Verifying Polytime Computability Automatically

dissertation

by

Martin Avanzini

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 of science

advisor: Assoc. Prof. Dr. Georg Moser

Innsbruck, 8 July 2013

dissertation

Verifying Polytime Computability Automatically

Martin Avanzini (0216396)
martin.avanzini@uibk.ac.at

8 July 2013

advisor: Assoc. 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

For Asal.

Abstract

We strive to advance the field of complexity analysis from a theoretical and practical perspective. We use *term rewrite systems* as machine model, which is a formal model of computation close to first order functional programs. A term rewrite system, the program, consists of a collection of directed equations, so called rewrite rules. Computation in this model is performed by successively applying equations from left to right.

The runtime complexity of a rewrite system, which relates the sizes of the inputs to the maximal number of steps in such computations, forms a natural cost model. In the first part of this work we argue that this cost model is indeed reasonable. Our *polytime invariance theorem* states that algorithms expressed as rewrite systems admit implementations on a conventional model of computation, the Turing machine, such that the computational complexity of this implementation is tightly related to runtime complexity of the underlying rewrite systems.

In the second part we present a novel technique to analyse the runtime complexity of rewrite systems. If this analysis is successful, we can deduce that the runtime complexity of the analysed rewrite system is bounded by a polynomial, whose degree can be precisely inferred. The described technique is purely syntactical, and as a consequence its automation is feasible. Beyond this practical application, the technique yields a resource free characterisation of the class of functions computable in polynomial time. Hence our method has also ramifications in the context of *implicit computational complexity theory*. We then generalise this technique so that exponential bounds can be inferred. In turn, this provides a resource free characterisation of the *exponential time computable functions*.

We have designed the fully automatic complexity analyser TCT, the *Tyrolean Complexity Tool*. TCT is a competitive tool that integrates a majority of the techniques known for the automated polytime complexity analysis. The final part of this work is concerned with this implementation and its underlying theoretical framework.

Acknowledgments

I would not have been able to complete this thesis without the aid and support of countless people over the last years.

First and foremost, I want to express my sincere gratitude to my supervisor Georg Moser for his patience and continuous support. His gentle guidance helped me in pursuing my research, without limiting my freedom. I could profit from Georg not only scientifically, but also on a personal level.

I want to thank Nao Hirokawa. He initially raised my interest in theoretical computer science, and teached me not to be afraid of proofs. I am thankful for the numerous fruitful discussions we had over the last years, and for the very pleasant days in Kanazawa.

I am indebted to my comrades from the CL working group in Innsbruck: Simon Bailey, Naohi Eguchi, Bertram Felgenhauer, Stéphane Gimenez, Martin Ingenhaeff, Cezary Kaliszyk, Cynthia Kop, Julian Nagele, Christian Sternagel, Thomas Sternagel, Andreas Schnabl, René Thiemann, Benjamin Winder, Sarah Winkler and last but not least Harald Zankl. The number of enjoyable discussions I had with members of the working group is uncountable. I want to thank them also for become true friends. Dedicated thanks go to Aart Middeldorp, who succeeded in establishing a professional and comfortable working environment. I am thankful for the fruitful collaborations with Andreas Schnabl and Naohi Eguchi, which is reflected also in this thesis. Without the efforts of Andreas, our jointly developed complexity tool TCT could not have matured to its current state. Joint research with Naohi helped me foster my understanding in complexity and proof theory. I am very grateful for numerous enlightening discussions.

My gratitude goes also to all my friends who still remember me, even after all these years of excusing myself from social events. I am also thankful for the discussions of the highest grade with Manu.

Last but not least, I am very indebted to my family for their ongoing support, and for providing an environment which allowed me to focus on my studies. I am particularly grateful to my wife, for her continuous support in all aspects, for her listened to my nagging, for the pleasant distractions, and for her feedback on the presentation of this thesis. It goes without saying that she had to endure a lot during the compilation of this thesis.

I gratefully acknowledge the financial support by FWF (Austrian Science Fund) and the University of Innsbruck. This work was supported via the FWF projects P20133 and I608-N18, as well as the grant 2011/2/Mip7 from the University of Innsbruck.

Contents

Chapter 1.

Introduction

Over the years, computer software has become more and more sophisticated. To produce robust software, this degree of sophistication needs to be reflected in verification techniques. Limited hardware, or in general limited computational resources, render *complexity analysis* a central topic in software verification. This form of verification is, even nowadays, often carried out manually. In order to *automate* this kind of analysis in a *broad* setting, a universal model of computation is required that is abstract enough to model the myriads of programming languages. At the same time, this formalism needs to be amenable to an automatic complexity analysis. Rooted in equational reasoning, *term rewriting* constitutes a powerful computational model. Rewriting is not only extensively applied in theorem proving [67], but underlies also much of declarative and functional programming. This work is concerned with the complexity analysis in this setting, with a strong focus on automation.

A *term rewrite system* (*TRS* for short) is a collection of *directed equations*, so called *rewrite rules*. Computation in this model is performed by successively applying rewrite rules from left to right. The *runtime complexity* of a rewrite system, which relates the sizes of the inputs to the maximal number of rewrite steps performed, forms a natural cost model for rewrite systems. Runtime complexity analysis can be seen as a refinement to *termination analysis*. And as already observed by Hofbauer and Lautemann [42], a proof of termination proves more than just the absence of infinite reductions. In many cases, a proof implies also an upper bound on the maximal length of reductions, that is, it *induces* a certain bound on the runtime complexity of the analysed rewrite system. Early research in this line [42, 41, 24] was mainly concerned with a quantitative comparison of the strength of different termination techniques. More recent work in this area takes a complementary view, viz, suiting termination techniques so that (asymptotically) precise bounds on the runtime complexity can be inferred. The seminal paper by Bonfante et al. [23] gives an early account on *taming* a termination technique so that the induced complexity is *polynomial*. Since then, a wealth of techniques have been introduced specifically to establish polynomial complexity bounds, see [59] for an overview.

With this work we provide three major contributions to the *automated polynomial runtime complexity* analysis of rewrite systems.

Polynomial Invariance Theorem: Polynomial runtime complexity analysis is motivated by Cobham's thesis, which identifies *feasible computations* with those that can be performed in polynomial time, measured in the input sizes. One may

wonder however, in which sense a bound on the runtime complexity of a rewrite system relates to the *intrinsic computational complexity* of the operations defined by the analysed rewrite system. This question is in particular pressing as a single rewrite step, which is attributed one time unit, is not an atomic operations. Instead, in unitary time rewrite systems are capable of copying arbitrary large objects. For this reason, it seems that the unitary cost model of rewriting is not *polynomially invariant* in general to the notion of cost on a conservative models of computation, for instance the Turing machine. In this work we reason that this issue is just a *representation problem*. Our *polynomial invariance theorem* states that, provided we use a representation of terms which can take sharing into account, any algorithm defined by a (terminating) rewrite system admits an implementations on a Turing machine, and this Turing machine runs in time that is *polynomially related* to the sizes of the input terms and the runtime complexity of the rewrite system. In particular, it follows that if a rewrite system defines a function and the runtime complexity is polynomially bounded, then this function is computable in polynomial time on a Turing machine.

Small Polynomial Path Orders: We introduce a novel termination techniques, the *small polynomial path order* (sPOP* for short). This order constitutes a restriction of the *recursive path order* with product status where *data tiering*, in the form of Bellantoni and Cook principle of *predicative recursion* [21], is imposed on *compatible* rewrite systems. This order delineates a class of (constructor) rewrite systems, the class of *predicative recursive* TRSs, that is, the class of TRSs compatible with sPOP*. The distinct feature of predicative recursive TRSs is that their *innermost* runtime complexity, i.e., the runtime complexity under call-by-value semantics, is bounded by a polynomial function. In particular, predictive recursive TRSs thus define only polytime computable functions. Conversely, any polytime computable function is definable with such a predicative recursive TRS. In total, the class of predicative recursive TRSs thus characterise the class of polytime computable functions. Hence the small polynomial path order gives a novel characterisation of the polytime computable functions. This *order-theoretic* characterisation is entirely *resource free*, and thus our work is closely related to similar studies in the field of *implicit computational complexity* (*ICC* for short).

On the other hand our research entails a new criteria to automatically establish polynomial runtime complexity of a given TRS. We remark that the constraints imposed by sPOP* are purely syntactic and easily verifiable, in a fully automatic setting: For any given TRS, it can be efficiently checked if it falls into the class of predicative recursive TRSs. Should this check succeed, we get an asymptotic bound on the runtime complexity directly from the parameters of the order. It should perhaps be emphasised that compatibility of a TRS with sPOP* implies termination and thus our complexity analysis technique does not presuppose termination.

This line of research is not restricted to polytime computation. In this work we also define an extension of small polynomial path orders, the *exponential path order* (EPO* for short). This order delineates the class of *predicative nested recursive TRSs*, that constitute an extension of predicative recursive TRSs. We

show that this class of TRSs admit an (innermost) runtime complexity that is bounded by an exponential, and moreover this class characterises exactly the *exponential time computable* functions.

Automated Runtime Complexity Analysis: The body of the literature provides a wealth of powerful and techniques for the automated runtime complexity analysis of rewrite systems [59]. Motivated by the theoretical advances, we have implemented a vast part of this theoretical body in a dedicated complexity analysers for rewrite systems, the *Tyrolean Complexity Tool* (TCT for short). In this work we present the theoretical framework underlying TCT. We adapt various known techniques to this framework. Noteworthy, we present a generalisation of *complexity pairs* [79]. We also adapt small polynomial path orders, *argument filterings* to increase the intensionality of the order. We prove correctness of the *dependency pair* approaches described in [38] and for innermost rewriting in [63] in our setting. Beside some obvious simplification techniques, we also present a novel transformation technique, called *dependency graph decomposition*. This technique is inspired by cycle analysis [35]. Its unique feature is that this complexity preserving transformation translates the input not only in syntactically, but also computationally simpler, sub-problems.

Outline. In the next Chapter we introduce notions and notations used throughout this work. The remainder of this work is then divided into three parts, in accordance to the categorisation of our contributions above.

In Part I, which consists of Chapters 3–7, we close the gap between the runtime complexity of rewrite systems and conventional cost models. The main results of this part, the invariance theorems for deterministic and non-deterministic time, are presented in Chapter 7.

Part II, which consists of Chapters 8–10, records our endeavour on order-theoretic characterisations. Chapter 9 deals with small polynomial path orders, in Chapter 10 is concerned with exponential path orders.

Part III consists of Chapters 11–15. This part covers our work on the automation of complexity analysis for rewrite systems. We conclude in Chapter 16.

Chapter 2.

Preliminaries

In this chapter we fix notions and notations used in this thesis. The next section is concerned with sets, relations, and orders. In Section 2.2 we recall basic notions from complexity theory, and in Section 2.3 we review definitions and notations for term rewriting.

2.1. Sets, Relations and Orders

Throughout this thesis, we denote by $\mathbb{N}$ the set of natural numbers $\mathbb{N} := \{0, 1, 2, \dots\}$, by $\mathbb{R}$ the set of reals and by $\mathbb{R}^+$ the set of non-negative reals $\mathbb{R}^+ := \{x \in \mathbb{R} \mid x \geqslant 0\}$. Given sets $A_1, \dots, A_n$ (for some $n \in \mathbb{N}$), we denote by $A_1 \times \cdots \times A_n$ the *cartesian product*

$$A_1 \times \cdots \times A_n := \{(a_1, \dots, a_n) \mid a_i \in A_i \text{ for all } i = 1, \dots, n\} .$$

For $A_1 = \cdots = A_n$ we abbreviate this product by A^n. For any set A, we denote by $\mathcal{P}(A)$ the *powerset* $\mathcal{P}(A) := \{A' \mid A \subseteq A\}$. A set of subsets $\{A_1, \dots, A_n\}$ of A is called a *partition* of A if the sets A_i $(i = 1, \dots, n)$ are pairwise disjoint and $A = \bigcup_{i=1}^{n} A_i$.

Given n sets A_i $(i = 1, \dots, n)$, an *n-ary relation* R over $A_1 \times \cdots \times A_n$ is a subset of $A_1 \times \cdots \times A_n$. In the special case $n = 2$, the relation R is called a *binary relation*, if $R \subseteq A \times A$ it is also called a *binary relation over the set A*. For a binary relation R, we frequently write $a \mathrel{R} b$ instead of $(a, b) \in R$.

If $f \subseteq A \times B$ constitutes a *function*, i.e., for every $a \in A$ there exists at most one $b \in B$ with $(a, b) \in f$, we denote this by $f : A \to B$. Functions are usually denoted by small letters $f, g, \dots$. For a function $f : A \to B$, we denote by $f(a)$ the value $b \in B$ with $(a, b) \in f$ if defined. If $f(a)$ is defined for all $a \in A$ we call f *total* (on A), otherwise it is called *partial*. To compare partial functions we use *Kleene equality*: two partial functions $f, g : A \to B$ are equal, in notation $f =_{\mathsf{k}} g$, if for all $a \in A$ either $f(a)$ and $g(a)$ are defined and $f(a) = g(a)$, or both $f(a)$ and $g(a)$ are undefined. We write $f \geqslant_{\mathsf{k}} g$ if for all $a \in A$ with $f(a)$ defined, $f(a) \geqslant_{\mathsf{k}} g(a)$ with $g(b)$ defined holds. Then $f =_{\mathsf{k}} g$ if and only if $f \geqslant_{\mathsf{k}} g$ and $g \geqslant_{\mathsf{k}} f$.

For two binary relations $R \subseteq A \times B$ and $S \subseteq B \times C$ we denote by $R \circ S \subseteq A \times C$ the *composition* of R and S:

$$R \circ S := \{(a, c) \mid \exists b \in B.\ a \mathrel{R} b \text{ and } b \mathrel{S} c\} .$$

Let R be a binary relation over A. For $n \in \mathbb{N}$ we denote by R^n the *n-fold composition of R*, i.e., $R^0 := \mathsf{id}_A$ and $R^{n+1} := R \circ R^n$, where id_A denotes the identity relation $\{(a, a) \mid a \in A\}$ over A.

Definition 2.1. A binary relation R over the set A is called

- *(ir)reflexive* if (not) $a\ R\ a$ holds for all $a \in A$;
- *symmetric* if $a\ R\ b$ implies $b\ R\ a$ for all $a, b \in A$;
- *transitive* if $a\ R\ b$ and $b\ R\ c$ implies $a\ R\ c$ for all $a, b, c \in A$;
- *well-founded* if there exists no infinite chain $a_0, a_1, \ldots$ $(a_i \in A)$ with $a_i\ R\ a_{i+1}$ for all $i \in \mathbb{N}$.
- *finitely branching* if for all elements $a \in A$, the set $\{b \mid a\ R\ b \text{ with } b \in A\}$ is finite.

Definition 2.2. Let R be a binary relation over the set A. The *reflexive closure* of R, that is, the least reflexive binary relation that contains R, is denoted by $R^{=}$. The *transitive closure* of R, that is, the least transitive binary relation that contains R, is denoted by R^{+}. The *transitive and reflexive closure* R^{*} of R is defined as the smallest transitive and reflexive binary relation that contains R.

Definition 2.3. A binary relation $\approx$ on a set A is an *equivalence relation* if it is reflexive, symmetric and transitive. For $a \in A$, we denote by $[a]_{\approx}$ the *$\approx$-equivalence class of a*, i.e., $[a]_{\approx} := \{b \in A \mid a \approx b\}$.

Definition 2.4.

(1) A (non-strict) *partial order* $\geqslant$ on a set A is a reflexive, anti-symmetric and transitive relation. The set A equipped with a partial order, $(A, \geqslant)$ is called a *partially ordered set*.

(2) A *proper order* on a set A is an irreflexive and transitive binary relation over A.

(3) A *pre-order* on A, also called *quasi-order*, is a reflexive and transitive binary relation over A.

Every pre-order $\succsim$ *induces* a proper order $\succ$ and an equivalence relation $\approx$.

Definition 2.5. Let $\succsim$ be a quasi-order, and let $\succ := \succsim \setminus \precsim$ and $\approx := \succsim \cap \precsim$. We call $\succ$ the proper order, and $\approx$ the equivalence *contained in* $\succsim$.

Observe that by definition $\succsim = \succ \cup \approx$. Conversely, the union of a proper order $\succ$ and equivalence $\approx$ does not constitute a quasi-order in general. However, whenever the *compatibility condition*

$$\approx \circ \succ \circ \approx \subseteq \succ\ , \tag{2.1}$$

is satisfied, then indeed $\succ \cup \approx$ is a quasi-order, with $\succ$ the contained proper order, and $\approx$ the contained equivalence.

A *multiset* (also sometimes called *bag*) is a collection in which elements are allowed to occur more than once. Multisets M are given as functions from A to the natural numbers, and $M(a)$ denotes the number of occurrences of a in M.

Definition 2.6. Let A be a set. A *(finite) multiset over* A is a function $M : A \to \mathbb{N}$ such that the set $\{a \in A \mid M(a) \neq 0\}$ is finite. The set of finite multisets over A is denoted by $\mathcal{M}(A)$.

We use set-like notation also for multisets, and $\{\!\{a_1, \ldots, a_n\}\!\}$ to denote multisets with (possibly repeated) elements $a_1, \ldots, a_n$. For instance, $\{\!\{1, 2, 2\}\!\}$ denotes the multiset M with $M(1) = 1$, $M(2) = 2$ and $M(a) = 0$ otherwise. Abusing notation, we overload the usual set operations $\in, \cup, \cap$ and $\setminus$ to operations on multisets, extended in the obvious way.

Definition 2.7 (Multiset Extension)**.** Let $\succ$ denote a proper order on a set A. The *multiset extension* of $\succ$ is a binary relation $\succ^{\mathsf{mul}}$ on $\mathcal{M}(A)$ defined as follows: $M_1 \succ^{\mathsf{mul}} M_2$ if $M_2 = (M_1 \setminus X) \uplus Y$ for some multisets $X, y \in \mathcal{M}(A)$ that satisfy

- $\varnothing \neq X \subseteq M_1$; and
- for all $y \in Y$ there is an $x \in X$ such that $x \succ y$.

We extend this definition to quasi-orders as follows. Let $\succcurlyeq$ denote a quasi-order, and let $\succ$ and $\approx$ denote the proper order and equivalence contained in $\succcurlyeq$. Define the extension $\sqsupset$ of $\succ$ to equivalence classes such that $[a]_\approx \sqsupset [b]_\approx$ if and only if $a \succ b$.

We define the *strict multiset extension* $\succ^{\mathsf{mul}}$ *of* $\succcurlyeq$ as $M_1 \succ^{\mathsf{mul}} M_2$ if and only if $[M_1]_\approx \sqsupset^{\mathsf{mul}} [M_2]_\approx$. Further, the *weak multiset extension* $\succcurlyeq^{\mathsf{mul}}$ *of* $\succcurlyeq$ is given by $M_1 \succcurlyeq^{\mathsf{mul}} M_2$ if and only if $[M_1]_\approx \sqsupset^{\mathsf{mul}} [M_2]_\approx$ or $[M_1]_\approx = [M_2]_\approx$ holds.

Observe that since $\succ$ and $\approx$ satisfy the compatibility condition, $\sqsupset$ is well defined. If the order $\succcurlyeq$ is a quasi-order on A then $\succ^{\mathsf{mul}}$ form a proper order, and $\succcurlyeq^{\mathsf{mul}}$ a quasi-order, on $\mathcal{M}(A)$, cf. [33].

2.2. Complexity Theory

In this section, we introduce the essential notions and notations related to *computability* and *complexity theory*. Mostly, we follow the presentation of Papadimitriou [65].

Definition 2.8 (Alphabet, Word, Language)**.** A finite set Σ is also called an *alphabet*, elements of this set are called the *letters* of Σ. A finite sequence $a_1 \cdots a_k$ of letters is called a *word* over Σ, the *empty word* (where $k = 0$) is denoted by ϵ. The set of all *words over* Σ is denoted by $\mathbb{W}(\Sigma)$. A set of words is also called a *language*.

Words are usually denoted by u, v, w, possibly followed by subscripts. We always use Σ to denote an alphabet. We denote by uv the concatenation of words $u, v \in \mathbb{W}(\Sigma)$. Note that concatenation is associative, with the empty word ϵ the neutral element. We denote by $|w|$ the *length* k of the word $w = a_1 \cdots a_k$.

Definition 2.9. Let $g : \mathbb{N} \to \mathbb{R}^+$ be a function.

(1) The set $\mathcal{O}(g)$ contains all functions $f : \mathbb{N} \to \mathbb{R}^+$ such that for some $n_0 \in \mathbb{N}$ and constant $c \in \mathbb{R}^+$, for all $n \geqslant n_0$ we have

$$f(n) \leqslant c \cdot g(n) \ .$$

(2) The set $\Omega(g)$ contains all functions $f : \mathbb{N} \to \mathbb{R}^+$ such that for some $n_0 \in \mathbb{N}$ and constant $c \in \mathbb{R}^+$, for all $n \geqslant n_0$ we have

$$c \cdot g(n) \leqslant f(n) \ .$$

If $f \in \mathcal{O}(g)$ we say that f is *asymptotically bounded from above* by g, conversely, if $f \in \Omega(g)$ we say that f is *asymptotically bounded from below* by g.

2.2.1. Turing Machines

We consider *Turing machines* with $k \geqslant 2$ working tapes, where the first tape denotes a dedicated read-only *input*, and the last tape a dedicated write-only *output* tape.

