitutions.*

Definition 2.20. A *substitution* is a mapping $\sigma\colon \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that the set $\{x \mid \sigma(x) \neq x\}$ is finite. A substitution σ is canonically extended to terms as follows (to denote the application of a substitution σ on a term t, we use the notation $t\sigma$):

$$t\sigma = \begin{cases} \sigma(t) & \text{if } t \in \mathcal{V} \\ f(t_1\sigma, \dots, t_n\sigma) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

Definition 2.21. Let $\Diamond$ be a fresh function symbol of arity 0, called the *hole.* A *context* is a term in $\mathcal{T}(\mathcal{F} \cup \{\Diamond\}, \mathcal{V})$ containing exactly one occurrence of $\Diamond$. For a context C and term t, the term $C[t]$ denotes the replacement of $\Diamond$ by t.

$$C[t] = \begin{cases} t & \text{if } C = \Diamond \\ f(s_1, \dots, s_i[t], \dots, s_n) & \text{if } C = f(s_1, \dots, s_n) \text{ and } \Diamond \in \mathcal{F}\mathsf{un}(s_i), \\ & \quad \text{i.e. } s_i \text{ is a context} \end{cases}$$

These concepts are enough to define *rewrite steps*, the primitive operation of term rewriting.

Definition 2.22. A *rewrite rule* over a signature $\mathcal{F}$ and a variable set $\mathcal{V}$ is a pair $l \to r$ of terms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $l \notin \mathcal{V}$ and $\mathcal{V}\mathsf{ar}(l) \supseteq \mathcal{V}\mathsf{ar}(r)$. A *term rewrite system* (TRS for short) over a signature $\mathcal{F}$ and a variable set $\mathcal{V}$ is a set of rewrite rules over $\mathcal{F}$ and $\mathcal{V}$. A TRS $\mathcal{R}$ induces the following binary *rewrite relation* $\to_{\mathcal{R}}$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$: we have $s \to_{\mathcal{R}} t$ if and only if there exist a context C, a substitution σ, and a rewrite rule $l \to r \in \mathcal{R}$ such that

$$s = C[l\sigma] \text{ and } t = C[r\sigma] \ .$$

The pair $s \to_{\mathcal{R}} t$ is also called a *rewrite step.* We call a sequence of rewrite steps $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \dots \to_{\mathcal{R}} t_n$ a *rewrite sequence* or a *derivation.* The position p such that $C|_p = \Diamond$ is the *redex position* of this rewrite step. The set of *normal forms* of $\mathcal{R}$ (or normal forms of $\to_{\mathcal{R}}$) is defined by $\mathsf{NF}(\mathcal{R}) = \{s \mid \nexists t \ s \to_{\mathcal{R}} t\}$. If for a rewrite rule $l \to r \in \mathcal{R}$ and a rewrite step $C[l\sigma] \to_{\mathcal{R}} C[r\sigma]$, we have $t \in \mathsf{NF}(\mathcal{R})$ for all $t \lhd l\sigma$, then we also call the step an *innermost rewrite step*, denoted as $C[l\sigma] \xrightarrow{\mathsf{i}}_{\mathcal{R}} C[r\sigma]$.

Unless mentioned otherwise, we assume rewrite rules and TRSs to be defined over $\mathcal{F}$ and $\mathcal{V}$ such that $\mathcal{F}$ consists exactly of the function symbols occurring in the TRS under consideration, and $\mathcal{V}$ is some countably infinite set including all variables occurring in that TRS. If $\mathcal{F}$ contains only unary and constant function symbols, we also call the TRS under consideration a *string rewrite system*[2] (SRS for short). When giving the rules of a TRS, we generally use sans serif font for function symbols, and roman font for variables. Unless specifically

[2]This is sometimes called *unary rewriting*, as opposed to the notion of string rewriting commonly used in the literature, where only unary function symbols are allowed. All results about string rewriting presented in this thesis hold for both flavours of string rewriting.

stated otherwise, we also assume TRSs to be finite throughout this thesis. If the TRS $\mathcal{R}$ is clear from the context, we sometimes write $\to$ instead of $\to_{\mathcal{R}}$. We write $s \xrightarrow{p} t$ to indicate the redex position p of the rewrite step $s \to t$. We write $s \xrightarrow{>\epsilon} t$ to indicate that the redex position of this rewrite step is not ϵ. If we wish to indicate the applied rewrite rule $l \to r$ in a rewrite step $s \to t$, we write $s \to_{l \to r} t$. In order to express that the reduct t of a rewrite step $s \to_{\mathcal{R}} t$ is a normal form of $\mathcal{R}$, we write $s \to_{\mathcal{R}}^{!} t$. We use $s \to_{\mathcal{R}}^{n,!}$ if $s \to_{\mathcal{R}}^{n} t$ and t is a normal form of $\mathcal{R}$.

In the sequel, we sometimes refer to specific subclasses of rewrite rules and TRSs, which are defined by simple syntactic criteria. We now define those such subclasses which we use in the remainder of this thesis.

Definition 2.23. A term t is called *linear* if for all $x \in \mathcal{V}\mathsf{ar}(t)$, we have $|t|_x = 1$. A rewrite rule $l \to r$ is called *left-linear* if l is linear, *right-linear* if r is linear, and *linear* if both l and r are linear. A TRS $\mathcal{R}$ is called left-linear (respectively right-linear, linear) if all rewrite rules of $\mathcal{R}$ are left-linear (respectively right-linear, linear).

A rewrite rule $l \to r$ is called *duplicating* if there exists a variable $x \in \mathcal{V}\mathsf{ar}(l)$ such that $|l|_x < |r|_x$. A TRS is called duplicating if at least one of its rewrite rules is duplicating.

A rewrite rule $l \to r$ is called *erasing* if $\mathcal{V}\mathsf{ar}(l) \supset \mathcal{V}\mathsf{ar}(r)$. We say that a TRS is erasing if at least one of its rewrite rules is erasing.

Given a rewrite step $s \to t$, in order to formalise whether two positions in s and t are the "same", the notion of *descendants* is used.

Definition 2.24. Let $A\colon s \xrightarrow{p'}_{l \to r} t$ be a rewrite step, and let $p \in \mathcal{P}\mathsf{os}(s)$. Then the *descendants of p in t* (denoted by $p \backslash A$) are defined as follows:

$$p \backslash A = \begin{cases} \{p\} & \text{if } p < p' \text{ or } p \parallel p', \\ \{p' q_3 q_2 \mid r|_{q_3} = l|_{q_1}\} & \text{if } p = p' q_1 q_2 \text{ with } q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(l), \\ \emptyset & \text{otherwise .} \end{cases}$$

For a set $P \subseteq \mathcal{P}\mathsf{os}(s)$, we define $P \backslash A = \bigcup_{p \in P} p \backslash A$. For derivations $A : s \to^* t$, we set $p \backslash A = \{p\}$, if A is the empty derivation. Otherwise, we can split A into $A_1 : s \to s'$ and $A_2 : s' \to^* t$, and set $p \backslash A = \{(p \backslash A_1) \backslash A_2\}$.

We also need a richer definition of descendants below. In contrast to descendants, this notion should consider the positions destroyed and created by rewrite steps, as well. We now give a simplified version of *progenies*, which are defined in [80].

Definition 2.25. Let $A\colon s \xrightarrow{p'}_{l \to r} t$ be a rewrite step, and let $p \in \mathcal{P}\mathsf{os}(s)$. Then the *simple progenies of p in t* (denoted by $p \mathbin{\backslash\!\backslash^{\mathrm{s}}} A$) are defined as follows:

$$p \mathbin{\backslash\!\backslash^{\mathrm{s}}} A = \begin{cases} \{p\} & \text{if } p < p' \text{ or } p \parallel p', \\ \{p' q_3 q_2 \mid r|_{q_3} = l|_{q_1}\} & \text{if } p = p' q_1 q_2 \text{ with } q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(l), \\ \{p q_1 \mid q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)\} & \text{if } p = p', \\ \emptyset & \text{otherwise .} \end{cases}$$

For a set $P \subseteq \mathcal{P}\mathsf{os}(s)$, we define $P \Lsh\!\!\backslash^{\mathrm{s}} A = \bigcup_{p \in P} p \backslash\!\!\backslash^{\mathrm{s}} A$. For derivations $A : s \to^* t$, we set $p \backslash\!\!\backslash^{\mathrm{s}} A = \{p\}$, if A is the empty derivation. Otherwise, we can split A into $A_1 : s \to s'$ and $A_2 : s' \to^* t$, and set $p \backslash\!\!\backslash^{\mathrm{s}} A = \{(p \backslash\!\!\backslash^{\mathrm{s}} A_1) \backslash\!\!\backslash^{\mathrm{s}} A_2\}$.

For similar modified definitions of the notion of descendants in the literature, which also include the positions destroyed and created by rewrite steps in some way, see [107] or [102, Section 8.6], for instance.

Definition 2.26. Let $\mathcal{R}$ be a TRS based on $\mathcal{F}$ and $\mathcal{V}$. Then the *defined symbols* of $\mathcal{R}$ are collected in the set $\mathcal{D}_{\mathcal{R}} = \{f \mid \mathsf{rt}(l) = f \text{ for some rewrite rule } l \to r \in \mathcal{R}\}$. The *constructors* of $\mathcal{R}$ are defined by $\mathcal{C}_{\mathcal{R}} = \mathcal{F} \setminus \mathcal{D}_{\mathcal{R}}$. The set of *basic terms* of $\mathcal{R}$ is defined to be the following set $\mathcal{B}_{\mathcal{R}}$:

$$\mathcal{B}_{\mathcal{R}} = \{f(v_1, \ldots, v_n) \mid f \in \mathcal{D}_{\mathcal{R}} \wedge v_1, \ldots, v_n \in \mathcal{T}(\mathcal{C}_{\mathcal{R}})\}$$

We call $\mathcal{R}$ a *constructor TRS* if for each rule $l \to r \in \mathcal{R}$, we have $l = f(l_1, \ldots, l_n)$ for some $f \in \mathcal{D}_{\mathcal{R}}$ and $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{C}_{\mathcal{R}}, \mathcal{V})$.

If $\mathcal{R}$ is clear from context, we often write $\mathcal{D}$, $\mathcal{C}$, $\mathcal{B}$ instead of $\mathcal{D}_{\mathcal{R}}$, $\mathcal{C}_{\mathcal{R}}$, and $\mathcal{B}_{\mathcal{R}}$, respectively.

We recall the notion of *relative rewriting*, cf. [31] or [102, Section 6.5.3].

Definition 2.27. Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs. The induced *relative rewrite relation* of $\mathcal{R}$ modulo $\mathcal{S}$ is defined by $\to_{\mathcal{R}/\mathcal{S}} = \to_{\mathcal{R}}/\to_{\mathcal{S}}$. The set of *normal forms* of $\mathcal{R}/\mathcal{S}$ is defined by $\mathsf{NF}(\mathcal{R}/\mathcal{S}) = \{s \mid \nexists t \; s \to_{\mathcal{R}/\mathcal{S}} t\}$.

Definition 2.28. A TRS $\mathcal{R}$ is *terminating* (or *strongly normalising*) if its rewrite relation is well-founded. A term t is *terminating with respect to* $\mathcal{R}$ if there exists no infinite sequence $t_0, t_1, \ldots$ such that $t = t_0$ and $t_i \to t_{i+1}$ for all $i \in \mathbb{N}$.

Termination analysis for TRSs has received a lot of attention during the last years, see [2, 42, 43, 33, 103, 27, 61], for instance. Most direct termination proof techniques are based on *reduction orders*, cf. [8, 102].

Definition 2.29. A binary relation $>$ on terms is *closed under substitutions* if $s > t$ implies $s\sigma > t\sigma$ for all substitutions σ. On the other hand, $>$ is *closed under contexts* if $s > t$ implies $C[s] > C[t]$ for all contexts C.

Definition 2.30. A *reduction order* is a proper order $\succ$ on terms which is well-founded and closed under substitutions and contexts.

Definition 2.31. A TRS $\mathcal{R}$ is *compatible* with a binary relation $\succ$ if for all rewrite rules $l \to r \in \mathcal{R}$, we have $l \succ r$.

The following classical theorem describes the generality of reduction orders:

Theorem 2.32. *A TRS $\mathcal{R}$ is terminating if and only if there exists a reduction order which is compatible with $\mathcal{R}$.*

Definition 2.33. The *derivation tree* of a term s with respect to a TRS $\mathcal{R}$ is the following directed graph: the nodes are all terms t such that $s \to_{\mathcal{R}}^{*} t$, and there exists an edge from t to t' if and only if $t \to_{\mathcal{R}} t'$. A derivation tree is *non-circular* if no path starting from the root in the tree contains the same term more than once.

Observe that if s is terminating with respect to a TRS $\mathcal{R}$, then the derivation tree of s with respect to $\mathcal{R}$ is finite and non-circular, and s is its single source node.

While termination of a TRS $\mathcal{R}$ certifies that $\mathcal{R}$ only gives rise to finite derivations, the length of these derivations is measured by the *derivational complexity* function of $\mathcal{R}$, see e.g. [50].

Definition 2.34. The *derivation height* of a term s with respect to a finitely branching, well-founded binary relation $\to$ is defined as $\mathsf{dh}(s, \to) = \max\{n \mid \exists t \; s \to^{n} t\}$. The *derivational complexity* of a TRS $\mathcal{R}$ over $\mathcal{F}$ and $\mathcal{V}$ is the following function $\mathsf{dc}_{\mathcal{R}} : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{dc}_{\mathcal{R}}(n) = \max\{\mathsf{dh}(t, \to_{\mathcal{R}}) \mid t \in \mathcal{T}(\mathcal{F}) \wedge |t| \leqslant n\}$$

The *innermost derivational complexity* of $\mathcal{R}$ is the following function $\mathsf{idc}_{\mathcal{R}}$:

$$\mathsf{idc}_{\mathcal{R}}(n) = \max\{\mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}(\mathcal{F}) \wedge |t| \leqslant n\}$$

Another measure akin to derivational complexity is the notion of *runtime complexity*. This is a natural complexity measure if term rewriting is seen as a first-order functional programming language: it only takes into account as starting terms of derivations those terms which model single function calls. This has been suggested in [14, 68] and formally cast into a definition in [44], for example.

Definition 2.35. The *runtime complexity* function of a TRS $\mathcal{R}$ is

$$\mathsf{rc}_{\mathcal{R}}(n) = \max\{\mathsf{dh}(t, \to_{\mathcal{R}}) \mid t \in \mathcal{B}_{\mathcal{R}} \wedge |t| \leqslant n\} \; .$$

Likewise, the *innermost runtime complexity* of $\mathcal{R}$ is the function

$$\mathsf{irc}_{\mathcal{R}}(n) = \max\{\mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid t \in \mathcal{B}_{\mathcal{R}} \wedge |t| \leqslant n\} \; .$$

Generally, we use sets of number theoretic functions to express upper bounds on the derivational or runtime complexity of TRSs. We now define some notions related to such sets.

Definition 2.36. We call a (finite or infinite) set of mappings $\mathbb{N} \to \mathbb{N}$ a *class of number theoretic functions*. Let $\mathcal{C}$ be a class of number theoretic functions.

For a function $f \colon \mathbb{N} \to \mathbb{N}$, we say that f *is bounded by a function in* $\mathcal{C}$ (denoted as $f \leqslant \mathcal{C}$) if there exists a function $g \in \mathcal{C}$ such that for all $n \in \mathbb{N}$, we have $f(n) \leqslant g(n)$.

The class $\mathcal{C}$ is *closed under polynomial slowdown* if, for any $g \in \mathcal{C}$ and any polynomial P over $\mathbb{N}$, we have $f \leqslant \mathcal{C}$ for $f(n) = P(g(n))$. We define the set of functions $\Xi(\mathcal{C})$ as $\bigcup_{g \in \mathcal{C}, a \in \mathbb{N}, b \in \mathbb{N}} g(a \cdot n + b)$.

Note that in particular, the set of all polynomials over $\mathbb{N}$ is closed under polynomial slowdown. Moreover, in that case we have $\mathcal{C} = \Xi(\mathcal{C})$, $\Xi(\mathcal{C})$ is closed under polynomial slowdown, and $f \leqslant \Xi(\mathcal{C})$ if and only if f is bounded by a polynomial. On the other hand, if $\mathcal{C} = \bigcup_{a \in \mathbb{N}} \{g_a\}$ with $g_a(x) = a^x$, then $\Xi(\mathcal{C})$ is again closed under polynomial slowdown, and $f \leqslant \Xi(\mathcal{C})$ if and only if f is bounded by an exponential function; in order to see that $\Xi(\mathcal{C})$ is closed under polynomial slowdown, let $g_c \in \mathcal{C}$, and let P be a polynomial of degree d with coefficients at most a; then $P(g_c(n)) \leqslant g_{c+2}(d \cdot n + (d+1) \cdot a)$. Finally, the sets of primitive and multiply recursive functions are both obviously closed under polynomial slowdown, as well.

With every finite class $\{f_1, \ldots, f_n\}$ of number theoretic functions, we naturally associate an infinite class $\{g_1, g_2, \ldots\}$ of number theoretic functions, where $g_i = f_i$ for all $i \leqslant n$, and $g_i = f_n$ for all $i > n$. Therefore, when considering any class of number theoretic functions, it suffices without loss of generality to consider any infinite class.

Many reduction orders establish upper bounds on the derivational complexity of the TRS under consideration. We formalise this notion as follows.

Definition 2.37. Let $\mathcal{C}$ be a class of number theoretic functions, and $>$ a reduction order. We say that $>$ *induces* complexity $\mathcal{C}$ if for every TRS $\mathcal{R}$ which is compatible with $>$, $\mathsf{dc}_{\mathcal{R}}$ is bounded by a function in $\mathcal{C}$.

An even stronger property is the *characterisation* of complexity classes by sets of reduction orders.

Definition 2.38. Let $\mathcal{O}$ be a set of reduction orders, and $\mathcal{C}$ a class of number theoretic functions. We say that $\mathcal{O}$ *characterises* the complexity class $\mathcal{C}$ if the following two properties hold:

1. Every reduction order contained in $\mathcal{O}$ induces complexity $\mathcal{C}$.
2. Every function in $\mathcal{C}$ can be encoded as a TRS which is compatible with a reduction order in $\mathcal{O}$.

Another central property of TRSs besides termination and (derivational or runtime) complexity is *confluence*, which roughly expresses determinism of the computation carried out by a TRS.

Definition 2.39. A TRS $\mathcal{R}$ is *confluent* if for all terms s, t, and u such that $u \to_{\mathcal{R}}^* s$ and $u \to_{\mathcal{R}}^* t$, there exists a term v such that $s \to_{\mathcal{R}}^* v$ and $t \to_{\mathcal{R}}^* v$.

An important criterion for asserting confluence of TRSs are *critical pairs*:

Definition 2.40. Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{V}$. A triple $(l \to r, p, l' \to r')$, where $l \to r$ and $l' \to r'$ are rules of $\mathcal{R}$, and $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l)$ is called a *critical pair* of $\mathcal{R}$ if there exist substitutions σ and τ such that $l\sigma|_p = l'\tau$, and either $p \neq \epsilon$ or the rules $l \to r$ and $l' \to r'$ are distinct. If $\mathcal{R}$ is left-linear and has no critical pairs, then it $\mathcal{R}$ is also called *orthogonal*.

At this place, we only mention one well-known criterion out of the many confluence criteria in the literature involving critical pairs: any orthogonal TRS is confluent.

2.4 Dependency Pair Framework

All termination proof techniques mentioned until here are *direct* proof techniques, i.e. they take a TRS as input and try to prove its termination in a single step. In contrast, the *dependency pair framework* [35, 36, 103], which is the most important modern termination proof technique, essentially works by combining a number of smaller termination proof arguments in a modular way. In this section, we recall the basics of the dependency pair framework.

Definition 2.41. Let $\mathcal{F}$ be a signature. Then $\mathcal{F}^\sharp = \mathcal{F} \cup \{f^\sharp \mid f \in \mathcal{F}\}$, where for each $f \in \mathcal{F}$, $f^\sharp$ is a fresh function symbol with the same arity as f. We call the symbols in $\mathcal{F}^\sharp \setminus \mathcal{F}$ *dependency pair symbols*. For a term t, $t^\sharp$ is defined to be t if t is a variable, and $f^\sharp(t_1, \dots, t_n)$ if $t = f(t_1, \dots, t_n)$.

Definition 2.42. Given a TRS $\mathcal{R}$ over $\mathcal{F}$ and $\mathcal{V}$, the *dependency pairs* of $\mathcal{R}$ are the following rewrite rules over $\mathcal{F}^\sharp$ and $\mathcal{V}$:

$$\mathsf{DP}(\mathcal{R}) = \{l^\sharp \to u^\sharp \mid l \to r \in \mathcal{R} \wedge u \trianglelefteq r \wedge u \ntriangleleft l \wedge \mathsf{rt}(u) \in \mathcal{D}\}^3$$

The central objects in termination proofs by the dependency pair framework are *DP problems*. The central idea in such proofs is to transform DP problems into equivalent sets of (generally easier) DP problems until only trivial DP problems remain.

Definition 2.43. A *DP problem*[4] is a pair $(\mathcal{P}, \mathcal{R})$ such that $\mathcal{P}$ and $\mathcal{R}$ are sets of rewrite rules over $\mathcal{F}^\sharp$ and $\mathcal{V}$. A DP problem is *finite* if there exists no infinite sequence of rules $s_0 \to t_0, s_1 \to t_1, \dots$ from $\mathcal{P}$ such that for all $i \in \mathbb{N}$, there exist substitutions σ and τ such that both $s_i\tau$ and $t_i\sigma$ are terminating with respect to $\mathcal{R}$, and $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\tau$. Such an infinite sequence of rules from $\mathcal{P}$ (in case $(\mathcal{P}, \mathcal{R})$ is not finite) is called a *minimal infinite chain* with respect to $(\mathcal{P}, \mathcal{R})$.

A *DP processor* is a mapping from DP problems to sets of DP problems. A DP processor Φ is *sound* if for every DP problem $(\mathcal{P}, \mathcal{R})$, finiteness of all problems in $\Phi((\mathcal{P}, \mathcal{R}))$ implies that the problem $(\mathcal{P}, \mathcal{R})$ is finite.

Theorem 2.44 ([2])**.** *A TRS $\mathcal{R}$ is terminating if and only if the DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is finite.*

The following is an immediate consequence of the definitions.

Lemma 2.45. *Every DP problem of the form $(\emptyset, \mathcal{R})$ is finite.*

Based on Theorem 2.44 and Lemma 2.45, termination proofs in the DP framework are formalised into *proof trees*:

[3]The observation that pairs $l^\sharp \to u^\sharp$ such that $u \triangleleft l$ need not be considered is due to Dershowitz [22]. Thus we sometimes refer to the restriction "$u \ntriangleleft l$" as *Dershowitz condition.*

[4]In this thesis, DP problems $(\mathcal{P}, \mathcal{R})$ are equivalent to DP problems of the shape $(\mathcal{P}, \emptyset, \mathcal{R}, \mathbf{m})$ in the notation of [35, 103]. We use this simplified notation because the processors considered in this thesis do not perform any actions on the two components we drop in our notation.

Definition 2.46. Let $\mathcal{R}$ be a TRS. A *proof tree* of $\mathcal{R}$ is a tree satisfying the following: the nodes are DP problems, the root is $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$, each node is either a leaf of the shape $(\emptyset, \mathcal{R})$, or a node $(\mathcal{P}, \mathcal{R}')$ for which there exists a sound DP processor Φ such that the elements of $\Phi((\mathcal{P}, \mathcal{R}'))$ are exactly the child nodes of $(\mathcal{P}, \mathcal{R}')$, and each of the edges from $(\mathcal{P}, \mathcal{R}')$ to the elements of $\Phi((\mathcal{P}, \mathcal{R}'))$ is labelled by Φ.

It is immediate from Definition 2.46, Theorem 2.44, and Lemma 2.45, that the existence of a proof tree of a TRS $\mathcal{R}$ implies termination of $\mathcal{R}$.

The natural complexity measure accompanying DP problems is the following.

Definition 2.47. Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rules over $\mathcal{F}^\sharp$ and $\mathcal{V}$. Then the *DP complexity* of $\mathcal{P}$ and $\mathcal{R}$ is the following function over $\mathbb{N}$:

$$\mathsf{DPc}_{\mathcal{P},\mathcal{R}}(n) = \max\{\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathcal{P}} / \to_{\mathcal{R}}) \mid |t| \leqslant n\}$$

2.4.1 Reduction Pairs

We now recall some fundamental DP processors. The first processor we mention is the *reduction pair processor*. Reduction pairs form a natural extension of reduction orders to DP problems. Already in the first incarnation of the dependency pair method [2], they have been the quintessential method of showing finiteness of DP problems.

Definition 2.48. A *reduction pair* is a pair of orders $(\succcurlyeq, \succ)$ satisfying the following properties:

1. $\succcurlyeq$ is a preorder which is closed under substitutions and contexts
2. $\succ$ is a proper order which is well-founded and closed under substitutions
3. $\succcurlyeq \cdot \succ \cdot \succcurlyeq \subseteq \succ$ (we say that $\succcurlyeq$ and $\succ$ are *compatible* if this property holds)

Definition 2.49 ([2, 35])**.** Let $(\succcurlyeq, \succ)$ be a reduction pair. Then the *reduction pair processor* for $(\succcurlyeq, \succ)$ is the following DP processor:

$$\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}((\mathcal{P}, \mathcal{R})) = \begin{cases} \{(\mathcal{P}', \mathcal{R})\} & \text{if } \mathcal{P} \setminus \mathcal{P}' \text{ is the maximal subset of } \mathcal{P} \\ & \quad \text{compatible with } \succ\text{, and } \mathcal{P}' \cup \mathcal{R} \text{ is} \\ & \quad \text{compatible with } \succcurlyeq \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

We say that $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$ *completely solves* a DP problem $(\mathcal{P}, \mathcal{R})$ if the first case of its definition applies with $\mathcal{P}' = \emptyset$.

Theorem 2.50 ([2, 35])**.** *For every reduction pair $(\succcurlyeq, \succ)$, the reduction pair processor $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$ is sound.*

Definition 2.51. Let $\mathcal{C}$ be a class of number theoretic functions, and $(\succcurlyeq, \succ)$ a reduction pair. We say that $(\succcurlyeq, \succ)$ *induces* complexity $\mathcal{C}$ if for every DP problem $(\mathcal{P}, \mathcal{R})$ such that $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$ completely solves $(\mathcal{P}, \mathcal{R})$, the function $\mathsf{DPc}_{\mathcal{P},\mathcal{R}}$ is bounded by a function in $\mathcal{C}$.

It is easy to construct reduction pairs directly from reduction orders.

Theorem 2.52. *Let $>$ be a reduction order and $\geqslant$ the reflexive closure of $>$. Then $(\geqslant, >)$ is a reduction pair. Moreover, whenever $>$ induces complexity $\mathcal{C}$, then $(\geqslant, >)$ induces complexity $\mathcal{C}$, as well.*

Proof. From the assumption that $>$ is a reduction order, it follows immediately that $(\geqslant, >)$ has all defining properties of a reduction pair. Let $(\mathcal{P}, \mathcal{R})$ be a DP problem which is completely solved by $(\geqslant, >)$. Divide $\mathcal{R}$ into $\mathcal{R}_= \uplus \mathcal{R}_>$ such that $\mathcal{R}_=$ is compatible with $=$, and $\mathcal{R}_>$ is compatible with $>$. Then, since $\mathcal{R}_=$ is compatible with $=$, for every derivation $A : s \to^*_{\mathcal{P}\cup\mathcal{R}} t$, there exists another derivation $B : s \to^*_{\mathcal{P}\cup\mathcal{R}_>} t$ such that the length of B is greater than or equal to the number of $\mathcal{P}$-steps in A. It follows that $\mathsf{DPc}_{\mathcal{P},\mathcal{R}}(n) \leqslant \mathsf{dc}_{\mathcal{P}\cup\mathcal{R}_>}(n)$ for all $n \in \mathbb{N}$. Moreover, $\mathcal{P} \cup \mathcal{R}_>$ is compatible with $>$, and thus the theorem follows. □

We can extend the second part of Theorem 2.52 to DP problems which are not completely solved by the reduction pair processor under consideration:

Theorem 2.53. *Let $>$ be a reduction order which induces complexity $\mathcal{C}$, and $(\mathcal{P}, \mathcal{R})$ a DP problem with $\{(\mathcal{P}', \mathcal{R})\} = \Phi^{\mathsf{RP}}_{(\geqslant,>)}((\mathcal{P}, \mathcal{R}))$. Then there exists a function in $\mathcal{C}$ which bounds $\mathsf{DPc}_{\mathcal{P}\setminus\mathcal{P}',\mathcal{P}'\cup\mathcal{R}}$ from above.*

Proof. Since $\{(\mathcal{P}', \mathcal{R})\} = \Phi^{\mathsf{RP}}_{(\geqslant,>)}((\mathcal{P}, \mathcal{R}))$, the DP processor $\Phi^{\mathsf{RP}}_{(\geqslant,>)}((\mathcal{P}, \mathcal{R}))$ completely solves $(\mathcal{P} \setminus \mathcal{P}', \mathcal{P}' \cup \mathcal{R})$. Hence, by Theorem 2.52, there exists a function in $\mathcal{C}$ which bounds $\mathsf{DPc}_{\mathcal{P}\setminus\mathcal{P}',\mathcal{P}'\cup\mathcal{R}}$ from above. □

However, the second part of Theorem 2.52 can probably not be generalised by replacing $\geqslant$ by an arbitrary preorder $\succcurlyeq$ such that $(\succcurlyeq, >)$ is reduction pair:

Example 2.54. Consider the sets of rewrite rules $\mathcal{S}_1 = \{\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(x)\}$ and $\mathcal{S}_2 = \{\mathsf{d}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{d}(x)))\}$. Then it is easy to verify that $\to^+_{\mathcal{S}_1/\mathcal{S}_2}$ is a reduction order, and $(\to^*_{\mathcal{S}_2}, \to^+_{\mathcal{S}_1/\mathcal{S}_2})$ is a reduction pair. Using the DP problem $(\mathcal{S}_1, \mathcal{S}_2)$ and the set of starting terms $\{\mathsf{f}(\mathsf{d}^n(\mathsf{s}(x))) \mid n \in \mathbb{N}\}$, one can see that $(\to^*_{\mathcal{S}_2}, \to^+_{\mathcal{S}_1/\mathcal{S}_2})$ does not induce linear (and not even polynomial) complexity. However, we conjecture that the reduction order $\to^+_{\mathcal{S}_1/\mathcal{S}_2}$ does induce linear complexity, i.e. there exists no terminating TRS $\mathcal{R}$ which is compatible with $\to^+_{\mathcal{S}_1/\mathcal{S}_2}$ and whose derivational complexity grows faster than linear.

Note that the trivially constructed reduction pair of Theorem 2.52 does not make any use of the fact that the strict part of a reduction pair is not required to be closed under contexts. One way to repair this problem is to use *argument filterings.*

Definition 2.55. An *argument filtering* for a signature $\mathcal{F}$ is a mapping π with domain $\mathcal{F}$ such that for every function symbol $f \in \mathcal{F}$ of arity n, we have either $\pi(f) = i$ for some $i \in \{1, \ldots, n\}$ or $\pi(f) = [i_1, \ldots, i_m]$ with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. Based on π, a signature $\mathcal{F}_\pi$ is defined, which contains each function symbol

f such that $\pi(f)$ has the shape $[i_1, \dots, i_m]$, and the arity of f in $\mathcal{F}_\pi$ is m. An argument filtering π is lifted to a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$ as follows:

$$\pi(t) = \begin{cases} t & \text{if } t \in \mathcal{V} \\ \pi(t_i) & \text{if } t = f(t_1, \dots, t_n) \text{ and } \pi(f) = i \\ f(\pi(t_{i_1}), \dots, \pi(t_{i_m})) & \text{if } t = f(t_1, \dots, t_n) \text{ and } \pi(f) = [i_1, \dots, i_m] \;. \end{cases}$$

The mapping π is further lifted to TRSs in the natural way:

$$\pi(\mathcal{R}) = \{\pi(s) \to \pi(t) \mid s \to t \in \mathcal{R}\}$$

For each binary relation $\succ$, an argument filtering π gives rise to the following relation $\succ^\pi$:

$$s \succ^\pi t \iff \pi(s) \succ \pi(t)$$

Theorem 2.56. *Let $(\succcurlyeq, \succ)$ be a reduction pair and π an argument filtering. Then $(\succcurlyeq^\pi, \succ^\pi)$ is a reduction pair. Moreover, whenever $(\succcurlyeq, \succ)$ induces complexity $\mathcal{C}$, then $(\succcurlyeq^\pi, \succ^\pi)$ induces complexity $\mathcal{C}$, as well.*

Proof. The first part of the theorem follows directly from [2, Theorem 11]. For the second part, let $(\mathcal{P}, \mathcal{R})$ be a DP problem which is completely solved by $\Phi^{\mathsf{RP}}_{(\succcurlyeq^\pi, \succ^\pi)}$. By definition, $s \succcurlyeq^\pi t \iff \pi(s) \succcurlyeq \pi(t)$, and $s \succ^\pi t \iff \pi(s) \succ \pi(t)$, therefore the DP problem $(\pi(\mathcal{P}), \pi(\mathcal{R}))$ is completely solved by $\Phi^{\mathsf{RP}}_{(\succcurlyeq, \succ)}$. Observe that $s \to_{\mathcal{R}} t$ implies $\pi(s) \to^{=}_{\pi(\mathcal{R})} \pi(t)$, and $s \xrightarrow{\epsilon}_{\mathcal{P}} t$ implies $\pi(s) \xrightarrow{\epsilon}_{\pi(\mathcal{P})} \pi(t)$. Moreover, $|\pi(t)| \leqslant |t|$ for any term t. Thus, $\mathsf{DPc}_{\mathcal{P},\mathcal{R}}(n) \leqslant \mathsf{DPc}_{\pi(\mathcal{P}),\pi(\mathcal{R})}(n)$ for all $n \in \mathbb{N}$, and the second part of the theorem follows. $\square$

2.4.2 Usable Rules

One way to facilitate the search for reduction pairs by reducing the size of the second component of a DP problem is to only consider the *usable rules* of that problem [36, 43] (also compare [103], where they are called *needed rules* instead, or [63], for alternative definitions). The basic idea here is that in a DP problem $(\mathcal{P}, \mathcal{R})$, only those rules in a certain subset of $\mathcal{R}$ are really relevant for minimal infinite chains. This subset contains those rules which can be “triggered” by $\mathcal{P}$-rewrite steps. It should be noted that none of the above mentioned references fully reflects this idea in its definition of usable rules. All of these definitions are rather safe approximations of this idea; in the following, we give a more general definition.

Definition 2.57. Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules. Then a term t is called a *usable term of $\mathcal{P}$ with respect to $\mathcal{R}$* if there exist a rule $l \to r \in \mathcal{P}$, a position $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, substitutions σ and τ, a derivation $A : r\sigma \to^*_{\mathcal{R}} u$, and a position $q \in p \mathbin{\backslash\!\!\backslash}^{\mathrm{s}} A$ such that $u|_q = t\tau$. A term t is also called a usable term of $\mathcal{P}$ with respect to $\mathcal{R}$ if alternatively, there exist no rewrite rule $l \to r \in \mathcal{R}$, and no substitutions σ and τ such that $t\sigma \to^*_{\mathcal{R}} l\tau$. We write $\mathsf{UT}_{\mathcal{R}}(\mathcal{P})$ to denote the set of usable terms of $\mathcal{P}$ with respect to $\mathcal{R}$.

The *usable rules of $\mathcal{P}$ with respect to $\mathcal{R}$* (denoted as $\mathsf{UR}_{\mathcal{R}}(\mathcal{P})$) are all rules $l \to r$ from $\mathcal{R}$ such that $l \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$.

A set $\mathcal{U}$ such that $\mathsf{UR}_{\mathcal{R}}(\mathcal{P}) \subseteq \mathcal{U} \subseteq \mathcal{R}$ is a *sound approximation of* $\mathsf{UR}_{\mathcal{R}}(\mathcal{P})$.

Note that the inclusion of the second alternative in the definition of $\mathsf{UT}_{\mathcal{R}}(\mathcal{P})$ does not add anything to $\mathsf{UR}_{\mathcal{R}}(\mathcal{P})$, so it could be omitted without loss. Still, we included it in order to ease some technical details in Section 6.9, where we analyse the derivational complexity of TRSs whose termination can be proved with the help of the usable rules processor defined below in Definition 2.58.

In order to see that this definition differs from the definition of needed rules given in [103], which is the most general definition in the above mentioned references (to the best of our knowledge, no more general definition existed up to now), consider the following DP problem $(\mathcal{P}, \mathcal{R})$:

$$\mathcal{P} = \{\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{f}(\mathsf{g}(x)))\} \qquad \mathcal{R} = \{\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{f}(\mathsf{g}(x))), \mathsf{g}(\mathsf{0}) \to \mathsf{1}\}$$

Then $\mathsf{UR}_{\mathcal{R}}(\mathcal{P}) = \{\mathsf{g}(\mathsf{0}) \to \mathsf{1}\}$. On the other hand, in the notation of and according to [103, Definition 4.5], we have $\mathcal{N}(\mathcal{P}, \emptyset, \mathcal{R}) = \mathcal{R}$.

Definition 2.58 ([36, 43, 103])**.** Let $\mathcal{U}$ be a set of rewrite rules and $(\succsim, \succ)$ a reduction pair. Furthermore, let $\mathcal{C}_\epsilon$ be the set containing the two rewrite rules $\mathsf{c}(x, y) \to x$ and $\mathsf{c}(x, y) \to y$ for some fresh function symbol c of arity 2. Then the *usable rules processor* $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succsim,\succ)}$ is the following DP processor:

$$\Phi^{\mathsf{UR}}_{\mathcal{U},(\succsim,\succ)}((\mathcal{P}, \mathcal{R})) = \begin{cases} \{(\mathcal{P}', \mathcal{R})\} & \text{if } \mathcal{P} \setminus \mathcal{P}' \text{ is the maximal subset of } \mathcal{P} \\ & \quad \text{compatible with } \succ\text{, } \mathcal{P}' \cup \mathcal{U} \cup \mathcal{C}_\epsilon \text{ is} \\ & \quad \text{compatible with } \succsim\text{, and } \mathcal{U} \text{ is a sound} \\ & \quad \text{approximation of } \mathsf{UR}_{\mathcal{R}}(\mathcal{P}) \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

We say that $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succsim,\succ)}$ *completely solves* a DP problem $(\mathcal{P}, \mathcal{R})$ if the first case of its definition applies with $\mathcal{P}' = \emptyset$.

We now give the soundness proof of the usable rules processor in our new definition. We follow the soundness proof given in [43], but give more attention to the parts of the proof which had to be modified for our new definition.

We start by defining an interpretation on terms, which is an adaptation of the mapping $\mathcal{I}_{\mathcal{G}}$ from [43, Definition 15] to our new definition of usable rules.

Definition 2.59. Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules. The *interpretation* $\mathcal{I}_{\mathcal{P},\mathcal{R}}$ is a mapping from terminating terms in $\mathcal{T}(\mathcal{F}^\sharp, \mathcal{V})$ to terms in $\mathcal{T}(\mathcal{F}^\sharp \uplus \{\mathsf{nil}, \mathsf{c}\}, \mathcal{V})$, where nil is a fresh function symbol and c is the function symbol introduced by $\mathcal{C}_\epsilon$, inductively defined as follows:

$$\mathcal{I}_{\mathcal{P},\mathcal{R}}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ f(\mathcal{I}_{\mathcal{P},\mathcal{R}}(t_1), \ldots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \\ & \text{and } t \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P}) \\ \mathsf{c}(f(\mathcal{I}_{\mathcal{P},\mathcal{R}}(t_1), \ldots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(t_n)), t') & \text{if } t = f(t_1, \ldots, t_n) \\ & \text{and } t \notin \mathsf{UT}_{\mathcal{R}}(\mathcal{P}) \end{cases}$$

where in the last clause t' denotes the term $\mathit{order}(\{\mathcal{I}_{\mathcal{P},\mathcal{R}}(u) \mid t \to_{\mathcal{R}} u\})$ with

$$\mathit{order}(T) = \begin{cases} \mathsf{nil} & \text{if } T = \emptyset \\ \mathsf{c}(t, \mathit{order}(T \setminus \{t\})) & \text{if } t \text{ is the minimum element of } T \end{cases}$$

Here an arbitrary but fixed total order on $\mathcal{T}(\mathcal{F}^\sharp \cup \{\mathsf{nil}, \mathsf{c}\}, \mathcal{V})$ is assumed.

The difference between Definition 2.59 and [43, Definition 15] is the side condition in the last two cases of the definition of $\mathcal{I}_{\mathcal{P},\mathcal{R}}$.

The next definition is essentially [43, Definition 16].

Definition 2.60. Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules, and let σ be a substitution such that $\sigma(x)$ is terminating with respect to $\mathcal{R}$ for all $x \in \mathcal{V}$. Then $\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}$ is the substitution defined by $\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}(x) = \mathcal{I}_{\mathcal{P},\mathcal{R}}(\sigma(x))$.

Lemma 2.61. *Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules, t a term, and σ a substitution. If $t\sigma$ is terminating with respect to $\mathcal{R}$, then $\mathcal{I}_{\mathcal{P},\mathcal{R}}(t\sigma) \to^*_{\mathcal{C}_\epsilon} t\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}$, and, if all non-variable subterms of t are contained in $\mathsf{UT}_{\mathcal{R}}(\mathcal{P})$, then $\mathcal{I}_{\mathcal{P},\mathcal{R}}(t\sigma) = t\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}$.*

Proof. Analogous to [43, Lemma 17]. □

Lemma 2.62. *Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules, $s \to t \in \mathsf{UR}_{\mathcal{R}}(\mathcal{P})$, and $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(t)$. Then $t|_p \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$.*

Proof. Since $s \to t \in \mathsf{UR}_{\mathcal{R}}(\mathcal{P})$, we have $s \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$, and trivially $s \to^*_{\mathcal{R}} s$. Therefore, by definition, there exist a rule $l \to r \in \mathcal{P}$, a position $q \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, substitutions σ and τ, a derivation $A : r\sigma \to^*_{\mathcal{R}} u$, and a position $q' \in q \mathbin{\backslash\!\backslash^{\mathrm{s}}} A$ such that $u|_{q'} = s\tau$. So, u can be written as $C[s\tau]$ for some context C. By assumption, we have the rewrite step $A_1 : C[s\tau] \to_{\mathcal{R}} C[t\tau]$, which can be appended to A, resulting in $B : r\sigma \to^*_{\mathcal{R}} C[s\tau] \to_{\mathcal{R}} C[t\tau]$. Moreover, we have $q'p \in q' \mathbin{\backslash\!\backslash^{\mathrm{s}}} A_1$ and hence $q'p \in q \mathbin{\backslash\!\backslash^{\mathrm{s}}} B$, and we have $C[t\tau]|_{q'p} = t\tau|_p$. Thus, the lemma follows. □

The next lemma is a new auxiliary lemma in order to help with the new proof of Lemma 2.64 below.

Lemma 2.63. *Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules, and suppose $A : s \to_{\mathcal{R}} t$, $s \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$. Then $t|_p \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$, for every position $p \in \epsilon \mathbin{\backslash\!\backslash^{\mathrm{s}}} A$.*

Proof. If there exist no rule $l \to r \in \mathcal{R}$ and no substitutions σ and τ such that $s\sigma \to^*_{\mathcal{R}} l\tau$, then due to $s \to_{\mathcal{R}} t$, there exist no rule $l \to r \in \mathcal{R}$ and no substitutions σ and τ such that $t\sigma \to^*_{\mathcal{R}} l\tau$, either. Hence $t \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$ in this case.

In the other case, there exist a rule $l \to r \in \mathcal{P}$, a position $q \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, substitutions σ and τ, a derivation $A_1 : r\sigma \to^*_{\mathcal{R}} u$, and a position $q' \in q \mathbin{\backslash\!\backslash^{\mathrm{s}}} A_1$ such that $u|_{q'} = s\tau$. So, u can be written as $C[s\tau]$ for some context C. We can concatenate A_1 and A to $B : r\sigma \to^*_{\mathcal{R}} C[s\tau] \to_{\mathcal{R}} C[t\tau]$. Moreover, by assumption $p \in \epsilon \mathbin{\backslash\!\backslash^{\mathrm{s}}} A$, hence $q'p \in q \mathbin{\backslash\!\backslash^{\mathrm{s}}} B$, and thus the lemma follows. □

The next lemma is equivalent to [43, Lemma 19]. Due to the fact that, in contrast to [43], the side conditions in the definition of $\mathcal{I}_{\mathcal{P},\mathcal{R}}(t)$ inspect not only the root symbol of t (this property is used in one part of the proof of [43, Lemma 19]), a part of the proof of this lemma had to be modified.

Lemma 2.64. *Let $\mathcal{P}$ and $\mathcal{R}$ be sets of rewrite rules, and suppose that $A : s \xrightarrow{p}_{\mathcal{R}} t$ and that s and t are terminating with respect to $\mathcal{R}$. Then $\mathcal{I}_{\mathcal{P},\mathcal{R}}(s) \to^+_{\mathsf{UR}_{\mathcal{R}}(\mathcal{P}) \cup \mathcal{C}_\epsilon} \mathcal{I}_{\mathcal{P},\mathcal{R}}(t)$.*

Proof. We distinguish two cases for p:

- Suppose that there exists a position $q \leqslant p$ such that $s|_q \notin \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$. Choose q to be such that $s|_q \notin \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$, but for all positions $q' < q$, we do have $s|_{q'} \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$. Then $\mathcal{I}_{\mathcal{P},\mathcal{R}}(s|_q) \to_{\mathcal{C}_\epsilon} order(\{\mathcal{I}_{\mathcal{P},\mathcal{R}}(u) \mid s|_q \to_{\mathcal{R}} u\}) \to^+_{\mathcal{C}_\epsilon} \mathcal{I}_{\mathcal{P},\mathcal{R}}(t|_q)$. By Lemma 2.63, we also have $t|_{q'} \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$ for all positions $q' < q$. Hence, for each $q' < q$, if $s|_{q'} = f(s_1, \dots, s_i, \dots, s_n)$ and $t|_{q'} = f(s_1, \dots, t_i, \dots, s_n)$, then by definition of $\mathcal{I}_{\mathcal{P},\mathcal{R}}$,

$$\begin{aligned}\mathcal{I}_{\mathcal{P},\mathcal{R}}(s|_{q'}) &= f(\mathcal{I}_{\mathcal{P},\mathcal{R}}(s_1), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(s_i), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(s_n))\\ \mathcal{I}_{\mathcal{P},\mathcal{R}}(t|_{q'}) &= f(\mathcal{I}_{\mathcal{P},\mathcal{R}}(s_1), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(t_i), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(s_n))\ .\end{aligned}$$

  By a simple inductive argument, we get $\mathcal{I}_{\mathcal{P},\mathcal{R}}(s) \to^+_{\mathcal{C}_\epsilon} \mathcal{I}_{\mathcal{P},\mathcal{R}}(t)$, so the lemma follows.

- Suppose that for all positions $q \leqslant p$, we have $s|_q \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$. The rule $l \to r$ applied in the step A must be contained in $\mathsf{UR}_{\mathcal{R}}(\mathcal{P})$, since $s|_p = l\sigma$ for some substitution σ. By Lemma 2.61, $\mathcal{I}_{\mathcal{P},\mathcal{R}}(s|_p) \to^*_{\mathcal{C}_\epsilon} l\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}$. Moreover, $t|_p = r\sigma$, and by Lemma 2.63, for all non-variable subterms r' of r, we have $r'\sigma \in \mathsf{UT}_{\mathcal{R}}(\mathcal{P})$. Hence by Lemma 2.61, $\mathcal{I}_{\mathcal{P},\mathcal{R}}(r\sigma) = r\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}$. Clearly, $l\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}} \to_{\mathsf{UR}_{\mathcal{R}}(\mathcal{P})} r\sigma_{\mathcal{I}_{\mathcal{P},\mathcal{R}}}$, and hence $\mathcal{I}_{\mathcal{P},\mathcal{R}}(s|_p) \to^+_{\mathsf{UR}_{\mathcal{R}}(\mathcal{P}) \cup \mathcal{C}_\epsilon} \mathcal{I}_{\mathcal{P},\mathcal{R}}(t|_p)$. By assumption, for all $q < p$, if $s|_q = f(s_1, \dots, s_i, \dots, s_n)$ and $t|_q = f(s_1, \dots, t_i, \dots, s_n)$, then

$$\begin{aligned}\mathcal{I}_{\mathcal{P},\mathcal{R}}(s|_q) &= f(\mathcal{I}_{\mathcal{P},\mathcal{R}}(s_1), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(s_i), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(s_n))\\ \mathcal{I}_{\mathcal{P},\mathcal{R}}(t|_q) &= f(\mathcal{I}_{\mathcal{P},\mathcal{R}}(s_1), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(t_i), \dots, \mathcal{I}_{\mathcal{P},\mathcal{R}}(s_n))\ .\end{aligned}$$

  The lemma now follows by a simple inductive argument.

□

Having transferred all needed auxiliary lemmata from [43] to our new definition, soundness of our new definition is concluded easily.

Theorem 2.65. *For every set of rules $\mathcal{U}$ and reduction pair $(\succcurlyeq, \succ)$, the usable rules processor $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ is sound.*

Proof. Analogous to [43, Theorem 20]. □

2.4.3 Dependency Graphs

We now recall the *dependency graph* processor. Just as reduction pairs, dependency graphs were already presented in [2]. Their purpose is to split a given DP problem into as many smaller parts as possible, making them one of the first ways to modularise termination proofs.

Definition 2.66. The *dependency graph* of a DP problem $(\mathcal{P}, \mathcal{R})$ (denoted by $\mathsf{DG}(\mathcal{P}, \mathcal{R})$) is a directed graph whose nodes are the elements of $\mathcal{P}$. It contains an edge from node $s \to t$ to the node $u \to v$ whenever there exist substitutions σ and τ such that $t\sigma \to^*_{\mathcal{R}} u\tau$. A directed graph $\mathcal{G}$ is *a sound approximation* of a dependency graph $\mathsf{DG}(\mathcal{P}, \mathcal{R})$ if the nodes of $\mathcal{G}$ are the elements of $\mathcal{P}$, and whenever there exists an edge from node $s \to t$ to the node $u \to v$ in $\mathsf{DG}(\mathcal{P}, \mathcal{R})$, there is also an edge from $s \to t$ to $u \to v$ in $\mathcal{G}$.

A *strongly connected component*[5] (SCC for short) of a directed graph $\mathcal{G}$ is a maximal subset of the nodes of $\mathcal{G}$ such that for each pair of nodes n_1, n_2 in $\mathcal{G}$, there exists a (possibly empty) path from n_1 to n_2. We call a SCC *trivial* if it consists of a single node n such that the only path from n to itself is the empty path. All other SCCs are called *nontrivial*.

Definition 2.67 ([2, 35])**.** Let $\mathcal{G}$ be a graph. Then the *dependency graph processor* for $\mathcal{G}$ is the following DP processor:

$$\Phi^{\mathsf{DG}}_{\mathcal{G}}((\mathcal{P}, \mathcal{R})) = \begin{cases} \{(\mathcal{P}', \mathcal{R}) \mid \mathcal{P}' \text{ is a nontrivial SCC of } \mathcal{G}\} & \text{if } \mathcal{G} \text{ is a sound approximation of } \mathsf{DG}(\mathcal{P}, \mathcal{R}) \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

Theorem 2.68 ([2, 35])**.** *For every directed graph $\mathcal{G}$, the dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$ is sound.*

2.4.4 Subterm Criterion

We now recall the *subterm criterion processor* [43]. It is very simple, but still powerful (due to only having to consider the first component of a given DP problem), and hence well-suited for automated termination provers.

Definition 2.69 ([43])**.** A *simple projection* π is an argument filtering such that for each dependency pair symbol $f^\sharp$, we have $\pi(f^\sharp) = i$ for some $1 \leqslant i \leqslant n$, where n is the arity of f, and for each other function symbol g, we have $\pi(g) = [1, \ldots, m]$, where m is the arity of g.

Since the filtering $\pi(f)$ for every non-dependency pair function symbol f is fixed, we only mention $\pi(f^\sharp)$ for dependency pair symbols $f^\sharp$ when defining simple projections.

Definition 2.70 ([43, 103])**.** Let π be a simple projection. Then the *subterm criterion processor* for π is defined as follows:

$$\Phi^{\mathsf{SC}}_{\pi}((\mathcal{P}, \mathcal{R})) = \begin{cases} \{(\mathcal{P}', \mathcal{R})\} & \text{if } \mathcal{P} \setminus \mathcal{P}' \text{ is the maximal subset of } \mathcal{P} \text{ compatible with } \rhd \text{, and } \mathcal{P}' \text{ is compatible with } \trianglerighteq \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

We say that Φ^{SC}_{π} *completely solves* a DP problem $(\mathcal{P}, \mathcal{R})$ if the first case of its definition applies with $\mathcal{P}' = \emptyset$.

Theorem 2.71 ([43, 103])**.** *For every simple projection π, the subterm criterion processor Φ^{SC}_{π} is sound.*

[5]Note that this is the standard definition of SCC from graph theory (cf. [18], for instance), which slightly differs from the definition that is often used in the literature about termination of term rewriting.

Chapter 3

The Exact Hardness of Deciding Derivational Complexity

Some problems are so complex that you have to be highly intelligent and well informed just to be undecided about them.

Laurence Johnston Peter

3.1 Introduction

Before delving into the main theme of this thesis, which is to investigate the derivational complexity of TRSs whose termination can be proved by certain techniques, we analyse the hardness of the decision problem we are investigating. The problem of deciding, given a TRS $\mathcal{R}$, whether its derivational complexity is bounded by a fixed family of functions, is (unsurprisingly) undecidable. In this chapter, we confirm the exact degree of undecidability of this decision problem, and a number of related decision problems. We follow recent investigations into the exact level of undecidability (in the arithmetical hierarchy) of questions in rewriting [92, 26], according to which many of the standard properties of rewriting (termination, normalisation, confluence) are known to be Π^0_2-complete. We also follow a much older set of investigations into the exact level of undecidability of intensional properties of programs [37, 91, 72, 3]. Note that compared to other models of computation such as Turing Machines, term rewriting operates in a quite nonstandard way, and hence it is a priori not clear that the classic results can be transferred to term rewriting. For instance, for nonlinear rewriting rules such as $\mathsf{f}(x, x, y) \to \mathsf{g}(x, y, y)$, it is assumed that both the equality check implicit in determining whether the rule can be applied (e.g. the first two arguments of f must be identical terms), and the copying of arbitrarily large terms (e.g. the term substituted for y can be large) can be done within a single computation step. Even more pertinent, the set of allowed "starting configurations" in derivational complexity analysis (and even in runtime complexity analysis) is defined much more liberally than in other models of computation. For Turing machines, only (a certain subset of all) well-formed configurations are considered, and in pure functional programs, the arguments of a function are always well-formed elements of a data type, e.g. $\mathsf{f}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$ – "f applied to 2". In contrast, derivational complexity must also consider "junk" terms that do not correspond to well-formed starting configurations such as `f(s(f(f(s(0)))))`, for instance, as starting terms. We verify that despite these obstacles, the classical results hold for TRSs.

The results presented in this chapter show that the decision problems for the complexity measures for TRSs are either Π^0_1-complete (for a specific function as an upper bound) or Σ^0_2-complete (for existence of an upper bound in a family of functions satisfying some mild conditions). This is in line with classical results on the degrees of undecidability of intensional complexity of programs.

All results easily carry over to a different flavour of complexity from formal language theory, confusingly also called derivational complexity, which is starkly different from the homonymous notion in term rewriting. In formal language theory, the derivational complexity relates an integer n to the maximum length of *shortest* derivations of sentences of length at most n [11, 99]. We name the corresponding complexity measures for term rewriting *minimal complexity* in the sequel.

For implicit complexity of TRSs, where the computational complexity of the mathematical function *computed* by a given TRS $\mathcal{R}$ is considered, the pertinent decision problems are even harder: Deciding whether the implicit complexity of $\mathcal{R}$ is, for instance, polynomial or exponential, is even harder than the previously mentioned problems: Σ^0_3-complete. Again, this is in line with the classical results [37, 91]. However, in practice, the additional existential quantifier (quantifying over the possibly more efficient TRS) might make it easier to establish *upper* bounds on the implicit complexity of a TRS than to establish upper bounds on its derivational or runtime complexity, as there is an additional degree of freedom in constructing the proof of the upper bound.

Finally, when using TRSs to reason about functional programs, the notion of *strategy* is usually employed to make evaluation deterministic and express for instance call-by-value and call-by-name. We show that all of our results remain valid for any computable strategy.

3.2 Runtime Complexity

We now consider the hardness of establishing upper bounds on the runtime complexity of TRSs by classifying the corresponding decision problems in the arithmetical hierarchy. We start by giving an encoding of Turing Machines as TRSs which essentially uses the encoding of [102, Section 5.3.1] lifted to dual-tape Machines. Moreover, we performed some changes in order to align the TRS with the semantics of basic terms and runtime complexity. Using this encoding, it is then possible to use existing results about the time complexity of Turing Machines [37] directly. Note that it is necessary to use dual-tape machines in order to obtain the following results exactly as formulated. If we used single-tape machines instead, there would be an additional quadratic slowdown in the proofs for both of the results from [37] we use. Allowing Turing Machines with any number of tapes instead of dual-tape machines would yield an additional speedup in the order of $n \log n$ according to [40]; however, this additional speedup is not needed for obtaining the results of this section.

We start by giving the above mentioned encoding of Turing Machines as TRSs, based on the encoding given in [102, Section 5.3.1]. Given a Turing Machine $M = (Q, \Sigma, \delta)$, the encoding produces a TRS $\Delta(M)$. For each symbol $b \in \Sigma$, the signature $\mathcal{F}$ of $\Delta(M)$ contains a constructor symbol b of arity 1. Moreover, $\mathcal{F}$ contains the nullary constructor symbol $\rhd$. For each state $q \in Q$, the defined function symbols in $\mathcal{F}$ contain q as a function symbol of arity 5. Additionally, $\mathcal{F}$ contains the unary defined function symbols ok and runM. As in [102, Section 5.3.1], words over Σ are translated to terms by a mapping ϕ, which is defined as follows:

$$\phi(w) = \begin{cases} \rhd & \text{if } w = \epsilon \\ b(\phi(w')) & \text{if } w = bw' \end{cases}$$

A configuration (q, w, i, v, j) such that $w = w_1 \dots w_n$ and $v = v_1 \dots v_m$ is encoded as the term

$$q(\mathsf{ok}(\rhd), \phi(w_{i-1} \dots w_1), \phi(w_i \dots w_n), \phi(v_{j-1} \dots v_1), \phi(v_j \dots v_m)) \ .$$

This way, it is easy to simulate Turing Machine computation steps by rewrite steps.

Notation 3.1. For ease of notation, we often identify w and $\phi(w)$ for $w \in \Sigma^*$ during the rest of this chapter.

Definition 3.2. Let $M = (Q, \Sigma, \delta)$ be a dual-tape Turing Machine. Then the orthogonal TRS $\Delta(M)$ is defined by the rules shown in Figure 3.1. Here we use $\overline{\mathsf{ok}}$ to abbreviate $\mathsf{ok}(\rhd)$

transition function	rewrite rule ($q \in Q \setminus \{q_a, q_r\}$ and $a, b, c, d \in \Sigma$)
$\delta(q, b, d) = (q', b', R, d', R)$	$q(\overline{\mathsf{ok}}, x, by, z, dw) \to q'(\overline{\mathsf{ok}}, b'x, y, d'z, w)$
$\delta(q, b, d) = (q', b', R, d', L)$	$q(\overline{\mathsf{ok}}, x, by, cz, dw) \to q'(\overline{\mathsf{ok}}, b'x, y, z, cd'w)$
$\delta(q, b, \Box) = (q', b', R, d', R)$	$q(\overline{\mathsf{ok}}, x, by, z, \rhd) \to q'(\overline{\mathsf{ok}}, b'x, y, d'z, \rhd)$
$\delta(q, b, d) = (q', b', L, d', R)$	$q(\overline{\mathsf{ok}}, ax, by, z, dw) \to q'(\overline{\mathsf{ok}}, x, ab'y, d'z, w)$
$\delta(q, b, d) = (q', b', L, d', L)$	$q(\overline{\mathsf{ok}}, ax, by, cz, dw) \to q'(\overline{\mathsf{ok}}, x, ab'y, z, cd'w)$
$\delta(q, b, \Box) = (q', b', L, d', R)$	$q(\overline{\mathsf{ok}}, ax, by, z, \rhd) \to q'(\overline{\mathsf{ok}}, x, ab'y, d'z, \rhd)$
$\delta(q, \Box, d) = (q', b', R, d', R)$	$q(\overline{\mathsf{ok}}, x, \rhd, z, dw) \to q'(\overline{\mathsf{ok}}, b'x, \rhd, d'z, w)$
$\delta(q, \Box, d) = (q', b', R, d', L)$	$q(\overline{\mathsf{ok}}, x, \rhd, cz, dw) \to q'(\overline{\mathsf{ok}}, b'x, \rhd, z, cd'w)$
$\delta(q, \Box, \Box) = (q', b', R, d', R)$	$q(\overline{\mathsf{ok}}, x, \rhd, z, \rhd) \to q'(\overline{\mathsf{ok}}, b'x, \rhd, d'z, \rhd)$
	additional rules
	$\mathsf{runM}(x) \to q_s(\overline{\mathsf{ok}}, \rhd, \vdash(x), \rhd, \vdash(\rhd))$
	$q_a(\overline{\mathsf{ok}}, x, y, z, w) \to q_a(\rhd, x, y, z, w)$
	$q_r(\overline{\mathsf{ok}}, x, y, z, w) \to q_r(\rhd, x, y, z, w)$
	$\mathsf{ok}(\overline{\mathsf{ok}}) \to \rhd$

Figure 3.1: The TRS $\Delta(M)$ defined by a dual-tape Turing Machine M

The purpose of demanding the subterm $\mathsf{ok}(\triangleright)$ to be present in the encoding of a configuration is to ensure that configurations (in particular, unreachable configurations) can not be encoded by basic terms. Therefore, in contrast to the construction in [102, Section 5.3.1], $\Delta(M)$ is not a constructor TRS. Apart from that, the rules for the transition function of the given Turing Machine are the natural lifting of the rules given in [102, Section 5.3.1] to our encoding of configurations. The first of the four additional rules is responsible for rewriting a basic term of the shape $\mathsf{runM}(w)$ into the encoding of a start configuration. The other three additional rules ensure that q_a, q_r, and ok are defined symbols without violating the orthogonality of the TRS. This also implies that each term in $\mathcal{T}(\mathcal{C})$ is a word over Σ.

Lemma 3.3. *Let M be a Turing Machine. Then $\mathsf{rc}_{\Delta(M)}(n) = 0$ for all $n < 2$, and $\mathsf{rc}_{\Delta(M)}(n) = \textsc{Time}_M(n-2) + 2$ for all $n \geqslant 2$.*

Proof. First, observe that the only basic terms t which are not normal forms of $\Delta(M)$ have runM as their root symbol. Hence, we assume $\mathsf{rt}(t) = \mathsf{runM}$. Moreover, the single argument of runM must be a word over Σ. Then the only one-step reduct of t is the encoding of a starting configuration of M. Moreover, clearly $|t| \geqslant 2$. A straightforward argument (compare [102, Exercise 5.3.3]) reveals the following:

- Whenever s is the encoding of a configuration α of M whose current state is not q_a or q_r, and $s \to_{\Delta(M)} s'$, then s' is the encoding of a configuration β of M such that M moves from α to β in a single step.
- Whenever s is the encoding of a configuration α of M whose current state is q_a or q_r, then $\mathsf{dh}(s, \to_{\Delta(M)}) = 1$.
- Whenever α and β are configurations of M such that M moves from α to β in a single step, then for each term encoding s of α, there exists a term encoding s' of β such that $s \to_{\Delta(M)} s'$.

From these three observations, the lemma follows immediately. □

Using Lemma 3.3, it is now possible to transfer existing results about the time complexity of Turing Machines [37] to runtime complexity of rewriting. We use the operator Ξ as specified in Definition 2.36.

Lemma 3.4. *Let $\mathcal{C} = \{g_1, g_2, \ldots\}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then there exists a computable, strictly increasing, and total function $f \colon \mathbb{N} \to \mathbb{N}$ such that $f \not\leqslant \Xi(\mathcal{C})$.*

Proof. Let $f(n) = 1 + \max\{g_1(n^2+n), \ldots, g_n(n^2+n)\}$. Then f is obviously computable, strictly increasing, and total, and for all $c, d, k \in \mathbb{N}$, we have $f(n) > g_k(c \cdot n + d)$ for all $n > \max\{c, d, k\}$. □

Theorem 3.5. *Let $\mathcal{C}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then the following decision problem is Σ_2^0-hard:*

Instance: A dual-tape Turing Machine M.