Definition 2.10. For $k \geqslant 2$, a *k-string Turing machine* (*TM* for short) with *input* and *output* tape is a quadruple (K, Σ, Δ, s) where

- K denotes a finite set of *states* of M; and
- Σ is the *alphabet* of M, containing two special symbols $\sqcup$ (the *blank*) and $\triangleright$ (the *left end marker*); and
- $\Delta \subseteq (K \times \Sigma^k) \times \big((K \cup \{\mathsf{a}, \mathsf{r}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k\big)$ is the *transition relation* of M, where a and r are called the *accepting* and *rejecting* state respectively, and $\leftarrow, \rightarrow$ and $-$ denote the cursor direction *left*, *right* and *stay*; and
- $s \in K$ the *start state* of M.

Suppose

$$\big((p, (a_1, \ldots, a_k)), (p', (a'_1, d_1), \ldots, (a'_k, d_k))\big) \in \Delta \ .$$

Then the machine M can *move*, if in state p and the letter a_i $(i = 1, \ldots, k)$ is written on the i^{th} tape under the cursor, to state p', overwriting a_i by a'_i and moving the i^{th} cursor according to direction d_i for $i = 1, \ldots, k$. We put the usual restrictions on the transition relation. We require

- for all $i = 1, \ldots, k$, if $a_i = \triangleright$ then $a'_i = \triangleright$ and $d_i = \rightarrow$, i.e., cursors cannot overwrite the left-end marker and have to move right in this case;
- $a'_1 = a_1$ and if $a_1 = \sqcup$ then $d_i = \leftarrow$, i.e., the TM M can neither overwrite nor leave the input stored on the first tape, the *input tape*;
- $d_k \neq \leftarrow$, i.e., the TM M can only proceed the cursor to the right on the last tape, the *output tape*.

We say that M is a *deterministic Turing machine* (*DTM* for short) if Δ is a function

$$\Delta : (K \times \Sigma^k) \to \big((K \cup \{\mathsf{a}, \mathsf{r}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k\big) .$$

Otherwise, it is called a *nondeterministic Turing machine* (*NTM* for short).

Definition 2.11. A *configuration* of a k-string TM $M = (K, \Sigma, \Delta, s)$ is a tuple

$$(p, w_1, u_1, \ldots, w_k, u_k) \in K \times \mathbb{W}(\Sigma)^{2 \cdot k} ,$$

where $p \in K$ denotes the *state* of the configuration, and for all $i = 1, \ldots, k$, w_i denotes the content to the left of the i^{th} cursor, including the symbol scanned by the cursor, and correspondingly u_i the tape content to the right of this cursor.

We say that the configuration $(p, w_1, u_1, \ldots, w_k, u_k)$ with $p \notin \{\mathsf{a}, \mathsf{r}\}$ *yields* the configuration $(p', w'_1, u'_1, \ldots, w'_k, u'_k)$, in notation

$$(p, w_1, u_1, \ldots, w_k, u_k) \to_M (p', w'_1, u'_1, \ldots, w'_k, u'_k) ,$$

if the following conditions are satisfied. For all $i \in 1, \ldots, k$, let a_i denote the last symbol in w_i, i.e., the symbol under the i^{th} cursor, and suppose that

$$\big((p, (a_1, \ldots, a_k)), (p', (a'_1, d_1), \ldots, (a'_k, d_k))\big) \in \Delta .$$

Then for $i = 1, \ldots, k$ we have the following:

- If $d_i = \leftarrow$, then w'_i is w_i with a_i omitted from its end, and u'_i is u_i with a'_i attached in the beginning.
- If $d_i = \rightarrow$, then w'_i is w_i with the last symbol a_i replaced by a'_i and the first symbol of u_i appended to the right ($\sqcup$ if u_i is empty), and u'_i is u_i with the first symbol removed if non-empty.
- If $d_i = -$ then w'_i is w_i with the last symbol a_i replaced by a'_i and u'_i is u_i.

The relation $\to_M$ is also called the *one step transition relation* of M. A sequence $(s, \triangleright, u, \varepsilon, \ldots, \varepsilon) \to_M c_1 \to_M c_2 \to_M \cdots$ is called a *run* of M on input u.

Definition 2.12. Let $M = (K, \Sigma, \Delta, s)$ denote a k-string TM. We say that M *accepts* $u \in \mathbb{W}(\Sigma)$ if there exist a (finite) run of M on input u ending in the accepting state a. The TM M *rejects* $u \in \mathbb{W}(\Sigma)$ if all runs of M on input u are finite and end in the reject state r. The TM M *decides* a language L, in notation $L = L(M)$, if it accepts all words $u \in \mathbb{W}(\Sigma)$ and rejects otherwise.

In the following, we give semantics to Turing machines. For the nondeterministic case we adopt the notion of function problem associated with a relation R.

Definition 2.13 (Function Problem)**.** Let $R \subseteq A \times B$ denote a binary relation. The *function problem* F_R associated with R is defined as follows: given $u \in A$ find some v such that $(u, v) \in R$ holds if v exists; otherwise reject the input.

Definition 2.14 (Computation by TM)**.** Let R denote a binary relation on $\mathbb{W}(\Sigma)$. We say that a TM M *computes the function problem* F_R if on any input $u \in L_R$, there exists for some $v \in \mathbb{W}(\Sigma)$ with $(u, v) \in R$ an accepting run

$$(s, \triangleright, u, \varepsilon, \ldots, \varepsilon) \to_M^\ell (\mathsf{a}, w_1, u_1, \ldots, w_k, u_k) \,,$$

such that for $v = w_k u_k$, i.e., v is written on the output tape. Further, for $u \notin L_R$, the TM M rejects its input.

Note that if M is deterministic, then R is a (partial) function. In this case we also say that M *computes the function* R.

Definition 2.15 (Runtime of TM)**.** For a k-string TM $M = (K, \Sigma, \Delta, s)$ we define the *runtime complexity function* $\mathsf{rc}_\mathsf{M} : \mathbb{N} \to \mathbb{N}$ by

$$\mathsf{rc}_\mathsf{M}(n) := \max\{\ell \mid \exists u.\ (s, \triangleright, u, \varepsilon, \ldots, \varepsilon) \to_M^\ell (q, w_1, u_1, \ldots, w_k, u_k) \text{ and } |u| \leqslant n\} \,.$$

Let $S : \mathbb{N} \to \mathbb{N}$ be a number-theoretic function. If $\mathsf{rc}_\mathsf{M}(n) \leqslant S(n)$ for all $n \in \mathbb{N}$ then we say that the TM M *operates in time* $S(n)$. If a TM M operates in time $S(m)$ and computes the function problem F then we simply say that M *computes* F *in time* $S(n)$. Vice versa, we say that a function problem F *is computable in (deterministic) time* $S(n)$ if there exists a (deterministic) Turing machine that computes F and runs in time $S(n)$. If S is a polynomial (linear, quadratic,...) function, we also say that F is *computable in (linear, quadratic,...) polynomial time*. Of particular interest for this work are the following classes of function problems.

Definition 2.16 (Polytime Computable Functions and Function Problems)**.**

(1) We denote by **FP** the class of *polytime computable functions*, i.e., the class of functions computable in polynomial time on deterministic Turing machines.

(2) We denote by **FNP** the class of *polytime computable function problems*, i.e., the class of function problems computable in polynomial time on Turing machines.

(3) We denote by **FEXP** the class of of *exponential time computable functions*, i.e., the class of functions computable in time $2^{p(n)}$ on deterministic Turing machines, for a polynomial $p(n) \in \mathcal{O}(n^k)$ with $k \in \mathbb{N}$.

Hence **FP** is the restriction of **FNP** if we consider only function problems that can be computed on deterministic machines. The above mentioned classes are closed under the following notion of reduction.

Definition 2.17 (Polytime Reduction)**.** We say that a function problem F *reduces* to a function problem G if there exist functions r, s, computable on Turing machines operating in time polynomial in the input, such that for any correct input x to F, $r(x)$ is a correct input to G. Furthermore, if z is a correct output of G on $s(x)$, then $s(z)$ is a correct output of F on input x.

Our definition of **FNP** departs from the one given by Papadimitriou's [65, Chapter 10], where **FNP** is defined in terms of *polynomially balanced* and *polytime decidable* relations associated with the class **NP**. Proposition 2.19 clarifies that this departure is only cosmetic. Here, a binary relation R on $\mathbb{W}(\Sigma)$ is called *polynomially balanced* if for all $(u, v) \in R$ we have $|v| \in p(|u|)$ for some polynomial function p. Further, the relation R is called *polytime decidable* if $(u, v) \in R$ is decided by a DTM M operating in polynomial time. We use the following characterisation of **NP**, compare [65, Chapter 9].

Proposition 2.18. *For a binary relation R on words, denote by $L_R := \{u \mid (u, v) \in R \text{ for some } v\}$ the* (input) language associated with R.

$$\mathbf{NP} = \{L_R \mid L_R \text{ is language associated with polynomially decidable and polynomially balanced relation } R\} .$$

Proposition 2.19. *The class* **FNP** *corresponds to the class of function problems associated with a polynomially balanced and polytime decidable relation R whose associated language L_R is in* **NP**.

Proof. First consider $F \in \mathbf{FNP}$, and let M be a nondeterministic Turing machine that computes the function problem F in polynomial time. Define the following relation R: $(u, v) \in R$ if and only if v is the encoding of an accepting run of M on input x. For this encoding it is sufficient to encode a successful sequence of configurations. Since M operates in polynomial time, the length of any computation, and also the size of each configuration, is polynomially bounded. It follows that R is polynomially balanced. Since it can be checked in linear time in $|v|$ that v encodes an accepting run of M on input v, R is polytime decidable. Thus Proposition 2.18 yields that the language L_R associated with R is in **NP**, and hence the function problem F_R associated with R, which computes an accepting runs v of M on input u, is in **FNP**.

Since **FNP** is closed under reductions, it now suffices to notice that F reduces to F_R. To see this, employ the following reduction: the function s is simply the identity function; the polytime computable function r extracts the result of M on input u from the accepting run v computed by F_R on input u.

For the inverse direction, let R be the polynomially balanced and polytime decidable relation underlying whose associated language L_R is in **NP**. Consider $(u, v) \in R$. Then on input u, an NTM N can simply guess the output v in polynomial time, and use the DTM M witnessing polytime decidability of R to see if $(u, v) \in R$ holds. We conclude that the function problem F_R with R belongs to **FNP**. □

2.2.2. Register Machines

In this thesis, we are considering *register machines* (*RM* for short) over words $\mathcal{W}(\Sigma)$ close to the initially proposed notion due to Shepherdson and Sturgis [71].

Definition 2.20. A *register machine* (*RM* for short) with k registers is a triple $M = (R, \Sigma, P)$ where

- $R = \{r_1, \ldots, r_k\}$ denotes a *finite* set of *registers*; and
- Σ denotes the *alphabet* of M; and
- $P = I_1, \ldots, I_l$ is a finite sequences of *labeled instructions.*

An *instruction* can be one of the following, for $a \in \Sigma$, $r \in R$ and $j \in \{1, \ldots, l+1\}$.

(1) Append instruction $A^{(a)}(r)$; or

(2) Delete instruction $D(r)$; or

(3) Conditional jump instruction $J^{(a)}(r)[j]$; or

(4) Copy instruction $C(r, r')$.

Informally, the effect of these operations can be stated as follows. Let $\langle r \rangle$ refer to the content of register $r \in R$. The append instruction $A^{(a)}(r)$ places $a \in \Sigma$ on the left end of $\langle r \rangle$. The delete instruction $D(r)$ removes the left-most character from $\langle r \rangle$, if $\langle r \rangle$ is not empty. The jump instruction $J^{(a)}(r)[j]$ performs a jump to instruction I_j, if the left-most character of $\langle r \rangle$ is a. By convention $l+1$ denotes a dedicated halting label, hence if $j = l+1$ the machine will simply halt. Finally, the copy instruction $C(r, r')$ overwrites $\langle r' \rangle$ by $\langle r \rangle$.

Definition 2.21. A *configuration* of a RM $M = (R, \Sigma, P)$ with k registers and instructions $P = I_1, \ldots, I_l$ is a tuple $(j, w_1, \ldots, w_k) \in \mathbb{W}(\Sigma) \times \mathbb{N}$ where

- $j \in \{1, \ldots, l+1\}$ denotes the label of the current instruction in $P = I_1, \ldots, I_l$, or the *halting label* $l+1$; and
- w_i denotes the content of the i^{th} register $(i = 1, \ldots, k)$.

We say that the configuration $(j, w_1, \ldots, w_k)$ *yields* configuration $(j', w_1', \ldots, w_k')$, in notation

$$(j, w_1, \ldots, w_k) \to_M (j', w_1', \ldots, w_k') \ ,$$

if $j \in \{1, \ldots, l\}$ and the following conditions are satisfied:

(1) if $I_j = J^{(a)}(r_i)[j'']$ and $w_i = av$ then $j' = j''$ and otherwise $j' = j$; and

(2) for $i = 1, \ldots, k$, the words w_i' are given as follows:

$$w_i' := \begin{cases} aw_i & \text{if } I_j = A^{(a)}(r_i), \\ \epsilon & \text{if } I_j = D(r_i) \text{ and } w_i = \epsilon, \\ v & \text{if } I_j = D(r_i) \text{ and } w_i = av \text{ with } a \in \Sigma, \\ w_i & \text{otherwise.} \end{cases}$$

Remark. Our definition departs from [71] in the following respects. Unlike in [71], we suppose that the set of registers is finite. Due to the absence of memory indirection instructions, this simplification does not impose any restriction on register machines. The instructions 2.20(1)–2.20(3) correspond to the *minimal* instruction set given in [71, Section 6], with the difference that in the append instruction appends to the right instead of to the left. The additional copy instruction (4) added from the extended instruction set of [71, Section 2] ensures that copying is an atomic operation.

The next definition gives semantics to register machines. By convention, we assume n dedicated *input registers* (which unlike for TMs can in the course of evaluation be modified), and the last register is a dedicated *output register*. Note that register machines as defined here act deterministically, every configuration yields at most one next configuration.

Definition 2.22 (Computation by RM)**.** Let $M = (R, \Sigma, P)$ denote a RM with k registers and l instructions. We say that M *computes* the (partial) function $f_M : \mathcal{W}(\Sigma)^n \to \mathcal{W}(\Sigma)$ with $n \leqslant k$ defined as follows:

$$f_M(u_1, \ldots, u_n) := v_k \quad :\Leftrightarrow \quad (u_1, \ldots, u_n, \vec{\epsilon}, 1) \to_M^{\ell} (v_1, \ldots, v_k, l+1) \; .$$

The first n registers are also called the *input registers*, and the final register the *output register*.

We adopt a *unit cost* measure for register machines. As for Turing machines each transition accounts for one step in time.

Definition 2.23 (Runtime of RM)**.** For a RM $M = (R, \Sigma, P)$, we define the *runtime complexity function* $\mathsf{rc}_\mathsf{M} : \mathbb{N} \to \mathbb{N}$ by

$$\mathsf{rc}_\mathsf{M}(n) := \max\{\ell \mid \exists u_1, \ldots, u_n. \; (u_1, \ldots, u_n, \vec{\epsilon}, 1) \to_M^{\ell} (v_1, \ldots, v_k, l) \text{ and } \sum_{i=1}^{n} |u_i| \leqslant m\} \; .$$

Again we say that for a RM M, if $\mathsf{rc}_\mathsf{M}(n) \leqslant S(n)$ for all $n \in \mathbb{N}$ then M *operates in time* $S(n)$. If M operates in time $S(n)$ and computes the function f then we say that M computes f in time $S(n)$. The class of polytime computable functions **FP** is quite robust with respect to the underlying computational model. In particular register machines and k-string Turing machines can simulate each other within polynomial overhead, see for instance the book of Jones [46].

Proposition 2.24. *The following classes of functions are equivalent:*

(1) *The class of functions computable by a deterministic Turing machine running in polynomial time; and*

(2) *The class of functions computable by a Register machine running in polynomial time.*

Note that we silently assumed here an encoding of input vectors $u_1, \ldots, u_n$ to register machines as input word $u_1; \ldots; u_n$ to Turing machines, and vice versa.

2.3. Term Rewriting

We assume modest familiarity with the basics of (first-order) term rewriting. See the book of Baader and Nipkow [16] for an introduction to rewriting.

Definition 2.25 (Signature)**.** Let $\mathcal{F}^0, \mathcal{F}^1, \dots$ be a family of sets. Then $\mathcal{F} := \bigcup_{k\in\mathbb{N}} \mathcal{F}^k$ is a *signature*. For each $f \in \mathcal{F}$, we call f a *function symbol*. If $f \in \mathcal{F}^k$, then we say that f is of *arity* k, or k-ary for brevity. If f has more than one arity, i.e., it occurs in $\mathcal{F}^k$ and $\mathcal{F}^l$ for $k \neq l$, then f is called *variadic*, otherwise it is called *non-variadic*. The signature $\mathcal{F}$ is called *variadic* if it contains at least one variadic function symbol, otherwise $\mathcal{F}$ is called *non-variadic*.

If not mentioned otherwise, we assume the signature to be finite and non-variadic. We sometimes write $f/k \in \mathcal{F}$ to indicate that the arity of f is k in $\mathcal{F}$. If not stated otherwise, $f, g, h, \dots$ denote function symbols.

Definition 2.26 (Terms)**.** Let $\mathcal{V}$ denote a countably infinite set of *variables* disjoint from the signature $\mathcal{F}$. The set of *terms* over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. It is defined as the smallest set satisfying the following conditions:

(1) if $x \in \mathcal{V}$, then $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$; and

(2) if $f/k \in \mathcal{F}$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ for all $i = 1, \dots, k$ then $f(t_1, \dots, t_k) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

Throughout this thesis, we fix a signature $\mathcal{F}$ and variables $\mathcal{V}$ that satisfy the conditions of Definition 2.26. If not mentioned otherwise, x, y, z, possibly followed by subscripts, denote variables, and $s, t, \dots$, possibly followed by subscripts, denote terms. For term t, $n \in \mathbb{N}$ and $f \in f/1$ we also write $f^n(t)$ for the term $f(\cdots f(t) \cdots)$ with n occurrences of the unary symbol f outside t. For a term $f(t_1, \dots, t_n)$, the function symbol f is called the *root* of $f(t_1, \dots, t_n)$. A term t is *ground* if it does not contain variables. We abbreviate the set of all *ground terms* over a signature $\mathcal{F}$ by $\mathcal{T}(\mathcal{F})$.

Definition 2.27 (Size, Depth)**.** Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. We define the *size* $|t|$ of t recursively by

$$|t| := \begin{cases} 1 & \text{if } t \text{ is a variable,} \\ 1 + \sum_{i=1}^{k} |t_i| & \text{if } t = f(t_1, \dots, t_k). \end{cases}$$

We define the *depth* $\mathsf{dp}(t)$ of t recursively as

$$\mathsf{dp}(t) := \begin{cases} 0 & \text{if } t \text{ is a variable or a constant,} \\ 1 + \max_{i=1}^{k} \mathsf{dp}(t_i) & \text{if } t = f(t_1, \dots, t_k) \text{ otherwise.} \end{cases}$$

Definition 2.28 (Positions)**.** A *position* is a finite sequence of positive natural numbers.

(1) The set $\mathcal{P}\mathrm{os}(t)$ of all positions in $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is inductively defined as follows:

$$\mathcal{P}\mathrm{os}(t) := \begin{cases} \{\epsilon\} & \text{if } t \in \mathcal{V}, \\ \{\epsilon\} \cup \{i{\cdot}p \mid p \in \mathcal{P}\mathrm{os}(t_i) \text{ and } i = 1, \dots, k\} & \text{if } t = f(t_1, \dots, t_k). \end{cases}$$

Here ϵ denotes the *empty position*, and $p{\cdot}q$ the *concatenation* of positions p and q.

(2) We say that a position p is *above* a position q if there exists a position r such that $p{\cdot}r = q$. If p is above q we also say that q is *below* r, and we write $p \leqslant q$. We write $p < q$ if $p \leqslant q$ and $p \neq q$. Positions p and q are called *parallel*, in notation $p \parallel q$, if neither $p \leqslant q$ nor $q \leqslant r$ holds.

Definition 2.29 (Subterm, Subterm Relation)**.**

(1) Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. The *sub-term* of t at position $p \in \mathcal{P}\mathrm{os}(t)$ is denoted by $t|_p$ and defined by

$$t|_p := \begin{cases} t & \text{if } p = \epsilon, \\ t_i|_q & \text{if } p = i{\cdot}q. \end{cases}$$

(2) We define the *sub-term relation* $\trianglelefteq$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $s \trianglelefteq t$ holds if (i) $s = t$, or (ii) $t = f(t_1, \ldots, t_k)$ and $s \trianglelefteq t_i$ for some $i = 1, \ldots, k$. We define $s \triangleleft t$ if $s \trianglelefteq t$ and $s \neq t$, and call $\triangleleft$ the *strict sub-term relation.*

Note that $s \trianglelefteq t$ holds iff $s = t|_p$ for some position $p \in \mathcal{P}\mathrm{os}(t)$. In the case $s \trianglelefteq t$ ($s \triangleleft t$) we call s a *(proper) sub-term of* t. We denote by $\trianglerighteq$ and $\triangleright$ the converse of $\trianglelefteq$ and $\triangleleft$ respectively.

Definition 2.30 (Substitution)**.** A *substitution* σ is a finite mapping from $\mathcal{V}$ into $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We also denote by σ the homomorphic extension of σ to terms:

$$\sigma(t) := \begin{cases} \sigma(t) & \text{if } t \text{ is a variable} \\ f(\sigma(t_1), \ldots, \sigma(t_k)) & \text{if } t = f(t_1, \ldots, t_k). \end{cases}$$

If not mentioned otherwise, σ, τ denote substitutions. We write $t\sigma$ instead of $\sigma(t)$. If for two terms s and t we have $s = t\sigma$ for some substitution σ, then s is called an *instance* of t. The terms s and t are *unifiable* if there exists a substitution σ with $s\sigma = t\sigma$.