Question: Is $\textsc{Time}_M \leqslant \Xi(\mathcal{C})$?

Proof. We show this theorem by giving a straightforward generalisation of the proof of [37, Theorem 2], which handles the special case that $\mathcal{C}$ is the set of polynomials. That proof proceeds by reduction from $\textsc{Fin}$: given a single-tape Turing Machine M, another dual-tape Turing Machine $\rho(M)$ is constructed such that $\textsc{Time}_{\rho(M)} \leqslant \Xi(\mathcal{G})$ if and only if $M \in \textsc{Fin}$.

The machine $\rho(M)$ starts by checking whether the input word has the shape $x\sharp^{|x|+k}$, where x is a word over the input alphabet of M, and $\sharp$ is not a symbol of the tape alphabet of M. If the input word does not have the desired shape, then $\rho(M)$ halts immediately; otherwise, x is copied to the second tape, and M is simulated with input x for k steps. If M halts after exactly k steps, then $\rho(M)$ continues running for at least $f(2|x|+k)$ steps, and then halts. Here, f is a computable, strictly increasing, and total function such that $f \nleqslant \Xi(\mathcal{G})$ (such a function exists by Lemma 3.4). Otherwise, $\rho(M)$ halts immediately.

Note that for all inputs which are either not of the shape $x\sharp^{|x|+k}$, or such that M does not halt on x after exactly k steps, $\rho(M)$ runs in linear time. If $M \in \textsc{Fin}$, then there are only finitely many inputs such that M runs for $f(2|x|+k)$ many steps, hence $\textsc{Time}_{\rho(M)}$ is linear in that case. As $\mathcal{G}$ contains a strictly increasing function, it then follows that $\textsc{Time}_{\rho(M)} \leqslant \Xi(\mathcal{G})$. On the other hand, if $M \notin \textsc{Fin}$, then $\textsc{Time}_{\rho(M)}$ majorises f, and thus $\textsc{Time}_{\rho(M)} \nleqslant \Xi(\mathcal{G})$. $\square$

Theorem 3.6. *Let $\mathcal{C}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then the following decision problem is Σ^0_2-complete, even if the problem instances are restricted to orthogonal TRSs:*

Instance: A TRS $\mathcal{R}$.

Question: Is $\mathsf{rc}_{\mathcal{R}} \leqslant \Xi(\mathcal{C})$?

Proof. To see that the problem is contained in Σ^0_2, let $P(x_1, x_2, x_3)$ be the ternary predicate on $\mathbb{N}$ that holds exactly if the i^{th} function g_i in $\mathcal{C}$ and the TRS $\mathcal{R}$ encoded by x_3 satisfy $\mathsf{rc}_{\mathcal{R}}(x_2) \leqslant g_i(j \cdot x_2 + k)$, where (i, j, k) is the triple encoded by x_1. Observe that $P(x_1, x_2, x_3)$ is a decidable predicate: as $\mathcal{C}$ is recursively enumerable and consists of computable functions, we may compute $g_i(j \cdot x_2 + k)$; as the signature and set of rules of $\mathcal{R}$ are both finite, we may compute the finite set of basic terms of size at most x_2, and for each of these compute their derivation trees up to depth $g_i(j \cdot x_2 + k)$ and subsequently check whether the leaves of each tree consist only of normal forms, and whether all trees are non-circular. Thus, the answer to the question to be decided is "yes" for the TRS encoded by x_3 if and only if the predicate $\exists x_1 \forall x_2\ P(x_1, x_2, x_3)$ holds, proving containment in Σ^0_2.

We now show Σ^0_2-hardness of the problem. By Theorem 3.5, it is Σ^0_2-hard to decide whether $\textsc{Time}_M \leqslant \Xi(\mathcal{C})$, given a dual-tape Turing Machine M. From Lemma 3.3, it follows that $\mathsf{rc}_{\Delta(M)} \leqslant \Xi(\mathcal{C})$ if and only if $\textsc{Time}_M \leqslant \Xi(\mathcal{C})$. The transformation Δ is obviously computable, and $\Delta(M)$ is orthogonal. Therefore, it is Σ_2^0-hard to decide whether $\mathsf{rc}_{\mathcal{R}} \leqslant \Xi(\mathcal{C})$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal). $\square$

Theorem 3.7. *Let f be a computable and total function $\mathbb{N} \to \mathbb{N}$ such that $f(n) > n$ for all $n \in \mathbb{N}$. Then the following decision problem is Π_1^0-hard:*

Instance: A dual-tape Turing Machine M.

Question: Is $\textsc{Time}_M(n) \leqslant f(n)$ for all $n \in \mathbb{N}$?

Proof. Straightforward generalisation of [37, Theorem 1]. $\square$

Theorem 3.8. *Let f be a computable and total function $\mathbb{N} \to \mathbb{N}$ such that $f(n) > n$ for all $n \in \mathbb{N}$. Then the following decision problem is Π_1^0-complete, even if the problem instances are restricted to orthogonal TRSs:*

Instance: A TRS $\mathcal{R}$.

Question: Is $\mathsf{rc}_{\mathcal{R}}(n) \leqslant f(n)$ for all $n \in \mathbb{N}$?

Proof. To see that the problem is contained in Π_1^0, consider the binary predicate $P(x_1, x_2)$ on $\mathbb{N}$ that holds if and only if $\mathsf{rc}_{\mathcal{R}}(x_1) \leqslant f(x_1)$, where $\mathcal{R}$ is the TRS encoded by x_2. As f is computable and total, and as the derivation tree of each of the finite number of terms of size at most x_1 can be computed up to depth $f(x_1)$, the predicate is obviously decidable. Hence, the answer to the question to be decided is "yes" if and only if the predicate $\forall x_1\ P(x_1, x_2)$ holds, and containment in Π_1^0 is shown.

We now show Π_1^0-hardness of the problem. Let $f'(n) = f(n+2)-2$, and note that $f'(n) > n$. By Theorem 3.7, it is Π_1^0-hard to decide whether $\textsc{Time}_M(n) \leqslant f'(n)$ for all $n \in \mathbb{N}$, given a dual-tape Turing Machine M. By Lemma 3.3, we have $\mathsf{rc}_{\Delta(M)}(n) \leqslant f(n)$ for all $n \in \mathbb{N}$ if and only if $\textsc{Time}_M(n) \leqslant f'(n)$ for all $n \in \mathbb{N}$. The transformation Δ is obviously computable, and $\Delta(M)$ is orthogonal. Therefore, it is Π_1^0-hard to decide whether $\mathsf{rc}_{\mathcal{R}}(n) \leqslant f(n)$ for all $n \in \mathbb{N}$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal). $\square$

3.3 Implicit Computational Complexity Analysis for Rewriting

In this section we establish Σ_3^0-completeness of deciding implicit complexity bounds on TRSs: deciding whether the computation carried out by a TRS can be done within a certain time bound, possibly by another, more efficient TRS. In the literature, there exist similar results about Turing Machines [37, 91]. In order to be able to apply them, we need to establish a link between computations carried out by TRSs and Turing Machines. For one direction of this link, we use Lemma 3.3. The existence of the other direction of the link has recently been shown by Avanzini and Moser [6]. In the following, we define a simple notion of computation by a TRS, and glue the above components together.

Definition 3.9. Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{V}$, let f be a specific n-ary defined function symbol (we call f the *main function* of $\mathcal{R}$), and a another specific symbol in $\mathcal{F}$ (we call a the *accepting symbol* of $\mathcal{R}$). Then for $t_1, \dots, t_n \in \mathcal{T}(\mathcal{C}_\mathcal{R})$ we say that $\mathcal{R}$ *accepts* $(t_1, \dots, t_n)$ if $f(t_1, \dots, t_n) \to^!_\mathcal{R} t$ such that $\mathsf{rt}(t) = a$. The *language accepted by* $\mathcal{R}$ is the set

$$\mathcal{L}(\mathcal{R}) = \{(t_1, \dots, t_n) \mid t_1, \dots, t_n \in \mathcal{T}(\mathcal{C}_\mathcal{R}) \wedge \mathcal{R} \text{ accepts } (t_1, \dots, t_n)\} \; .$$

Definition 3.10. Let $\mathcal{R}$ be a TRS with main function f of arity n, accepting symbol a. Let $L \subseteq \mathcal{T}(\mathcal{C}_\mathcal{R})^n$, and let $\mathcal{C}$ be a set of computable, strictly increasing, and total functions. We say that $\mathcal{R}$ *(deterministically) accepts* L *in time* $\Xi(\mathcal{C})$ if $\mathcal{L}(\mathcal{R}) = L$, $\mathcal{R}$ is confluent, and $\mathsf{rc}_\mathcal{R} \leqslant \Xi(\mathcal{C})$.

The notions of acceptance for Turing Machines and TRSs are indeed related in the natural way by Δ:

Lemma 3.11. *Let M be a dual-tape Turing Machine with tape alphabet Σ and accepting state q_a. For each word $x \in \Sigma^*$, M accepts x if and only if $\Delta(M)$ with main function* runM *and accepting symbol q_a accepts $\phi(x)$. Moreover, $\mathcal{L}(\Delta(M))$ is exactly the language accepted by M.*

Proof. Follows by the arguments used to prove Lemma 3.3. □

Theorem 3.12. *Let $\mathcal{C}$ be a recursively enumerable set of computable, strictly increasing, and total functions. Then the following decision problem is Σ^0_3-hard:*

Instance: A dual-tape Turing Machine M.

Question: Does there exist a dual-tape Turing Machine M' accepting the same language as M such that $\textsc{Time}_{M'} \leqslant \Xi(\mathcal{C})$?

Proof. By [91, Corollary 3], for each set C of decidable languages containing an infinite language A and all languages B such that $A \setminus B$ is finite, the following problem is Σ^0_3-hard:

Instance: A (dual-tape) Turing Machine M.

Question: Is the language accepted by M contained in C?

Fix C to be the set of all languages L decided by any (dual-tape) Turing Machine M' with $\textsc{Time}_{M'} \leqslant \Xi(\mathcal{C})$. As $\mathcal{C}$ contains a strictly increasing function, C contains an infinite language A and all languages B such that $A \setminus B$ is finite. For instance, Σ^*, where Σ is the tape alphabet of M, is a suitable instance of A here. Thus C satisfies the assumptions of [91, Corollary 3], and the theorem follows. □

Now we have all necessary ingredients to show the main theorem of this section. Theorem 3.12 yields the corresponding result for Turing Machines, while Lemma 3.11 and [6] form the bridge to term rewriting.

Theorem 3.13. *Let $\mathcal{C}$ be a recursively enumerable set of computable, strictly increasing, and total functions such that $\Xi(\mathcal{C})$ is closed under polynomial slowdown. Then the following decision problem is Σ_3^0-complete, even if the problem instances are restricted to orthogonal TRSs:*

Instance: A TRS $\mathcal{R}$.

Question: Does there exist a TRS which accepts $\mathcal{L}(\mathcal{R})$ in time $\Xi(\mathcal{C})$?

Proof. First we show that the problem is contained in Σ_3^0. Let $P(x_1, x_2, x_3, x_4)$ be the predicate on $\mathbb{N}$ that holds exactly if the i^{th} function g_i in $\mathcal{C}$, the TRS $\mathcal{S}$ encoded by l, and the TRS $\mathcal{R}$ encoded by x_4 satisfy the following properties:

- x_1 encodes the 4-tuple (i, j, k, l).
- $\mathsf{rc}_{\mathcal{R}}(x_2) \leqslant x_3$ and $\mathsf{rc}_{\mathcal{S}}(x_2) \leqslant g_i(j \cdot x_2 + k)$.
- $\mathcal{R}$ and $\mathcal{S}$ have the same main function f, accepting symbol a, and constructors $\mathcal{C}_{\mathcal{R}}$ in their signatures $\mathcal{F}_{\mathcal{R}}$ and $\mathcal{F}_{\mathcal{S}}$.
- For all $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}_{\mathcal{R}})$ with $|f(t_1, \ldots, t_n)| \leqslant x_2$, there exists $u_1 \in \mathcal{T}(\mathcal{F}_{\mathcal{R}})$ with $\mathsf{rt}(u_1) = a$ and $f(t_1, \ldots, t_n) \to_{\mathcal{R}}^! u_1$ if and only if there exists $u_2 \in \mathcal{T}(\mathcal{F}_{\mathcal{S}})$ with $\mathsf{rt}(u_2) = a$ and $f(t_1, \ldots, t_n) \to_{\mathcal{S}}^! u_2$.

Observe that $P(x_1, x_2, x_3, x_4)$ is a decidable predicate: As $\mathcal{C}$ is recursively enumerable and consists of computable functions, we may compute $g_i(j \cdot x_2 + k)$; as the signature and set of rules of both $\mathcal{R}$ and $\mathcal{S}$ are finite, we may compute the finite set of basic terms over $\mathcal{F}_{\mathcal{R}}$ (respectively $\mathcal{F}_{\mathcal{S}}$) of size at most x_2, and for each of these compute their derivation trees up to depth x_3 (respectively $g_i(j \cdot x_2 + k)$) and subsequently check whether the leaves of each derivation tree consist only of normal forms, and whether all trees are non-circular. If that is the case, then the set of normal forms of the considered terms is finite, as well, and hence it is computable whether $f(t_1, \ldots, t_n) \to_{\mathcal{R}}^! u_1$ and $f(t_1, \ldots, t_n) \to_{\mathcal{S}}^! u_2$ for all relevant $t_1, \ldots, t_n, u_1, u_2$. As the answer to the question to be decided is "yes" for the TRS encoded by x_4 if and only if the predicate $\exists x_1 \forall x_2 \exists x_3\, P(x_1, x_2, x_3, x_4)$ holds, containment in Σ_3^0 is proved.

We now show Σ_3^0-hardness of the problem. By Theorem 3.12, it is Σ_3^0-hard to decide whether there exists a dual-tape Turing Machine M' accepting the same language as M such that $\textsc{Time}_{M'} \leqslant \Xi(\mathcal{C})$, given a dual-tape Turing Machine M. Let q_a be the accepting state of M. We set runM to be the main function, and q_a the accepting symbol of $\Delta(M)$. Note that $\Delta(M)$ is orthogonal, so the reduction described here works regardless of whether the problem instance is restricted to be orthogonal. By Lemma 3.11, $L = \mathcal{L}(\Delta(M))$ is the language accepted by M. It remains to show that there exists a dual-tape Turing machine M' accepting L with $\textsc{Time}_{M'} \leqslant \Xi(\mathcal{C})$ if and only if there exists a TRS $\mathcal{R}'$ (deterministically) accepting L in time $\Xi(\mathcal{C})$.

In order to show the direction from left to right, suppose that there exists a dual-tape Turing machine M' accepting L with $\textsc{Time}_{M'} \leqslant \Xi(\mathcal{C})$. Then by employing Lemma 3.11 again, we obtain $\mathcal{L}(\Delta(M')) = L$ if we set the main function to runM again, and the accepting symbol of $\Delta(M')$ to the accepting state of M'. Moreover, by the construction of Δ, the TRS $\Delta(M')$ is orthogonal. Thus, $\Delta(M')$ (deterministically) accepts L in time $\Xi(\mathcal{C})$.

For the direction from right to left, suppose that there exists a confluent TRS $\mathcal{R}'$ with main function f, accepting symbol a, and $\mathsf{rc}_{\mathcal{R}'} \leqslant \Xi(\mathcal{C})$. Then by [6, Theorem 6.2] there exists a deterministic (dual-tape) Turing Machine M' such that $\textsc{Time}_{M'}(n) \in \mathsf{O}(\log(\mathsf{rc}_{\mathcal{R}'}(n))^3 \cdot \mathsf{rc}_{\mathcal{R}'}(n)^7)$. Since $\mathsf{rc}_{\mathcal{R}'} \leqslant \Xi(\mathcal{C})$, and $\Xi(\mathcal{C})$ is by assumption closed under polynomial slowdown, we have $\textsc{Time}_{M'} \leqslant \Xi(\mathcal{C})$, as well. ☐

3.4 Derivational Complexity

We proceed to give the completeness result for establishing upper bounds on the derivational complexity of TRSs. Unfortunately, we cannot lift the results of Section 3.2 directly from runtime complexity to derivational complexity. The definition of the derivational complexity of a TRS places no restrictions on the considered starting term; in particular, we have to consider encodings of unreachable configurations in the underlying Turing Machine. The crucial ingredient of the main theorem in this section is an investigation by Herman [41] of the *mortality problem* for Turing Machines. Herman's proof gives a concrete reduction of the mortality problem from the halting problem that involves only a polynomial overhead in time complexity. In order to use this reduction, we switch from dual-tape to single-tape Turing Machines for this section.

Theorem 3.14 ([39, Theorem 6])**.** *Let M be a dual-tape Turing Machine. Then there exists a single-tape Turing Machine M' which accepts and rejects exactly the same input words as M, and satisfies the property* $\textsc{Time}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^2, n^2\})$.

Lemma 3.15. *Let M be a single-tape Turing Machine with tape alphabet Σ. Then there exists a single-tape Turing Machine M' such that M' accepts and rejects exactly the same input words from Σ^* as M, M' halts on all configurations if and only if M halts on all input words, and* $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^3, n^3\})$.

Proof. By [41, Theorem 1], there exists a single-tape Turing Machine M' which accepts the same input words from Σ^* as M, and halts on all configurations if and only if M halts on all input words. We start by giving the construction of [41, Theorem 1], adapted to our particular setting. The only major difference is that [41] considers Turing machines with a tape growing both left and right, whereas our machines only have tapes growing to the right.

Let $\Box$ be the blank symbol, $*$ the left end marker, Q the set of states, q_s the starting state, and q_a and q_r the accepting and rejecting states of M. The tape alphabet Σ' of M' is Σ, extended by the following symbols:

d_1, d_2, d_3	delimiters of tape sections
$\vdash$	left end marker of M'
$p_{(q,x)}$	"position of the head of M, scanning symbol x in state q"

For each state $q \in Q$ and tape symbol $x \in \Sigma$, there exists a symbol $p_{(q,x)}$ in Σ'. The left end marker of M' is $\vdash$, and the blank symbol of M' is $\Box$, the same as for M.

The Turing Machine M' is built from five submachines called **START**, **SIM**, **RSPACE**, **CHECK**, and **RESTART**. The submachine **START** sets up the format of the tape required for the simulation of M. **SIM** is responsible for simulating a single move of M in M'. **RSPACE** extends the tape space dedicated to storing the tape contents of M to the right when necessary. **CHECK** checks whether the tape of M' currently has the format assumed for the simulation. Finally, **RESTART** is responsible for restoring the initial configuration of M in the simulation.

Whenever control is passed to **SIM**, from **CHECK** or **RESTART**, the tape contents has the shape

$$\vdash \Box^{u-v} *^{v} d_1 * w d_2 * w_1 p_{(q,x)} w_2 d_3 \ ,$$

with $w, w_1, w_2 \in \Sigma^*$, $q \in Q$, $x \in \Sigma$ and the head pointing somewhere between $\vdash$ and d_1. This corresponds to the configuration $(q, w_1 x w_2, i)$ of M, where i points to the symbol x.

The five submachines work as follows:

- The submachine **START** copies the input word and puts in some of the new symbols introduced in Σ'. Concretely, on an input word w over Σ^*, **START** passes control to **SIM** with the tape

  $$\vdash d_1 * w d_2 p_{(q_s,*)} w \Box d_3$$

  and the head pointing to $p_{(q_s,*)}$. On the other hand, if $w \notin \Sigma^*$, then M' immediately rejects.

- The submachine **SIM** starts by moving the tape head rightwards until it finds a symbol $p_{(q,x)}$. If $q = q_a$ or $q = q_r$, then M' immediately accepts or rejects, respectively. If q is neither q_a nor q_r, then a single step done by M on state q and tape symbol x is simulated by **SIM**. If the symbol immediately to the right of the new $p_{(q',x')}$ symbol (added by the simulation of the move of M) is d_3 at this point, then **SIM** passes control to **RSPACE**. On the other hand, if any symbol from Σ (the tape alphabet of M) is immediately to the right of $p_{(q',x')}$, then control is passed to **CHECK**.

- The submachine **RSPACE** moves (the unique occurrence of) the symbol d_3 one cell to the right. A $\Box$ is inserted in the old position of d_3. Finally, control is given to **CHECK**.

- The submachine **CHECK** checks whether the contents of the tape is of the form assumed by the simulation. If any of the checks described below fails, M' immediately rejects. First, **CHECK** moves the tape head to the right, searching for a d_3 symbol. Next, it moves the tape head to the left until it reaches d_2. While moving towards d_2, it checks whether all symbols encountered on the way are elements of Σ, with exactly one exception, which must be $p_{(q,x)}$ for some $q \in Q$ and $x \in \Sigma$. Moreover, the last symbol encountered before d_2 must be either $*$, or $p_{(q,*)}$ for some $q \in Q$. After reading d_2, **CHECK** moves further to the left until it encounters d_1. While moving towards d_1, it checks whether all symbols encountered on that way are elements of Σ, and the last symbol right of d_1 must be $*$. Finally, it moves to the left even further, until a $\square$ or $\vdash$ is encountered (other than that, only $*$ symbols are allowed to the left of d_1). If a $\square$ is encountered first, it is replaced by a $*$, and control is passed to **SIM**. If $\vdash$ is encountered first, then all $*$ symbols are replaced by $\square$ symbols. Afterwards, the whole contents of the tape (up to d_3) is moved one space to the right, inserting an additional $\square$ to the right of $\vdash$. Then control is passed to **RESTART**. This mechanism ensures that the number of steps simulated without any call to **RESTART** is bounded by the number of $\square$ symbols to the left of d_1.

- If the number of $\square$ symbols to the left of d_1 is not sufficient for simulating M on the given input, the submachine **RESTART** increases this upper bound by one, and restores the starting configuration of M. Specifically, everything between d_2 and d_3 is deleted from the tape. Then, **RESTART** copies the string between d_1 and d_2 between d_2 and d_3, replacing the leftmost $*$ by the symbol $p_{(q_s,*)}$. Finally, control is passed to **SIM**.

This finishes the description of the construction given in [41, Theorem 1]. We now show that $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\textsc{Time}_M(n)^3)$. First, observe that for each submachine $N \in \{$**START**, **SIM**, **RSPACE**, **CHECK**, **RESTART**$\}$ on its own, we have $\textsc{LifeTime}_N(n) \in \mathsf{O}(n)$. Hence, we only consider runs of M' containing loops involving **SIM** and **CHECK** (and possibly **RSPACE** and **RESTART**). Note that such a loop can only have more than one iteration if at the end of the run of the current submachine, the tape contents t is of the form

$$s_l \vdash \square^{u-v} *^v d_1 * w d_2 * w_1 p_{(q,x)} w_2 d_3 s_r \ ,$$

with $s_l, s_r \in \Sigma'^*$, $w, w_1, w_2 \in \Sigma^*$, $q \in Q$, $x \in \Sigma$, and the head pointing somewhere between $\vdash$ and d_3.

In that case, M is simulated for $u - v$ many steps, then control is passed to **RESTART** (unless M halts during the simulation). Thanks to **RESTART**, the initial configuration of M for input word w is restored, and u is incremented by one. This process is repeated until u is large enough to allow the simulation to reach the halting state, i.e. at most $\textsc{Time}_M(|w|)$ many times. Hence, the total number of loop iterations is bounded quadratically in $\max\{\textsc{Time}_M(|t|), |t|\}$. Since the number of moves done in each loop iteration is bounded linearly in $\textsc{Time}_M(|t|)$, the Lemma follows. $\square$

We now encode single-tape Turing Machines M as TRSs $\Delta_1(M)$. Similar to Section 3.2, we use the encoding of [102, Section 5.3.1]: a configuration (q, w, i) such that $w = w_1 \dots w_n$ is encoded as the term $q(\phi(w_{i-1} \dots w_1), \phi(w_i \dots w_n))$. However, we slightly change the rules of Δ_1 to reflect that we consider machines with only one-way infinite tapes for simplification purposes. Note that Δ_1 does not contain any mechanism to enforce any restriction on the starting term of a derivation; this is because we are considering derivational complexity (rather than runtime complexity) in this section.

Definition 3.16. Let $M = (Q, \Sigma, \delta)$ be a single-tape Turing Machine. Then the orthogonal constructor TRS $\Delta_1(M)$ is defined by the rules shown in Figure 3.2.

transition function	rewrite rule ($q \in Q \setminus \{q_a, q_r\}$ and $a, b \in \Sigma$)
$\delta(q, b) = (q', b', R)$	$q(x, by) \to q'(b'x, y)$
$\delta(q, b) = (q', b', L)$	$q(ax, by) \to q'(x, ab'y)$
$\delta(q, \Box) = (q', b', R)$	$q(x, \rhd) \to q'(b'x, \rhd)$

Figure 3.2: The TRS $\Delta_1(M)$ defined by a single-tape Turing Machine M

We call a ground term of the shape $q(s, t)$ over the signature of $\Delta_1(M)$ a *restricted term* if $q \in Q$, and $s, t \in \Sigma^*$, and the first symbol of $s^{-1}t$ is $\vdash$ (here $(\cdot)^{-1}$ denotes string reversal).

Lemma 3.17. *Let M be a single-tape Turing Machine. Then $\mathsf{dc}_{\Delta_1(M)}(n) \in$ $\textsc{LifeTime}_M(\Omega(n))$ and $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$.*

Proof. The following holds by straightforward arguments (compare [102, Exercise 5.3.3]):

- For each restricted term s encoding a configuration α of M such that $s \to_{\Delta_1(M)} s'$, the term s' is also restricted, and encodes a configuration β of M. Moreover, M moves from α to β in a single step.

- Whenever α and β are configurations of M such that M moves from α to β in a single step, then for each (restricted) term encoding s of α, there exists a (restricted) term encoding s' of β such that $s \to_{\Delta_1(M)} s'$.

The above implies that for each configuration α of M, the derivation height $\mathsf{dh}(s, \to_{\Delta_1(M)})$ is exactly the number of moves that can be done from α until M halts, where s is a (restricted) term which encodes α. Therefore, $\mathsf{dc}_{\Delta_1(M)}(n) \in$ $\textsc{LifeTime}_M(\Omega(n))$.

It remains to show that $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$. It easily follows from the above observations that $\mathsf{rc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\mathsf{O}(n))$. We use the construction of [29, Appendix B.2], which allows us to lift this upper bound to starting terms of arbitrary shape. We define two functions f and g. The function f maps ground terms over the signature $\mathcal{F}$ of $\Delta_1(M)$ to pairs containing a string over the tape alphabet, and a multiset of restricted terms over $\mathcal{F}$. The purpose of f (compare [29, Lemma B.5]) is to extract a number of restricted terms from a term. The helper function g ensures that the leftmost symbol on the tape of each configuration encoded by a restricted term is indeed a $\vdash$.

$$\begin{aligned}
f(\triangleright) &= (\triangleright, \emptyset) \\
f(a(x)) &= (a(w), \mathcal{M}) && \text{if } a \in \Sigma,\ f(x) = (w, \mathcal{M}) \\
f(q(x,y)) &= (\triangleright, \{q(g(w,v))\} \cup \mathcal{M}_1 \cup \mathcal{M}_2) && \text{if } q \in Q,\ f(x) = (w, \mathcal{M}_1), \\
& && \quad \text{and } f(y) = (v, \mathcal{M}_2) \\
g(\triangleright, \vdash(v)) &= (\triangleright, \vdash v) \\
g(\triangleright, v) &= (\vdash(\triangleright), v) \\
g(\vdash(\triangleright), v) &= (\vdash(\triangleright), v) \\
g(a(\triangleright), v) &= (a(\vdash(\triangleright)), v) && \text{if } a \in \Sigma \setminus \{\vdash\} \\
g(a(x), v) &= (a(y), z) && \text{otherwise, if } a \in \Sigma \\
& && \quad \text{and } (y, z) = g(x, v)
\end{aligned}$$

By [29, Lemma B.8], we get that for every term t over $\mathcal{F}$ with $f(t) = (w, \mathcal{M})$, the inequality $\mathsf{dh}(t, \to_{\Delta_1(M)}) \leqslant \sum_{s \in \mathcal{M}} \mathsf{dh}(s, \to_{\Delta_1(M)})$ holds. Moreover, $\sum_{s \in \mathcal{M}} |s| \leqslant |t|$. Hence, $\mathsf{dc}_{\Delta_1(M)}(n) \leqslant n \cdot \mathsf{rc}_{\Delta_1(M)}(n)$. Thus, the above observations about restricted terms suffice in order to conclude $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$. ☐

An alternative idea for showing Lemma 3.17 might be to transfer [115, Theorem 14], which states that termination of a many-sorted TRS is equivalent to termination of the corresponding TRS without any sort information under certain conditions, from termination to complexity. The sort information could then be used to enforce that only restricted terms are considered. However, the proof of [115, Theorem 14] is an indirect one, based on the nonexistence of *minimal counterexamples*. The role of these minimal counterexamples in this proof is akin to the role of minimal infinite chains in the dependency pair framework. Hence, lifting this proof argument from termination to complexity (as done in Chapter 6 for the dependency pair framework) seems to be a nontrivial task.

We are now able to transfer Theorem 3.5 to derivational complexity of term rewriting. Theorem 3.14 and Lemma 3.15 take care of the the unrestrictedness of the considered starting terms, and Lemma 3.17 performs the actual transfer from Turing Machines to TRSs.

Theorem 3.18. *Let $\mathcal{C}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$ such that $\Xi(\mathcal{C})$ is closed under polynomial slowdown. Then the following decision problem is Σ_2^0-complete, even if the problem instances are restricted to orthogonal or constructor TRSs:*

Instance: A TRS $\mathcal{R}$.

Question: Is $\mathsf{dc}_{\mathcal{R}} \leqslant \Xi(\mathcal{C})$?

Proof. The proof of containment of the problem in Σ_2^0 is identical to Theorem 3.6 *mutatis mutandis*, whence we only show its Σ_2^0-hardness. By Theorem 3.5, it is Σ_2^0-hard to decide, given a dual-tape Turing Machine M, whether $\textsc{Time}_M \leqslant \Xi(\mathcal{C})$. By Theorem 3.14 and Lemma 3.15, there exists a single-tape

Turing machine M' such that $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^6, n^6\})$, and M' accepts the same language as M. As $\Xi(\mathcal{C})$ is by assumption closed under polynomial slowdown, and contains a strictly increasing function (and hence also a function dominating $i'(n) = n^6$), we have $\textsc{LifeTime}_{M'} \leqslant \Xi(\mathcal{C})$ if and only if $\textsc{Time}_M \leqslant \Xi(\mathcal{C})$. Moreover, by Lemma 3.17, we have $\mathsf{dc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\Omega(n))$ and $\mathsf{dc}_{\Delta_1(M')} \in n \cdot \textsc{LifeTime}_{M'}(\mathsf{O}(n))$. As $\Xi(\mathcal{C})$ is closed under polynomial slowdown, it follows that $\mathsf{dc}_{\Delta_1(M')} \leqslant \Xi(\mathcal{C})$ if and only if $\textsc{LifeTime}_{M'} \leqslant \Xi(\mathcal{C})$. The transformations used in Theorem 3.14 and Lemmata 3.15 and 3.17 are obviously computable, and $\Delta_1(M')$ is an orthogonal constructor TRS. Therefore, it is Σ^0_2-hard to decide whether $\mathsf{dc}_\mathcal{R} \leqslant \Xi(\mathcal{C})$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal or a constructor TRS). $\square$

Note that in the above proof, $\mathsf{dc}_{\Delta_1(M')} \leqslant \Xi(\mathcal{C})$ if and only if $\mathsf{rc}_{\Delta_1(M')} \leqslant \Xi(\mathcal{C})$, hence this is also an alternative proof of the Σ^0_2-completeness of determining whether $\mathsf{rc}_\mathcal{R} \leqslant \Xi(\mathcal{C})$, which places slightly stricter assumptions on $\mathcal{C}$, but allows $\mathcal{R}$ to be restricted to constructor TRSs.

3.5 Minimal Complexity

The proofs in this section and Section 3.6 below are based on the observation that the simulation of a Turing machine M by the TRS $\Delta(M)$ has exactly one redex in each term encoding a configuration of M. Every ilk of problem we consider concerns sets of reductions to some normal form; if there is only one possible reduction starting from every term encoding of each Turing Machine configuration, the proofs of *hardness* of the various kinds of problems we consider remain virtually identical, regardless of whether we consider minimal or maximal reductions, and regardless of reduction strategy. This crucial observation is stated in Lemma 3.20 below.

Definition 3.19. We define the *minimal height* of a term s with respect to a finitely branching, well-founded binary relation $\to$ by $\mathsf{mh}(s, \to) = \min\{n \mid \exists t\ s \to^{n,!}_\mathcal{R} t\}$. The twin notions of *minimal derivational complexity* and *minimal runtime complexity* of a TRS $\mathcal{R}$ are then defined by:

$$\begin{aligned}\mathsf{mdc}_\mathcal{R}(n) &= \max\{\mathsf{mh}(s, \to_\mathcal{R}) \mid |s| \leqslant n\}\\ \mathsf{mrc}_\mathcal{R}(n) &= \max\{\mathsf{mh}(s, \to_\mathcal{R}) \mid |s| \leqslant n \land s \in \mathcal{B}_\mathcal{R}\}\end{aligned}$$

The next lemma states the crucial observation for extending our previous results to the notions of minimal complexity.

Lemma 3.20. *Let M be a dual-tape Turing machine and let s be a term in the signature of $\Delta(M)$ containing exactly one redex. If $s \to_{\Delta(M)} t$, then t contains at most one redex.*

Proof. By assumption, the only redex of s is the one contracted by the step $s \to_{\Delta(M)} t$. Hence, t only contains redexes created by that step. As $\Delta(M)$ is left-linear, redexes can only be created if the right-hand side of the rule $l \to r$

employed in $s \to_{\Delta(M)} t$ overlaps with a left-hand side of some other rule. Write $s \to_{\Delta(M)} t$ as $C[l\sigma] \to_{\Delta(M)} C[r\sigma]$ for a suitable context C and substitution σ. Split on cases as follows:

- If $\mathsf{rt}(l)$ is either some state q or runM, inspection of the rules of $\Delta(M)$ yields that r is of the form $q'(\mathsf{ok}(\rhd), x, y, z, w)$ or $q'(\rhd, x, y, z, w)$ for some state q and words x, y, z, and w. Clearly, r can only overlap with the left-hand side of a rule $l' \to r'$ if the overlap occurs at the root of l' and r. As $\Delta(M)$ is orthogonal, at most one such rule $l' \to r'$ can exist, and hence there is at most one redex in t.
- If $l = \mathsf{ok}(\mathsf{ok}(\rhd))$, then $r = \rhd$. Obviously, $\rhd$ is a normal form on its own. By assumption, C contains no redex on its own, and $\Delta(M)$ is orthogonal. Therefore, $C[\rhd]$ contains at most one redex.

□

Theorem 3.21. *Theorems 3.6, 3.8, and 3.13 all hold with the notion of* $\mathsf{rc}_{\mathcal{R}}$ *replaced by* $\mathsf{mrc}_{\mathcal{R}}$ mutatis mutandis.

Proof. Every basic term s in an orthogonal TRS (such as $\Delta(M)$ for a dual-tape Turing Machine M) contains at most one redex. For each TRS on the form $\Delta(M)$, it is therefore immediate by Lemma 3.20 that the minimum and maximum lengths of reduction to normal form from s are the same. Therefore, all arguments in the hardness proofs of Theorems 3.6, 3.8, and 3.13 remain sound if we replace $\mathsf{rc}_{\mathcal{R}}$ by $\mathsf{mrc}_{\mathcal{R}}$, so the hardness results follow.

For containment in the respective complexity classes, observe that in the proofs of Theorems 3.6, 3.8, and 3.13 (each of the three distinct variations of) the predicate P considers longest maximal paths (maximal in the sense that the path is not a strict prefix of any other path) in the derivation tree of terms; this can obviously be replaced by the shortest maximal paths, as required by $\mathsf{mrc}_{\mathcal{R}}$, without affecting computability of P. □

Theorem 3.22. *Theorem 3.18 holds with the notion of* $\mathsf{dc}_{\mathcal{R}}$ *replaced by* $\mathsf{mdc}_{\mathcal{R}}$ mutatis mutandis.

Proof. Containment in Σ^0_2 follows in the same way as in the proof of Theorem 3.21.

We now show Σ^0_2-hardness. Observe that for any (single-tape) Turing Machine M, the TRS $\Delta_1(M)$ is orthogonal, right-linear, and nonerasing. Therefore, for all terms s, t, and u such that $u \to_{\Delta_1(M)} s$ and $u \to_{\Delta(M)} t$, either $s = t$ or there exists a term v such that $s \to_{\Delta(M)} v$ and $t \to_{\Delta(M)} v$: due to orthogonality and nonerasingness, any rewrite step in $\Delta(M)$ keeps all other redexes of the considered term intact. Moreover, due to right-linearity, rewrite steps do not create additional copies of other redexes, either, hence the order in which any two redexes of a term are contracted does not change the resulting term. Due to this property, it follows by a straightforward inductive argument that $\mathsf{dh}(t, \to_{\Delta_1(M)}) = \mathsf{mh}(t, \to_{\Delta_1(M)})$ for any term t. With this equality, the hardness result follows by arguments identical to those in the proof of Theorem 3.18. □

3.6 Strategies

The results so far concern TRSs with unconstrained rewrite relation. In the modelling of programming languages, it is common to consider TRSs with strategies dictating the redex to be contracted in each term. Using the same ideas as in the last section, the previous results in the paper carry over to the setting of TRSs with strategies. Thus, the results of the previous sections of the paper remain valid under, for example, any innermost strategy, and under deterministic strategies such as the leftmost-outermost strategy.

Definition 3.23. Let $\mathcal{R}$ be a TRS. A *strategy* $\mathbb{S}$ for $\mathcal{R}$ is defined by a relation $\to_{\mathbb{S}} \subseteq \to_{\mathcal{R}}$ such that any term t is a normal form of $\to_{\mathcal{R}}$ if and only if it is a normal form of $\to_{\mathbb{S}}$. We call a strategy for $\mathcal{R}$ *computable* if, given a term t, the (finite) set $\{t' \mid t \to_{\mathbb{S}} t'\}$ is computable.

Here we use the notion "strategy" according to [102, Definition 9.1.1]. Note that this does not cover everything that is commonly called a "strategy" in term rewriting. For instance, the proofs of this section can not be directly carried over to *context-sensitive rewriting*. Indeed, for restrictive context-sensitive "strategies", the blocking of redexes might decrease the complexity of the considered TRS dramatically. For example, a context-sensitive "strategy" which blocks all redexes would make any derivational complexity bounds easily decidable. Hence, we stick to the notion of "strategy" given in Definition 3.23.

The notions of runtime and derivational complexity of TRSs with strategies are defined *mutatis mutandis*. For the next theorem, Lemma 3.20 is again the crucial proof ingredient.

Theorem 3.24. *Let f be a computable mapping returning a computable strategy $f(\mathcal{R})$ for each TRS $\mathcal{R}$. Theorems 3.6, 3.8, and 3.13, 3.18, 3.21 and 3.22 all hold for the rewrite relation of $\mathcal{R}$ with strategy $\mathbb{S} = f(\mathcal{R})$ (where the instance in each decision problem is $\mathcal{R}$).*

Proof. Observe that if a term s contains exactly one redex, then for any term t and strategy $\mathbb{S}$ for $\mathcal{R}$, we have $s \to_{\mathcal{R}} t$ if and only if $s \to_{\mathbb{S}} t$. For every TRS of the form $\Delta(M)$, each basic term of $\Delta(M)$ has at most one redex. By Lemma 3.20, it is immediate that the lengths of all reductions to normal form from s are the same. Therefore, all arguments in the hardness proofs of Theorems 3.6, 3.8, 3.13, and 3.21, remain sound under $\mathbb{S}$.

For Σ_2^0-hardness of the remaining two properties, observe that for any (single-tape) Turing Machine M, the TRS $\Delta_1(M)$ is orthogonal, right-linear, and non-erasing. Therefore, for all terms s, t, and u such that $u \to_{\Delta_1(M)} s$ and $u \to_{\Delta(M)} t$, either $s = t$ or there exists a term v such that $s \to_{\Delta(M)} v$ and $t \to_{\Delta(M)} v$. It follows that for any term t, all reductions from t to its (unique) normal form have the same length. In particular, we have $\mathsf{dh}(t, \to_{\Delta_1(M)}) = \mathsf{dh}(t, \to_{\mathbb{S}})$. Hence, the hardness proofs for Theorems 3.18 and 3.22 remain sound when restricted to $\mathbb{S}$.

To prove containment in the respective classes of the arithmetical hierarchy, observe that each containment proof in Theorems 3.6, 3.8, 3.13, 3.18, 3.21, and 3.22 is done by computing the derivation tree starting from a term s to a certain depth. The derivation tree with respect to a strategy can be obtained by pruning the full derivation tree: a branch $t \to_{\mathcal{R}} t''$ (and thus, the entire subtree starting from t'') is cut off if and only if $t'' \notin \{t' \mid t \to_{\mathbb{S}} t'\}$. As the strategy is computable, the pruning operation is clearly computable, hence also the pruned derivation trees, and we may thus replace the trees in the proofs of the above theorems by their pruned versions, concluding the proof. □

3.7 Conclusion

In this chapter, we have proved that a number of problems related to bounding the derivational and runtime complexity of rewrite systems are complete for classes in the arithmetical hierarchy. In particular, it is Σ^0_2-complete to decide whether the derivational complexity, the innermost derivational complexity, the runtime complexity, or the innermost runtime complexity of a given TRS is bounded by a polynomial. This sets the stage for the remainder of this thesis: since by Theorem 3.18, the problem whether the derivational complexity of a given TRS is within some fixed complexity class is highly undecidable (for many frequently used complexity classes), it is necessary to resort to partial decision procedures for this problem. Moreover, due to derivational complexity analysis being in a sense an extension of termination analysis (once termination of some program or TRS is established, the question arises how fast termination happens), the method used in the subsequent chapters to establish upper bounds on the derivational complexity of TRSs (inference of the bounds from a termination proof) is a very natural one. However, it should be noted that the two problems are located in slightly different positions in the arithmetical hierarchy: while deciding termination is Π^0_2-complete [26], deciding complexity bounds is Σ^0_2-complete, putting derivational complexity analysis into the same position in the arithmetical hierarchy as *non*termination analysis. This suggests that it might be promising to try to certify polynomial (or other) complexity bounds for TRSs in a completely novel way.

We also hope that our results may be used to prove the exact hardness of other problems in applied logic—this would avoid the tedium of pure reduction from Turing Machines.

A related open problem is Problem #107 of the *RTA list of open problems*[1], a list of open problems collected by researchers in term rewriting: what are complete characterisations of polynomial derivational complexity?

[1]`http://rtaloop.mancoosi.univ-paris-diderot.fr/`

Chapter 4

Classical Termination Proof Techniques

Science is what we understand well enough to explain to a computer. Art is everything else we do.

Donald Ervin Knuth

The idea of investigating upper bounds on the derivational complexity of TRSs whose termination can be proved by specific techniques ranges back more than 20 years [50]. In the meantime, many (mechanisable) termination proof techniques have been analysed in this way. In this chapter, we recall the most important of these techniques and the corresponding historic complexity results.

Generally, reduction orders can be divided into those operating on a purely *syntactic* basis (i.e. comparing terms purely by their structure), and those using *semantic* means to compare terms (e.g. by assigning values from some other domain to terms) [115]. Both kinds of methods are represented in this chapter.

4.1 Well-founded Monotone Algebras

Most semantic termination proof techniques (in particular, the techniques presented in this section) are based on *well-founded monotone algebras*, cf. [115]. Well-founded monotone algebras purely depend on interpreting terms into some well-founded domain.

Definition 4.1 ([70])**.** An *$\mathcal{F}$-algebra* $\mathcal{A}$ consists of a set A (called the *carrier* of $\mathcal{A}$) and, for every function symbol $f \in \mathcal{F}$, an *interpretation function* $f_\mathcal{A}\colon A^n \to A$, where n is the arity of f. Given an *assignment* $\alpha\colon \mathcal{V} \to A$, we write $[\alpha]_\mathcal{A}(t)$ to denote the evaluation of a term t by $\mathcal{A}$. This evaluation is defined inductively as follows:

$$[\alpha]_\mathcal{A}(t) = \begin{cases} \alpha(t) & \text{if } t \in \mathcal{V} \\ f_\mathcal{A}([\alpha]_\mathcal{A}(t_1), \dots, [\alpha]_\mathcal{A}(t_n)) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

A *well-founded monotone $\mathcal{F}$-algebra* is a pair $(\mathcal{A}, >)$, where $\mathcal{A}$ is an $\mathcal{F}$-algebra and $>$ is a well-founded proper order such that for every function symbol $f \in \mathcal{F}$, $f_\mathcal{A}$ is strictly monotone in all coordinates with respect to $>$. For any well-founded monotone $\mathcal{F}$-algebra $(\mathcal{A}, >)$, the following order $>_\mathcal{A}$ is defined:

$$s >_\mathcal{A} t \iff [\alpha]_\mathcal{A}(s) > [\alpha]_\mathcal{A}(t) \text{ for all assignments } \alpha$$

Theorem 4.2 ([115])**.** *For every well-founded monotone algebra $(\mathcal{A}, >)$, the order $>_\mathcal{A}$ is a reduction order. Moreover, for any terminating TRS $\mathcal{R}$, there exists a well-founded monotone algebra $(\mathcal{A}, >)$ such that $\mathcal{R}$ is compatible with $>_\mathcal{A}$.*

A well-known natural extension of well-founded monotone algebras for generating reduction pairs in the dependency pair framework is the concept of *weakly monotone algebras.*

Definition 4.3. A *weakly monotone $\mathcal{F}$-algebra* is a triple $(\mathcal{A}, \geqslant, >)$, where $\mathcal{A}$ is an $\mathcal{F}$-algebra, $\geqslant$ is a preorder such that for every function symbol $f \in \mathcal{F}$, $f_\mathcal{A}$ is monotone in all coordinates with respect to $\geqslant$, and $>$ is a well-founded proper order such that $\geqslant \cdot > \cdot \geqslant \subseteq >$. For any weakly monotone $\mathcal{F}$-algebra $(\mathcal{A}, \geqslant, >)$, the following orders $\geqslant_\mathcal{A}$ and $>_\mathcal{A}$ are defined:

$$\begin{aligned} s \geqslant_\mathcal{A} t &\iff [\alpha]_\mathcal{A}(s) \geqslant [\alpha]_\mathcal{A}(t) \text{ for all assignments } \alpha \\ s >_\mathcal{A} t &\iff [\alpha]_\mathcal{A}(s) > [\alpha]_\mathcal{A}(t) \text{ for all assignments } \alpha \end{aligned}$$

Theorem 4.4. *For any weakly monotone algebra $(\mathcal{A}, \geqslant, >)$, the pair of orders $(\geqslant_\mathcal{A}, >_\mathcal{A})$ is a reduction pair.*

Proof. Straightforward extension of Theorem 4.2. □

We mention three specific instances of well-founded monotone $\mathcal{F}$-algebras: *polynomial interpretations*, *matrix interpretations*, and *arctic interpretations.*

Definition 4.5 ([65, 66])**.** A *polynomial interpretation* is an $\mathcal{F}$-algebra $\mathcal{A}$ such that:

1. the carrier of $\mathcal{A}$ is $\mathbb{N}$, and
2. for each function symbol $f \in \mathcal{F}$, the interpretation function $f_\mathcal{A}$ is a polynomial with range $\mathbb{N}$.

If for each function symbol $f \in \mathcal{F}$, the interpretation function $f_\mathcal{A}$ is a linear function, then we call $\mathcal{A}$ a *linear interpretation.* If for each $f \in \mathcal{F}$, there exists some $c \in \mathbb{N}$ such that $f_\mathcal{A}(x_1, \ldots, x_n) = c + \sum_{i=1}^n x_i$, then $\mathcal{A}$ is a *strongly linear interpretation.*

Theorem 4.6 ([65])**.** *Let $\mathcal{A}$ be a polynomial interpretation such that for each function symbol $f \in \mathcal{F}$, the interpretation function $f_\mathcal{A}$ is strictly monotone in all arguments. Then $(\mathcal{A}, >)$, where $>$ is the usual strict order on $\mathbb{N}$, is a well-founded monotone $\mathcal{F}$-algebra.*

Theorem 4.7. *Let $\mathcal{A}$ be a polynomial interpretation such that for each function symbol $f \in \mathcal{F}$, the interpretation function $f_\mathcal{A}$ is weakly monotone in all arguments. Then $(\mathcal{A}, \geqslant, >)$, where $\geqslant$ and $>$ are the usual weak and strict orders on $\mathbb{N}$, respectively, is a weakly monotone $\mathcal{F}$-algebra.*

Proof. Analogous to Theorem 4.6. □

If $\mathcal{A}$ is a polynomial interpretation, we often only write $\mathcal{A}$ for $(\mathcal{A}, >)$ if the premises of Theorem 4.6 hold for $(\mathcal{A}, >)$; we call $\mathcal{A}$ a *strict polynomial interpretation* in that case. Similarly, we often write $\mathcal{A}$ for $(\mathcal{A}, \geqslant, >)$ if the premises of Theorem 4.7 hold for $(\mathcal{A}, \geqslant, >)$; then we call $\mathcal{A}$ a *weak polynomial interpretation.*

The following general upper bounds on derivation heights are easy to see:

Lemma 4.8. *Let $\mathcal{R}$ be a TRS and $\mathcal{A}$ a strict polynomial interpretation such that $>_{\mathcal{A}}$ is compatible with $\mathcal{R}$. Then for all terms t and assignments α, we have* $\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant [\alpha]_{\mathcal{A}}(t)$.

Lemma 4.9. *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and $\mathcal{A}$ a weak polynomial interpretation. Suppose that $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$ completely solves $(\mathcal{P}, \mathcal{R})$. Then for all terms t and assignments α, we have* $\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathcal{P}} / \to_{\mathcal{R}}) \leqslant [\alpha]_{\mathcal{A}}(t)$.

Based on Lemma 4.8, Hofbauer and Lautemann have shown the following general complexity bound for polynomial interpretations:

Theorem 4.10 ([50])**.** *For any strict polynomial interpretation $\mathcal{A}$, the reduction order $>_{\mathcal{A}}$ induces double exponential complexity.*

It is not difficult to extend the proof of Theorem 4.10 given in [50] to restricted classes of polynomial interpretations.

Theorem 4.11 ([50])**.** *For any strict linear interpretation $\mathcal{A}$, the reduction order $>_{\mathcal{A}}$ induces exponential complexity.*

Theorem 4.12. *For any strict strongly linear interpretation $\mathcal{A}$, the reduction order $>_{\mathcal{A}}$ induces linear complexity.*

Using Lemma 4.9 instead of Lemma 4.8, the last three theorems are easily lifted to reduction pairs:

Theorem 4.13. *For any weak polynomial (respectively linear, strongly linear) interpretation $\mathcal{A}$, the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces double exponential (respectively exponential, linear) complexity.*

The second specific example of $\mathcal{F}$-algebras we mention are *matrix interpretations.* The initial formulation of matrix interpretations was given by Hofbauer and Waldmann [52]. It has been extended by Endrullis et al. [27] to the version most widely used in automatic termination proving.

We use $M_{i,j}$ to refer to the entry of a matrix M in row i and column j. We write $\|M\|$ to denote the maximum entry of a matrix M.

Definition 4.14 ([27])**.** A *matrix interpretation* is an $\mathcal{F}$-algebra $\mathcal{A}$ such that:

1. the carrier of $\mathcal{A}$ is $\mathbb{N}^d$ for some $d \in \mathbb{N} \setminus \{0\}$ (we call d the *dimension* of the interpretation), and
2. for each function symbol $f \in \mathcal{F}$, the interpretation function $f_{\mathcal{A}}$ has the shape $f_{\mathcal{A}}(x_1, \ldots, x_n) = b + \sum_{i=1}^{n} A_i \cdot x_i$, such that n is the arity of f, $b \in \mathbb{N}^d$, and A_i is a $d \times d$-matrix over $\mathbb{N}$ for all $1 \leqslant i \leqslant n$, where $+$ is the usual component-wise addition for vectors over $\mathbb{N}$, and $\cdot$ is the usual matrix-vector multiplication over $\mathbb{N}$.

Theorem 4.15 ([27]). *Let $\mathcal{A}$ be a matrix interpretation of dimension d such that for each function symbol $f \in \mathcal{F}$ and $1 \leqslant i \leqslant n$, we have $A_{i1,1} > 0$, where $f_{\mathcal{A}}(x_1, \ldots, x_n) = b + \sum_{i=1}^{n} A_i \cdot x_i$ and n is the arity of f. Let $>$ be the order on $\mathbb{N}^d$ defined by $(a_1, \ldots, a_d) > (b_1, \ldots, b_d) \iff a_1 >_{\mathbb{N}} b_1 \wedge \bigwedge_{i=2}^{d} a_i \geqslant_{\mathbb{N}} b_i$, and $>_{\mathbb{N}}$ and $\geqslant_{\mathbb{N}}$ be the usual orders on $\mathbb{N}$. Then $(\mathcal{A}, >)$ is a well-founded monotone $\mathcal{F}$-algebra.*

Theorem 4.16 ([27]). *Let $\mathcal{A}$ be a matrix interpretation of dimension d. Moreover, let $\geqslant$ and $>$ be the following orders on $\mathbb{N}^d$, where $\geqslant_{\mathbb{N}}$ and $>_{\mathbb{N}}$ are the usual orders on $\mathbb{N}$:*

$$(a_1, \ldots, a_d) \geqslant (b_1, \ldots, b_d) \iff \bigwedge_{i=1}^{d} a_i \geqslant_{\mathbb{N}} b_i$$

$$(a_1, \ldots, a_d) > (b_1, \ldots, b_d) \iff a_1 >_{\mathbb{N}} b_1 \wedge \bigwedge_{i=2}^{d} a_i \geqslant_{\mathbb{N}} b_i$$

Then $(\mathcal{A}, \geqslant, >)$ is a weakly monotone $\mathcal{F}$-algebra.

If $\mathcal{A}$ is a matrix interpretation, we often only write $\mathcal{A}$ for $(\mathcal{A}, >)$ if the premises of Theorem 4.15 hold for $(\mathcal{A}, >)$; we call $\mathcal{A}$ a *strict matrix interpretation* in that case. Similarly, we often write $\mathcal{A}$ in order to refer to $(\mathcal{A}, \geqslant, >)$, calling $\mathcal{A}$ a *weak matrix interpretation.*

As described by Waldmann in [109], there is a direct translation from matrix interpretations to *weighted automata* over the semiring constructed by $\mathbb{N}$ and standard addition and multiplication over $\mathbb{N}$. This allows the usage of results from weighted automata theory on matrix interpretations. For instance, as mentioned in Section 5.4 below, this is done in [110, 76] in order to help determine the complexity induced by reduction orders and reduction pairs based on matrix interpretations. Ingredients of termination proofs which can be translated into weighted automata over other semirings include *arctic interpretations* (described later in this section) and match-*bounds* (described in Section 4.4 below).

The following general upper bounds on derivation heights are easy to see:

Lemma 4.17. *Let $\mathcal{R}$ be a TRS and $\mathcal{A}$ a strict matrix interpretation such that $>_{\mathcal{A}}$ is compatible with $\mathcal{R}$. Then for all terms t and assignments α, $[\alpha]_{\mathcal{A}}(t) = (a_1, \ldots, a_d)$ implies $\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant a_1$.*

Lemma 4.18. *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and $\mathcal{A}$ a weak matrix interpretation. Suppose that $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$ completely solves $(\mathcal{P}, \mathcal{R})$. Then for all terms t and assignments α, $[\alpha]_{\mathcal{A}}(t) = (a_1, \ldots, a_d)$ implies $\mathsf{dh}(t, \overset{\epsilon}{\to}_{\mathcal{P}} / \to_{\mathcal{R}}) \leqslant a_1$.*

Based on Lemma 4.17, Endrullis et al. have shown the following complexity bound:

Theorem 4.19 ([27]). *For any strict matrix interpretation $\mathcal{A}$, the reduction order $>_{\mathcal{A}}$ induces exponential complexity.*

Using Lemma 4.18 instead of Lemma 4.17, the last theorem is easily lifted to reduction pairs:

Theorem 4.20. *For any weak matrix interpretation $\mathcal{A}$, the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces exponential complexity.*

The last instance of $\mathcal{F}$-algebras we mention here are *arctic interpretations* [61]. In their basic idea, they are quite similar to matrix interpretations. However, they work on a different domain, and due to technical reasons, they are rather restricted when used as the foundation of a reduction order. These restrictions will get lifted when they are used as the basis for reduction pairs instead. We now recall the carrier used by arctic interpretations.

Definition 4.21 ([61])**.** The *arctic semiring* is the set $\mathbb{A} = \mathbb{N} \cup \{-\infty\}$, together with the following operations $+_{\mathbb{A}}$ and $*_{\mathbb{A}}$:

$$m +_{\mathbb{A}} n = \begin{cases} n & \text{if } m = -\infty \\ m & \text{if } n = -\infty \\ \max\{m, n\} & \text{if } m, n \in \mathbb{N} \end{cases}$$

$$m *_{\mathbb{A}} n = \begin{cases} -\infty & \text{if } m = -\infty \text{ or } n = -\infty \\ m +_{\mathbb{N}} n & \text{if } m, n \in \mathbb{N} \end{cases}$$

The order $>_{\mathbb{A}}$ on $\mathbb{A}$ is the usual strict order on $\mathbb{N}$ extended by $a >_{\mathbb{A}} -\infty$ for all $a \in \mathbb{A}$ (in particular, $-\infty > -\infty$), and $\geqslant_{\mathbb{A}}$ is its reflexive closure. Addition of vectors or matrices over $\mathbb{A}$ is defined to be the component-wise application of $+_{\mathbb{A}}$. Matrix-vector multiplication for matrices and vectors over $\mathbb{A}$ is defined as usual, employing $+_{\mathbb{A}}$ as addition and $*_{\mathbb{A}}$ as multiplication for matrix and vector entries.

Definition 4.22 ([61])**.** An *arctic interpretation* is an $\mathcal{F}$-algebra $\mathcal{A}$ satisfying the following properties:

1. the carrier of $\mathcal{A}$ is $\mathbb{A}^d$ for some $d \in \mathbb{N}$ (we call d the *dimension* of the interpretation), and

2. for each function symbol $f \in \mathcal{F}$, the interpretation function $f_{\mathcal{A}}$ has the shape $f_{\mathcal{A}}(x_1, \ldots, x_n) = b +_{\mathbb{A}} \sum_{i=1}^{n} A_i *_{\mathbb{A}} x_i$, such that n is the arity of f, $b \in \mathbb{A}^d$, and A_i is a $d \times d$-matrix over $\mathbb{A}$ for all $1 \leqslant i \leqslant n$.

Theorem 4.23 ([61])**.** *Let $\mathcal{A}$ be an arctic interpretation of dimension d over signature $\mathcal{F}$ satisfying the following properties:*

1. *all function symbols in $\mathcal{F}$ have arity at most* 1*,*

2. *for each function symbol $f \in \mathcal{F}$ of arity* 0*, we have $f_{\mathcal{A}} \in \mathbb{N} \times \mathbb{A}^{d-1}$, and*

3. *for each function symbol $f \in \mathcal{F}$ of arity* 1*, we have $A_{1,1} \in \mathbb{N}$ and $b = (-\infty, \ldots, -\infty)$, where $f_{\mathcal{A}}(x) = b +_{\mathbb{A}} A *_{\mathbb{A}} x$.*

Let $>$ be the order on $\mathbb{A}^d$ defined by $(a_1, \ldots, a_d) > (b_1, \ldots, b_d) \iff \bigwedge_{i=1}^{d} a_i >_{\mathbb{A}} b_i$, restricted to $\mathbb{N} \times \mathbb{A}^{d-1}$. Then $(\mathcal{A}, >)$ is a well-founded monotone $\mathcal{F}$-algebra.

Theorem 4.24 ([61])**.** *Let $\mathcal{A}$ be an arctic interpretation of dimension d such that for every function symbol $f \in \mathcal{F}$, either $b \in \mathbb{N} \times \mathbb{A}^{d-1}$ or there exists some $1 \leqslant i \leqslant n$ such that $A_{i1,1} \in \mathbb{N}$, where $f_{\mathcal{A}}(x_1, \dots, x_n) = b +_{\mathbb{A}} \sum_{i=1}^{n} A_i *_{\mathbb{A}} x_i$ and n is the arity of f. Moreover, let $\geqslant$ and $>$ be the following orders on $\mathbb{A}^d$, restricted to $\mathbb{N} \times \mathbb{A}^{d-1}$:*

$$(a_1, \dots, a_d) \geqslant (b_1, \dots, b_d) \iff \bigwedge_{i=1}^{d} a_i \geqslant_{\mathbb{A}} b_i$$

$$(a_1, \dots, a_d) > (b_1, \dots, b_d) \iff \bigwedge_{i=1}^{d} a_i >_{\mathbb{A}} b_i$$

Then $(\mathcal{A}, \geqslant, >)$, is a weakly monotone $\mathcal{F}$-algebra.

If $\mathcal{A}$ is an arctic interpretation, we often only write $\mathcal{A}$ for $(\mathcal{A}, >)$ if the premises of Theorem 4.23 hold for $(\mathcal{A}, >)$; we call $\mathcal{A}$ a *strict arctic interpretation* in that case. Similarly, we often write $\mathcal{A}$ for $(\mathcal{A}, \geqslant, >)$ if the premises of Theorem 4.24 hold for $(\mathcal{A}, \geqslant, >)$; then we call $\mathcal{A}$ a *weak arctic interpretation*.

The following general upper bounds on derivation heights are easy to see:

Lemma 4.25. *Let $\mathcal{R}$ be a TRS and $\mathcal{A}$ a strict arctic interpretation such that $>_{\mathcal{A}}$ is compatible with $\mathcal{R}$. Then for all terms t and assignments α, $[\alpha]_{\mathcal{A}}(t) = (a_1, \dots, a_d)$ implies $\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant a_1$.*

Lemma 4.26. *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and $\mathcal{A}$ a weak arctic interpretation. Suppose that $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$ completely solves $(\mathcal{P}, \mathcal{R})$. Then for all terms t and assignments α, $[\alpha]_{\mathcal{A}}(t) = (a_1, \dots, a_d)$ implies $\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathcal{P}} / \to_{\mathcal{R}}) \leqslant a_1$.*

Essentially based on a variant of Lemma 4.25, the following complexity bound is shown in [61, Lemma 17]:

Theorem 4.27 ([61])**.** *For any strict arctic interpretation $\mathcal{A}$, the reduction order $>_{\mathcal{A}}$ induces linear complexity.*

Using Lemma 4.26 instead of Lemma 4.25, the last theorem is easily lifted to reduction pairs:

Theorem 4.28. *For any weak arctic interpretation $\mathcal{A}$, the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces linear complexity.*

4.2 Knuth-Bendix Orders

Knuth-Bendix orders (KBOs for short), originally defined in [60], but see also [8] or [102, Section 6.4.4], are the earliest of the classical reduction orders we present here. They share some ingredients both with interpretations (cf. Section 4.1) and recursive path orders (cf. Section 4.3). A KBO is defined by a *precedence* and a *weight function*.

Definition 4.29. A *precedence* is a proper order on function symbols. A *weight function* is a pair (w, w_0), where $w_0 \in \mathbb{R}^+$, and w is a mapping from $\mathcal{F}$ to $\mathbb{R}_0^+$. A weight function (w, w_0) is *admissible* with respect to the precedence $>$ if for every function symbol $f \in \mathcal{F}$, we have either $w(f) \geqslant w_0$, or f has arity 1, $w(f) = 0$, and $f > g$ for each $g \in \mathcal{F} \setminus \{f\}$. A weight function (w, w_0) is lifted to a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathbb{R}_0^+$ (also denoted as w) as follows:

$$w(t) = \begin{cases} w_0 & \text{if } t \in \mathcal{V} \\ w(f) + \sum_{i=1}^{n} w(t_i) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

Definition 4.30. Let $>$ be a precedence and (w, w_0) an admissible weight function. Then the *Knuth-Bendix order* based on $>$ and (w, w_0), denoted as $>_{\mathsf{KBO}(w,w_0)}$, is defined as follows: $s >_{\mathsf{KBO}(w,w_0)} t$ if $|s|_x \geqslant |t|_x$ for all $x \in \mathcal{V}\mathsf{ar}(s)$, and one of the following alternatives holds:

1. $w(s) > w(t)$
2. $s = f^n(t)$ for some $n > 0$
3. $w(s) = w(t)$ and $f > g$, if $s = f(s_1, \dots, s_n)$ and $t = g(t_1, \dots, t_m)$
4. $w(s) = w(t)$ and $(s_1, \dots, s_n) >^{\text{lex}}_{\mathsf{KBO}(w,w_0)} (t_1, \dots, t_n)$, if $s = f(s_1, \dots, s_n)$ and $t = f(t_1, \dots, t_n)$

As apparent from their definition, KBOs are both syntactic and semantic reduction orders. Most criteria in the recursive definition are purely syntactic criteria, while the weight function forms the semantic component of KBOs.

Theorem 4.31 ([60])**.** *For any precedence $>$ and weight function (w, w_0) that is admissible with respect to $>$, the resulting KBO $>_{\mathsf{KBO}(w,w_0)}$ is a reduction order.*

The general upper bound on the derivational complexity of TRSs whose termination can be proved by a KBO is due to Lepper.

Theorem 4.32 ([67])**.** *For any precedence $>$ and weight function (w, w_0) which is admissible with respect to $>$, the resulting KBO $>_{\mathsf{KBO}(w,w_0)}$ induces complexity $\mathsf{Ack}(\mathsf{O}(n), 0)$. This bound is essentially optimal.*

Proof Sketch (for the upper bound). Let $>_{\mathsf{KBO}(w,w_0)}$ be a KBO, and $\mathcal{R}$ a TRS which is compatible with $>_{\mathsf{KBO}(w,w_0)}$. Based on $>_{\mathsf{KBO}(w,w_0)}$ and $\mathcal{R}$, a mapping $\mathcal{I}\colon \mathcal{T}(\mathcal{F}) \to \mathbb{N}$ is defined which employs the fast growing functions. It is then shown that for all ground terms s and t, $s \to_{\mathcal{R}} t$ implies $\mathcal{I}(s) > \mathcal{I}(t)$, and hence for all ground terms t, $\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathcal{I}(t)$. Finally, it is shown that there exists a constant C (depending only on $\mathcal{R}$ and $>_{\mathsf{KBO}(w,w_0)}$) such that for all ground terms t, $\mathcal{I}(t) \leqslant \mathsf{Ack}(C \cdot n, 0)$. □

A weaker upper bound has previously been shown by Hofbauer [47]. For restricted classes of TRSs which terminate by Theorem 4.31, Hofbauer has shown that $\mathsf{dc}_{\mathcal{R}}(n)$ is contained in $2^{\mathsf{O}(n)}$:

Theorem 4.33 ([50, 47])**.** *If $\mathcal{F}$ contains no function symbol of arity greater than 1, then for any precedence $>$ and weight function (w, w_0) which is admissible with respect to $>$, the resulting KBO $>_{\mathsf{KBO}(w,w_0)}$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ induces complexity $2^{\mathsf{O}(n)}$. This bound is essentially optimal.*

Theorem 4.34 ([50, 47])**.** *Let $\mathcal{R}$ be a TRS, and $>_{\mathsf{KBO}(w,w_0)}$ a KBO which is compatible with $\mathcal{R}$. If $w(f) = 0$ implies $|l|_f \geqslant |r|_f$ for each rule $l \to r \in \mathcal{R}$, then $\mathsf{dc}_{\mathcal{R}}(n) \in 2^{\mathsf{O}(n)}$. This bound is essentially optimal.*

Finally, Moser [77] has extended Theorem 4.32 to infinite TRSs under some restrictions.

4.3 Recursive Path Orders

Recursive path orders are another classical set of reduction orders. While they operate on a purely syntactic basis, their definitions are reasonably simple, and it is rather easy to verify whether a given TRS is compatible with a recursive path order, they are still very powerful with respect to the derivational complexity of TRSs whose termination they can prove. In this thesis, we mention two specific instances of recursive path orders: *multiset path orders* (MPOs for short), and *lexicographic path orders* (LPOs for short).

Multiset path orders [20], originally just called *recursive path orders*, were the first kind of recursive path orders appearing in the literature. They are defined as follows:

Definition 4.35 ([20])**.** Let $>$ be a precedence. Then the *multiset path order* based on $>$, denoted as $>_{\mathsf{MPO}}$, is defined as follows: we have $s >_{\mathsf{MPO}} t$ if one of the following alternatives holds:

1. $s = f(s_1, \ldots, s_n)$ such that $s_i \gtrsim_{\mathsf{MPO}} t$ for some $1 \leqslant i \leqslant n$, where $\gtrsim_{\mathsf{MPO}}$ is $>_{\mathsf{MPO}} \cup \sim$
2. $s = f(s_1, \ldots, s_n)$ and $t = g(t_1, \ldots, t_m)$ such that $f > g$ and $s >_{\mathsf{MPO}} t_i$ for all $1 \leqslant i \leqslant m$
3. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ such that $\{\{s_1, \ldots, s_n\}\} >^{\mathrm{mul}}_{\mathsf{MPO}} \{\{t_1, \ldots, t_n\}\}$

Here $\sim$ denotes the *permutative equivalence* of terms: we have $s \sim t$ if either $s = t$, or $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, and there exists a permutation π of $\{1, \ldots, n\}$ such that $s_i \sim t_{\pi(i)}$ for all $1 \leqslant i \leqslant n$.

Theorem 4.36 ([20])**.** *For any precedence $>$, the MPO $>_{\mathsf{MPO}}$ based on $>$ is a reduction order.*

The following is a straightforward extension of Theorem 4.36:

Theorem 4.37. *For any precedence $>$, the corresponding pair $(\gtrsim_{\mathsf{MPO}}, >_{\mathsf{MPO}})$ is a reduction pair.*

Hofbauer has shown the complexity induced by MPOs:

Theorem 4.38 ([48])**.** *For any precedence $>$, the MPO $>_{\mathsf{MPO}}$ based on $>$ induces primitive recursive complexity. This bound is essentially optimal.*

Proof Sketch. Let $>_{\mathsf{MPO}}$ be some MPO, and $\mathcal{R}$ a TRS which is compatible with $>_{\mathsf{MPO}}$. Based on $>_{\mathsf{MPO}}$ and $\mathcal{R}$, a well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$ with the following properties is defined: the carrier of $\mathcal{A}$ is $\mathbb{N}$, $>_{\mathbb{N}}$ is the usual order on $\mathbb{N}$, for every function symbol $f \in \mathcal{F}$, the interpretation function $f_{\mathcal{A}}$ is primitive recursive, and $\mathcal{R}$ is compatible with $>_{\mathbb{N}\mathcal{A}}$. Hence, the upper complexity bound follows.

In order to see optimality (and, moreover, that MPOs characterise the set of primitive recursive functions), observe that every primitive recursive function f can be computed by a TRS compatible with a MPO, which is constructed as follows: define f using only initial functions, composition, and primitive recursion, and transform the resulting definition into rewrite rules in the natural way. □

An alternative proof of Theorem 4.38 has been given by Buchholz in [12, Section 3]. It starts out by proving that compatibility with an MPO implies termination of a given TRS. Since this termination proof is given using only a certain restricted fragment of Peano arithmetic (namely, Σ^0_1-IA), it then follows from classical proof-theoretic results (namely, [88]) that MPOs induce primitive recursive complexity.

Theorem 4.39. *For any precedence $>$, the reduction pair $(\gtrsim_{\mathsf{MPO}}, >_{\mathsf{MPO}})$ induces primitive recursive complexity. This bound is essentially optimal.*

Proof. This theorem is proved exactly in the same way as Theorem 4.38. The only crucial piece of information which is needed to transfer the proof of this theorem is that for all terms s and t, $s \sim t$ implies $[\alpha]_{\mathcal{A}}(s) = [\alpha]_{\mathcal{A}}(t)$ for all assignments α, which is [48, Lemma 5.2(v)]. Optimality follows from Theorem 4.38, since for every TRS $\mathcal{R}$ which is compatible with some MPO $>_{\mathsf{MPO}}$, the DP problem $(\mathcal{R}, \emptyset)$ is completely solved by $\Phi^{\mathsf{RP}}_{(\gtrsim_{\mathsf{MPO}}, >_{\mathsf{MPO}})}$. □

The second kind of recursive path orders we consider are *lexicographic path orders* [58]:

Definition 4.40 ([58])**.** Let $>$ be a precedence. Then the *lexicographic path order* based on $>$, denoted as $>_{\mathsf{LPO}}$, is defined as follows: $s >_{\mathsf{LPO}} t$ if one of the following alternatives holds:

1. $s = f(s_1, \ldots, s_n)$ such that $s_i \geqslant_{\mathsf{LPO}} t$ for some $1 \leqslant i \leqslant n$, where $\geqslant_{\mathsf{LPO}}$ is the reflexive closure of $>_{\mathsf{LPO}}$
2. $s = f(s_1, \ldots, s_n)$ and $t = g(t_1, \ldots, t_m)$ such that $f > g$ and $s >_{\mathsf{LPO}} t_i$ for all $1 \leqslant i \leqslant m$
3. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ such that $s >_{\mathsf{LPO}} t_i$ for all $1 \leqslant i \leqslant n$, and $(s_1, \ldots, s_n) >^{\mathrm{lex}}_{\mathsf{LPO}} (t_1, \ldots, t_n)$

Theorem 4.41 ([58])**.** *For any precedence $>$, the LPO $>_{\mathsf{LPO}}$ based on $>$ is a reduction order.*

Weiermann has shown the complexity induced by LPOs:

Theorem 4.42 ([113])**.** *For any precedence $>$, the LPO $>_{\mathsf{LPO}}$ based on $>$ induces multiply recursive complexity. This bound is essentially optimal.*

Proof Sketch. Let $>_{\mathsf{LPO}}$ be some LPO, and $\mathcal{R}$ a TRS which is compatible with $>_{\mathsf{LPO}}$. Based on $>_{\mathsf{LPO}}$ and $\mathcal{R}$, a well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$ with the following properties is defined: the carrier of $\mathcal{A}$ is $\mathbb{N}$, $>_{\mathbb{N}}$ is the usual order on $\mathbb{N}$, for every function symbol $f \in \mathcal{F}$, the interpretation function $f_{\mathcal{A}}$ is multiply recursive (the interpretation employs the fast growing functions F_α for $\alpha < \omega^\omega$, which are known to be multiply recursive functions [89]), and $\mathcal{R}$ is compatible with $>_{\mathbb{N}\mathcal{A}}$. Hence, the upper bound follows.

Optimality of this upper bound follows from the fact that LPOs characterise the set of multiply recursive functions, which can be seen as follows: by definition, any multiply recursive function f can be defined using only initial functions, composition, primitive recursion, and k-ary Ackermann functions. Any function defined this way can be immediately transformed into a TRS which is compatible with a LPO. $\square$

An alternative proof of Theorem 4.42 has been given by Buchholz in [12, Sections 1-2]. It proceeds in the same spirit as the alternative proof of Theorem 4.38, but uses a different fragment of Peano arithmetic (namely, Π^0_2-IA).

Finally, Arai [1, Section 8] has given better (but, naturally, still multiply recursive) complexity bounds, depending on the signature of the TRS under consideration. Although the general upper bound given in [113] was indeed shown to be optimal, the specific bounds for fixed signatures could be improved by [1, Section 8].

4.4 Bounds

Finally, we describe (*match*- and other) *bounds*, a family of semantic termination proof techniques which are not based on well-founded monotone algebras. Here the basic idea is to annotate the function symbols with natural numbers in such a way that the annotations are strictly increasing under rewriting, and the increase of annotations under rewrite sequences is limited by a fixed bound (and hence, a finite and easily proved terminating annotated TRS suffices to faithfully simulate the original TRS). The technique has been initially proposed for string rewriting [32], and later been extended to left-linear TRSs [33]. Further improvements which allow the technique to be used for non-left-linear TRSs and DP problems have been made in [63, 62].

Definition 4.43 ([32])**.** Let $\mathcal{F}$ be a signature. Then $\mathcal{F}^{\mathrm{b}}$ is the infinite signature $\{f_n \mid f \in \mathcal{F} \wedge n \in \mathbb{N}\}$, where for each $f \in \mathcal{F}$ and $n \in \mathbb{N}$, f_n is a fresh function symbol with the same arity as f. We have the mappings $\mathrm{base}\colon \mathcal{F}^{\mathrm{b}} \to \mathcal{F}$, $\mathrm{height}\colon \mathcal{F}^{\mathrm{b}} \to \mathbb{N}$, and $\mathrm{lift}_n\colon \mathcal{F} \to \mathcal{F}^{\mathrm{b}}$ for each $n \in \mathbb{N}$, which are defined by $\mathrm{base}(f_n) = f$, $\mathrm{height}(f_n) = n$, and $\mathrm{lift}_n(f) = f_n$. These mappings are extended to terms by applying them pointwise to all function symbols in the given term.

Using these correspondences between annotated and unannotated terms, a given TRS $\mathcal{R}$ over unannotated terms is lifted to a TRS $\mathcal{R}'$ over annotated terms; $\mathcal{R}'$ is said to be a *cover* of $\mathcal{R}$ in that case. The definition given here is slightly more general than the definitions found in the literature.

Definition 4.44 ([104])**.** Let $\mathcal{R}$ be a left-linear TRS over $\mathcal{F}$ and $\mathcal{V}$. Then an infinite TRS $\mathcal{R}'$ over $\mathcal{F}^{\mathrm{b}}$ and $\mathcal{V}$ is called a *cover* of $\mathcal{R}$ if for all rules $l \to r \in \mathcal{R}$ and all terms l' such that $\mathrm{base}(l') = l$, there exists a term r' such that $\mathrm{base}(r') = r$ and $l' \to r' \in \mathcal{R}'$.

An *enrichment* is a mapping e from terms l' and rewrite rules $l \to r$ such that $\mathrm{base}(l') = l$ to terms r' such that $\mathrm{base}(r') = r$. An enrichment e is extended to a cover by setting $e(\mathcal{R}) = \{l' \to r' \mid l \to r \in \mathcal{R} \wedge \mathrm{base}(l') = l \wedge e(l', l \to r) = r'\}$.

We mention the following two specific covers/enrichments which appear in the literature:

Definition 4.45 ([33])**.** The mappings top and match from pairs of terms to sets of positions are defined as follows:

$$\mathrm{top}(l, r) = \{\epsilon\} \qquad \mathrm{match}(l, r) = \mathcal{P}\mathsf{os}_{\mathcal{F}}(l)$$

Each of these mappings $e \in \{\mathrm{top}, \mathrm{match}\}$ is then lifted to an enrichment (and hence further to a cover) by setting $e(l', l \to r) = \mathrm{lift}_c(r)$, where $c = 1 + \min\{\mathrm{height}(\mathsf{rt}(l'|_p)) \mid p \in e(l, r)\}$.

These covers have a property which eases termination proofs:

Lemma 4.46 ([33])**.** *Let $\mathcal{R}$ be a left-linear TRS. Then every finite subset of* top$(\mathcal{R})$ *is terminating. Moreover, if $\mathcal{R}$ is right-linear, then every finite subset of* match$(\mathcal{R})$ *is terminating.*

Since, by the definition of covers, every derivation in $\mathcal{R}$ starting from a term t can be simulated by a derivation in $e(\mathcal{R})$ for any $e \in \{\mathrm{top}, \mathrm{match}\}$ starting from $\mathrm{lift}_0(t)$, this gives rise to the following termination criterion:

Definition 4.47 ([33])**.** Let $\mathcal{R}$ be a left-linear TRS and e a mapping such that $e(\mathcal{R})$ is a cover of $\mathcal{R}$. Suppose that there exists some $c \in \mathbb{N}$ such that for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and u with $\mathrm{lift}_0(t) \to_{\mathcal{R}}^* u$, we have $\mathcal{F}\mathsf{un}(u) \subseteq \{f_n \in \mathcal{F}^{\mathrm{b}} \mid n \leqslant c\}$. Then $\mathcal{R}$ is said to be *e-bounded by c*.

Theorem 4.48 ([33])**.** *Any left-linear and* top*-bounded, or linear and* match*-bounded TRS is terminating.*

All of these concepts have been extended to DP problems in [63, 62]. We start with an extension of Definition 4.44 to DP problems.

Definition 4.49. Let $\mathcal{P}$ and $\mathcal{R}$ be left-linear TRSs, and $\mathcal{P}' \subseteq \mathcal{P}$. Then an infinite TRS $\mathcal{R}'$ is called a *DP-cover* of the triple $(\mathcal{P}, \mathcal{P}', \mathcal{R})$ if for all rules $l \to r \in \mathcal{P} \cup \mathcal{R}$ and all terms l' such that $\mathrm{base}(l') = l$, there exists a term r' such that $\mathrm{base}(r') = r$ and $l' \to r' \in \mathcal{R}'$.

Specifically, covers created by enrichments such as top and match can be extended to DP-covers as follows:

Definition 4.50 ([63, 62])**.** Let e be an enrichment such that for all rules $l \to r \in \mathcal{R}$ and terms l' with $\mathrm{base}(l') = l$ we have $e(l', l \to r) = \mathrm{lift}_c(r)$ for some $c \in \mathbb{N}$. Then e-DP is the enrichment defined by $e\text{-DP}(l', l \to r) = \mathrm{lift}_{c'}(r)$, where $c' = \min\{c, \mathrm{height}(\mathsf{rt}(l'))\}$ and $e(l', l \to r) = \mathrm{lift}_c(r)$.

An enrichment e and the associated enrichment e-DP are lifted to a DP-cover by setting $e\text{-DP}(\mathcal{P}, \mathcal{P}', \mathcal{R}) = \{l' \to r' \mid l \to r \in (\mathcal{P} \setminus \mathcal{P}') \cup \mathcal{R} \wedge \mathrm{base}(l') = l \wedge e\text{-DP}(l', l \to r) = r'\} \cup \{l' \to r' \mid l \to r \in \mathcal{P}' \wedge \mathrm{base}(l') = l \wedge e(l', l \to r) = r'\}$.

Definition 4.47 can be extended to DP-covers in a straightforward way:

Definition 4.51. Let $\mathcal{P}$ and $\mathcal{R}$ be left-linear TRSs, $\mathcal{P}' \subseteq \mathcal{P}$, and e a mapping such that $e(\mathcal{P}, \mathcal{P}', \mathcal{R})$ is a DP-cover of $(\mathcal{P}, \mathcal{P}', \mathcal{R})$. Suppose that there exists some $c \in \mathbb{N}$ such that for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and u with $\mathrm{lift}_0(t) \to^*_{\mathcal{P} \cup \mathcal{R}} u$, we have $\mathcal{F}\mathsf{un}(u) \subseteq \{f_n \in \mathcal{F}^{\mathrm{b}} \mid n \leqslant c\}$. Then $(\mathcal{P}, \mathcal{P}', \mathcal{R})$ is said to be *e-bounded by c.*

Definition 4.52 ([63, 62])**.** Let $\mathcal{P}'$ be a set of rewrite rules. Then the top-*bounds processor* for $\mathcal{P}'$ is the following DP processor:

$$\Phi^{\mathrm{top}}_{\mathcal{P}'}((\mathcal{P}, \mathcal{R})) = \begin{cases} \{(\mathcal{P} \setminus \mathcal{P}', \mathcal{R})\} & \text{if } \mathcal{P}' \subseteq \mathcal{P},\ \mathcal{P} \cup \mathcal{R} \text{ is left-linear, and} \\ & \quad (\mathcal{P}, \mathcal{P}', \mathcal{R}) \text{ is top-DP-bounded} \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

Moreover, the match-*bounds processor* for $\mathcal{P}'$ is the following DP processor:

$$\Phi^{\mathrm{match}}_{\mathcal{P}'}((\mathcal{P}, \mathcal{R})) = \begin{cases} \{(\mathcal{P} \setminus \mathcal{P}', \mathcal{R})\} & \text{if } \mathcal{P}' \subseteq \mathcal{P},\ \mathcal{P} \cup \mathcal{R} \text{ is linear, and} \\ & \quad (\mathcal{P}, \mathcal{P}', \mathcal{R}) \text{ is match-DP-bounded} \\ \{(\mathcal{P}, \mathcal{R})\} & \text{otherwise} \end{cases}$$

Theorem 4.53 ([63, 62])**.** *For any TRS $\mathcal{P}'$, the DP processors $\Phi^{\mathrm{top}}_{\mathcal{P}'}$ and $\Phi^{\mathrm{match}}_{\mathcal{P}'}$ are sound.*

The following general upper bounds on derivation heights follow easily from Definitions 4.44 and 4.49:

Lemma 4.54. *Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{R}'$ a cover of $\mathcal{R}$, and t, t' be terms such that* $\mathrm{base}(t') = t$. *Then* $\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathsf{dh}(t', \to_{\mathcal{R}'})$.

Lemma 4.55. *Let $\mathcal{P}$ and $\mathcal{R}$ be left-linear TRSs, $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{R}'$ a DP-cover of $(\mathcal{P}, \mathcal{P}', \mathcal{R})$, and t, t' terms such that* $\mathrm{base}(t') = t$. *Then* $\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathcal{P}'} / \to_{(\mathcal{P} \setminus \mathcal{P}') \cup \mathcal{R}}) \leqslant \mathsf{dh}(t, \to_{\mathcal{P} \cup \mathcal{R}}) \leqslant \mathsf{dh}(t', \to_{\mathcal{R}'})$.

Termination proofs by Theorem 4.48 induce the following complexity bounds:

Theorem 4.56. *Let $\mathcal{R}$ be a left-linear and* top*-bounded TRS. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded by an exponential function.*

Proof. Suppose that $\mathcal{R}$ is left-linear and top-bounded by c. Let $\mathcal{R}'$ be the restriction of $\mathrm{top}(\mathcal{R})$ to rules containing function symbols of height at most c, and note that $\mathcal{R}'$ is finite. Since $\mathcal{R}$ is match-bounded by c, for every term t, the equality $\mathsf{dh}(\mathrm{lift}_0(t), \rightarrow_{\mathrm{top}(\mathcal{R})}) = \mathsf{dh}(\mathrm{lift}_0(t), \rightarrow_{\mathcal{R}'})$ holds, and hence by Lemma 4.54, $\mathsf{dh}(t, \rightarrow_{\mathcal{R}}) \leqslant \mathsf{dh}(\mathrm{lift}_0(t), \rightarrow_{\mathcal{R}'})$. There exists a linear interpretation $\mathcal{A}$ such that $\mathcal{R}'$ is compatible with $>_{\mathcal{A}}$, so by Theorem 4.11, $\mathsf{dc}_{\mathcal{R}'}$, and thus $\mathsf{dc}_{\mathcal{R}}$, is bounded by an exponential function.

The linear interpretation $\mathcal{A}$ is given inductively as follows. For each $f \in \mathcal{F}$ of arity n, set

$$f_{c\mathcal{A}}(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \ .$$

Let a be the maximum arity of any function symbol in $\mathcal{F}$, g a function symbol in $\mathcal{F}$ of arity a, and $b = 1 + \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$. Define $G_i^0(x) = x$ and $G_i^{k+1}(x) = g_i(G_i^k(x), \ldots, G_i^k(x))$. Then for each $f \in \mathcal{F}$ of arity n, we set

$$f_{i-1\mathcal{A}}(x_1, \ldots, x_n) = 1 + \sum_{j=1}^{n} [\alpha_j]_{\mathcal{A}}(G_i^b(x)) \ ,$$

where $\alpha_j(x) = x_j$.

It is easy to check by induction on $c - i$ that for every function symbol $f_i \in \mathcal{F}^{\mathrm{b}}$, the interpretation function $f_{i\mathcal{A}}$ is a linear function, and hence $\mathcal{A}$ is a linear interpretation: it is obvious that $f_{c\mathcal{A}}$ is linear for all $f \in \mathcal{F}$. Now we consider $f_{i-1\mathcal{A}}$: by induction hypothesis, $g_{i\mathcal{A}}$ is a linear function. Since the set of linear functions is closed under composition and addition, it follows that $f_{i-1\mathcal{A}}$ is a linear function, as well.

Finally, we verify that $\mathrm{top}(\mathcal{R})$ is compatible with $>_{\mathcal{A}}$. Let $f_i(l_1, \ldots, l_n) \to r \in \mathrm{top}(\mathcal{R})$. Moreover, obviously $\mathcal{V}\mathsf{ar}(r) \subseteq \bigcup_{j=1}^{n} \mathcal{V}\mathsf{ar}(l_j)$, hence by strict monotonicity of all interpretation functions, we have $\max\{[\alpha]_{\mathcal{A}}(l_1), \ldots, [\alpha]_{\mathcal{A}}(l_n)\} \geqslant \alpha(v)$ for all $v \in \mathcal{V}\mathsf{ar}(r)$. By straightforward induction on d, for any term t of depth at most d containing only function symbols of height between $i+1$ and c with $\mathcal{V}\mathsf{ar}(t) \subseteq \mathcal{V}\mathsf{ar}(r)$, we have $[\alpha]_{\mathcal{A}}(t) \leqslant [\alpha']_{\mathcal{A}}(G_{i+1}^{d+1}(x))$, where $\alpha'(x) = \max\{[\alpha]_{\mathcal{A}}(l_j) \mid 1 \leqslant j \leqslant n\}$. By definition, the height of each function symbol in r is at least $i+1$. Thus, $f_{i\mathcal{A}}([\alpha]_{\mathcal{A}}(l_1), \ldots, [\alpha]_{\mathcal{A}}(x_n)) > [\alpha']_{\mathcal{A}}(G_{i+1}^{\mathsf{dp}(r)+1}(x)) \geqslant [\alpha]_{\mathcal{A}}(r)$. □

Theorem 4.57 ([33]). *Let $\mathcal{R}$ be a linear and* match-*bounded TRS. Then* $\mathsf{dc}_{\mathcal{R}}$ *is bounded by a linear function.*

Proof Sketch. Suppose that $\mathcal{R}$ is linear and match-bounded by c. The following is entailed by the premises in a straightforward manner: $s \rightarrow_{\mathrm{match}(\mathcal{R})} t$ implies that the multiset of heights of s is strictly greater than the multiset of heights of t with respect to $>_{\mathsf{rev}}^{\mathrm{mul}}$, where $>_{\mathsf{rev}}$ is the reverse of the usual strict order on $\mathbb{N}$ (note that $>_{\mathsf{rev}}$ restricted to $\{0, \ldots, c\}$ is obviously well-founded). From this multiset decrease, it can be shown that the derivational complexity of $\mathrm{match}(\mathcal{R})$ restricted to heights at most c, and thus also $\mathsf{dc}_{\mathcal{R}}$, is bounded by a linear function. □

Theorem 4.58. *Let $(\mathcal{P},\mathcal{R})$ be a DP problem, and $\mathcal{P}'$ a set of rewrite rules. Let $\{(\mathcal{Q},\mathcal{R})\} = \Phi^{\text{top}}_{\mathcal{P}'}((\mathcal{P},\mathcal{R}))$. Then $\mathcal{Q} \subseteq \mathcal{P}$. Moreover, $\mathsf{DPc}_{\mathcal{P}\setminus\mathcal{Q},\mathcal{Q}\cup\mathcal{R}}$ is bounded by a constant function.*

Proof. The first part of the theorem follows directly from the definitions, so we only prove the second part. If $\mathcal{Q} = \mathcal{P}$, then $\mathsf{DPc}_{\mathcal{P}\setminus\mathcal{Q},\mathcal{Q}\cup\mathcal{R}}$ is the constant zero function. Otherwise, $\mathcal{Q} = \mathcal{P} \setminus \mathcal{P}'$, and we have that $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{P} \cup \mathcal{R}$ is left-linear, and $(\mathcal{P},\mathcal{P}',\mathcal{R})$ is top-DP-bounded by c for some $c \in \mathbb{N}$. Let $\mathcal{R}' =$ top-DP$(\mathcal{P},\mathcal{P}',\mathcal{R})$. Whenever $s \rightarrow_{\mathcal{Q}\cup\mathcal{R}} t$, then for any s' such that $\text{base}(s') = s$, there exists some t' such that $\text{base}(t') = t$ and $s' \rightarrow_{\mathcal{R}'} t'$, and by definition of top-DP, we have $\text{height}(\mathsf{rt}(s')) = \text{height}(\mathsf{rt}(t'))$. On the other hand, suppose that $s \xrightarrow{\epsilon}_{\mathcal{P}'} t$. Then for any s' such that $\text{base}(s') = s$, there exists some t' such that $\text{base}(t') = t$ and $s' \xrightarrow{\epsilon}_{\mathcal{R}'} t'$, and by definition of top-DP, we have $\text{height}(\mathsf{rt}(s')) + 1 = \text{height}(\mathsf{rt}(t'))$. Since $(\mathcal{P},\mathcal{P}',\mathcal{R})$ is top-DP-bounded by c, we have $\mathsf{dh}(\text{lift}_0(u), \rightarrow_{\mathcal{R}'}) \leqslant c$ for all $u \in \mathcal{T}(\mathcal{F},\mathcal{V})$. Thus by Lemma 4.55, $\mathsf{dh}(u, \xrightarrow{\epsilon}_{\mathcal{P}\setminus\mathcal{Q}}/\rightarrow_{\mathcal{Q}\cup\mathcal{R}}) \leqslant c$, which is what we wanted to show. □

However, we cannot give any complexity bounds for match-DP-bounded triples. The reason for this is the following: let $(\mathcal{P},\mathcal{R})$ be a DP problem and $\mathcal{P}' \subseteq \mathcal{P}$ such that $\Phi^{\text{match}}_{\mathcal{P}'}(\mathcal{P},\mathcal{R}) = \{\mathcal{P}\setminus\mathcal{P}',\mathcal{R}\}$. Suppose even that $\mathcal{P}' = \mathcal{P}$. Then by soundness of $\Phi^{\text{match}}_{\mathcal{P}}$, there exists no minimal infinite chain with respect to $(\mathcal{P},\mathcal{R})$. However, this does not necessarily entail that $\xrightarrow{\epsilon}_{\mathcal{P}}/\rightarrow_{\mathcal{R}}$ is well-founded. In particular, this scenario of finiteness, but nontermination (and hence absence of any complexity bound) may occur for match-DP-bounded triples $(\mathcal{P},\mathcal{P},\mathcal{R})$, as illustrated by the next example:

Example 4.59. Let $\mathcal{P} = \{\mathsf{f}(\mathsf{s}(x)) \rightarrow \mathsf{f}(x)\}$, and $\mathcal{R} = \{\mathsf{s}(x) \rightarrow \mathsf{s}(\mathsf{s}(x))\}$. It is easy to check that $(\mathcal{P},\mathcal{P},\mathcal{R})$ is match-DP-bounded by 1, and hence, by Theorem 4.52, the DP problem $(\mathcal{P},\mathcal{R})$ is finite. However, the corresponding relation $\xrightarrow{\epsilon}_{\mathcal{P}}/\rightarrow_{\mathcal{R}}$ is not well-founded, as witnessed by the infinite derivation created by infinite repetition of the rewrite steps $\mathsf{f}(\mathsf{s}(x)) \rightarrow_{\mathcal{R}} \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) \xrightarrow{\epsilon}_{\mathcal{P}} \mathsf{f}(\mathsf{s}(x))$. Note that this derivation does not give rise to a minimal infinite chain, since $\mathsf{f}(\mathsf{s}(x))$ is not a normal form with respect to $\mathcal{R}$.

Even if we restrict to DP problems $(\mathcal{P},\mathcal{R})$ such that $\mathcal{P} \cup \mathcal{R}$ is linear and terminating, and $(\mathcal{P},\mathcal{P},\mathcal{R})$ is match-DP-bounded, we cannot give any upper bound on $\mathsf{DPc}_{\mathcal{P},\mathcal{R}}$. Let $\mathcal{P} = \{\mathsf{f}(\mathsf{s}(x)) \rightarrow \mathsf{f}(x)\}$ again. Let g be any computable function. By Turing-completeness of string rewriting, there exists a terminating SRS $\mathcal{S}$ such that $\mathsf{r}^n(x) \rightarrow^k_{\mathcal{S}} \mathsf{s}^{g(n)}(x)$ with $k \geqslant g(n)$ for all $n \in \mathbb{N}$, and f is not contained in the signature of $\mathcal{S}$. It is easy to check that $(\mathcal{P},\mathcal{P},\mathcal{S})$ is match-DP-bounded by 1, and that $\mathcal{P} \cup \mathcal{S}$ is terminating. However, $\mathsf{DPc}_{\mathcal{P},\mathcal{S}}$ grows at least as fast as g, which is allowed to be any computable function.

Chapter 5

Interpretation Methods for Derivational Complexity Analysis

The computing scientist's main challenge is not to get confused by the complexities of his own making.

Edsger Wybe Dijkstra

5.1 Introduction

In the last chapter, we have presented several well-known direct termination proof techniques. Alongside, we have listed the upper bounds on the derivational complexity of TRSs whose termination can be proved by those techniques.

The proof techniques specified in the last chapter are not at all specific to complexity analysis: their main use lies in proving termination of TRSs. Virtually all of them are implemented in leading modern automatic termination provers such as AProVE [34] and $\mathsf{T_TT_2}$ [64]. In contrast, in this chapter, we mention techniques which are specific to derivational complexity analysis.

Consider polynomial interpretations: by Theorem 4.10, the reduction order created by a polynomial interpretation induces double exponential complexity. Moreover, by Theorem 4.11, the reduction order created by a linear interpretation induces exponential complexity. Now consider the following example:

Example 5.1 ([49]). Let $\mathcal{R}_{\mathsf{Assoc}}$ be the TRS given by the following single rewrite rule over the signature containing the binary function symbol $\circ$ and the constant function symbol c:

$$(x \circ y) \circ z \to x \circ (y \circ z)$$

Let $\mathcal{A}$ be the polynomial interpretation given through the interpretation functions $\circ_{\mathcal{A}}(x, y) = 2x + y + 1$ and $\mathsf{c}_{\mathcal{A}} = 0$. It is easy to check that $\mathcal{R}_{\mathsf{Assoc}}$ is compatible with the resulting reduction order $>_{\mathcal{A}}$. By Theorem 4.11, $\mathsf{dc}_{\mathcal{R}_{\mathsf{Assoc}}}$ is bounded by an exponential function.

The upper complexity bound given in Example 5.1 is not optimal: the derivational complexity of $\mathcal{R}_{\mathsf{Assoc}}$ is bounded by a polynomial of degree 2. This overestimation is typical for polynomial interpretations: by Theorem 4.12, strongly linear interpretations induce linear complexity. However, even the slightest relaxation in the definition of strongly linear interpretations, such as allowing a single coefficient greater than 1 in the interpretation functions (as in Example 5.1) already makes it possible to prove termination of TRSs whose derivational complexity is exponential. For instance, the TRS given by the single rule $(x \circ y) \circ z \to x \circ x$ is compatible with the reduction order based on the linear interpretation given in Example 5.1. However, the TRS is duplicating, hence its derivational complexity is at least exponential, as witnessed by the family of starting terms defined by $t_0 = \mathsf{c}$ and $t_{n+1} = (t_n \circ \mathsf{c}) \circ \mathsf{c}$, and the derivation $t_{n+1} \to t_n \circ t_n \to (t_{n-1} \circ t_{n-1}) \circ t_n \to (t_{n-1} \circ t_{n-1}) \circ (t_{n-1} \circ t_{n-1}) \to \dots$. So, using polynomial interpretations, it seems impossible to infer any nonlinear polynomial upper bounds on the derivational complexity of TRSs. Even worse, none of the theorems about complexity bounds induced by termination proof techniques presented in Chapter 4 yields any nonlinear, but polynomial upper bounds.

In 2001, Hofbauer introduced *context dependent interpretations* as a remedy, cf. [49]. These interpretations extend polynomial interpretations by introducing an additional parameter. The parameter changes in the course of evaluating a term, which makes the interpretation dependent on the context. Concerning Example 5.1, a context dependent interpretation can be found which proves termination of $\mathcal{R}_{\mathsf{Assoc}}$ and gives an optimal upper bound on its derivational complexity, compare [49]. In [49], several other examples were presented where context dependent interpretations allowed to infer tighter complexity bounds than polynomial interpretations did. However, all context dependent interpretations given in [49], including the corresponding monotonicity and compatibility proofs, were hand-crafted, and automation was left as an open problem.

A technique to *automatically* search for context dependent interpretations was described by Moser and the author of this thesis in [79, 96]. This was achieved by delineating two subclasses of context dependent interpretations that made automation possible. With that technique, most of the manually inferred complexity bounds shown in [49] could be reproduced automatically. However the proof of the quadratic upper bound on $\mathsf{dc}_{\mathcal{R}_{\mathsf{Assoc}}}$ could not be automated this way. In this chapter, we give an overview of context dependent interpretations and the subclasses used for their automation. We outline the technical reasons why the previous automation failed on Example 5.1, and give an extension which remedies that problem.

Beside context dependent interpretations, we also investigate matrix interpretations in this chapter. By Theorem 4.19, the reduction order created from a matrix interpretation generally induces exponential complexity. However, for certain restricted shapes of matrix interpretations, the corresponding reduction orders induce polynomial complexity bounds. We specifically focus on *triangular matrix interpretation*, a simple syntactic such restriction of matrix interpretations. Reduction orders based on triangular matrix interpretations induce polynomial complexity, where the degree of the polynomial depends on the dimension of the interpretation.

We proceed to identify a subclass of two-dimensional matrix interpretations which corresponds to the class of context dependent interpretations we describe.

This correspondence is understood to be with respect to orientability: for any context dependent interpretation $\mathcal{C}$ from this class whose reduction order is compatible with a TRS $\mathcal{R}$ there exists a matrix interpretation $\mathcal{A}$ from the identified subclass such that $>_{\mathcal{A}}$ is compatible with $\mathcal{R}$, and vice versa. This theoretical result is interesting in its own right, as it links two different termination proof techniques which were previously conceived as incomparable.

Finally, we give an overview of another method for restricting matrix interpretations for automated polynomial derivational complexity analysis from the recent literature [86, 110, 76]. This method is based on weighted automata theory and linear algebra; more precisely, it is the synthesis of two independently developed methods, one of which was built upon weighted automata [110], and the other was based on linear algebra [86]. It is complete in the sense that it characterises the set of matrix interpretations $\mathcal{A}$ such that the entries of $[\alpha]_{\mathcal{A}}(t)$ grow polynomially in $|t|$, and hence the polynomial growth in $|t|$ of $[\alpha]_{\mathcal{A}}(t)$ follows by Lemma 4.17.

5.2 Triangular Matrix Interpretations

We now introduce a specific form of matrix interpretations called *triangular matrix interpretations*. The reduction order based on a triangular matrix interpretation induces polynomial complexity. This contrasts with general matrix interpretations, which can prove termination of TRSs whose derivational complexity is exponential. Hence the introduced restriction defines a strict subclass of those TRSs that can be shown terminating by a matrix interpretation.

Definition 5.2. An *upper triangular matrix* of dimension d is a matrix M in $\mathbb{N}^{d\times d}$ such that for all $d \geqslant i > j \geqslant 1$, we have $M_{i,j} = 0$, and for all $d \geqslant i \geqslant 1$, we have $M_{i,i} \leqslant 1$. A *triangular matrix interpretation* (*TMI* for short) of dimension d is a matrix interpretation $\mathcal{A}$ of dimension d such that for each interpretation function $f_{\mathcal{A}}$, all $d \times d$-matrices in $f_{\mathcal{A}}$ are upper triangular matrices.

Note that the main diagonal entries of any upper triangular matrix are from the set $\{0, 1\}$.

Example 5.3 (continued from Example 5.1)**.** We define a triangular matrix interpretation $\mathcal{B}$, as follows:

$$\circ_{\mathcal{B}}(\vec{x}, \vec{y}) = \begin{pmatrix}1 & 1\\ 0 & 1\end{pmatrix} \cdot \vec{x} + \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \cdot \vec{y} + \begin{pmatrix}0\\ 1\end{pmatrix} \qquad \mathsf{c}_{\mathcal{B}} = \begin{pmatrix}0\\ 0\end{pmatrix}$$

It is easy to check that $\mathcal{B}$ is compatible with $\mathcal{R}_{\mathsf{Assoc}}$.

Lemma 5.4. *Let M be an upper triangular matrix of dimension d and $n \in \mathbb{N}$. Then all entries of M^n are polynomially bounded in n. More precisely, if $i > j$ then $(M^n)_{i,j} = 0$, otherwise $(M^n)_{i,j} \leqslant (j-i)! \cdot (a \cdot n)^{j-i}$, where $a = \max\{M_{i,j} \mid 1 \leqslant i, j \leqslant d\}$.*

Proof. The case $i > j$ is easy to see. In the other case, we have $j \geqslant i$. Further, the lemma is easy to see for $n \leqslant 1$. So we only prove the other case, where

$n \geqslant 2$. For $j \geqslant i$ and $n \geqslant 2$, we prove the lemma by induction on $j - i$. If $j - i = 0$, then $(M^n)_{i,j} = (M_{i,j})^n \leqslant 1$. If $j > i$, then by induction hypothesis, for every $i < k < j$, we have $(M^n)_{i,k} \leqslant (k-i)! \cdot (a \cdot n)^{k-i}$ and $(M^n)_{k,j} \leqslant (j-k)! \cdot (a \cdot n)^{j-k}$. For every $1 \leqslant l \leqslant n$, we have $M^l = M^{l-1} \cdot M$ and therefore $(M^l)_{i,j} \leqslant (M^{l-1})_{i,j} + \sum_{k=i}^{j-1} (M^{l-1})_{i,k} \cdot M_{k,j}$. It follows by straightforward induction on n that $(M^n)_{i,j} \leqslant M_{i,j} + \sum_{m=1}^{n-1} \sum_{k=i}^{j-1} (M^m)_{i,k} \cdot M_{k,j}$. We have $(M^n)_{i,j} \leqslant a + \sum_{m=1}^{n-1} \sum_{k=i}^{j-1} (M^m)_{i,k} \cdot a$. Furthermore, by induction hypothesis, we have $(M^m)_{i,k} \leqslant (k-i)! \cdot (a \cdot m)^{k-i}$ for every $i \leqslant k < j$ and $m \in \mathbb{N}$, hence

$$\begin{aligned}
(M^n)_{i,j} &\leqslant a + \sum_{m=0}^{n-1} \sum_{k=i}^{j-1} (k-i)! \cdot (a \cdot m)^{k-i} \cdot a \\
&\leqslant a + \sum_{m=0}^{n-1} (j-i) \cdot (j-i-1)! \cdot (a \cdot m)^{j-i-1} \cdot a \\
&= a + (j-i)! \cdot a^{j-i} \cdot \sum_{m=0}^{n-1} m^{j-i-1} \\
&\leqslant a + (j-i)! \cdot a^{j-i} \cdot (n-1)^{j-i} \\
&\leqslant (j-i)! \cdot a^{j-i} \cdot n^{j-i-1} + (j-i)! \cdot a^{j-i} \cdot n^{j-i-1} \cdot (n-1) \\
&= (j-i)! \cdot a^{j-i} \cdot n^{j-i} \ ,
\end{aligned}$$

which is what we wanted to show. □

Due to Lemma 5.4, for any finite set $\mathcal{M} \subseteq \mathbb{N}^{d \times d}$ of upper triangular matrices, there is a polynomial p of degree $d-1$ such that for each sequence $M_1, \ldots, M_n \in \mathcal{M}$, and for each i, j, it holds that $(M_1 \cdot \ldots \cdot M_n)_{i,j} \leqslant p(n)$. Such products occur when computing values of matrix interpretations on (ground) terms: for example, let $\mathcal{A}$ denote a matrix interpretation of dimension d with interpretation functions of the shape

$$\mathsf{f}_{\mathcal{A}}(\vec{x}, \vec{y}) = F_1 \cdot \vec{x} + F_2 \cdot \vec{y} + \vec{f} \qquad \mathsf{g}_{\mathcal{A}}(\vec{x}, \vec{y}) = G_1 \cdot \vec{x} + G_2 \cdot \vec{y} + \vec{g}$$
$$\mathsf{a}_{\mathcal{A}} = \vec{a} \qquad \mathsf{b}_{\mathcal{A}} = \vec{b} \qquad \mathsf{c}_{\mathcal{A}} = \vec{c}\,.$$

Let α be an arbitrary assignment and $t = \mathsf{f}(\mathsf{g}(\mathsf{a}, \mathsf{b}), \mathsf{c})$. Then

$$[\alpha]_{\mathcal{A}}(t) = F_1 \cdot \vec{g} + F_1 \cdot G_1 \cdot \vec{a} + F_1 \cdot G_2 \cdot \vec{b} + F_2 \cdot \vec{c} + \vec{f}\,.$$

Clearly the length of each product is at most the depth of the term, which is smaller than or equal to its size. By the previous argument, entries in each product are polynomially bounded (with degree $d-1$) in the size of t. The number of products which are finally summed up equals the number of subterms of t, which is exactly the size of t. Therefore, the entries in $[\alpha]_{\mathcal{A}}(t)$ are bounded by a polynomial of degree d in the size of t. This observation leads us almost directly to the main result of this section.

Theorem 5.5. *For any strict triangular matrix interpretation $\mathcal{A}$ of dimension d, the reduction order $>_{\mathcal{A}}$ induces polynomial complexity, and the degree of the polynomial is d.*

Proof. Any k-step derivation $s \rightarrow_{\mathcal{R}}^{k} t$ implies $s_1 \geqslant t_1 + k$, where $[\alpha]_{\mathcal{A}}(s) = (s_1, \ldots, s_d)$, $[\alpha]_{\mathcal{A}}(t) = (t_1, \ldots, t_d)$, and α is any assignment, also compare Lemma 4.17. In conjunction with Lemma 5.4 and the above observation, this suffices to prove the theorem. □

Using exactly the same reasoning (but employing Lemma 4.18 rather than Lemma 4.17), the corresponding theorem for reduction pairs can be shown:

Theorem 5.6. *For any weak triangular matrix interpretation $\mathcal{A}$ of dimension d, the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces polynomial complexity, and the degree of the polynomial is d.*

Example 5.7. It is easy to see that the derivational complexity of the TRS $\mathcal{R}$ consisting of the rewrite rules $\{\mathsf{a}(\mathsf{b}(x)) \rightarrow \mathsf{b}(\mathsf{a}(x)), \mathsf{c}(\mathsf{a}(x)) \rightarrow \mathsf{b}(\mathsf{c}(x)), \mathsf{c}(\mathsf{b}(x)) \rightarrow \mathsf{a}(\mathsf{c}(x))\}$ is (at least) cubic: for instance, consider the family of starting terms $\{\mathsf{c}^n(\mathsf{a}^n(\mathsf{b}^n(x))) \mid n \in \mathbb{N}\}$. Let $\mathcal{A}$ be the following TMI $\mathcal{A}$.

$$\mathsf{a}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad \mathsf{b}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\mathsf{c}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 1 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

The TRS $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$, so by Theorem 5.5 we conclude that $\mathsf{dc}_{\mathcal{R}}$ is a polynomial of degree 3.

Observe that the criterion is not complete: there exist TRSs with polynomial derivational complexity which are not compatible with any reduction order based on a TMI. One such example can be found in [27]: let $\mathcal{R} = \{\mathsf{f}(\mathsf{a}, \mathsf{b}) \rightarrow \mathsf{f}(\mathsf{b}, \mathsf{b}), \mathsf{f}(\mathsf{b}, \mathsf{a}) \rightarrow \mathsf{f}(\mathsf{a}, \mathsf{a})\}$. Then $\mathcal{R}$ has linear derivational complexity, but in fact no compatible matrix interpretation can exist, cf. [27, Example 7]. Further note that the the degree mentioned in Theorem 5.5 need not be reached, as illustrated by the following example.

Example 5.8. Consider the TRS $\mathcal{R}$ consisting of the single rule $\mathsf{f}(\mathsf{f}(x)) \rightarrow \mathsf{f}(\mathsf{c}(\mathsf{f}(x)))$. The derivation height of any term is given by the number of (possibly overlapping) occurrences of coupled f's in the initial term t, which is linearly bounded in the size of t. For the following matrix interpretation $\mathcal{A}$, $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$. However, $>_{\mathcal{A}}$ only induces quadratic complexity:

$$\mathsf{f}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \cdot \vec{x} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \mathsf{c}_{\mathcal{A}}(\vec{x}) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \vec{x}$$

There even exist TRSs which are compatible with the reduction order based on a matrix interpretation and whose derivational complexity is polynomial, but which are not compatible with any reduction order based on a triangular matrix interpretation. Let $\mathcal{R} = \{\mathsf{f}(\mathsf{f}(x)) \rightarrow \mathsf{f}(\mathsf{c}(\mathsf{f}(x))), \mathsf{c}(\mathsf{c}(x)) \rightarrow x\}$. It is demonstrated in [86, Example 8] that $\mathsf{dc}_{\mathcal{R}}$ is bounded by a polynomial of degree 2, and there exists a matrix interpretation $\mathcal{A}$ such that $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$. However, as also shown in [86, Example 8], there exists no triangular matrix interpretation such that $>_{\mathcal{A}}$ is compatible with $\mathcal{A}$.

5.3 Context Dependent Interpretations and Matrices

In this section, we describe *context dependent interpretations*, compare [49]. We recall previously introduced subclasses of context dependent interpretations which are suitable for automation [96, 79], and extend them. Finally, we show a tight correspondence between (triangular) matrix interpretations as introduced in Section 5.2 and context dependent interpretations. More precisely, we define a subclass of context dependent interpretations such that any such interpretation $\mathcal{C}$ gives rise to a restricted TMI $\mathcal{A}$ and vice versa. Moreover, $\mathcal{C}$ proves termination and polynomial derivational complexity of a TRS $\mathcal{R}$ if and only if $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$.

Essentially, a *context dependent $\mathcal{F}$-algebra* is a family of $\mathcal{F}$-algebras which is parametrised by an element Δ of a *parameter domain.* The crucial idea is that Δ changes throughout the computation of the interpretation of a term, thus making the interpretation of a subterm dependent on its context. Note that, as in [96], our definition of context dependent $\mathcal{F}$-algebras is slightly more general than the notions introduced by Hofbauer in [49].

Definition 5.9. A *context dependent $\mathcal{F}$-algebra* (*CDA* for short) consists of a *carrier A*, a *parameter domain D*, and for each function symbol $f \in \mathcal{F}$ of arity n an *interpretation function $f_{\mathcal{C}} \colon D \times A^n \to A$* and a collection of n *parameter functions $f^i_{\mathcal{C}} \colon D \to D$* for $1 \leqslant i \leqslant n$. Given a *$\Delta$-assignment* $\alpha \colon D \times \mathcal{V} \to A$ and a parameter Δ, we write $[\alpha, \Delta]_{\mathcal{C}}(t)$ to denote the evaluation of a term t by $\mathcal{C}$. This evaluation is defined inductively as follows:

$$\alpha(\Delta, t) = \begin{cases} \alpha(\Delta, t) & \text{if } t \in \mathcal{V} \\ f_{\mathcal{C}}(\Delta, [\alpha, f^1_{\mathcal{C}}(\Delta)]_{\mathcal{C}}(t_1), \dots, [\alpha, f^n_{\mathcal{C}}(\Delta)]_{\mathcal{C}}(t_n)) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

A *well-founded Δ-monotone CDA* is a pair $(\mathcal{C}, \{>_\Delta \mid \Delta \in D\})$, where $\mathcal{C}$ is a CDA with carrier A and parameter domain D. Moreover, every $\Delta \in D$ must be a well-founded proper order, and $a_i >_\Delta b$ must imply $f_{\mathcal{C}}(\Delta, a_1, \dots, a_i, \dots, a_n) >_{f^i_{\mathcal{C}}(\Delta)} f_{\mathcal{C}}(\Delta, a_1, \dots, b, \dots, a_n)$ for all function symbols f of arity n and all $1 \leqslant i \leqslant n$. The last property is called *Δ-monotonicity* of $\mathcal{C}$ with respect to $\{>_\Delta \mid \Delta \in D\}$. For any well-founded Δ-monotone CDA, the following order $>_{\mathcal{C}}$ is defined:

$$s >_{\mathcal{C}} t \iff [\alpha, \Delta]_{\mathcal{C}}(s) >_\Delta [\alpha, \Delta]_{\mathcal{C}}(t) \qquad \text{for all } \Delta \in D \text{ and } \Delta\text{-assignments } \alpha$$

Due to the special role of the parameter Δ, in the remainder of this section we often write $f_{\mathcal{C}}[\Delta](a_1, \dots, a_n)$ instead of $f_{\mathcal{C}}(\Delta, a_1, \dots, a_n)$.

Theorem 5.10. *For any well-founded Δ-monotone CDA $(\mathcal{C}, \{>_\Delta \mid \Delta \in D\})$, the order $>_{\mathcal{C}}$ is a reduction order.*

Proof. Since $>_\Delta$ is a well-founded proper order, it easily follows from the definition of $>_{\mathcal{C}}$ that $>_{\mathcal{C}}$ is a well-founded proper order, as well.

Next, we show closure under substitutions: by definition, $s >_{\mathcal{C}} t$ implies $[\alpha, \Delta]_{\mathcal{C}}(s) >_\Delta [\alpha, \Delta]_{\mathcal{C}}(t)$ for all $\Delta \in D$ and Δ-assignments α. In particular, for all substitutions σ and assignments α', we have $[\alpha, \Delta]_{\mathcal{C}}(s) >_\Delta [\alpha, \Delta]_{\mathcal{C}}(t)$, where α is defined by $\alpha(\Delta, x) = [\alpha', \Delta]_{\mathcal{C}}(\sigma(x))$. Hence, $s\sigma >_{\mathcal{C}} t\sigma$.

Finally, closure under contexts follows from [49, Lemma 1] (if the carrier of $\mathcal{C}$ is $\mathbb{R}_0^+$, and its parameter domain is $\mathbb{R}^+$), or from [96, Lemma 3.5] (for the easy generalisation to the setting considered here). $\square$

Note the obvious similarities between well-founded monotone $\mathcal{F}$-algebras and well-founded Δ-monotone CDAs. Based on that, we proceed by defining the logical sibling of weakly monotone $\mathcal{F}$-algebras.

Definition 5.11. A *weakly monotone CDA* is a triple $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in D\})$, where $\mathcal{C}$ is a CDA with carrier A and parameter domain D, $\geqslant$ is a preorder such that for every function symbol $f \in \mathcal{F}$, $f_\mathcal{C}$ is monotone in all coordinates except the first (i.e., abusing notation, $f_\mathcal{C}[\Delta]$ is monotone in all coordinates) with respect to $\geqslant$, and for all $\Delta \in D$, $>_\Delta$ is a well-founded proper order such that $\geqslant \cdot >_\Delta \cdot \geqslant \subseteq >_\Delta$. For any weakly monotone CDA $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in D\})$, the following orders $\geqslant_\mathcal{C}$ and $>_\mathcal{C}$ are defined:

$$s \geqslant_\mathcal{C} t \iff [\alpha, \Delta]_\mathcal{C}(s) \geqslant [\alpha, \Delta]_\mathcal{C}(t) \qquad \text{for all } \Delta \in D \text{ and } \Delta\text{-assignments } \alpha$$
$$s >_\mathcal{C} t \iff [\alpha, \Delta]_\mathcal{C}(s) >_\Delta [\alpha, \Delta]_\mathcal{C}(t) \qquad \text{for all } \Delta \in D \text{ and } \Delta\text{-assignments } \alpha$$

Theorem 5.12. *For any weakly monotone CDA $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in D\})$, the pair of orders $(\geqslant_\mathcal{C}, >_\mathcal{C})$ is a reduction pair.*

Proof. Straightforward extension of Theorem 5.10. $\square$

We now introduce a specific subclass of CDAs called Δ^2*-interpretations*. They are a generalisation of Δ*-linear interpretations* and Δ*-restricted interpretations*, which were considered in [96, 79].

Definition 5.13. A Δ^2*-interpretation* is a CDA $\mathcal{C}$ with carrier $\mathbb{R}_0^+$, parameter domain $\mathbb{R}^+$, and interpretation functions and parameter functions of the following form:

$$f_\mathcal{C}[\Delta](x_1, \ldots, x_n) = \sum_{i=1}^{n} a_{(f,i)} x_i + \sum_{i=1}^{n} b_{(f,i)} x_i \Delta + g_f + h_f \Delta \tag{5.1}$$
$$f_\mathcal{C}^i(\Delta) = \frac{c_{(f,i)} + d_{(f,i)}\Delta}{a_{(f,i)} + b_{(f,i)}\Delta}\,, \tag{5.2}$$

where $a_{(f,i)} > 0$ or $b_{(f,i)} > 0$, and $c_{(f,i)} > 0$ or $d_{(f,i)} > 0$ (for each $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$), and the occurring coefficients are natural numbers.

A Δ*-linear interpretation* is a Δ^2-interpretation, where for the parameter functions as presented in (5.2) we have $c_{(f,i)} = 0$ and $d_{(f,i)} = 1$ for all $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$. A Δ*-restricted interpretation* is a Δ-linear interpretation with the additional requirement that in the interpretation and parameter functions as presented in (5.1) and (5.2), we have $a_{(f,i)} \in \{0, 1\}$.

Theorem 5.14. *Let $\mathcal{C}$ be a Δ^2-interpretation, and let $>_\Delta$ be defined by $s >_\Delta t \iff s \geqslant t + \Delta$ for all $\Delta \in \mathbb{R}^+$, where $\geqslant$ is the usual weak order on $\mathbb{R}_0^+$. Suppose that $\mathcal{C}$ is Δ-monotone with respect to $\{>_\Delta \mid \Delta \in \mathbb{R}^+\}$. Then $(\mathcal{C}, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$ is a well-founded Δ-monotone CDA.*

Proof. Obviously, for every $\Delta \in \mathbb{R}^+$, the order $>_\Delta$ is a well-founded proper order on $\mathbb{R}_0^+$. Moreover, by assumption, $\mathcal{C}$ is a CDA, and is Δ-monotone with respect to $\{>_\Delta \mid \Delta \in \mathbb{R}^+\}$. Thus $(\mathcal{C}, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$ is a well-founded Δ-monotone CDA. □

Theorem 5.15. *Let $\mathcal{C}$ be a Δ^2-interpretation, let $\geqslant$ be the usual weak order on $\mathbb{R}_0^+$, and let $>_\Delta$ be defined by $s >_\Delta t \iff s \geqslant t + \Delta$ for all $\Delta \in \mathbb{R}^+$. Then $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$ is a weakly monotone CDA.*

Proof. Obviously, $\geqslant$ is a preorder, and for every $\Delta \in \mathbb{R}^+$, the order $>_\Delta$ is a proper order on $\mathbb{R}_0^+$. Moreover, $\geqslant \cdot >_\Delta \cdot \geqslant \; \subseteq \; >_\Delta$ for all $\Delta \in \mathbb{R}^+$. By the definition of Δ^2-interpretations, for every $f \in \mathcal{F}$, $f_\mathcal{C}$ is monotone with respect to $\geqslant$ in all coordinates except the first. Thus, $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$ is a weakly monotone CDA. □

If $\mathcal{C}$ is a Δ^2-interpretation, we often only write $\mathcal{C}$ for $(\mathcal{C}, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$ if the premises of Theorem 5.14 hold for $(\mathcal{C}, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$; we call $\mathcal{C}$ a *strict Δ^2-interpretation* in that case. Similarly, we often write $\mathcal{C}$ for $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$ if the premises of Theorem 5.15 hold for $(\mathcal{C}, \geqslant, \{>_\Delta \mid \Delta \in \mathbb{R}^+\})$; then we call $\mathcal{C}$ a *weak Δ^2-interpretation.*

For Δ^2-interpretations, there is a simple syntactic criterion allowing the inference of Δ-monotonicity.

Lemma 5.16. *Let $\mathcal{C}$ be a Δ^2-interpretation. Let $\geqslant$ be the usual weak order on $\mathbb{R}_0^+$, and let $>_\Delta$ be defined by $s >_\Delta t \iff s \geqslant t + \Delta$ for all $\Delta \in \mathbb{R}^+$. Then for all $f \in \mathcal{F}$, $f_\mathcal{C}$ is monotone in all coordinates except the first with respect to $\geqslant$. Moreover, if for all $f \in \mathcal{F}$, $1 \leqslant i \leqslant n$ in (5.2), we have $d_{(f,i)} \geqslant 1$, then $\mathcal{C}$ is Δ-monotone with respect to $\{>_\Delta \mid \Delta \in \mathbb{R}^+\}$.*

Proof. For all $f \in \mathcal{F}$, monotonicity of $f_\mathcal{C}$ in all coordinates with respect to $\geqslant$ is immediate from Definition 5.13. Hence, the first part of the lemma is concluded. Moreover, due to this, having

$$f_\mathcal{C}[\Delta](x_1 \dots, x_i + f^i_\mathcal{C}(\Delta), \dots, x_n) - f_\mathcal{C}[\Delta](x_1 \dots, x_i, \dots, x_n) \geqslant \Delta$$

for all function symbols $f \in \mathcal{F}$ of arity n, $1 \leqslant i \leqslant n$, $\Delta \in \mathbb{R}^+$, and $x_1, \dots, x_n \in \mathbb{R}_0^+$ is sufficient to conclude Δ-monotonicity of $\mathcal{C}$ with respect to $\{>_\Delta \mid \Delta \in \mathbb{R}^+\}$. This inequality can be shown as follows:

$$\begin{aligned} &f_\mathcal{C}[\Delta](x_1 \dots, x_i + f^i_\mathcal{C}(\Delta), \dots, x_n) - f_\mathcal{C}[\Delta](x_1 \dots, x_i, \dots, x_n) \\ &\qquad = (a_i + b_i\Delta)(x_i + \frac{c_i + d_i\Delta}{a_i + b_i\Delta}) - (a_i + b_i\Delta)x_i = c_i + d_i\Delta \geqslant \Delta \end{aligned}$$

□

Example 5.17 ([49], continued from Example 5.1)**.** Consider the following Δ-linear interpretation $\mathcal{C}$:

$$\circ_\mathcal{C}[\Delta](x,y) = (1+\Delta)x + y + 1 \quad \circ^1_\mathcal{C}(\Delta) = \frac{\Delta}{1+\Delta} \quad \circ^2_\mathcal{C}(\Delta) = \Delta \quad \mathsf{c}_\mathcal{C}[\Delta] = 0$$

For all $\Delta \in \mathbb{R}^+$, ground terms r, s, t, and Δ-assignments α, we have $[\alpha, \Delta]_{\mathcal{C}}((r \circ s) \circ t) - [\alpha, \Delta]_{\mathcal{C}}(r \circ (s \circ t)) \geqslant \Delta$. This is shown in [49, Lemma 3] by an argument using induction on r. However, this argument is not well-suited for automation: the implementation described in [79], which constructs Δ-linear and Δ-restricted interpretations $\mathcal{C}$, ensuring compatibility of the given TRS with $>_{\mathcal{C}}$ automatically, can not find this interpretation.

The following general upper bounds on derivation heights follow quite easily from the definition of strict and weak Δ^2-interpretations, and the structure of the employed orders $>_\Delta$ for $\Delta \in \mathbb{R}^+$.

Lemma 5.18 ([49])**.** *Let $\mathcal{R}$ be a TRS and $\mathcal{C}$ a strict Δ^2-interpretation such that $>_{\mathcal{C}}$ is compatible with $\mathcal{R}$. Then for all terms t and Δ-assignments α, we have*

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \inf_{\Delta \in \mathbb{R}^+} \frac{[\alpha, \Delta]_{\mathcal{C}}(t)}{\Delta} .$$

Lemma 5.19. *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and $\mathcal{C}$ a weak Δ^2-interpretation. Suppose that $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{C}}, >_{\mathcal{C}})}$ completely solves $(\mathcal{P}, \mathcal{R})$. Then for all terms t and Δ-assignments α, we have*

$$\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathcal{P}} / \to_{\mathcal{R}}) \leqslant \inf_{\Delta \in \mathbb{R}^+} \frac{[\alpha, \Delta]_{\mathcal{C}}(t)}{\Delta} .$$

Based on Lemma 5.18, it has been shown in [96, Theorem 5.3] and [79, Theorem 29] that for any strict Δ-restricted interpretation $\mathcal{C}$, the reduction order $>_{\mathcal{C}}$ induces quadratic complexity. Moreover, by [79, Theorem 25], for any strict Δ-linear interpretation $\mathcal{C}$, the reduction order $>_{\mathcal{C}}$ induces exponential complexity. Using Lemma 5.19 instead of Lemma 5.18, it is easy to lift these two theorems to the reduction pair $(\geqslant_{\mathcal{C}}, >_{\mathcal{C}})$ induced by a weak Δ-restricted or Δ-linear interpretation, respectively.

In the remainder of this section, we give an alternative proof of these theorems. We show a tight correspondence between matrix interpretations of dimension 2 and Δ^2-interpretations. Using this correspondence, we can then easily show the above theorems by reducing them to Theorems 5.5, 5.6, 4.19, and 4.20.

Example 5.20 ([49], continued from Example 5.17)**.** The Δ-linear interpretation $\mathcal{C}$ is also a Δ-restricted interpretation. Due to [79, Theorem 29] we conclude quadratic derivational complexity for $\mathcal{R}$. Note that this upper bound is optimal: $\mathsf{dc}_{\mathcal{R}_{\mathsf{Assoc}}}$ is at least quadratic, as witnessed by the family of starting terms defined by $t_0 = \mathsf{c} \circ \mathsf{c}$ and $t_{n+1} = t_n \circ \mathsf{c}$.

It seems worthy of note that the interpretation $\mathcal{B}$ employed in Example 5.3 is obtained fully automatically, while the interpretation $\mathcal{C}$ is obtained by hand. Further notice that the context dependent interpretation employed in Example 5.17 above uses exactly the same coefficients as the TMI given in Example 5.3. This observation motivates the next definition and lemma.