Consider a fresh constant symbol $\Box \notin \mathcal{F}$, named *hole*. Terms that contain holes are called *contexts*.

Definition 2.31 (Context)**.** We call an element $C \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{V})$ with $n \geqslant 1$ occurrences of $\Box$ an n-holed *context*. Let $p_1, \ldots, p_n$ denote all positions in $\mathcal{P}\mathrm{os}(C)$ such that $C|_{p_i} = \Box$, sorted in lexicographic order. For terms $t_1, \ldots, t_n$, we denote by $C[t_1, \ldots, t_n]$ the term obtained by replacing the holes at position p_i with t_i.

We also call the context $\Box$ the *empty context*. For a one-holed context C and term t, we also write $C[t]_p$ where p indicates the (unique) position of the hole $\Box$ in p.

Definition 2.32 (Closure under Substitutions, Contexts)**.** A binary relation R on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is *closed under substitutions* if whenever $s \mathrel{R} t$ holds then $s\sigma \mathrel{R} t\sigma$ holds for any substitution σ. The relation R is *closed under contexts* if $s \mathrel{R} t$ implies $C[s] \mathrel{R} C[t]$ for all one-holed contexts C over $\mathcal{F}$ and $\mathcal{V}$. If R is closed under substitutions and contexts then it is also called a *rewrite relation*.

Definition 2.33 (Rewrite Rule, Term Rewrite System, Defined Symbols)**.**

(1) A *rewrite rule* over the signature $\mathcal{F}$ and variables $\mathcal{V}$ is a pair $(l, r) \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})^2$, in notation $l \to r$, such that l is not a variable and all variables in r occur also in l. Here l is called the *left-hand*, and r the *right-hand side* of $l \to r$.

(2) A *term rewrite system* (*TRS* for short) $\mathcal{R}$ over $\mathcal{F}$ and $\mathcal{V}$ is a set of *rewrite rules* over $\mathcal{F}$ and $\mathcal{V}$.

(3) If $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$ then f is called a *defined symbol* of $\mathcal{R}$. The set of all defined symbols of $\mathcal{R}$ is denoted by $\mathcal{D}_\mathcal{R}$.

To distinguish function symbols from variables, we draw in examples function symbols always in serif font.

The rewrite relation of $\mathcal{R}$ is the least extension of $\mathcal{R}$ that is closed under contexts and substitutions. It can be defined directly as follows

Definition 2.34 (Rewrite Relation)**.** For terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ we define $s \to_\mathcal{R} t$ if there exists a (one-holed) context C, substitution σ and rewrite rule $l \to r \in \mathcal{R}$ such that $s = C[l\sigma]_p$ and $t = C[r\sigma]$. The relation $\to_\mathcal{R}$ is also called the *rewrite relation of* $\mathcal{R}$. The position p is called the *rewrite position*, the term $l\sigma$ a *redex* in s.

If not mentioned otherwise, $\mathcal{R}, \mathcal{S}, \ldots$, possibly followed by subscripts, denote *finite* rewrite systems over the signature $\mathcal{F}$ and variables $\mathcal{V}$. In order to indicate the rewrite rule $l \to r$ and rewrite position p involved in a step $s \to_\mathcal{R} t$ we sometimes write $s \to_{\mathcal{R}, l\to r, p} t$.

Definition 2.35. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is called a *normal form* with respect to a relation $\to$ on terms if $t \to s$ does not hold for any term s. The set of all such normal forms is denoted by $\mathrm{NF}(\to)$. For a TRS $\mathcal{R}$ over $\mathcal{F}$ and $\mathcal{V}$, we abbreviate $\mathrm{NF}(\mathcal{R}) := \mathrm{NF}(\to_\mathcal{R})$ and call $\mathrm{NF}(\mathcal{R})$ the *normal forms of* $\mathcal{R}$.

In this thesis, we will sometimes adopt *call-by-value* semantics.

Definition 2.36 (Innermost Rewrite Relation)**.** Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{V}$. For terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ we define $s \xrightarrow{\mathrm{i}}_\mathcal{R} t$ if there exists a (one-holed) context C, substitution σ and rewrite rule $l \to r \in \mathcal{R}$ such that $s = C[l\sigma]$, $t = C[r\sigma]$ and the redex $l\sigma$ is *argument normalised*, i.e., all proper sub-terms of $l\sigma$ are $\mathcal{R}$ normal forms. The relation $\xrightarrow{\mathrm{i}}_\mathcal{R}$ is also called the *innermost rewrite relation of* $\mathcal{R}$.

2.3.1. Rewriting as Computational Model

To give semantics to rewrite systems, we suppose an a priori separation of the signature $\mathcal{F}$ into *defined symbols* $\mathcal{D}$ and constructors $\mathcal{C}$.

Definition 2.37 (Values, Basic Terms)**.** Let $\mathcal{D} \subseteq \mathcal{F}$ be a set of *defined symbols*, and $\mathcal{C} \subseteq \mathcal{F}$ be a set of *constructors*, such that $\{\mathcal{D}, \mathcal{C}\}$ forms a partition of $\mathcal{F}$.

(1) Elements from $\mathcal{T}(\mathcal{C})$ are called *values*;

(2) The set $\mathcal{T}_{\mathsf{b}}(\mathcal{D} \uplus \mathcal{C})$ of *basic terms* over defined symbols $\mathcal{D}$ and constructors $\mathcal{C}$ is the least set of terms $f(v_1, \ldots, v_k)$ with $f/k \in \mathcal{D}$ and $v_i \in \mathcal{T}(\mathcal{C})$ for all $i = 1, \ldots, k$.

If not mentioned otherwise, we use u, v, w, possibly followed by subscripts for values. We suppose $\mathcal{C}$ contains at least one constant, thus $\mathcal{T}(\mathcal{C}) \neq \varnothing$. A (finite) *computation* of $f \in \mathcal{D}$ on input values $v_1, \ldots, v_k$ is given by a rewrite sequence

$$f(v_1, \ldots, v_k) = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \cdots \to_{\mathcal{R}} t_\ell = w \,.$$

If the above computation ends in a value, i.e., $w \in \mathcal{T}(\mathcal{C})$, we also say that f *computes* on input $v_1, \ldots, v_n$ in ℓ steps the value w. Term rewrite systems are inherently non-deterministic. In general, a computation of $f(v_1, \ldots, v_k)$ is not unique since there might be more than one choice of a rewrite rule and rewrite position to reduce a given term t_i.

Definition 2.38 (Terminating, Confluent, Complete)**.** A TRS $\mathcal{R}$ is called

- *terminating* if the rewrite relation $\to_{\mathcal{R}}$ is well-founded;
- *confluent* if for every pair of terms u, v with $u \,{}^*_{\mathcal{R}}\!\leftarrow s \to^*_{\mathcal{R}} v$ for some term s there is a common reduct t of u and v: $u \to^*_{\mathcal{R}} t \,{}^*_{\mathcal{R}}\!\leftarrow v$;
- *complete* if it is terminating and confluent.

Completeness of $\mathcal{R}$ ensures determinism in the input output behaviour of $\mathcal{R}$. Since $\mathcal{R}$ is terminating, reduction of a term t under any strategy will eventually hit some normal form, which by confluence is even unique. Of course the length of reductions might differ considerably when reducing a term t under different reduction strategies.

To account for nondeterministic computation, we capture semantics of $\mathcal{R}$ by assigning to each n-ary defined symbol $f/k \in \mathcal{D}$ a relation $[\![f]\!]_{\mathcal{R}}$ that maps input arguments $(v_1, \ldots, v_k) \in \mathcal{T}(\mathcal{C})^k$ to computed values w. A *finite* set $\mathcal{N}$ of *non-accepting patterns* is used to distinguish meaningful outputs w from outputs that should not be considered part of the computation. A value w *is accepting* with respect to $\mathcal{N}$ if no $p \in \mathcal{N}$ and no substitution σ exists, such that $p\sigma = w$ holds.

Definition 2.39 (Computation by Rewriting)**.** Let $\mathcal{R}$ be a TRS and let $\mathcal{N}$ be a set of non-accepting patterns. For each $f/k \in \mathcal{D}$ the *relation* $[\![f]\!]_{\mathcal{R},\mathcal{N}} \subseteq \mathcal{T}(\mathcal{C})^k \times \mathcal{T}(\mathcal{C})$ *defined by* f *in* $\mathcal{R}$ is given by

$$((v_1, \ldots, v_k), w) \in [\![f]\!]_{\mathcal{R},\mathcal{N}} \;:\Leftrightarrow\; f(v_1, \ldots, v_k) \to^!_{\mathcal{R}} w \text{ and } w \in \mathcal{T}(\mathcal{C}) \text{ is accepting} \,.$$

We say that $\mathcal{R}$ *computes* the *function problem* associated with $[\![f]\!]_{\mathcal{R},\mathcal{N}}$.

The assertion that for values w, w is accepting amounts to our notion of *accepting run* of a TRS $\mathcal{R}$. When $\mathcal{N} = \varnothing$, i.e., when all values are accepting, we simply write $[\![\cdot]\!]_{\mathcal{R}}$ instead of $[\![\cdot]\!]_{\mathcal{R},\mathcal{N}}$. When no confusion can arise we may also drop the reference to $\mathcal{R}$. If $\mathcal{R}$ is confluent, then $[\![f]\!]_{\mathcal{R},\mathcal{N}}$ is in fact a function from $\mathcal{T}(\mathcal{C})^k$

to $\mathcal{T}(\mathcal{C})$. In this case we also say that $\mathcal{R}$ *computes the function* $[\![f]\!]_{\mathcal{R},\mathcal{N}}$. Note that even $[\![f]\!]_{\mathcal{R}}$ can be partial, either because $\mathcal{R}$ is not terminating, or because the normal form w as above is not a value.

Both termination and confluence are undecidable properties in general. Nevertheless, the techniques developed in this thesis will often imply termination. The following conditions give a natural and decidable, although very strong, condition on rewrite systems that implies confluence. Call a bijective substitution $\sigma : \mathcal{V} \to \mathcal{V}$ a *renaming*. Then a rewrite rule $l' \to r'$ is called a variant of $l \to r$ if there exists a renaming σ such that $l' = l\sigma$ and $r' = r\sigma$.

Definition 2.40 (Left-Linear, Non-Overlapping, Orthogonal)**.** Let $\mathcal{R}$ denote a TRS. Then $\mathcal{R}$ is called

(1) *left-linear* if left-hand sides are *linear*, that is, every variable occurs at most once in the term;

(2) *non-overlapping* if $\mathcal{R}$ does not give rise to *overlaps*. Here an *overlap* is a triple $\langle l_1 \to r_1, p, l_2 \to r_2 \rangle$ satisfying

- $l_1 \to r_1$ and $l_2 \to r_2$ are variants of rules from $\mathcal{R}$ without common variables,
- $l_1|_p \notin \mathcal{V}$,
- $l_1|_p$ and l_2 are unifiable, and
- if $p = \varepsilon$ then $l_1 \to r_1$ and $l_2 \to r_2$ are not variants;

(3) *orthogonal*, if it is *left-linear* and *non-overlapping*.

Proposition 2.41. *Every orthogonal TRS $\mathcal{R}$ is confluent, in particular $[\![f]\!]_{\mathcal{R}}$ is a partial function from $\mathcal{T}(\mathcal{C})^k$ to $\mathcal{T}(\mathcal{C})$ for every $f \in \mathcal{D}$.*

In this thesis, we also consider a nicely behaved sub-class of TRSs. For the lack of a better name we call TRSs from this class ML-*like*, as we impose the usual restriction[1] from functional programming languages in the spirit of ML [57].

Definition 2.42 (Constructor TRS, Completely Defined, ML-like)**.** Let $\mathcal{R}$ denote a TRS. Then $\mathcal{R}$ is called

(1) a *constructor TRS* if left-hand sides of rules in $\mathcal{R}$ are *constructor-based*, where a term $f(s_1, \ldots, s_n)$ is constructor-based if $f \in \mathcal{D}$ and $s_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ for all $i = 1, \ldots n$.

(2) completely defined, if every defined symbol $f \in \mathcal{D}$ is *completely defined* with respect to $\mathcal{R}$. Here $f \in \mathcal{D}$ is called *completely defined* if it does not occur in ground normal forms of $\mathcal{R}$.

(3) ML-*like*, if it is a *completely defined*, *orthogonal constructor TRS*.

[1] ML does not require that functions are completely defined, i.e., pattern matches cover all cases. Rather, this is considered good programming practice.

The set of ML-like TRSs constitutes a deterministic computational model that respects our separation of data and computation, as imposed by the separation of defined and constructor symbols. The following observation is immediate from the definition.

Proposition 2.43. *Let $\mathcal{R}$ be an* ML*-like TRS. Then normal forms coincide with values:* $\mathrm{NF}(\mathcal{R}) = \mathcal{T}(\mathcal{C})$. *In particular, if $\mathcal{R}$ is terminating then $[\![f]\!]_{\mathcal{R}}$ constitutes a total function for every $f \in \mathcal{D}$.*

The requirement that an ML-like TRS is completely defined is not a severe restriction, using a constant $\bot \in \mathcal{C}$, any constructor TRS can be extended by sufficiently many rules $f(l_1, \dots, l_n) \to \bot$ so that f is completely defined, without modifying semantics.

2.3.2. Complexity Analysis of Rewrite Systems

Hofbauer and Lautemann proposed first to assess the complexity of a given TRS as the maximal length of derivation sequences, the *derivation height*. More precisely the *derivational complexity function* relates the derivation height with the size of the starting term [42]. This notion is frequently used to assess the complexity and the strength of termination techniques, a line of research that has been widely explored [41, 78, 58, 60].

Definition 2.44 (Derivation Height)**.** The *derivation height* $\mathsf{dh}(t, \to)$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with respect to a binary relation $\to$ on terms is defined as

$$\mathsf{dh}(t, \to) := \max\{\ell \mid \exists t_1, \dots, t_\ell.\ t = t_0 \to t_1 \to \cdots \to t_l\} \ .$$

Example 2.45. Consider the TRS $\mathcal{R}_{\mathsf{double}}$ consisting of the following two rules:

$$1\colon \mathsf{double}(0) \to 0 \qquad 2\colon \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \ .$$

For $n \in \mathbb{N}$, let $\mathbf{n} := \mathsf{s}^n(0)$. The TRS $\mathcal{R}_{\mathsf{double}}$ computes the function

$$\begin{aligned} [\![\mathsf{double}]\!]_{\mathcal{R}} \colon \mathcal{T}(\{0, \mathsf{s}\}) &\to \mathcal{T}(\{0, \mathsf{s}\}) \\ \mathbf{n} &\mapsto \mathbf{2} \cdot \mathbf{n} \ , \end{aligned}$$

that is, it doubles natural numbers given in unary notation. Note that the length of every computation of $\mathsf{double}(\mathbf{n})$ is linear in n. To be more precise, we have $\mathsf{dh}(\mathsf{double}(\mathbf{n}), \to_{\mathcal{R}}) = n + 1$. On the other hand, for arbitrary terms t, the length of derivations starting from t is bounded from below by an exponential, as witnessed by the family of terms $\mathsf{double}^n(\mathbf{1})$, for all $n \in \mathbb{N}$. ◁

In order to account for the fact that computations start only from basic terms, Hirokawa and Moser proposed later [38] to study the *runtime complexity function*, a variation of the derivational complexity function that takes only derivations of basic terms into account. Following Hirokawa and Moser we study (primarily) the runtime complexity function of TRSs. We will justify this cost model in the next part.

Definition 2.46. For a set $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $n \in \mathbb{N}$, let $\mathsf{cp}(n, T, \to)$ denote the derivation height of terms up to size n, that is

$$\mathsf{cp}(n, T, \to) := \max\{\mathsf{dh}(t, \to) \mid \exists t \in T \text{ and } |t| \leqslant n\} .$$

Let $\mathcal{R}$ denote a TRS.

(1) The *derivational complexity function* $\mathrm{dc}_\mathcal{R}(n) : \mathbb{N} \to \mathbb{N}$ of $\mathcal{R}$ is defined as

$$\mathrm{dc}_\mathcal{R}(n) := \mathsf{cp}(n, \mathcal{T}(\mathcal{F}), \to_\mathcal{R}) .$$

(2) The *runtime complexity function* $\mathrm{rc}_\mathcal{R}(n) : \mathbb{N} \to \mathbb{N}$ of $\mathcal{R}$ is defined as

$$\mathrm{rc}_\mathcal{R}(n) := \mathsf{cp}(n, \mathcal{T}_\mathsf{b}(\mathcal{D} \uplus \mathcal{C}), \to_\mathcal{R}) .$$

(3) The *derivational complexity function* $\mathrm{dci}_\mathcal{R}$ and *runtime complexity functions* of $\mathcal{R}$ are obtained from above by replacing the rewrite relation $\to_\mathcal{R}$ by the innermost rewrite relation $\xrightarrow{\mathsf{i}}_\mathcal{R}$.

Note that the derivation height of a term t with respect to a relation $\to$ is defined whenever $\to$ is finitely branching and terminating. Since $\to_\mathcal{R}$ is finitely branching whenever $\mathcal{R}$ is finite, the (innermost) derivational and runtime complexity are well-defined whenever $\mathcal{R}$ is finite and terminating. Suppose $\mathrm{dc}_\mathcal{R}, \mathrm{rc}_\mathcal{R}, \mathrm{dci}_\mathcal{R}$ or $\mathrm{rci}_\mathcal{R}$ are asymptotically bounded from above by a linear, quadratic,..., polynomial function. Then we simply say that the (innermost) derivational or runtime complexity of $\mathcal{R}$ is linear, quadratic,..., or polynomial.

Proposition 2.47. *Let $\mathcal{R}$ denote a terminating TRS. Then the following relationship holds for all $n \in \mathbb{N}$.*

$$\mathrm{rci}_\mathcal{R}(n) \begin{array}{c} \leqslant \\ \leqslant \end{array} \begin{array}{c} \mathrm{rc}_\mathcal{R}(n) \\ \mathrm{dci}_\mathcal{R}(n) \end{array} \begin{array}{c} \leqslant \\ \leqslant \end{array} \mathrm{dc}_\mathcal{R}(n).$$

2.3.3. Termination Analysis of Rewrite Systems

Termination analysis is a well established research area in rewriting [16, 75, 33]. Rewriting forms a Turing complete model of computation, cf. for instance [16, Section 5], and termination is in general undecidable. Nevertheless, a wealth of powerful, although incomplete or undecidable, techniques exist to show termination of rewrite systems. Prominent in rewriting is the use of *reduction orders*.

Definition 2.48 (Reduction Order, Compatibility)**.**

(1) A proper order on terms that is also a rewrite relation is called a *rewrite order*;

(2) A *reduction order* is a well-founded rewrite order.

Note that $\to_{\mathcal{R}} \subseteq \succ$ for reduction orders $\succ$, and thus termination, can be proven by orienting rules from left to right. This motivates the following definition.

Definition 2.49 (Compatible)**.** We say that the TRS $\mathcal{R}$ is *compatible* with an order $\succ$ on terms if rules are *oriented* from left to right: $l \succ r$ for all rules $l \to r \in \mathcal{R}$.

Reduction orders characterise termination of rewrite systems in the following sense.

Proposition 2.50. *A TRS $\mathcal{R}$ is terminating if and only if there exists a reduction order $\succ$ which is compatible with $\mathcal{R}$.*

Proof.

- $\Rightarrow$ For a terminating TRS $\mathcal{R}$, the relation $\to_{\mathcal{R}}^{+}$ gives a reduction order compatible with $\mathcal{R}$.
- $\Leftarrow$ Suppose $\succ$ is a reduction order compatible with $\mathcal{R}$. Then by closure under context and substitutions we have $\to_{\mathcal{R}} \subseteq \succ$. Hence an infinite rewrite sequence
$$t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots ,$$
translates to an infinite descent
$$t_0 \succ t_1 \succ t_2 \succ \cdots ,$$
contradicting that $\succ$ is well-founded. □

Consider a TRS $\mathcal{R}$ compatible with a reduction order $\succ$. The proof illustrates that the maximal length of derivations is bounded by the maximal length of a $\succ$ descending sequence. Hence such orders can in principle be used to assess the complexity of a rewrite system.

Recursive Path Orders

Recursive path orders (*RPOs* for short) [29, 47] are in particular interesting for automation, as they provide a purely syntactic termination criterion.

Definition 2.51. A *quasi-precedence* $\succsim$ (or simply *precedence*) is a quasi-order on $\mathcal{F}$.

We extend equivalence $\sim$ underlying a precedence $\succsim$ to terms by identifying equivalent symbols.

Definition 2.52. Let $\sim$ be an equivalence on $\mathcal{F}$. We define *term equivalence* $\approx$ induced by $\sim$ inductively as follows: $s \approx t$ if one of the following alternatives hold:

(1) $s = t$; or

(2) $s = f(s_1, \ldots, s_k)$, $t = g(t_1, \ldots, t_l)$ with $f \sim g$ and $s_i \approx t_i$ holds for all $i = 1, \ldots, k$.

The following definition of RPO is taken from Steinbach [73], and uses a dedicated status function τ.

Definition 2.53. A *status function* on $\mathcal{F}$ is a function $\tau : \mathcal{F} \to \{\mathsf{mul}, \mathsf{lex}\}$. If $\tau(f) = \mathsf{mul}$ we say that f has *multiset status*, for $\tau(f) = \mathsf{lex}$ we say that f has *lexicographic status*. The status function *agrees* with a precedence $\succsim$ if $f \sim g$ implies $\tau(f) = \tau(g)$ for all $f, g \in \mathcal{F}$.

Definition 2.54 (Recursive Path Order)**.** Let $\succsim$ denote a quasi-precedence on $\mathcal{F}$, with underlying proper order $\succ$ and equivalence $\sim$, and let τ denote a status function on $\mathcal{F}$. Then $s \succ_{\mathsf{rpo},\tau} t$ for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $s = f(s_1, \ldots, s_k)$ if one of the following alternatives hold.

(1) $s_i \succsim_{\mathsf{rpo},\tau} t$ for some $i \in \{1, \ldots, k\}$.

(2) $t = g(t_1, \ldots, t_l)$, $s \succ_{\mathsf{rpo},\tau} t_j$ for all $j = 1, \ldots, l$ and either
- $f \succ g$; or
- $f \sim g$ and $\langle s_1, \ldots, s_k\rangle \succ^{\tau(f)}_{\mathsf{rpo},\tau} \langle t_1, \ldots, t_l\rangle$.

Here $s \succsim_{\mathsf{rpo},\tau} t$ denotes that either $s \approx t$ or $s \succ_{\mathsf{rpo},\tau} t$ holds. In the last clause, $\succ^{\tau(f)}_{\mathsf{rpo},\tau}$ is used for the extension of $\succ_{\mathsf{rpo},\tau}$ to sequences as given by the status $\tau(f)$. We define

- $\langle s_1, \ldots, s_k\rangle \succ^{\mathsf{mul}}_{\mathsf{rpo},\tau} \langle t_1, \ldots, t_k\rangle$ if $\{\!\{s_1, \ldots, s_k\}\!\}\ (\succ_{\mathsf{rpo},\tau})^{\mathsf{mul}}\ \{\!\{t_1, \ldots, t_k\}\!\}$ holds; and
- $\langle s_1, \ldots, s_k\rangle \succ^{\mathsf{lex}}_{\mathsf{rpo},\tau} \langle t_1, \ldots, t_l\rangle$ if there exists $j \in \{1, \ldots, \min(k, l)\}$ such that
  - $s_i \approx t_i$ holds for all $i < j$;
  - and $s_j \succ_{\mathsf{rpo},\tau} t_j$ holds.

When every symbol in $\mathcal{F}$ admits a lexicographic status, i.e., $\tau(f) = \mathsf{lex}$ for all $f \in \mathcal{F}$, we denote the order $\succ_{\mathsf{rpo},\tau}$ also by $\succ_{\mathsf{lpo}}$ and call $\succ_{\mathsf{lpo}}$ a *lexicographic path order* (*LPO* for short). Conversely, when every symbol admits a multiset status we denote the order $\succ_{\mathsf{rpo},\tau}$ by $\succ_{\mathsf{mpo}}$ and call $\succ_{\mathsf{mpo}}$ a *multiset path order* (MPO for short). In case the precedence $\succsim$ is a proper order, $\succ_{\mathsf{lpo}}$ corresponds to Kamin and Levy's lexicographic path order [47], and $\succ_{\mathsf{mpo}}$ to the multiset path order of Dershowitz and Manna [29]. It is not difficult to prove that $\succ_{\mathsf{rpo},\tau}$ is a rewrite order. Whenever the strict part $\succ$ underlying the precedence $\succsim$ is well-founded, $\succ_{\mathsf{rpo},\tau}$ is well-founded and constitutes thus a reduction order, provided the signature is non-variadic. The following proposition forms a special case of [33, Theorem 4.38].

Proposition 2.55. *Let $\succsim$ denote a quasi-precedence on a finite but possibly variadic signature $\mathcal{F}$, and let τ denote a status function on $\mathcal{F}$ that agrees with $\succsim$.*

(1) *$\succ_{\mathsf{rpo},\tau}$ is a rewrite order; and*

(2) *suppose that whenever $f \in \mathcal{F}$ is variadic then $\tau(f) = \mathsf{mul}$. Then $\succ_{\mathsf{rpo},\tau}$ is well-founded on $\mathcal{T}(\mathcal{F}, \mathcal{V})$.*

Corollary 2.56. *Suppose that $\mathcal{R}$ is compatible with an instance of $\succ_{\mathsf{rpo},\tau}$ such that τ and $\gtrsim$ satisfy the side-conditions of Proposition 2.55. Then $\mathcal{R}$ is terminating.*

If the above corollary applies we also say that $\mathcal{R}$ is $\succ_{\mathsf{rpo},\tau}$ *terminating.* A program as first suggested by Hofbauer and Lautemann is to classify the strength of such orders based on the bound *induced* by a compatible reduction order on the derivation height of terms. Following results are due to Hofbauer and Weiermann respectively.

Proposition 2.57 ([41] and [78])**.** *Let $\mathcal{R}$ denote a TRS.*

(1) *If $\mathcal{R}$ is terminating by $\succ_{\mathsf{mpo}}$ then $\mathrm{dc}_{\mathcal{R}}(n)$ is bounded by a primitive recursive function.*

(2) *If $\mathcal{R}$ is terminating by $\succ_{\mathsf{lpo}}$ then $\mathrm{dc}_{\mathcal{R}}(n)$ is bounded by a multiple recursive function.*

For the notion of primitive and multiple recursive function we kindly refer the reader to a standard text book on recursion theory, for instance [68]. Important for our concern is that the bounding functions can grow *very fast*, and that both results are optimal in the sense that in general the established bounds are tight.

The Interpretation Method

Another popular instance of reduction order is given by the *interpretation* of terms into a carrier A equipped with a well founded order $\succ$. Termination of a TRS $\mathcal{R}$ is established if the provided interpretation embeds the rewrite relation $\to_{\mathcal{R}}$ into $\succ$.

Definition 2.58 ($\boldsymbol{\mathcal{F}}$-algebra)**.** An *$\mathcal{F}$-algebra* $\mathcal{A}$ for a signature $\mathcal{F}$ is a set A equipped with operations $f_{\mathcal{A}} : A^n \to A$ for every n-ary function symbol $f \in \mathcal{F}$. The set A is called the carrier of $\mathcal{A}$. A mapping from $\mathcal{V}$ to A is called an *assignment* (into A). We inductively define the *interpretation* of a term t under assignment α as follows:

$$[\alpha]_{\mathcal{A}}(t) := \begin{cases} \alpha(t) & \text{if } t \text{ is a variable} \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_k)) & \text{if } t = f(t_1, \ldots, t_k). \end{cases}$$

Definition 2.59 (Monotone $\boldsymbol{\mathcal{F}}$-algebra)**.**

(1) A *monotone $\mathcal{F}$-algebra* $(\mathcal{A}, \succ)$ consists of an $\mathcal{F}$-algebra $\mathcal{A}$ and a proper order $\succ$ on the carrier A of $\mathcal{A}$ such that every operation $f_{\mathcal{A}} : A^n \to A$ in $\mathcal{A}$ is monotone in all its coordinates with respect to $\succ$, that is, for every $f \in \mathcal{F}$ we have

$$f_{\mathcal{A}}(a_1, \ldots, a_i, \ldots, a_k) \succ f_{\mathcal{A}}(a_1, \ldots, b, \ldots, a_k) \,,$$

for all $i = 1, \ldots, k$ and $a_1, \ldots, a_k, b \in A$, whenever $a_i \succ b$.

We call the $\mathcal{F}$-algebra $(\mathcal{A}, \succ)$ *well-founded* if $\succ$ is well-founded.

(2) For a monotone $\mathcal{F}$-algebra $(\mathcal{A}, \succ)$ we define the relation $\succ_{\mathcal{A}}$ on terms such that

$$s \succ_{\mathcal{A}} t \quad :\Longleftrightarrow \quad [\alpha]_{\mathcal{A}}(s) \succ [\alpha]_{\mathcal{A}}(s) \ .$$

(3) We say that a TRS $\mathcal{R}$ is *compatible* with a monotone $\mathcal{F}$-algebra $(\mathcal{A}, \succ)$ if it is compatible with $\succ_{\mathcal{A}}$.

Note that for any monotone $\mathcal{F}$-algebra $(\mathcal{A}, \succ)$, the order $\succ_{\mathcal{A}}$ gives a rewrite relation, hence a reduction order if $\succ_{\mathcal{A}}$ is well-founded.

Proposition 2.60 ([16])**.** *A TRS $\mathcal{R}$ is terminating if and only if there exists a well-founded monotone $\mathcal{F}$-algebra which is compatible with $\mathcal{R}$.*

For proving termination of rewrite systems with the *interpretation method* as given by the proposition, one often resorts to *polynomial interpretations* [51, 26].

Definition 2.61 (Polynomial Interpretation)**.** A monotone $\mathcal{F}$-algebra $(\mathcal{A}, \succ)$ is called a *polynomial interpretation* if

- the carrier of $\mathcal{A}$ is $\mathbb{N}$; and
- the proper order $\succ$ is the standard order $>$ on $\mathbb{N}$; and
- the operations $f_{\mathcal{A}}$ are polynomial functions.

We abbreviate a polynomial interpretation $(\mathcal{A}, >)$ by $\mathcal{A}$. No confusion can arise from this as $>$ is fixed. As for recursive path orders, the polynomial interpretation method is restricted. This can be shown for instance by analysing the induced derivational complexity. The following proposition is due to Lautemann and Hofbauer.

Proposition 2.62 ([42])**.** *If $\mathcal{R}$ is compatible with a polynomial interpretation, then $\mathrm{dc}_{\mathcal{R}}(n)$ is bounded by a function in $2^{2^{\mathcal{O}(n)}}$.*

More recently, also *matrix interpretations* are used to define termination orders, see [32] but also [43].

Definition 2.63 (Matrix Interpretation)**.** We call a monotone $\mathcal{F}$-algebra $(\mathcal{A}, \succ)$ a *matrix interpretation* if

(1) the carrier of $\mathcal{A}$ is $\mathbb{N}^d$;

(2) the proper order compares *vectors*

$$(x_1, x_2, \ldots, x_d) \succ (y_1, y_2, \ldots, y_d) :\Leftrightarrow x_1 > x_2 \wedge x_2 \geqslant y_2 \wedge \cdots \wedge x_d \geqslant x_d \ ,$$

for the standard order $>$ on $\mathbb{N}$; and

(3) the operations $f_{\mathcal{A}}$ are interpreted by linear functions of the form

$$f_{\mathcal{A}} : \ (\vec{x}_1, \ldots, \vec{x}_k) \mapsto F_1 \cdot \vec{x}_1 + \cdots + F_k \cdot \vec{x}_k + f \ ,$$

for (column) vectors of variables $\vec{x}_1, \ldots, \vec{x}_k$, matrices F_i $(i = 1, \ldots, k)$ of size $d \times d$ and f a vector over $\mathbb{N}$. Moreover, for any i $(i \in \{1, \ldots, k\})$ the top left entry $(F_i)_{1,1}$ is positive.

Part I.

Closing the Gap

Chapter 3.

Introduction

Term rewriting forms an abstract model of computation that is Turing complete [16]. More precisely, any function computed by a Turing machine can be computed by a rewrite system, under the semantics imposed in Definition 2.39. The runtime complexity of a rewrite system forms a *natural* cost model in this setting. One may wonder however, if the runtime complexity of a TRS $\mathcal{R}$ is really related to the *intrinsic complexity* of the problem solved by $\mathcal{R}$. Van Embde Boas articulated in his *invariance thesis* [22] that a time cost model is *reasonable* if it is polynomially related to the standard notion of time on a conventional abstract model of computation, the Turing machine. The main result of this part, the *invariance theorems for rewriting* (Theorem 7.2 and Theorem 7.5), confirm that the runtime complexity forms a reasonable cost model for rewriting.

Although effectively computable, a single rewrite step is not an atomic operation. In particular, a single rewrite step can involve the duplication of arbitrary large objects. As a consequence, the runtime complexity is a priori not an invariant cost model. Consider the following example.

Example 3.1. The orthogonal TRS $\mathcal{R}_{\mathsf{btree}}$ is given by the following rules:

$$\mathsf{btree}(n) \to \mathsf{f}(n, \mathsf{leaf}) \qquad \mathsf{f}(0, t) \to t \qquad \mathsf{f}(\mathsf{s}(n), t) \to \mathsf{f}(n, \mathsf{c}(t, t)) \; .$$

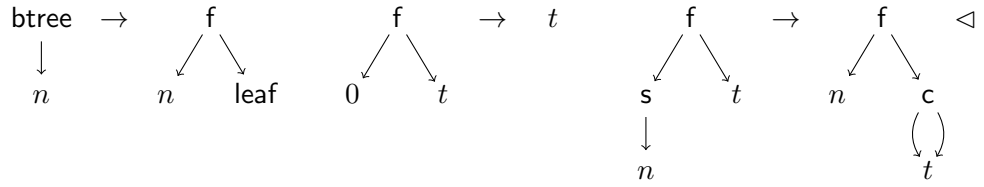
The runtime complexity of $\mathcal{R}_{\mathsf{btree}}$ is linear, since the size of the first argument decreases in each recursion step. Let $\mathbf{n}$ denote the representation of $n \in \mathbb{N}$ as numeral build from the constructors 0 and s. The function $[\![\mathsf{btree}]\!]_{\mathcal{R}_{\mathsf{btree}}}$ produces on input $\mathbf{n}$ a binary tree of height n, i.e., a result whose size is exponential in the size of the input. ◁

This final tree cannot even be written down in polynomial time. At first sight there appears to be a *gap* between the runtime complexity of $\mathcal{R}_{\mathsf{btree}}$, and the actual cost of an implementation of the function computed by $\mathcal{R}_{\mathsf{btree}}$. And if one sticks to an explicit, linear representation of terms, then indeed this gap persists. In our understanding, the issue illustrated by $\mathcal{R}_{\mathsf{btree}}$ is only a *representation* problem. To close this gap, we resort to a compact encoding of terms which allows us to take *sharing* into account.

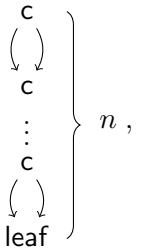
To this end, we employ *term graph rewriting* (*graph rewriting* for short) as an intermediate machinery. This is common practice in the implementation of term rewriting languages [69]. A *graph rewrite system* (*GRS* for short) is like a term rewrite system, but operates on *term graphs*. Term graphs can be understood as terms, where sharing of common sub-terms is allowed. Duplication

is always resolved by sharing in this setting. To illustrate this, the TRS $\mathcal{R}_{\mathsf{btree}}$ is formulated as the following GRS $\mathcal{G}_{\mathsf{btree}}$.

Example 3.2. The GRS $\mathcal{G}_{\mathsf{btree}}$ consists of the following three rules:



Apart from the representation of left- and right-hand sides as term graphs, the GRS $\mathcal{G}_{\mathsf{btree}}$ differs from the TRS $\mathcal{R}_{\mathsf{btree}}$ only in the treatment of the variable t in the right-hand side of the last rule. Whereas the corresponding rewrite rule duplicates the tree given as t, the GRS $\mathcal{G}_{\mathsf{btree}}$ introduces two pointers instead. Reducing the term graph that corresponds to $\mathsf{btree}(\mathbf{n})$ yields the term graph



which encodes exactly the normal form of $\mathsf{btree}(\mathbf{n})$. The $\mathcal{G}_{\mathsf{btree}}$ simulates $\mathcal{R}_{\mathsf{btree}}$ in a step wise manner. In this sense, the GRS $\mathcal{G}_{\mathsf{btree}}$ provides a *sound* implementation for $\mathcal{R}_{\mathsf{btree}}$. This implementation is also effective. A single graph rewrite step is effectively computable. Moreover, the output and intermediate graphs are sufficiently small.

In general, plain graph rewriting does not yield a *complete* implementation of term rewriting. On the one hand, sharing common sub-terms disallows certain term rewriting sequences, namely those where the shared sub-terms are rewritten differently. On the other hand, graph rewriting employs a fine grained matching mechanism that takes the sharing structure into account. In particular, the matching mechanism of graph rewriting employs a form of pointer equality, whereas matching in term rewriting relies on structural equality.

Completeness can be recovered by adding mechanisms for *folding* (also known as *collapsing*) and *unfolding* (also known as *copying*) to the graph rewrite relation. This has been observed quite early, see for instance Plump's survey [66]. Graph rewriting is often propagated as an efficient implementation of term rewriting. Surprisingly, this correspondence has never been analysed from a complexity related perspective.

As a first step towards an effective implementation of term rewriting, we re-investigate *adequacy* of graph rewriting for term rewriting. To avoid ineffectiveness of the implementation, we define *small step approximations* of folding

and unfolding. Our *adequacy theorem* states that extending the graph rewriting relation by these operations results in a sound and complete implementation of term rewriting. As a second step, we show that this implementation is effective. More precise, the cost of a reduction sequence on graphs is polynomially related to the size of the start graph and the length of reductions.

Putting things together, we establish a polynomial relationship between the runtime complexity of a TRS $\mathcal{R}$, and the cost of $\mathcal{R}$-reductions on graphs (Theorem 7.1). The provided implementation witnesses that the intrinsic computational complexity of a function computed by $\mathcal{R}$ is polynomially related to the runtime complexity of $\mathcal{R}$ (Theorem 7.2). Noteworthy, non of our reasoning relies on any particular evaluation strategy. This allows us to capture non-deterministic computations, as defined by non-confluent TRSs (Theorem 7.5).

Related Work. Adequacy of graph rewriting has been studied under different aspects in the literature, cf. Plump's [66] survey for an introduction to the topic. Nowadays it is also treated in standard textbooks on rewriting [75, 64]. Here we only mention the paper by Barendregt et al. [19], which gives a first account on a sound and complete implementation of term rewriting through graph rewriting. We essentially follow the notion of graph rewriting as defined there. Our notion of adequacy is taken from Kenneway et al. [48], where graph rewriting is shown to provide an adequate implementation of orthogonal term rewrite systems (even in the infinitary setting).

This work is closely related to the work of Dal Lago and Martini [28], and Accattoli and Dal Lago [1] on the invariance of the unitary cost model for the λ-calculus. As in rewriting, a single reduction step in the λ-calculus can duplicate arbitrary data. Unsurprisingly, essentially the same approach is used to tackle this problem, namely, sharing of common sub-expressions. In [1] λ-calculus under head reduction semantics is investigated. Sharing is integrated by means of an explicit substitution calculus. In [28], λ-reductions under weak semantics, i.e., where reduction can take place only outside of lambdas, are implemented by orthogonal term rewrite systems in a step-wise manner. As a by-product, the invariance of the unitary cost measure for orthogonal rewrite systems is proven. Similar to here, this is done by way of a graph rewriting implementation. Compare also [27] where this invariance result is presented independently, for innermost and outermost rewriting. We extend upon this result. Our invariance theorems require neither a restriction on the rewrite system, nor any particular evaluation strategy.

Outline. This part presents a unified account of the authors contributions on work published together with Moser [13, 12]. In [13] we have studied adequacy with respect to innermost rewriting, which requires collapsing in the general case. This result has been extended to rewriting without imposing a rewriting strategy in [12]. The latter paper is essentially an extension of [13] that also integrates collapsing.

In the next chapter we introduce notions and notations related to term graph rewriting. Also, we show that the employed formalisation of graph rewriting

provides the basis for a *sound* implementation of term rewriting. In Chapter 5 we make precise our notion of adequacy of graph rewriting for term rewriting. We introduce *restricted folding* and *unfolding* relations. By integrating these into the standard graph rewrite relation, we obtain our adequacy theorem for full rewriting. We also present a refined adequacy theorem for innermost rewriting. Noteworthy, this theorem does not rely on unfolding, which we exploit in the actual implementation. In Chapter 6 we then show that the graph reductions employed in the adequacy theorems can be effectively implemented.

In Chapter 7 we finally put things together. Theorem 7.2 provides our deterministic invariance theorem for complete TRSs, in Theorem 7.5 we formulate the invariance theorem for the general case.

Chapter 4.

Term Graph Rewriting

4.1. Term Graphs

We introduce the central concepts and notions of term graph rewriting used here. See the survey of Plump [66] for an introduction, or [75] which is notationally closer to the present work. Plump represents term graphs based on *hypergraphs*. In contrast, we mostly follow the notion of term graphs from Barendregt et al, [19] which is rooted in the notion of graph rewriting due to Staples [72]. Here directed graphs, with an implicitly imposed order on outgoing edges, are used to denote terms. This order allows us to distinguish graphs representations of terms with permuted arguments, like $\mathsf{f}(\mathsf{a},\mathsf{b})$ and $\mathsf{f}(\mathsf{b},\mathsf{a})$.

Definition 4.1 (Directed and Ordered Graph)**.** Let $\mathcal{N}$ be a countable infinite set of *nodes*, and let $\mathcal{L}$ denote a set of *labels*. A *directed and ordered graph* G with nodes in $\mathcal{N}$ and labels in $\mathcal{L}$ is a triple $(N_G, \mathsf{succ}_G, \mathsf{lab}_G)$ involving:

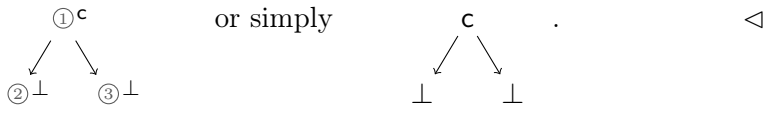
(1) a finite set $N_G \subseteq \mathcal{N}$, the *nodes* of $\mathcal{G}$; and

(2) a function $\mathsf{succ} : N_G \to N_G^*$, associating with each node an *ordered* sequence of *successors*; and

(3) a function $\mathsf{lab}_G : N_G \to \mathcal{L}$ that associates with each node its *label*.