Definition 5.21. Let $\mathcal{C}$ be a Δ^2-interpretation and let $\mathcal{A}$ be a matrix interpretation of dimension 2. We say the Δ-assignment $\alpha\colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}_0^+$ and the assignment $\alpha'\colon \mathcal{V} \to \mathbb{N}^2$ are *corresponding* if for all variables x and all $\Delta \in \mathbb{R}^+$, we have

$$\alpha(\Delta, x) = a_x + b_x\Delta \qquad \text{and} \qquad \alpha'(x) = \begin{pmatrix} b_x \\ a_x \end{pmatrix}$$

for some $a_x, b_x \in \mathbb{N}$.

Lemma 5.22. *Let $\mathcal{C}$ denote a Δ^2-interpretation having the shape*

$$f_{\mathcal{C}}[\Delta](x_1, \dots, x_n) = \sum_{i=1}^{n} a_{(f,i)}x_i + \sum_{i=1}^{n} b_{(f,i)}x_i\Delta + g_f + h_f\Delta$$
$$f_{\mathcal{C}}^i(\Delta) = \frac{c_{(f,i)} + d_{(f,i)}\Delta}{a_{(f,i)} + b_{(f,i)}\Delta}\ ,$$

and let $\mathcal{A}$ denote a matrix interpretation of the following form:

$$f_{\mathcal{A}}(\vec{x}_1, \dots, \vec{x}_n) = \Big(\sum_{i=1}^{n} \begin{pmatrix} d_{(f,i)} & b_{(f,i)} \\ c_{(f,i)} & a_{(f,i)} \end{pmatrix} \cdot \vec{x}_i\Big) + \begin{pmatrix} h_f \\ g_f \end{pmatrix}\ ,$$

where $d_{(f,i)} \geqslant 1$, and $a_{(f,i)} > 0$ or $b_{(f,i)} > 0$, and $c_{(f,i)} > 0$ or $d_{(f,i)} > 0$ for all $f \in \mathcal{F}$ and $1 \leqslant i \leqslant n$. Then for any term t, Δ-assignment α, and assignment α', we have

$$[\alpha, \Delta]_{\mathcal{C}}(t) = s_1 + s_2\Delta \qquad \text{and} \qquad [\alpha']_{\mathcal{A}}(t) = \begin{pmatrix} s_2 \\ s_1 \end{pmatrix}$$

for some $s_1, s_2 \in \mathbb{N}$ whenever α and α' are corresponding.

Proof. Let $\alpha\colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}_0^+$ and $\alpha'\colon \mathcal{V} \to \mathbb{N}^2$ be corresponding (Δ-)assignments. In order to prove the lemma, we proceed by induction on t.

If $t \in \mathcal{V}$, then the lemma follows by the assumption that α and α' are corresponding. Now assume $t = f(t_1, \dots, t_n)$. By induction hypothesis we have that there exist $u_i, v_i \in \mathbb{N}$ such that $[\alpha, \Delta]_{\mathcal{C}}(t_i) = u_i + v_i\Delta$ and $[\alpha']_{\mathcal{A}}(t_i) = \begin{pmatrix} v_i \\ u_i \end{pmatrix}$ for all $1 \leqslant i \leqslant n$. First we consider the evaluation of t with respect to $\mathcal{C}$ and α:

$$\begin{aligned}
&[\alpha, \Delta]_{\mathcal{C}}(f(t_1, \dots, t_n)) \\
&\quad = \sum_{i=1}^{n} a_{(f,i)}[\alpha, f_{\mathcal{C}}^i(\Delta)](t_i) + \sum_{i=1}^{n} b_{(f,i)}[\alpha, f_{\mathcal{C}}^i(\Delta)](t_i)\Delta + g_f + h_f\Delta \\
&\quad = \sum_{i=1}^{n} a_{(f,i)}(u_i + v_i f_{\mathcal{C}}^i(\Delta)) + \sum_{i=1}^{n} b_{(f,i)}\Delta(u_i + v_i f_{\mathcal{C}}^i(\Delta)) + g_f + h_f\Delta \\
&\quad = \sum_{i=1}^{n} a_{(f,i)}u_i + \sum_{i=1}^{n} b_{(f,i)}u_i\Delta + \sum_{i=1}^{n}(a_{(f,i)} + b_{(f,i)}\Delta)v_i f_{\mathcal{C}}^i(\Delta) + g_f + h_f\Delta \\
&\quad = \sum_{i=1}^{n} a_{(f,i)}u_i + \sum_{i=1}^{n} b_{(f,i)}u_i\Delta + \sum_{i=1}^{n} c_{(f,i)}v_i + \sum_{i=1}^{n} d_{(f,i)}v_i\Delta + g_f + h_f\Delta \\
&\quad = \big(\sum_{i=1}^{n} a_{(f,i)}u_i + c_{(f,i)}v_i + g_f\big) + \big(\sum_{i=1}^{n} b_{(f,i)}u_i + d_{(f,i)}v_i + h_f\big)\Delta
\end{aligned}$$

Now, consider the evaluation of t with respect to $\mathcal{A}$ and α':

$$\begin{aligned}
&[\alpha']_{\mathcal{A}}(f(t_1,\dots,t_n)) = \\
&\quad = \sum_{i=1}^{n} \begin{pmatrix} d_{(f,i)} & b_{(f,i)} \\ c_{(f,i)} & a_{(f,i)} \end{pmatrix} [\alpha']_{\mathcal{A}}(t_i) + \begin{pmatrix} h_f \\ g_f \end{pmatrix} \\
&\quad = \sum_{i=1}^{n} \begin{pmatrix} d_{(f,i)} & b_{(f,i)} \\ c_{(f,i)} & a_{(f,i)} \end{pmatrix} \begin{pmatrix} v_i \\ u_i \end{pmatrix} + \begin{pmatrix} h_f \\ g_f \end{pmatrix} \\
&\quad = \begin{pmatrix} \sum_{i=1}^{n} b_{(f,i)} u_i + d_{(f,i)} v_i + h_f \\ \sum_{i=1}^{n} a_{(f,i)} u_i + c_{(f,i)} v_i + g_f \end{pmatrix}
\end{aligned}$$

From this, we immediately see that the lemma holds. □

We say a matrix interpretation $\mathcal{A}$ *corresponds* to a Δ^2-interpretation $\mathcal{C}$ if $\mathcal{A}$ and $\mathcal{C}$ are defined as in Lemma 5.22. Note that any matrix interpretation corresponding to a Δ-restricted interpretation is a triangular matrix interpretation.

Example 5.23. Consider the triangular matrix interpretation $\mathcal{A}$ introduced in Example 5.3 and the Δ-restricted interpretation $\mathcal{C}$ from Example 5.17. The interpretations $\mathcal{A}$ and $\mathcal{C}$ are corresponding.

Theorem 5.24. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a strict Δ^2-interpretation such that $\mathcal{R}$ is compatible with $>_{\mathcal{C}}$. Then there exists a corresponding matrix interpretation $\mathcal{A}$ of dimension 2 such that $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$.*

Proof. To prove the theorem, we need to show that for any rule $l \to r \in \mathcal{R}$, the inequality $l >_{\mathcal{A}} r$ holds, where $\mathcal{A}$ is the matrix interpretation constructed in Lemma 5.22. Let α be an arbitrary, but fixed assignment. Let α' be a Δ-assignment which corresponds to α. For every $l \to r \in \mathcal{R}$ and every $\Delta \in \mathbb{R}^+$, we have $[\alpha', \Delta]_{\mathcal{C}}(l) - [\alpha', \Delta]_{\mathcal{C}}(r) = a + b\Delta$ for some $a, b \in \mathbb{Z}$. Here we make use of the fact that for any term t, we have $[\alpha', \Delta]_{\mathcal{C}}(t) = c + d\Delta$ for some $c, d \in \mathbb{N}$. This follows from an inductive argument employing the assumed form of the assignment α', as in the proof of Lemma 5.22. Moreover, as $a + b\Delta \geqslant \Delta$ for all $\Delta \in \mathbb{R}^+$ due to compatibility of $\mathcal{R}$ with $>_{\mathcal{C}}$, we conclude $a \geqslant 0$ and $b \geqslant 1$. Thus an application of Lemma 5.22 yields

$$[\alpha]_{\mathcal{A}}(l) - [\alpha]_{\mathcal{A}}(r) = \begin{pmatrix} b \\ a \end{pmatrix} \geqslant \begin{pmatrix} 1 \\ 0 \end{pmatrix} ,$$

from which compatibility of $\mathcal{R}$ with $>_{\mathcal{A}}$ directly follows. □

Theorem 5.25. *Let $(\mathcal{P}, \mathcal{R})$ be a DP problem, let $\mathcal{C}$ be a weak Δ^2-interpretation, and let $\{(\mathcal{P}', \mathcal{R})\} = \Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{C}}, >_{\mathcal{C}})}((\mathcal{P}, \mathcal{R}))$. Then there exists a corresponding weak matrix interpretation $\mathcal{A}$ of dimension 2 with $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}', \mathcal{R})\}$.*

Proof. Analogous to Theorem 5.24. □

The following corollaries are easy consequences of Theorem 5.24 in conjunction with Theorems 5.5 and 4.19, and Theorem 5.25 in conjunction with Theorems 5.6 and 4.20, respectively.

Corollary 5.26. *For any strict Δ-restricted interpretation (respectively Δ^2-interpretation) $\mathcal{C}$, the reduction order $>_{\mathcal{C}}$ induces quadratic (respectively exponential) complexity.*

Corollary 5.27. *For any weak Δ-restricted interpretation (respectively Δ^2-interpretation) $\mathcal{C}$, the reduction pair $(\geqslant_{\mathcal{C}}, >_{\mathcal{C}})$ induces quadratic (respectively exponential) complexity.*

Theorem 5.24 raises the question whether the other direction may hold for Δ^2-interpretations: Given a matrix interpretation $\mathcal{A}$ of dimension 2 such that $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$, does there exist a Δ^2-interpretation $\mathcal{C}$ such that $\mathcal{R}$ is compatible with $>_{\mathcal{C}}$? Or, for DP problems and Theorem 5.25: given a matrix interpretation of dimension 2 and a DP problem $(\mathcal{P}, \mathcal{R})$ such that $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}', \mathcal{R})\}$, does there exist a Δ^2-interpretation $\mathcal{C}$ such that $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{C}}, >_{\mathcal{C}})}((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}', \mathcal{R})\}$? The orders $>_{\mathcal{C}}$ and $\geqslant_{\mathcal{C}}$ are not well-suited for a reversal of Theorems 5.24 and 5.25. This is illustrated by the next example.

Example 5.28 (continued from Example 5.17)**.** The Δ-restricted interpretation $\mathcal{C}$ is not compatible with $\mathcal{R}_{\mathsf{Assoc}}$ as defined above, i.e. we do not have $[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) - [\alpha, \Delta]_{\mathcal{C}}(x \circ (y \circ z)) \geqslant \Delta$ for arbitrary Δ-assignments α. To construct a counter-example we set $\alpha(\Delta, x) = \Delta^2$, and $\alpha(\Delta, u)$ arbitrary for $u \neq x$. We obtain

$$\begin{aligned}
&[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) - [\alpha, \Delta]_{\mathcal{C}}(x \circ (y \circ z)) \\
&\quad = (1 + 2\Delta)[\alpha, \frac{\Delta}{1 + 2\Delta}]_{\mathcal{C}}(x) + \Delta - (1 + \Delta)[\alpha, \frac{\Delta}{1 + \Delta}]_{\mathcal{C}}(x) \\
&\quad = \Delta + \frac{\Delta^2}{1 + 2\Delta} - \frac{\Delta^2}{1 + \Delta} \ngeqslant \Delta \,.
\end{aligned}$$

This violates the compatibility condition. The second line is obtained by straightforward simplification of the interpretations of the terms, as demonstrated in the proof of [49, Lemma 3].

Example 5.28 motivates the next definition.

Definition 5.29. We say that a Δ-assignment $\alpha \colon \mathbb{R}^+ \times \mathcal{V} \to \mathbb{R}^+_0$ over the real numbers is *linear* if for each variable x there exist natural numbers a and b such that for all $\Delta \in \mathbb{R}^+$, we have $\alpha(\Delta, x) = a + b\Delta$.

Example 5.30 (continued from Example 5.28)**.** For any linear Δ-assignment α, we have $[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) - [\alpha, \Delta]_{\mathcal{C}}(x \circ (y \circ z)) \geqslant \Delta$. This can be seen by just applying a linear Δ-assignment of the following general form: $\alpha(\Delta, x) = x_1 + x_2\Delta$, $\alpha(\Delta, y) = y_1 + y_2\Delta$, and $\alpha(\Delta, z) = z_1 + z_2\Delta$. Then

$$\begin{aligned}
&[\alpha, \Delta]_{\mathcal{C}}((x \circ y) \circ z) \\
&\quad = (1 + 2\Delta)[\alpha, \frac{\Delta}{1 + 2\Delta}]_{\mathcal{C}}(x) + (1 + \Delta)[\alpha, \frac{\Delta}{1 + \Delta}]_{\mathcal{C}}(y) + [\alpha, \Delta]_{\mathcal{C}}(z) + \Delta + 2 \\
&\quad = (1 + 2\Delta)x_1 + x_2\Delta + (1 + \Delta)y_1 + y_2\Delta + z_1 + z_2\Delta + \Delta + 2
\end{aligned}$$

and

$$\begin{aligned}
&[\alpha,\Delta]_{\mathcal{C}}(x \circ (y \circ z)) \\
&\quad = (1+\Delta)[\alpha, \frac{\Delta}{1+\Delta}]_{\mathcal{C}}(x) + (1+\Delta)[\alpha, \frac{\Delta}{1+\Delta}]_{\mathcal{C}}(y) + [\alpha,\Delta]_{\mathcal{C}}(z) + 2 \\
&\quad = (1+\Delta)x_1 + x_2\Delta + (1+\Delta)y_1 + y_2\Delta + z_1 + z_2\Delta + 2 \;.
\end{aligned}$$

Thus, $(x \circ y) \circ z >_{\mathcal{C}} x \circ (y \circ z)$.

Lemma 5.31. *Let σ be a substitution, let $\mathcal{C}$ be a Δ^2-interpretation, and let α be a linear Δ-assignment. Then there exists a linear Δ-assignment α' such that $[\alpha,\Delta]_{\mathcal{C}}(t\sigma) = [\alpha',\Delta]_{\mathcal{C}}(t)$ for all terms t and $\Delta \in \mathbb{R}^+$.*

Proof. We set $\alpha'(\Delta, x) = [\alpha,\Delta]_{\mathcal{C}}(\sigma(x))$ for any variable x. Now it follows by easy inductions (on $t\sigma$ and $\sigma(x)$, respectively) that $[\alpha,\Delta]_{\mathcal{C}}(t\sigma) = [\alpha',\Delta]_{\mathcal{C}}(t)$, and that $[\alpha',\Delta]_{\mathcal{C}}(x\sigma)$ has a linear shape. □

Example 5.28 and Lemma 5.31 motivate the next definition.

Definition 5.32. For any Δ^2-interpretation $\mathcal{C}$, we define the following orders $\geqslant_{\mathcal{C}}^{\text{lin}}$ and $>_{\mathcal{C}}^{\text{lin}}$:

$$\begin{aligned}
s \geqslant_{\mathcal{C}}^{\text{lin}} t &\iff [\alpha,\Delta]_{\mathcal{C}}(s) \geqslant [\alpha,\Delta]_{\mathcal{C}}(t) \qquad \text{for all } \Delta \in \mathbb{R}^+ \text{ and linear } \alpha \\
s >_{\mathcal{C}}^{\text{lin}} t &\iff [\alpha,\Delta]_{\mathcal{C}}(s) >_{\Delta} [\alpha,\Delta]_{\mathcal{C}}(t) \qquad \text{for all } \Delta \in \mathbb{R}^+ \text{ and linear } \alpha
\end{aligned}$$

Theorem 5.33. *For any strict Δ^2-interpretation $\mathcal{C}$, the order $>_{\mathcal{C}}^{\text{lin}}$ is a reduction order which induces exponential complexity. Moreover, if $\mathcal{C}$ is a Δ-restricted interpretation, then $>_{\mathcal{C}}^{\text{lin}}$ induces quadratic complexity.*

Proof. Closure under substitutions of $>_{\mathcal{C}}^{\text{lin}}$ follows by Lemma 5.31, while the other properties of reduction orders follow analogous to Theorem 5.10.

Now we show that $>_{\mathcal{C}}^{\text{lin}}$ induces quadratic complexity. Let $\mathcal{R}$ be a TRS which is compatible with $>_{\mathcal{C}}^{\text{lin}}$. By Lemma 5.22 and Theorem 5.24, there exists a matrix interpretation $\mathcal{A}$ of dimension 2 corresponding to $\mathcal{C}$ such that $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$. If $\mathcal{C}$ is a Δ-restricted interpretation, then $\mathcal{A}$ is a triangular matrix interpretation. Hence, $\mathsf{dc}_{\mathcal{R}}$ is bounded by an exponential function (and by a polynomial of degree 2, if $\mathcal{C}$ is a Δ-restricted interpretation), and the theorem follows. □

Theorem 5.34. *For any weak Δ^2-interpretation $\mathcal{C}$, the pair of orders $(\geqslant_{\mathcal{C}}^{\text{lin}}, >_{\mathcal{C}}^{\text{lin}})$ is a reduction pair which induces exponential complexity. Moreover, if $\mathcal{C}$ is a Δ-restricted interpretation, then $(\geqslant_{\mathcal{C}}^{\text{lin}}, >_{\mathcal{C}}^{\text{lin}})$ induces quadratic complexity.*

Proof. Analogous to Theorem 5.33. □

Theorem 5.35. *Let $\mathcal{A}$ be a strict matrix interpretation of dimension 2 such that no zero column occurs in any matrix in the interpretation function $f_{\mathcal{A}}$ of any $f \in \mathcal{F}$, let $\mathcal{C}$ be a corresponding Δ^2-interpretation and let $\mathcal{R}$ be a TRS. Then $\mathcal{R}$ is compatible with $>_{\mathcal{A}}$ if and only if $\mathcal{R}$ is compatible with $>_{\mathcal{C}}^{\text{lin}}$.*

Proof. Let $\mathcal{C}$ be a Δ^2-interpretation corresponding to $\mathcal{A}$. Note that since no matrix in any interpretation function of $\mathcal{A}$ contains any zero columns, the side condition that $a_{(f,i)} > 0$ or $b_{(f,i)} > 0$, and $c_{(f,i)} > 0$ or $d_{(f,i)} > 0$ for all $f \in \mathcal{F}$ of arity n and $1 \leqslant i \leqslant n$ in Definition 5.13 is satisfied in any case. By Lemma 5.22, for any linear Δ-assignment α and assignment α' which are corresponding, and any term t, the interpretations of t have the shapes

$$[\alpha, \Delta]_{\mathcal{C}}(t) = s_1 + \Delta s_2 \qquad \text{and} \qquad [\alpha']_{\mathcal{A}}(t) = \begin{pmatrix} s_2 \\ s_1 \end{pmatrix} .$$

Moreover, for any linear Δ-assignment α, there exists a corresponding assignment α', and vice versa. Hence, for any rewrite rule $l \to r$, we have $l >_{\mathcal{C}}^{\mathrm{lin}} r$ if and only if $l >_{\mathcal{A}} r$, and the theorem follows. □

Theorem 5.36. *Let $\mathcal{A}$ be a weak matrix interpretation of dimension* 2 *such that no zero column occurs in any matrix in the interpretation function $f_{\mathcal{A}}$ of any symbol f, let $\mathcal{C}$ be a corresponding Δ^2-interpretation, and let $(\mathcal{P}, \mathcal{R})$ be a DP problem. Suppose that $\{(\mathcal{P}', \mathcal{R})\} = \Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}((\mathcal{P}, \mathcal{R}))$. Then we also have $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{C}}^{\mathrm{lin}}, >_{\mathcal{C}}^{\mathrm{lin}})}((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}', \mathcal{R})\}$.*

Proof. Analogous to Theorem 5.35. □

5.4 Matrix Interpretations of Polynomial Growth

Triangular matrix interpretations, as defined in Section 5.2 above, form a simple (since they are a straightforward syntactic restriction of matrix interpretations), but effective tool for inferring polynomial bounds on the derivational complexity of TRSs. The key observation in the proof of the polynomial complexity bounds is that the entries of any product of matrices from the interpretation grow only polynomially in the length of the considered product. For triangular matrix interpretations, this observation was shown in Lemma 5.4. With that, polynomial derivation complexity of the TRS under consideration essentially follows from Lemma 4.17. However, it is possible to find matrix interpretations such that the entries of products of matrices from such an interpretation still grow polynomially although the interpretation is not a triangular one. In this section, we present a complete characterisation of matrix interpretations with polynomial growth from the recent literature [86, 110, 76]. We start by giving a generalisation of the above mentioned concepts.

Definition 5.37 ([76, Definitions 1 and 9])**.** Let $\mathcal{M}$ be a finite set of matrices from $K^{d \times d}$ for some $d \in \mathbb{N}$ and $K \in \{\mathbb{N}, \mathbb{Q}_0^+, \mathbb{R}_0^+\}$. The *growth function* of $\mathcal{M}$, denoted by $\mathsf{growth}_{\mathcal{M}} \colon \mathbb{N} \to \mathbb{N}$, is defined by

$$\mathsf{growth}_{\mathcal{M}}(n) = \max\{\|M_1 \cdot \ldots \cdot M_n\| \mid M_i \in \mathcal{M}, 1 \leqslant i \leqslant n\} .$$

Let $\mathcal{A}$ be a matrix interpretation, and $\mathcal{M}$ the set of matrices occurring in the interpretation functions of $\mathcal{A}$. Then we say that $\mathcal{A}$ is *polynomially bounded (with degree d)* if $\mathsf{growth}_{\mathcal{M}}(n) \in \mathsf{O}(n^d)$.

As an example, let $\mathcal{A}$ be any triangular matrix interpretation of dimension d. Then by Lemma 5.4, $\mathsf{growth}_{\mathcal{M}}$ is bounded by a polynomial of degree $d-1$, and hence $\mathcal{A}$ is polynomially bounded with degree $d-1$.

Definition 5.37 gives rise to a simple generalisation of the arguments given in Section 5.2:

Lemma 5.38 ([76, Lemma 2])**.** *For any strict matrix interpretation $\mathcal{A}$ which is polynomially bounded with degree d, the reduction order $>_{\mathcal{A}}$ induces polynomial complexity, and the degree of the polynomial is $d+1$.*

Lemma 5.39. *For any weak matrix interpretation $\mathcal{A}$ which is polynomially bounded with degree d, the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces polynomial complexity, and the degree of the polynomial is $d+1$.*

Strictly speaking, Definition 5.37 does not capture exactly what is needed to prove polynomial derivational complexity of TRSs via Lemma 4.17: according to [76, Example 27], there exist TRSs $\mathcal{R}$ and matrix interpretations $\mathcal{A}$ such that $\mathcal{A}$ is not polynomially bounded, but for all terms t, the evaluation $[\alpha_0]_{\mathcal{A}}(t)$ of t is bounded polynomially in $|t|$, where $\alpha_0(x) = 0$ for all $x \in \mathcal{V}\mathsf{ar}(t)$. However, it follows by [76, Lemma 29] that any matrix interpretation $\mathcal{A}$ which is not polynomially bounded, but satisfies that $[\alpha_0]_{\mathcal{A}}(t)$ is bounded polynomially (with degree d) in $|t|$, can be transformed into matrix interpretation $\mathcal{B}$ which is polynomially bounded with the same degree d. Moreover, $\mathcal{B}$ can be obtained from $\mathcal{A}$ just by stripping away some rows and columns in the interpretation functions of $\mathcal{A}$. Therefore, we ignore this slight inaccuracy of Definition 5.37.

Polynomial boundedness of matrix interpretations can be characterised using the *joint spectral radius* of the matrices occurring in the interpretation functions.

Definition 5.40 ([57, Definition 1.1])**.** Let $\mathcal{M}$ be a finite set of matrices from $K^{d\times d}$ for some $d \in \mathbb{N}$ and $K \in \{\mathbb{N}, \mathbb{Q}_0^+, \mathbb{R}_0^+\}$. Then the *joint spectral radius* of $\mathcal{M}$, denoted by $\rho(\mathcal{M})$, is defined by

$$\rho(\mathcal{M}) = \lim_{k\to\infty} \max\{\|M_1 \cdot \ldots \cdot M_k\|^{1/k} \mid M_i \in \mathcal{M} \text{ for all } 1 \leqslant i \leqslant k\}$$

Theorem 5.41 ([10, Theorem 1.2], see also [76, Theorem 11])**.** *Let $\mathcal{M}$ be a finite set of matrices from $K^{d\times d}$ for some $d \in \mathbb{N}$ and $K \in \{\mathbb{N}, \mathbb{Q}_0^+, \mathbb{R}_0^+\}$. Then $\mathsf{growth}_{\mathcal{M}}$ is bounded by a polynomial if and only if $\rho(\mathcal{M}) \leqslant 1$. More precisely, if $\rho(\mathcal{M}) \leqslant 1$, then $\mathsf{growth}_{\mathcal{M}}(n) \in \mathsf{O}(n^{d-1})$.*

In general, it is not decidable whether the joint spectral radius of a set of matrices is at most 1. However, if the domain of the matrix entries is restricted to $\mathbb{N}$, or any subset of $\mathbb{Q}_0^+$ or $\mathbb{R}_0^+$ which contains no element x such that $0 < x < 1$, then this problem becomes decidable in polynomial time. The following lemma gives rise to a polynomial-time decision procedure for this case:

Lemma 5.42 ([57, Lemma 3.3], see also [76, Lemma 12])**.** *Let $\mathcal{M}$ be a finite set of matrices from $K^{d\times d}$ for some $d \in \mathbb{N}$ and $K \in \{\mathbb{N}, \mathbb{Q}_0^+, \mathbb{R}_0^+\}$. Suppose that for no entry x of any matrix in $\mathcal{M}$, we have $0 < x < 1$. Then $\rho(\mathcal{M}) > 1$ if and only if there exists a matrix product $N = M_1 \cdot \ldots \cdot M_n$ such that $M_1, \ldots, M_n \in \mathcal{M}$ and $N_{i,i} > 1$ for some $1 \leqslant i \leqslant d$.*

Let $\mathcal{A}$ be a matrix interpretation and $\mathcal{M}$ the set of matrices occurring in $\mathcal{A}$. Then the criterion exhibited in Lemma 5.42, which is equivalent to $\rho(\mathcal{M}) > 1$, essentially coincides with the criterion *EDA* for weighted automata, which is defined in [111], and applied for deciding polynomial ambiguity of weighted automata (and hence, as an alternative means for deciding polynomial boundedness of matrix interpretations) in [110, Theorem 5.2] and [76, Theorem 22]. We also call the criterion exhibited in Lemma 5.42 *EDA*.

It is also decidable in polynomial time whether a matrix interpretation is polynomially bounded with some specific degree:

Theorem 5.43 ([57, Theorem 3.3], see also [76, Theorem 14])**.** *Let $\mathcal{M}$ be a finite set of matrices from $K^{d\times d}$ for some $d \in \mathbb{N}$ and $K \in \{\mathbb{N}, \mathbb{Q}_0^+, \mathbb{R}_0^+\}$. Suppose that $\rho(\mathcal{M}) \leqslant 1$, and for no entry x of any matrix in $\mathcal{M}$, we have $0 < x < 1$. Let $1 \leqslant k \leqslant d$, and let $(i_1, j_1), \dots, (i_k, j_k)$ be k different pairs of indices such that for each pair (i_l, j_l), i_l and j_l are different and there exist matrix products $N = M_1 \cdot \ldots \cdot M_m$ and $N' = M'_1 \cdot \ldots \cdot M'_{m'}$ with $M_1, \dots M_m \in \mathcal{M}$, $M'_1, \dots, M'_{m'} \in \mathcal{M}$, $N_{i_l,i_l} \geqslant 1$, $N_{i_l,j_l} \geqslant 1$, and $N_{j_l,j_l} \geqslant 1$, and $l < k$ implies $N'_{j_l,i_{l+1}} \geqslant 1$. Then $\mathsf{growth}_{\mathcal{M}}(n) \in \Omega(n^k)$. On the other hand, if there exist no k different pairs of indices $(i_1, j_1), \dots, (i_k, j_k)$ satisfying the above conditions, then $\mathsf{growth}_{\mathcal{M}}(n) \in \mathsf{O}(n^{k-1})$.*

Let $\mathcal{A}$ be a matrix interpretation and $\mathcal{M}$ the set of matrices occurring in $\mathcal{A}$. Then the criterion exhibited in Theorem 5.43, which is equivalent to the property $\mathsf{growth}_{\mathcal{M}}(n) \in \Omega(n^k)$, essentially coincides with the criterion IDA_k for weighted automata, which is defined in [111], and applied for deciding polynomial ambiguity with degree k of weighted automata (and hence, as an alternative means for deciding polynomial boundedness with degree k of matrix interpretations) in [110, Proposition 7.4] and [76, Theorem 24]. We also call the criterion exhibited in Theorem 5.43 IDA_k.

Combining all of the above yields the following complexity results for matrix interpretations:

Corollary 5.44 (compare [76, Theorem 15], [76, Theorem 22], [110, Theorem 5.2])**.** *Let $\mathcal{A}$ be a strict matrix interpretation of dimension d, and let $\mathcal{M}$ be the set of matrices occurring in $\mathcal{A}$. If $\mathcal{M}$ does not satisfy EDA, then the reduction order $>_{\mathcal{A}}$ induces polynomial complexity, and the degree of the polynomial is d.*

Proof. Follows by Lemma 5.38, Theorem 5.41, and Lemma 5.42. □

Corollary 5.45. *Let $\mathcal{A}$ be a weak matrix interpretation of dimension d, and let $\mathcal{M}$ be the set of matrices occurring in $\mathcal{A}$. If $\mathcal{M}$ does not satisfy EDA, then the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces polynomial complexity, and the degree of the polynomial is d.*

Proof. Follows by Lemma 5.39, Theorem 5.41, and Lemma 5.42. □

Corollary 5.46 (compare [76, Theorem 16], [76, Theorem 24], and [110, Proposition 7.4])**.** *Let $\mathcal{A}$ be a strict matrix interpretation, and let $\mathcal{M}$ be the set of matrices occurring in $\mathcal{A}$. If there exists some $k \in \mathbb{N}$ such that $\mathcal{M}$ satisfies neither*

EDA nor IDA_k, then the reduction order $>_{\mathcal{A}}$ induces polynomial complexity, and the degree of the polynomial is k.

Proof. Follows by Lemmata 5.38 and 5.42, and Theorem 5.43. □

Corollary 5.47. *Let $\mathcal{A}$ be a weak matrix interpretation, and let $\mathcal{M}$ be the set of matrices occurring in $\mathcal{A}$. If there exists some $k \in \mathbb{N}$ such that $\mathcal{M}$ satisfies neither EDA nor IDA_k, then the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces polynomial complexity, and the degree of the polynomial is k.*

Proof. Follows by Lemmata 5.39 and 5.42, and Theorem 5.43. □

5.5 Experiments

We have implemented the methods described in this chapter, and tested their viability to analyse polynomial derivational complexity on version 8.0 of the Termination Problem Database (TPDB for short), which is used in the annual international termination competition[1]. In order to create our testbed from this database, we used the same restrictions as the complexity category of the termination competition used to obtain the basic set from which the TRSs used in the competition were selected: we filtered out duplicate instances of the same TRS (modulo strategy), and removed all TRSs with relative rewriting problems, conditional rules, or theory annotations. Moreover, since the derivational complexity of any duplicating TRS is at least exponential, we ignored these TRSs, as well. The result is a testbed containing a total of 1855 TRSs.

We briefly sketch our implementation of Δ-restricted and matrix interpretations; Section 7.3 gives some more details on the implementation of matrix interpretations: Similar to [16], we build a set of Diophantine constraints which express all necessary restrictions on the matrix interpretation. Then, we put a finite upper bound on the values of the variables in the constraints and encode these constraints as a problem of propositional logic (see [30], but also [27]). We give the final SAT problem to MiniSAT [25] and use a satisfying assignment to construct a suitable matrix interpretation.

In order to compare Δ-restricted interpretations (referred to by CDI), triangular matrix interpretations (TRI), and matrix interpretations which do not satisfy EDA or IDA_k for some $k \in \mathbb{N}$ (referred to by EDA and IDA, respectively) to other results, we compared them to the match-bound technique (BOUNDS for short). Note that by Theorem 4.57, match-boundedness implies linear derivational complexity for linear TRSs. Last, we specifically tested triangular matrix interpretations of dimension 2 (referred to by TRI2), which are (according to Theorems 5.24 and 5.35) the closest to Δ-restricted interpretations. For Δ-restricted and matrix interpretations, we restricted all coefficients to at most 3 (allowing us to use at most 2 bits to encode each coefficient). In order to obtain the results for CDI, we used the final version of our software `cdiprover3` (see [97] for a system description). For the CDI test, `cdiprover3`

[1]`http://termcomp.uibk.ac.at/`

was given the following command line arguments:

```
$ ./cdiprover3 -c deltarestricted -b 3 <file> <timeout>
```

Here `<file>` points to the file containing the considered TRS, and `<timeout>` is a timeout in seconds given to `cdiprover3`. For all other tests, we employed version 1.8 of TCT, which is described in Chapter 7. The strategies (see Section 7.2 for an explanation of how to use strategies in TCT) used for these tests are listed in Table 5.1.

All tests were executed on a server equipped with 8 AMD Opteron$^{\text{TM}}$ 2.8 GHz dual core processors with 64GB of RAM. We used a timeout of 60 seconds for each strategy and TRS. The results of the tests are shown in Table 5.2[2]. The times given in the table are seconds.

As we can see, in absolute numbers, BOUNDS is clearly the strongest of the analysed techniques. Still, as can be seen from the detailed results page, there are many systems which can not be handled by BOUNDS, but one of the other techniques. This is not surprising, since by Theorem 4.57, BOUNDS can only handle TRSs whose derivational complexity is bounded by a linear function.

On the other hand, as suggested by our results in Section 5.3, the systems that can be handled by CDI are a strict subset of the problems solved by TRI2. Due to directly encoding the constraints created by the matrices rather than handling all the calculations introduced by the parameter functions and the Δ, TRI2 is also much more efficient than CDI. This explains why the gap between CDI and TRI2 is much further than just suggested by Theorem 5.24.

It should be no surprise that TRI2 is weaker than TRI. Other than suggested by theory, the TRSs handled by TRI2 are not a subset of those handled by TRI. This is due to timeouts. However, there is only one TRS in the testbed which could be handled by TRI2, but not by TRI.

Finally, we can see that the numbers of systems handled by TRI, EDA, and IDA, are not too far from each other. Theoretically, the strategy used for EDA should be the strongest one, but timeouts (and size of the search space) change this relationship again.

In total, these results suggest that matrix interpretations, while absolutely weaker than match-bounds, are still a competitive direct method for proving polynomial upper bounds on the derivational complexity of TRSs. A combination of BOUNDS and a strategy based on matrix interpretations seems to be the best means for that end. Indeed, in the derivational complexity category of the latest international termination competition, both techniques were very prevalent, and they were the most used direct proof techniques in that competition.

5.6 Conclusion

In this chapter we studied the complexity induced by reduction orders and reduction pairs built upon matrix and context dependent interpretations. The

[2]See `http://cl-informatik.uibk.ac.at/users/aschnabl/experiments/thesis/poly/` for the full experimental evidence, and the used testbeds, tools, and ways to call the tools.

Test	Strategy
TRI2	`matrix :dim 2 :bound 3 :cert algebraic`
TRI	`fastest`
	`    (matrix :dim 1 :bound 3 :cbits 4 :cert algebraic)`
	`    (matrix :dim 2 :bound 3 :cbits 4 :cert algebraic)`
	`    (matrix :dim 3 :bound 3 :cbits 4 :cert algebraic)`
	`    (matrix :dim 4 :bound 3 :cbits 4 :cert algebraic)`
	`    (matrix :dim 5 :bound 3 :cbits 4 :cert algebraic)`
	`    (matrix :dim 6 :bound 3 :cbits 4 :cert algebraic)`
EDA	`fastest`
	`    (matrix :dim 1 :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 2 :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 3 :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 4 :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 5 :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 6 :bound 3 :cbits 4 :cert automaton)`
IDA	`fastest`
	`    (matrix :dim 1 :degree 1`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 2 :degree 1`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 2 :degree 2`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 3 :degree 1`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 3 :degree 2`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 3 :degree 3`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 4 :degree 1`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 4 :degree 2`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 4 :degree 3`
	`        :bound 3 :cbits 4 :cert automaton)`
	`    (matrix :dim 4 :degree 4`
	`        :bound 3 :cbits 4 :cert automaton)`
BOUNDS	`fastest (bounds :initial minimal :enrichment match)`
	`    (bounds :initial perSymbol :enrichment match)`

Table 5.1: Strategies used in the experiments of Section 5.5

	$\mathsf{O}(n)$	$\mathsf{O}(n^2)$	$\mathsf{O}(n^3)$	$\mathsf{O}(n^k)$	avg. succ. time
CDI	0	89	89	89	3.692
TRI2	33	157	157	157	0.464
TRI	66	179	204	210	1.999
EDA	41	151	187	195	3.285
IDA	53	168	177	177	2.901
BOUNDS	363	363	363	363	2.038

Table 5.2: Direct termination proof methods as complexity analysers

following diagram provides a condensed view of the studied classes of interpretations, where the right column gives the complexity induced by the respective reduction orders and reduction pairs. Here the arrows depict set inclusions, and the dashed arrows refer to the additional restriction on the zero columns demanded by Theorem 5.35.

$$\begin{tikzcd}
\Delta^2\text{-int.} \arrow[r, bend left, "\text{Thm 5.24}"] & \text{matrix (dim. 2)} \arrow[l, dashed, bend left, "\text{Thm 5.35}"] \arrow[r] & \text{matrix} & \textbf{exponential} \\
 & \text{TMI} \arrow[r] & \neg EDA/\neg IDA_k \arrow[u] & \textbf{polynomial} \\
\Delta\text{-restricted} \arrow[uu] \arrow[r, bend left, "\text{Thm 5.24}"] & \text{TMI (dim. 2)} \arrow[l, dashed, bend left, "\text{Thm 5.35}"] \arrow[u] \arrow[r] & \neg EDA \text{ (dim. 2)} \ \neg IDA_2 \arrow[u] \arrow[uul] & \textbf{quadratic}
\end{tikzcd}$$

We want to emphasise the pictured correspondence result: Consider Δ^2-interpretations and matrix interpretations of dimension 2 without any zero columns. These interpretations are *equivalent* for orientability. This correspondence sheds light on the expressivity of matrix interpretations and (significantly) extends the class of rewrite systems whose compatibility with context dependent interpretations can be shown automatically. As witnessed by Example 5.1, it is sometimes possible to automatically obtain a context dependent interpretation via the correspondence result, where the direct approach fails. Further, we have mentioned a full characterisation of matrix interpretations which are polynomially bounded, and for each $k \in \mathbb{N}$, a full characterisation of matrix interpretations which are polynomially bounded with degree k, by Middeldorp et al. [76].

The techniques mentioned in this chapter have been implemented. The experimental data shows the viability of the considered subclasses of matrix interpretations for automatically proving polynomial derivational complexity bounds of TRSs.

What we have not considered in this chapter are *multi-step termination proofs*, which often occur in the dependency pair framework, and hence in the proof output of many modern automatic termination provers. This is the subject of the next chapter.

Chapter 6

The Dependency Pair Framework and Derivational Complexity

The whole is more than the sum of its parts.

Aristotle

6.1 Introduction

For direct termination proof methods, a considerable number of results establish essentially optimal upper bounds on the derivational complexity of TRSs whose termination can be proved by the respective method. In Chapter 4, we mentioned a number of existing results about commonly used termination proof techniques. Chapter 5 listed some further such results about direct termination proof techniques which have been restricted specifically for complexity analysis.

In this chapter, we focus on the dependency pair framework. Beside the contents of this chapter, some results have been established there, as well, but all of them consider variations of the original definition of dependency pairs and/or bound other complexity measures than derivational complexity. For instance, [44, 45] introduce a notion called *weak dependency pairs* for proving upper bounds on the runtime complexity and innermost runtime complexity of TRSs. The essential syntactic difference between weak dependency pairs and dependency pairs as specified in Definition 2.47 is that all dependency pairs based on a rewrite rule are (roughly) merged into a single weak dependency pair. In [87], a similar variant of dependency pairs is used in order to obtain upper bounds on the innermost runtime complexity of TRSs. On the other hand, in [73], the computational (time and space) complexity of functions computed by TRSs whose initial DP problem is completely solved by reduction pair processors based on certain recursive path orders is investigated. However, at the same time, compatibility with the weak ordering based on a polynomial interpretation satisfying certain restrictions (such an interpretation is called a *quasi-interpretation* in [73]) is demanded. Finally, in [74], space complexity bounds are established on TRSs using a generalisation of quasi-interpretations called *sup-interpretations*, and *fraternities*, which share similarities with weak dependency pairs.

This chapter contains the first investigation into general upper bounds on the derivational complexity of TRSs whose termination can be proved using the original dependency pair method, as described in [2], or its generalisation, the dependency pair framework (cf. [35, 36, 103], for instance), with the most fundamental DP processors. Note that by Theorem 2.44, termination of any terminating TRS can be proved in the dependency pair framework. Therefore, any meaningful complexity analysis must restrict the set of considered DP processors. The investigations of this chapter are focused on the DP processors outlined in Section 2.4: reduction pair, dependency graph, subterm criterion, and usable rules processors. Moreover, any finite DP problem $(\mathcal{P}, \mathcal{R})$ can be completely solved by the reduction pair processor $\Phi^{\mathsf{RP}}_{(\to^*_{\mathcal{R}}, >)}$ or some usable rules processor $\Phi^{\mathsf{UR}}_{\mathcal{U},(\to^*_{\mathcal{R}}, >)}$ (for a suitable instance of $\mathcal{U}$), where $>$ is the transitive closure of $\xrightarrow{\epsilon}_{\mathcal{P}}/\to_{\mathcal{R}}$. Hence, all given complexity bounds are parametrised in the complexity induced by the reduction pairs used by the reduction pair and usable rules processors in the considered termination proof.

The next examples provide some motivation of our study.

Example 6.1. Consider the TRS $\mathcal{R}_{\mathsf{Hof2}}$ given below:

$$\begin{aligned} \mathsf{i}(x) \circ (y \circ z) &\to \mathsf{f}(x, \mathsf{i}(x)) \circ (\mathsf{i}(\mathsf{i}(y)) \circ z) & \mathsf{i}(x) &\to x \\ \mathsf{i}(x) \circ (y \circ (z \circ w)) &\to \mathsf{f}(x, \mathsf{i}(x)) \circ (z \circ (y \circ w)) & \mathsf{f}(x, y) &\to x \end{aligned}$$

$\mathcal{R}_{\mathsf{Hof2}}$ is a variation of a TRS encoding the Ackermann function, introduced by Hofbauer [50] (also compare [47]), which we call $\mathcal{R}_{\mathsf{Hof}}$ in Example 6.4 below. Note that the derivational complexity of $\mathcal{R}_{\mathsf{Hof2}}$ grows at least as fast as the Ackermann function (this follows by analogy to [50, Proposition 5], also compare [47, Proposition 5.9]).

Termination of $\mathcal{R}_{\mathsf{Hof2}}$ can be shown as follows using essentially the basic dependency pair method described in [2] in conjunction with the reduction pair based on a KBO, refined by an argument filtering. The set $\mathsf{DP}(\mathcal{R}_{\mathsf{Hof2}})$ contains nine dependency pairs:

$$\begin{aligned} \mathsf{i}(x) \circ^\sharp (y \circ z) &\to \mathsf{f}(x, \mathsf{i}(x)) \circ^\sharp (\mathsf{i}(\mathsf{i}(y)) \circ z) \\ \mathsf{i}(x) \circ^\sharp (y \circ z) &\to \mathsf{f}^\sharp(x, \mathsf{i}(x)) \\ \mathsf{i}(x) \circ^\sharp (y \circ z) &\to \mathsf{i}(\mathsf{i}(y)) \circ^\sharp z \\ \mathsf{i}(x) \circ^\sharp (y \circ z) &\to \mathsf{i}^\sharp(\mathsf{i}(y)) \\ \mathsf{i}(x) \circ^\sharp (y \circ z) &\to \mathsf{i}^\sharp(y) \\ \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) &\to \mathsf{f}(x, \mathsf{i}(x)) \circ^\sharp (z \circ (y \circ w)) \\ \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) &\to \mathsf{f}^\sharp(x, \mathsf{i}(x)) \\ \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) &\to z \circ^\sharp (y \circ w) \\ \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) &\to y \circ^\sharp w \end{aligned}$$

We set π to be the argument filtering with $\pi(\mathsf{f}) = \pi(\mathsf{f}^\sharp) = \pi(\mathsf{i}^\sharp) = 1$, $\pi(\mathsf{i}) = [1]$, and $\pi(\circ) = \pi(\circ^\sharp) = [1, 2]$. Let $(\geqslant_{\mathsf{KBO}(w,w_0)}, >_{\mathsf{KBO}(w,w_0)})$ be the reduction pair based on the KBO using the precedence $\mathsf{i} > \circ, \circ^\sharp$ and the admissible weight function (w, w_0) with $w_0 = 1$, $w(\circ) = w(\circ^\sharp) = 1$, and $w(\mathsf{i}) = 0$. Then the initial DP problem $(\mathsf{DP}(\mathcal{R}_{\mathsf{Hof2}}), \mathcal{R}_{\mathsf{Hof2}})$ is completely solved by the reduction pair $(\geqslant^\pi_{\mathsf{KBO}(w,w_0)}, >^\pi_{\mathsf{KBO}(w,w_0)})$.

Note that in Example 6.1, the TRS $\mathcal{R}_{\mathsf{Hof2}}$ is not compatible with any KBO, but we solve the initial DP problem by a reduction pair processor based on a KBO. In order to measure the strength of the basic dependency pair method (which is the same as completely solving the initial DP problem by a reduction pair in the notation of the DP framework), we express our inferred complexity bound relative to the induced complexities of the employed reduction pair. By Theorems 4.32 and 2.52, any reduction pair based on a KBO induces complexity $\mathsf{Ack}(\mathsf{O}(n), 0)$. By Theorem 2.56, the same holds for the reduction pair $(\geqslant^{\pi}_{\mathsf{KBO}(w,w_0)}, >^{\pi}_{\mathsf{KBO}(w,w_0)})$. As shown in Section 6.7 below, due to the presented termination proof, this is also an upper bound on $\mathsf{dc}_{\mathcal{R}_{\mathsf{Hof2}}}$. So at first glance, it may appear that the dependency pair method does not add any complexity-theoretic power.

Example 6.2. Let $\mathcal{R}_{\mathsf{Ack}}$ be the TRS defined by the following rules:

$$\begin{aligned} \mathsf{Ack}(0, y) &\to \mathsf{s}(y) & \mathsf{Ack}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{Ack}(x, \mathsf{Ack}(\mathsf{s}(x), y)) \\ \mathsf{Ack}(\mathsf{s}(x), 0) &\to \mathsf{Ack}(x, \mathsf{s}(0)) \end{aligned}$$

The TRS $\mathcal{R}_{\mathsf{Ack}}$ encodes the Ackermann function. Hence its derivational complexity grows faster than any primitive recursive function. Furthermore, it is easy to see that the derivational complexity of $\mathcal{R}_{\mathsf{Ack}}$ is bounded by a multiply recursive function.

We now give a termination proof of $\mathcal{R}_{\mathsf{Ack}}$. The set of dependency pairs of $\mathcal{R}_{\mathsf{Ack}}$ contains the following rules:

$$\begin{aligned} \mathsf{Ack}^\sharp(\mathsf{s}(x), 0) &\to \mathsf{Ack}^\sharp(x, \mathsf{s}(0)) & \mathsf{Ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{Ack}^\sharp(x, \mathsf{Ack}(\mathsf{s}(x), y)) \\ & & \mathsf{Ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{Ack}^\sharp(\mathsf{s}(x), y) \end{aligned}$$

We apply the subterm criterion processor $\Phi^{\mathsf{SC}}_{\pi_1}$ to the initial DP problem, where the simple projection π_1 is defined by $\pi_1(\mathsf{Ack}^\sharp) = 1$. The resulting set of DP problems $\Phi^{\mathsf{SC}}_{\pi_1}((\mathsf{DP}(\mathcal{R}_{\mathsf{Ack}}), \mathcal{R}_{\mathsf{Ack}}))$ consists of the single problem $(\mathcal{P}, \mathcal{R}_{\mathsf{Ack}})$, where $\mathcal{P} = \{\mathsf{Ack}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{Ack}^\sharp(\mathsf{s}(x), y)\}$. Continuing from $(\mathcal{P}, \mathcal{R}_{\mathsf{Ack}})$, an application of another subterm processor $\Phi^{\mathsf{SC}}_{\pi_2}$, where π_2 is defined by $\pi_2(\mathsf{Ack}^\sharp) = 2$, yields the DP problem $(\emptyset, \mathcal{R}_{\mathsf{Ack}})$, which is trivially finite. Thus termination of $\mathcal{R}_{\mathsf{Ack}}$ follows.

As an alternative termination proof consider the LPO defined by the precedence $\mathsf{Ack} > \mathsf{s}$. Then the TRS $\mathcal{R}_{\mathsf{Ack}}$ is compatible with $>_{\mathsf{LPO}}$, hence termination of $\mathcal{R}_{\mathsf{Ack}}$ follows by Theorem 4.41. Moreover, by Theorem 4.42, $\mathsf{dc}_{\mathcal{R}_{\mathsf{Ack}}}$ is bounded by a multiply recursive function.

Note the contrast between the high derivational complexity of $\mathcal{R}_{\mathsf{Ack}}$ (as stated above, it is not primitive recursive) and the apparent simplicity of its first termination proof: only subterm criterion processors were needed, and two applications of subterm criterion processors sufficed. Other than Example 6.1, this gives the impression that even for simple DP processors, the complexity-theoretic power added by the DP framework is already huge.

So, as shown in Examples 6.1 and 6.2, it seems not at all obvious what the complexity-theoretic power added by various applications of the DP framework is. The results shown in this chapter provide a clearer picture with respect to that. We establish the following (technical) results about upper bounds on the derivational complexity of TRSs whose termination is provable in the DP framework:

1. If we can show termination of a TRS in the DP framework using reduction pair (based on reduction pairs which induce multiply recursive complexity only), dependency graph, and subterm criterion processors in any combination, then the derivational complexity of the considered TRS is bounded by a multiply recursive function.

2. If we can prove termination of a TRS by completely solving the DP problem related to each SCC of the dependency graph of the initial DP problem by a reduction pair processor (this is essentially the same as the basic dependency pair method of [2] with the dependency graph refinement), then the derivational complexity of the considered TRS is bounded primitive recursively in the maximum of the complexities induced by the applied reduction pairs.

3. For a TRS whose termination is proved by completely solving the initial DP problem with a reduction pair processor (this is the basic dependency pair method as described in [2]), its derivational complexity is bounded double exponentially in the complexity induced by the applied reduction pair. If the considered TRS is also a SRS, then its derivational complexity is bounded exponentially in the complexity induced by the reduction pair.

4. Finally, if a usable rules processor can completely solve the initial DP problem of a TRS, then its derivational complexity is bounded by an iteration of a function which is elementary in the complexity induced by the considered reduction pair (this is always primitive recursive in the complexity induced by the reduction pair).

Complementing these results, we exhibit examples which show that all of the above mentioned complexity bounds are essentially optimal. Note that Result 3 is an improvement of the upper bound shown in [80, Section 4] and [82, Section 5].

To exemplify these results, we momentarily focus on reduction pairs based on matrix interpretations. By Theorem 4.20, any such reduction pair induces exponential complexity. Let $\mathcal{R}$ be a TRS and suppose termination of $\mathcal{R}$ has been established by applying the basic dependency pair method, where matrix interpretations are used to define the employed reduction pair. According to Result 3 the derivational complexity function of $\mathcal{R}$ is bounded by $2_3(\mathsf{O}(n))$, i.e, triple exponentially in n. On the other hand, if in addition the dependency graph or usable rules refinement is used, then Results 2 and 4 yield that the derivational complexity of $\mathcal{R}$ is bounded by a primitive recursive function. Finally, if termination of $\mathcal{R}$ is established in the dependency pair framework by multiple applications of such reduction pair processors, then by Result 1, $\mathsf{dc}_{\mathcal{R}}$ is bounded multiply recursively.

Thus seemingly easy refinements of the dependency pair method like dependency graphs or, more prominently, multi-step termination proofs (which are typical for the dependency pair framework), may lead to noteworthy speed-ups of the growth rates of the derivational complexity function of the TRSs whose termination can be proved. On the other hand if strong techniques (with respect to the complexity induced by the corresponding reduction pairs) are employed in conjunction with the dependency pair method, then the derivational complexity of the analysed TRS may only depend on the complexity induced by the reduction pairs. This is illustrated by Example 6.1, where the employed reduction pair is based on a KBO. Since the complexity induced by such reduction pairs is closed under primitive recursion, the inherent complexity-theoretic power of the basic dependency pair method, even when refined by argument filtering and either a dependency graph or usable rules, becomes negligible in comparison.

Finally, the results (in particular, Result 1) given in this chapter can be understood as negative results: all of them imply multiply recursive complexity bounds as long as the employed reduction pairs induce multiply recursive complexity. Kindly note that this assumption on the complexity induced by the used reduction pairs is rather weak, as all classes of reduction pairs which have currently been mechanised, and whose induced complexity has been analysed, induce multiply recursive complexity. Using only processors based on such reduction pairs, dependency graph processors, and subterm criterion processors, it is theoretically impossible to prove termination of any TRS whose derivational complexity is not bounded by a multiply recursive function. Most prominently, this includes TRSs encoding the well-known Battle between Hercules and the Hydra [59]. Such TRSs can be found in [21, 106, 23, 78], for example. On the other hand, our result immediately turns automatic termination provers into automatic complexity provers, albeit rather weak ones. Furthermore it provides the basis for further investigations into termination proof techniques which imply tighter upper bounds on the derivational complexities of TRSs whose termination they can prove.

Note the challenges of our investigation: in order to estimate the derivational complexity of a TRS, we only consider the complexities induced by the employed reduction pairs after roughly limiting the considered classes of DP processors. This entails that we exploit the upper bound on the maximal number of dependency pair steps (or even of just certain subsets of the dependency pair steps) to bound the length of derivations.

The main idea in the proof of Result 1 is the embedding of the considered TRS into a generic simulating TRS. This embedding is based on the DP processors used to establish termination of the considered TRS. The derivational complexity of the simulating TRS can then be analysed directly. The proofs of Results 2 and 3 are built up in a similar fashion. However, due to the stricter assumptions on the considered termination proof, the simulating TRS can be simplified for these results. As a consequence, it becomes possible to establish the tighter complexity bounds stated in Results 2 and 3. Finally, the complexity bound given in Result 4 is essentially an iteration of the upper bound from Result 3. It is established by a close inductive inspection of the mapping

$\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}$ given in Definition 2.59, which is a crucial part of the soundness proof for usable rules processors. Here, Result 3 is applied once in each inductive step.

6.2 The DP Framework: a Lower Complexity Bound

For the next two sections, we consider TRSs whose termination can be proved by the following theorem:

Theorem 6.3. *Let $\mathcal{R}$ be a TRS such that there exists a proof tree* PT *of $\mathcal{R}$. Suppose that each edge label of* PT *is a reduction pair, dependency graph, or subterm criterion processor. Then $\mathcal{R}$ is terminating.*

Proof. By Theorems 2.50, 2.68, and 2.71, reduction pair, dependency graph, and subterm criterion processors are sound. By assumption, for each inner node $(\mathcal{P},\mathcal{R})$ of PT, all edges leading from $(\mathcal{P},\mathcal{R})$ to a child node of $(\mathcal{P},\mathcal{R})$ are labelled by some reduction pair, dependency graph, or subterm criterion processor Φ. Hence, for each node $(\mathcal{P},\mathcal{R})$ of PT, finiteness of all children of $(\mathcal{P},\mathcal{R})$ implies finiteness of $(\mathcal{P},\mathcal{R})$. By Lemma 2.45, all leaf nodes of PT are finite. Hence, the root node $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$ of PT is also finite, and thus, by Theorem 2.44, $\mathcal{R}$ is terminating. $\square$

From the viewpoint of derivational complexity, this is a rather powerful theorem: as shown in the next example, there exist TRSs whose termination can be shown via Theorem 6.3 (using only reduction pairs which induce linear complexity), but whose derivational complexity cannot be bounded by a primitive recursive function.

Example 6.4. Consider the following TRS $\mathcal{R}_{\mathsf{Hof}}$, taken from [50, 47]:

$$\mathsf{i}(x) \circ (y \circ z) \to x \circ (\mathsf{i}(\mathsf{i}(y)) \circ z) \qquad \mathsf{i}(x) \circ (y \circ (z \circ w)) \to x \circ (z \circ (y \circ w))$$

It is shown in [50, Proposition 5] that $\mathsf{dc}_{\mathcal{R}_{\mathsf{Hof}}}$ is not primitive recursive as the system encodes the Ackermann function.

Following Endrullis et al. [27, Example 11] we show termination of $\mathcal{R}_{\mathsf{Hof}}$ employing Theorem 6.3. The dependency pairs of $\mathcal{R}_{\mathsf{Hof}}$ are:

$$\begin{array}{rc}
1: & \mathsf{i}(x) \circ^\sharp (y \circ z) \to x \circ^\sharp (\mathsf{i}(\mathsf{i}(y)) \circ z) \\
2: & \mathsf{i}(x) \circ^\sharp (y \circ z) \to \mathsf{i}(\mathsf{i}(y)) \circ^\sharp z \\
3: & \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) \to x \circ^\sharp (z \circ (y \circ w)) \\
4: & \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) \to z \circ^\sharp (y \circ w) \\
5: & \mathsf{i}(x) \circ^\sharp (y \circ (z \circ w)) \to y \circ^\sharp w
\end{array}$$

First, consider the polynomial interpretation $\mathcal{A}$ defined by the interpretation functions $\circ^\sharp_{\mathcal{A}}(x,y) = y$, $\circ_{\mathcal{A}}(x,y) = y+1$, and $\mathsf{i}_{\mathcal{A}}(x) = 0$. An application of $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}},>_{\mathcal{A}})}$ yields $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}},>_{\mathcal{A}})}((\{1,2,3,4,5\},\mathcal{R})) = \{(\{1,3\},\mathcal{R})\}$, so effectively, it removes the dependency pairs $\{2,4,5\}$ from the initial DP problem. Next, we apply the reduction pair based on the polynomial interpretation $\mathcal{B}$ defined by the interpretation functions $\circ^\sharp_{\mathcal{B}}(x,y) = x$, $\circ_{\mathcal{B}}(x,y) = 0$, and $\mathsf{i}_{\mathcal{B}}(x) = x+1$, which completely solves the remaining DP problem: we have $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{B}},>_{\mathcal{B}})}((\mathcal{P},\mathcal{R})) = \{(\emptyset,\mathcal{R})\}$. Putting everything together, we obtain a proof tree which uses only $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}},>_{\mathcal{A}})}$ and $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{B}},>_{\mathcal{B}})}$. Thus we conclude termination of $\mathcal{R}$ by Theorem 6.3.

Note the following particularity of Example 6.4: On one hand, the derivational complexity function $\mathsf{dc}_{\mathcal{R}_{\mathsf{Hof}}}$ grows faster than any primitive recursive function. On the other hand, by Theorem 4.13, both reduction pairs used in the presented termination proof of $\mathcal{R}_{\mathsf{Hof}}$ induce linear complexity. We now go a step further and show that for any multiply recursive function f, there exists a TRS $\mathcal{R}$ which can be proved terminating by Theorem 6.3 (without even using any reduction pairs), but whose derivational complexity grows faster than f.

Example 6.5. Let $k \geqslant 2$ and consider the following schematic rewrite rules over the signature containing the k-ary function symbol Ack_k, the unary function symbol s, and the constant function symbol $\mathsf{0}$, denoted as $\mathcal{R}_{\mathsf{Pet}}(k)$. It is easy to see that for fixed k, the TRS $\mathcal{R}_{\mathsf{Pet}}(k)$ encodes the k-ary Ackermann function. Also note that $\mathcal{R}_{\mathsf{Pet}}(2)$ and the TRS $\mathcal{R}_{\mathsf{Ack}}$ from Example 6.2 coincide.

$$\begin{aligned}
\mathsf{Ack}_k(\mathsf{0},\ldots,\mathsf{0},l_k) &\to \mathsf{s}(l_k)\\
\mathsf{Ack}_k(l_1,\ldots,\mathsf{s}(l_{k-1}),\mathsf{0}) &\to \mathsf{Ack}_k(l_1,\ldots,l_{k-1},\mathsf{s}(\mathsf{0}))\\
\mathsf{Ack}_k(l_1,\ldots,\mathsf{s}(l_{k-1}),\mathsf{s}(l_k)) &\to \mathsf{Ack}_k(l_1,\ldots,l_{k-1},\mathsf{Ack}_k(l_1,\ldots,\mathsf{s}(l_{k-1}),l_k))\\
\mathsf{Ack}_k(l_1,\ldots,\mathsf{s}(l_i),\mathsf{0},\ldots,\mathsf{0},l_k) &\to \mathsf{Ack}_k(l_1,\ldots,l_i,l_k,\mathsf{0},\ldots,\mathsf{0},l_k)
\end{aligned}$$

Here, the last rule is a schema instantiated for all $1 \leqslant i \leqslant k-2$.

Let π_i for $1 \leqslant i \leqslant k$ be the simple projection defined by $\pi_i(\mathsf{Ack}_k) = i$. Finiteness of $(\mathsf{DP}(\mathcal{R}_{\mathsf{Pet}}(k)),\mathcal{R}_{\mathsf{Pet}}(k))$ can be shown by sequentially applying the subterm criterion processors $\Phi^{\mathsf{SC}}_{\pi_1},\ldots,\Phi^{\mathsf{SC}}_{\pi_k}$. This yields a proof tree using only the edge labels $\Phi^{\mathsf{SC}}_{\pi_1},\ldots,\Phi^{\mathsf{SC}}_{\pi_k}$. Thus, $\mathcal{R}_{\mathsf{Pet}}(k)$ is terminating by Theorem 6.3.

Lemma 6.6. *For any multiple recursive function f, there exists a TRS $\mathcal{R}$ whose derivational complexity function $\mathsf{dc}_{\mathcal{R}}$ majorises f. Furthermore termination of $\mathcal{R}$ follows by an application of Theorem 6.3, and the termination proof employs no reduction pairs.*

Proof. By Theorem 2.16, there exists some $k \in \mathbb{N}$ such that f is bounded by the k-ary Ackermann function, and hence by $\mathsf{dc}_{\mathcal{R}_{\mathsf{Pet}}(k)}$. By Example 6.5, termination of $\mathcal{R}_{\mathsf{Pet}}(k)$ follows from Theorem 6.3. □

6.3 The DP Framework: an Upper Complexity Bound

Examples 6.4 and 6.5 show that the lowest possible upper bound on the derivational complexity of TRSs whose termination can be proved by Theorem 6.3, even for massive restrictions on the used reduction pairs, is the class of multiply recursive functions. In this section, we show that this is indeed an upper bound on the derivational complexity of TRSs from this class, provided that each of the reduction pairs used in the termination proof induces multiply recursive complexity, as well. The next definition formalises this side condition, and also plays a technical role the proof of the multiply recursive upper bound.

Definition 6.7. Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.3, and PT the proof tree employed by the theorem. Let k be a natural number such that for any dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$ used as an edge label in PT, the number of SCCs in $\mathcal{G}$ is less than or equal to k. Let $\mathcal{C}$ be a class of number theoretic functions such that for every reduction pair processor $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$ which is an edge label in PT, the reduction pair $(\succcurlyeq, \succ)$ induces complexity $\mathcal{C}$. Then we call a monotone function $g \colon \mathbb{N} \to \mathbb{N}$ a *reduction pair function* of PT if it satisfies

$$g(n) \geqslant \max(\{k\} \cup \{f(n) \mid f \in \mathcal{C}\}) \ .$$

Note that some reduction pair function can often be computed just by inspection of the reduction pair and dependency graph processors used as edge labels in PT. Moreover, most of the known reduction pairs (in particular, those reduction pairs presented in Chapter 4, and further, all reduction pairs currently applied by automatic termination provers) induce multiply recursive complexity. Hence, for many proof trees occurring in practise, it easy to find a multiply recursive reduction pair function.