For brevity we call a directed and ordered graph G simply graph. Typically the set of labels $\mathcal{L}$ is clear from context and not explicitly mentioned. If not mentioned otherwise, we denote by G, H graphs, nodes are denoted by u, v, w, possibly extended by subscripts. We also write $u \in G$ instead of $u \in N_G$. When we draw graphs, a directed edge will go from u to each node in $\mathsf{succ}(u)$, where the left-to-right ordering of the source of the edge will correspond to the ordering in $\mathsf{succ}(u)$. We indicate the label $\mathsf{lab}(u)$ to the right of the node u. If the node itself is unimportant, we simply write the label $\mathsf{lab}(u)$ instead u.

Example 4.2. Consider the graph $G_1 = (N_{G_1}, \mathsf{succ}_{G_1}, \mathsf{lab}_{G_1})$ with labels $\mathcal{L} = \{\mathsf{c}, \bot\}$ where $N_{G_1} = \{①, ②, ③\}$, successors are given as $\mathsf{succ}_{G_1}(①) = [②, ③]$ and $\mathsf{succ}_{G_1}(②) = \mathsf{succ}_{G_1}(③) = [\,]$, and labels are given as $\mathsf{lab}_{G_1}(①) = \mathsf{c}$ and $\mathsf{lab}_{G_1}(②) = \mathsf{lab}_{G_1}(③) = \bot$. Then G_1 is drawn as



. ◁

The following definition introduces standard notions on graphs.

Definition 4.3 (Size, Path, Rooted, Acyclic)**.** Let $G = (N_G, \mathsf{succ}_G, \mathsf{lab}_G)$ denote a graph.

(1) The *size* $|G|$ of G refers to the number of nodes in G, i.e., $|G| := N_G$;

(2) Consider $u \in G$ with $\mathsf{succ}_G(u) = [u_1, \ldots, u_k]$. For $i = 1, \ldots, k$, we set $\mathsf{succ}^i_G(u) := u_i$ and call u_i the i^{th} *successor* of u. When it is more convenient, we also write $u \xrightarrow{i}_G u_i$.

(3) For $u, v \in G$, we define $u \rightharpoonup_G v$ if for some i, $u \xrightarrow{i}_G v$ holds and say that there is an *edge* from u to v in G. A non-empty sequence of nodes $u_1, \ldots, u_{n+1}$ $(n \in \mathbb{N})$ with

$$u_1 \rightharpoonup_G \cdots \rightharpoonup_G u_{n+1} \ ,$$

is called a *path* from u_1 to u_{n+1} in G (of *length* n). We also say that u_{n+1} is *reachable* from u_1 in G.

(4) The graph G is called *rooted* if there exists a *unique* node u such that every other node in G is reachable from u. This unique node u is called the *root* of G and denoted by $\mathrm{rt}(G)$.

(5) A graph G is called *cyclic* if $u \rightharpoonup^+_G u$ holds for some node $u \in G$, otherwise G is called *acyclic*.

In the following, we usually consider only rooted and acyclic graphs. Following notions apply to such graphs.

Definition 4.4 (Depth, Below, Above)**.** Let $G = (N_G, \mathsf{succ}_G, \mathsf{lab}_G)$ denote a rooted and acyclic graph.

(1) The *depth* $\mathsf{dp}(G)$ of G is the length of the longest path in G. Necessarily this path starts in $\mathrm{rt}(G)$.

(2) Suppose there is a path from $u \in G$ to $v \in G$. Then u is also called *above* v, and conversely v is called *below* u, in S. If $u \neq v$ then we also call u *strictly above* v, and v *strictly below* u respectively.

Definition 4.5 (Sub-Graph, Graph Union, Redirection)**.** Consider two graphs $G = (N_G, \mathsf{succ}_G, \mathsf{lab}_G)$ and $H = (N_H, \mathsf{succ}_H, \mathsf{lab}_H)$.

(1) We write $G{\restriction_u}$ for the *sub-graph* of G reachable from $u \in G$, i.e., $G{\restriction_u} := (N', \mathsf{succ}', \mathsf{lab}')$ where $N' = \{v \mid u \rightharpoonup^+_G v\}$ and succ', lab' are succ and lab restricted to the domain N'.

(2) Let $u, v \in G$ be two distinct nodes of G. We denote by $G[v \leftarrow u]$ the graph obtained by *redirecting* all edges going to u to the node v. More precise

$G[v \leftarrow u] := (N_{G[v\leftarrow u]}, \mathsf{succ}_{G[v\leftarrow u]}, \mathsf{lab}_{G[v\leftarrow u]})$ where

$$\begin{aligned} N_{G[v\leftarrow u]} &:= N_G \\ \mathsf{succ}^i_{G[v\leftarrow u]}(w) &:= \begin{cases} v & \text{if } w = u, \\ w & \text{otherwise.} \end{cases} \\ \mathsf{lab}_{G[v\leftarrow u]}(w) &:= \mathsf{lab}_G(w)\ . \end{aligned}$$

For a sequence of nodes $\vec{u} = u_1, \ldots, u_n \in G$ and $\vec{v} = v_1, \ldots, v_n \in G$ we use $G[\vec{v} \leftarrow \vec{u}]$ as an abbreviation for $((G[v_1 \leftarrow u_1])[v_2 \leftarrow u_2] \ldots)[v_n \leftarrow u_n]$.

(3) We denote by $G \oplus H$ the (left-biased) *union* of G and H, defined by

$$G \oplus H := (N_G \cup N_H, \mathsf{succ}_G \oplus \mathsf{succ}_H, \mathsf{lab}_G \oplus \mathsf{lab}_H)\ .$$

Here, for $f \in \{\mathsf{succ}, \mathsf{lab}\}$, $f_G \oplus f_H$ is given by

$$(f_G \oplus f_H)(u) := \begin{cases} f_G(u) & \text{if } u \in N_G, \text{ and} \\ f_H(u) & \text{if } u \in N_H. \end{cases}$$

Definition 4.6 (Term Graph)**.** Let $\mathcal{F}$ denote a signature and let $\mathcal{V}$ be a set of variables. A *term graph* over $\mathcal{F}$ and $\mathcal{V}$ is an *acyclic* and *rooted* graph $S = (N, \mathsf{succ}, \mathsf{lab})$ with labels $\mathcal{F} \cup \mathcal{V}$ that satisfies for all nodes $u \in S$ the following conditions:

(1) if $\mathsf{lab}(u) \in \mathcal{F}$ is a k-ary function symbol then $\mathsf{succ}(u) = [u_1, \ldots, u_k]$,

(2) if $\mathsf{lab}(u) \in \mathcal{V}$ then $\mathsf{succ}(u) = [\,]$.

The *set of all term graphs* over $\mathcal{F}$ and $\mathcal{V}$ with nodes in $\mathcal{N}$ is denoted by $\mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$.

Usually the set of nodes is not important, in this case we also write $\mathcal{G}(\mathcal{F}, \mathcal{V})$ for $\mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$. Below $S, T, \ldots$ and L, R, possibly followed by subscripts, always denote term graphs.

In Barendregt et al. [19], a partial labeling function with codomain in $\mathcal{F}$ is used, variables are expressed by unlabeled nodes. In contrast, we use a total labeling function. This allows us to define the *unfolding* of a graph to a term as follows. Note that since term graphs are acyclic, the unfolding is well-defined.

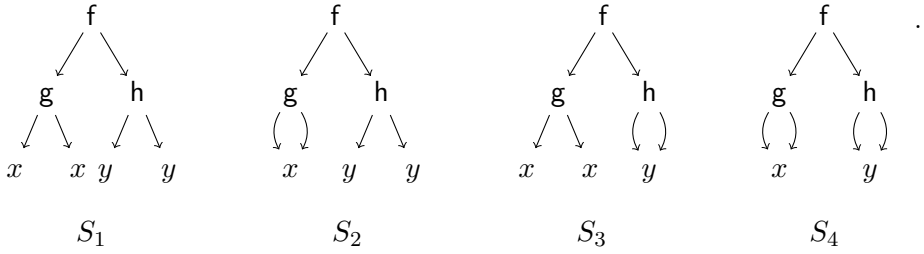
Definition 4.7 (Unfolding)**.** We define the function $\mathsf{U} : \mathcal{G}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ as follows.

$$\mathsf{U}(S) := \begin{cases} x & \text{if } \mathsf{lab}(\mathrm{rt}(S)) = x \in \mathcal{V} \\ f(\mathsf{U}(S{\restriction_{u_1}}), \ldots, \mathsf{U}(S{\restriction_{u_k}})) & \text{if } \mathsf{lab}(\mathrm{rt}(S)) = f \in \mathcal{F}. \end{cases}$$

Here we suppose $\mathsf{succ}(\mathrm{rt}(S)) = [u_1, \ldots, u_k]$. We say that the term graph S *unfolds* to the term $\mathsf{U}(S)$, or conversely that $\mathsf{U}(S)$ is the *unfolding* of S.

Although unfolding is surjective, due to sharing it is not injective in general.

Example 4.8. Following four term graphs S_1, S_2, S_3 and S_4, all unfold to the term $s = \mathsf{f}(\mathsf{g}(x,x),\mathsf{h}(y,y))$.



The term graph S_1 exploits no sharing, similar to the term s it is simply a *tree*. The term graphs S_2 and S_3 are obtained from S_1 by collapsing two nodes, the nodes labeled by x and y respectively, to a single node. Likewise, the term graph S_4 can be obtained from S_2 and S_3 by further collapsing nodes. The term graph S_4 is *maximally shared*, no further sharing can be introduced. It constitutes thus the most compact representation of s. ◁

Let $l \in \mathcal{F} \cup \mathcal{V}$. If $\mathsf{lab}_S(u) = l$ then u is also called an *l-node* in S. This notion is extended from labels to sets of labels in the obvious way.

Definition 4.9 (Variable Nodes, Function Symbol Nodes)**.** Let $S \in \mathcal{G}(\mathcal{F},\mathcal{V})$. We denote by $\mathsf{Var}(S) \subseteq N_S$ the set of all $\mathcal{V}$-nodes in S. Dual, the set $\mathsf{Fun}(S)$ denotes the set of all $\mathcal{F}$-nodes in S.

On term graphs, the following induction principles are justified.

Definition 4.10 (Structural Induction on Term Graphs)**.** Let S denote a term graph and let P be a predicate on nodes.

- Base Case: Show that property P holds in all $u \in \mathsf{Var}(S)$.
- Inductive Step: Show that property P holds in all $u \in \mathsf{Fun}(S)$. The induction hypothesis (IH) states that property P holds for all nodes in $v \in S$ with $u \rightharpoonup_S v$.

It follows that property P holds in all nodes $u \in S$.

Definition 4.11 (Induction on the Depth)**.** Let S denote a term graph and let P be a predicate on nodes.

- Base Case: Show that property P holds for $\mathrm{rt}(S)$.
- Inductive Step: Show that property P holds in all $u \in S$ with $u \neq \mathrm{rt}(S)$. The induction hypothesis (IH) states that property P holds for all nodes in v with $v \rightharpoonup_S u$.

It follows that property P holds in all nodes $u \in S$.

4.1.1. Term Graph Morphisms

In the literature [66, 19, 17], *sharing* is usually captured in by an order $\succ$ on term graphs: $S \succ T$ if T is obtained from S by collapsing one or more nodes, or equivalently, $S \succ T$ if there exists a *term graph morphism* that embeds S into T. We follow [17] and generalise the concept of *term graph morphism* to Δ-morphisms. These allow the treatment of both *sharing* and *matching*.

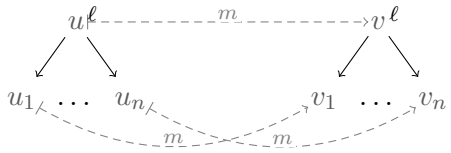
Definition 4.12 ($\boldsymbol{\Delta}$-morphism)**.** Let S and T be two term graphs, and let $\Delta \subseteq \mathcal{F} \cup \mathcal{V}$ denote a set of labels. A function $m : N_S \to N_T$ is called *morphic* in $u \in S$ if

(1) $\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u))$; and *(labeling condition)*

(2) if $u \xrightarrow{i}_S v$ then $m(u) \xrightarrow{i}_S m(v)$ for all appropriate i. *(successor condition)*

A Δ*-morphism* from S to T, denoted as $m : S \to_\Delta T$, is a mapping $m : N_S \to N_S$ that satisfies $m(\mathrm{rt}(S)) = \mathrm{rt}(T)$ and that is morphic in all nodes $u \in S$ with $\mathsf{lab}_S(u) \notin \Delta$.

When Δ is clear from context we call a Δ-morphism $m : S \to_\Delta T$ simply a *morphism*. By $\underline{m} : \mathcal{N} \to \mathcal{N}$ we denote the extension of $m : S \to_\Delta T$ to all nodes $\mathcal{N}$, given by $\underline{m}(u) := m(u)$ if $u \in S$, and $\underline{m}(u) := u$ otherwise. The morphism condition on node u can be visualised as follows.

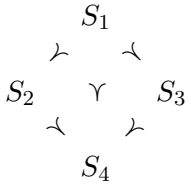


Sharing is addressed by $\varnothing$-morphisms as follows.

Definition 4.13 (Folding, Unfolding)**.** We define $S \succcurlyeq T$ if there exists a morphism $m : S \to_\varnothing T$. If $S \succcurlyeq T$ holds we also say that S *folds* (or *collapses*) to T, dual, we say that T *unfolds* to S. If $S \succcurlyeq T$ and $T \succcurlyeq S$ holds then S and T are *isomorphic*, in notation $S \cong T$. We define $S \succ T$ if $S \succcurlyeq T$ and not $S \cong T$.

We sometimes also write $S \succcurlyeq_m T$, $S \succ_m T$ or $S \cong^m T$ to indicate the underlying morphism $m : S \to_\varnothing T$. We denote by $\preccurlyeq$ and $\prec$ the inverse of $\succcurlyeq$ and $\succ$ respectively.

Example 4.14 (Continued from Example 4.8). Reconsider the term graphs S_1, S_2, S_3 and S_4 depicted in Example 4.8. We have



Note that S_2 and S_3 are incomparable, that is, neither $S_2 \succcurlyeq S_3$ nor $S_3 \succcurlyeq S_2$ holds. $\lhd$

Throughout the following, we will tacitly assume that the relation $\succcurlyeq$ enjoys the following properties.

Lemma 4.15. *Let S and T be term graphs.*

(1) *If $S \succcurlyeq_m T$ then m is surjective.*

(2) *If $S \cong^m T$ then m is bijective.*

(3) *If $S \succ_m T$ then m is not injective.*

Proof. For the first assertion one shows that every $u \in S$ is in the domain of m, by induction on the depth of S. In the base case $m(\mathrm{rt}(S)) = \mathrm{rt}(T)$, the inductive step follows directly from the successor condition.

For the second assertion, consider $S \succcurlyeq_{m_1} T$ and $T \succcurlyeq_{m_2} S$. By (1) both functions $m_1 : N_S \to N_T$ and $m_2 : N_T \to N_S$ are surjective. It thus suffices to show that m_1 and m_2 are inverses of each other. By induction on the depth of S we prove $m_2(m_1(u)) = u$ for all nodes $u \in S$. For $u = \mathrm{rt}(S)$ this is immediate from the definition. Consider the inductive step where $u \xrightarrow{i}_S v$. Hence $u = m_2(m_1(u)) \xrightarrow{i}_S m_2(m_1(v))$ by induction hypothesis and successor conditions of m_1 and m_2. Since the i^{th} successor v of u is unique, we conclude $v = m_2(m_1(v))$.

The third assertion follows from the former two. $\square$

As a consequence of the next lemma, $\succ$ defines a proper and $\cong$ an equivalence on $\mathcal{G}(\mathcal{F}, \mathcal{V})$.

Lemma 4.16. *The relation $\succcurlyeq$ is a preorder on $\mathcal{G}(\mathcal{F}, \mathcal{V})$.*

Proof. Obviously $\succcurlyeq$ is reflexive using the identity morphism $id_S : S \to_\varnothing S$. To show transitivity, consider term graphs S, T and U such that $S \succcurlyeq_{m_1} T$ and $T \succcurlyeq_{m_2} U$. Set $m := m_2 \circ m_1$ and let $u \in S$. Observe

$$\begin{aligned} \mathsf{lab}_S(u) &= \mathsf{lab}_T(m_1(u)) && \text{(labeling condition on } m_1\text{)} \\ &= \mathsf{lab}_U(m_2(m_1(u))) && \text{(labeling condition on } m_2\text{)} \\ &= \mathsf{lab}_U(m(u)) \end{aligned}$$

and thus m enjoys the labeling condition in u. Suppose now $u \xrightarrow{i}_S v$ for some $i \in \mathbb{N}$ node $v \in S$. Then the successor conditions of m_1 on u give $m_1(u) \xrightarrow{i}_T m_1(v)$ similar we obtain $m_2(m_1(u)) \xrightarrow{i}_U m_2(m_1(v))$. So m enjoys the successor condition on u, in total we obtain that m is morphic in all $u \in S$. As further $m(\mathrm{rt}(S)) = m_2(m_1(\mathrm{rt}(S))) = m_2(\mathrm{rt}(T)) = \mathrm{rt}(U)$ by the assumptions, we conclude $S \succcurlyeq_m U$. $\square$

The morphism conditions underlying $\succcurlyeq$ ensure that comparable term graphs are equal up to sharing, in the sense that they denote the same term.

Lemma 4.17. *If $S \succcurlyeq_m T$ then $\mathsf{U}(S) = \mathsf{U}(T)$ holds.*

Proof. The lemma follows by structural induction on S.

4.1.2. Positions and Sharing

We introduce *positions* in the context of term graphs.

Definition 4.18 (Positions in Term Graphs, Addressing)**.** The set of *positions* $\mathcal{P}\mathrm{os}_S(u)$ of a node $u \in S$ is defined inductively as

$$\mathcal{P}\mathrm{os}_S(u) := \begin{cases} \{\epsilon\} & \text{if } u = \mathrm{rt}(S), \text{ and} \\ \{p{\cdot}i \mid \exists v \in S.\ v \xrightarrow{i}_S u \text{ and } p \in \mathcal{P}\mathrm{os}_S(v)\} & \text{otherwise.} \end{cases}$$

The set of all positions in S is $\mathcal{P}\mathrm{os}(S) := \bigcup_{u \in S} \mathcal{P}\mathrm{os}_S(u)$. We say that the position p *addresses* the node u with $p \in \mathcal{P}\mathrm{os}_S(u)$.

Note that set $\mathcal{P}\mathrm{os}_S(u)$ contains at least one element, moreover a standard induction reveals that $\mathcal{P}\mathrm{os}_S(u)$ and $\mathcal{P}\mathrm{os}_S(v)$ are disjoint for all nodes $u \neq v$ in S. As a consequence, the node addressed by p is unique. Justified by this observation, we will use positions $p \in \mathcal{P}\mathrm{os}_S(u)$ in notation for u. For instance, when more convenient we write $S\restriction_p$ for the sub-graph $S\restriction_u$ at node u addressed by p in S. No confusion can arise from this.

Lemma 4.19. *Let $S \in \mathcal{G}(\mathcal{F}, \mathcal{V})$ and let s be the unfolding of S.*

(1) *We have $\mathcal{P}\mathrm{os}(S) = \mathcal{P}\mathrm{os}(s)$, and moreover*

(2) *for any node $u \in S$ and position $p \in \mathcal{P}\mathrm{os}_S(u)$, we have $\mathsf{U}(S\restriction_u) = s|_p$.*

Proof. The lemma follows by a straight forward induction. □

By Lemma 4.17, if $S \succ T$ holds then S and T unfold to the same term. The above lemma strengthens this observation to $\mathsf{U}(S\restriction_p) = \mathsf{U}(T\restriction_p)$ for all positions $p \in \mathcal{P}\mathrm{os}(S)$ (or $p \in \mathcal{P}\mathrm{os}(T)$).

Definition 4.20 (Shared Node, Unshared Node)**.** Let $S \in \mathcal{G}(\mathcal{F}, \mathcal{V})$. We call a node $u \in S$ *shared* if $\mathcal{P}\mathrm{os}_S(u)$ is not singleton, otherwise the node u is called *unshared*.

Consider a term graph S that unfolds to the term s. The above definition is guided by the observation that for a node $u \in S$, the sub-graph $S\restriction_u$ represents exactly the sub-terms at positions $\mathcal{P}\mathrm{os}_S(u)$ in the unfolding s. The term graph S admits the least degree of sharing when $\mathcal{P}\mathrm{os}_S(u)$ is singleton for every node $u \in S$. In this case S is simply a *tree*. The most space efficient representation of s is given by S when equal sub-terms are represented by a single node. We call such graphs *fully collapsed*. Fully collapsed term graphs are minimal in the order $\succ$ [66].

Definition 4.21 (Shared Node, Tree, Fully Collapsed)**.** Let $S \in \mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$ be a term graph.

(1) The term graph S is called a *tree* if no node $u \in S$ is shared. We denote by $\triangle_{\mathcal{N}}(\mathcal{F}, \mathcal{V}) \subseteq \mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$ the set of all trees.

(2) The term graph S is called *fully collapsed* if $\mathsf{U}(S\restriction_u) = \mathsf{U}(S\restriction_v)$ implies $u = v$ for all nodes $u, v \in S$. We denote by $\triangledown_{\mathcal{N}}(\mathcal{F}, \mathcal{V}) \subseteq \mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$ set of all fully collapsed term graphs.

(3) We say that a *term* $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *is shared* in S if there is a shared node $u \in S$ such that the sub-graph $S\restriction_u$ unfolds to t. For a set of terms $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, we denote by $\diamondsuit^T_{\mathcal{N}}(\mathcal{F}, \mathcal{V}) \subseteq \mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$ the set of term graphs sharing only terms $t \in T$.