Example 6.8 (continued from Examples 6.4 and 6.5)**.** Let $\mathsf{PT}_{\mathsf{Hof}}$ be the proof tree constructed from the termination proof of $\mathcal{R}_{\mathsf{Hof}}$ given in Example 6.4, and let $\mathsf{PT}_{\mathsf{Pet}}(k)$ be the proof tree constructed from the termination proof of $\mathcal{R}_{\mathsf{Pet}}(k)$ given in Example 6.5 for any $k \in \mathbb{N}$. Then the linear function $f(n) = n$ is a reduction pair function of $\mathsf{PT}_{\mathsf{Hof}}$, since both reduction pairs induce complexity $\{f\}$ (this is because both reduction pairs used in $\mathsf{PT}_{\mathsf{Hof}}$ are based on strongly linear interpretations using only the constants 0 and 1). On the other hand, for no $k \in \mathbb{N}$, the proof tree $\mathsf{PT}_{\mathsf{Pet}}(k)$ uses any reduction pair or dependency graph processors. Hence, the function $g(n) = 0$ is a reduction pair function of $\mathsf{PT}_{\mathsf{Pet}}(k)$.

The remainder of this section is devoted to the proof of the following theorem:

Theorem 6.9. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.3 using a proof tree PT, and suppose that there exists a multiply recursive reduction pair function of PT. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded by a multiply recursive function.*

The proof of Theorem 6.9 makes use of a combinatorial argument, of which we present a general overview now. To prove the theorem, we essentially use three different ideas. First, we exploit the given proof tree PT. We observe that, if we restrict our attention to termination of single terms, we can essentially focus on specific branches of the proof tree. Secondly, we define a TRS $\mathcal{R}_{\mathsf{sim}}$ simulating the initial TRS $\mathcal{R}$. Here, "simulating" means that $s \to_{\mathcal{R}} t$ implies $\mathsf{tr}(s) \to^{+}_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(t)$, where tr denotes a suitable interpretation of terms into the signature of the simulating TRS $\mathcal{R}_{\mathsf{sim}}$. The term $\mathsf{tr}(t)$ aggregates the termination arguments for t given by the DP processors in the branch of the proof tree which has been identified as particularly relevant for t in the first step. Finally, $\mathcal{R}_{\mathsf{sim}}$ will be simple enough to be compatible with a LPO. Hence, we employ Theorem 4.42 to deduce a multiply recursive upper bound on $\mathsf{dc}_{\mathcal{R}_{\mathsf{sim}}}$, and thus on $\mathsf{dc}_{\mathcal{R}}$.

We start the proof of Theorem 6.9 with some preliminary definitions.

Notation 6.10. For the remainder of this section, we fix a TRS $\mathcal{R}$ such that termination of $\mathcal{R}$ follows by Theorem 6.3 using some proof tree PT, and there exists a multiply recursive reduction pair function of PT. We assume without loss of generality for PT that no node $(\mathcal{P}, \mathcal{R})$ of PT has a child node $(\mathcal{P}, \mathcal{R})$ (if we did have such a node in PT, another proof tree of $\mathcal{R}$ could be obtained by just pruning PT).

For each of the DP processors Φ considered here, the following facts are obvious: $(\mathcal{P}', \mathcal{R}') \in \Phi((\mathcal{P}, \mathcal{R}))$ implies $\mathcal{P}' \subset \mathcal{P}$ and $\mathcal{R}' = \mathcal{R}$. Therefore, we assume throughout the rest of this section that for each DP problem $(\mathcal{P}, \mathcal{R})$, $\mathcal{P} \subseteq \mathsf{DP}(\mathcal{R})$. In particular, each rule in $\mathcal{P}$ has the shape $s^\sharp \to t^\sharp$ for some $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Moreover, $(\mathcal{P}', \mathcal{R}) \in \Phi((\mathcal{P}, \mathcal{R}))$, $(\mathcal{P}'', \mathcal{R}) \in \Phi((\mathcal{P}, \mathcal{R}))$, and $\mathcal{P}' \neq \mathcal{P}''$ imply $\mathcal{P}' \cap \mathcal{P}'' = \emptyset$. Therefore, each dependency pair can only appear in a single branch of PT.

We order the SCCs of a dependency graph by assigning a *rank* to each of them:

Definition 6.11. Let $\mathcal{G}$ be a dependency graph; we order the SCCs of $\mathcal{G}$ by assigning a *rank* to each of them. Let $\mathcal{P}$ and $\mathcal{Q}$ denote two distinct SCCs of $\mathcal{G}$. We call $\mathcal{Q}$ *reachable* from $\mathcal{P}$ if there exist nodes $u \in \mathcal{P}$, $v \in \mathcal{Q}$ and a path in $\mathcal{G}$ from u to v. Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ be the SCCs of $\mathcal{G}$. Let $\mathsf{rk}_\mathcal{G} \colon \{\mathcal{Q}_1, \ldots, \mathcal{Q}_k\} \to \{1, \ldots, k\}$ be an arbitrary, but fixed bijective mapping respecting the topological ordering of $\mathcal{G}$, i.e. $\mathsf{rk}_\mathcal{G}(\mathcal{Q}_i) > \mathsf{rk}_\mathcal{G}(\mathcal{Q}_j)$ whenever $\mathcal{Q}_j$ is reachable from $\mathcal{Q}_i$. We call $\mathsf{rk}_\mathcal{G}(\mathcal{P})$ the *rank* of an SCC $\mathcal{P}$ in $\mathcal{G}$. The *rank of a dependency pair* $l \to r$, denoted by $\mathsf{rk}_\mathcal{G}(l \to r)$, is the rank of $\mathcal{P}$ in $\mathcal{G}$ such that $l \to r \in \mathcal{P}$. Finally, the *rank of a term* t such that $t^\sharp \notin \mathsf{NF}(\mathcal{P}/\mathcal{R})$ for some SCC $\mathcal{P}$ of $\mathcal{G}$ is defined by

$$\mathsf{rk}_\mathcal{G}(t) = \max\{\mathsf{rk}_\mathcal{G}(l \to r) \mid \text{there exists } \sigma \text{ such that } t^\sharp \to_\mathcal{R}^* l\sigma\} \ .$$

Observe that $\mathsf{rk}_\mathcal{G}(t)$ need not be defined, although t has a redex at the root position. This is due to the fact that this redex need not be governed by a dependency pair. On the other hand, observe that if $t^\sharp \notin \mathsf{NF}(\mathcal{P}/\mathcal{R})$ for some SCC $\mathcal{P}$ of $\mathcal{G}$, then $\mathsf{rk}_\mathcal{G}(t)$ is defined. Furthermore in this case $\mathsf{rk}_\mathcal{G}(t) > 0$ and $\mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_\mathcal{P}/\to_\mathcal{R}) > 0$.

We now change the definition of proof trees to better suit our needs. The main change is that for dependency graph processors $\Phi^{\mathsf{DG}}_\mathcal{G}$, all SCCs of $\mathcal{G}$ are taken into account (not just, as usual, the nontrivial ones). While termination of trivial SCCs follows trivially, they might still form a bridge between non-trivial SCCs in a dependency graph, thus crucially increasing the total length of derivations. Example 6.13 below illustrates this. Moreover, as mentioned above, we use the proof tree to track its currently relevant part with respect to showing termination of a given term. This relevant part may very well include the DP problem belonging to a trivial SCC of a dependency graph.

Definition 6.12. A *complexity proof tree* PT of $\mathcal{R}$ is a tree satisfying:

1. The nodes of PT are DP problems, and $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is the root of PT.

2. For every inner node $(\mathcal{P}, \mathcal{R})$ in PT, there exists a sound DP processor Φ such that for each DP problem $(\mathcal{Q}, \mathcal{R}) \in \Phi((\mathcal{P}, \mathcal{R}))$, there exists an edge from $(\mathcal{P}, \mathcal{R})$ to $(\mathcal{Q}, \mathcal{R})$ in PT labelled by Φ.

$$\begin{tikzcd}
1 \arrow[loop above] & 3 \arrow[loop above] \arrow[d] & 5 \arrow[loop above] \arrow[d] \\
 & 2 \arrow[ul] & 4 \arrow[ul] \arrow[l]
\end{tikzcd}$$

Figure 6.1: Dependency Graph of the DP problem $(\mathsf{DP}(\mathcal{R}_{\mathsf{sup}}), \mathcal{R}_{\mathsf{sup}})$

3. Further, suppose that Φ is a dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$. Then there exists an edge from $(\mathcal{P}, \mathcal{R})$ to a leaf $(\mathcal{Q}, \mathcal{R})$ (labelled by Φ) for every trivial SCC $\mathcal{Q}$ of $\mathsf{DG}(\mathcal{P}, \mathcal{R})$. Moreover the successors of $(\mathcal{P}, \mathcal{R})$ are ordered from left to right in decreasing order with respect to the function $\mathsf{rk}_{\mathcal{G}}$.

The positions of nodes in a complexity proof tree PT are defined as usual as finite sequences of numbers. We write Greek letters for positions in PT. It is easy to verify that there is a one-to-one correspondence between proof trees according to Definition 2.46 and complexity proof trees according to Definition 6.12. We extend the definition of reduction pair functions directly to complexity proof trees via this correspondence, and hence use Definition 6.7 for complexity proof trees, as well.

Example 6.13. Consider the TRS $\mathcal{R}_{\mathsf{sup}}$ given by the following set of rewrite rules:

$$\begin{aligned}
\mathsf{d}(\mathsf{0}) &\to \mathsf{0} & \mathsf{e}(\mathsf{s}(x), y) &\to \mathsf{e}(x, \mathsf{d}(y)) \\
\mathsf{d}(\mathsf{s}(x)) &\to \mathsf{s}(\mathsf{s}(\mathsf{d}(x))) & \mathsf{sup}(\mathsf{s}(x), \mathsf{e}(\mathsf{0}, y)) &\to \mathsf{sup}(x, \mathsf{e}(y, \mathsf{s}(\mathsf{0})))
\end{aligned}$$

The dependency pairs $\mathsf{DP}(\mathcal{R}_{\mathsf{sup}})$ of $\mathcal{R}_{\mathsf{sup}}$ are:

$$\begin{aligned}
&1\colon\quad \mathsf{d}^\sharp(\mathsf{s}(x)) \to \mathsf{d}^\sharp(x) \\
&2\colon\ \mathsf{e}^\sharp(\mathsf{s}(x), y) \to \mathsf{d}^\sharp(y) && 4\colon\ \mathsf{sup}^\sharp(\mathsf{s}(x), \mathsf{e}(\mathsf{0}, y)) \to \mathsf{e}^\sharp(y, \mathsf{s}(\mathsf{0})) \\
&3\colon\ \mathsf{e}^\sharp(\mathsf{s}(x), y) \to \mathsf{e}^\sharp(x, \mathsf{d}(y)) && 5\colon\ \mathsf{sup}^\sharp(\mathsf{s}(x), \mathsf{e}(\mathsf{0}, y)) \to \mathsf{sup}^\sharp(x, \mathsf{e}(y, \mathsf{s}(\mathsf{0})))
\end{aligned}$$

We now prove termination of $\mathcal{R}_{\mathsf{sup}}$ by Theorem 6.3. Let $\mathcal{G}$ be the dependency graph $\mathsf{DG}(\mathsf{DP}(\mathcal{R}_{\mathsf{sup}}), \mathcal{R}_{\mathsf{sup}})$, which is the graph shown in Figure 6.1.

We start by applying the dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$ to the initial DP problem $(\mathsf{DP}(\mathcal{R}_{\mathsf{sup}}), \mathcal{R}_{\mathsf{sup}})$. The dependency graph $\mathcal{G}$ contains the three non-trivial SCCs $\{1\}$, $\{3\}$, and $\{5\}$, and the two trivial SCCs $\{2\}$ and $\{4\}$. Therefore, we have $\Phi^{\mathsf{DG}}_{\mathcal{G}}(\mathsf{DP}(\mathcal{R}_{\mathsf{sup}}), \mathcal{R}_{\mathsf{sup}}) = \{(\{1\}, \mathcal{R}_{\mathsf{sup}}), (\{3\}, \mathcal{R}_{\mathsf{sup}}), (\{5\}, \mathcal{R}_{\mathsf{sup}})\}$, so there are three smaller DP problems left to be shown finite. For all three of these remaining DP problems, we use the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ based on the linear interpretation $\mathcal{A}$, which is defined by the interpretation functions $\mathsf{s}_{\mathcal{A}}(x) = x + 1$, $\mathsf{0}_{\mathcal{A}} = 0$, $\mathsf{d}_{\mathcal{A}}(x) = 2x$, $\mathsf{e}_{\mathcal{A}}(x, y) = 0$, $\mathsf{sup}_{\mathcal{A}}(x, y) = 0$, and $\mathsf{d}^\sharp_{\mathcal{A}}(x) = \mathsf{e}^\sharp_{\mathcal{A}}(x, y) = \mathsf{sup}^\sharp_{\mathcal{A}}(x, y) = x$. For all three DP problems $(\mathcal{Q}, \mathcal{R}_{\mathsf{sup}})$ with $\mathcal{Q} \in \{\{1\}, \{3\}, \{5\}\}$, we have $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}((\mathcal{Q}, \mathcal{R}_{\mathsf{sup}})) = \{\emptyset, \mathcal{R}_{\mathsf{sup}}\}$.

From the above considerations, a proof tree of $\mathcal{R}_{\mathsf{sup}}$, and hence also a complexity proof tree $\mathsf{PT}_{\mathsf{sup}}$ of $\mathcal{R}_{\mathsf{sup}}$ can be directly constructed. Figure 6.2 shows

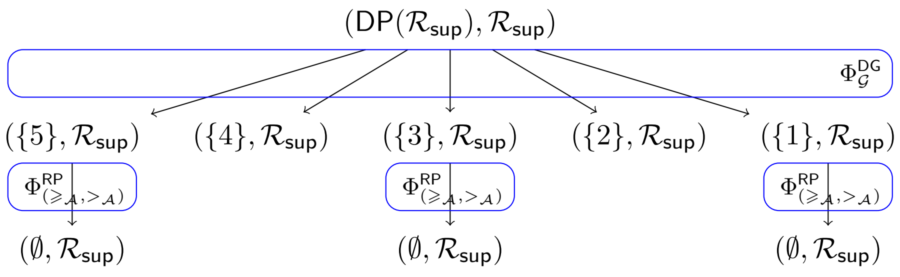
$(\mathsf{DP}(\mathcal{R}_{\mathsf{sup}}), \mathcal{R}_{\mathsf{sup}})$
$\Phi^{\mathsf{DG}}_{\mathcal{G}}$
$(\{5\}, \mathcal{R}_{\mathsf{sup}})$
$(\{4\}, \mathcal{R}_{\mathsf{sup}})$
$(\{3\}, \mathcal{R}_{\mathsf{sup}})$
$(\{2\}, \mathcal{R}_{\mathsf{sup}})$
$(\{1\}, \mathcal{R}_{\mathsf{sup}})$
$\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$
$\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$
$\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$
$(\emptyset, \mathcal{R}_{\mathsf{sup}})$
$(\emptyset, \mathcal{R}_{\mathsf{sup}})$
$(\emptyset, \mathcal{R}_{\mathsf{sup}})$

Figure 6.2: A complexity proof tree of $\mathcal{R}_{\mathsf{sup}}$

the complexity proof tree $\mathsf{PT}_{\mathsf{sup}}$, where we make use of a simplified notation for edge labels. The nodes at positions 11, 31, and 51 are leaves in this proof tree because they are labelled by the DP problem $(\emptyset, \mathcal{R}_{\mathsf{sup}})$, which is trivially finite. The nodes at positions 2 and 4 are leaves because $\{2\}$ and $\{4\}$ are trivial SCCs of the dependency graph $\mathcal{G}$. The following derivation illustrates the importance of trivial SCCs in our analysis:

$$\mathsf{e}^\sharp(\mathsf{s}^{n+1}(0), \mathsf{s}(0)) \to^*_{\{3\}\cup\mathcal{R}_{\mathsf{sup}}} \mathsf{e}^\sharp(\mathsf{s}(0), \mathsf{s}^{2^n}(0)) \to_{\{2\}} \mathsf{d}^\sharp(\mathsf{s}^{2^n}(0)) \to^*_{\{1\}\cup\mathcal{R}_{\mathsf{sup}}} \mathsf{s}^{2^{n+1}}(0)$$

Observe that the step within the trivial SCC $\{2\}$ connects two subderivations using the (otherwise unconnected) SCCs $\{3\}$ and $\{1\}$, thus increasing the length of the total derivation. While the length of each of these subderivations is only linear in the size of its respective starting term, the length of the total derivation is exponential in the size of its starting term. In order to capture this behaviour, we keep track of trivial SCCs in complexity proof trees.

Notation 6.14. For the remainder of this section, we fix PT to be the complexity proof tree corresponding to the proof tree used to prove termination of $\mathcal{R}$ via Theorem 6.3. Further, we fix a multiply recursive reduction pair function g of PT (by assumption, such a reduction pair function exists). Let d be the height of PT plus one.

As stated in the proof plan, we now determine which part of the termination proof is active with respect to a given term. Intuitively, for many terms, only a part of PT is relevant for showing termination of that particular term. More specifically, for any term t, only a certain subset of the dependency pairs can be used for rewriting $t^\sharp$. Of these dependency pairs, we view the one occurring in the leftmost positions of PT (with respect to the order of PT) as the *current dependency pair*. We call the sequence of positions in which the current dependency pair occurs, the *current path of t in* PT.

Example 6.15 (continued from Example 6.13)**.** Consider the terms $t_1 = \mathsf{sup}(\mathsf{s}(0), \mathsf{e}(0, \mathsf{s}(0)))$, $t_2 = \mathsf{sup}(0, \mathsf{e}(\mathsf{s}(0), \mathsf{s}(0)))$, and $t_3 = \mathsf{e}(\mathsf{s}(0), \mathsf{s}(0))$. We have the following derivations: $t_1^\sharp \to_{\mathsf{DP}(\mathcal{R}_{\mathsf{sup}})} t_2^\sharp$ and $t_1^\sharp \to_{\mathsf{DP}(\mathcal{R}_{\mathsf{sup}})} t_3^\sharp$. Hence the term $t_1^\sharp$ is neither a normal form of $\{5\}/\mathcal{R}_{\mathsf{sup}}$ nor of $\{4\}/\mathcal{R}_{\mathsf{sup}}$. Similarly, $t_3^\sharp$ is not a

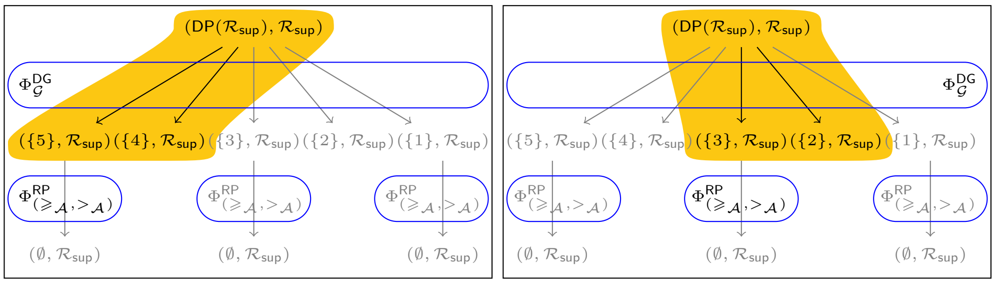
$(\mathsf{DP}(\mathcal{R}_{\mathsf{sup}}), \mathcal{R}_{\mathsf{sup}})$
$\Phi^{\mathsf{DG}}_{\mathcal{G}}$
$(\{5\}, \mathcal{R}_{\mathsf{sup}})$
$(\{4\}, \mathcal{R}_{\mathsf{sup}})$
$(\{3\}, \mathcal{R}_{\mathsf{sup}})$
$(\{2\}, \mathcal{R}_{\mathsf{sup}})$
$(\{1\}, \mathcal{R}_{\mathsf{sup}})$
$\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}$
$(\emptyset, \mathcal{R}_{\mathsf{sup}})$

Figure 6.3: The relevant parts of $\mathsf{PT}_{\mathsf{sup}}$ for two terms

normal form of $\{3\}/\mathcal{R}_{\mathsf{sup}}$ or $\{2\}/\mathcal{R}_{\mathsf{sup}}$. Therefore, the parts of $\mathsf{PT}_{\mathsf{sup}}$ highlighted in Figure 6.3 are particularly relevant for t_1 and t_3, respectively. The term $t_2^\sharp$ is a normal form of $\mathsf{DP}(\mathcal{R}_{\mathsf{sup}})/\mathcal{R}_{\mathsf{sup}}$, therefore no part of the complexity proof tree $\mathsf{PT}_{\mathsf{sup}}$ is relevant to show termination for t_2.

The next definition formalises the relevant parts of a proof tree. As indicated above, we restrict the notion to a single path. Recall that according to Definition 6.12, paths in the complexity proof tree PT are ordered from left to right according to the rank functions $\mathsf{rk}_{\mathcal{G}}$ for the considered dependency graphs $\mathcal{G}$.

Definition 6.16. The *current path* $\mathsf{CP}(t)$ of a term t in PT is defined as follows. If $t^\sharp \in \mathsf{NF}(\mathsf{DP}(\mathcal{R})/\mathcal{R})$, then $\mathsf{CP}(t)$ is the empty path, denoted as $()$. Otherwise, for each dependency pair $l \to r$ such that $t^\sharp \notin \mathsf{NF}(\{l \to r\}/\mathcal{R})$, consider the set of nodes whose label contains $l \to r$. By previous observations, each of these sets forms a path starting at the root node of PT. The set of positions forming the leftmost of these paths is $\mathsf{CP}(t)$. We use $\mathsf{CP}_i(t)$ to project on single elements of $\mathsf{CP}(t) = (\alpha_1 = \epsilon, \alpha_2, \dots, \alpha_n)$: if $i > n$, then $\mathsf{CP}_i(t) = \bot$, otherwise $\mathsf{CP}_i(t) = \alpha_i$.

Example 6.17 (continued from Example 6.15)**.** The current paths of t_1, t_2, and t_3 are the following: we have $\mathsf{CP}(t_1) = (\epsilon, 1)$, $\mathsf{CP}(t_2) = ()$, and $\mathsf{CP}(t_3) = (\epsilon, 3)$. For t_1, the projections on the single elements of the path are the following: $\mathsf{CP}_1(t_1) = \epsilon$, $\mathsf{CP}_2(t_1) = 1$, and $\mathsf{CP}_i(t_1) = \bot$ for $i > 2$.

Using the DP processors applied to the nodes which are referred to in $\mathsf{CP}(t)$, we now define a complexity measure $\mathsf{norm}(t)$, which we assign to t. For each DP processor, we use whatever value is naturally decreasing in the termination argument of that processor in order to get the associated measure. Given the reduction pair function g, an upper bound on $\mathsf{norm}(t)$ is easily computable.

Definition 6.18. We define the mapping $\mathsf{norm}_i \colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathbb{N} \cup \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \{\bot\}$ for $i \in \mathbb{N}$ as follows: let $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\alpha = \mathsf{CP}_i(t)$.

1. If $\alpha = \bot$, we set $\mathsf{norm}_i(t) = 0$ if $\mathsf{rt}(t)$ is a defined symbol, and $\mathsf{norm}_i(t) = \bot$ otherwise.
2. If $\alpha \neq \bot$, and $(\mathcal{P}, \mathcal{R})$ is the node at position α and a leaf in PT, then either $\mathcal{P} = \emptyset$, or $\mathcal{P}$ is a trivial SCC of a dependency graph. In both cases, we set $\mathsf{norm}_i(t) = \mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathcal{P}} / \to_{\mathcal{R}})$.

3. If $\alpha \neq \bot$, and $(\mathcal{P}, \mathcal{R})$ is the node at position α and an inner node in PT, then let Φ be the label of each edge starting from $(\mathcal{P}, \mathcal{R})$:
   - If Φ is a reduction pair processor with $\Phi((\mathcal{P}, \mathcal{R})) = \{(\mathcal{Q}, \mathcal{R})\}$, then we set $\mathsf{norm}_i(t) = \mathsf{dh}(t^\sharp, \overset{\epsilon}{\to}_{(\mathcal{P} \setminus \mathcal{Q})} / \to_{(\mathcal{Q} \cup \mathcal{R})})$.
   - If Φ is a dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$, then we set $\mathsf{norm}_i(t) = \mathsf{rk}_{\mathcal{G}}(t)$.
   - If Φ is a subterm criterion processor Φ^{SC}_{π}, then we set $\mathsf{norm}_i(t) = \pi(t^\sharp)$.

We extend the mappings norm_i to the *norm of a term* as follows:

$$\mathsf{norm}(t) = (\mathsf{norm}_1(t), \ldots, \mathsf{norm}_d(t))$$

The central idea behind the complexity measures used in the mapping norm is that rewrite steps induce lexicographical decreases in the norm of the considered term. We now define the according well-founded proper order on the range of the mappings norm_i.

Definition 6.19. We define the following well-founded binary relation $\gg$ on $\mathbb{N} \cup \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \{\bot\}$. We have $a \gg b$ if and only if one of the following properties holds:

1. $a \in \mathbb{N}$, $b \in \mathbb{N}$, and $a > b$, where $>$ is the natural strict order on $\mathbb{N}$, or
2. $a \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $b \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and $a \rhd b$, or
3. $a \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $b = 0$, or
4. $a \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathbb{N}$ and $b = \bot$.

We define $\gg^=$ to be the reflexive closure of $\gg$.

Moreover, we define another well-founded binary relation $\blacktriangleright$ on the same domain as follows: we have $a \blacktriangleright b$ if and only if $a \gg b$ or $a(\to_{\mathcal{R}} \cup \rhd)^+ b$ with $a, b \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

We define the relation $\sqsupset$ on tuples: we set $(a_1, \ldots, a_d) \sqsupset (b_1, \ldots, b_d)$ if and only if there exists some $1 \leqslant i \leqslant d$ such that $a_i \gg b_i$, and $a_j \blacktriangleright b_j$ or $a_j = b_j$ for all $1 \leqslant j < i$.

Finally, let $\sqsupseteq$ be the following relation on tuples: $(a_1, \ldots, a_d) \sqsupseteq (b_1, \ldots, b_d)$ if and only if either $(a_1, \ldots, a_d) \sqsupset (b_1, \ldots, b_d)$, or for all $1 \leqslant i \leqslant d$, we have $a_i \blacktriangleright b_i$ or $a_i = b_i$.

Note that termination of $\mathcal{R}$ implies well-foundedness of $(\to_{\mathcal{R}} \cup \rhd)^+$. Moreover, by assumption, $\mathcal{R}$ is terminating. It is easy to see that extending $(\to_{\mathcal{R}} \cup \rhd)^+$ by $\gg$ preserves well-foundedness, hence $\blacktriangleright$ is well-founded, and it is obviously a proper order. Consequently, also $\sqsupset$ and the relative relation $\sqsupset / \sqsupseteq$ are well-founded proper orders.

Remark 6.20. Actually, $\sqsupset$ is a restriction of $\blacktriangleright^{\text{lex}}$, where only decreases with respect to $\gg$ are considered to be “strict decreases”. This restriction is necessary because Lemma 6.33.1 would not hold if $\sqsupseteq$ was replaced by the reflexive closure of $\blacktriangleright^{\mathrm{lex}}$. This means that [83, Lemma 4.15.1)] is incorrect. However, as shown throughout this section, the restricted "lexicographic order" $\sqsupset$ and Lemma 6.33.1 are still sufficient to prove the main result.

Lemma 6.21. *Let s and t be terms such that $s \xrightarrow{>\epsilon}_{\mathcal{R}} t$. For all $1 \leqslant i < d$, if $\mathsf{CP}_i(s) = \mathsf{CP}_i(t)$, and $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s)(\to_{\mathcal{R}} \cup \rhd)^+\mathsf{norm}_i(t)$ with $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, then either $\mathsf{CP}_{i+1}(t) = \bot$, or $\mathsf{CP}_{i+1}(s) = \mathsf{CP}_{i+1}(t)$.*

Proof. Let $\alpha = \mathsf{CP}_i(s) = \mathsf{CP}_i(t)$.

1. If $\alpha = \bot$, then $\mathsf{CP}_{i+1}(s) = \mathsf{CP}_{i+1}(t) = \bot$, as well, so the lemma holds in this case.

2. If $\alpha \neq \bot$, and $(\mathcal{P}, \mathcal{R})$ is the node at position α and a leaf in PT, then either $\mathcal{P} = \emptyset$, or $\mathcal{P}$ is a trivial SCC of a dependency graph. Again, $\mathsf{CP}_{i+1}(s) = \mathsf{CP}_{i+1}(t) = \bot$.

3. If $\alpha \neq \bot$, and $(\mathcal{P}, \mathcal{R})$ is the node at position α and an inner node in PT, then let Φ be the label of each edge starting from $(\mathcal{P}, \mathcal{R})$:
   - If Φ is a reduction pair or subterm criterion processor, let $\{(\mathcal{Q}, \mathcal{R})\} = \Phi((\mathcal{P}, \mathcal{R}))$, i.e. $(\mathcal{P}, \mathcal{R})$ only has a single child node. As $\alpha = \mathsf{CP}_i(s) = \mathsf{CP}_i(t)$, neither $s^\sharp$ nor $t^\sharp$ is a normal form of $\xrightarrow{\epsilon}_{\mathcal{P}}/\to_{\mathcal{R}}$. Since $s^\sharp \to_{\mathcal{R}} t^\sharp$, $s^\sharp$ can only be a normal form of $\xrightarrow{\epsilon}_{\mathcal{Q}}/\to_{\mathcal{R}}$ if $t^\sharp$ is one, as well. If $t^\sharp$ is indeed a normal form of $\xrightarrow{\epsilon}_{\mathcal{Q}}/\to_{\mathcal{R}}$, then $\mathsf{CP}_{i+1}(t) = \bot$. Otherwise, $\mathsf{CP}_{i+1}(s) = \mathsf{CP}_{i+1}(t) = \alpha 1$.
   - Now assume that Φ is a dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$. Since $\mathsf{norm}_i(s), \mathsf{norm}_i(t) \in \mathbb{N}$ in this case, the assumptions imply that $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$. Hence, $\mathsf{rk}_{\mathcal{G}}(s) = \mathsf{rk}_{\mathcal{G}}(t)$. Therefore, by the definition of CP and the order on the children of α, we know that $\mathsf{CP}_{i+1}(s) = \mathsf{CP}_{i+1}(t)$. Thus the lemma is shown.

$\square$

Lemma 6.22. *For any terms s and t such that $s \xrightarrow{>\epsilon}_{\mathcal{R}} t$, we have $\mathsf{norm}(s) \sqsupseteq \mathsf{norm}(t)$.*

Proof. We can assume that $\mathsf{rt}(t)$ is a defined symbol. Otherwise, $\mathsf{norm}_i(t) = \bot$ for all $1 \leqslant i \leqslant d$, and hence $\mathsf{norm}(t) = (\bot, \ldots, \bot)$, so the lemma would be trivial. As $\mathsf{rt}(s) = \mathsf{rt}(t)$, $\mathsf{rt}(s)$ is also defined. Hence, $s^\sharp \xrightarrow{>\epsilon}_{\mathcal{R}} t^\sharp$.

We now show the following by induction on $d - i$:

Claim 6.23. *If for all $1 \leqslant j < i$, we have either $\mathsf{norm}_j(s) = \mathsf{norm}_j(t)$ or $\mathsf{norm}_j(s)(\to_{\mathcal{R}} \cup \rhd)^+\mathsf{norm}_j(t)$ with $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, then we also have*

$$(\mathsf{norm}_i(s), \ldots, \mathsf{norm}_d(s)) \sqsupseteq (\mathsf{norm}_i(t), \ldots, \mathsf{norm}_d(t)) \ .$$

Clearly, this claim implies the lemma, so the remainder of this proof is devoted to this claim. Applying Lemma 6.21 $i - 1$ times reveals that $\mathsf{CP}_i(t)$ is either $\bot$ or the same as $\mathsf{CP}_i(s)$. We perform case distinction on $\mathsf{CP}_i(t)$.

1. If $\mathsf{CP}_i(t) = \bot$, then $(\mathsf{norm}_i(t), \ldots, \mathsf{norm}_d(t)) = (0, \ldots, 0)$. Since $\mathsf{rt}(s)$ is a defined function symbol, we have $\mathsf{norm}_j(s) \neq \bot$ for all $j \in \mathbb{N}$. Thus the claim follows immediately.

2. If $\mathsf{CP}_i(s) = \mathsf{CP}_i(t) = \alpha$, $\alpha \neq \bot$, and $(\mathcal{P}, \mathcal{R})$ is the node at position α and a leaf in PT, then either $\mathcal{P} = \emptyset$, or $\mathcal{P}$ is a trivial SCC of a dependency graph. Then $\mathsf{CP}_{i+1}(s) = \mathsf{CP}_{i+1}(t) = \bot$, and $(\mathsf{norm}_{i+1}(s), \ldots, \mathsf{norm}_d(s)) = (\mathsf{norm}_{i+1}(t), \ldots, \mathsf{norm}_d(t)) = (0, \ldots, 0)$. Moreover, we have $\mathsf{norm}_i(s) = \mathsf{dh}(s^\sharp, \xrightarrow{\epsilon}_{\mathcal{P}} / \rightarrow_{\mathcal{R}}) \geqslant \mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathcal{P}} / \rightarrow_{\mathcal{R}}) = \mathsf{norm}_i(t)$, so the claim holds.

3. If $\mathsf{CP}_i(s) = \mathsf{CP}_i(t) = \alpha$, $\alpha \neq \bot$, and $(\mathcal{P}, \mathcal{R})$ is the node at position α and an inner node in PT, then let Φ be the label of each edge starting from $(\mathcal{P}, \mathcal{R})$. We proceed to show that $\mathsf{norm}_i(s) \blacktriangleright \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$ by case distinction on Φ.

   - If Φ is a reduction pair processor with $\Phi((\mathcal{P}, \mathcal{R})) = \{(\mathcal{Q}, \mathcal{R})\}$, then because of $s^\sharp \rightarrow_{\mathcal{R}} t^\sharp$, we have the inequality $\mathsf{dh}(s^\sharp, \xrightarrow{>\epsilon}_{\mathcal{P} \setminus \mathcal{Q}} / \rightarrow_{\mathcal{Q} \cup \mathcal{R}}) \geqslant \mathsf{dh}(t^\sharp, \xrightarrow{>\epsilon}_{\mathcal{P} \setminus \mathcal{Q}} / \rightarrow_{\mathcal{Q} \cup \mathcal{R}})$. Thus $\mathsf{norm}_i(s) \blacktriangleright \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$ obtains.

   - If Φ is a dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$, then for each SCC $\mathcal{P}_j$ in $\mathcal{G}$, $s^\sharp$ can only be a normal form of $\mathcal{P}_j / \mathcal{R}$ if $t^\sharp$ is one, as well. Therefore, we have $\mathsf{norm}_i(s) \blacktriangleright \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$ in that case, too.

   - If Φ is a subterm criterion processor Φ^{SC}_{π}, then $\mathsf{norm}_i(s) = \pi(s^\sharp) \rightarrow^{=}_{\mathcal{R}} \pi(t^\sharp) = \mathsf{norm}_i(t)$, and hence $\mathsf{norm}_i(s) \blacktriangleright \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$.

   So, regardless of which DP processor Φ is, we have $\mathsf{norm}_i(s) \blacktriangleright \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$. If $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(t)$, then the claim follows by Definition 2.7. On the other hand, if $\mathsf{norm}_i(s) = \mathsf{norm}_i(t)$ or $\mathsf{norm}_i(s)(\rightarrow_{\mathcal{R}} \cup \rhd)^{+}\mathsf{norm}_i(t)$ with $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, then the claim holds by induction hypothesis.

□

The following lemma extends Lemma 6.22 to root steps $s \xrightarrow{\epsilon}_{\mathcal{R}} t$. However, in this case, we do not consider only the root position of t, but all positions that were "created" by the rewrite step. So essentially, we show that such a step causes a decrease in $\sqsupset$ from s to subterms of t. The restriction on positions p below takes care of the Dershowitz condition (cf. Definition 2.47) in the definition of dependency pairs and the substitution of the applied rewrite rule.

Lemma 6.24. *For any terms s and t such that $s \xrightarrow{\epsilon}_{\mathcal{R}} t$, we have* $\mathsf{norm}(s) \gg^{\mathrm{lex}} \mathsf{norm}(t|_p)$, *and hence* $\mathsf{norm}(s) \sqsupset \mathsf{norm}(t|_p)$ *for all $p \in \mathcal{P}\mathsf{os}(t)$ such that $t|_p \ntrianglelefteq s$.*

Proof. For this proof, we fix p, and let $u = t|_p$. We can assume that $\mathsf{rt}(u)$ is a defined symbol. Otherwise, $\mathsf{norm}(u) = (\bot, \ldots, \bot)$, but $\mathsf{norm}(s) \sqsupseteq (0, \ldots, 0)$ (note that $\mathsf{rt}(s)$ is defined), so the lemma would be immediate. Hence, we have

$s^\sharp \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})} u^\sharp$ using some dependency pair $l \to r$. Let j be the greatest number between 1 and d such that $\mathsf{CP}_j(s) \neq \bot$, the node at position $\mathsf{CP}_j(s)$ is $(\mathcal{Q}, \mathcal{R})$, and $\mathcal{Q}$ contains $l \to r$. Note that such a number exists: since $s^\sharp \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})} u^\sharp$, we have $\mathsf{CP}_1(s) = \epsilon$. By definition, the node at position ϵ in PT is $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$, and $\mathsf{DP}(\mathcal{R})$ contains $l \to r$. Let $\alpha = \mathsf{CP}_j(s)$. We distinguish whether $\mathsf{CP}_j(u) = \alpha$. This determines whether the strict part of the lexicographic decrease must happen at index j or at an earlier index.

First, suppose $\mathsf{CP}_j(u) = \alpha$, so we have $\mathsf{CP}_j(s) = \mathsf{CP}_j(u)$. Since CP is a tree, this implies $\mathsf{CP}_i(s) = \mathsf{CP}_i(u)$ for all $1 \leqslant i \leqslant j$. We show that for all $1 \leqslant i \leqslant j$, $\mathsf{norm}_i(s) \gg^= \mathsf{norm}_i(u)$ holds, and $\mathsf{norm}_j(s) \gg \mathsf{norm}_j(u)$. From these two properties, the lemma follows. In order to show them, we fix some $1 \leqslant i \leqslant j$. Let $\beta = \mathsf{CP}_i(s) = \mathsf{CP}_i(u)$.

1. If the node at position β is a leaf of PT, then $i = j$, and $\mathcal{Q}$ is a trivial SCC of a dependency graph. By assumption, $l \to r \in \mathcal{Q}$. Therefore, $\mathsf{dh}(s^\sharp, \xrightarrow{\epsilon}_{\mathcal{Q}}/\to_{\mathcal{R}}) > \mathsf{dh}(u^\sharp, \xrightarrow{\epsilon}_{\mathcal{Q}}/\to_{\mathcal{R}})$, and thus $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$.

2. If the node $(\mathcal{P}, \mathcal{R})$ at position β is an inner node of PT, let Φ be the label of each edge starting from $(\mathcal{P}, \mathcal{R})$. Obviously, $\mathcal{Q} \subseteq \mathcal{P}$, and therefore $l \to r \in \mathcal{P}$. Regardless of which processor Φ is, the semantics of Φ imply that $\mathsf{norm}_i(s) \gg^= \mathsf{norm}_i(u)$. Moreover, if $i = j$, then $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$ follows. In more detail:

   - If Φ is a reduction pair processor, then let $\{(\mathcal{P}', \mathcal{R})\} = \Phi((\mathcal{P}, \mathcal{R}))$. Recall that $l \to r \in \mathcal{P}$, so the inequality $\mathsf{dh}(s^\sharp, \xrightarrow{\epsilon}_{\mathcal{P}\setminus\mathcal{P}'}/\to_{\mathcal{P}'\cup\mathcal{R}}) \geqslant \mathsf{dh}(u^\sharp, \xrightarrow{\epsilon}_{\mathcal{P}\setminus\mathcal{P}'}/\to_{\mathcal{P}'\cup\mathcal{R}})$ follows, and thus $\mathsf{norm}_i(s) \gg^= \mathsf{norm}_i(u)$. If $i = j$, then by definition of j, $l \to r$ is contained in $\mathcal{P} \setminus \mathcal{P}'$. It follows that $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$ in that case.

   - If Φ is a dependency graph processor $\Phi^{\mathsf{DG}}_{\mathcal{G}}$, then by definition of $\mathsf{rk}_{\mathcal{G}}$, we have $\mathsf{rk}_{\mathcal{G}}(s) \geqslant \mathsf{rk}_{\mathcal{G}}(l \to r) \geqslant \mathsf{rk}_{\mathcal{G}}(u)$. Hence, $\mathsf{norm}_i(s) \gg^= \mathsf{norm}_i(u)$. If $i = j$, then the equality $\mathsf{rk}_{\mathcal{G}}(s) = \mathsf{rk}_{\mathcal{G}}(l \to r)$ is impossible: otherwise, the node at position $\mathsf{CP}_{i+1}(s)$ would have to be of the shape $(\mathcal{Q}', \mathcal{R})$ with $l \to r \in \mathcal{Q}'$, which contradicts the definition of j. Thus, $\mathsf{rk}_{\mathcal{G}}(s) > \mathsf{rk}_{\mathcal{G}}(l \to r) \geqslant \mathsf{rk}_{\mathcal{G}}(u)$ and hence $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$ in that case.

   - If Φ is a subterm criterion processor Φ^{SC}_{π} with $\Phi((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}', \mathcal{R})\}$, then $\pi(s^\sharp) \trianglerighteq \pi(u^\sharp)$, and hence $\mathsf{norm}_i(s) \gg^= \mathsf{norm}_i(u)$. If $i = j$, then by definition of j, $l \to r \in \mathcal{P} \setminus \mathcal{P}'$, and hence $\pi(s^\sharp) \vartriangleright \pi(u^\sharp)$ and $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$ in that case.

In all cases, it follows that for all $1 \leqslant i \leqslant j$, $\mathsf{norm}_i(s) \gg^= \mathsf{norm}_i(u)$ holds, and $\mathsf{norm}_j(s) \gg \mathsf{norm}_j(u)$. This concludes the case where $\mathsf{CP}_j(u) = \alpha$.

Now suppose $\mathsf{CP}_j(u) \neq \alpha$. Then let i be the greatest number between 1 and j such that $\mathsf{CP}_i(s) = \mathsf{CP}_i(u) = \beta$. As β is a proper prefix of α, the node $(\mathcal{P}, \mathcal{R})$ at position β is an inner node of PT. Let Φ be the label of each edge starting from $(\mathcal{P}, \mathcal{R})$. Using the arguments from above, we see that $\mathsf{norm}_{i'}(s) \gg^= \mathsf{norm}_{i'}(u)$ for all $1 \leqslant i' \leqslant i$. We now show that $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$ or $\mathsf{norm}_{i+1}(s) \gg \mathsf{norm}_{i+1}(u)$ holds.

1. If Φ is a reduction pair or subterm criterion processor, then by our assumptions, $\mathsf{CP}_{i+1}(s) = \beta 1$. Since β has only one child in this case, and by assumption $\mathsf{CP}_{i+1}(u) \neq \mathsf{CP}_{i+1}(s)$, this implies $\mathsf{CP}_{i+1}(u) = \bot$. Thus, $\mathsf{norm}_{i+1}(s) > 0 = \mathsf{norm}_{i+1}(u)$.
2. If Φ is a dependency graph processor, then $\mathsf{norm}_i(s) \neq \mathsf{norm}_i(u)$, since $\mathsf{CP}_{i+1}(s) \neq \mathsf{CP}_{i+1}(u)$ by assumption. Thus $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(u)$.

In both cases, it follows that $\mathsf{norm}(s) \sqsupset \mathsf{norm}(u)$, which is what we wanted to show. □

Up to now, we have shown that norm decreases lexicographically under rewriting. For rewrite steps whose redex position is at the root, this decrease is even strict. In order to turn this into an upper bound on $\mathsf{dc}_{\mathcal{R}}$, we still have to do some work: we also have to consider the norms of all proper subterms of a considered term, and the range of norm is not suitable for direct complexity analysis yet: the order $\sqsupset$ on the range of norm is almost a lexicographic product, and the order $\blacktriangleright$ on the single coordinates still contains the rewrite relation $\to_{\mathcal{R}}$ itself. We now solve these problems by lifting the range of norm to the term level and simulating derivations of $\mathcal{R}$ at that level.

Notation 6.25. For the rest of this section let A be the maximum arity of any function symbol occurring in $\mathcal{R}$, and $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

Depending on PT, d, A, C, and g, we now define a TRS $\mathcal{R}_{\mathsf{sim}}$ which simulates $\mathcal{R}$ and is compatible with a LPO. The simulating TRS $\mathcal{R}_{\mathsf{sim}}$ is based on a mapping tr (see Definition 6.30 below) such that $s \to_{\mathcal{R}} t$ implies $\mathsf{tr}(s) \to^{+}_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(t)$. Given a term t, tr employs a $d + A$-ary function symbol f. The first d arguments of f are used to represent $\mathsf{norm}(t)$; the last A arguments of f are used to represent $\mathsf{tr}(t')$ for each direct subterm t' of t.

In the simulation, we often have to recalculate the first d arguments of each f. Due to the definition of norm, we know that for each term t and $1 \leqslant i \leqslant d$, either $\mathsf{norm}_i(t) \in \mathbb{N}$ and $\mathsf{norm}_i(t) \leqslant g(|t|)$, or $\mathsf{norm}_i(t) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathsf{norm}_i(t) \lhd t$, or $\mathsf{norm}_i(t) = \bot$. We use a unary function symbol choice such that $\mathsf{choice}(\mathsf{tr}(t))$ rewrites to the representations of $g(|t|)$, $\mathsf{tr}(t')$ for each immediate subterm t' of t, and $\bot$. In particular, we often use terms of the shape $\mathsf{choice}(\mathsf{f}(0, \ldots, 0, x_1, \ldots, x_A))$ in the definition of $\mathcal{R}_{\mathsf{sim}}$, so we define an abbreviation for terms of this shape below.

The main tools for achieving the simulation of a root rewrite step $s \xrightarrow{\epsilon}_{\mathcal{R}} t$ are rules which build the new f symbols for the positions in t "created" by the step. These are at most $(C + 1) \cdot A^C$ many new positions, and each proper subterm of s may be duplicated at most that many times. As a very simple example, if $d = 3$, $A = 1$, and $C = 1$, this behaviour is simulated by rules of the following shape, where N is an abbreviation defined in Notation 6.26 below:

$$\begin{aligned}
\mathsf{f}(u_1, \mathsf{s}(u_2), u_3, x) &\to \mathsf{f}(u_1, u_2, N(x), \mathsf{f}(u_1, u_2, N(x), x)) \\
\mathsf{f}(u_1, \mathsf{f}(v_1, v_2, v_3, y), u_3, x) &\to \mathsf{f}(u_1, y, N(x), \mathsf{f}(u_1, y, N(x), x)) \\
\mathsf{f}(u_1, \mathsf{f}(v_1, v_2, v_3, y), u_3, x) &\to \mathsf{f}(u_1, \mathsf{0}, N(x), \mathsf{f}(u_1, \mathsf{0}, N(x), x)) \\
\mathsf{f}(u_1, \mathsf{0}, u_3, x) &\to \mathsf{f}(u_1, \bot, N(x), \mathsf{f}(u_1, \bot, N(x), x))
\end{aligned}$$

We use similar rules for decreases of u_1 or u_3 with respect to the ordering $\gg$. We introduce another abbreviation in order to represent the right-hand sides of these rules concisely.

Notation 6.26. We use the abbreviation $N(x_1,\dots,x_A)$ to represent the term $\mathsf{choice}(\mathsf{f}(\mathsf{0},\dots,\mathsf{0},x_1,\dots,x_A))$. Further, we make use of the following abbreviation M_i^k (for $i \in \{1,\dots,d\}$ and $k \in \mathbb{N}$):

$$\begin{aligned}
&M_i^0(u_1,\dots,u_i,x_1,\dots,x_A) = \mathsf{f}(u_1,\dots,u_i,\overline{N(x_1,\dots,x_A)},x_1,\dots,x_A)\\
&M_i^{k+1}(u_1,\dots,u_i,x_1,\dots,x_A)\\
&\qquad = \mathsf{f}(u_1,\dots,u_i,\overline{N(\overline{M_i^k(u_1,\dots,u_i,x_1,\dots,x_A)})},\overline{M_i^k(u_1,\dots,u_i,x_1,\dots,x_A)})
\end{aligned}$$

Here u_j ($j \in \{1,\dots,i\}$) and $x_{j'}$ ($j' \in \{1,\dots,A\}$) denote variables and we write $\overline{t}$ to denote $t,\dots,t$, where the number of repetitions of t follows from the context.

Consider the reduction pair function g of $\mathcal{R}$. Since g is assumed to be a multiply recursive function, it is an easy exercise to define a TRS $\mathcal{R}'_{\mathsf{sim}}$ (employing the constructors s, $\mathsf{0}$) that computes the function g: one can simply define g using only initial functions, composition, primitive recursion, and k-ary Ackermann functions, and directly turn the resulting definition of g into rewrite rules. So we obtain a TRS $\mathcal{R}'_{\mathsf{sim}}$ and a defined function symbol g such that $\mathsf{g}(\mathsf{s}^n(\mathsf{0})) \to^*_{\mathcal{R}'_{\mathsf{sim}}} \mathsf{s}^{g(n)}(\mathsf{0})$ for all $n \in \mathbb{N}$. Moreover, if defined this way, $\mathcal{R}'_{\mathsf{sim}}$ is compatible with a LPO $>_{\mathsf{LPO}}$ such that the precedence $>$ of the LPO includes $\mathsf{g} > \mathsf{s} > \mathsf{0}$.

Definition 6.27. Consider the following (schematic) TRS $\mathcal{R}_{\mathsf{sim}}$, where $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant A$. In order to save (horizontal) space, we use $\vec{x}$ as a shorthand for $x_1,\dots,x_A$ here.

$$\begin{array}{rrcl}
1_i\colon & \mathsf{f}(u_1,\dots,u_{i-1},\mathsf{s}(u_i),u_{i+1},\dots,u_d,\vec{x}) & \to & M_i^C(u_1,\dots,u_i,\vec{x})\\
2_{i,j}\colon & \mathsf{f}(u_1,\dots,u_{i-1},\mathsf{f}(v_1,\dots,v_d,\vec{y}),u_{i+1},\dots,u_d,\vec{x}) & \to & M_i^C(u_1,\dots,u_{i-1},y_j,\vec{x})\\
3_i\colon & \mathsf{f}(u_1,\dots,u_{i-1},\mathsf{f}(v_1,\dots,v_d,\vec{y}),u_{i+1},\dots,u_d,\vec{x}) & \to & M_i^C(u_1,\dots,u_{i-1},\mathsf{0},\vec{x})\\
4_i\colon & \mathsf{f}(u_1,\dots,u_{i-1},\mathsf{0},u_{i+1},\dots,u_d,\vec{x}) & \to & M_i^C(u_1,\dots,u_{i-1},\bot,\vec{x})\\
5_j\colon & \mathsf{size}(\mathsf{f}(u_1,\dots,u_d,\vec{x})) & \to & \times_A(\mathsf{size}(x_j))\\
6\colon & \mathsf{size}(x) & \to & \mathsf{s}(\mathsf{0})\\
7\colon & \times_A(\mathsf{s}(x)) & \to & \mathsf{s}^A(\times_A(x))\\
8\colon & \times_A(\mathsf{0}) & \to & \mathsf{0}\\
9\colon & \mathsf{f}(u_1,\dots,u_d,\vec{x}) & \to & \mathsf{c}\\
10_j\colon & \mathsf{f}(u_1,\dots,u_d,\vec{x}) & \to & x_j\\
11\colon & \mathsf{h}(x) & \to & \mathsf{f}(\overline{N(\overline{x})},\overline{x})\\
12\colon & \mathsf{z} & \to & \mathsf{f}(\overline{N(\overline{\mathsf{c}})},\overline{\mathsf{c}})\\
13_j\colon & \mathsf{choice}(\mathsf{f}(u_1,\dots,u_d,\vec{x})) & \to & x_j\\
14\colon & \mathsf{choice}(x) & \to & \mathsf{g}(\mathsf{size}(x))\\
15\colon & \mathsf{choice}(x) & \to & \bot
\end{array}$$

These rules are augmented by $\mathcal{R}'_{\mathsf{sim}}$ defining the function symbol g. The signatures of $\mathcal{R}'_{\mathsf{sim}}$ and $\mathcal{R}_{\mathsf{sim}} \setminus \mathcal{R}'_{\mathsf{sim}}$ are disjoint with the exception of g and the constructors s and $\mathsf{0}$.

Note that $\mathcal{R}_{\mathsf{sim}}$ depends only on the proof tree, the constants d, A, C, and the reduction pair function g. The rules 1_i–4_i are the main rules for the simulation of the effects of a single rewrite step $s \xrightarrow{\epsilon}_{\mathcal{R}} t$ in $\mathcal{R}_{\mathsf{sim}}$. These rules simulate the case that $\mathsf{norm}_i(s) \gg \mathsf{norm}_i(t|_p)$ for all $p \in \mathcal{P}\mathsf{os}(t)$ such that $t|_p \ntrianglelefteq s$, and $\mathsf{norm}_{i'}(s) \gg^{=} \mathsf{norm}_{i'}(t|_p)$ for all $1 \leqslant i' \leqslant i$. They are also responsible for creating the at most $(C+1) \cdot A^C$ many new positions and copies of each subterm of s in t. The rules 5_j–8 define a function symbol size, that is, $\mathsf{size}(s)$ reduces to a numeral $\mathsf{s}^n(\mathsf{0})$ such that $n \geqslant |s|$. The rules 9–10_j make sure that any superfluous positions and copies of subterms created by the rules 1_i–4_i can be deleted. The rules 11 and 12 guarantee that the simulating derivation can be started by a term whose size is not greater than the size of the starting term in the original derivation (see Lemma 6.36 below). The rules 13_j–15 define the function symbol choice introduced in the abbreviations M_i^C, and N. Loosely speaking, $\mathsf{choice}(t)$ is an upper bound of $\mathsf{norm}_i(t)$ with respect to $\gg$.

Lemma 6.28. *The derivational complexity of $\mathcal{R}_{\mathsf{sim}}$ is bounded by a multiply recursive function.*

Proof. By our construction, the TRS $\mathcal{R}'_{\mathsf{sim}}$ computing g can be shown terminating using an LPO $>_{\mathsf{LPO}}$ such that the precedence $>$ of the LPO contains $\mathsf{g} > \mathsf{s} > \mathsf{0}$. It is easy to check that extending this precedence by

$$\mathsf{h}, \mathsf{z} > \mathsf{f} > \mathsf{choice} > \mathsf{g}, \mathsf{size} > \times_A > \mathsf{s}, \mathsf{0}, \mathsf{c}, \bot$$

makes the whole TRS $\mathcal{R}_{\mathsf{sim}}$ compatible with this LPO. Therefore, by Theorem 4.42, $\mathsf{dc}_{\mathcal{R}_{\mathsf{sim}}}$ is bounded by a multiply recursive function. □

Notation 6.29. For the remainder of this section, let $\mathcal{F}_{\mathsf{sim}}$ denote the signature of $\mathcal{R}_{\mathsf{sim}}$.

In the following sequence of lemmata, we show that the TRS $\mathcal{R}_{\mathsf{sim}}$ indeed simulates $\mathcal{R}$ as requested.

Definition 6.30. The mapping $\mathsf{tr}\colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}_{\mathsf{sim}}, \mathcal{V})$ is defined by the following equation:

$$\mathsf{tr}(t) = \begin{cases} t & \text{if } x \in \mathcal{V} \\ \mathsf{f}(\mathsf{norm}_1(t)^*, \ldots, \mathsf{norm}_d(t)^*, \mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Here the operator $(\cdot)^*$ is defined as follows:

$$u^* = \begin{cases} \bot & \text{if } u = \bot \\ \mathsf{s}^u(\mathsf{0}) & \text{if } u \in \mathbb{N} \\ \mathsf{tr}(u) & \text{if } u \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \end{cases}$$

Definition 6.31. We define an equivalence $\approx$ on $\mathcal{T}(\mathcal{F}_{\mathsf{sim}}, \mathcal{V})$. We have $s \approx t$ if and only if one of the following properties holds:

1. $s = t$, or
2. $s = \mathsf{f}(u_1, \ldots, u_d, s_1, \ldots, s_A)$, $t = \mathsf{f}(v_1, \ldots, v_d, t_1, \ldots, t_A)$, and $s_i \approx t_i$ for all $1 \leqslant i \leqslant A$.

Lemma 6.32. *For all terms s and t with $\mathsf{tr}(s) \approx t$, we have $\mathsf{size}(t) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{s}^m(\mathsf{0})$ for some $m \geqslant |s|$.*

Proof. We show the lemma by induction on $|s|$. If $s \in \mathcal{V}$, then $|s| = 1$ and $\mathsf{size}(s)$ rewrites to $\mathsf{s}(\mathsf{0})$ by rule 6. Hence, the lemma follows in this case.

Now assume that $s = f(s_1, \ldots, s_n)$. Then $t = \mathsf{f}(u_1, \ldots, u_d, t_1, \ldots, t_n, \bar{\mathsf{c}})$, where $\mathsf{tr}(s_j) \approx t_j$ for all $1 \leqslant j \leqslant n$. If $|s| = 1$, then the lemma follows since $\mathsf{size}(t) \to_{\mathcal{R}_{\mathsf{sim}}} \mathsf{s}(\mathsf{0})$ by applying rule 6. Otherwise, suppose $|s| > 1$. Then fix $1 \leqslant j \leqslant n$ such that $|s_j|$ is maximal. By induction hypothesis, we have $\mathsf{size}(t_j) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{s}^{m_j}(\mathsf{0})$ for some $m_j \geqslant |s_j|$. Hence, by applying rules 5_j, 7, and 8, we obtain $\mathsf{size}(t) \to_{\mathcal{R}_{\mathsf{sim}}} \times_A(\mathsf{size}(t_j)) \to^*_{\mathcal{R}_{\mathsf{sim}}} \times_A(\mathsf{s}^{m_j}(\mathsf{0})) \to^*_{\mathcal{R}_{\mathsf{sim}}} \mathsf{s}^{A \cdot m_j}(\mathsf{0})$. Due to $A \cdot m_j \geqslant |s|$, the lemma follows. □

Lemma 6.33. *The following properties of $\mathcal{R}_{\mathsf{sim}}$ hold:*

1. *If $s = \mathsf{f}(u_1^*, \ldots, u_d^*, \vec{s})$, $t = \mathsf{tr}(t') = \mathsf{f}(v_1^*, \ldots, v_d^*, \vec{s})$, and $(u_1, \ldots, u_d) \sqsupseteq (v_1, \ldots, v_d)$, then $s \to^*_{\mathcal{R}_{\mathsf{sim}}} t$.*
2. *For any terms $s = \mathsf{tr}(s')$ and $t = \mathsf{tr}(t')$, $s' \xrightarrow{\epsilon}_{\mathcal{R}} t'$ implies $s \to^+_{\mathcal{R}_{\mathsf{sim}}} t$.*
3. *If $a \to_{\mathcal{R}} b$ and $\mathsf{tr}(a) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(b)$, then for any n-ary function symbol $f \in \mathcal{F}$, we have $s = \mathsf{tr}(f(t_1, \ldots, a, \ldots, t_n)) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(f(t_1, \ldots, b, \ldots, t_n))$.*

Proof. We show these items by mutual induction on $\mathsf{dh}(s, \to_{\mathcal{R}_{\mathsf{sim}}} \cup \rhd)$. Note that by Lemma 6.28, $\mathcal{R}_{\mathsf{sim}}$ terminates, and hence $\to_{\mathcal{R}_{\mathsf{sim}}} \cup \rhd$ is well-founded.

1. In order to show Property 1, observe that it suffices to show the following items for all $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant A$ (compare Definition 6.19):
   - $\mathsf{f}(w_1, \ldots, \mathsf{s}(w_i), \ldots, w_d, \vec{x}) \to^+_{\mathcal{R}_{\mathsf{sim}}} M_i^0(w_1, \ldots, w_i, \vec{x})$
   - $\mathsf{f}(w_1, \ldots, w_{i-1}, \mathsf{f}(w_1', \ldots, w_d', \vec{y}), \ldots, w_d, \vec{x})$ $\to^+_{\mathcal{R}_{\mathsf{sim}}} M_i^0(w_1, \ldots, w_{i-1}, y_j, \vec{x})$
   - $\mathsf{f}(w_1, \ldots, w_{i-1}, \mathsf{tr}(a), \ldots, w_d, \vec{x})$ $\to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(w_1, \ldots, w_{i-1}, \mathsf{tr}(b), \ldots, w_d, \vec{x})$ whenever $a \to_{\mathcal{R}} b$
   - $\mathsf{f}(w_1, \ldots, w_{i-1}, \mathsf{0}, \ldots, w_d, \vec{x}) \to^+_{\mathcal{R}_{\mathsf{sim}}} M_i^0(w_1, \ldots, w_{i-1}, \bot, \vec{x})$
   - $\mathsf{f}(w_1, \ldots, w_{i-1}, \mathsf{f}(w_1', \ldots, w_d', \vec{y}), \ldots, w_d, \vec{x})$ $\to^+_{\mathcal{R}_{\mathsf{sim}}} M_i^0(w_1, \ldots, w_{i-1}, \mathsf{0}, \vec{x})$

   The first, second, fourth and fifth items follow directly by applying rules 10_1 and 1_i–4_i of $\mathcal{R}_{\mathsf{sim}}$. The third item follows by applying items 2 and 3 of the induction hypothesis.

2. We now show Property 2. Let $l \to r$ be the rewrite rule, and σ the substitution applied in the step $s' \xrightarrow{\epsilon}_{\mathcal{R}} t'$. Let $(v_1, \ldots, v_n) = \mathsf{norm}(s')$. Since l is not a variable, we have $l = f(l_1, \ldots, l_n)$. By Lemma 6.24, we have $\mathsf{norm}(s') \sqsupset \mathsf{norm}(t'|_p)$ for all $p \in \mathcal{P}\mathsf{os}(t')$ such that $t'|_p \not\trianglelefteq s'$. Let i be the greatest number such that $1 \leqslant i \leqslant d$, and for some $p \in \mathcal{P}\mathsf{os}(t')$ with $t'|_p \not\trianglelefteq s'$, we have $v_i \gg \mathsf{norm}_i(t'|_p)$, and $v_j = \mathsf{norm}_j(t'|_p)$ for all $1 \leqslant j \leqslant i$. If v_i and $\mathsf{norm}_i(t'|_p)$ are both contained in $\mathbb{N}$, then s has the shape $\mathsf{f}(v_1^*, \ldots, \mathsf{s}^{v_i}(\mathsf{0}), \ldots, v_d^*, \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{c})$, and we can apply rules 1_i and 10_1 to obtain $M_i^{\mathsf{dp}(r)}(v_1^*, \ldots, v_{i-1}^*, \mathsf{s}^{v_i-1}(\mathsf{0}), \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{c})$. (We show the rest of the proof only for this case. On the other hand, if v_i and $\mathsf{norm}_i(t'|_p)$ are both contained in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ or $\mathsf{norm}_i(t'|_p) \in \{0, \bot\}$, we proceed by analogy using rule $2_{i,j}$, 3_i, or 4_i instead of 1_i). Note that because of $v_i > \mathsf{norm}_i(t'|_p)$ and $\mathsf{norm}_i(t'|_p) \in \mathbb{N}$ we certainly have $v_i > 0$. We now show the following claim by side induction on $\mathsf{dp}(u)$.

Claim 6.34. *For every term u such that $u \trianglelefteq r$, we have the derivation $M_i^{\mathsf{dp}(u)}(v_1^*, \ldots, v_{i-1}^*, \mathsf{s}^{v_i-1}(\mathsf{0}), \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}}^* \mathsf{tr}(u\sigma)$.*

Note that since $r \trianglelefteq r$, showing this claim suffices to conclude Property 2 of the lemma. Hence, the remainder of this proof is devoted to the proof of the claim. We perform case distinction on u.

Suppose $u \triangleleft l$. Then there exists some $j \in \{1, \ldots, n\}$ with $u \trianglelefteq l_j$. Using rule 10_1, we get

$$M_i^{\mathsf{dp}(u)}(v_1^*, \ldots, v_{i-1}^*, \mathsf{s}^{v_i-1}(\mathsf{0}), \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}}^* \mathsf{tr}(l_j\sigma) .$$

Further applications of rule $10_{j'}$ for some values of j' yield $\mathsf{tr}(l_j\sigma) \to_{\mathcal{R}_{\mathsf{sim}}}^* \mathsf{tr}(u\sigma)$, which concludes the first case.

Now suppose that $u \not\trianglelefteq l$. Since variables occurring in u also occur in r and hence in l, the condition $u \not\trianglelefteq l$ implies that u is not a variable. Hence, $u = h(u_1, \ldots, u_m)$. Let $\mathsf{norm}(u\sigma) = (w_1, \ldots, w_d)$. By side induction hypothesis and employing rule 10_1 $\mathsf{dp}(u) - 1 - \mathsf{dp}(u_j)$ many times, we obtain for all $1 \leqslant j \leqslant m$

$$M_i^{\mathsf{dp}(u)-1}(v_1^*, \ldots, v_{i-1}^*, \mathsf{s}^{v_i-1}(\mathsf{0}), \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}}^* \mathsf{tr}(u_j\sigma) .$$

By combining this with $A - m$ many applications of rule 9, we obtain

$$\begin{aligned}&M_i^{\mathsf{dp}(u)}(v_1^*, \ldots, \mathsf{s}^{v_i-1}(\mathsf{0}), \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}}^* \\ &\quad \mathsf{f}(v_1^*, \ldots, \mathsf{s}^{v_i-1}(\mathsf{0}), \overline{N(\mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{\mathsf{c}})}, \mathsf{tr}(u_1\sigma), \ldots, \mathsf{tr}(u_m\sigma), \bar{\mathsf{c}}) .\end{aligned}$$

Recall that by Lemma 6.24, we have $(v_1, \ldots, v_d) \sqsupset (w_1, \ldots, w_d)$. Moreover, by the definitions of norm and g, for all $i < j \leqslant d$, either $w_j \in \mathbb{N}$ and $w_j \leqslant g(|u\sigma|)$, or $w_j \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $w_j \trianglelefteq u\sigma$, or $w_j = \bot$. In either case, it is easy to check that $N(\mathsf{tr}(u_1\sigma), \ldots, \mathsf{tr}(u_m\sigma), \ldots, \bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}} w_j'^*$ such that $w_j' \gg^= w_j$. Therefore, we obtain

$$\begin{aligned}&M_i^{\mathsf{dp}(u)}(v_1^*, \ldots, v_{i-1}^*, \mathsf{s}^{v_i-1}(\mathsf{0}), \mathsf{tr}(l_1\sigma), \ldots, \mathsf{tr}(l_n\sigma), \bar{\mathsf{c}}) \\ &\quad \to_{\mathcal{R}_{\mathsf{sim}}}^* \mathsf{f}(v_1^*, \ldots, v_{i-1}^*, \mathsf{s}^{v_i-1}(\mathsf{0}), w_{i+1}'^*, \ldots, w_d'^*, \mathsf{tr}(u_1\sigma), \ldots, \mathsf{tr}(u_m\sigma), \bar{\mathsf{c}})\end{aligned}$$

such that $w'_j \gg^= w_j$ for all $i < j \leqslant d$. Observe that $(v_1, \dots, v_{i-1}, v_i - 1, w'_{i+1}, \dots, w'_d) \sqsupseteq (w_1, \dots, w_d)$. Therefore, item 1 of the induction hypothesis yields

$$\mathsf{f}(v_1^*, \dots, \mathsf{s}^{v_i - 1}(\mathsf{0}), w'^*_{i+1}, \dots, w'^*_d, \mathsf{tr}(u_1\sigma), \dots, \mathsf{tr}(u_m\sigma), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(u\sigma) \ ,$$

and the claim and thus Property 2 follow.

3. We now prove Property 3. Let $\mathsf{norm}(f(t_1, \dots, a, \dots, t_n)) = (v_1, \dots, v_d)$. Then the term $\mathsf{tr}(f(t_1, \dots, a, \dots, t_n))$ has the shape

$$\mathsf{f}(v_1^*, \dots, v_d^*, \mathsf{tr}(t_1), \dots, \mathsf{tr}(a), \dots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \ .$$

By assumption,

$$\begin{aligned} &\mathsf{f}(v_1^*, \dots, v_d^*, \mathsf{tr}(t_1), \dots, \mathsf{tr}(a), \dots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \\ &\quad \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(v_1^*, \dots, v_d^*, \mathsf{tr}(t_1), \dots, \mathsf{tr}(b), \dots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \ . \end{aligned}$$

Let $\mathsf{norm}(f(t_1, \dots, b, \dots, t_n)) = (w_1, \dots, w_d)$. We have $(v_1, \dots, v_d) \sqsupseteq (w_1, \dots, w_d)$ by Lemma 6.22. By item 1 of the induction hypothesis,

$$\begin{aligned} &\mathsf{f}(v_1^*, \dots, v_d^*, \mathsf{tr}(t_1), \dots, \mathsf{tr}(b), \dots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \\ &\quad \to^*_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(w_1^*, \dots, w_d^*, \mathsf{tr}(t_1), \dots, \mathsf{tr}(b), \dots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \\ &\quad = \mathsf{tr}(f(t_1, \dots, b, \dots, t_n)) \ , \end{aligned}$$

concluding Property 3 and the lemma.

□

Lemma 6.35. *For any terms s and t, $s \to_{\mathcal{R}} t$ implies $\mathsf{tr}(s) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(t)$.*

Proof. Easy consequence of Lemma 6.33.2 and 6.33.3. □

Lemma 6.35 yields that the length of any derivation in $\mathcal{R}$ can be estimated by the length of the corresponding derivation in $\mathcal{R}_{\mathsf{sim}}$, where the correspondence is established by the mapping tr. However, since a term t may be considerably smaller than the corresponding term $\mathsf{tr}(t)$, we cannot bound $\mathsf{dc}_{\mathcal{R}}$ in $\mathsf{dc}_{\mathcal{R}_{\mathsf{sim}}}$ yet. In order to fill this gap, we make use of the following lemma.

Lemma 6.36. *For any ground term t, we have $\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(t)$.*

Proof. We proceed by induction on $\mathsf{dp}(t)$. If $\mathsf{dp}(t) = 0$, then t is a constant. Rule 12 yields the rewrite step $\mathsf{z} \to_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(\overline{N(\bar{\mathsf{c}})}, \bar{\mathsf{c}})$. Let $\mathsf{norm}(t) = (v_1, \dots, v_d)$. By the definition of norm and g, for all $1 \leqslant i \leqslant d$, we have either $v_i \in \mathbb{N}$ and $v_i \leqslant g(1)$, or $v_i \in \mathcal{T}(\mathcal{F})$ and $v_i \trianglelefteq t$, or $v_i = \perp$. In all three cases, it is easy to check that $N(\bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}} v'^*_i$ for some v'_i with $v'_i \gg^= v_i$. Hence, we obtain $\mathsf{z} \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(v'^*_1, \dots, v'^*_d, \bar{\mathsf{c}})$ such that $(v'_1, \dots, v'_d) \sqsupseteq (v_1, \dots, v_d)$. By Lemma 6.33.1, $\mathsf{f}(v'^*_1, \dots, v'^*_d, \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(v_1^*, \dots, v_d^*, \bar{\mathsf{c}}) = \mathsf{tr}(t)$, hence the lemma follows.

Assume $\mathsf{dp}(t) > 0$, so t has the shape $f(t_1, \dots, t_n)$. Then rule 11 yields the rewrite step

$$\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}) \to_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(\overline{N(\overline{\mathsf{h}^{\mathsf{dp}(t)-1}(\mathsf{z})})}, \overline{\mathsf{h}^{\mathsf{dp}(t)-1}(\mathsf{z})}) \ .$$

Using rules 11 and 10_1, we obtain $\mathsf{h}^{\mathsf{dp}(t)-1}(\mathsf{z}) \to^*_{\mathcal{R}_{\mathsf{sim}}} \mathsf{h}^{\mathsf{dp}(t_j)}(\mathsf{z})$ for all $1 \leqslant j \leqslant n$, and by induction hypothesis, $\mathsf{h}^{\mathsf{dp}(t_j)}(\mathsf{z}) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{tr}(t_j)$. Therefore,

$$\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(\overline{N(\mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}})}, \mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \; .$$

Let $\mathsf{norm}(t) = (v_1, \ldots, v_d)$. By the definition of norm and g, for all $1 \leqslant i \leqslant d$, we have either $v_i \in \mathbb{N}$ and $v_i \leqslant g(|t|)$, or $v_i \in \mathcal{T}(\mathcal{F})$ and $v_i \trianglelefteq t$, or $v_i = \bot$. In all three cases, it is easy to check that $N(\mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \to_{\mathcal{R}_{\mathsf{sim}}} v_i'^*$ for some v_i' with $v_i' \gg^= v_i$. Hence, we obtain

$$\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}) \to^+_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(v_1'^*, \ldots, v_d'^*, \mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}})$$

such that $(v_1', \ldots, v_d') \sqsupseteq (v_1, \ldots, v_d)$. By Lemma 6.33.1,

$$\mathsf{f}(v_1'^*, \ldots, v_d'^*, \mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{sim}}} \mathsf{f}(v_1^*, \ldots, v_d^*, \mathsf{tr}(t_1), \ldots, \mathsf{tr}(t_n), \bar{\mathsf{c}}) = \mathsf{tr}(t) \; ,$$

thus the lemma follows. □

Now we can prove the main result of this section, Theorem 6.9.

Theorem. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.3 using a proof tree* PT*, and suppose that there exists a multiply recursive reduction pair function of* PT*. Then* $\mathsf{dc}_{\mathcal{R}}$ *is bounded by a multiply recursive function.*

Proof. Let $\mathcal{R}_{\mathsf{sim}}$ be the simulating TRS for $\mathcal{R}$, as defined over the course of this section. Due to Lemma 6.28, $\mathsf{dc}_{\mathcal{R}_{\mathsf{sim}}}$ is multiply recursively bounded. Let t be a ground term. By Lemmata 6.35 and 6.36, we have the following inequalities:

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathsf{dh}(\mathsf{tr}(t), \to_{\mathcal{R}_{\mathsf{sim}}}) \leqslant \mathsf{dh}(\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}), \to_{\mathcal{R}_{\mathsf{sim}}})$$

Note that $|\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z})| \leqslant |t|$. Hence for all $n \in \mathbb{N}$, we have $\mathsf{dc}_{\mathcal{R}}(n) \leqslant \mathsf{dc}_{\mathcal{R}_{\mathsf{sim}}}(n)$. Thus, $\mathsf{dc}_{\mathcal{R}}$ is bounded by a multiply recursive function. □

For the general class of TRSs whose termination can be shown via Theorem 6.3 using a proof tree PT with a multiply recursive reduction pair function g, no better upper complexity bound than in Theorem 6.9 can be given, even if g is a constant function. This is demonstrated in Example 6.5: by Lemma 6.6 for any multiply recursive function f, there exists a k such that $\mathsf{dc}_{\mathcal{R}_{\mathsf{Pet}(k)}}$ dominates f, and $\mathsf{dc}_{\mathcal{R}_{\mathsf{Pet}(k)}}$ terminates by Theorem 6.3. Moreover, the proof tree based on the termination proof exhibited in Example 6.5 has a constant reduction pair function.

In fact, termination proofs by Theorem 6.3, even when restricted to proof trees with constant reduction pair functions, characterise the class of multiply recursive functions (if we lift the notion of characterisation from sets of reduction orders, as in Definition 2.38, to more general termination theorems). Let f be any multiply recursive function. Then f can be defined using only initial functions, composition, primitive recursion, and k-ary Ackermann functions. Let $\mathcal{R}$ be the TRS over $\mathcal{F}$ and $\mathcal{V}$ constructed directly from this definition. Then termination of $\mathcal{R}$ can be proved as follows: first, we apply the dependency graph processor $\Phi^{\mathsf{DG}}_{\mathsf{DG}(\mathsf{DP}(\mathcal{R}),\mathcal{R})}$ to the initial DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$. By inspection of the definitions of initial functions, composition, primitive recursion, and k-ary Ackermann functions, we see that all resulting DP problems have the shape $(\mathcal{Q}, \mathcal{R})$ with $\mathcal{Q} = \{h^\sharp(\mathsf{s}(y), x_1, \ldots, x_n) \to h^\sharp(y, x_1, \ldots, x_n)\}$ for some $h \in \mathcal{F}$, or $\mathcal{Q} = \mathsf{DP}(\mathcal{R}_{\mathsf{Pet}}(k))$ for some $k \in \mathbb{N}$, where $\mathcal{R}_{\mathsf{Pet}}(k)$ is defined as in Example 6.5. Whenever the first alternative holds for the DP problem $(\mathcal{Q}, \mathcal{R})$, it can obviously be completely solved by the subterm criterion processor Φ^{SC}_{π} with $\pi(h^\sharp) = 1$. If the second alternative holds for $(\mathcal{Q}, \mathcal{R})$, then its finiteness can be shown by a sequence of applications of subterm criterion processors, as in Example 6.5. For any combination of such DP problems resulting from the application of $\Phi^{\mathsf{DG}}_{\mathsf{DG}(\mathsf{DP}(\mathcal{R}),\mathcal{R})}$, termination of $\mathcal{R}$ by Theorem 6.3 follows. Moreover, the constant function $g(n) = k$, where k is the number of SCCs in $\mathsf{DG}(\mathsf{DP}(\mathcal{R}), \mathcal{R})$, is a reduction pair function of the resulting proof tree.

6.4 Dependency Graphs: a Lower Complexity Bound

In the previous section, we have shown a general multiply recursive upper bound on the derivational complexity of TRSs whose termination can be proved by Theorem 6.3, as long as there exists a multiply recursive reduction pair function for the proof tree based on the considered termination proof. We now try to obtain better complexity bounds by weakening the considered termination theorem. Specifically, we consider termination proofs whose proof trees have strictly limited height and restrictions on the applied DP processors. In this section, we consider the following termination theorem:

Theorem 6.37. *Let $\mathcal{R}$ be a TRS such that there exists a proof tree PT of $\mathcal{R}$. Suppose that each edge label of PT is a reduction pair or dependency graph processor. Moreover, whenever an edge starting from $(\mathcal{P}, \mathcal{R})$ is labelled by a reduction pair processor $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$, suppose that $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}((\mathcal{P}, \mathcal{R})) = \{(\emptyset, \mathcal{R})\}$. Then $\mathcal{R}$ is terminating.*

Proof. Direct consequence of Theorem 6.3. □

This theorem is significantly weaker than Theorem 6.3. Any considered proof tree PT (assuming that PT contains no “useless applications of DP processors”, i.e. edges with label Φ starting from a node $(\mathcal{P}, \mathcal{R})$ such that $\Phi((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}, \mathcal{R})\}$) is limited to at most height 2. Generally, here a dependency graph processor is applied to the DP problem at the root of PT, and a reduction pair processor (which completely solves the respective DP problem) is applied to each of the children of the root. Still, for any primitive recursive function f, there exists a TRS whose termination follows by Theorem 6.37, but whose derivational complexity grows faster than f. This is demonstrated in the next example.

Example 6.38. Consider the following family of TRSs, denoted as $\mathcal{R}_{\mathsf{Hof}}(l)$, and parametrised by $l \in \mathbb{N}$. Each TRSs $\mathcal{R}_{\mathsf{Hof}}(l)$ is a labelled version of Hofbauer’s TRS $\mathcal{R}_{\mathsf{Hof}}$ [50, 47], which we have used in Example 6.4.

$$\begin{aligned} \mathsf{i}(x) \circ_k (y \circ_{k-1} z) &\to x \circ_k (\mathsf{i}(\mathsf{i}(y)) \circ_{k-1} z) & 2 \leqslant k \leqslant l \\ \mathsf{i}(x) \circ_k (y \circ_{k-1} (z \circ_{k-2} w)) &\to x \circ_k (z \circ_{k-1} (y \circ_{k-2} w)) & 3 \leqslant k \leqslant l \end{aligned}$$

For all $m, n \geqslant 0$, set

$$t_{m,n} = \mathsf{i}^{2(n+1)}(\mathsf{e}) \circ_{m+2} (\mathsf{e} \circ_{m+1} (\ldots (\mathsf{e} \circ_1 \mathsf{e}) \ldots)) \,.$$

Then $\mathsf{dh}(t_{m,n}, \rightarrow_{\mathcal{R}_{\mathsf{Hof}}(l)}) \geqslant \mathsf{Ack}(m,n)$, whenever $l \geqslant m{+}2$. This follows from [47, Proposition 5.9]. Hence, for every primitive recursive function f there exists some l such that $\mathsf{dc}_{\mathcal{R}_{\mathsf{Hof}}(l)}$ dominates f.

On the other hand, consider the following termination proof of $\mathcal{R}_{\mathsf{Hof}}(l)$. The dependency pairs $\mathsf{DP}(\mathcal{R}_{\mathsf{Hof}}(l))$ of $\mathcal{R}_{\mathsf{Hof}}(l)$ are the following rules:

1_k:	$\mathsf{i}(x) \circ_k^\sharp (y \circ_{k-1} z) \rightarrow x \circ_k^\sharp (\mathsf{i}(\mathsf{i}(y)) \circ_{k-1} z)$	$2 \leqslant k \leqslant l$
2_k:	$\mathsf{i}(x) \circ_k^\sharp (y \circ_{k-1} z) \rightarrow \mathsf{i}(\mathsf{i}(y)) \circ_{k-1}^\sharp z$	$2 \leqslant k \leqslant l$
3_k:	$\mathsf{i}(x) \circ_k^\sharp (y \circ_{k-1} (z \circ_{k-2} w)) \rightarrow x \circ_k^\sharp (z \circ_{k-1} (y \circ_{k-2} w))$	$3 \leqslant k \leqslant l$
4_k:	$\mathsf{i}(x) \circ_k^\sharp (y \circ_{k-1} (z \circ_{k-2} w)) \rightarrow z \circ_{k-1}^\sharp (y \circ_{k-2} w)$	$3 \leqslant k \leqslant l$
5_k:	$\mathsf{i}(x) \circ_k^\sharp (y \circ_{k-1} (z \circ_{k-2} w)) \rightarrow y \circ_{k-2}^\sharp w$	$3 \leqslant k \leqslant l$

We apply the dependency graph processor $\Phi_{\mathcal{G}}^{\mathsf{DG}}$), where $\mathcal{G}$ is the dependency graph of the initial DP problem. We have $\Phi_{\mathcal{G}}^{\mathsf{DG}}((\mathsf{DP}(\mathcal{R}_{\mathsf{Hof}}(l)), \mathcal{R}_{\mathsf{Hof}}(l))) = \{(\{1_k, 3_k\}, \mathcal{R}_{\mathsf{Hof}}(l)) \mid 3 \leqslant k \leqslant l\} \cup \{(\{1_2\}, \mathcal{R}_{\mathsf{Hof}}(l))\}$. Each of the resulting $k-1$ DP problems is completely solved by the reduction pair processor based on the strongly linear interpretation $\mathcal{A}$. The interpretation $\mathcal{A}$ is defined by the following interpretation functions, where k ranges between 1 and l:

$$(\circ_k^\sharp)_{\mathcal{A}}(m,n) = m \qquad (\circ_k)_{\mathcal{A}}(m,n) = 0 \qquad \mathsf{i}_{\mathcal{A}}(m) = m + 1$$

Putting everything together, we obtain a proof tree which uses only $\Phi_{\mathcal{G}}^{\mathsf{DG}}$ and $\Phi_{(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})}^{\mathsf{RP}}$. Thus we conclude termination of $\mathcal{R}_{\mathsf{Hof}}(l)$ by Theorem 6.37.

Note the contrast between the derivational complexities of the TRSs $\mathcal{R}_{\mathsf{Hof}}(l)$ and the complexity induced by the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$: for every primitive recursive function f, there exists some $l \in \mathbb{N}$ such that $\mathsf{dc}_{\mathcal{R}_{\mathsf{Hof}}(l)}$ grows faster than f. On the other hand, the identity function $f(n) = n$ is a reduction pair function of each of the proof trees constructed from the termination proofs of the TRSs $\mathcal{R}_{\mathsf{Hof}}(l)$.

Remark 6.39. In [80, Section 6] it was falsely claimed that the derivational complexity of a TRS whose termination can be proved by Theorem 6.37 would have an upper bound which is *elementary* in the complexity of any reduction pair function of a proof tree for that TRS based on a termination proof by Theorem 6.37. Example 6.38 clearly contradicts this claim.

6.5 Dependency Graphs: an Upper Complexity Bound

Example 6.38 shows that the lowest possible upper bound on the derivational complexity of TRSs whose termination can be proved by Theorem 6.37 is the class of primitive recursive functions. In this section, we show that the derivational complexity of TRSs from this class is indeed bounded primitive recursively in any reduction pair function of a proof tree for that TRS based on a termination proof by Theorem 6.37. We prove the following theorem in the remainder of this section:

Theorem 6.40. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.37 using a proof tree* PT*, and let g be a reduction pair function of* PT*. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded by a function which is primitive recursive in g.*

The proof of Theorem 6.40 uses essentially the same ideas as the proof of Theorem 6.9: based on the mapping norm as defined in Definition 6.18, we define a TRS $\mathcal{R}_{\mathsf{DG\,sim}}$ simulating the initial TRS $\mathcal{R}$. Again, the simulation is based on a suitable interpretation $\mathsf{tr}_{\mathsf{DG}}$, i.e. $s \to_{\mathcal{R}} t$ implies $\mathsf{tr}_{\mathsf{DG}}(s) \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(t)$. Finally, we can simply prove an upper bound on the derivational complexity of $\mathcal{R}_{\mathsf{DG\,sim}}$ in order to obtain an upper bound on $\mathsf{dc}_{\mathcal{R}}$.

However, we want to show a tighter complexity bound than in Section 6.3 here. Therefore, a termination proof of $\mathcal{R}_{\mathsf{DG\,sim}}$ by an LPO combined with Theorem 4.42 is not sufficient for the objective of this section. The main ingredients of the simulation, $\mathsf{tr}_{\mathsf{DG}}$ and $\mathcal{R}_{\mathsf{DG\,sim}}$, need to differ from the mapping tr and the TRS $\mathcal{R}_{\mathsf{sim}}$ used in the proof of Theorem 6.9. This finally allows us to give a termination proof of the simulating TRS which is much simpler from a complexity-theoretic point of view than the termination proof given in the proof of Lemma 6.28.

Notation 6.41. For the remainder of this section, we fix a TRS $\mathcal{R}$ such that termination of $\mathcal{R}$ follows by Theorem 6.37 using some complexity proof tree PT. We assume without loss of generality for PT that the edges labelled by a dependency graph processor are exactly those edges starting from the root node, and fix $\Phi^{\mathsf{DG}}_{\mathcal{G}}$ to be that processor.

We fix a reduction pair function g of PT. Let k be the number of (trivial and nontrivial) SCCs in $\mathcal{G}$, A the maximum arity of any function symbol occurring in $\mathcal{R}$, and $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

Due to our assumptions on PT, for any term t, the norm of t has the shape (u_1, u_2, u_3) with $u_1 \in \{\bot, 0, 1, \dots, k\}$, $u_2 \in \mathbb{N} \cup \{\bot\}$, and $u_3 \in \{0, \bot\}$. This simplifies the simulation of derivations in $\mathcal{R}$ considerably. Moreover, note that whenever $\mathsf{norm}(s) \sqsupset \mathsf{norm}(t)$ for some terms s and t, the lexicographic decrease must happen in one of the first two components: if $\mathsf{norm}_3(s) \gg \mathsf{norm}_3(t)$, then $\mathsf{norm}_3(s) = 0$ and $\mathsf{norm}_3(t) = \bot$. Since $\mathsf{norm}_1(u) = \bot$ and $\mathsf{norm}_2(u) = \bot$ if and only if $\mathsf{norm}_3(u) = \bot$ for all terms u, we also have $\mathsf{norm}_1(s) \gg \mathsf{norm}_1(t)$ and $\mathsf{norm}_2(s) \gg \mathsf{norm}_2(t)$ in that case. Thus, we have:

$$\begin{aligned}
\mathsf{norm}(s) \sqsupset \mathsf{norm}(t) &\Longrightarrow (\mathsf{norm}_1(s), \mathsf{norm}_2(s)) \sqsupset (\mathsf{norm}_1(t), \mathsf{norm}_2(t))\\
\mathsf{norm}(s) \sqsupseteq \mathsf{norm}(t) &\Longrightarrow (\mathsf{norm}_1(s), \mathsf{norm}_2(s)) \sqsupseteq (\mathsf{norm}_1(t), \mathsf{norm}_2(t)) \,.
\end{aligned}$$

Moreover, note that on this restricted range of norm, the orders $\sqsupset$ and $\gg^{\mathrm{lex}}$ coincide.

Based on these simplifying assumptions, we now build the interpretation $\mathsf{tr}_{\mathsf{DG}}$ and the simulating TRS $\mathcal{R}_{\mathsf{DG\,sim}}$. Given a term t, $\mathsf{tr}_{\mathsf{DG}}$ employs a $1 + A$-ary function symbol f_i. The index i of f_i and the first argument of f_i are used to represent $\mathsf{norm}_1(t)$ and $\mathsf{norm}_2(t)$, respectively; the last A arguments of f_i

are used to represent $\mathsf{tr}_{\mathsf{DG}}(t')$ for each direct subterm t' of t. Analogous to Definition 6.27, we use some abbreviations.

Notation 6.42. We make use of the following abbreviation M_i^j (for $j \in \mathbb{N}$ and $i \in \{0, \dots, k\}$):

$$\begin{aligned} M_i^0(u, x_1, \dots, x_A) &= \mathsf{f}_i(u, x_1, \dots, x_A) \\ M_i^{j+1}(u, x_1, \dots, x_A) &= \mathsf{f}_i(u, M_i^j(u, x_1, \dots, x_A), \dots, M_i^j(u, x_1, \dots, x_A)) \end{aligned}$$

We write $\overline{t}$ to denote $t, \dots, t$, where the number of repetitions of t follows from the context.

Consider the reduction pair function g of $\mathcal{R}$. If g is computable, then it is an easy exercise to define a terminating TRS $\mathcal{R}'_{\mathsf{DG\,sim}}$ (employing the constructors s and $\mathsf{0}$) which computes the function g (i.e. $\mathsf{g}(\mathsf{s}^n(\mathsf{0})) \rightarrow^*_{\mathcal{R}'_{\mathsf{DG\,sim}}} \mathsf{s}^{g(n)}(\mathsf{0})$) such that $\mathsf{dc}_{\mathcal{R}'_{\mathsf{DG\,sim}}}$ is primitive recursive in g [38]. Moreover, if g is primitive recursive, we assume that $\mathcal{R}'_{\mathsf{DG\,sim}}$ is constructed as follows: define g using only initial functions, composition, and primitive recursion, and directly turn the resulting definition of g into rewrite rules.

Definition 6.43. Consider the following (schematic) TRS $\mathcal{R}_{\mathsf{DG\,sim}}$, where $0 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant A$.

$$\begin{aligned}
1_i\colon &\quad & \mathsf{f}_i(\mathsf{s}(u), x_1, \dots, x_A) &\to M_i^C(u, x_1, \dots, x_A) \\
2_i\colon && \mathsf{f}_{i+1}(u, x_1, \dots, x_A) &\to \mathsf{f}_i(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{0}, x_1, \dots, x_A)))), x_1, \dots, x_A) \\
3_{i,j}\colon && \mathsf{size}(\mathsf{f}_i(u, x_1, \dots, x_A)) &\to \times_A(\mathsf{size}(x_j)) \\
4\colon && \mathsf{size}(x) &\to \mathsf{s}(\mathsf{0}) \\
5\colon && \times_A(\mathsf{s}(x)) &\to \mathsf{s}^A(\times_A(x)) \\
6\colon && \times_A(\mathsf{0}) &\to \mathsf{0} \\
7_i\colon && \mathsf{f}_i(u, x_1, \dots, x_A) &\to \mathsf{c} \\
8_{i,j}\colon && \mathsf{f}_i(u, x_1, \dots, x_A) &\to x_j \\
9\colon && \mathsf{h}(x) &\to \mathsf{f}_k(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{0}, \overline{x})))), \overline{x}) \\
10\colon && \mathsf{z} &\to \mathsf{f}_k(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{0}, \overline{\mathsf{c}})))), \overline{\mathsf{c}})
\end{aligned}$$

These rules are augmented by a TRS $\mathcal{R}'_{\mathsf{DG\,sim}}$ defining the function symbol g. The signatures of $\mathcal{R}'_{\mathsf{DG\,sim}}$ and $\mathcal{R}_{\mathsf{DG\,sim}} \setminus \mathcal{R}'_{\mathsf{DG\,sim}}$ are disjoint with the exception of g and the constructors s and $\mathsf{0}$.

The rules of $\mathcal{R}_{\mathsf{DG\,sim}}$ match the rules of $\mathcal{R}_{\mathsf{sim}}$ (see Definition 6.27) in spirit. The rules 1_i–2_i are the main rules for the simulation of the effects of a single step $s \to_{\mathcal{R}} t$ in $\mathcal{R}_{\mathsf{DG\,sim}}$. Similar to rules 1_i–4_i of $\mathcal{R}_{\mathsf{sim}}$, they execute the lexicographic decrease of the norm of s, and are responsible for creating the at most $(C{+}1){\cdot}A^C$ many new positions and copies of each subterm of s in t. The rules $3_{i,j}$–6 define the function symbol size, just as the rules 5_j–8 of $\mathcal{R}_{\mathsf{sim}}$ do. The rules 7_i–$8_{i,j}$ make sure that any superfluous positions and copies of subterms can be deleted , see rules 9–10_j of $\mathcal{R}_{\mathsf{sim}}$. Finally, the rules 9 and 10 guarantee that the simulating derivation can be started with a suitably small initial term, similar to rules 11 and 12 of $\mathcal{R}_{\mathsf{sim}}$.

Analogous to Section 6.3, the derivational complexity of $\mathcal{R}_{\mathsf{DG\,sim}}$ can be analysed directly. However, due to the lower complexity bound we would like to show, this analysis is more involved than the proof of Lemma 6.28, therefore the proof of the following lemma is deferred to Sections 6.5.1 and 6.5.2 below.

Lemma 6.44. *The derivational complexity of* $\mathcal{R}_{\mathsf{DG\,sim}}$ *is bounded by a function which is primitive recursive in* g.

Notation 6.45. For the remainder of this section, let $\mathcal{F}_{\mathsf{DG\,sim}}$ denote the signature of $\mathcal{R}_{\mathsf{DG\,sim}}$.

In the following, we show that the TRS $\mathcal{R}_{\mathsf{DG\,sim}}$ indeed simulates $\mathcal{R}$ as requested.

Definition 6.46. The mapping $\mathsf{tr}_{\mathsf{DG}}\colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F}_{\mathsf{DG\,sim}},\mathcal{V})$ is defined by the following equation:

$$\mathsf{tr}_{\mathsf{DG}}(t) = \begin{cases} t & \text{if } t \in \mathcal{V} \\ \mathsf{f}_{\mathsf{norm}_1(t)}(\mathsf{norm}_2(t)^*, \mathsf{tr}_{\mathsf{DG}}(t_1), \ldots, \mathsf{tr}_{\mathsf{DG}}(t_n), \bar{\mathsf{c}}) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Here we understand $\mathsf{f}_\perp$ to be f_0. Moreover, the operator $(\cdot)^*$ is defined as follows:

$$u^* = \begin{cases} \mathsf{0} & \text{if } u = \perp \\ \mathsf{s}^{u+1}(\mathsf{0}) & \text{if } u \in \mathbb{N} \end{cases}$$

Definition 6.47. We define an equivalence $\approx_{\mathsf{DG}}$ on $\mathcal{T}(\mathcal{F}_{\mathsf{DG\,sim}},\mathcal{V})$. We have $s \approx_{\mathsf{DG}} t$ if and only if one of the following properties holds:

1. $s = t$, or
2. $s = \mathsf{f}_i(u, s_1, \ldots, s_A)$, $t = \mathsf{f}_{i'}(v, t_1, \ldots, t_A)$, and $s_j \approx t_j$ for all $1 \leqslant j \leqslant A$.

Lemma 6.48. *For all terms s and t with $\mathsf{tr}_{\mathsf{DG}}(s) \approx_{\mathsf{DG}} t$, we have $\mathsf{size}(t) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{s}^m(\mathsf{0})$ for some $m \geqslant |s|$.*

Proof. Analogous to Lemma 6.32, using rules $3_{i,j}$–6 from $\mathcal{R}_{\mathsf{DG\,sim}}$ in place of rules 5_j–8 from $\mathcal{R}_{\mathsf{sim}}$. □

Lemma 6.49. *The following properties of* $\mathcal{R}_{\mathsf{DG\,sim}}$ *hold:*

1. *If $s = \mathsf{f}_i(u^*, s_1, \ldots, s_A)$, $t = \mathsf{tr}_{\mathsf{DG}}(t') = \mathsf{f}_{i'}(v^*, s_1, \ldots, s_A)$, and $(i,u) \sqsupseteq (i',v)$, then $s \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} t$.*
2. *For any terms $s = \mathsf{tr}_{\mathsf{DG}}(s')$ and $t = \mathsf{tr}_{\mathsf{DG}}(t')$, $s' \xrightarrow{\epsilon}_{\mathcal{R}} t'$ implies $s \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} t$.*
3. *If $a \to_{\mathcal{R}} b$ and $\mathsf{tr}_{\mathsf{DG}}(a) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(b)$, then for any n-ary function symbol $f \in \mathcal{F}$, $\mathsf{tr}_{\mathsf{DG}}(f(t_1, \ldots, a, \ldots, t_n)) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(f(t_1, \ldots, b, \ldots, t_n))$.*

Proof. We prove the three items in succession.

1. In order to show Property 1, observe that it suffices to show the following items for all $1 \leqslant i \leqslant d$ (compare Definitions 6.19 and 6.46). Also note that our identification of f_0 and $\mathsf{f}_\perp$ poses no problem, since $\mathsf{norm}_1(t') = \perp$ implies $\mathsf{norm}_2(t') = \perp$, and $\mathsf{norm}_1(t') \neq \perp$ implies $\mathsf{norm}_2(t') \gg \perp$.

   - $\mathsf{f}_i(\mathsf{s}(u), x_1, \ldots, x_A) \rightarrow^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_i(u, x_1, \ldots, x_A)$
   - $\mathsf{f}_{i+1}(u, x_1, \ldots, x_A)$
     $\rightarrow^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_i(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(0, x_1, \ldots, x_A)))), x_1, \ldots, x_A)$

   The first item follows directly by applying rules 1_i and $8_{i,1}$ of $\mathcal{R}_{\mathsf{DG\,sim}}$. The second item follows by a single application of rule 2_i from $\mathcal{R}_{\mathsf{DG\,sim}}$.

2. We now show Property 2. Let $l \to r$ be the rewrite rule and σ the substitution applied in the step $s' \xrightarrow{\epsilon}_{\mathcal{R}} t'$. Let $i = \mathsf{norm}_1(s')$ and $m = \mathsf{norm}_2(s')$. Since l is not a variable, we have $l = f(l_1, \ldots, l_n)$. Hence, $\mathsf{tr}_{\mathsf{DG}}(l\sigma) = \mathsf{f}_i(\mathsf{s}^{m+1}(0), \mathsf{tr}_{\mathsf{DG}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(l_n\sigma), \bar{\mathsf{c}})$. Since s' is a redex, we have $i \gg 0$ and $m \gg 0$. By rules 1_i and $8_{i,1}$, we have

$$\mathsf{tr}_{\mathsf{DG}}(l\sigma) \rightarrow^+_{\mathcal{R}_{\mathsf{DG\,sim}}} M_i^{\mathsf{dp}(r)}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DG}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(l_n\sigma), \bar{\mathsf{c}}) \ .$$

We show the following claim by induction on $\mathsf{dp}(u)$.

Claim 6.50. *We have $M_i^{\mathsf{dp}(u)}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DG}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(l_n\sigma), \bar{\mathsf{c}}) \rightarrow^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(u\sigma)$ whenever $u \trianglelefteq r$.*

Since $r \trianglelefteq r$, showing the claim suffices to conclude Property 2 of the lemma. Hence, the remainder of this proof is devoted to showing the claim. We perform case distinction on u.

Suppose $u \vartriangleleft l$. Then there exists some $j \in \{1, \ldots, n\}$ with $u \trianglelefteq l_j$. Using rule $8_{i,1}$, we get

$$M_i^{\mathsf{dp}(u)}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DG}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(l_n\sigma), \bar{\mathsf{c}}) \rightarrow^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(l_j\sigma) \ .$$

Further applications of rule $8_{i',j'}$ for some values of i' and j' then yield $\mathsf{tr}_{\mathsf{DG}}(l_j\sigma) \rightarrow^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(u\sigma)$, which concludes the first case.

Now suppose that $u \not\vartriangleleft l$. Since variables occurring in u also occur in r and hence in l, the condition $u \not\vartriangleleft l$ implies that u is not a variable. Hence, $u = h(u_1, \ldots, u_{n'})$. Let $i' = \mathsf{norm}_1(u\sigma)$ and $m' = \mathsf{norm}_2(u\sigma)$. By induction hypothesis and employing rule $8_{i,1}$ $\mathsf{dp}(u) - 1 - \mathsf{dp}(u_j)$, many times, we have for all $1 \leqslant j \leqslant n'$

$$M_i^{\mathsf{dp}(u)-1}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DG}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(l_n\sigma), \bar{\mathsf{c}}) \rightarrow^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(u_j\sigma) \ .$$

By combining this with $A - n'$ many application of rule 7_i, we obtain

$$\begin{aligned} &M_i^{\mathsf{dp}(u)}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DG}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(l_n\sigma), \bar{\mathsf{c}}) \\ &\qquad \rightarrow^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_i(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DG}}(u_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DG}}(u_{n'}\sigma), \bar{\mathsf{c}}) \ . \end{aligned}$$

By Lemma 6.24, we have $\mathsf{norm}(s') \sqsupset \mathsf{norm}(u\sigma)$, and therefore $(i, m) \sqsupset (i', m')$. By Property 1, we have

$$\mathsf{f}_i(\mathsf{s}^m(\mathsf{0}), \mathsf{tr}_{\mathsf{DG}}(u_1\sigma), \dots, \mathsf{tr}_{\mathsf{DG}}(u_{n'}\sigma), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(u\sigma) ,$$

so the claim and thus Property 2 follow.

3. We now prove Property 3. Let $i = \mathsf{norm}_1(f(t_1, \dots, a, \dots, t_n))$ and $m = \mathsf{norm}_2(f(t_1, \dots, a, \dots, t_n))$. Further, let $i' = \mathsf{norm}_1(f(t_1, \dots, b, \dots, t_n))$ and $m' = \mathsf{norm}_2(f(t_1, \dots, b, \dots, t_n))$. By assumption,

$$\begin{aligned}&\mathsf{f}_i(m, \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(a), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n))\\ &\quad \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_i(m, \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(b), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n)) .\end{aligned}$$

By Lemma 6.22, we have $(i, m) \sqsupseteq (i', m')$. Therefore, by Property 1, it follows that

$$\begin{aligned}&\mathsf{f}_i(m, \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(b), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n))\\ &\quad \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_{i'}(m', \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(b), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n)) .\end{aligned}$$

Again, we conclude Property 3 and thus the lemma.

□

Lemma 6.51. *For any terms s and t, $s \to_{\mathcal{R}} t$ implies* $\mathsf{tr}_{\mathsf{DG}}(s) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(t)$.

Proof. Easy consequence of Lemma 6.49.2 and 6.49.3. □

Lemma 6.52. *For any ground term t, we have* $\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(t)$.

Proof. We proceed by induction on $\mathsf{dp}(t)$. If $\mathsf{dp}(t) = 0$, then t is a constant. Rule 10 yields the rewrite step $\mathsf{z} \to_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_k(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{0}, \bar{\mathsf{c}})))), \bar{\mathsf{c}})$. Let $i = \mathsf{norm}_1(t)$ and $m = \mathsf{norm}_2(t)$. By definition, we have $k + 1 \gg i$, $g(1) + 1 \gg m$, and $\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{0}, \bar{\mathsf{c}}))) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{s}^{g(1)+1}(\mathsf{0})$. By Lemma 6.49.1, we have

$$\mathsf{f}_k(\mathsf{s}^{g(1)+1}(\mathsf{0}), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_i(\mathsf{s}^{m+1}, \bar{\mathsf{c}}) = \mathsf{tr}_{\mathsf{DG}}(t) ,$$

hence the lemma follows.

Now assume $\mathsf{dp}(t) > 0$, so t has the shape $f(t_1, \dots, t_n)$. Then rule 9 of $\mathcal{R}_{\mathsf{DG\,sim}}$ yields the rewrite step

$$\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}) \to_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_k(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{0}, \overline{\mathsf{h}^{\mathsf{dp}(t)-1}(\mathsf{z})})))), \overline{\mathsf{h}^{\mathsf{dp}(t)-1}(\mathsf{z})}) .$$

Using rules 9 and $8_{k,1}$, we obtain $\mathsf{h}^{\mathsf{dp}(t)-1}(\mathsf{z}) \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{h}^{\mathsf{dp}(t_j)}(\mathsf{z})$ for all $1 \leqslant j \leqslant n$, and by induction hypothesis, $\mathsf{h}^{\mathsf{dp}(t_j)}(\mathsf{z}) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{tr}_{\mathsf{DG}}(t_j)$. Therefore,

$$\begin{aligned}&\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z})\\ &\quad \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_k(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n), \bar{\mathsf{c}})))), \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n), \bar{\mathsf{c}}) .\end{aligned}$$

Let $i = \mathsf{norm}_1(t)$ and $m = \mathsf{norm}_2(t)$. By definition, we have $k + 1 \gg i$, $g(|t|) + 1 \gg m$, and $\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(\mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n), \bar{\mathsf{c}})))) \to^+_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{s}^{m'+1}(\mathsf{0}) = m'^*$ for some $m' \geqslant |t|$. Since by definition, $(k, m') \sqsupseteq (i, m)$, it follows by Lemma 6.49.1 that

$$\begin{aligned}&\mathsf{f}_k(\mathsf{s}^{m'+1}(\mathsf{0}), \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n), \bar{\mathsf{c}})\\ &\quad \to^*_{\mathcal{R}_{\mathsf{DG\,sim}}} \mathsf{f}_i(\mathsf{s}^{m+1}, \mathsf{tr}_{\mathsf{DG}}(t_1), \dots, \mathsf{tr}_{\mathsf{DG}}(t_n), \bar{\mathsf{c}}) = \mathsf{tr}_{\mathsf{DG}}(t) \ ,\end{aligned}$$

concluding the lemma. □

Now we can prove the main result of this section, Theorem 6.40.

Theorem. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.37 using a proof tree* PT*, and let g be a reduction pair function of* PT*. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded by a function which is primitive recursive in g.*

Proof. Let $\mathcal{R}_{\mathsf{DG\,sim}}$ be the simulating TRS for $\mathcal{R}$, as defined over the course of this section. By Lemma 6.44, $\mathsf{dc}_{\mathcal{R}_{\mathsf{DG\,sim}}}$ is bounded by a function which is primitive recursive in g. Let t be a ground term. Due to Lemmata 6.51 and 6.52, we have the following inequalities:

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathsf{dh}(\mathsf{tr}_{\mathsf{DG}}(t), \to_{\mathcal{R}_{\mathsf{DG\,sim}}}) \leqslant \mathsf{dh}(\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z}), \to_{\mathcal{R}_{\mathsf{DG\,sim}}})$$

Note that $|\mathsf{h}^{\mathsf{dp}(t)}(\mathsf{z})| \leqslant |t|$. Hence for all $n \in \mathbb{N}$, we have $\mathsf{dc}_{\mathcal{R}}(n) \leqslant \mathsf{dc}_{\mathcal{R}_{\mathsf{DG\,sim}}}(n)$. Thus, $\mathsf{dc}_{\mathcal{R}}$ is bounded primitive recursively in g. □

The result given in Theorem 6.40 is essentially optimal for the class of TRSs whose termination can be shown via Theorem 6.37. As demonstrated in Example 6.38, for every primitive recursive function f, there exists some $l \in \mathbb{N}$ such that $\mathsf{dc}_{\mathcal{R}_{\mathsf{Hof}}(l)}$ grows faster than f, and $\mathcal{R}_{\mathsf{Hof}}(l)$ terminates by Theorem 6.37 using a proof tree which has a linear reduction pair function.

6.5.1 Complexity of the Simulating TRS: First Proof

In the above proof of Theorem 6.40, one issue remains open up to now: we have postponed the proof of Lemma 6.44, which asserts an upper bound on the derivational complexity of the simulating TRS $\mathcal{R}_{\mathsf{DG\,sim}}$. In the sequel, we give two different proofs. The first proof only manages to prove a slightly weaker lemma, but we still exhibit it because sticks closer in spirit to the ideas of Section 6.3, and particularly to the proof of Lemma 6.28. On the other hand, the second proof is a direct proof of Lemma 6.44.

The remainder of this subsection is devoted to the first of these two proofs. This proof shows the following slightly weaker variant of Lemma 6.44:

Lemma 6.53. *If g is primitive recursive, then the derivational complexity of $\mathcal{R}_{\mathsf{DG\,sim}}$ is bounded by a primitive recursive function.*

The crucial observation in the proof of Lemma 6.28 was that $\mathcal{R}_{\mathsf{sim}}$ is compatible with a LPO. The TRS $\mathcal{R}_{\mathsf{DG\,sim}}$ is compatible with a LPO, as well, as long as g is a multiply recursive function. However, this alone is not sufficient to conclude that $\mathsf{dc}_{\mathcal{R}_{\mathsf{DG\,sim}}}$ is primitive recursive.

Theoretically, Theorem 4.38, which shows that compatibility of a TRS with a MPO is sufficient to conclude that its derivational complexity is primitive recursive, would be a remedy. Of course, neither $\mathcal{R}_{\mathsf{sim}}$ nor $\mathcal{R}_{\mathsf{DG\,sim}}$ is compatible with a MPO; the crucial difference of LPOs from MPOs (the lexicographic comparison of function arguments, compare Clause 3 of Definition 4.40, rather than a multiset comparison, as in Clause 3 of Definition 4.35) is essential for orienting rules 1_i–4_i of $\mathcal{R}_{\mathsf{sim}}$, and for orienting the family of rules 1_i of $\mathcal{R}_{\mathsf{DG\,sim}}$. In rules 1_i–4_i, the strict part of the lexicographic decrease may occur in any of the first d argument positions of f, where d is the depth of the considered complexity proof tree plus one. On the other hand, all complexity proof trees considered in this section have height 2. Moreover, the range of norm_1 is finite (it is $\{\bot, 0, 1, \ldots, k\}$), hence $\mathsf{tr}_{\mathsf{DG}}(t)$ encodes $\mathsf{norm}_1(t)$ in the index i of the function symbol f_i rather than its first argument position (which would be analogous to what tr does). As a result, in the orientation of the family of rules 1_i of $\mathcal{R}_{\mathsf{DG\,sim}}$ by a suitable LPO, the strict decrease in the lexicographic comparison of function arguments always occurs in the first argument position of f_i. This is roughly akin to *simple nested recursion*, a recursion schema which is syntactically slightly more liberal than primitive recursion. It has been shown that the primitive recursive functions are closed under simple nested recursion [89], so it is to be expected that "LPOs with lexicographic decreases only in the first argument position" induce primitive recursive complexity, as well. Indeed, in an unpublished manuscript [112], Weiermann formally introduced such a subclass of LPOs called *generalised ramified lexicographic path orders* (GRLPOs for short) and showed that any GRLPO induces primitive recursive complexity.

We give the definition of GRLPOs and recapitulate the main result of [112]. We start with an auxiliary concept used in Definition 6.55 below.

Definition 6.54 ([112]). Let $\succ$ be a binary relation on a subset of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ for some signature $\mathcal{F}$ and some set of variables $\mathcal{V}$, and $>$ a precedence on $\mathcal{F}$. Let $f \in \mathcal{F}$ and $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. We define $\mathcal{F}^{<f} = \{g \in \mathcal{F} \mid f > g\}$. Moreover, we inductively define $\mathcal{T}(\mathcal{F}^{<f} \cup \{f(\prec s)\}, \mathcal{V})$ as the smallest subset of $\mathcal{T}(\mathcal{F}, \mathcal{V})$ satisfying the following conditions:

- $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}^{<f} \cup \{f(\prec s)\}, \mathcal{V})$
- if $f > h$ and $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{F}^{<f} \cup \{f(\prec s)\}, \mathcal{V})$, then $h(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F}^{<f} \cup \{f(\prec s)\}, \mathcal{V})$
- if $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}^{<f} \cup \{f(\prec s)\}, \mathcal{V})$ and $s \succ t_1$, then $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}^{<f} \cup \{f(\prec s)\}, \mathcal{V})$

Definition 6.55 ([112]). Based on a precedence $>$ over some signature $\mathcal{F}$, we define an auxiliary relation $>_{\mathrm{grlpo}'}$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ for some set of variables $\mathcal{V}$ as follows. We have $s >_{\mathrm{grlpo}'} t$ if one of the following alternatives holds:

1. $s = f(s_1, \ldots, s_n)$ such that $s_i \geqslant_{\mathrm{grlpo}'} t$ for some $1 \leqslant i \leqslant n$
2. $s = f(s_1, \ldots, s_n)$ and $t = h(t_1, \ldots, t_m)$ such that $f > h$ and $s >_{\mathrm{grlpo}'} t_i$ for all $1 \leqslant i \leqslant m$

3. $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ such that either
   a) there exists $i \in \{2, \ldots, n\}$ such that for all $1 \leqslant j \leqslant i-1$, we have $s_j = t_j$, $s_i >_{\mathrm{grlpo}'} t_i$, and for all $i{+}1 \leqslant j \leqslant n$, we have $t_j \in \mathcal{T}(\mathcal{F}^{<f}, \mathcal{V})$ and $\mathcal{V}\mathsf{ar}(t_j) \subseteq \mathcal{V}\mathsf{ar}(s)$, or
   b) $s_1 >_{\mathrm{grlpo}'} t_1$, and $t_i \in \mathcal{T}(\mathcal{F}^{<f} \cup \{f(<_{\mathrm{grlpo}'} s_1)\}, \mathcal{V})$ and $\mathcal{V}\mathsf{ar}(t_i) \subseteq \mathcal{V}\mathsf{ar}(s)$ for all $2 \leqslant i \leqslant n$

Here $\geqslant_{\mathrm{grlpo}'}$ denotes the reflexive closure of $>_{\mathrm{grlpo}'}$. We set $s >_{\mathrm{grlpo}} t$ if and only if there exist terms $s', t' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and a substitution σ such that $s = s'\sigma$, $t = t'\sigma$, and $s' >_{\mathrm{grlpo}'} t'$.

Theorem 6.56 ([112, Theorem 2.3(5,6) and Theorem 3.1]). *Let $\mathcal{R}$ be a TRS and $>$ a precedence. If $\mathcal{R}$ is compatible with $>_{\mathrm{grlpo}}$, then $\mathcal{R}$ is terminating, and $\mathsf{dc}_{\mathcal{R}}$ is bounded by a primitive recursive function.*

Proof Sketch. First, it is shown that $s >_{\mathrm{grlpo}} t$ implies $s >_{\mathsf{LPO}} t$ for all terms s and t. Then termination of $\mathcal{R}$ follows by Theorem 4.41. In order to prove the complexity bound on $\mathcal{R}$, based on $>_{\mathrm{grlpo}}$ and $\mathcal{R}$, a well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$ with the following properties is defined: the carrier of $\mathcal{A}$ is $\mathbb{N}$, $>_{\mathbb{N}}$ is the usual order on $\mathbb{N}$, for every function symbol $f \in \mathcal{F}$, the interpretation function $f_{\mathcal{A}}$ is primitive recursive, and $\mathcal{R}$ is compatible with the resulting reduction order $>_{\mathbb{N}\mathcal{A}}$. □

Using Theorem 6.56, we can now show the lemma we wanted to show:

Lemma. *If g is primitive recursive, then the derivational complexity of $\mathcal{R}_{\mathsf{DG\,sim}}$ is bounded by a primitive recursive function.*

Proof. Since g is primitive recursive, by assumption, the TRS $\mathcal{R}'_{\mathsf{DG\,sim}}$ is defined by a translation of initial functions, composition, and primitive recursion into rewrite rules. It is straightforward to check that all instances of the definition schemata for initial rules, composition, and primitive recursion can be oriented by a GRLPO. Hence, $\mathcal{R}'_{\mathsf{DG\,sim}}$ is compatible with a GRLPO $>_{\mathrm{grlpo}}$. Moreover, the precedence $>$ of the GRLPO includes $\mathsf{g} > \mathsf{s} > \mathsf{0}$. If we extend this precedence by

$$\mathsf{h}, \mathsf{z} > \mathsf{f}_{i+1} > \mathsf{f}_i > \mathsf{g}, \mathsf{size} > \times_A > \mathsf{s}, \mathsf{0}, \mathsf{c}$$

for all $0 \leqslant i \leqslant k$, then the whole TRS $\mathcal{R}_{\mathsf{DG\,sim}}$ is compatible with $>_{\mathrm{grlpo}}$. Thus by Theorem 6.56, $\mathsf{dc}_{\mathcal{R}_{\mathsf{DG\,sim}}}$ is bounded by a primitive recursive function. □

6.5.2 Complexity of the Simulating TRS: Second Proof

The proof of Lemma 6.53 presented in Section 6.5.1 above has its merits as the logical extension of the complexity proof of $\mathcal{R}_{\mathsf{sim}}$ in Lemma 6.28. However, it also has some shortfalls: it crucially relies on results which - to our best knowledge - can only be found in an unpublished paper, and, with Lemma 6.53, it produces a weaker result than Lemma 6.44. Hence, in the sequel we give an alternative proof of an upper bound of $\mathcal{R}_{\mathsf{DG\,sim}}$, resulting in a proof of Lemma 6.44. Recall Lemma 6.44:

Lemma. *The derivational complexity of $\mathcal{R}_{\mathsf{DG\,sim}}$ is bounded by a function which is primitive recursive in g.*

The proof of this lemma is based on the same principle as the proofs of Theorems 4.38, 4.42, and 6.56 given in [48, 113, 112] are. We define a well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$ with carrier $\mathbb{N} \setminus \{0\}$ such that $>_{\mathbb{N}}$ is the usual strict order on $\mathbb{N}$, each interpretation function $f_{\mathcal{A}}$ is primitive recursive in g, and $\mathcal{R}$ is compatible with the resulting reduction order $>_{\mathbb{N}\mathcal{A}}$. Since for each term t, the interpretation $[\alpha]_{\mathcal{A}}(t)$ is primitive recursive in g and $|t|$, and each rewrite step $s \to_{\mathcal{R}_{\mathsf{DG\,sim}}} t$ implies $[\alpha]_{\mathcal{A}}(s) >_{\mathbb{N}} [\alpha]_{\mathcal{A}}(t)$, it then follows that $\mathsf{dc}_{\mathcal{R}_{\mathsf{DG\,sim}}}$ is primitive recursive in g.

Some interpretation functions of $\mathcal{A}$ make use of a family of fast growing functions, which is defined below. This definition is parametrised in some natural number d. The exact value of the parameter d will become clear below. To simplify the notation we assume that the function g is primitive recursive. Otherwise Definition 6.57 has to be replaced by a function hierarchy that is parametrised in g.

Definition 6.57. Let $d \geqslant 2$ be some given natural number. We define:

$$\mathsf{F}_0(m) = d^{m+1} \qquad \mathsf{F}_{n+1}(m) = \mathsf{F}_n^{m+1}(m) \ .$$

The following properties of the family of functions $\{\mathsf{F}_n \mid n \in \mathbb{N}\}$ are straightforward to verify.

Lemma 6.58. *Let n, m, a, and b be natural numbers.*

1. $\mathsf{F}_n(a) \geqslant d^{a+1} \geqslant d \cdot a > a$.
2. *If $a > b$, then $\mathsf{F}_n(a) > \mathsf{F}_n(b)$.*
3. *If $n > m$ and $a \geqslant 1$, then $\mathsf{F}_n(a) > \mathsf{F}_m(a)$.*
4. $\mathsf{F}_m(a + b) \geqslant \mathsf{F}_m(a) + b$ *and* $\mathsf{F}_m(a + 1) \geqslant 2 \cdot \mathsf{F}_m(a)$.
5. *Each function F_n is primitive recursive.*
6. *For every n-ary primitive recursive function f, there exists a number k such that for all $m_1, \dots, m_n \in \mathbb{N}$, we have $\mathsf{F}_k(\max\{m_1, \dots, m_n\}) \geqslant f(m_1, \dots, m_n)$.*

Lemma 6.59 ([47, Lemma 2.19]). *Let $(\mathcal{A}, >_{\mathbb{N}})$ be a well-founded monotone algebra with a subset of $\mathbb{N}$ as carrier such that $>_{\mathbb{N}}$ is the usual strict order on $\mathbb{N}$. Let $\mathcal{R}$ be a TRS which is compatible with the resulting reduction order $>_{\mathbb{N}\mathcal{A}}$, and let p be a strictly monotone unary function on $\mathbb{N}$ such that for all $f \in \mathcal{F}$, we have*

$$p(n) \geqslant f_{\mathcal{A}}(n, \dots, n) \qquad \textit{for all } n \in \mathbb{N} \ .$$

Then for all $n \in \mathbb{N}$, we have $\mathsf{dc}_{\mathcal{R}}(n) \leqslant p^n(0)$.

We start the construction of the well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$, whose carrier we define to be $\mathbb{N} \setminus \{0\}$, by giving the interpretation function of the function symbol g, which defines the reduction pair function g on the term rewriting level. By definition $\mathsf{g} \in \mathcal{F}_{\mathsf{DG\,sim}}$, and we assume that g is primitive recursive. The rules $\mathcal{R}'_{\mathsf{DG\,sim}}$ defining g constitute a (terminating) subset of $\mathcal{R}_{\mathsf{DG\,sim}}$, see Definition 6.43. A complication in the definition of $\mathsf{g}_{\mathcal{A}}$ is that the TRS $\mathcal{R}'_{\mathsf{DG\,sim}}$ has only been defined implicitly above. However, following the construction in [48], we conclude the existence of a well-founded monotone algebra $(\mathcal{A}', >_{\mathbb{N}})$ such that $\mathcal{R}'_{\mathsf{DG\,sim}}$ is compatible with $>_{\mathbb{N}\mathcal{A}'}$, and $\mathsf{g}_{\mathcal{A}'}$ is primitive recursive. More specifically, we can assume that there exists some $\ell \in \mathbb{N}$ such that $\mathsf{g}_{\mathcal{A}'}(n) = \mathsf{F}_{\ell}(n)$, and that $\mathsf{s}_{\mathcal{A}'}(n) = n + 1$ and $\mathsf{0}_{\mathcal{A}'} = 1$.

Preparing the definition of the well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$, we define the interpretation functions $\mathsf{g}_{\mathcal{A}}$, $\mathsf{s}_{\mathcal{A}}$, and $\mathsf{0}_{\mathcal{A}}$ as follows:

$$\mathsf{g}_{\mathcal{A}}(n) = \mathsf{F}_{\ell}(n) \qquad \mathsf{s}_{\mathcal{A}}(n) = n + 1 \qquad \mathsf{0}_{\mathcal{A}} = 1$$

Next, we give the mappings associated with the function symbols f_i $(0 \leqslant i \leqslant k)$. Let $d \geqslant \max\{C + 2, 3 \cdot A + 2\}$.

$$\mathsf{f}_{i\mathcal{A}}(n, x_1, \ldots, x_A) = \mathsf{F}_{\ell+2i}^{d^{n+1} \cdot (C+1)}(n + x_1 + \ldots + x_A)$$

Before we continue the definition of $(\mathcal{A}, >_{\mathbb{N}})$ we give two auxiliary results. Let α be an arbitrary assignment. For any variable x, let $\overline{x}$ abbreviate $[\alpha]_{\mathcal{A}}(x)$.

Lemma 6.60. *For every $i \in \mathbb{N}$, there exists some $q \in \mathbb{N}$ such that for each assignment α, we have $\mathsf{F}_q(\overline{n} + \overline{x}_1 + \ldots + \overline{x}_A) \geqslant [\alpha]_{\mathcal{A}}(\mathsf{f}_i(n, x_1, \ldots, x_A))$.*

Proof. By definition $[\alpha]_{\mathcal{A}}(\mathsf{f}_i(n, x_1, \ldots, x_A)) = \mathsf{F}_{\ell+2i}^{d^{\overline{n}+1} \cdot (C+1)}(\overline{n} + \overline{x}_1 + \ldots + \overline{x}_A)$. We set $p = \ell + 2i$ and abbreviate $\overline{x}_1 + \ldots + \overline{x}_A$ as $\overline{x}$. Recall that the carrier of $\mathcal{A}$ is defined to be $\mathbb{N} \setminus \{0\}$, so for every variable x, we have $\alpha(x) \geqslant 1$. Due to Lemma 6.58.1 and 6.58.3, and the assumption on d, we have $\mathsf{F}_1(\overline{n} + \overline{x}) \geqslant d^{\overline{n}+2+\overline{x}} \geqslant d^{\overline{n}+2} + \overline{x} > d^{\overline{n}+1} \cdot (C + 1) + \overline{x}$ for $\overline{n} \geqslant 1$. We obtain:

$$\begin{aligned}
\mathsf{F}_{p+2}(\overline{n} + \overline{x}) &\geqslant \mathsf{F}_{p+1} \circ \mathsf{F}_{p+1}(\overline{n} + \overline{x}) \\
&\geqslant \mathsf{F}_{p+1} \circ \mathsf{F}_1(\overline{n} + \overline{x}) \\
&> \mathsf{F}_{p+1}(d^{\overline{n}+1} \cdot (C + 1) + \overline{x}) \\
&\geqslant \mathsf{F}_p^{d^{\overline{n}+1} \cdot (C+1)+1}(d^{\overline{n}+1} \cdot (C + 1) + \overline{x}) \\
&> \mathsf{F}_p^{d^{\overline{n}+1} \cdot (C+1)}(\overline{n} + \overline{x}) \ .
\end{aligned}$$

Hence the lemma follows by setting $q = p + 2$. □

Lemma 6.61. *For every assignment α, every $0 \leqslant i \leqslant k$ and every $0 \leqslant j \leqslant C$ we have*

$$\mathsf{F}_{\ell+2i}^{d^{\overline{n}+2} \cdot (j+1)}(\overline{n} + \overline{x}_1 + \ldots + \overline{x}_A) \geqslant [\alpha]_{\mathcal{A}}(M_i^j(n, x_1, \ldots, x_A)) \ .$$

Proof. We fix i and proceed by induction on j. Set $p = \ell + 2i$. Suppose $j = 0$, then we have $M_i^j(n, x_1, \ldots, x_A) = \mathsf{f}_i(n, x_1, \ldots, x_A)$. We obtain

$$\begin{aligned}\mathsf{F}_p^{d^{\overline{n}+2}}(\overline{n} + \overline{x}_1 + \ldots + \overline{x}_A) &> \mathsf{F}_p^{d^{\overline{n}+1}\cdot(C+1)}(\overline{n} + \overline{x}_1 + \ldots + \overline{x}_A) \\ &= [\alpha]_{\mathcal{A}}(\mathsf{f}_i(n, x_1, \ldots, x_A)) ,\end{aligned}$$

where we use Lemma 6.58.1 together with the fact $d > C + 1$.

Now we consider the case $j > 0$. We abbreviate $\overline{x}_1 + \ldots + \overline{x}_A$ as $\overline{x}$.

$$\begin{aligned}&\mathsf{F}_p^{d^{\overline{n}+2}\cdot(j+1)}(\overline{n} + \overline{x}) \\ &\quad \geqslant \mathsf{F}_p^{d^{\overline{n}+2}\cdot j + d^{\overline{n}+2} - d^{\overline{n}+1} + 1}(\overline{n} + \overline{x}) \\ &\quad = \mathsf{F}_p^{d^{\overline{n}+2}\cdot j + d^{\overline{n}+1}\cdot(d-1) + 1}(\overline{n} + \overline{x}) \\ &\quad \geqslant \mathsf{F}_p^{d^{\overline{n}+2}\cdot j + d^{\overline{n}+1}\cdot(C+1) + 1}(\overline{n} + \overline{x}) \\ &\quad = \mathsf{F}_p^{d^{\overline{n}+1}\cdot(C+1)} \circ \mathsf{F}_p \circ \mathsf{F}_p^{d^{\overline{n}+2}\cdot j}(\overline{n} + \overline{x}) \\ &\quad \geqslant \mathsf{F}_p^{d^{\overline{n}+1}\cdot(C+1)}(\overline{n} + A \cdot \mathsf{F}_p^{d^{\overline{n}+2}\cdot j}(\overline{n} + \overline{x})) \\ &\quad \geqslant \mathsf{f}_{i\mathcal{A}}(\overline{n}, [\alpha]_{\mathcal{A}}(M_i^{j-1}(n, x_1, \ldots, x_A)), \ldots, [\alpha]_{\mathcal{A}}(M_i^{j-1}(n, x_1, \ldots, x_A))) \\ &\quad = [\alpha]_{\mathcal{A}}(M_i^j(n, x_1, \ldots, x_A))\end{aligned}$$

Here we frequently use Lemma 6.58.1 and 6.58.2 together with the assumptions $d \geqslant C + 2$ and $d \geqslant 3 \cdot A + 2$. The induction hypothesis is applied A times in the seventh line. □

We complete the definition of $(\mathcal{A}, >_{\mathbb{N}})$ as follows:

$$\mathsf{c}_{\mathcal{A}} = 3 \qquad \mathsf{size}_{\mathcal{A}}(n) = n + 2 \qquad \times_{A\mathcal{A}}(n) = (A + 1) \cdot n + 1$$

$$\mathsf{h}_{\mathcal{A}}(n) = \mathsf{F}_{\ell+2k}^{d^{\mathsf{F}_\ell(\mathsf{F}_\ell^{d^2\cdot(C+1)}(1+A\cdot n)+2)+2}\cdot(C+1)}(\mathsf{F}_\ell(\mathsf{F}_\ell^{d^2\cdot(C+1)}(1 + A \cdot n) + 2) + 1 + A \cdot n)$$

$$\mathsf{z}_{\mathcal{A}} = \mathsf{F}_{\ell+2k}^{d^{\mathsf{F}_\ell(\mathsf{F}_\ell^{d^2\cdot(C+1)}(1+3\cdot A)+2)+2}\cdot(C+1)}(\mathsf{F}_\ell(\mathsf{F}_\ell^{d^2\cdot(C+1)}(1 + 3 \cdot A) + 2) + 1 + 3 \cdot A)$$

Now we are ready to show the main lemma of this subsection.