Usually we drop the reference to $\mathcal{N}$ in $\triangle_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$, $\triangledown_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$ and $\diamondsuit^T_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$, if not important or clear from context. Notice that independent on T, $\diamondsuit^T(\mathcal{F}, \mathcal{V})$ contains all trees, i.e., $\triangle(\mathcal{F}, \mathcal{V}) \subseteq \diamondsuit^T(\mathcal{F}, \mathcal{V})$. Of particular interest later will be the set $\diamondsuit^{\mathcal{V}}(\mathcal{F}, \mathcal{V})$ and $\diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, for some TRS $\mathcal{R}$. In $\diamondsuit^{\mathcal{V}}(\mathcal{F}, \mathcal{V})$, only variables can be shared, nodes labeled by a function symbol are never shared. In $\diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, only normal forms of $\mathcal{R}$ can be shared, or dual, $\mathcal{R}$-reducible terms must be represented by nodes that are not shared.

The next lemma verifies that $\succcurlyeq$ properly accounts for sharing.

Lemma 4.22. *Let S and T be term graphs. The following two statements are equivalent.*

(1) *It holds that $S \succcurlyeq_m T$.*

(2) *The function $m : N_S \to N_T$ satisfies $\mathcal{P}\mathrm{os}_S(u) \subseteq \mathcal{P}\mathrm{os}_T(m(u))$ and $\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u))$ for all nodes $u \in S$.*

Proof. Consider first the direction from (1) to (2), suppose $S \succcurlyeq_m T$. Note that $\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u))$ follows from the labeling condition that holds on all nodes $u \in S$. We show that for all $u \in S$, if $p \in \mathcal{P}\mathrm{os}_S(u)$ then $p \in \mathcal{P}\mathrm{os}_T(m(u))$. The proof is by induction on the depth of S. The property holds for $\mathrm{rt}(S)$ since $m(\mathrm{rt}(S)) = \mathrm{rt}(T)$ and $\mathcal{P}\mathrm{os}_T(\mathrm{rt}(T)) = \{\epsilon\} = \mathcal{P}\mathrm{os}_S(\mathrm{rt}(S))$. For the inductive step, consider $u \xrightarrow{i}_S u_i$. Let $p = q{\cdot}i \in \mathcal{P}\mathrm{os}_S(u_i)$ and thus by definition $q \in \mathcal{P}\mathrm{os}_S(u)$. By induction hypothesis we have $q \in \mathcal{P}\mathrm{os}_T(m(u))$, and the successor condition on u gives $m(u) \xrightarrow{m}_T (u_i)$, hence $p \in \mathcal{P}\mathrm{os}_T(m(u_i))$.

For the direction (2) to (1) consider $m : N_S \to N_T$ satisfying the properties stated in (2). The condition that $\mathcal{P}\mathrm{os}_S(\mathrm{rt}(S)) = \{\epsilon\} \subseteq \mathcal{P}\mathrm{os}_S(m(\mathrm{rt}(S)))$ implies $m(\mathrm{rt}(S)) = \mathrm{rt}(S)$. Since by assumption m satisfies the labeling condition in all $u \in S$, $S \succcurlyeq_m T$ holds if m satisfies the successor condition in all $u \in S$. Consider $u \xrightarrow{i}_S v$ for appropriate i and node $v \in S$. Hence there is a position $p \in \mathcal{P}\mathrm{os}_S(u)$ such that $pi \in \mathcal{P}\mathrm{os}_S(v)$. By assumption $p \in \mathcal{P}\mathrm{os}_T(m(u))$ and likewise $pi \in \mathcal{P}\mathrm{os}_S(m(v))$. By definition we conclude $m(u) \xrightarrow{i}_T m(v)$. □

As a consequence, isomorphic term graphs are equal up to renaming of nodes.

Lemma 4.23. *Let S and T be term graphs. The following two statements are equivalent.*

(1) *It holds that $S \cong T$.*

(2) *The function* $m : N_S \to N_T$ *satisfies* $\mathcal{P}\mathrm{os}_S(u) = \mathcal{P}\mathrm{os}_T(m(u))$ *and* $\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u))$ *for all nodes* $u \in S$.

Proof. Suppose $S \cong^m T$. Using Lemma 4.15 we thus have $S \succcurlyeq_m T$ and $T \succcurlyeq_{m^{-1}} T$ where m^{-1} denotes the inverse of the bijection m. We conclude the lemma by applying Lemma 4.22 twice. □

4.1.3. Canonical Term Graphs

Isomorphic term graphs do not collapse to the same object. This is a mere artefact of our representation as labeled and ordered graphs. To remove this artefact, we introducing a canonical form of term graphs. This avoids reasoning up to isomorphism below. The following definition is taken from Plump [66].

Definition 4.24 (Canonical Term Graphs)**.** A term graph S is called *canonical* if $u = \mathcal{P}\mathrm{os}_S(u)$ for all nodes $u \in S$. The set of all canonical term graphs is denoted by $\mathcal{G}_{\mathcal{C}}(\mathcal{F}, \mathcal{V})$. For a term graph $S \in \mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$, we define the *canonical term graph*

$$\mathcal{C}(S) = (N_{\mathcal{C}(S)}, \mathsf{succ}_{\mathcal{C}(S)}, \mathsf{lab}_{\mathcal{C}(S)}) \,,$$

of S where

$$\begin{aligned} N_{\mathcal{C}(S)} &= \{\mathcal{P}\mathrm{os}_S(u) \mid u \in N_S\} \,, \\ \mathsf{lab}_{\mathcal{C}(S)}(\mathcal{P}\mathrm{os}_S(u)) &= \mathsf{lab}_S(u) && \text{for } u \in S, \\ \mathsf{succ}^i_{\mathcal{C}(S)}(\mathcal{P}\mathrm{os}_S(u)) &= \mathcal{P}\mathrm{os}_S(\mathsf{succ}^i_S(u)) && \text{for } u \in S \text{ and appropriate } i. \end{aligned}$$

The canonical term graph $\mathcal{C}(S)$ is well-defined as for pairwise different nodes $u, v \in S$, the set of positions in S is disjoint. We emphasise that by definition u is shared if it is not singleton. The next lemma confirms our intention.

Lemma 4.25. *Following properties hold for all term graphs* S *and* T.

(1) $\mathcal{C}(S)$ *is canonical.*

(2) $S \cong \mathcal{C}(S)$.

(3) *If* $S \cong T$ *if and only if* $\mathcal{C}(S) = \mathcal{C}(T)$.

Proof. The first assertion follows by a standard induction on the depth of S. For the second assertion, observe that the function $\mathcal{P}\mathrm{os}_S : N_S \to N_{\mathcal{C}(S)}$ satisfies $\mathcal{P}\mathrm{os}_S(u) = \mathcal{P}\mathrm{os}_{\mathcal{C}(S)}(\mathcal{P}\mathrm{os}_S(u))$ and $\mathsf{lab}_S(u) = \mathsf{lab}_{\mathcal{C}(S)}(\mathcal{P}\mathrm{os}_S(u))$. By Lemma 4.23, it thus defines an isomorphism between S and $\mathcal{C}(S)$. For the third assertion, suppose $S \cong^m T$. By Lemma 4.23 we have $\mathcal{P}\mathrm{os}_S(u) = \mathcal{P}\mathrm{os}_T(m(u))$ and $\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u))$ for all $u \in S$. A standard induction on the depth of u gives that $\mathcal{C}(S)$ and $\mathcal{C}(T)$ coincide. For the converse direction, observe that by the first assertion from the assumption $\mathcal{C}(S) = \mathcal{C}(T)$ we have $S \cong \mathcal{C}(S) = \mathcal{C}(T) \cong T$, conclusively $S \cong T$ as $\cong$ is an equivalence. □

4.2. Term Graph Rewriting Systems

We arrive at the definition of *term graph rewriting*, or *graph rewriting* for short.

Definition 4.26 (Graph Rewrite Rule and Graph Rewrite System)**.**

(1) A *graph rewrite rule* over the signature $\mathcal{F}$ and variables $\mathcal{V}$ is a triple (G, l, r) where G is a graph (with labels $\mathcal{F} \cup \mathcal{V}$) and $l, r \in G$ are nodes such that $L := G|_l \in \mathcal{G}(\mathcal{F}, \mathcal{V})$ and $R := G|_r \in \mathcal{G}(\mathcal{F}, \mathcal{V})$. Further, the following conditions have to hold.

(i) the root l of L is not a $\mathcal{V}$-node,

(ii) $\mathsf{Var}(R) \subseteq \mathsf{Var}(L)$ holds, i.e., all variable nodes of R appear also in L, and

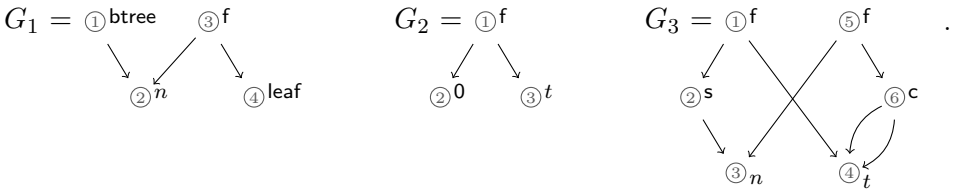
(iii) each variable is represented by a single node: if $\mathsf{lab}_G(u) = \mathsf{lab}_G(v) \in \mathcal{V}$ then $u = v$.

The term graphs L and R are called the *left-* and *right-hand* side of the rule (G, l, r). If no confusion can arise the graph rewrite rule (G, l, r) is denoted by $L \to R$.

(2) A *graph rewrite system* (*GRS* for short) $\mathcal{G}$ is a set of graph rewrite rules.

The restrictions (i) and (ii) in the definition of graph rewrite rule translate to the usual restriction on term rewrite rules. Restriction (iii) is required for the morphism based notion of matching underlying term graph rewriting. Specifically, it will assure that variables are consistently mapped to equal sub-graphs.

Example 4.27 (Continued from Example 3.2). The GRS $\mathcal{G}_{\mathsf{btree}}$ consists of the three graph rewrite rules $(G_1, ①, ③)$, $(G_2, ①, ③)$ and $(G_3, ①, ⑤)$, defined by



Using the notation $L_i \to R_i$ $(i = 1, 2, 3)$ results in the presentation of $\mathcal{G}_{\mathsf{btree}}$ given in Example 3.2. ◁

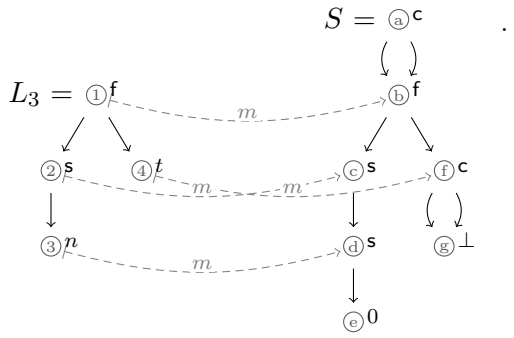
As in Bahr [17] we define a notion of *pre-reduction steps* on term graphs $\mathcal{G}(\mathcal{F}, \mathcal{V})$. The *graph rewrite relation* $\longrightarrow_{\mathcal{G}}$ is then obtained as the projection of pre-reduction steps to canonical term graphs.

Definition 4.28 (Matching Morphism and Redex)**.** Let $L \to R$ be a graph rewrite rule.

(1) We say that L *matches* a term graph $S \in \mathcal{G}(\mathcal{F}, \mathcal{V})$ if there is a morphism $m : L \to_{\mathcal{V}} S$ that suspends the morphism condition on variable nodes. The morphism $m : L \to_{\mathcal{V}} S$ is also called the *matching morphism*.

(2) If $m : L \to_{\mathcal{V}} S\restriction_u$ holds we say that m matches L in S at node $u \in S$. The pair $\langle L \to R, m\rangle$ is called a *redex* in S, and the node $u \in S$ the *redex node*.

Example 4.29 (Continued from Example 4.27). The following depicts a matching morphism $m : L_3 \to_{\mathcal{V}} S\restriction_{ⓑ}$.



Hence $\langle L_3 \to R_3, m\rangle$ is a redex in S, with respect to the graph rewrite rule $L_3 \to R_3$ from Example 4.27. Observe that the redex node in S, viz the node ⓑ, is given by $m(\mathrm{rt}(L_3))$. ◁

Once a redex $\langle L \to R, m\rangle$ in S has been identified, we wish to replace the matched left-hand side L by the instantiated right-hand side of the considered graph rewrite rule $L \to R$. Instantiation of the right-hand side is covered by the next definition.

Definition 4.30 (Application of Matching Morphism)**.** Let S be a term graph and let $L \to R$ be a graph rewrite rule such that $N_R \cap N_S = \varnothing$. Suppose $\langle L \to R, m\rangle$ is a redex in S, with redex node $u \in S$. We denote by $m_S(R)$ the *application of the matching morphism* $m : L \to_{\mathcal{V}} S\restriction_u$ *to* R, obtained by redirecting nodes from R to S according to the morphism m:

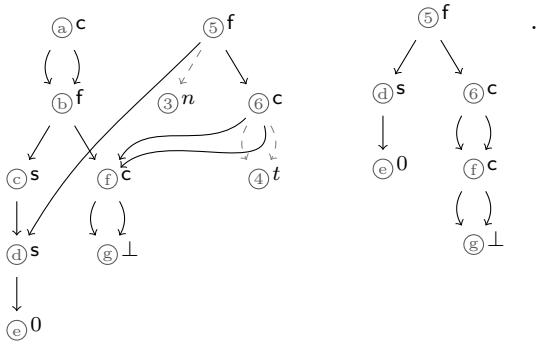
$$m_S(R) := (R \oplus S)[m(v_1), \ldots, m(v_n) \leftarrow v_1, \ldots, v_n]\restriction_{\underline{m}(\mathrm{rt}(R))} ,$$

where $\{v_1, \ldots, v_n\} := N_R \cap N_L$ are the nodes appearing both in R and L.

Note that the graph $m_S(R)$ is by definition rooted. The side condition that nodes in R and S are disjoint ensures that $m_S(R)$ is acyclic, and thus a term graph.

Example 4.31 (Continued from Example 4.29). Reconsider the matching morphism $m : L_3 \to_{\mathcal{V}} S\restriction_{ⓑ}$ from Example 4.29. The set of nodes common to L_3 and L_4 is given by $\{③, ④\}$. The term graph $m_S(R_3)$ is obtained by (i) redirecting edges going to ③ to $m(③) = ⓓ$, redirecting edges going to ④ to $m(④) = ⓕ$ and

(ii) removing nodes inaccessible from the root of R_3.



$(R_3 \oplus S)[ⓓ, ⓕ \leftarrow ③, ④]$ $\qquad$ $m_S(R_3)$

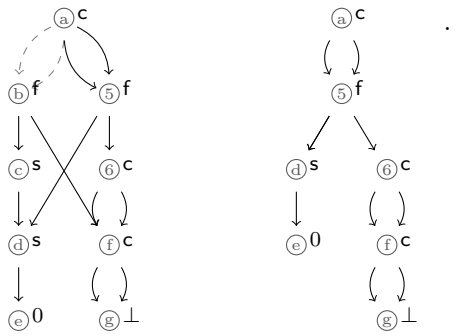
Definition 4.32 (Sub-graph Replacement)**.** Let S and T be two term graph and $u \in S$ be a node. We define

$$S[T]_u := \begin{cases} T & \text{if } u = \mathrm{rt}(S) \\ (S \oplus T)[\mathrm{rt}(T) \leftarrow u]\restriction_{\mathrm{rt}(S)} & \text{otherwise,} \end{cases}$$

to denote the *replacement of* the *sub-graph* $S\restriction_u$ by T in S.

For a position p addressing $u \in S$, we also write $S[T]_p$ instead of $S[T]_u$. Notice that when $\mathsf{lab}_S(v) = \mathsf{lab}_T(v)$ and $\mathsf{succ}_S(v) = \mathsf{succ}_T(v)$ holds for all $v \in N_S \cap N_T$, i.e., $S\restriction_v = T\restriction_v$ holds for common nodes v, then $S[T]_u$ is again a term graph. Term graphs S and T that obey this condition are called *properly sharing*. Observe that by construction, S and $m_S(R)$ are always properly sharing.

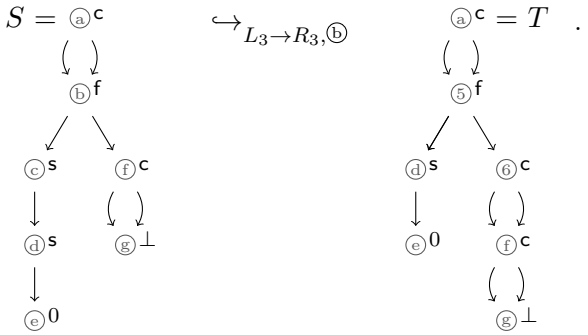
Example 4.33 (Continued from Example 4.31). Reconsider the term graph S from Example 4.29 and the term graph $m_S(R_3)$ constructed in Example 4.31. The term graph $S[m_S(R_3)]_{ⓑ}$ is constructed in the following two steps.



$(S \oplus m_S(R_3))[⑤ \leftarrow ⓑ]$ $\qquad$ $S[m_S(R_3)]_{ⓑ}$

Definition 4.34 (Pre-Reduction Step)**.** Let S be a term graph and let $L \to R$ be a graph rewrite rule such that $N_R \cap N_S = \varnothing$. Suppose $\langle L \to R, m\rangle$ is a redex in S, with redex node $u \in S$. Then S *reduces* to $T := S[m_S(R)]_u$ at redex node u with rule $L \to R$, in notation $S \hookrightarrow_{L\to R,u} T$.

Example 4.35 (Continued from Example 4.31). By combining Example 4.29, Example 4.33 and Example 4.31 we obtain that the graph rewrite rule $L_3 \to R_3$ depicted in Example 4.27 gives rise to the following pre-reduction step.



The following lemma states that for isomorphic term graphs S and S', pre-reduction steps coincide up to renaming of nodes. Justified by this observation, we define the graph rewrite relation $\longrightarrow_{\mathcal{G}}$ as the projection of $\hookrightarrow_{\mathcal{G}}$ to canonical term graphs.

Lemma 4.36. *If $S \hookrightarrow_{L\to R,u} T$ then for any isomorphic term graph $S' \cong^i S$ with $N_R \cap N_{S'} = \varnothing$, $S' \hookrightarrow_{L\to R,u'} T'$ holds for $T' \cong T$ and $u' \in S'$.*

Proof. Consider isomorphic term graphs $S' \cong^i S$ such that $S \hookrightarrow_{L\to R,i(u)} T$ holds with redex $\langle L \to R, m\rangle$, where by definition $T = S[m_S(R)]_{i(u)}$. Define the morphism $m' := m \circ i$. A standard induction on L shows that $\langle L \to R, m'\rangle$ defines a redex in S', with redex node u. By the side-condition $N_R \cap N_{S'} = \varnothing$ we have thus $S' \hookrightarrow_{L\to R,} T'$ for the term graph $T' = S'[m'_{S'}(R)]_u$.

Denote by $j : N_{T'} \to N_T$ the function such that for all $v \in T'$, $j(v) := i(v)$ if $v \in S'$, and $j(v) := v$ if $v \in R$. Then it can be shown that j is bijective and defines an isomorphism $T' \cong^j T$. □

Definition 4.37 (Graph Rewrite Relation)**.** Let $\mathcal{G}$ be a GRS. We define

$$S \longrightarrow_{L\to R,u} T \quad :\Longleftrightarrow \quad S' \hookrightarrow_{L\to R,u'} T'$$

for some term graphs $S = \mathcal{C}(S')$ and $T = \mathcal{C}(T')$ and $u = \mathcal{P}\mathrm{os}_{S'}(u')$. The node u is called the redex node of the step $S \longrightarrow_{L\to R,u} T$.

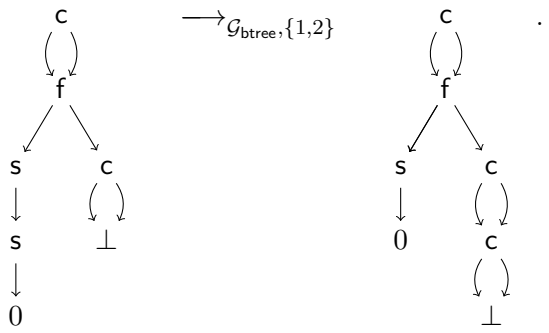
We set $S \longrightarrow_{\mathcal{G}} T$ if $S \longrightarrow_{L\to R,u} T$ holds for some rule $L \to R \in \mathcal{G}$ and node $u \in S$. The relation $\longrightarrow_{\mathcal{G}}$ is called the *graph rewrite relation* induced by $\mathcal{G}$. We also write $S \longrightarrow_{\mathcal{G},u} T$ or $S \longrightarrow_{\mathcal{G},p} T$ where $p \in u \in S$ to indicate the redex node or a position p that addresses u in S respectively. In this case we also say that S is *$\mathcal{G}$-reducible* at node u, or position p.

As a consequence of Lemma 4.25 we have

$$S \longrightarrow_{\mathcal{G},u} T \quad \Longleftrightarrow \quad S \cong S' \hookrightarrow_{\mathcal{G},u'} T' \cong T$$

for some term graphs S', T'. We emphasise that the redex node $u = \mathcal{P}\mathrm{os}_{S'}(u')$ in S is the image of the morphism underlying $S' \cong S$, compare Lemma 4.25(2).

Example 4.38 (Continued from Example 4.35). Witnessed by the pre-reduction step $S \hookrightarrow_{L_3 \to R_3, ⓑ} T$ depicted in Example 4.35, we have



4.3. Simulating Term Rewriting by Graph Rewriting

In this section we show that our formalisation of graph rewriting provides a *sound implementation* of term rewriting.