Lemma 6.62. *Let $(\mathcal{A}, >_{\mathbb{N}})$ be the well-founded monotone algebra defined over the course of this subsection. Then $\mathcal{R}_{\mathsf{DG\,sim}}$ is compatible with $>_{\mathbb{N}\mathcal{A}}$.*

Proof. As mentioned above, we assume that the reduction pair function g is primitive recursive. Otherwise, a straightforward extension of Definition 6.57 suffices to prove the more general lemma.

By construction, all rules from $\mathcal{R}'_{\mathsf{DG\,sim}}$ are strictly oriented by the reduction order $>_{\mathbb{N}\mathcal{A}}$ based on $(\mathcal{A}, >_{\mathbb{N}})$. Moreover, it is straightforward to check that each of the rules $3_{i,j}$–10 is strictly oriented by $>_{\mathbb{N}\mathcal{A}}$, so we restrict our attention to the families of rules 1_i and 2_i. We abbreviate $\overline{x}_1 + \ldots + \overline{x}_A$ as $\overline{x}$.

We start by considering the rule 1_i for some arbitrary, but fixed i. Using Lemmata 6.58.2 and 6.61, we obtain for any assignment α:

$$\begin{aligned}[\alpha]_{\mathcal{A}}(\mathsf{f}_i(\mathsf{s}(n), x_1, \ldots, x_A)) &= \mathsf{F}_{\ell+2i}^{d^{\overline{n}+2}\cdot(C+1)}(\overline{n} + 1 + \overline{x}) \\ &> \mathsf{F}_{\ell+2i}^{d^{\overline{n}+2}\cdot(C+1)}(\overline{n} + \overline{x}) \\ &\geqslant [\alpha]_{\mathcal{A}}(M_i^C(n, x_1, \ldots, x_A))\end{aligned}$$

Now we consider the rule 2_i for some arbitrary, but fixed i. We set $p = \ell + 2i$, hence $p + 2 = \ell + 2(i + 1)$. Recall that we previously assumed $d \geqslant C + 2 \geqslant 2$. For any assignment α, we obtain:

$$\begin{aligned}
&[\alpha]_{\mathcal{A}}(\mathsf{f}_{i+1}(n, x_1, \ldots, x_A)) \\
&= \mathsf{F}_{p+2}^{d^{\overline{n}+1}\cdot(C+1)}(\overline{n} + \overline{x}) \\
&\geqslant \mathsf{F}_{p+2} \circ \mathsf{F}_{p+2} \circ \mathsf{F}_{p+2}^{d\cdot(C+1)}(\overline{n} + \overline{x}) \\
&> \mathsf{F}_{p+2} \circ \mathsf{F}_{p+2} \circ \mathsf{F}_{p}^{d^{\overline{n}+2}\cdot(C+1)}(\overline{n} + \overline{x}) \\
&\geqslant \mathsf{F}_{p+2} \circ \mathsf{F}_{\ell}^{2} \circ \mathsf{F}_{p}^{d^{\overline{n}+2}\cdot(C+1)}(1 + \overline{x}) \\
&\geqslant \mathsf{F}_{p}^{d^{(\mathsf{F}_\ell^2 \circ \mathsf{F}_p^{d^2\cdot(C+1)}(1+\overline{x}))+2}\cdot(C+1)} \circ \mathsf{F}_{\ell}^{2} \circ \mathsf{F}_{p}^{d^{\overline{n}+2}\cdot(C+1)}(1 + \overline{x}) \\
&\geqslant \mathsf{F}_{p}^{d^{\mathsf{F}_\ell(\mathsf{F}_\ell^{d^2\cdot(C+1)}(1+\overline{x})+2)+2}\cdot(C+1)}(\mathsf{F}_\ell(\mathsf{F}_\ell^{d^2\cdot(C+1)}(1 + \overline{x}) + 2) + 1 + \overline{x}) \\
&= [\alpha]_{\mathcal{A}}(\mathsf{f}_i(\mathsf{s}(\mathsf{g}(\mathsf{size}(\mathsf{f}_0(0, x_1, \ldots, x_A))), x_1, \ldots, x_A)))
\end{aligned}$$

In lines 4, 5, and 6 we apply slight variants of the proof of Lemma 6.60, and in line 7 we use Lemma 6.58.4. Moreover, we make frequent use of Lemma 6.58.1, 6.58.2, and 6.58.3 here. □

With this result, Lemma 6.44, which we initially wanted to show, easily follows:

Lemma. *The derivational complexity of $\mathcal{R}_{\mathsf{DG\,sim}}$ is bounded by a function which is primitive recursive in g.*

Proof. Let $(\mathcal{A}, >_{\mathbb{N}})$ be the algebra constructed over the course of this subsection. By Lemma 6.62, the TRS $\mathcal{R}_{\mathsf{DG\,sim}}$ is compatible with the resulting reduction order $>_{\mathbb{N}\mathcal{A}}$. Moreover, all interpretation functions of $\mathcal{A}$ are primitive recursive in g. Hence, by Lemma 6.59, $\mathsf{dc}_{\mathcal{R}_{\mathsf{DG\,sim}}}$ is primitive recursive in g, as well. □

6.6 The Basic Dependency Pair Method: a Lower Complexity Bound

Now we investigate an even more restricted version of the dependency pair framework. Naturally, we are able to prove even lower complexity bounds for TRSs whose termination can still be shown by these restricted means. Specifically, we consider the following termination theorem:

Theorem 6.63. *Let $\mathcal{R}$ be a TRS such that there exists a proof tree PT of $\mathcal{R}$. Suppose that every edge starting from a non-leaf node $(\mathcal{P}, \mathcal{R})$ of PT is labelled by a reduction pair processor $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$, and $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$ completely solves $(\mathcal{P}, \mathcal{R})$. Then $\mathcal{R}$ is terminating.*

Proof. Direct consequence of Theorem 6.3. □

Any proof tree PT considered by Theorem 6.63 has at most height 1. This is essentially the most basic version of the dependency pair method, as described in [2, Section 2.2]: in the notation of this thesis, only a single reduction pair processor, which completely solves the initial DP problem, is applied.

However, even though only a single reduction pair is used to show finiteness of the initial DP problem (and hence termination of the underlying TRS $\mathcal{R}$), the derivational complexity of $\mathcal{R}$ may still be well beyond the complexity induced by that reduction pair.

Example 6.64. Consider the TRS $\mathcal{R}_{\mathsf{dexp}}$ consisting of the following rules:

$$1\colon\ \mathsf{f}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{f}(\mathsf{f}(x))) \qquad 2\colon\ \mathsf{f}(x) \to \mathsf{cons}(x,x)$$

The derivational complexity of $\mathcal{R}_{\mathsf{dexp}}$ is at least double exponential. This can be seen as follows: let $C(x)$ be a shorthand for the term $\mathsf{cons}(x,x)$. Consider the family of terms $\mathsf{f}(\mathsf{s}^n(\mathsf{0}))$ parametrised by $n \in \mathbb{N}$. As can be easily seen, the n^{th} term of this family rewrites to $\mathsf{s}^n(\mathsf{f}^{2^n}(\mathsf{0}))$ in $2^n - 1$ steps using rule 1. Now we can use rule 2, reducing outermost redexes first, to reach $\mathsf{s}^n(C^{2^n}(\mathsf{0}))$ in $2^{2^n} - 1$ steps. Thus $\mathsf{dc}_{\mathcal{R}_{\mathsf{dexp}}}$ is at least double exponential.

On the other hand consider the following termination proof of $\mathcal{R}_{\mathsf{dexp}}$ by Theorem 6.63: the set of dependency pairs of $\mathcal{R}_{\mathsf{dexp}}$ consists of the two rules

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{f}(x)) \qquad \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(x)\ .$$

Let $\mathcal{A}$ be the strongly linear interpretation defined by

$$\mathsf{f}^\sharp_{\mathcal{A}}(m) = \mathsf{f}_{\mathcal{A}}(m) = m \qquad \mathsf{s}_{\mathcal{A}}(m) = m+1 \qquad \mathsf{cons}_{\mathcal{A}}(m,n) = \mathsf{0}_{\mathcal{A}} = 0\ .$$

Then the resulting pair processor $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}},>_{\mathcal{A}})}$ completely solves the initial DP problem $(\mathsf{DP}(\mathcal{R}_{\mathsf{dexp}}), \mathcal{R}_{\mathsf{dexp}})$. This results in a proof tree PT which uses only $\Phi^{\mathsf{RP}}_{(\geqslant_{\mathcal{A}},>_{\mathcal{A}})}$. Hence, $\mathcal{R}_{\mathsf{dexp}}$ is terminating by Theorem 6.63. Moreover, the identity function $g(n) = n$ is a reduction pair function of PT.

Note the contrast between $\mathsf{dc}_{\mathcal{R}_{\mathsf{dexp}}}$ and the complexity induced by the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$: the derivational complexity of $\mathcal{R}_{\mathsf{dexp}}$ is at least double exponential. On the other hand, by Theorem 4.13, the reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces linear complexity.

As shown in the next example, even for SRSs this difference can be at least exponential.

Example 6.65. Consider the SRS $\mathcal{R}_{\mathsf{exp}}$, which is a restriction of $\mathcal{R}_{\mathsf{dexp}}$ consisting of the single rewrite rule

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{f}(\mathsf{f}(x)))\ .$$

Consider the family of terms $\mathsf{f}(\mathsf{s}^n(\mathsf{0}))$. As can be easily seen, the n^{th} term of this family rewrites to $\mathsf{s}^n(\mathsf{f}^{2^n}(\mathsf{0}))$ in $2^n - 1$ steps. Thus $\mathsf{dc}_{\mathcal{R}_{\mathsf{exp}}}$ is at least exponential.

On the other hand, termination of $\mathcal{R}_{\mathsf{exp}}$ can be shown by Theorem 6.63, using essentially the same proof tree as the termination proof of $\mathcal{R}_{\mathsf{dexp}}$ in Example 6.64 above. Again, the identity function $g(n) = n$ is a reduction pair function of this proof tree.

6.7 The Basic Dependency Pair Method: an Upper Complexity Bound

In Example 6.64, we exhibited a TRS which terminates by Theorem 6.63, but whose derivational complexity grows at least double exponentially faster than the complexity induced by the employed reduction pair. In this section, we show that this is the worst possible behaviour for this class of TRSs. More precisely, we prove the following theorem:

Theorem 6.66. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.63 using a proof tree* PT*, and let g be a reduction pair function of* PT*. Then* $\mathsf{dc}_{\mathcal{R}}$ *is bounded double exponentially in g. More specifically, we have*

$$\mathsf{dc}_{\mathcal{R}}(n) \leqslant (A+1)^{(2\cdot C+2)^{g(n)+2}\cdot n+n} ,$$

where A is the maximum arity of any function symbol in the signature of $\mathcal{R}$, but at least 1*, and* $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

In order to show that this theorem holds, we follow the ideas we used to prove Theorems 6.9 and 6.40 in Sections 6.3 and 6.5 once again: based on the mapping norm and a suitable interpretation $\mathsf{tr}_{\mathsf{DP}}$, we define a TRS $\mathcal{R}_{\mathsf{DP\,sim}}$ simulating the initial TRS $\mathcal{R}$. Finally, we can simply prove an upper bound on the derivational complexity of $\mathcal{R}_{\mathsf{DP\,sim}}$ in order to obtain an upper bound on $\mathsf{dc}_{\mathcal{R}}$.

However, the complexity bound we want to show in this section is considerably smaller than in both of the Sections 6.3 and 6.5. Therefore, simple syntactic means such as LPOs or GRLPOs (compare Section 6.5.1) are not useful for proving a good upper bound on the complexity of $\mathcal{R}_{\mathsf{DP\,sim}}$. Instead, similar to Section 6.5.2, we construct a well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$ with a subset of $\mathbb{N}$ as carrier such that $>_{\mathbb{N}}$ is the usual order on $\mathbb{N}$ and $\mathcal{R}_{\mathsf{DP\,sim}}$ is compatible with the resulting reduction order $>_{\mathbb{N}\mathcal{A}}$.

Notation 6.67. For the remainder of this section, we fix a TRS $\mathcal{R}$ such that termination of $\mathcal{R}$ follows by Theorem 6.63 using some proof tree PT.

Let g be a reduction pair function of PT, A the maximum arity of any function symbol occurring in $\mathcal{R}$, but at least 1, and $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

The simple structure of PT extends to the range of the norm function. For any term t, the norm of t has the shape (u_1, u_2) with $u_1 \in \mathbb{N} \cup \{\bot\}$, and $u_2 \in \{0, \bot\}$. Moreover, whenever $\mathsf{norm}(s) \sqsupset \mathsf{norm}(t)$ for some terms s and t, we also have $\mathsf{norm}_1(s) \gg \mathsf{norm}_1(t)$, because $\mathsf{norm}_1(u) = \bot$ if and only if $\mathsf{norm}_2(u) = \bot$ for all terms u. In order to write down the rules of $\mathcal{R}_{\mathsf{DP\,sim}}$ concisely, we again use some abbreviations.

Notation 6.68. We make use of the following abbreviation M^j (for $j \in \mathbb{N}$):

$$\begin{aligned} M^0(u, x_1, \ldots, x_A) &= \mathsf{f}(u, x_1, \ldots, x_A) \\ M^{j+1}(u, x_1, \ldots, x_A) &= \mathsf{f}(u, M^j(u, x_1, \ldots, x_A), \ldots, M^j(u, x_1, \ldots, x_A)) \end{aligned}$$

We write $\bar{t}$ to denote $t, \ldots, t$, where the number of repetitions of t follows from the context.

Definition 6.69. We define the following (schematic) TRS $\mathcal{R}_{\mathsf{DP\,sim}}$, where $1 \leqslant j \leqslant A$.

$$\begin{aligned}
1\colon &\qquad \mathsf{f}(\mathsf{s}(u), x_1, \dots, x_A) \to M^C(u, x_1, \dots, x_A)\\
2\colon &\qquad \mathsf{f}(u, x_1, \dots, x_A) \to \mathsf{c}\\
3_j\colon &\qquad \mathsf{f}(u, x_1, \dots, x_A) \to x_j
\end{aligned}$$

The rules of $\mathcal{R}_{\mathsf{DP\,sim}}$ are essentially a subset of the rules of $\mathcal{R}_{\mathsf{DG\,sim}}$ given in Definition 6.43. The labels i from the function symbols f_i have been stripped.

In contrast to Lemmata 6.28 and 6.44, we are not able to show anymore that the derivational complexity of $\mathcal{R}_{\mathsf{DP\,sim}}$ is low enough in order imply Theorem 6.66. Due to rule 1, the interpretation function of f in the constructed well-founded monotone algebra must grow at least exponentially in its first argument. Therefore, there exist families of terms whose interpretations can not be bounded elementarily in their sizes. However, this problem does not occur for the terms we are really interested in (the range of the translation function $\mathsf{tr}_{\mathsf{DP}}$): for any of these terms, the first argument of any f symbol has the shape $\mathsf{s}^n(\mathsf{0})$ for some $n \in \mathbb{N}$. Thus we just prove an upper bound on the derivation heights of terms in the range of $\mathsf{tr}_{\mathsf{DP}}$.

Notation 6.70. For the remainder of this section, let $\mathcal{F}_{\mathsf{DP\,sim}}$ denote the signature of $\mathcal{R}_{\mathsf{DP\,sim}}$.

Definition 6.71. The mapping $\mathsf{tr}_{\mathsf{DP}}\colon \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}_{\mathsf{DP\,sim}}, \mathcal{V}))$ is defined by the following equation:

$$\mathsf{tr}_{\mathsf{DP}}(t) = \begin{cases} t & \text{if } t \in \mathcal{V}\\ \mathsf{f}(\mathsf{norm}_1(t)^*, \mathsf{tr}_{\mathsf{DP}}(t_1), \dots, \mathsf{tr}_{\mathsf{DP}}(t_n), \bar{\mathsf{c}}) & \text{if } t = f(t_1, \dots, t_n)\end{cases}$$

Here, the operator $(\cdot)^*$ is defined as follows:

$$u^* = \begin{cases} \mathsf{0} & \text{if } u = \bot\\ \mathsf{s}^{u+1}(\mathsf{0}) & \text{if } u \in \mathbb{N}\end{cases}$$

We now construct a well-founded monotone algebra $(\mathcal{A}, >_{\mathbb{N}})$. We define the carrier of $\mathcal{A}$ to be $\mathbb{N} \setminus \{0\}$, and $>_{\mathbb{N}}$ to be the usual strict order on $\mathbb{N}$. Finally, the interpretation functions of $\mathcal{A}$ are

$$\begin{gathered}
\mathsf{f}_{\mathcal{A}}(u, x_1, \dots, x_A) = (A+1)^{(2\cdot C+2)^u} \cdot (x_1 + \ldots + x_A)\\
\mathsf{s}_{\mathcal{A}}(n) = n + 1 \qquad \mathsf{0}_{\mathcal{A}} = 1 \qquad \mathsf{c}_{\mathcal{A}} = 1\ .
\end{gathered}$$

Given an assignment α, we use $\overline{x}$ to abbreviate $[\alpha]_{\mathcal{A}}(x)$ for any variable x.

Lemma 6.72. *For all $j \in \mathbb{N}$ and assignments α, we have*

$$[\alpha]_{\mathcal{A}}(M^j(u, x_1, \dots, x_A)) \leqslant (A+1)^{(2\cdot C+2)^{\overline{u}}\cdot(2\cdot j+1)} \cdot (\overline{x}_1 + \ldots + \overline{x}_A)\ .$$

Proof. We show the lemma by induction on j.

Suppose $j = 0$. We have

$$\begin{aligned}[\alpha]_{\mathcal{A}}(M^0(u, x_1, \ldots, x_A)) &= [\alpha]_{\mathcal{A}}(\mathsf{f}(u, x_1, \ldots, x_A)) \\ &= (A+1)^{(2\cdot C+2)^{\overline{u}}} \cdot (\overline{x}_1 + \ldots + \overline{x}_A) \\ &= (A+1)^{(2\cdot C+2)^{\overline{u}}\cdot(2\cdot 0+1)} \cdot (\overline{x}_1 + \ldots + \overline{x}_A) \ ,\end{aligned}$$

hence the base case is concluded.

Now assume the inductive case that $j > 0$. Then we have

$$\begin{aligned}&[\alpha]_{\mathcal{A}}(M^j(u, x_1, \ldots, x_A)) \\ &\quad = [\alpha]_{\mathcal{A}}(\mathsf{f}(u, M^{j-1}(u, x_1, \ldots, x_A), \ldots, M^{j-1}(u, x_1, \ldots, x_A))) \\ &\quad = (A+1)^{(2\cdot C+2)^{\overline{u}}} \cdot A \cdot [\alpha]_{\mathcal{A}}(M^{j-1}(u, x_1, \ldots, x_A)) \\ &\quad \leqslant (A+1)^{(2\cdot C+2)^{\overline{u}}} \cdot A \cdot (A+1)^{(2\cdot C+2)^{\overline{u}}\cdot(2\cdot j-1)} \cdot (\overline{x}_1 + \ldots + \overline{x}_A) \\ &\quad < (A+1)^{(2\cdot C+2)^{\overline{u}}\cdot(2\cdot j+1)} \cdot (\overline{x}_1 + \ldots + \overline{x}_A) \ .\end{aligned}$$

Here we apply the induction hypothesis in line 4. □

Lemma 6.73. *For any term t and assignment α, we have*

$$[\alpha]_{\mathcal{A}}(\mathsf{tr}_{\mathsf{DP}}(t)) \leqslant (A+k)^{(2\cdot C+2)^{g(|t|)+2}\cdot|t|+|t|} \ ,$$

where $k = \max(\{1\} \cup \{\alpha(x) \mid x \in \mathcal{V}\mathsf{ar}(t)\})$.

Proof. We prove the lemma by induction on $|t|$. If $t \in \mathcal{V}$, then $|t| = 1$ and

$$[\alpha]_{\mathcal{A}}(\mathsf{tr}_{\mathsf{DP}}(t)) = \alpha(t) \leqslant k < (A+k)^{(2\cdot C+2)^{g(1)+2}+1} \ ,$$

so the lemma holds in this case.

Now suppose that t has the shape $f(t_1, \ldots, t_n)$. Then we have

$$\begin{aligned}[\alpha]_{\mathcal{A}}(\mathsf{tr}_{\mathsf{DP}}(t)) &= [\alpha]_{\mathcal{A}}(\mathsf{f}(\mathsf{norm}_1(t)^*, \mathsf{tr}_{\mathsf{DP}}(t_1), \ldots, \mathsf{tr}_{\mathsf{DP}}(t_n), \overline{\mathsf{c}})) \\ &\leqslant (A+1)^{(2\cdot C+2)^{g(|t|)+2}} \cdot (A - n + \sum_{i=1}^{n} (A+k)^{(2\cdot C+2)^{g(|t_i|)+2}\cdot|t_i|+|t_i|}) \\ &\leqslant (A+1)^{(2\cdot C+2)^{g(|t|)+2}+1} \cdot \sum_{i=1}^{n} (A+k)^{(2\cdot C+2)^{g(|t_i|)+2}\cdot|t_i|+|t_i|} \\ &\leqslant (A+k)^{(2\cdot C+2)^{g(|t|)+2}+1} \cdot \prod_{i=1}^{n} (A+k)^{(2\cdot C+2)^{g(|t|)+2}\cdot|t_i|+|t_i|} \\ &= (A+k)^{(2\cdot C+2)^{g(|t|)+2}+1} \cdot (A+k)^{(2\cdot C+2)^{g(|t|)+2}\cdot(|t|-1)+(|t|-1)} \\ &= (A+k)^{(2\cdot C+2)^{g(|t|)+2}\cdot|t|+|t|} \ .\end{aligned}$$

In the second line, we applied the induction hypothesis n times, and the fact that $g(t) \gg^{=} \mathsf{norm}_1(t)$, and hence $[\alpha]_{\mathcal{A}}(\mathsf{norm}_1(t)^*) \leqslant g(|t|) + 2$. □

Lemma 6.74. *For every term t, we have*

$$\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant (A+1)^{(2\cdot C+2)^{g(|t|)+2}\cdot|t|+|t|}$$

Proof. It is easy to see that rules 2 and 3_j of $\mathcal{R}_{\mathsf{DP\,sim}}$ are strictly oriented by $>_{\mathbb{N}\mathcal{A}}$. Lemma 6.72 makes it obvious that rule 1 is strictly oriented by $>_{\mathbb{N}\mathcal{A}}$, as well. Therefore, $\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant [\alpha]_{\mathcal{A}}(\mathsf{tr}_{\mathsf{DP}}(t))$ for all assignments α (in particular, for the assignment α_0 with $\alpha_0(x) = 1$ for all $x \in \mathcal{V}\mathsf{ar}(t)$), compare Lemma 4.8. By Lemma 6.73, we have $[\alpha_0]_{\mathcal{A}}(\mathsf{tr}_{\mathsf{DP}}(t)) \leqslant (A+1)^{(2\cdot C+2)^{g(|t|)+2}\cdot|t|+|t|}$, thus the lemma follows. □

Now that we have established an upper bound on the length of certain derivations in $\mathcal{R}_{\mathsf{DP\,sim}}$, we show that $\mathcal{R}_{\mathsf{DP\,sim}}$ indeed simulates $\mathcal{R}$, and that this simulation is established by the mapping $\mathsf{tr}_{\mathsf{DP}}$.

Lemma 6.75. *The following properties of $\mathcal{R}_{\mathsf{DP\,sim}}$ hold:*

1. *If $s = \mathsf{f}(u^*, s_1, \ldots, s_A)$, $t = \mathsf{tr}_{\mathsf{DP}}(t') = \mathsf{f}(v^*, s_1, \ldots, s_A)$, and $u \gg v$, then $s \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} t$.*
2. *For any terms $s = \mathsf{tr}_{\mathsf{DP}}(s')$ and $t = \mathsf{tr}_{\mathsf{DP}}(t')$, $s' \xrightarrow{\epsilon}_{\mathcal{R}} t'$ implies $s \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} t$.*
3. *If $a \to_{\mathcal{R}} b$ and $\mathsf{tr}_{\mathsf{DP}}(a) \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(b)$, then for any n-ary function symbol $f \in \mathcal{F}$, $\mathsf{tr}_{\mathsf{DP}}(f(t_1, \ldots, a, \ldots, t_n)) \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(f(t_1, \ldots, b, \ldots, t_n))$.*

Proof. We prove the three items successively.

1. In order to show Property 1, observe that $u \gg v$ implies $u^* = \mathsf{s}^a(\mathsf{0})$ and $v^* = \mathsf{s}^b(\mathsf{0})$ for some $a, b \in \mathbb{N}$ with $a > b$. Therefore, it suffices to show $\mathsf{f}(\mathsf{s}(u), x_1, \ldots, x_A) \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{f}(u, x_1, \ldots, x_A)$. This follows directly by applying rules 1 and 3_1 of $\mathcal{R}_{\mathsf{DP\,sim}}$.

2. We now show Property 2. Let $l \to r$ be the rewrite rule and σ the substitution applied in the step $s' \xrightarrow{\epsilon}_{\mathcal{R}} t'$. Let $m = \mathsf{norm}_1(s')$. Since l is not a variable, l has the shape $f(l_1, \ldots, l_n)$. Therefore, we have $\mathsf{tr}_{\mathsf{DP}}(l\sigma) = \mathsf{f}(\mathsf{s}^{m+1}(\mathsf{0}), \mathsf{tr}_{\mathsf{DP}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(l_n\sigma), \bar{\mathsf{c}})$. Since s' is a redex, it follows that $m \in \mathbb{N}$. By rules 1 and 3_1,

$$\mathsf{tr}_{\mathsf{DP}}(l\sigma) \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} M^{\mathsf{dp}(r)}(\mathsf{s}^m(\mathsf{0}), \mathsf{tr}_{\mathsf{DP}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(l_n\sigma), \bar{\mathsf{c}}) .$$

   We show the following claim by induction on $\mathsf{dp}(u)$.

   Claim 6.76. *We have $M^{\mathsf{dp}(u)}(\mathsf{s}^m(\mathsf{0}), \mathsf{tr}_{\mathsf{DP}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(l_n\sigma), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(u\sigma)$ whenever $u \trianglelefteq r$.*

   Since $r \trianglelefteq r$, showing the claim suffices to conclude Property 2 of the lemma. Hence, the remainder of this proof is devoted to showing the claim. We perform case distinction on u.

   Suppose $u \vartriangleleft l$. Then there exists some $j \in \{1, \ldots, n\}$ with $u \trianglelefteq l_j$. Using rule 3_1, we get

$$M^{\mathsf{dp}(u)}(\mathsf{s}^m(\mathsf{0}), \mathsf{tr}_{\mathsf{DP}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(l_n\sigma), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(l_j\sigma) .$$

Some potential further applications of rule $3_{j'}$ for some values of j' then yield $\mathsf{tr}_{\mathsf{DP}}(l_j\sigma) \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(u\sigma)$, which concludes the first case.

Now suppose that $u \ntrianglelefteq l$. Since variables occurring in u also occur in r and hence in l, the condition $u \ntrianglelefteq l$ implies that u is not a variable. Hence, $u = h(u_1, \ldots, u_{n'})$. Let $m' = \mathsf{norm}_1(u\sigma)$. By induction hypothesis, employing rule 3_1 $\sum_{j=1}^{n'} \mathsf{dp}(u) - 1 - \mathsf{dp}(u_j)$ many times, and rule 2 $A - n'$ times, we obtain

$$\begin{aligned} &M^{\mathsf{dp}(u)}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DP}}(l_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(l_n\sigma), \bar{\mathsf{c}}) \\ &\quad \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{f}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DP}}(u_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(u_{n'}\sigma), \bar{\mathsf{c}}) \ . \end{aligned}$$

By Lemma 6.24, we have $\mathsf{norm}(s') \sqsupset \mathsf{norm}(u\sigma)$, and therefore $m \gg m'$. Due to Property 1,

$$\mathsf{f}(\mathsf{s}^m(0), \mathsf{tr}_{\mathsf{DP}}(u_1\sigma), \ldots, \mathsf{tr}_{\mathsf{DP}}(u_{n'}\sigma), \bar{\mathsf{c}}) \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(u\sigma) \ ,$$

so the claim and thus Property 2 follow.

3. We now prove Property 3. Let $m = \mathsf{norm}_1(f(t_1, \ldots, a, \ldots, t_n))$ and $m' = \mathsf{norm}_1(f(t_1, \ldots, b, \ldots, t_n))$. By assumption,

$$\begin{aligned} &\mathsf{f}(m, \mathsf{tr}_{\mathsf{DP}}(t_1), \ldots, \mathsf{tr}_{\mathsf{DP}}(a), \ldots, \mathsf{tr}_{\mathsf{DP}}(t_n)) \\ &\quad \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{f}(m, \mathsf{tr}_{\mathsf{DP}}(t_1), \ldots, \mathsf{tr}_{\mathsf{DP}}(b), \ldots, \mathsf{tr}_{\mathsf{DP}}(t_n)) \ . \end{aligned}$$

By Lemma 6.22, we have $m \gg^= m'$. Hence, using Property 1, we obtain

$$\begin{aligned} &\mathsf{f}(m, \mathsf{tr}_{\mathsf{DP}}(t_1), \ldots, \mathsf{tr}_{\mathsf{DP}}(b), \ldots, \mathsf{tr}_{\mathsf{DP}}(t_n)) \\ &\quad \to^*_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{f}(m', \mathsf{tr}_{\mathsf{DP}}(t_1), \ldots, \mathsf{tr}_{\mathsf{DP}}(b), \ldots, \mathsf{tr}_{\mathsf{DP}}(t_n)) \ . \end{aligned}$$

Property 3 and thus the lemma follow.

□

Lemma 6.77. *For any terms s and t, $s \to_{\mathcal{R}} t$ implies $\mathsf{tr}_{\mathsf{DP}}(s) \to^+_{\mathcal{R}_{\mathsf{DP\,sim}}} \mathsf{tr}_{\mathsf{DP}}(t)$.*

Proof. Easy consequence of Lemma 6.75.2 and 6.75.3. □

Now we can prove the main result of this section, Theorem 6.66.

Theorem. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.63 using a proof tree* PT*, and let g be a reduction pair function of* PT*. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded double exponentially in g. More specifically, we have*

$$\mathsf{dc}_{\mathcal{R}}(n) \leqslant (A+1)^{(2\cdot C+2)^{g(n)+2}\cdot n+n} \ .$$

where A is the maximum arity of any function symbol in the signature of $\mathcal{R}$, but at least 1, and $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

Proof. Let $\mathcal{R}_{\mathsf{DP\,sim}}$ be the simulating TRS for $\mathcal{R}$, as defined over the course of this section, and $\mathsf{tr}_{\mathsf{DP}}$ the corresponding mapping on terms. By Lemma 6.74, we have

$$\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant (A+1)^{(2\cdot C+2)^{g(|t|)+2}\cdot|t|+|t|}$$

for all terms t. Moreover, due to Lemma 6.77, we have the inequality

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) .$$

Thus, the theorem follows. □

Note that with the assumptions of Theorem 6.66, a double exponential upper bound (as given by Theorem 6.66) is the lowest theoretically possible bound on $\mathsf{dc}_{\mathcal{R}}$. As demonstrated in Example 6.64, there exists such a proof tree of $\mathcal{R}_{\mathsf{dexp}}$ with a linear reduction pair function, but $\mathsf{dc}_{\mathcal{R}_{\mathsf{dexp}}}$ is at least double exponential.

Remark 6.78. In [80, Section 4] and [82, Section 5], only a triple exponential complexity bound was shown for TRSs whose termination can be proved by Theorem 6.63. It was left as an open problem whether this upper bound can be improved to a double exponential one. Theorem 6.66 solves this open problem in the positive.

Corollary 6.79. *For any terminating TRS $\mathcal{R}$ and term t, we have*

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant (A+1)^{(2\cdot C+2)^{\mathsf{dh}(t^\sharp, \overset{\epsilon}{\to}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})+2}\cdot|t|+|t|} ,$$

where A is the maximum arity of any function symbol in the signature of $\mathcal{R}$, but at least 1*, and* $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$*.*

Proof. Since $\mathcal{R}$ is assumed to be terminating, the pair of orders $(\to_{\mathcal{R}}^*, >)$, where $>$ is the transitive closure of $\overset{\epsilon}{\to}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}$, is a reduction pair. The reduction pair processor $\Phi^{\mathsf{RP}}_{(\to_{\mathcal{R}}^*,>)}$ completely solves the initial DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$, so termination of $\mathcal{R}$ follows by Theorem 6.63. Due to Lemma 6.77, we have $\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}})$. Inspection of the proofs of Lemmata 6.73 and 6.74 yields

$$\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant (A+1)^{(2\cdot C+2)^{[\alpha]_{\mathcal{A}}(\mathsf{norm}_1(t)^*)}\cdot|t|+|t|} ,$$

and clearly, $[\alpha_0]_{\mathcal{A}}(\mathsf{norm}_1(t)^*) \leqslant \mathsf{dh}(t^\sharp, \overset{\epsilon}{\to}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) + 2$, where α_0 is the assignment such that $\alpha_0(x) = 1$ for every variable $x \in \mathcal{V}\mathsf{ar}(t)$. Thus, the corollary follows. □

6.7.1 String Rewriting

If we restrict our attention to *SRSs* whose termination can be proved by Theorem 6.63, the upper bound in Theorem 6.66 can be improved significantly. The goal of this subsection is to show the following theorem stating this improved bound:

Theorem 6.80. *Let $\mathcal{R}$ be a SRS whose termination is shown via Theorem 6.63 using a proof tree* PT*, and let g be a reduction pair function of* PT*. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded exponentially in g. More specifically, we have*

$$\mathsf{dc}_{\mathcal{R}}(n) \leqslant (2 \cdot C + 2)^{g(n)+2} \cdot n + 1 \ ,$$

where $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

In order to show Theorem 6.80, we can reuse the lion's share of the proof of Theorem 6.66. Specifically, we can employ the simulating TRS $\mathcal{R}_{\mathsf{DP\,sim}}$, the mapping $\mathsf{tr}_{\mathsf{DP}}$, and the main result about the simulation of $\mathcal{R}$ by $\mathcal{R}_{\mathsf{DP\,sim}}$ (Lemma 6.77). However, due to the assumption that $\mathcal{R}$ is a SRS, the constant A is fixed to be one. Therefore, we construct a new well-founded monotone algebra $(\mathcal{B}, >_{\mathbb{N}})$, which allows us to replace Lemma 6.74 by a stronger result. We set the carrier of $\mathcal{B}$ to $\mathbb{N} \setminus \{0\}$, and define $>_{\mathbb{N}}$ to be the usual strict order on $\mathbb{N}$. The interpretation functions of $\mathcal{B}$ are the following:

$$\mathsf{f}_{\mathcal{B}}(u,x) = (2 \cdot C + 2)^u + x \qquad \mathsf{s}_{\mathcal{B}}(n) = n + 1 \qquad \mathsf{0}_{\mathcal{B}} = 1 \qquad \mathsf{c}_{\mathcal{B}} = 1$$

Given an assignment α, we use $\overline{x}$ to abbreviate $[\alpha]_{\mathcal{B}}(x)$ for any variable x.

Lemma 6.81. *For all $j \in \mathbb{N}$ and assignments α, we have*

$$[\alpha]_{\mathcal{B}}(M^j(u,x)) \leqslant (2 \cdot C + 2)^{\overline{u}} \cdot (2 \cdot j + 1) + \overline{x} \ .$$

Proof. We show the lemma by induction on j.

Suppose $j = 0$. We have

$$[\alpha]_{\mathcal{B}}(M^0(u,x)) = [\alpha]_{\mathcal{B}}(\mathsf{f}(u,x)) = (2 \cdot C + 2)^{\overline{u}} \cdot (2 \cdot 0 + 1) + \overline{x} \ ,$$

concluding the base case.

Now assume the inductive case that $j > 0$. Then we have

$$\begin{aligned} [\alpha]_{\mathcal{B}}(M^j(u,x)) &= [\alpha]_{\mathcal{B}}(\mathsf{f}(u, M^{j-1}(u,x))) \\ &= (2 \cdot C + 2)^{\overline{u}} + [\alpha]_{\mathcal{B}}(M^{j-1}(u,x)) \\ &\leqslant (2 \cdot C + 2)^{\overline{u}} + (2 \cdot C + 2)^{\overline{u}} \cdot (2 \cdot j - 1) + \overline{x} \\ &< (2 \cdot C + 2)^{\overline{u}} \cdot (2 \cdot j + 1) + \overline{x} \ . \end{aligned}$$

Here we applied the induction hypothesis in line 3. □

Lemma 6.82. *If $\mathcal{R}$ is a SRS, then for any term t and assignment α, we have*

$$[\alpha]_{\mathcal{B}}(\mathsf{tr}_{\mathsf{DP}}(t)) \leqslant (2 \cdot C + k + 1)^{g(|t|)+2} \cdot |t| + 1 \ ,$$

where $k = \max(\{1\} \cup \{\alpha(x) \mid x \in \mathcal{V}\mathsf{ar}(t)\})$.

Proof. We prove the lemma by induction on $|t|$. If $t \in \mathcal{V}$, then $|t| = 1$ and

$$[\alpha]_{\mathcal{B}}(\mathsf{tr}_{\mathsf{DP}}(t)) = \alpha(t) \leqslant k < (2 \cdot C + k + 1)^{g(1)+2} + 1 \ ,$$

so the lemma holds in this case.

If t is a constant function symbol, then

$$[\alpha]_{\mathcal{B}}(\mathsf{tr}_{\mathsf{DP}}(t)) = [\alpha]_{\mathcal{B}}(\mathsf{f}(\mathsf{norm}_1(t)^*, \mathsf{c})) \leqslant (2 \cdot C + 2)^{g(1)+2} + 1 \ ,$$

entailing the lemma again.

Finally, suppose that t has the shape $f(t_1)$. Then we have

$$\begin{aligned}[\alpha]_{\mathcal{B}}(\mathsf{tr}_{\mathsf{DP}}(t)) &= [\alpha]_{\mathcal{B}}(\mathsf{f}(\mathsf{norm}_1(t)^*, \mathsf{tr}_{\mathsf{DP}}(t_1))) \\ &\leqslant (2 \cdot C + k + 1)^{g(|t|)+2} + (2 \cdot C + k + 1)^{g(|t|-1)+2} \cdot (|t| - 1) + 1 \\ &\leqslant (2 \cdot C + k + 1)^{g(|t|)+2} \cdot |t| + 1 \ .\end{aligned}$$

In the second line, we applied the induction hypothesis and the fact that $g(t) \gg^{=} \mathsf{norm}_1(t)$, and hence $[\alpha]_{\mathcal{A}}(\mathsf{norm}_1(t)^*) \leqslant g(|t|) + 2$. □

Lemma 6.83. *If $\mathcal{R}$ is a SRS, then for every string t, we have*

$$\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant (2 \cdot C + 2)^{g(|t|)+2} \cdot |t| + 1$$

Proof. It is easy to see that rules 2 and 3_1 of $\mathcal{R}_{\mathsf{DP\,sim}}$ are strictly oriented by $>_{\mathbb{N}\mathcal{B}}$. Lemma 6.81 makes it obvious that rule 1 is strictly oriented by $>_{\mathbb{N}\mathcal{B}}$, as well. Therefore, $\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant [\alpha]_{\mathcal{B}}(\mathsf{tr}_{\mathsf{DP}}(t))$ for all assignments α (in particular, for the assignment α_0 with $\alpha_0(x) = 1$ for all $x \in \mathcal{V}\mathsf{ar}(t)$), compare Lemma 4.8. By Lemma 6.82, we have $[\alpha_0]_{\mathcal{A}}(\mathsf{tr}_{\mathsf{DP}}(t)) \leqslant (2 \cdot C + 2)^{g(|t|)+2} \cdot |t| + 1$, thus the lemma follows. □

This is sufficient for proving the main theorem of this subsection, Theorem 6.80.

Theorem. *Let $\mathcal{R}$ be a SRS whose termination is shown via Theorem 6.63 using a proof tree PT, and let g be a reduction pair function of PT. Then $\mathsf{dc}_{\mathcal{R}}$ is bounded exponentially in g. More specifically, we have*

$$\mathsf{dc}_{\mathcal{R}}(n) \leqslant (2 \cdot C + 2)^{g(n)+2} \cdot n + 1 \ ,$$

where $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

Proof. Let $\mathcal{R}_{\mathsf{DP\,sim}}$ be the simulating TRS for $\mathcal{R}$ and $\mathsf{tr}_{\mathsf{DP}}$ the corresponding mapping on terms. By Lemma 6.83, we have

$$\mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \leqslant (2 \cdot C + 2)^{g(|t|)+2} \cdot |t| + 1$$

for all terms t. Moreover, due to Lemma 6.77, we have the inequality

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant \mathsf{dh}(\mathsf{tr}_{\mathsf{DP}}(t), \to_{\mathcal{R}_{\mathsf{DP\,sim}}}) \ .$$

Thus, the theorem follows. □

Note that for SRSs whose termination is proved via Theorem 6.63, the complexity bound given by Theorem 6.80 cannot be improved to anything lower than exponential. As shown in Example 6.64, there exists a proof tree of $\mathcal{R}_{\mathsf{exp}}$ of the shape specified in Theorem 6.63 with a linear reduction pair function, but $\mathsf{dc}_{\mathcal{R}_{\mathsf{exp}}}$ is at least exponential.

Corollary 6.84. *For any terminating SRS $\mathcal{R}$ and term t, we have*

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant (2 \cdot C + 2)^{\mathsf{dh}(t^\sharp, \overset{\epsilon}{\to}_{\mathsf{DP}(\mathcal{R})} / \to_{\mathcal{R}}) + 2} \cdot n + 1 \ ,$$

where $C = \max\{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

Proof. Analogous to Corollary 6.79. □

6.8 Usable Rules: a Lower Complexity Bound

In the previous sections, we have analysed the derivational complexity of TRSs whose termination can be proved by a proof tree using reduction pair, dependency graph, and subterm criterion processors. A natural question that has been left open up to now is how the complexity bounds in Theorems 6.9, 6.40, and 6.66 change if usable rules processors are allowed in place of reduction pair processors in the assumptions of those theorems. We cannot transfer the proof ideas from the last sections to usable rules processors: given a usable rules processor $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ and a DP problem $(\mathcal{P}, \mathcal{R})$ with $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}((\mathcal{P}, \mathcal{R})) = \{(\mathcal{P}', \mathcal{R})\}$ and $\mathcal{P} \neq \mathcal{P}'$, it is easy to give an upper bound on $\mathsf{DPc}_{\mathcal{P} \setminus \mathcal{P}', \mathcal{P}' \cup \mathcal{U} \cup \mathcal{C}_\epsilon}$ for many instances of the reduction pair $(\succcurlyeq, \succ)$. Hence, it seems intuitive to base an extension of the functions norm_i in Definition 6.18 to usable rules processors on this measure. However, since $\mathcal{R}$ is often not a subset of $\mathcal{P}' \cup \mathcal{U} \cup \mathcal{C}_\epsilon$, such an extension of the definition of norm_i would destroy the proof of Lemma 6.22. In general, we can not even bound $\mathsf{DPc}_{\mathcal{P} \setminus \mathcal{P}', \mathcal{P}' \cup \mathcal{R}}$ in $\mathsf{DPc}_{\mathcal{P} \setminus \mathcal{P}', \mathcal{P}' \cup \mathcal{U} \cup \mathcal{C}_\epsilon}$:

Example 6.85. Let g be an arbitrary, but fixed computable function. Consider the DP problem $(\mathcal{P}, \mathcal{R})$, where $\mathcal{P}$ consists of the single rule

$$\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(x) \ ,$$

and $\mathcal{R}$ is a terminating TRS not containing the symbol $\mathsf{f}^\sharp$ such that for all $n \in \mathbb{N}$, we have $\mathsf{g}(\mathsf{s}^n(\mathsf{0})) \to^+_{\mathcal{R}} \mathsf{s}^{g(n)}(\mathsf{0})$ (by Turing-completeness of term rewriting, such a TRS $\mathcal{R}$ exists). Then $\mathsf{UR}_{\mathcal{R}}(\mathcal{P}) = \emptyset$. Let $\mathcal{A}$ be the strongly linear interpretation defined by the following interpretation functions:

$$\mathsf{f}^\sharp_{\mathcal{A}}(m) = m \qquad \mathsf{s}_{\mathcal{A}}(m) = m + 1 \qquad \mathsf{c}_{\mathcal{A}}(m, n) = m + n$$

Then the usable rules processor $\Phi^{\mathsf{UR}}_{\emptyset,(\geqslant_{\mathcal{A}},>_{\mathcal{A}})}$ completely solves the DP problem $(\mathcal{P}, \mathcal{R})$. Moreover, by Theorem 4.13, the applied reduction pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ induces linear complexity, hence $\mathsf{DPc}_{\mathcal{P},\mathcal{C}_\epsilon}$ is bounded by a linear function. On the other hand, $\mathsf{DPc}_{\mathcal{P},\mathcal{R}}$ grows as least as fast as g, as witnessed by the family of derivations $\mathsf{f}^\sharp(\mathsf{g}(\mathsf{s}^n(\mathsf{0}))) \to^+_{\mathcal{R}} \mathsf{f}^\sharp(\mathsf{s}^{g(n)}(\mathsf{0})) \to^{g(n)}_{\mathcal{P}} \mathsf{f}^\sharp(\mathsf{0})$ parametrised by $n \in \mathbb{N}$.

Based on this example, we can see that that it is generally impossible to bound $\mathsf{DPc}_{\mathcal{P} \setminus \mathcal{P}', \mathcal{P}' \cup \mathcal{R}}$ in $\mathsf{DPc}_{\mathcal{P} \setminus \mathcal{P}', \mathcal{P}' \cup \mathcal{U} \cup \mathcal{C}_\epsilon}$ by any computable function. However, if we only consider a restricted setting, such an analysis might become feasible again. In this spirit, we now consider TRSs whose termination can be shown by the following theorem, which is essentially Theorem 6.63, the most simple of the previously considered termination theorems, transferred to usable rules processors:

Theorem 6.86. *Let $\mathcal{R}$ be a TRS such that there exists a proof tree PT of $\mathcal{R}$. Suppose that every edge starting from a non-leaf node $(\mathcal{P},\mathcal{R})$ of PT is labelled by a usable rules processor $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$, and $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ completely solves $(\mathcal{P},\mathcal{R})$. Then $\mathcal{R}$ is terminating.*

Proof. Analogous to Theorem 6.3, using soundness of usable rules processors (cf. Theorem 2.65). □

The complexity theoretic strength of Theorem 6.86 compared to its "reduction pair processor version", Theorem 6.63, is still considerable, as witnessed by the next examples.

Example 6.87. Consider the TRS $\mathcal{R}_{\mathsf{ebin}}$ given by the following set of rules:

$$\begin{aligned} \mathsf{d}(\mathsf{0}) &\to \mathsf{0} & \mathsf{e}(\mathsf{0},x) &\to x \\ \mathsf{d}(\mathsf{s}(x)) &\to \mathsf{s}(\mathsf{s}(\mathsf{d}(x))) & \mathsf{e}(\mathsf{s}(x),y) &\to \mathsf{e}(x,\mathsf{d}(y)) \end{aligned}$$

The dependency pairs of $\mathcal{R}_{\mathsf{ebin}}$ are the following rules:

$$\mathsf{d}^\sharp(\mathsf{s}(x)) \to \mathsf{d}^\sharp(x) \qquad \mathsf{e}^\sharp(\mathsf{s}(x),y) \to \mathsf{e}^\sharp(x,\mathsf{d}(y)) \qquad \mathsf{e}^\sharp(\mathsf{s}(x),y) \to \mathsf{d}^\sharp(y)$$

The following two rules are contained in $\mathsf{UR}_{\mathcal{R}_{\mathsf{ebin}}}(\mathsf{DP}(\mathcal{R}_{\mathsf{ebin}}))$:

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

In order to give an upper complexity bound, we now construct a weakly monotone algebra $(\mathcal{A},\geqslant,>)$ with carrier $\mathbb{N}$, where $\geqslant$ and $>$ are the usual weak and strict orders on $\mathbb{N}$, respectively. The interpretation functions of $\mathcal{A}$ are

$$\begin{aligned} \mathsf{e}^\sharp_{\mathcal{A}}(m,n) &= 2^m\cdot(n+1)+1 & \mathsf{d}^\sharp_{\mathcal{A}}(m) &= m & \mathsf{d}_{\mathcal{A}}(m) &= 2\cdot m \\ \mathsf{0}_{\mathcal{A}} &= 0 & \mathsf{s}_{\mathcal{A}}(m) &= m+1 & \mathsf{c}_{\mathcal{A}}(m,n) &= m+n\ . \end{aligned}$$

It is easy to check that $\mathsf{UR}_{\mathcal{R}_{\mathsf{ebin}}}(\mathsf{DP}(\mathcal{R}_{\mathsf{ebin}}))\cup\mathcal{C}_\epsilon$ is compatible with $\geqslant_{\mathcal{A}}$, and that $\mathsf{DP}(\mathcal{R}_{\mathsf{ebin}})$ is compatible with $>_{\mathcal{A}}$. Moreover, for any term $t\in\mathcal{T}(\mathcal{F}\cup\{\mathsf{c}\})$, the value $[\alpha]_{\mathcal{A}}(t^\sharp)$ is double exponentially bounded in $|t|$. Thus, we obtain a double exponential upper bound on $\mathsf{DPc}_{\mathsf{DP}(\mathcal{R}_{\mathsf{ebin}}),\mathsf{UR}_{\mathcal{R}_{\mathsf{ebin}}}(\mathsf{DP}(\mathcal{R}_{\mathsf{ebin}}))}$.

On the other hand both $\mathsf{DPc}_{\mathsf{DP}(\mathcal{R}_{\mathsf{ebin}}),\mathcal{R}_{\mathsf{ebin}}}$ and $\mathsf{dc}_{\mathcal{R}_{\mathsf{ebin}}}$ are clearly at least superexponential functions. The first is witnessed e.g. by the family of starting terms $\mathsf{e}^\sharp(E^n(\mathsf{0}),\mathsf{s}(\mathsf{0}))$, where $E(x)$ is a shorthand for $\mathsf{e}(x,\mathsf{s}(\mathsf{0}))$. The second is witnessed by the family of starting terms $E^n(\mathsf{0})$, for instance.

Remark 6.88. Note that in Example 6.87 it is essential that arbitrary starting terms, as for example $E^k(\mathsf{0})$, are considered by the definition of derivational complexity. If we restricted the starting terms to *basic* terms, then the results from Section 6.7 would directly extend to Theorem 6.86. This is a consequence of [44, Lemma 16].

We can generalise Example 6.87 from a double exponential function to primitive recursion by employing the Ackermann function.

Example 6.89. We employ a unary notation for the Ackermann function: we write $\mathsf{Ack}_i(x)$ instead of $\mathsf{Ack}(i, x)$. Consider the following family of schematic TRSs $\mathcal{R}_{\mathsf{aack}}(l)$, parametrised by $l \in \mathbb{N}$. Here we assume $0 \leqslant i < l$.

$$\begin{aligned} \mathsf{Ack}_0(x) &\to \mathsf{s}(x) & \mathsf{l}(\mathsf{0}, x) &\to \mathsf{Ack}_l(x) \\ \mathsf{Ack}_{i+1}(\mathsf{0}) &\to \mathsf{Ack}_i(\mathsf{s}(\mathsf{0})) & \mathsf{l}(\mathsf{s}(x), y) &\to \mathsf{l}(x, \mathsf{Ack}_l(y)) \\ \mathsf{Ack}_{i+1}(\mathsf{s}(x)) &\to \mathsf{Ack}_i(\mathsf{Ack}_{i+1}(x)) \end{aligned}$$

Note that $\mathsf{UR}_{\mathcal{R}_{\mathsf{aack}}(l)}(\mathsf{DP}(\mathcal{R}_{\mathsf{aack}}(l)))$ contains only the rules in the left column. We define a weakly monotone algebra $(\mathcal{A}, \geqslant, >)$ with carrier $\mathbb{N}$, where $\geqslant$ and $>$ are the usual weak and strict orders on $\mathbb{N}$. We set the interpretation functions of $\mathcal{A}$ to

$$\begin{aligned} (\mathsf{Ack}_i)_{\mathcal{A}}(m) &= \mathsf{Ack}_i(m) & \mathsf{c}_{\mathcal{A}}(m, n) &= m + n \\ (\mathsf{Ack}_i^\sharp)_{\mathcal{A}}(m) &= \mathsf{Ack}_i(m) + i & \mathsf{s}_{\mathcal{A}}(m) &= m + 1 \\ \mathsf{l}^\sharp_{\mathcal{A}}(m, n) &= \mathsf{Ack}_l^{m+1}(n) + m + l + 1 & \mathsf{0}_{\mathcal{A}} &= 0 \end{aligned}$$

It is easy to check that $\mathsf{UR}_{\mathcal{R}_{\mathsf{aack}}(l)}(\mathsf{DP}(\mathcal{R}_{\mathsf{aack}}(l))) \cup \mathcal{C}_\epsilon$ is compatible with $\geqslant_{\mathcal{A}}$, and that $\mathsf{DP}(\mathcal{R}_{\mathsf{aack}}(l))$ is compatible with $>_{\mathcal{A}}$. Moreover, for any term $t \in \mathcal{T}(\mathcal{F} \cup \{\mathsf{c}\})$, the value $[\alpha]_{\mathcal{A}}(t^\sharp)$ is bounded by $\mathsf{Ack}^2_{l+1}(\mathsf{O}(|t|))$. On the other hand the derivational complexity of $\mathcal{R}_{\mathsf{aack}}(l)$ is bounded from below by $\mathsf{Ack}_{l+2}(\Omega(n))$, as witnessed by derivations starting from the family of terms $F^k(\mathsf{0})$, where $F(x)$ is a shorthand for $\mathsf{l}(x, \mathsf{s}(\mathsf{0}))$.

As we can see from Example 6.89, the complexity induced by the reduction pair used in the termination proof of $\mathcal{R}_{\mathsf{aack}}(l)$ belongs to level $l + 1$ of the hierarchy of Ackermann functions with fixed first argument, while the derivational complexity of $\mathcal{R}_{\mathsf{aack}}(l)$ belongs at least to level $l + 2$ of that hierarchy.

6.9 Usable Rules: an Upper Complexity Bound

In this section, we investigate the upper bound on the derivational complexity of TRSs whose termination can be proved by Theorem 6.86, depending on the complexity induced by the reduction pair of the usable rules processor employed in the considered termination proof. Examples 6.87 and 6.89 suggest that this upper bound might be an iteration of the function obtained from Theorem 6.66. We show that this intuition is indeed correct. The remainder of this section is devoted to the proof of the following theorem:

Theorem 6.90. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.86 using a proof tree* PT*, and let $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ be the label of every edge starting from the root of* PT*. Moreover, let $g \colon \mathbb{N} \to \mathbb{N}$ be a strictly monotone function such that $(\succcurlyeq, \succ)$ induces complexity $\{g\}$. Then we have*

$$\mathsf{dc}_{\mathcal{R}}(n) \leqslant G^{n+1}(1)$$

for some function G which is elementary in g.

As argued in the previous section, in order to prove Theorem 6.90, it is not sufficient to simply extend the proof principle we used in Sections 6.3, 6.5, and 6.7 to show Theorems 6.9, 6.40, and 6.66, respectively. Rather, we do a close study of the correctness proof of Theorem 2.65 (compare [43, Section 3]).

The main ingredient of the correctness proof of Theorem 2.65 is the definition of the interpretation $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}$, cf. Definition 2.59. According to the proof of [43, Theorem 20], any $\mathsf{DP}(\mathcal{R}) \cup \mathcal{R}$-derivation starting from a term t can be transformed into a $\mathsf{DP}(\mathcal{R}) \cup \mathsf{UR}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R})) \cup \mathcal{C}_\epsilon$-derivation starting from $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)$. Therefore, estimating $|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)|$ is the key to the connection between $\mathsf{DPc}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}$ and $\mathsf{DPc}_{\mathsf{DP}(\mathcal{R}),\mathsf{UR}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R}))\cup\mathcal{C}_\epsilon}$. Suppose there exists a function f that bounds $\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathsf{UR}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R}))\cup\mathcal{C}_\epsilon})$ in $|t|$. Then $\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})$ can be bounded in $|t|$ by $f(|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)|)$.

However, the difficulty of this estimation lies in the following mutual dependence between the definition of the interpretation $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}$ and the derivation heights. On one hand, we bound $\mathsf{dh}(t, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})$ in $|t|$ by $f(|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)|)$. On the other hand, $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)$ depends on $\mathsf{dh}(t, \to_{\mathcal{R}})$ since $\mathsf{dh}(t, \to_{\mathcal{R}})$ determines the number of recursive calls of the shape $\mathit{order}(\{\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(u) \mid t \to_{\mathcal{R}} u\})$ in the definition of $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)$. In the remainder of this section we show how this mutual dependence can be resolved.

Notation 6.91. For the rest of this section, we fix a TRS $\mathcal{R}$ such that termination of $\mathcal{R}$ follows by Theorem 6.86 using some proof tree PT where the label of each edge starting from the root of PT is $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$.

Let $g\colon \mathbb{N} \to \mathbb{N}$ be a strictly monotone function (note that this implies $g(n) \geqslant n$) such that $(\succcurlyeq, \succ)$ induces complexity $\{g\}$, and let $E = \max\{A, B, 2 \cdot C\} + 3$, where A is the maximum arity of all function symbols, but at least 1, B is the number of rules in $\mathcal{R}$, and C is chosen such that it is larger than the size of any right-hand side of any rule in $\mathcal{R}$, and hence also larger than the depth of any right-hand side and the number of occurrences of any variable on any right-hand side.

Definition 6.92. Let $h\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the function satisfying the following recursive definition:

$$h(m,n) = \begin{cases} E \cdot (n+1) & \text{if } m = 0 \\ E \cdot h(m-1, 0) & \text{if } m > 0 \text{ and } n = 0 \\ E \cdot h(m-1, n) + E \cdot m \cdot h(E \cdot m, n-1) & \text{otherwise} \end{cases}$$

The next two lemmata estimate the size $|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)|$ of the interpretation $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)$ in the size of t and $\mathsf{dh}(t, \to_{\mathcal{R}})$.

Lemma 6.93. *The following properties of h hold:*

1. *The function h is well-defined and strictly monotone in each argument.*
2. *For all m, n, we have $E \leqslant h(m,n)$.*
3. *For any term t, we have $|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)| \leqslant h(|t|, \mathsf{dh}(t, \to_{\mathcal{R}}))$.*

Proof. Properties 1 and 2 are obvious, so we only show Property 3. We proceed by induction on the lexicographic order over the pair $(\mathsf{dh}(t, \rightarrow_{\mathcal{R}}), |t|)$. If t is a variable, then $|t| = 1$ and $\mathsf{dh}(t, \rightarrow_{\mathcal{R}}) = 0$. We have

$$|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)| = 1 \leqslant E^2 = h(1, 0) \ ,$$

so Property 3 follows in this case. If t has the shape $f(t_1, \ldots, t_n)$ with $t \in \mathsf{UT}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R}))$, then

$$\begin{aligned}
|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)| &= 1 + \sum_{i=1}^{n} |\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t_i)| \\
&\leqslant 1 + \sum_{i=1}^{n} h(|t_i|, \mathsf{dh}(t_i, \rightarrow_{\mathcal{R}})) \\
&\leqslant 1 + A \cdot h(|t| - 1, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) \\
&\leqslant E \cdot h(|t| - 1, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) \\
&\leqslant h(|t|, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) \ .
\end{aligned}$$

Property 3 follows. Here we applied the induction hypothesis n times in the second line. Now assume that t has the shape $f(t_1, \ldots, t_n)$, $t \notin \mathsf{UT}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R}))$, and $\mathsf{dh}(t, \rightarrow_{\mathcal{R}}) = 0$. Then we have

$$\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t) = \mathsf{c}(f(\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t_1), \ldots, \mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t_n)), \mathsf{nil}) \ ,$$

and hence, in analogy to the previous case,

$$\begin{aligned}
|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)| &= 3 + \sum_{i=1}^{n} |\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t_i)| \\
&\leqslant 3 + A \cdot h(|t| - 1, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) \\
&\leqslant h(|t|, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) \ .
\end{aligned}$$

Again, Property 3 follows. Finally, we consider the case where t has the shape $f(t_1, \ldots, t_n)$ with $t \notin \mathsf{UT}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R}))$ and $\mathsf{dh}(t, \rightarrow_{\mathcal{R}}) > 0$. We obtain

$$\begin{aligned}
&|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)| \\
&\quad = 2 + \sum_{i=1}^{n} |\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t_i)| + |\mathit{order}(\{\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(u) \mid t \rightarrow_{\mathcal{R}} u\})| \\
&\quad \leqslant 2 + \sum_{i=1}^{n} |\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t_i)| + 1 + B \cdot |t| \cdot (1 + \max\{|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(u)| \mid t \rightarrow_{\mathcal{R}} u\}) \\
&\quad \leqslant 3 + A \cdot h(|t| - 1, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) + B \cdot |t| \cdot (1 + h(C \cdot |t| + C, \mathsf{dh}(t, \rightarrow_{\mathcal{R}}) - 1)) \\
&\quad \leqslant E \cdot h(|t| - 1, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) + E \cdot |t| \cdot h(E \cdot |t|, \mathsf{dh}(t, \rightarrow_{\mathcal{R}}) - 1) \\
&\quad = h(|t|, \mathsf{dh}(t, \rightarrow_{\mathcal{R}})) \ .
\end{aligned}$$

In the third line we use the fact that any term t has at most $B \cdot |t|$ many reducts. In the fourth line, we apply the induction hypothesis and the fact that $|u| < C \cdot |t| + C$ whenever $t \rightarrow_{\mathcal{R}} u$. □

Lemma 6.94. *There exists some $d \in \mathbb{N}$ such that for all $m, n \in \mathbb{N}$, we have $h(m,n) \leqslant 2^{2^{d\cdot(m+n+1)}}$.*

Proof. First, we show by induction on the lexicographic order over (n, m) that $h(m,n) \leqslant (E\cdot(m+1))^{(m+1)\cdot E^{2\cdot n+1}}$. For $m = 0$, we have $h(0,n) = E \leqslant E^{E^{2\cdot n+1}}$. If $m > 0$ and $n = 0$, then

$$h(m,0) = E\cdot h(m-1,0) \leqslant E\cdot(E\cdot m)^{m\cdot E} \leqslant (E\cdot(m+1))^{(m+1)\cdot E} \ .$$

Here we applied the induction hypothesis in the first inequality. Finally, we consider the case that $m, n > 0$:

$$\begin{aligned} h(m,n) &= E\cdot h(m-1,n) + E\cdot n\cdot h(E\cdot m, n-1) \\ &\leqslant E\cdot(E\cdot m)^{m\cdot E^{2\cdot n+1}} + E\cdot m\cdot(E\cdot(E\cdot m+1))^{(E\cdot m+1)\cdot E^{2\cdot n-1}} \\ &\leqslant (E\cdot(m+1))^{1+m\cdot E^{2\cdot n+1}} + (E\cdot(m+1))^{1+2\cdot(m+1)\cdot E^{2\cdot n}} \\ &\leqslant \frac{1}{E}\cdot(E\cdot(m+1))^{2+m\cdot E^{2\cdot n+1}} + \frac{1}{E}\cdot(E\cdot(m+1))^{2+2\cdot(m+1)\cdot E^{2\cdot n}} \\ &\leqslant \frac{2}{E}\cdot(E\cdot(m+1))^{(m+1)\cdot E^{2\cdot n+1}} \\ &\leqslant (E\cdot(m+1))^{(m+1)\cdot E^{2\cdot n+1}} \end{aligned}$$

Here we applied the induction hypothesis twice in the second line.