Definition 4.39 (Unfolding of Graph Rewrite Systems)**.** For a graph rewrite system $\mathcal{G}$ we define its *unfolding* as

$$\mathsf{U}(\mathcal{G}) := \{\mathsf{U}(L \to R) \mid L \to R \in \mathcal{G}\} \,,$$

where $\mathsf{U}(L \to R) := \mathsf{U}(L) \to \mathsf{U}(R)$ denotes the unfolding of the graph rewrite rule $L \to R$.

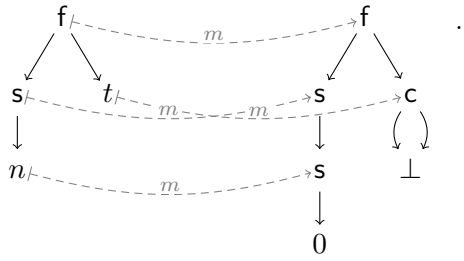
Example 4.40 (Continued from Example 3.2). The GRS $\mathcal{G}_{\mathsf{btree}}$ depicted in Example 3.2 is the unfolding of the TRS $\mathcal{R}_{\mathsf{btree}}$ from Example 3.1. ◁

As a first step towards our soundness result, we show that the morphism based matching mechanism provides a sound implementation of the substitution based matching underlying term rewriting.

Definition 4.41 (Induced Substitution)**.** Let $L \to R$ be a rewrite rule and let S be a term graph connected by a matching morphism $m : L \to_{\mathcal{V}} S$. Then m *induces* a substitution σ_m as follows: $\sigma_m(x) := \mathsf{U}(S\restriction_{m(u_x)})$ for all variables x occurring in L.

Notice that the induced substitution is well defined as each variable $x \in \mathcal{V}$ occurring in the left-hand side L is represented by a unique node, compare clause (i) in Definition 4.26. The following example illustrates the construction of the induced substitution.

Example 4.42 (Continued from Example 4.29). Consider the following matching morphism



The induced substitution is given by $\sigma_m(n) := \mathsf{s}(0)$ and $\sigma_m(t) := \mathsf{c}(\bot, \bot)$. ◁

Lemma 4.43. *Let $L \to R$ be a graph rewrite rule and S be term graphs connected by a matching morphism $m : L \to_{\mathcal{V}} S$. Then $\mathsf{U}(L)\sigma_m = \mathsf{U}(S)$ for the induced substitution σ_m.*

Proof. To prove this one shows the stronger statement $\mathsf{U}(L\restriction_u)\sigma_m = \mathsf{U}(S\restriction_{m(u)})$ for all $u \in L$ by structural induction on L. Using $m(\mathrm{rt}(L)) = \mathrm{rt}(S)$, the lemma follows. □

Lemma 4.44. *Let $L \to R$ be a graph rewrite rule and let S be a term graph such that $N_R \cap N_S = \varnothing$. Suppose there is a matching morphism $m : L \to_{\mathcal{V}} S$. Then $\mathsf{U}(R)\sigma_m = \mathsf{U}(m_S(R))$, for σ_m the substitution induced by m.*

Proof. Using that nodes in S and R are disjoint, one can verify that a path

$$\mathrm{rt}(R) = u_0 \xrightarrow{i_1}_R u_1 \xrightarrow{i_2}_R \cdots \xrightarrow{i_n}_R u_{n+1} \,,$$

translates to a path

$$\underline{m}(\mathrm{rt}(R)) = \underline{m}(u_0) \xrightarrow{i_1}_{m_S(R)} \underline{m}(u_1) \xrightarrow{i_2}_{m_S(R)} \cdots \xrightarrow{i_n}_{m_S(R)} \underline{m}(u_{n+1}) \,,$$

where $\underline{m}$ denotes the extension of m to all nodes. This observation follows by a standard induction on the length n of the path.

To prove the lemma, we now prove the stronger statement $\mathsf{U}(R\restriction_u)\sigma_m = \mathsf{U}(m_S(R)\restriction_{\underline{m}(u)})$ for all nodes $u \in R$. The proof is by induction on the depth of R. We distinguish two cases. Suppose first $u \in N_L$. In this case $\underline{m}(u) = m(u)$, and $\mathsf{U}(R\restriction_u)\sigma_m = \mathsf{U}(L\restriction_u)\sigma_m = \mathsf{U}(S\restriction_{m(u)})$ holds due to Lemma 4.43. By the initial observation, and since S and R are disjoint, we have $\mathsf{U}(S\restriction_{m(u)}) = \mathsf{U}(m_S(R)\restriction_{\underline{m}(u)})$ as desired.

Otherwise $u \notin N_L$ and u is not a variable node. Suppose $\mathsf{lab}_R(u) = f \in \mathcal{F}$ and $\mathsf{succ}_R(u) = [u_1, \ldots, u_k]$. Using the above observation we have $\mathsf{succ}_{m_S(R)}(\underline{m}(u)) = [\underline{m}(u_1), \ldots, \underline{m}(u_k)]$, since also $\mathsf{lab}_{m_S(R)}(u) = \mathsf{lab}_R(u)$ holds, we conclude by induction hypothesis

$$\begin{aligned}
\mathsf{U}(R\restriction_u)\sigma_m &= f(\mathsf{U}(R\restriction_{u_1}), \ldots, \mathsf{U}(R\restriction_{u_k})) \\
&= f(\mathsf{U}(m_S(R)\restriction_{\underline{m}(u_1)}), \ldots, \mathsf{U}(m_S(R)\restriction_{\underline{m}(u_k)})) \\
&= \mathsf{U}(m_S(R)\restriction_{\underline{m}(u)}) \,.
\end{aligned}$$

□

Lemma 4.45. *Let S and T be two properly sharing term graphs. Then*

$$\mathsf{U}(S[T]_u) = C[\mathsf{U}(T), \ldots, \mathsf{U}(T)] \ ,$$

where $C = \mathsf{U}(S[\Box]_u)$. Here $\Box$ is also used to denote a term graph that unfolds to the empty context, where we suppose that single node does not occur in S.

Proof. If $u = \mathrm{rt}(S)$ then the lemma follows by definition, so suppose $u \neq \mathrm{rt}(S)$. For all $v \in S$, abbreviate

$$v^T := \begin{cases} \mathrm{rt}(T) & \text{if } v = u, \\ v & \text{otherwise,} \end{cases} \qquad v^\Box := \begin{cases} \mathrm{rt}(\Box) & \text{if } v = u, \\ v & \text{otherwise.} \end{cases}$$

Thus we can write

$$\begin{aligned} S[T]_u &= (S \oplus T)[u^T \leftarrow u]\restriction_{\mathrm{rt}(S)} \\ S[\Box]_u &= (S \oplus \Box)[u^\Box \leftarrow u]\restriction_{\mathrm{rt}(S)} \ . \end{aligned}$$

For all v with $v^\Box \in S[\Box]_u$, define the context $C_v := \mathsf{U}(S[\Box]_u\restriction_{v^\Box})$. We show

$$C_v[\overline{\mathsf{U}(T)}] = \mathsf{U}(S[T]_u\restriction_{v^T}) \text{ for all } v^\Box \in S[\Box]_u \ .$$

Here $\overline{\mathsf{U}(T)}$ denotes a sequence of terms $\mathsf{U}(T)$ of appropriate length. From this, the lemma follows by taking $\mathrm{rt}(S)$ for v. First consider the case $u = v$, where $v^T = \mathrm{rt}(T)$ and $v^\Box = \mathrm{rt}(\Box)$. Employing that S and T are properly sharing, we see that $C_v = \Box$ and $S[T]_u\restriction_{\mathrm{rt}(T)} = T$. The claim follows for this case. Hence consider the remaining case $u \neq v$, where $v^T = v = v^\Box$ by definition. We proceed by structural induction on $S[\Box]_u$. In the base case v is a variable node where in particular $\mathrm{rt}(S) \rightharpoonup_S^* v$, thus we even have that

$$S[\Box]_u\restriction_{v^\Box} = S[\Box]_u\restriction_v = S[T]_u\restriction_v = S[T]_u\restriction_{v^T} \ ,$$

holds as desired. For the inductive step, suppose $\mathsf{lab}_{S[\Box]_u}(v) = f \in \mathcal{F}$ and $\mathsf{succ}_{S[\Box]_u}(v) = [v_1^\Box, \ldots, v_k^\Box]$. Using again $\mathrm{rt}(S) \rightharpoonup_S^* v$ and the identity $v^T = v = v^\Box$, we see $\mathsf{lab}_{S[T]_u}(v) = f$ and $\mathsf{succ}_{S[T]_u}(v) = [v_1^T, \ldots, v_k^T]$. Using the induction hypothesis we conclude

$$\begin{aligned} C_v[\overline{\mathsf{U}(T)}] &= f(C_{v_1}(\overline{\mathsf{U}(T)}), \ldots, C_{v_1}(\overline{\mathsf{U}(T)})) \\ &= f(\mathsf{U}(S[T]_u\restriction_{v_1^T}), \ldots, \mathsf{U}(S[T]_u\restriction_{v_k^T})) \\ &= \mathsf{U}(S[T]_u\restriction_{v^T}) \ . \end{aligned}$$

□

Putting Lemma 4.43, Lemma 4.44 and Lemma 4.45 together, we obtain our simulation result.

Lemma 4.46. *Suppose $S \longrightarrow_{L \to R, u} T$ for $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ and node $u \in S$. Then*

$$\mathsf{U}(S) = C[\overline{l\sigma}] \rightarrow_{l \to r}^n C[\overline{r\sigma}] = \mathsf{U}(T) \ ,$$

where C is an n-holed context, with holes exactly at position $u = \{p_1, \ldots, p_n\}$, and $l \to r = \mathsf{U}(L \to R)$.

Proof. Suppose $S \longrightarrow_{L\to R,u} T$, thus there exists graphs S', T' where $\mathcal{C}(S') = S$ and $\mathcal{C}(T') = T$ such that $S' \hookrightarrow_{L\to R,u'} T'$ for the node $u' \in S$ with $u = \mathcal{P}\mathrm{os}_{S'}(u')$. Let $\langle L \to R, m\rangle$ denote the redex in S'. Define the context $C := \mathsf{U}(S'[\Box]_{u'})$. We have $C|_p = \Box$ exactly for $p \in \mathcal{P}\mathrm{os}_{S'}(u) = \{p_1, \ldots, p_n\}$ as required by the lemma. Note that S' and $S'\restriction_{u'}$ are trivially properly sharing. Thus

$$\begin{aligned}
\mathsf{U}(S') &= \mathsf{U}(S'[S'\restriction_{u'}]_{u'}) \\
&= C[\mathsf{U}(S'\restriction_{u'}), \ldots, \mathsf{U}(S'\restriction_{u'})] && \text{by Lemma 4.45} \\
&= C[l\sigma_m, \ldots, l\sigma_m] && \text{by Lemma 4.43,}
\end{aligned}$$

for the substitution σ_m induces by m. Using that S' and $m_S(R)$ are properly sharing, and that by assumption $S' \hookrightarrow_{L\to R,u'} T'$ we have $N_R \cap N_{S'} = \varnothing$. We obtain

$$\begin{aligned}
\mathsf{U}(T) &= S'[m_S(R)]_{u'} \\
&= C[\mathsf{U}(m_S(R)), \ldots, \mathsf{U}(m_S(R))] && \text{by Lemma 4.45} \\
&= C[r\sigma_m, \ldots, r\sigma_m] && \text{by Lemma 4.44 .}
\end{aligned}$$

In total, $\mathsf{U}(S') = C[\overline{l\sigma_m}] \to^n_{l\to r} C[\overline{r\sigma_m}] = \mathsf{U}(T')$ holds. The lemma follows from this by applying Lemma 4.25(2) and Lemma 4.17 twice. □

Let $\mathcal{G}$ be a GRS, and denote by $\mathcal{R}$ its unfolding. In total we obtain that any reduction

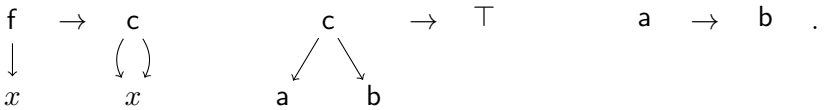
$$S_0 \longrightarrow_{\mathcal{G}} S_1 \longrightarrow_{\mathcal{G}} \cdots \longrightarrow_{\mathcal{G}} S_\ell \,,$$

implements a term rewriting sequence

$$\mathsf{U}(S_0) \to^+_{\mathcal{R}} \mathsf{U}(S_1) \to^+_{\mathcal{R}} \cdots \to^+_{\mathcal{R}} \mathsf{U}(S_\ell) \,.$$

In this sense, graph rewriting provides a *sound* implementation of term rewriting. In graph rewriting we cannot avoid sharing. As a consequence, graph rewriting does not constitute a *complete* implementation in general. We clarify this in the next example.

Example 4.47. Consider the GRS $\mathcal{G}_\mathsf{f}$ given by the three rules



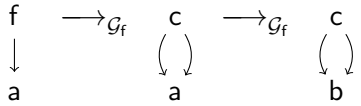
The unfolding $\mathcal{R}_\mathsf{f}$ is given by the rule

$$\mathsf{f}(x) \to \mathsf{c}(x, x) \qquad \mathsf{c}(\mathsf{a}, \mathsf{b}) \to \top \qquad \mathsf{a} \to \mathsf{b} \,,$$

and admits the following two normalising derivations

$$\begin{array}{llllll}
\mathsf{f}(\mathsf{a}) & \to_{\mathcal{R}_\mathsf{f}} & \mathsf{c}(\mathsf{a},\mathsf{a}) & \to_{\mathcal{R}_\mathsf{f}} & \mathsf{c}(\mathsf{a},\mathsf{b}) & \to_{\mathcal{R}_\mathsf{f}} \mathsf{c}(\mathsf{b},\mathsf{b}) \,, \\
\mathsf{f}(\mathsf{a}) & \to_{\mathcal{R}_\mathsf{f}} & \mathsf{c}(\mathsf{a},\mathsf{a}) & \to_{\mathcal{R}_\mathsf{f}} & \mathsf{c}(\mathsf{a},\mathsf{b}) & \to_{\mathcal{R}_\mathsf{f}} \top \,,
\end{array}$$

starting from $\mathsf{f}(\mathsf{a})$. Whereas $\mathcal{G}_\mathsf{f}$ can simulate the former reduction, by contracting both redexes in $\mathsf{c}(\mathsf{a},\mathsf{a})$ simultaneously, that is,



$\mathcal{G}_\mathsf{f}$ cannot simulate the second $\mathcal{R}_\mathsf{f}$ reduction. ◁

Another obstacle that destroys completeness is that the matching mechanism takes also sharing into account. More precise, the inverse direction of Lemma 4.43 does not hold in general. This is clarified in the next example.

Example 4.48. Consider the GRS $\mathcal{G}_\mathsf{h}$ consisting of the graph rewrite rules

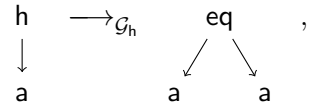
that unfolds to the TRS $\mathcal{R}_\mathsf{h}$ given by the rules

$$\mathsf{h}(x) \to \mathsf{eq}(x,\mathsf{a}) \qquad\qquad \mathsf{eq}(x,x) \to \top\ .$$

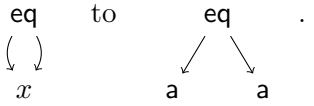
The TRS $\mathcal{R}_\mathsf{h}$ admits the derivation

$$\mathsf{h}(\mathsf{a}) \to_{\mathcal{R}_\mathsf{h}} \mathsf{eq}(\mathsf{a},\mathsf{a}) \to_{\mathcal{R}_\mathsf{h}} \top\ .$$

The GRS $\mathcal{G}_\mathsf{h}$ can simulate the first step,



but then the derivation gets stuck because the obtained term graph is a normal form, in particular there exists no matching morphism from the left-hand side



Notice that the morphism conditions would require that the x-node is simultaneously mapped to both a nodes. ◁

The problem indicated in the second example arises when the intended redex admits more sharing than the corresponding left-hand side L. We can overcome this problem by either requiring that L is a tree, or that S is fully collapsed.

Lemma 4.49. *Consider terms $l, s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $l\sigma = s$ for some substitution σ. Let $L, S \in \mathcal{G}(\mathcal{F}, \mathcal{V})$ be term graphs that unfold to l and s respectively: $\mathsf{U}(L) = l$ and $\mathsf{U}(S) = s$. Then there exists a matching morphism $m : L \to_{\mathcal{V}} S$, if (i) $L \in \triangle(\mathcal{F}, \mathcal{V})$, or (ii) $S \in \triangledown(\mathcal{F}, \mathcal{V})$.*

Proof. We prove the lemma by structural induction on l. If $l = x$ is a variable, then L consists by assumption of a single node u_x with $\mathsf{lab}_L(u_x) = x \in \mathcal{V}$. Define $m(u_x) := \mathrm{rt}(S)$ and $m : L \to_{\mathcal{V}} S$ holds as desired. For the inductive step, suppose $l = f(l_1, \ldots, l_k)$ and hence $s = f(l_1\sigma, \ldots, l_k\sigma)$. For $i = 1, \ldots, k$, denote by L_i and S_i the direct sub-graphs of L and S respectively. By induction hypothesis, $m_i : L_i \to_{\mathcal{V}} S_i$ exist for $i = 1, \ldots, k$.

We claim that if $u \in N_{L_i} \cap N_{L_j}$ then $m_i(u) = m_j(u)$. For the case (i) $L \in \triangle(\mathcal{F}, \mathcal{V})$, this is clear as nodes in L_i and L_j are disjoint for pairwise different $i, j = 1, \ldots, k$. Consider the case (ii) $S \in \triangledown(\mathcal{F}, \mathcal{V})$, and consider arbitrary $u \in N_{L_i} \cap N_{L_j}$. Let $p \in \mathcal{P}\mathrm{os}_{L_i}(u)$ and $q \in \mathcal{P}\mathrm{os}_{L_j}(u)$. Using the matching morphism $m_i : L_i \to_{\mathcal{V}} S_i$ we see by Lemma 4.22 that $p \in \mathcal{P}\mathrm{os}_{S_i}(m_i(u))$ so in particular $l_i\sigma|_p = \mathsf{U}(S_i\restriction_{m_i(u)}) = \mathsf{U}(S\restriction_{m_i(u)})$, where the first equality follows by Lemma 4.19 and the second follows since S_i is a sub-graph of S. By identical reasoning we obtain $l_j\sigma|_q = \mathsf{U}(S\restriction_{m_j(u)})$. Since $L_i\restriction_u = L_j\restriction_u$ by assumption, we have $l_i|_p\sigma = l_j|_q\sigma$ using Lemma 4.19 twice. Thus

$$\mathsf{U}(S\restriction_{m_i(u)}) = l_i\sigma|_p = l_i|_p\sigma = l_j|_q\sigma = l_j\sigma|_q = \mathsf{U}(S\restriction_{m_j(u)}) \; .$$

As by assumption $S \in \triangledown(\mathcal{F}, \mathcal{V})$ we obtain that $m_i(u) = m_j(u)$.

Define the function $m : N_L \to N_S$ such that: for all $u \in L$, $m(u) := m(\mathrm{rt}(S))$ if $u = \mathrm{rt}(L)$; otherwise $m(u) := m_i(u)$ if $u \in L_i$. As $m_i(u) = m_j(u)$ for shared nodes u appearing in L_i and L_j, m is unambiguous, in particular m restricted to nodes in L_i results in m_i again. Using that $m_i : L_i \to_{\mathcal{V}} S_i$ for $i = 1, \ldots, k$, it follows that m defines a matching morphism $m : L \to_{\mathcal{V}} S$. □

Note that if $L \in \triangle(\mathcal{F}, \mathcal{V})$ for a graph rewrite rule $L \to R$ then by the condition (iii) in Definition 4.26 the unfolding of $L \to R$ is left-linear.

In Example 4.47 and Example 4.48 we indicated two reasons why graph rewriting is incomplete. The final lemma of this section confirms that there are no further surprises hidden.

Lemma 4.50. *Let $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ and let $u = \{p\}$ be a node in S which is not shared. If (i) $L \in \triangle(\mathcal{F}, \mathcal{V})$ or (ii) $S\restriction_p \in \triangledown(\mathcal{F}, \mathcal{V})$ then the following properties hold:*

(1) *if $S \longrightarrow_{L\to R,p} T$ then $\mathsf{U}(S) \to_{\mathsf{U}(L\to R),p} \mathsf{U}(T)$; and*

(2) *if $\mathsf{U}(S) \to_{\mathsf{U}(L\to R),p} t$ then $S \longrightarrow_{L\to R,p} T$ for some $T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ with $\mathsf{U}(T) = t$.*

Proof. The first property is an immediate consequence of Lemma 4.46. Consider the second property. Suppose $\mathsf{U}(S) = C[\mathsf{U}(L)\sigma] \to_{\mathsf{U}(L\to R),p} C[\mathsf{U}(R)\sigma]$ for some context C and substitution σ. We have $S\restriction_u = \mathsf{U}(S)|_p$ by Lemma 4.19. Using the assumptions on L or $S\restriction_p$ as premises for Lemma 4.49, we see that there

exists a matching morphism $m : L \to_{\mathcal{V}} S|_p$. Hence $\langle L \to R, m\rangle$ is a redex in S at redex-node u. As a consequence, $S \longrightarrow_{L\to R,u} T$ holds for some term graph T. Then Lemma 4.46 gives a substitution σ' and context C', with hole at position p, such that

$$\mathsf{U}(S) = C'[\mathsf{U}(L)\sigma'] \to_{\mathsf{U}(L\to R)} C'[\mathsf{U}(R)\sigma'] = \mathsf{U}(T) \ .$$

As $C[\mathsf{U}(L)\sigma] = \mathsf{U}(S) = C'[\mathsf{U}(L)\sigma']$ where both for C and C' the hole is positioned at p, we conclude that the context C and C', as well as the substitutions σ and σ' coincide. In particular this gives $\mathsf{U}(T) = C[\mathsf{U}(R)\sigma]$ as desired. □

Chapter 5.