It is easy to see that for suitable d we have $(E\cdot(n+1))^{(n+1)\cdot E^{2\cdot m+1}} \leqslant 2^{2^{d\cdot(m+n+1)}}$. Thus the lemma follows. □

Notation 6.95. For the remainder of this section, let d be the minimum number such that $h(m,n) \leqslant 2^{2^{d\cdot(m+n+1)}}$ for all $m, n \in \mathbb{N}$. We set $H(m,n) = 2^{2^{d\cdot(m+n+1)}}$. We define $G\colon \mathbb{N} \to \mathbb{N}$, based on the mapping g, as follows:

$$G(m) = g(1 + A\cdot H(m, (A+1)^{(2\cdot C+2)^{m+2}\cdot m+m}))$$

Lemma 6.96. *We have $\mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) \leqslant G^{|t|}(1)$.*

Proof. We show the lemma by induction on $t^\sharp$. If $t^\sharp = t$ is a variable, the lemma is trivial since $\mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) = 0$, but $G(1) > 0$. Now we assume the case where $t^\sharp$ has the shape $f^\sharp(t_1, \dots, t_n)$. Obviously, there exist no rule $l \to r \in \mathcal{R}$ and substitutions σ and τ such that $t^\sharp\sigma \to^*_{\mathcal{R}} l\tau$, hence $t^\sharp \in \mathsf{UT}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R}))$. Due to Definition 2.59 in conjunction with Lemma 6.93 there exists some $1 \leqslant i \leqslant n$ such that

$$|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t^\sharp)| \leqslant 1 + A\cdot h(|t_i|, \mathsf{dh}(t_i, \to_{\mathcal{R}})) \ .$$

By Corollary 6.79, we have

$$\mathsf{dh}(t_i, \to_{\mathcal{R}}) \leqslant (A+1)^{(2\cdot C+2)^{\mathsf{dh}(t_i^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})+2}\cdot|t_i|+|t_i|} \ .$$

Combining this with Lemma 6.94, we obtain

$$h(|t_i|, \mathsf{dh}(t_i, \to_{\mathcal{R}})) \leqslant H(|t_i|, (A+1)^{(2\cdot C+2)^{\mathsf{dh}(t_i^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})+2}\cdot|t_i|+|t_i|}) \ .$$

As mentioned above, any $\mathsf{DP}(\mathcal{R}) \cup \mathcal{R}$-derivation starting from $t^\sharp$ can be transformed into a $\mathsf{DP}(\mathcal{R}) \cup \mathsf{UR}_{\mathcal{R}}(\mathsf{DP}(\mathcal{R})) \cup \mathcal{C}_\epsilon$-derivation starting from $\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t)$. Thus, we also have $\mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) \leqslant g(|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t^\sharp)|)$. In sum we obtain:

$$
\begin{aligned}
\mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) &\leqslant g(|\mathcal{I}_{\mathsf{DP}(\mathcal{R}),\mathcal{R}}(t^\sharp)|) \\
&\leqslant g(1 + A \cdot h(|t_i|, \mathsf{dh}(t_i, \to_{\mathcal{R}}))) \\
&\leqslant g(1 + A \cdot H(|t_i|, (A+1)^{(2\cdot C+2)^{\mathsf{dh}(t_i^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})+2} \cdot |t_i| + |t_i|})) \\
&\leqslant G(\max\{|t_i|, \mathsf{dh}(t_i^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})\}) \ .
\end{aligned}
$$

It is easy to verify that $|t_i| \leqslant G^{|t_i|}(1) \leqslant G^{|t|-1}(1)$. Moreover, by induction hypothesis, we have $\mathsf{dh}(t_i^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) \leqslant G^{|t_i|}(1) \leqslant G^{|t|-1}(1)$. From this the lemma follows. □

Now we are ready to show the main theorem of this section, Theorem 6.90:

Theorem. *Let $\mathcal{R}$ be a TRS whose termination is shown via Theorem 6.63 using a proof tree* PT*, and let $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ be the label of every edge starting from the root of* PT*. Moreover, let $g\colon \mathbb{N} \to \mathbb{N}$ be a strictly monotone function such that $(\succcurlyeq, \succ)$ induces complexity $\{g\}$. Then we have*

$$\mathsf{dc}_{\mathcal{R}}(n) \leqslant G^{n+1}(1)$$

for some function G which is elementary in g.

Proof. Let G be the homonymous function defined throughout the course of this section. By Corollary 6.79 and the definition of G, for every term t, it follows that

$$\mathsf{dh}(t, \to_{\mathcal{R}}) \leqslant G(\max\{|t|, \mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}})\}) \ .$$

It is easy to verify that $|t| \leqslant G^{|t|}(1)$. Moreover, by Lemma 6.96, we have $\mathsf{dh}(t^\sharp, \xrightarrow{\epsilon}_{\mathsf{DP}(\mathcal{R})}/\to_{\mathcal{R}}) \leqslant G^{|t|}(1)$. Thus, the theorem follows. □

Given a reduction pair $(\succcurlyeq, \succ)$ such that Theorem 6.63 using the reduction pair processor $\Phi^{\mathsf{RP}}_{(\succcurlyeq,\succ)}$ can only prove termination of TRSs whose derivational complexity is bounded by some function g, Theorem 6.90 asserts that the derivational complexity whose termination can be proved by Theorem 6.86 using a usable rules processor of the shape $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ is bounded (roughly) by the iteration of g. Moreover, Examples 6.87 and 6.89 illustrate that this upper bound can indeed be reached.

For instance, if the reduction pair $(\succcurlyeq, \succ)$ is based on a polynomial interpretation, then g is bounded by an elementary function, and hence, Theorem 6.86 using $\Phi^{\mathsf{UR}}_{\mathcal{U},(\succcurlyeq,\succ)}$ can only prove termination of TRSs whose derivational complexity is bounded by a superexponential function. As another example, if $(\succcurlyeq, \succ)$ is based on a LPO, then both g and the iteration of g are bounded by multiply recursive functions.

6.10 Experiments

In this section, we report on experiments indicating what portion of the proofs produced by modern automatic termination provers are covered by the results of this chapter. To this end, we have employed version 1.07 of the automatic termination prover $\mathsf{T_TT_2}$ by Korp, Sternagel, Zankl, and Middeldorp [64], which has been (and still is) the second most powerful, and by far the fastest tool in the standard TRS category of the annual international termination competition for several years. As in Section 5.5, we used version 8.0 of the TPDB. However, in contrast to Section 5.5, we experiment with a termination prover rather than complexity provers in this section, hence we did not exclude duplicating TRSs from our tests. The resulting testbed contains a total of 3070 TRSs. Moreover, for the same reason, we did not ignore strategy annotations in the input files when feeding them to $\mathsf{T_TT_2}$.

Our tests are based on the $\mathsf{T_TT_2}$ termination competition strategy for standard termination of TRSs. We compare the performance of $\mathsf{T_TT_2}$ on the selected testbed for the competition strategy of $\mathsf{T_TT_2}$ (which we call COMP) and restrictions of this strategy such that only techniques covered by the main theorems of this chapter are used. In all restricted strategies, we kept all techniques for creating reduction orders and reduction pairs used by $\mathsf{T_TT_2}$ intact: polynomial interpretations, matrix interpretations, arctic interpretations, LPOs, and KBOs. By the results listed in Chapter 4, all reduction orders and reduction pairs based on these techniques induce multiply recursive complexity. In order to make $\mathsf{T_TT_2}$ satisfy the assumptions of Theorem 6.9, we removed everything from the COMP strategy except all applications of reduction pairs, dependency graph and subterm criterion processors, and Theorem 2.44 (we call this strategy MREC). Another strategy (DG) further removes all applications of subterm criterion processors from the strategy, and restricts all applications of reduction pair processors to completely solve the DP problems they are applied to. This corresponds to the assumptions of Theorem 6.40. A further restriction of DG (which we call DP) eliminates all applications of dependency graph processors in the strategy. This conforms to the assumptions of Theorem 6.66. Next, we extend DP to allow the reduction pair processors to only consider the usable rules of the given DP problem. This strategy (which we name UR) corresponds to the assumptions of Theorem 6.90. Finally, we extend MREC by allowing some preprocessing steps in COMP (before the application of Theorem 2.44) where other known results about upper bounds on the derivational complexity of TRSs are applied. This final strategy (which is more liberal than MREC, but more restricted than COMP) is called TOTAL. In particular, TOTAL allows uncurrying of applicative systems (which preserves and reflects derivational complexity modulo a constant factor by the results of [114]), removal of rules by proving relative termination based on a reduction order (using [51], it follows that the derivational complexity of a TRS is primitive recursive in the complexity induced by the employed reduction order, and the derivational complexity of the remaining TRS), and direct termination proofs by match-bounds (which assert linear upper complexity bounds by Theorem 4.57). Techniques considered by TOTAL, but not by COMP, are *RFC*-match-*bounds* (a more sophisticated

	# termination proofs	avg. succ. time
COMP	1402	4.263
DP	564	2.264
UR	605	2.134
DG	691	2.805
MREC	1041	3.223
TOTAL	1250	4.405

Table 6.1: The power of various termination theorems in the dependency pair framework whose corresponding complexity bounds have been analysed

version of the techniques described in Section 4.4), *semantic labeling* (see [116]) with very simple models, a more complex way of applying uncurrying, and applications of usable rules processor which are more liberal than allowed by Theorem 6.90.

Akin to the tests of Section 5.5, all tests were executed on a server equipped with 8 AMD Opteron$^{\text{TM}}$ 2.8 GHz dual core processors with 64GB of RAM. We used a timeout of 60 seconds for each strategy and TRS. The results of the tests are shown in Table 6.1[1]. The times given in the table are seconds.

By design, COMP is the most powerful termination proving strategy in this experiment, since all other strategies are essentially restrictions of COMP. Also, it does not come as a surprise that TOTAL is the second most powerful of the considered strategies: out of all strategies which consider only termination proof techniques whose applicability guarantees a multiply recursive complexity bound, it includes the highest number of techniques. Finally, note that the ranking of the remaining strategies (MREC, DG, UR, and DP) is in line with the respective complexity bounds shown in Chapter 6.

The experimental results show how far-reaching this chapter's theorems are in the context of the current state of automatic termination analysis. Out of the systems whose termination can be proved by $\mathsf{T_TT_2}$ using its termination competition strategy, about 89% have a derivational complexity function which is provably bounded multiply recursively (or by an even tighter bound, depending on the used termination proof), using the current knowledge we have about upper bounds on the derivational complexity of TRSs (encoded into the TOTAL strategy). For 74% of the systems handled by $\mathsf{T_TT_2}$, even Theorem 6.9 in combination with all knowledge we currently have about the complexity induced by reduction pairs suffices, see the results for the strategy MREC. Even the weaker theorems of Chapter 6 (Theorems 6.40, 6.90, and 6.66) are still applicable to about half of those TRS, as shown by the results for the strategies DG, UR, and DP.

The high number of TRSs successfully handled by the TOTAL and MREC strategies (relative to the number of systems handled by COMP) confirms the

[1]See `http://cl-informatik.uibk.ac.at/users/aschnabl/experiments/thesis/mrec/` for the full experimental evidence, the testbed, and the used strategies.

presumption that the DP processors considered in Theorem 6.9 are not only the most fundamental, but also the most important ones. Moreover, note that the way Theorem 6.9 was proved in Section 6.3 above exhibited that the termination proofs considered in the assumptions of Theorem 6.9 are essentially the lexicographic combination of a number of (individually) rather simple well-foundedness arguments. The experimental results fortify the idea that this might be the case for more (if not all currently known and automated) DP processors.

6.11 Conclusion

In this chapter, we have investigated the derivational complexity of TRSs whose termination can be proved by various restrictions of the dependency pair framework, cf. Theorems 6.3, 6.37, 6.63, and 6.86.

We have established the following results: firstly, if finiteness of the initial DP problem $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ based on a TRS $\mathcal{R}$ is proved using reduction pairs which induce multiply recursive complexity, dependency graphs, and applications of the subterm criterion in an arbitrary combination, then the derivational complexity of $\mathcal{R}$ is bounded by a multiply recursive function, cf. Theorem 6.9. Secondly, if finiteness of $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ is proved by a reduction pair for each SCC of (a sound approximation of) its dependency graph, then the derivational complexity of $\mathcal{R}$ is bounded primitive recursively in the complexity induced by the "worst" of the used reduction pairs. Thirdly, if finiteness of $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ can be proved by a single reduction pair, then the derivational complexity of $\mathcal{R}$ is bounded double exponentially in the complexity induced by the reduction pair. This upper bound can be improved to a single exponential one if $\mathcal{R}$ is a SRS. Finally, if finiteness of $(\mathsf{DP}(\mathcal{R}), \mathcal{R})$ can be proved by a single reduction pair employing the usable rules refinement, then the derivational complexity of $\mathcal{R}$ is bounded by the iteration of a function which is elementary in the complexity induced by the reduction pair (this is still primitive recursive in the complexity induced by the reduction pair). For all of these results, we have presented examples which show that the stated upper bounds on the derivational complexity of the considered TRS are essentially optimal.

We have chosen to present our results in terms of derivational complexity as this simplifies the comparison to well-known results in this area. However, as mentioned in Chapter 1, derivational complexity is not the only measure of the complexity of a TRS suggested in the literature. Still, our results can be easily extended to runtime complexity, and yield upper bounds on the complexity of computing the function encoded by the considered TRS on a Turing Machine. By definition of derivational and runtime complexity, it is immediate that all upper bound results hold as well if we would study the runtime complexity of a TRS. Furthermore, the runtime complexity of a TRS is an invariant cost model [19] and thus it is straightforward to rephrase our results in terms of the complexity of the function computed by the TRS in question. Let f be a function computable by a TRS $\mathcal{R}$ and let g denote a bounding function that grows at least linearly. Suppose the runtime complexity of $\mathcal{R}$ is bounded by $g(n)$. Then there exists a Turing machine running in time polynomial in $g(n)$ that computes f [6]. Thus our results also characterise the complexity of functions computed by rewrite systems, whose termination has been shown by the dependency pair method together with natural refinements.

From the original viewpoint of derivational complexity analysis, as an analysis of the strength of termination methods, the implications of our results are easy to state. For example, our results imply that the technically simple refinement of the basic dependency pair method by dependency graphs greatly increases the strength of the method. On the other hand, our results also provide limitations on the strength of the studied techniques.

Example 6.97. Consider the following TRS $\mathcal{R}_{\mathsf{Tou}}$, which was introduced by Touzet in [105].

$$\begin{array}{ccc}
\mathsf{s}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x)))) & \mathsf{s}(\mathsf{u}(x)) \to \mathsf{s}(\mathsf{s}(x)) & \mathsf{s}(\mathsf{b}(\mathsf{s}(x))) \to \mathsf{b}(\mathsf{t}(x)) \\
\mathsf{b}(\mathsf{u}(x)) \to \mathsf{b}(\mathsf{s}(x)) & \mathsf{t}(\mathsf{b}(x)) \to \mathsf{b}(\mathsf{s}(x)) & \mathsf{t}(\mathsf{s}(x)) \to \mathsf{t}(\mathsf{t}(x)) \\
\mathsf{t}(\mathsf{b}(\mathsf{s}(x))) \to \mathsf{u}(\mathsf{t}(\mathsf{b}(x))) & \mathsf{t}(\mathsf{u}(x)) \to \mathsf{u}(\mathsf{t}(x)) &
\end{array}$$

As shown in [105], this TRS encodes the Ackermann function. Therefore, its derivational complexity is not a primitive recursive function.

Our results imply that any successful termination proof of $\mathcal{R}_{\mathsf{Tou}}$ has to employ techniques that go beyond the basic dependency pair method and simple refinements such as argument filterings, dependency graphs, and usable rules. Very recently, Sternagel and Middeldorp presented in [101] an automatic termination proof of $\mathcal{R}_{\mathsf{Tou}}$. Based on our work it is indeed no surprise that crucial ingredients of this proof are a proof tree of nontrivial depth and reduction pair or usable rules processors which do not completely solve the DP problems they are applied to. The TRSs shown in the next example make this point even clearer.

Example 6.98. Consider the following TRS $\mathcal{R}_{\mathsf{Der}}$, which was introduced by Dershowitz in [21]:

$$\begin{aligned}
\mathsf{h}(z, \mathsf{e}(x)) &\to \mathsf{h}(\mathsf{c}(x), \mathsf{d}(z, x)) \\
\mathsf{d}(z, \mathsf{g}(0, 0)) &\to \mathsf{e}(0) \\
\mathsf{d}(z, \mathsf{g}(x, y)) &\to \mathsf{g}(\mathsf{e}(x), \mathsf{d}(z, y)) \\
\mathsf{d}(\mathsf{c}(z), \mathsf{g}(\mathsf{g}(x, y), 0)) &\to \mathsf{g}(\mathsf{d}(\mathsf{c}(z), \mathsf{g}(x, y)), \mathsf{d}(z, \mathsf{g}(x, y))) \\
\mathsf{g}(\mathsf{e}(x), \mathsf{e}(y)) &\to \mathsf{e}(\mathsf{g}(x, y))
\end{aligned}$$

A variant of $\mathcal{R}_{\mathsf{Der}}$, which has been presented in [23], for instance, is the following TRS $\mathcal{R}_{\mathsf{Der2}}$:

$$\begin{aligned}
\mathsf{h}(\mathsf{e}(x), y) &\to \mathsf{h}(\mathsf{d}(x, y), \mathsf{s}(y)) \\
\mathsf{d}(\mathsf{g}(0, x), y) &\to \mathsf{e}(x) \\
\mathsf{d}(\mathsf{g}(x, y), z) &\to \mathsf{g}(\mathsf{d}(x, z), \mathsf{e}(y)) \\
\mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), 0) &\to \mathsf{e}(y) \\
\mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), \mathsf{s}(z)) &\to \mathsf{g}(\mathsf{e}(x), \mathsf{d}(\mathsf{g}(\mathsf{g}(0, x), y), z)) \\
\mathsf{g}(\mathsf{e}(x), \mathsf{e}(y)) &\to \mathsf{e}(\mathsf{g}(x, y))
\end{aligned}$$

It has been shown in [23] that $\mathcal{R}_{\mathsf{Der2}}$ faithfully simulates the Battle between Hercules and the Hydra (cf. [59]), and hence, the derivational complexity of $\mathcal{R}_{\mathsf{Der2}}$ grows far faster than any multiply recursive function, and termination of $\mathcal{R}_{\mathsf{Der2}}$ can not be proved in Peano arithmetic. The related TRS $\mathcal{R}_{\mathsf{Der}}$ has been used as a problem for the international termination competition. While it has been noted in [23] that due to a mistake, $\mathcal{R}_{\mathsf{Der}}$ does not faithfully simulate the Hydra Battle in all cases, no automatic termination prover could affirm termination of $\mathcal{R}_{\mathsf{Der}}$. However, manually created termination proofs of both $\mathcal{R}_{\mathsf{Der}}$ and $\mathcal{R}_{\mathsf{Der2}}$ have been exhibited in [23, 78].

Another related TRS we would like to mention is the following TRS $\mathcal{R}_{\mathsf{Tou2}}$ by Touzet, taken from [106]:

$$\begin{array}{rlrl}
\circ(x) &\to \bullet(\Box(x)) & & \\
\bullet(\Box(x)) &\to \Box(\bullet(\bullet(x))) & \mathsf{H}(\mathsf{0},x) &\to \circ(x) \\
\bullet(\mathsf{H}(\mathsf{H}(\mathsf{0},y),z)) &\to \mathsf{c}^1(y,z) & \bullet(\mathsf{H}(\mathsf{H}(\mathsf{H}(\mathsf{0},x),y),z)) &\to \mathsf{c}^2(x,y,z) \\
\bullet(\mathsf{c}^1(x,y)) &\to \mathsf{c}^1(x,\mathsf{H}(x,y)) & \bullet(\mathsf{c}^2(x,y,z)) &\to \mathsf{c}^2(x,\mathsf{H}(x,y),z) \\
\mathsf{c}^1(y,z) &\to \circ(z) & \mathsf{c}^2(x,y,z) &\to \circ(\mathsf{H}(y,z)) \\
\Box(\circ(x)) &\to \circ(\Box(x)) & \bullet(x) &\to x
\end{array}$$

It has been shown in [106] that $\mathcal{R}_{\mathsf{Tou2}}$ faithfully simulates the Hydra Battle for Hydrae up to a certain depth. Consequently [106, Corollary 1], the derivational complexity of $\mathcal{R}_{\mathsf{Tou2}}$ is not multiply recursive. A manually created termination proof of $\mathcal{R}_{\mathsf{Tou2}}$ is given in [106].

It follows from our results that it is theoretically impossible to give termination proofs of $\mathcal{R}_{\mathsf{Der2}}$ and $\mathcal{R}_{\mathsf{Tou2}}$ (and probably of $\mathcal{R}_{\mathsf{Der}}$, as well) by any of the termination theorems discussed in this chapter. This conclusion extends to any other proof trees for which a result similar to Theorem 6.9 can be shown.

However, we have only analysed a few of the currently known DP processors (though arguably the fundamental, and as demonstrated in Section 6.10 above, the most important ones), and for one class of these processors (the usable rules processors), only a very basic way of applying them in termination proofs was considered. It remains open how far our results extend to termination proofs by proof trees which are not captured by the theorems analysed in this chapter.

To the author's best knowledge, all reduction pairs currently used in modern automatic termination provers such as AProVE or $\mathsf{T_TT_2}$ induce multiply recursive complexity. Moreover, we have seen in our analysis in this chapter that the analysed DP processors essentially rely on rather simple termination arguments which are then combined by iteration (as shown in the analysis of the usable rules processor, where the length of a derivation starting from an "unusable term" t is essentially bounded by the combined lengths of at most $|t|$ derivations starting from "usable terms") or lexicographically (as in the termination argument underlying the joining of DP processors into a proof tree, which is illustrated by the order $\sqsupset$ on the range of the norm function). Other existing DP processors (see [103], for instance) do not seem fundamentally different in that regard. For instance, consider usable rules processors without the restrictions imposed in the assumptions of Theorem 6.90. Even though Example 6.85

demonstrates that usable rules processors cannot be directly integrated into the proof scheme we built in Section 6.3, these processors still conceptually follow the just outlined principles of lexicographic combination and iteration (note that the dependency pairs based on the rules ignored by a usable rules processor are still considered elsewhere in the proof tree).

It would not at all appear as a surprise to us if the following held true:

Conjecture 6.99. *Let $\mathcal{R}$ be a TRS whose termination can be proved in the DP framework using any proof tree which can be currently (at the time of the submission of this thesis) be obtained automatically by a termination prover. Then the derivational complexity of $\mathcal{R}$ is bounded by a multiply recursive function.*

Should this conjecture be true, then for instance, none of the existing automated termination proof techniques would in theory be powerful enough to prove termination of $\mathcal{R}_{\mathsf{Der2}}$, $\mathcal{R}_{\mathsf{Tou2}}$, possibly $\mathcal{R}_{\mathsf{Der}}$, any other term rewriting formalisations of the Hydra Battle, and finally any TRS whose derivational complexity can not be bounded by any multiply recursive function.

Chapter 7

Tyrolean Complexity Tool

He who loves practise without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast.

Leonardo da Vinci

In this chapter, we describe the Tyrolean Complexity Tool (TCT for short), which has been developed jointly by Martin Avanzini, Georg Moser, and the author of this thesis. It is a tool for automatically proving upper bounds on the derivational, innermost derivational, runtime, and innermost runtime complexity of TRSs, specialised to polynomial upper bounds. In order to derive these upper bounds, TCT uses a set of suitably restricted termination proof techniques. In particular, the techniques described in this thesis which can be used to infer polynomial upper bounds on the derivational complexity of TRSs are implemented in TCT. The tool is available online at

`http://cl-informatik.uibk.ac.at/software/tct/` .

The current release version of TCT is 1.8, therefore all information in this chapter pertains to version 1.8 of TCT. It is the fully revamped successor of the tool TCT$_1$, which is an automatic complexity prover built upon the termination prover TTT$_2$ (see [64]). TCT is free software[1] and licensed under the GNU Lesser General Public License[2].

7.1 General Design

The tool TCT is fully written in Haskell[3], and consists of approximately 16,000 lines of code. About 18% of that code is devoted to encoding various constraints imposed by the proof techniques, such as monotonicity of the interpretation functions in a well-founded or weakly monotone algebra, or compatibility

[1]Here, the term “free software” is understood as defined by the Free Software Foundation at `http://www.gnu.org/philosophy/free-sw.html`.

[2]See `http://www.gnu.org/licenses/lgpl.html` for the text of the license, and `http://www.gnu.org/licenses/gpl.html` for the text of the GNU General Public License, which the text of the GNU Lesser General Public License refers to.

[3]`http://www.haskell.org`

of a reduction order with a given TRS, into satisfiability problems in propositional logic (SAT). A further 17% provides functionality for handling terms and rewriting. For the actual termination and complexity proof techniques, 38% of the code is used. The final 27% of the code deals with the control flow of the program, provides facilities for input, output, and parallel proof attempts, and establishes convenient general interfaces for the proof techniques. All the functionality described above is divided into the following packages:

parfold: This small package provides a basic parallelism interface, which allows the tool to conveniently try out multiple proof techniques for a given TRS in parallel.

qlogic: This package implements fundamental functionality for handling formulae in propositional logic. Moreover, it provides a convenient interface for creating propositional formulae from constraints over domains such as $\mathbb{N}$ or $\mathbb{A}$.

termlib: This package serves the data structures and functionality for handling terms, TRSs, and rewrite steps. This is also where the most important input routines (the parsing of a TRS, from a file in either a plain text or an XML format) are defined.

tct: This is the main package, and it holds the largest portion of the code of TCT. It contains the implementation of all proof techniques which are used by the tool for inferring upper complexity bounds.

Moreover, it contains the encodings of *strategies*, *processors*, and *proofs*. Here, strategies are essentially directives which dictate which techniques should be used in what order for proving a complexity bound on the considered TRS. Processors are essentially executable version of basic complexity proof techniques. Conceptually (but not always factually), they are comparable to the equally named DP processors from the dependency pair framework. Finally, proofs are created when processors are successfully applied. They contain all necessary information for reconstructing a successful series of applications of processors.

This package contains the most important output routines of TCT, since proofs form the main object of its output. Furthermore, this main package includes the routines which govern the main control flow of TCT.

7.2 Usage

TCT can be obtained by either downloading an immediately runnable binary (compiled for 32-bit GNU/Linux), or compiling its source code using Haskell's cabal[4] package manager. Both the binary and the source code are available from the website of TCT, which is listed above. The tool is invoked in the following way:

```
$ ./tct <option>* <file>
```

[4]http://www.haskell.org/cabal/

Here every instance of `<option>` should be a command line flag, and `<file>` should point to a file containing the TRS which T_CT should consider. The TRS can be given in either the plain text format or the XML format used for the annual international termination competition[5]. Invoking

```
$ ./tct --help
```

produces a list of available command line flags. In the following, we focus on the flags `-a <kind>`, `-s <strategy>`, and `-S <file>`, which directly influence how TCT searches for a proof. The flag `-a <kind>` specifies whether TCT should find an upper bound on the derivational complexity, the innermost derivational complexity, the runtime complexity, or the innermost runtime complexity of the given TRS. This is indicated by the values `dc`, `idc`, `rc`, or `irc` of `<kind>`, respectively. The same information can also be given in the input file; if it is given both in the `-a` flag and the input file, then the information given by the `-a` flag takes priority. Alternatively, `<kind>` may take the values `dc!`, `idc!`, `rc!`, and `irc!`, which make TCT throw an error and exit in the case that the `-a` flag and the information given in the input file conflict. The flags `-s <strategy>` and `-S <file>` specify the strategy TCT should use when trying to prove a complexity bound for the given TRS. A strategy is written as a string, which can either be given directly as an argument to the `-s` flag, or be contained in a file which is specified by the argument of the `-S` flag. A strategy is defined by giving the name of a processor, followed by lists of *optional* and *positional* arguments for that processor, i.e., a strategy is specified in the following way:

```
<name> <optarg>* <posarg>*
```

The number and types of optional and positional arguments are specific to the particular processor. The types of arguments can be (depending on the respective processor) natural numbers, strings from a given enumeration, (sub-)strategies, Booleans (denoted as `On` or `Off`), a list of any of those types, or the option type associated with any of those types (i.e. either `none` or any value from the given type, which is mapped to the `Maybe` data type of Haskell). Every processor has a fixed list of types of positional arguments it takes. Every instance of `<posarg>` consists of exactly a value of the type demanded in the respective position, setting that positional argument to the given value. Every processor also has a list of optional arguments it may take, each given by an identifier, a type, and a default value of that type. Every instance of `<optarg>` has the shape `<ident> <value>`, where `<ident>` is an identifier of an optional argument, and `<value>` is a value of that argument's type. For each optional argument of a processor, if an instance of `<optarg>` using its identifier `<ident>` is given, then it is set to the corresponding `<value>`. Otherwise, it is set to its default value. A list of processors and the optional and positional arguments taken by them is available through the `-l` option of TCT. Finally, any substrategy `<strat>` can be given a timeout (after which it aborts and gives up) of `n` seconds by writing `[n] <strat>`.

[5]The plain text TRS format is given at `http://www.lri.fr/~marche/tpdb/format.html`, and the XML format is available at `http://www.termination-portal.org/wiki/XTC_Format_Specification`.

Example 7.1. The processor `poly` of $\mathsf{T_CT}$, which is used for finding polynomial interpretations, takes no positional arguments. However, it takes the following optional arguments:

- The argument `:kind` fixes the subclass of polynomial interpretations that should be applied. Its range of values is the enumeration of strings `stronglylinear`, `linear`, `simple`, `simplemixed`, and `quadratic`. Here, the values `stronglylinear` and `linear` refer to strongly linear and linear interpretations as in Definition 4.5, respectively. The values `simple`, `simplemixed`, and `quadratic` refer to other shapes of polynomial interpretations (see [100]), which use polynomials of higher degree. The default value is `stronglylinear`.

- The argument `:bound`, whose range of values is the set of natural numbers, specifies an upper bound on all coefficients occurring in the interpretation functions of a produced polynomial interpretation. Internally, a suitable polynomial interpretation is found by encoding the constraints for monotonicity of the interpretation functions and compatibility of the given TRS with the resulting reduction order into a satisfiability problem in propositional logic using an encoding along the lines of what is described in [30], also compare [27] for a similar encoding for matrix interpretations. Due to this encoding, it is necessary to fix an upper bound on the coefficients a priori. The default value of `:bound` is `3`.

- The argument `:bits`, whose range is the option type associated with the natural numbers, plays a similar role as the `:bound` argument. Its default value is `none`. If it is not set to `none`, i.e., it is set to some natural number `n`, then it overwrites the value of `:bound` by $\mathtt{2^n - 1}$. In other words, it provides an easy way for setting the range of the values of coefficients of the searched interpretation to "whatever can be expressed by `n` bits".

- The argument `:cbits`, whose range is the option type associated with natural numbers, provides a more implicit upper bound on the coefficients in the produced interpretation. If it is not set to `none`, it restricts the number of bits used in any coefficient occurring in $[\alpha]_{\mathcal{A}}(t)$ for any term t such that $t \trianglelefteq l$ or $t \trianglelefteq r$ for some rule $l \to r$ in the considered TRS. The default value of `:cbits` is `none`.

- The argument `:uargs` determines whether the *usable arguments* refinement, an optimisation of polynomial (and matrix) interpretations for runtime complexity analysis (see [46]), should be used if applicable. It ranges over the Booleans, and its default value is `On`.

Then, for instance, the strategy `poly :bits 5 :uargs Off` tries to find a strongly linear interpretation whose interpretation functions do not contain any coefficients larger than 31. There are no additional constraints on the values of the coefficients, and the usable arguments refinement is not used, even if it is applicable.

Other processors in TCT for direct proofs include `matrix` and `bounds` for matrix interpretations and match-bounds, respectively. Two more processors we would like to mention here are `fastest` and `best`, because they allow more than just a single strategy to be applied to the given TRS. Both of these processors take a list of strategies as their only positional argument, and no optional arguments. This list of strategies is executed in parallel. The processor `fastest` then waits for the first of these strategies which successfully produces a proof, and returns it. On the other hand, the processor `best` waits until all substrategies finish running, and finally returns the best result given by them (i.e. the lowest successfully inferred complexity bound). Finally, we mention the strategy `if <strategy> then <strategy> else <strategy>`, whose syntax, as can be seen, differs from the general syntax given above. It first applies the substrategy given to the `if`, which is typically a check for a simple syntactic property of the current problem. If it is successful, then the substrategy next to the `then` is applied subsequently. Otherwise, the substrategy next to the `else` is used.

The output of TCT is divided into two parts. The first line gives a short summary of the result produced by TCT. It is either `YES(?,X)`, for some class of functions `X`, or it is `MAYBE`, `NO`, or `ERROR`. Here `YES(?,X)` means that TCT could successfully prove that the class of functions `X` is an upper complexity bound for the TRS under consideration. In the place of the '`?`', a lower bound (if available) should be output according to the semantics fixed by the complexity division of the termination competition[6]. However, no techniques for proving lower complexity bounds for TRSs are implemented in TCT yet. The answer `MAYBE` means that the given strategy has been exhausted, but no proof of a complexity bound could be found. The answer `NO` is currently only given when the processor `fail` is explicitly called in the given strategy. Finally, `ERROR` indicates that some unexpected error occurred while TCT was running.

The second part of the output given by TCT is devoted to substantiating the initial answer unless the verbosity of TCT is set to the minimum via the `-v` flag. If the initial answer was `YES(?,X)`, then the details of the proof, and possibly some information about decisions made by the strategy (depending on the verbosity level set via the `-v` flag) are given. If the initial answer was `MAYBE` or `NO`, then details of the proof attempt, and (if existing) the part of the proof that was successfully constructed are given. Finally, if an `ERROR` was shown, then the second part of the output provides some more concrete information about the error which occurred.

Example 7.2. Consider the call of TCT shown in Figure 7.1. In this call, we try to obtain a bound on the derivational complexity (this is expressed by the flag `-a dc`) of the TRS contained in the file `TRS/SK90/2.11.trs` (this is a TRS from the TPDB, which can be obtained from the termination competition website), which defines addition and subtraction of natural numbers in unary notation. The strategy `fastest (matrix :dim 1) (matrix :dim 2) (matrix :dim 3) (matrix :dim 4)` tells TCT to try using matrix interpretations in order to prove a complexity bound. Short of `:dim`, no optional arguments are

[6]See `http://www.termination-portal.org/wiki/Complexity:Rules`

given to any of the `matrix` processors, so mostly the default way of applying the `matrix` processor is chosen (also see the information given by `./tct -l`, or, more specifically, by `./tct -l matrix`): TCT tries to apply Corollary 5.44 in order to infer complexity bounds, and the matrix entries in the interpretation function may not be greater than 3. Moreover, the dimension of the matrix interpretation should be between 1 and 4, and interpretations of all four dimensions should be searched in parallel.

From the first line of the given answer (`YES(?,O(n^2))`), we can see that TCT managed to prove that the derivational complexity of the given TRS is bounded by a polynomial of degree 2. Below, we can see a review of the proof obligation, and the details of the proof that led to the quadratic bound. The found matrix interpretation is defined by giving its interpretation functions. This interpretation has dimension 2, so the second of the four substrategies given to the `fastest` processor was the first to produce a complexity bound. Due to the nature of the `fastest` processor, it is in principle unknown which of the other three substrategies would have produced a result, and whether the resulting complexity bound would have been tighter than the quadratic one given here. It is straightforward to check that the interpretation also satisfies all demands made by the other (default-valued) arguments of the `matrix` processor: the set of matrices used in the interpretation does not satisfy the property EDA, and only 0, 1, 2, and 3 are used as matrix entries. However, note that incidentally, this interpretation is also a triangular matrix interpretation, so in this case, Theorem 5.5 would yield the same complexity bound as Corollary 5.44.

7.3 Specific Implementation Details

In this section, we discuss some implementation details of TCT for two of its most important direct termination and complexity proof techniques: polynomial interpretations and matrix interpretations. As mentioned above, suitable interpretations are found by using a SAT encoding. We now discuss this approach, and some peculiarities, of TCT in more detail.

When TCT searches for a polynomial or matrix interpretation, in a first step, the general shape of the interpretation is fixed. This is in accordance with the method for automatically finding polynomial interpretations described in [16]. For polynomial interpretations, the shape is fixed by the `:kind` argument for the corresponding processor in TCT. On the other hand, the shape of matrix interpretations in TCT is always the linear one exhibited in Definition 4.14. All coefficients of the polynomials (respectively, all matrix entries) are represented by existentially quantified variables. If the range of potential values of some variable is known to be restricted to a single value (this is the case for the zeroes in triangular matrices), then this variable is never created; instead, that single value is immediately inserted in its place. Moreover, some values (such as the coefficients of a strongly linear interpretation, or the main diagonal of a triangular matrix) are known to be restricted to be at most 1. These variables are then annotated with this restriction, which simplifies the SAT encoding later on. Let $\mathcal{A}$ be a polynomial or matrix interpretation whose coefficients (or

```
$ ./tct -a dc -s "fastest (matrix :dim 1) (matrix :dim 2) (matrix \
:dim 3) (matrix :dim 4)" TRS/SK90/2.11.trs
YES(?,O(n^2))

We consider the following Problem:

  Strict Trs:
    {  -(s(x), s(y)) -> -(x, y)
     , -(x, 0()) -> x
     , -(0(), y) -> 0()
     , +(s(x), y) -> s(+(x, y))
     , +(0(), y) -> y}
  StartTerms: all
  Strategy: none

Certificate: YES(?,O(n^2))

Proof:
  We have the following EDA-non-satisfying matrix interpretation:
  Interpretation Functions:
   0() = [0]
         [0]
   +(x1, x2) = [1 3] x1 + [1 0] x2 + [1]
               [0 1]      [0 1]      [0]
   s(x1) = [1 0] x1 + [0]
           [0 1]      [1]
   -(x1, x2) = [1 1] x1 + [1 0] x2 + [3]
               [0 1]      [0 0]      [0]

Hurray, we answered YES(?,O(n^2))
```

Figure 7.1: Sample input and output

matrix entries) are kept abstract as just described, and let $\mathcal{X}$ be the set of all such variables occurring in $\mathcal{A}$.

In order to make Theorem 4.15 (respectively, Theorem 4.6), and hence Theorem 4.19 or 5.5, or Corollary 5.44 or 5.46 (respectively, Theorem 4.10, 4.11, or 4.12) applicable, generally two sets of constraints need to be satisfied, see [16, 30]. First, all interpretation functions need to be strictly monotone in all arguments in order to make the interpretation a well-founded monotone algebra. For matrix interpretations, this is ensured by demanding that the upper left entry of each matrix is at least 1 (compare Theorem 4.15). For polynomial interpretations, we demand for every function symbol f of arity n, and every $1 \leqslant i \leqslant n$ that the polynomial $f_{\mathcal{A}}(x_1, \dots, x_n)$ contains a monomial $a \cdot x_i$, and that the constraint $a \geqslant 1$ is satisfied. Other ways to enforce strict monotonicity of polynomial interpretation functions (which are not yet implemented in T$_\mathsf{C}$T) are given by Neurauter et al. in [85].

Second, one needs to make sure that the given TRS is compatible with the resulting reduction order. This means that for each rule $l \to r$ in the given TRS, the constraint

$$\underset{v \in \mathcal{X}}{\exists v} \; \forall \alpha \;\; [\alpha]_{\mathcal{A}}(l) > [\alpha]_{\mathcal{A}}(r)$$

must hold, where $>$ is the order accompanying the considered algebra. If $\mathcal{A}$ is a polynomial interpretation, then $[\alpha]_{\mathcal{A}}(l)$ and $[\alpha]_{\mathcal{A}}(r)$ are polynomials over

$\{\alpha(v) \mid v \in \mathcal{V}\mathsf{ar}(l)\}$ and $\mathcal{X}$. If $\mathcal{A}$ is a matrix interpretation of dimension d, then $[\alpha]_{\mathcal{A}}(l)$ and $[\alpha]_{\mathcal{A}}(r)$ are both vectors of polynomials over $\{\alpha(v) \mid v \in \mathcal{V}\mathsf{ar}(l)\}$ and $\mathcal{X}$, so the constraint can be broken down into a list of d polynomial inequalities in that case. As for instance done in [16], we use the (incomplete) criterion of *absolute positivity* in order to solve these inequalities: we demand that for every power product of variables from $\{\alpha(v) \mid v \in \mathcal{V}\mathsf{ar}(l)\}$ occurring in $[\alpha]_{\mathcal{A}}(l)$ or $[\alpha]_{\mathcal{A}}(r)$, the coefficient of that power product in $[\alpha]_{\mathcal{A}}(l)$ is at least as great as its coefficient in $[\alpha]_{\mathcal{A}}(r)$. This leaves a set of *Diophantine constraints*, i.e. a set of polynomial inequalities with only existentially quantified variables.

In general, the satisfiability of Diophantine constraints is undecidable [75]. However, putting an upper bound on the values of the variables makes the problem decidable. This restricted problem is transformed into a satisfiability problem in propositional logic by essentially performing all arithmetic operations on the bit-vector level (this approach is known as *bit blasting*). Examples for such encodings of this problem in propositional logic can be found in [30, 27]. An alternative approach would be to solve the Diophantine constraints directly, as described in [16].

In the following, we sketch the propositional logic encoding used by T_CT. The data structure for propositional formulae used in the `qlogic` package of T_CT is represented by the following grammar:

$$\begin{aligned}\phi = x \mid \bigwedge[\phi,\dots,\phi] \mid \bigvee[\phi,\dots,\phi] \mid \phi \leftrightarrow \phi \mid \phi \to \phi \mid \mathsf{ite}(\phi,\phi,\phi)\\ \mid \mathsf{maj}(\phi,\phi,\phi) \mid \mathsf{odd}(\phi,\phi,\phi) \mid \neg\phi \mid \top \mid \bot\end{aligned}$$

Here x is a propositional variable, $\bigwedge$ and $\bigvee$ are conjunction and disjunction lifted to arbitrary numbers of arguments, and $\leftrightarrow$, $\to$, $\bot$, and $\top$ are the usual logical operators for equivalence, implication, contradiction, and tautology, respectively. The semantics of the ternary logical operators ite, maj, and odd are defined by the following equivalences:

$$\mathsf{ite}(x,y,z) = (\neg x \lor y) \land (x \lor z) \qquad \mathsf{maj}(x,y,z) = (x \lor y) \land (x \lor z) \land (y \lor z)$$
$$\mathsf{odd}(x,y,z) = (x \lor y \lor z) \land (x \lor \neg y \lor \neg z) \land (\neg x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor z)$$

We use these operators in our propositional language (which are more than in the propositional languages exhibited in [30, 27]) in order to express the logical constraints created by the encoding of addition and multiplication over natural numbers more concisely.

Using this structure for propositional formulae, T_CT encodes operations over natural numbers on the propositional level. In order to simulate addition and multiplication, we use the encoding from [27, Section 7.6], which produces no overflow on their own (rather, we increase bit widths before additions and multiplications as needed), and introduces new helper variables in order to avoid an exorbitant blowup of the formula. Equality and inequality of natural numbers are essentially encoded according to [30, Section 3].

The number of bits used to encode the variables in the Diophantine constraints is given by the `:bound` and `:bits` arguments of the `poly` and `matrix` processors. The number of bits for encoding results of additions and multiplications is determined by computing the maximum possible result of the operation, taking into account the maximum possible values of variables (see [30, Section 5]). An additional upper bound on these bit widths may be given by the `:cbits` argument of the `poly` and `matrix` processors.

In order to solve the resulting satisfiability problem, TCT employs the SAT solver MiniSat [25]. Since MiniSat expects its input in conjunctive normal form, TCT transforms the formula into this shape using the Plaisted-Greenbaum extension of Tseitin's transformation [90]. In order not to treat equal subformulae twice, we keep a record of all subresults while performing the transformation. Moreover, as a further optimisation, while building the logical constraints for a polynomial or matrix interpretation, TCT keeps them separated into two parts. The first part consists of simple, globally valid constraints ("side conditions"), independent of the polarity of the current working context; for instance, this includes the defining equivalences for fresh variables. The second part is the actual working constraint. Only after TCT finishes building the constraints, it connects these parts by a conjunction, thus bringing the initial formula closer to a conjunctive normal form, and hence reducing the number of new variables introduced by Tseitin's transformation.

This concludes the high-level description of the way monotonicity and compatibility constraints for polynomial and matrix interpretations are handled in TCT. The issue of ensuring that the resulting reduction order induces polynomial complexity remains. For strongly linear interpretations and triangular matrix interpretations, already the shapes of the interpretations certify polynomial complexity bounds. On the other hand, for matrix interpretations for which Corollary 5.44 or 5.46 is applicable, the condition EDA (respectively IDA_k for some specified $k \in \mathbb{N}$) is encoded directly into propositional logic. Here, we essentially follow the encoding presented in [76, Section 6.1].

Chapter 8

Conclusion

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Winston Churchill

In this thesis, we showed that many termination proofs of TRSs can be used to show not only termination of the respective TRS, but also an upper bound on its derivational complexity. To that end, we investigated both existing termination proof methods as used in automatic termination provers, and restricted termination proof methods specifically crafted for proving low complexity bounds.

In Chapter 3, we set the stage by specifying the hardness of deciding whether the derivational complexity of a given TRS is bounded by some fixed class of number-theoretic functions. Unsurprisingly, we found this problem to be highly undecidable in general. Its exact position in the arithmetical hierarchy is similar to, but not exactly the same as the position of deciding termination of a given TRS.

Chapter 4 initiated the theme of determining an upper bound on the derivational complexity of TRSs such that some specific proof technique may suffice to establish their termination. In this chapter, we listed a number of well-known direct termination proof techniques, and the corresponding known complexity results. This listing illustrated that existing direct termination proof techniques are already (complexity-wise) well-investigated, motivating us to focus on termination proof techniques specifically crafted for complexity analysis and the dependency pair framework for the rest of the thesis.

We discussed the first of these two items, techniques custom-made for derivational complexity analysis, in Chapter 5. In this chapter, we considered context dependent interpretations, a variant of polynomial interpretations, and various restrictions of matrix interpretations. Other than polynomial and matrix interpretations, their counterparts in termination analysis, the reduction orders and reduction pairs based on these techniques induce polynomial complexity. Moreover, despite the apparent difference between these techniques, we showed a connection between two interesting (with respect to mechanisability) subclasses of context dependent interpretations and matrix interpretations. Experimental evidence suggested that some restriction of matrix interpretations should indeed be one of the main methods for proving polynomial upper bounds on the derivational complexity of TRSs automatically.

Chapter 6 contains our complexity analysis of the dependency pair framework. We took the, as indicated by our experiments, most important DP processors (which are at the same time the most simple and fundamental ones), and considered termination proofs using various combinations of these processors. For each of the considered combinations, we gave an upper bound on the derivational complexity of TRSs whose termination can be proved by it. Every such upper bound was parametrised only in the complexity induced by the reduction pairs used for any reduction pair or usable rules processors in the considered termination proof. These upper bounds showed the gradual increase of the derivational complexity of TRSs whose termination can be proved when more termination proof techniques are considered. The by far most significant such increase is caused by allowing the "relative removal" of dependency pairs through the use of (reduction pair and subterm criterion) processors which do not completely solve the DP problems they are applied to. Finally, this chapter showed that a large portion of the TRSs whose termination can currently be proved terminating by automatic tools can also be proved to have multiply recursively bounded derivational complexity due to our analysis. Actually, all things shown in this chapter point toward the conjecture that every TRS whose termination can currently be proved fully automatically has multiply recursively bounded derivational complexity.

Finally, in Chapter 7, we described the tool TCT, which is an automatic prover for upper bounds on the derivational, innermost derivational, runtime, or innermost runtime complexity of TRSs. It specialises in polynomial bounds, which is the complexity class generally viewed as feasible (and we view the primary objective of an automatic complexity prover to be the provision of feasible complexity bounds). In particular, all variants of matrix interpretations described in Chapter 5 have been implemented in TCT. We gave a general overview of TCT, demonstrated how to use it, and went more into detail about the implementation of matrix interpretations (and polynomial interpretations, which are implemented in a similar way as matrix interpretations).

Of course, this work is still far away from concluding all research on derivational complexity of term rewriting. Rather, it paved the way for several new possible avenues of research, of which we would like to point out some. All methods to obtain upper bounds on the derivational complexity of TRSs we investigated were based on termination proof techniques. However, the equivalence of deciding these bounds and nontermination in the arithmetical hierarchy suggests that this might not be the only sensible approach. For instance, an inference system for the deduction of resource bounds (the considered resource is not fixed a priori; it may be time, but it may be anything else, too) in a functional programming language has been described in [56, 55, 54].

Another path for future work is the restriction of existing termination proof techniques in order to obtain low complexity bounds. In this thesis, we have discussed such variants of polynomial and matrix interpretations for derivational complexity analysis, but restricting other termination proof techniques in a similar way is imaginable, as well. There exist several such results for complexity measures of term rewriting other than derivational complexity. For instance, restrictions of MPOs exist for runtime complexity analysis [5] and implicit complexity analysis [71]. A restriction of LPOs for exponential upper bounds on the runtime complexity is described in [4]. A variant of the basic dependency pair method for runtime complexity analysis [44, 45], and a variant of the dependency pair framework for innermost runtime complexity analysis [87] exist, as well.

It also remains to check whether Conjecture 6.99 holds true. While Section 6.3 provided a complexity bound for the dependency pair framework using the (as suggested by Section 6.10) most important DP processors, and Section 6.11 argued why these results should "morally" extend to other currently known DP processors, it still has to be shown whether this is indeed the case. We do not think that it will be possible to do this in the modular manner of bounding $\mathsf{DPc}_{\mathcal{P},\mathcal{R}}$ in $\{\mathsf{DPc}_{\mathcal{Q},\mathcal{S}} \mid (\mathcal{Q},\mathcal{S}) \in \Phi((\mathcal{P},\mathcal{R}))\}$ for each DP processor Φ. This is because of DP processors which prove only finiteness of DP problems, but not well-foundedness of the underlying relative rewrite relation, such as the match-bounds (see Example 4.59), subterm criterion, and usable rules processors (see Example 6.85). Rather, we believe that, as for our simulating TRSs, the whole termination proof needs to be considered at once, and modularity can be expressed by something akin to the norm_i functions.

The complexity bounds contributed by this thesis were the result of combinatorial arguments. It might have merit to try and reconstruct these results (or further-reaching results) using proof-theoretic means, as done in [12] for Theorems 4.38 and 4.42. This could also be an alternative way of obtaining complexity bounds for other DP processors or restrictions of direct termination proof techniques.

Finally, in this thesis, we focused on *upper* bounds on the derivational complexity of TRSs. Another field of interest would be to investigate ways of obtaining *lower* complexity bounds, i.e. of proving that a certain class of functions is *not* an upper complexity bound. First steps in this direction have been made by Hofbauer and Waldmann [53].

Bibliography

[1] T. Arai. Some results on cut-elimination, provable well-orderings, induction, and reflection. *Annals of Pure and Applied Logic*, 95(1-3):93–184, 1998.

[2] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1,2):133–178, 2000.

[3] A. Asperti. The intensional content of Rice's theorem. In *Proceedings of the 35th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '08)*, pages 113–119, 2008.

[4] M. Avanzini, N. Eguchi, and G. Moser. A Path Order for Rewrite Systems that Compute Exponential Time Functions. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA'11)*, volume 10 of *LIPIcs*, pages 123–138, 2011.

[5] M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proceedings of the 9th International Symposium on Functional and Logic Programming (FLOPS '08)*, volume 4989 of *LNCS*, pages 130–146, 2008.

[6] M. Avanzini and G. Moser. Closing the gap between runtime complexity and polytime computability. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA '10)*, volume 6 of *LIPIcs*, pages 33–48, 2010.

[7] M. Avanzini, G. Moser, and A. Schnabl. Automated implicit computational complexity analysis (system description). In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR '08)*, volume 5195 of *LNCS*, pages 132–138, 2008.

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

[9] P. Baillot, J.-Y. Marion, and S. Ronchi Della Rocca. Guest editorial: Special issue on implicit computational complexity. *ACM Transactions on Computational Logic*, 10(4), 2009.

[10] J. P. Bell. A gap result for the norms of semigroups of matrices. *Linear Algebra and its Applications*, 402:101–110, 2005.

[11] R. V. Book. Time-bounded grammars and their languages. *Journal of Computer and System Sciences*, 5(4):397–429, 1971.

[12] W. Buchholz. Proof-theoretic analysis of termination proofs. *Annals of Pure and Applied Logic*, 75(1-2):57–65, 1995.

[13] C. Choppy, S. Kaplan, and M. Soria. Complexity analysis of term-rewriting systems. *Theoretical Computer Science*, 67(2–3):261–282, 1989.

[14] A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *Proceedings of the 11th International Conference on Automated Deduction (CADE '92)*, volume 607 of *LNCS*, pages 139–147, 1992.

[15] É. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Automated Certified Proofs with CiME3. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA' 11)*, volume 10 of *LIPIcs*, pages 21–30, 2011.

[16] E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

[17] S. A. Cook. Computational complexity of higher type functions. In *Proceedings of 1990 International Congress of Mathematicians*, pages 55–69. Springer Verlag, 1991.

[18] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001.

[19] U. Dal Lago and S. Martini. On constructor rewrite systems and the lambda-calculus. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP '09), Part II*, volume 5556 of *LNCS*, pages 163–174, 2009.

[20] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[21] N. Dershowitz. 33 examples of termination. In *French Spring School of Theoretical Computer Science*, volume 909 of *LNCS*, pages 16–26, 1995.

[22] N. Dershowitz. Termination by abstraction. In *Proceedings of the 20th International Conference on Logic Programming (ICLP '04)*, volume 3132 of *LNCS*, pages 1–18, 2004.

[23] N. Dershowitz and G. Moser. The Hydra battle revisited. In *Rewriting, Computation and Proof, Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *LNCS*, pages 1–27, 2007.

[24] N. Dershowitz and M. Okada. Proof-theoretic techniques for term rewriting theory. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88)*, pages 104–111. IEEE Computer Society, 1988.

[25] N. Eèn and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, volume 2919 of *LNCS*, pages 272–286, 2003.

[26] J. Endrullis, H. Geuvers, J. G. Simonsen, and H. Zantema. Levels of undecidability in rewriting. *Information and Computation*, 209(2):227–245, 2011.

[27] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(3):195–220, 2008.

[28] M. Fernández. *Models of Computation: An Introduction to Computability Theory*. Undergraduate topics in computer science. Springer Verlag, 2009.

[29] M. C. F. Ferreira. *Termination of term rewriting*. PhD thesis, Universiteit Utrecht, 1995.

[30] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *LNCS*, pages 340–354, 2007.

[31] A. Geser. *Relative Termination*. PhD thesis, Universität Passau, 1990.

[32] A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 15:149–171, 2004.

[33] 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.

[34] J. Giesl, P. Schneider-Kamp, and R. Thiemann. Aprove 1.2: Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *LNAI*, pages 281–286, 2006.

[35] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '05)*, volume 3452 of *LNAI*, pages 301–331, 2005.

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

[37] P. Hájek. Arithmetical hierarchy and complexity of computation. *Theoretical Computer Science*, 8:227–237, 1979.

[38] W. G. Handley and S. S. Wainer. Equational derivation vs. computation. *Annals of Pure and Applied Logic*, 70(1):17–49, 1994.

[39] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[40] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing Machines. *Journal of the ACM*, 13(4):533–546, 1966.

[41] G. T. Herman. Strong computability and variants of the uniform halting problem. *Mathematical Logic Quarterly*, 17:115–131, 1971.

[42] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.

[43] N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.

[44] N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR '08)*, volume 5195 of *LNCS*, pages 364–379, 2008.

[45] N. Hirokawa and G. Moser. Complexity, graphs, and the dependency pair method. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '08)*, volume 5330 of *LNCS*, pages 652–666, 2008.

[46] N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method, 2011. Submitted. Available online at `http://arxiv.org/abs/1102.3129`.

[47] D. Hofbauer. *Termination Proofs and Derivation Lengths in Term Rewriting Systems*. PhD thesis, Technische Universität Berlin, 1992.

[48] D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *Theoretical Computer Science*, 105(1):129–140, 1992.

[49] D. Hofbauer. Termination proofs by context-dependent interpretations. In *Proceedings of the 12th International Conference on Rewriting Techniques and Applications (RTA '01)*, volume 2051 of *LNCS*, pages 108–121, 2001.

[50] D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications (RTA '89)*, volume 355 of *LNCS*, pages 167–177, 1989.

[51] D. Hofbauer and J. Waldmann. Complexity bounds from relative termination proofs. Workshop on Rewriting and and Proof Theory,

2006. Slides available online at `http://www.imn.htwk-leipzig.de/~waldmann/talk/06/rpt/rel/`.

[52] D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proceedings of the 17th International Conference on Term Rewriting and Applications (RTA '06)*, volume 4098 of *LNCS*, pages 328–342, 2006.

[53] D. Hofbauer and J. Waldmann. Constructing lower bounds on the derivational complexity of rewrite systems. 2nd Workshop on Rewriting and and Proof Theory, 2010. Slides available online at `http://www.imn.htwk-leipzig.de/~waldmann/talk/10/pr/`.

[54] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*, pages 357–370. ACM, 2011.

[55] S. Jost, K. Hammond, H.-W. Loidl, and M. Hofmann. Static determination of quantitative resource usage for higher-order programs. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '10)*, pages 223–236. ACM, 2010.

[56] S. Jost, H.-W. Loidl, K. Hammond, N. Scaife, and M. Hofmann. "carbon credits" for resource-bounded computations using amortised analysis. In *Proceedings of the Second World Congress on Formal Methods (FM '09)*, volume 5850 of *LNCS*, pages 354–369, 2009.

[57] R. M. Jungers. *The Joint Spectral Radius: Theory and Applications.* Springer Verlag, 2009.

[58] S. Kamin and J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

[59] L. Kirby and J. Paris. Accessible independence results for Peano arithmetic. *Bulletin of the London Mathematical Society*, 14:285–293, 1982.

[60] D. Knuth and P. Bendix. Simple word problems in universal algebra. In *Computational Problems in Abstract Algebra*, Pergamon Press, pages 263–297, 1970.

[61] A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 202–216, 2008.

[62] M. Korp. *Termination Analysis by Tree Automata Completion.* PhD thesis, Universität Innsbruck, 2010.

[63] M. Korp and A. Middeldorp. Match-bounds revisited. *Information and Computation*, 207(11):1259–1283, 2009.

[64] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean termination tool 2. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 295–304, 2009.

[65] D. Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Texas, Austin, TX, USA, 1975.

[66] D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

[67] I. Lepper. Derivation lengths and order types of Knuth-Bendix orders. *Theoretical Computer Science*, 269(1,2):433–450, 2001.

[68] P. Lescanne. Termination of rewrite systems by elementary interpretations. *Formal Aspects of Computing*, 7(1):77–90, 1995.

[69] S. Lucas. Mu-term: A tool for proving termination of context-sensitive rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *LNCS*, pages 200–209, 2004.

[70] Z. Manna and S. Ness. On the termination of Markov algorithms. In *Proceedings of the Third Hawaii International Conference on System Science*, pages 789–792, 1970.

[71] J.-Y. Marion. Analysing the implicit complexity of programs. *Information and Computation*, 183(1):2–18, 2003.

[72] J.-Y. Marion and J.-Y. Moyen. Heap-size analysis for assembly programs, 2006. Unpublished manuscript. Available online at `http://hal.archives-ouvertes.fr/docs/00/06/78/38/PDF/main.pdf`.

[73] J.-Y. Marion and R. Péchoux. Characterizations of polynomial complexity classes with a better intensionality. In *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP '08)*, pages 79–88, 2008.

[74] J.-Y. Marion and R. Péchoux. Sup-interpretations, a semantic method for static analysis of program resources. *ACM Transactions on Computational Logic*, 10(4), 2009.

[75] Y. Matiyasevich. Enumerable sets are Diophantine. *Soviet Mathematics (Doklady)*, 11(2):354–357, 1970.

[76] A. Middeldorp, G. Moser, F. Neurauter, J. Waldmann, and H. Zankl. Joint spectral radius theory for automated complexity analysis of rewrite systems. In *Proceedings of the 4th International Conference on Algebraic Informatics (CAI '11)*, volume 6742 of *LNCS*, pages 1–20, 2011.

[77] G. Moser. Derivational complexity of Knuth-Bendix orders revisited. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '06)*, volume 4246 of *LNCS*, pages 75–89, 2006.

[78] G. Moser. The Hydra battle and Cichon's principle. *Applicable Algebra in Engineering, Communication and Computing*, 20(2):133–158, 2009.

[79] G. Moser and A. Schnabl. Proving quadratic derivational complexities using context dependent interpretations. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 276–290, 2008.

[80] G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 255–269, 2009.

[81] G. Moser and A. Schnabl. Dependency graphs, relative rule removal, the subterm criterion and derivational complexity. 11th International Workshop on Termination (WST '10), 2010. Available online at `http://cl-informatik.uibk.ac.at/users/aschnabl`.

[82] G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3:1):1–38, 2011.

[83] G. Moser and A. Schnabl. Termination proofs in the dependency pair framework may induce multiple recursive derivational complexity. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA '11)*, volume 10 of *LIPIcs*, pages 235–250, 2011.

[84] G. Moser, A. Schnabl, and J. Waldmann. Complexity analysis of term rewriting based on matrix and context dependent interpretations. In *Proceedings of the 28th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '08)*, volume 2 of *LIPIcs*, pages 304–315, 2008.

[85] F. Neurauter, A. Middeldorp, and H. Zankl. Monotonicity criteria for polynomial interpretations over the naturals. In *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR '10)*, volume 6173 of *LNCS*, pages 502–517, 2010.

[86] F. Neurauter, H. Zankl, and A. Middeldorp. Revisiting matrix interpretations for polynomial derivational complexity of term rewriting. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR (Yogyakarta) '10)*, volume 6397 of *LNCS (ARCoSS)*, pages 550–564, 2010.

[87] L. Noschinski, F. Emmes, and J. Giesl. The dependency pair framework for automated complexity analysis of term rewrite systems. In *Proceedings of the 23rd International Conference on Automated Deduction (CADE '11)*, volume 6803 of *LNCS*, pages 422–438, 2011.

[88] C. Parsons. Ordinal recursion in partial systems of number theory. *Notices of the American Mathemtical Society*, 13:857–858, 1966.

[89] R. Péter. *Recursive Functions.* Academic Press, 1967.

[90] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[91] K. W. Regan. Arithmetical degrees of index sets for complexity classes. In *Logic and Machines*, volume 171 of *LNCS*, pages 118–130, 1983.

[92] G. Roşu. Equality of streams is a Π^0_2-complete problem. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP '06)*, pages 184–191, 2006.

[93] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* The MIT Press, 1987.

[94] H. E. Rose. *Subrecursion - function and hierarchies.* Clarendon Press, 1984.

[95] F. Schernhammer and B. Gramlich. VMTL-a modular termination laboratory. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 285–294, 2009.

[96] A. Schnabl. Context Dependent Interpretations. Master's thesis, Universität Innsbruck, 2007.

[97] A. Schnabl. Cdiprover3: A tool for proving derivational complexities of term rewriting systems. In *Interfaces: Explorations in Logic, Language, and Computation, Selected Papers of the ESSLLI 2008 and 2009 Student Sessions*, volume 6211 of *LNCS*, pages 142–154, 2010.

[98] A. Schnabl and J. G. Simonsen. The exact hardness of deciding derivational and runtime complexity. In *Proceedings of the 25th International Workshop on Computer Science Logic / 20th Annual Conference of the EACSL (CSL '11)*, volume 12 of *LIPIcs*, pages 481–495, 2011.

[99] S. Sippu. Derivational complexity of context-free grammars. *Information and Control*, 53(1–2):52–65, 1982.

[100] J. Steinbach. Proving polynomials positive. In *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '92)*, volume 652 of *LNCS*, pages 191–202, 1992.

[101] C. Sternagel and A. Middeldorp. Root-labeling. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *LNCS*, pages 336–350, 2008.

[102] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[103] R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting*. PhD thesis, University of Aachen, 2007.

[104] R. Thiemann. Personal communication, 2011.

[105] H. Touzet. A complex example of a simplifying rewrite system. In *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming (ICALP '98)*, volume 1443 of *LNCS*, pages 507–517, 1998.

[106] H. Touzet. Encoding the Hydra battle as a rewrite system. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS '98)*, volume 1450 of *LNCS*, pages 267–276, 1998.

[107] V. van Oostrom. Normalisation in weakly orthogonal rewriting. In *Proceedings of the 10th Internation Conference on Rewriting Techniques and Applications (RTA '99)*, volume 1631 of *LNCS*, pages 60–74, 1999.

[108] J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *LNCS*, pages 85–94, 2004.

[109] J. Waldmann. Automatic termination. In *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 1–16, 2009.

[110] J. Waldmann. Polynomially bounded matrix interpretations. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA '10)*, volume 6 of *LIPIcs*, pages 357–372, 2010.

[111] A. Weber and H. Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

[112] A. Weiermann. A termination ordering for primitive recursive schemata, 1995. Unpublished manuscript.

[113] A. Weiermann. Termination proofs for term rewriting systems with lexicographic path orderings imply multiply recursive derivation lengths. *Theoretical Computer Science*, 139(1,2):355–362, 1995.

[114] H. Zankl, N. Hirokawa, and A. Middeldorp. Uncurrying for innermost termination and derivational complexity. In *Proceedings of the 5th International Workshop on Higher-Order Rewriting (HOR '10)*, volume 49 of *EPTCS*, pages 46–57, 2010.

[115] H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.

[116] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1,2):89–105, 1995.