The Adequacy Theorem

We use following notion of *adequacy* to relate term rewriting and graph rewriting.

Definition 5.1 (Adequacy)**.** Let $\mathcal{G} \subseteq \mathcal{G}_c(\mathcal{F},\mathcal{V})$ be a set of canonical term graphs, and let $\longrightarrow_G \subseteq \mathcal{G}_c(\mathcal{F},\mathcal{V})^2$ and $\to_R \subseteq \mathcal{T}(\mathcal{F},\mathcal{V})^2$ be binary relations on term graphs and terms respectively. Then $\longrightarrow_G$ is called *adequate* on $\mathcal{G}$ for $\to_R$ if the following conditions are satisfied:

(1) Surjectivity of unfolding on $\mathcal{G}$: for every $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ there exists some $T \in \mathcal{G}$ that unfolds to t: $\mathsf{U}(T) = t$.

(2) Closure under reductions of $\mathcal{G}$: if $S \longrightarrow_G T$ for $S \in \mathcal{G}$, then $T \in \mathcal{G}$.

(3) Preservation of reductions: if $S \longrightarrow_G T$ for $S \in \mathcal{G}$, then $\mathsf{U}(S) \to_R \mathsf{U}(T)$.

(4) Simulation of reductions: if $\mathsf{U}(S) \to_R t$ for $S \in \mathcal{G}$, then $S \longrightarrow_G T$ where $\mathsf{U}(T) = t$.

If there exists a set of graphs $\mathcal{G}$ such that the above conditions hold, we simply say that $\longrightarrow_G$ is adequate for $\to_R$.

Surjectivity of unfolding ensures that every term has a graph representation in the restricted set of graphs $\mathcal{G}$, and closure under reductions of $\mathcal{G}$ ensures that graph rewrite steps do not lead outside $\mathcal{G}$. Preservation of reductions gives a *soundness* property: graph rewriting reductions on $\mathcal{G}$ implement only term rewrite sequences. On the other hand, simulation of reductions states a *completeness* property: every term rewriting derivation is simulated by some graph rewriting derivation.

Remark. Our notion of adequacy is a refinement of the notion of adequacy found in Kennaway et al. [48]. In Kennaway et al., the clauses corresponding to Assertions (3) and (4) relate complete derivations, we relate steps. More severe, for simulation of reduction requires, Kennaway et al., require only that $S \longrightarrow_G^* U$ gives an extension of $\mathsf{U}(S) \to_R^* t$, that is, $t \to_R^* \mathsf{U}(U)$ holds. We depart from this definition, on the one hand to avoid problems with non-confluent TRSs. On the other hand, we precisely want to relate the number of steps in both formalisms.

It is well known that by integrating folding and unfolding into the graph rewrite relation, recovers adequacy [75, 66]. In fact, it is not difficult to proof that $\longrightarrow_{\mathcal{G}}$ extended by folding and unfolding is adequate for $\to_{\mathcal{R}}$, even on $\triangle_c(\mathcal{F},\mathcal{V})$. We have

$$\mathsf{U}(S) \to_{\mathcal{R}} \mathsf{U}(T) \text{ if and only if } S \succcurlyeq \cdot \longrightarrow_{\mathcal{R}} \cdot \preccurlyeq T\ .$$

Since S is supposed to be a tree here, the intended redex node in S is unshared, which addresses the problem indicated in Example 4.47. Folding S before the reduction step is used to fully collapse the sub-graph of S rooted at the intended redex-node, which addresses the problem highlighted in Example 4.48. Unfolding $\preccurlyeq$ is used to translate the reduct back to a tree T. Of course, this way we cannot hope to achieve our ultimate goal, the efficient implementation of rewriting.

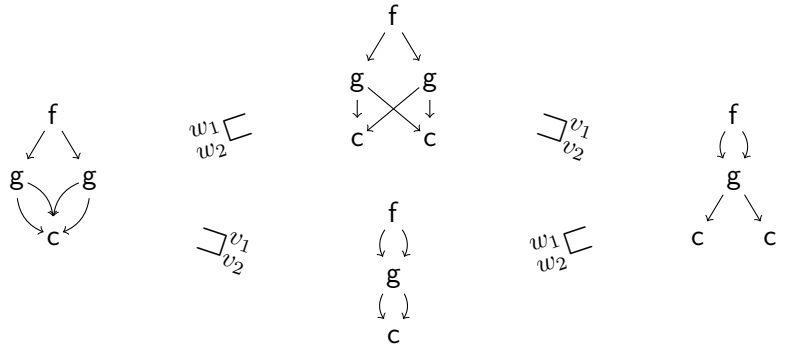
5.1. Restricted Folding and Unfolding

In this work, we take a fresh look at adequacy, from a complexity related point of view. Our adequacy theorems presented below is based on *restricted unfolding* $\lhd_p$ and *restricted folding* $\blacktriangleright_p$, that allow for a precise control of the resources copied. These relations are given in Definition 5.6 below. Both relations preserve term structure. When $S \blacktriangleright_p T$ holds then the sub-graph $T|_p$ admits strictly *more sharing* than $S|_p$. Conversely, when $S \lhd_p T$ holds, nodes above p in T admit *less sharing* than nodes above p in S. The relations $\lhd_p$ and $\blacktriangleright_p$ are based on *single step* approximations $\sqsupset_v^u$ of $\succcurlyeq$.

Definition 5.2. Let $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$, and let $u, v \in S$ be two nodes in S. We define $S \sqsupseteq_v^u T$ if $S \succcurlyeq_m T$ for the morphism m identifying only u and v, more precisely, $m(u) = m(v)$ and $m(w) = w$ for all $w \in N_S \setminus \{u, v\}$. We define $S \sqsupset_v^u T$ if $S \sqsupseteq_v^u T$ and $u \neq v$.

We write $S \sqsupseteq_v T$ ($S \sqsupset_v T$) if there exists $u \in S$ such that $S \sqsupseteq_v^u T$ ($S \sqsupset_v^u T$) holds. We denote by ${}_v^u\sqsubseteq$ and ${}_v^u\sqsubset$ the inverse of $\sqsupseteq_v^u$ and $\sqsupset_v^u$. Note that $\sqsupseteq_v^u = \sqsupseteq_u^v$ for all nodes $u, v \in S$, and $S \sqsupseteq_u^u T$ holds if and only if $S = T$.

Example 5.3. Let $v_1 = \{1\}$, $v_2 = \{2\}$, $w_1 = \{1{\cdot}1, 1{\cdot}2\}$ and $w_1 = \{2{\cdot}1, 2{\cdot}2\}$. The following diagram depicts four term graphs which are related by $\sqsupset_v^u$, for nodes $u, v \in \{v_1, v_2, w_1, w_2\}$.



◁

Recall that for $S \succcurlyeq T$, $\mathcal{P}\mathrm{os}(S)$ and $\mathcal{P}\mathrm{os}(T)$ coincide. Thus when $S \sqsupset_v^u T$ holds, as a consequence of Lemma 4.22 the nodes $u, v \in S$ are collapsed to the node $u \cup v \in T$.

Lemma 5.4. *Let $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ and let $u, v \in S$ be two distinct nodes. Then there exists a term graph T with $S \sqsupseteq_v^u T$ if and only if (i) $\mathsf{lab}_S(u) = \mathsf{lab}_S(v)$ and (ii) $\mathsf{succ}_S(u) = \mathsf{succ}_S(v)$.*

Proof. Consider first the direction from left to right. Suppose $S \sqsupseteq_v^u T$ and let m be the morphism underlying $\sqsupseteq_v^u$, where in particular $m(u) = m(v)$. By the labeling condition it follows that

$$\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u)) = \mathsf{lab}_T(m(v)) = \mathsf{lab}_S(v) \ .$$

Now pick successors u_i and v_i such that $u \xrightarrow{i}_S u_i$ and $v \xrightarrow{i}_S v_i$. The successor condition gives $m(u) \xrightarrow{i}_T m(u_i)$ and $m(v) \xrightarrow{i}_T m(v_i)$, and hence $m(u_i) = m(v_i)$. Since S is acyclic, neither $u = u_i$ nor $u = v_i$, and by the definition of m we thus conclude $u_i = m(u_i) = m(v_i) = v_i$. Overall $\mathsf{succ}_S(u) = \mathsf{succ}_S(v)$ follows.

Finally consider the direction from right to left. Let S be a term graph with distinct nodes u, v such that $\mathsf{lab}_S(u) = \mathsf{lab}_S(v)$ and $\mathsf{succ}_S(u) = \mathsf{succ}_S(v)$ holds. We obtain a canonical term graph by identifying nodes u and v in S as follows:

$$N_T := (N_S \setminus \{u, v\}) \cup \{u \cup v\} \ ,$$

labels are derived from S by

$$\mathsf{lab}_T(w) := \begin{cases} \mathsf{lab}_S(u) = \mathsf{lab}_S(v) & \text{if } w = u \cup v, \\ \mathsf{lab}_S(w) & \text{otherwise.} \end{cases}$$

We redirect all edges going to u or v in S to the fresh node $u \cup v$, and use the outgoing edges of u (v respectively) as outgoing edges of $u \cup v$.

$$\mathsf{succ}_T^i(w) := \begin{cases} \mathsf{succ}_T^i(u) = \mathsf{succ}_T^i(v) & \text{if } w = u \cup v, \\ u \cup v & \text{if } \mathsf{succ}_S^i(w) = u \text{ or } \mathsf{succ}_S^i(w) = v, \\ \mathsf{succ}_S^i(w) & \text{otherwise.} \end{cases}$$

Then T defines a canonical term graph. Define the function $m : N_S \to N_T$ such that $m(u) := u \cup v =: m(v)$ and $m(w) := w$ otherwise. Using Lemma 4.22 we see $S \succcurlyeq_m T$, in particular $S \sqsupseteq_v^u T$ holds. □

Note that the relation $\sqsupseteq_u$ enjoys the following diamond property.

Lemma 5.5. . *We have ${}_u\sqsubseteq \cdot \sqsupseteq_v \subseteq \sqsupseteq_v \cdot {}_u\sqsubseteq$.*

Proof. Assume $T_1 \, {}_u^{u'}\sqsubseteq S \sqsupseteq_v^{v'} T_2$ for some canonical term graphs S, T_1 and T_2. The only non-trivial case is $T_1 \, {}_u^{u'}\sqsubset S \sqsupseteq_v^{v'} T_2$ for $\{u', u\} \neq \{v', v\}$, where in particular $u' \neq u$ and $v' \neq v$. Otherwise either $T_1 = S$ or $T_2 = S$ and the peak is trivial to join. We analyse two sub-cases.

- Case $\{u', u\} \cap \{v', v\} = \{w\}$: Without loss of generality, suppose $T_1 \, {}_u^w\sqsubset S \sqsupseteq_v^w T_2$ with $u \neq v$. Using Lemma 5.4 from left to right twice we see that $\mathsf{lab}_S(u) = \mathsf{lab}_S(w) = \mathsf{lab}_S(v)$ and $\mathsf{succ}_S^i(u) = \mathsf{succ}_S^i(w) = \mathsf{succ}_S^i(v)$ for all appropriate i. Consider the morphism $m_1 : S \to_\varnothing T_1$ underlying

$S \sqsupseteq_u^w T_1$. Then m_1 satisfies $m_1(u) = u \cup w = m_1(w)$ and $m_1(v) = v$ since by assumption $u \neq v$ and $w \neq v$. The morphism conditions thus give

$$\begin{aligned}
\mathsf{lab}_{T_1}(u \cup w) &= \mathsf{lab}_{T_1}(m_1(u)) && \text{by assumption} \\
&= \mathsf{lab}_S(u) && \text{labeling condition} \\
&= \mathsf{lab}_S(v) && \\
&= \mathsf{lab}_{T_1}(m_1(v)) && \text{labeling condition} \\
&= \mathsf{lab}_{T_1}(v)\ , && \text{using } m_1(v) = v.
\end{aligned}$$

Similar, for i in range, we have

$$\begin{aligned}
\mathsf{succ}^i_{T_1}(u \cup w) &= \mathsf{succ}^i_{T_1}(m_1(u)) && \text{by assumption} \\
&= m_1(\mathsf{succ}^i_S(u)) && \text{successor condition} \\
&= m_1(\mathsf{succ}^i_S(v)) && \text{using } \mathsf{succ}^i_S(u) = \mathsf{succ}^i_S(v) \\
&= \mathsf{succ}^i_{T_1}(m_1(v)) && \text{successor condition} \\
&= \mathsf{succ}^i_{T_1}(v) && \text{using } m_1(v) = v.
\end{aligned}$$

Hence $\mathsf{succ}_{T_1}(u \cup w) = \mathsf{succ}_{T_1}(v)$. Lemma 5.4 thus gives a canonical term graph U_1 such that $T_1 \sqsupseteq_v^{u \cup w} U_1$, where we denote the underlying morphism by m_2.

In total we obtain $S \sqsupseteq_u^w T_1 \sqsupseteq_v^{u \cup w} U_1$ and $S \sqsupseteq_v^w T_2 \sqsupseteq_u^{v \cup w} U_2$ by symmetric reasoning. Set $m := m_2 \circ m_1$. Comparing Lemma 4.16, we see that $S \succ_m U_1$. By construction the morphism m exactly identifies the nodes u, v and w, that is, $m(u) = m(v) = m(w) = u \cup v \cup w$, and $m(u') = m(u)$ otherwise. By symmetric reasoning we see $S \succ_m U_2$. By the morphism conditions imposed by m one finally obtains $U_1 = U_2$.

- Case $\{u', u\} \cap \{v', v\} = \varnothing$: Suppose $T_1 \,{}^{u'}_{u}\!\sqsubset S \sqsupseteq_v^{v'} T_2$ where the nodes u, u', v and v' are pairwise distinct. Consider $S \sqsupseteq_u^{u'} T_1$ with underlying morphism $m_1 : S \to_\varnothing T_1$. Note that in the consider case $m_1(v) = v$ and $m_1(v') = v'$ holds. Hence

$$\begin{aligned}
\mathsf{lab}_{T_1}(v) &= \mathsf{lab}_{T_1}(m_1(v)) && \text{using } m_1(v) = v \\
&= \mathsf{lab}_S(v) && \text{labeling condition} \\
&= \mathsf{lab}_S(v') && \text{by Lemma 5.4} \\
&= \mathsf{lab}_{T_1}(m_1(v')) && \text{labeling condition} \\
&= \mathsf{lab}_{T_1}(v')\ , && \text{using } m_1(v') = v'.
\end{aligned}$$

Similar, for i in range, we have

$$\begin{aligned}
\mathsf{succ}^i_{T_1}(v) &= \mathsf{succ}^i_{T_1}(m_1(v)) && \text{using } m_1(v) = v \\
&= m_1(\mathsf{succ}^i_S(v)) && \text{successor condition} \\
&= m_1(\mathsf{succ}^i_S(v')) && \text{by Lemma 5.4} \\
&= \mathsf{succ}^i_{T_1}(m_1(v')) && \text{successor condition} \\
&= \mathsf{succ}^i_{T_1}(v') && \text{using } m(v') = v'.
\end{aligned}$$

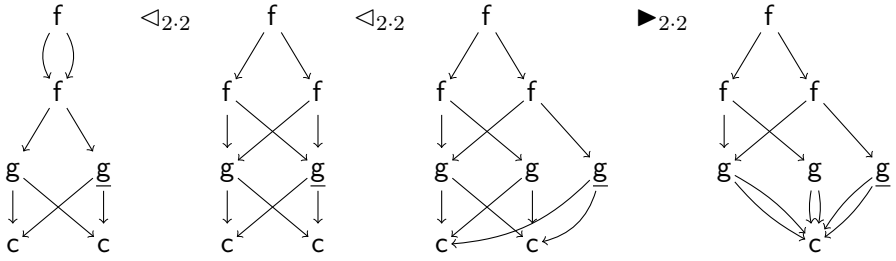
Hence using Lemma 5.4 again we see $S \sqsupset_u^{u'} T_1 \sqsupset_v^{v'} U_1$ for some canonical term graph U_1. Symmetrically we have $S \sqsupset_v^{v'} T_2 \sqsupset_u^{u'} U_2$. One verifies $U_1 = U_2$ exactly as above, which concludes the lemma. □

Definition 5.6. Let $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ be a canonical term graphs and let p be a position in S.

(1) We say that S *folds strictly below* p *to the term graph* T, in notation $S \blacktriangleright_p T$, if $S \sqsupset_v^u T$ for nodes $u, v \in S$ strictly below p in S.

(2) The graph S *unfolds above* p *to the term graph* T, in notation $S \lhd_p T$, if $S \,{}_v^u\!\sqsubset T$ for some unshared node $u \in T$ above p in T, i.e., $\mathcal{P}\mathrm{os}_T(u) = \{q\}$ for $q \leqslant p$.

Example 5.7. The following depicts an exhausting sequence of unfoldings and foldings with respect to 2·2. Nodes addressed by this position are underlined below.



Observe that in the final term graph, the node addressed by 2·2 is not shared, and the sub-graph at position 2·2 is fully collapsed. ◁

By Lemma 5.5 both relations $\blacktriangleright_p$ and $\lhd_p$ enjoy the diamond property. It is not difficult to see that $\blacktriangleright_p$ and $\lhd_p$ are also well founded. The next lemma establishes precise bounds on the length of descending sequences.

Lemma 5.8. *Let* $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ *and consider a position* $p \in \mathcal{P}\mathrm{os}(S)$.

(1) *If* $S \lhd_p^\ell T$ *then* $\ell \leqslant |p|$ *and* $|T| \leqslant |S| + |p|$.

(2) *If* $S \blacktriangleright_p^\ell T$ *then* $\ell \leqslant |S|_p|$ *and* $|T| \leqslant |S|$.

Proof. We consider the first assertion. Consider a possibly infinite sequence

$$S_0 \lhd_p S_1 \lhd_p S_2 \lhd_p \cdots .$$

For $i \in \mathbb{N}$, define $P_i = \{u \mid u \in S_i \text{ with } u = \{q\} \text{ is not shared and } q \leqslant p\}$.

Consider $S_i \lhd_p S_{i+1}$ for arbitrary $i \in \mathbb{N}$, we show that $P_i \subset P_{i+1}$ holds: By definition we have $S_{i+1} \sqsupset_u S_i$ with $u = \{q\}$ for some position $q \leqslant p$. Clearly $P_i \subseteq P_{i+1}$, but moreover $u \in P_{i+1}$ whereas $u \notin P_i$. As a consequence,

$$P_0 \subset P_1 \subset P_2 \subset \dots ,$$

holds. Since there are at most $|p|+1$ nodes above p that are not shared, i.e., the size of P_i ($i \in \mathbb{N}$) is bounded by $|p|+1$, and since P_0 contains at least one node (namely the root), the length of any $\lhd_p$ derivation is bounded by $|p|$. Using that each $\lhd_p$ step increases the size of term graphs by one, the first assertion follows.

For the second assertion, we can prove that if

$$S_0 \blacktriangleright_p S_1 \blacktriangleright_p S_2 \blacktriangleright_p \cdots ,$$

holds, then $|S_i| > |S_{i+1}|$, in particular $|S_i|_p| > |S_{i+1}|_p|$, holds for all $i \in \mathbb{N}$. The assertion follows. □

Definition 5.9. Let $\mathcal{G}$ denote a graph rewrite system. For canonical term graphs S and T we define

$$S \mathrel{\blacktriangleleft\!\!\blacktriangleright}_{\mathcal{G}} T \quad :\Longleftrightarrow \quad S \lhd_p^! \cdot \blacktriangleright_p^! \cdot \longrightarrow_{\mathcal{G},p} T ,$$

where $p \in \mathcal{P}\mathrm{os}(S)$. Similar, we set

$$S \mathrel{\blacktriangleleft\!\!\rightarrow}_{\mathcal{G}} T \quad :\Longleftrightarrow \quad S \lhd_p^! \cdot \longrightarrow_{\mathcal{G},p} T .$$

In the next two lemmas we prove that relations $\lhd_p$ and $\blacktriangleright_p$ fulfil their intended purpose.

Lemma 5.10. *Let S be a term graph and p a position in S.*

(1) *If S is $\lhd_p$-maximal then the node addressed by p is not shared.*

(2) *If S is $\blacktriangleright_p$-minimal then the sub-graph $S|_p$ is fully collapsed, i.e., $S|_p \in \nabla(\mathcal{F},\mathcal{V})$.*

Proof. We consider first Assertion (1). By way of contradiction, suppose S is $\lhd_p$-maximal but the node $u \in S$ addressed by positions p is shared. Recall that in canonical term graphs, nodes coincide with their set of positions, and thus $p \in u$. We construct T such that $S \lhd_p T$.

Since u is shared, there exists some position q and $i \in \mathbb{N}$ such that $p = q \cdot i \cdot p'$ where $\{q\} \in S$ is an unshared node but a node $v = \{q \cdot i\} \uplus v'$ addressed by $q \cdot i$ is shared ($v' \neq \varnothing$). By construction

$$\{q\} \xrightarrow{i}_S \{q \cdot i\} \uplus v' \rightarrow_S^* u .$$

Note that neither v' nor $\{q \cdot i\}$ appear as nodes in S. We obtain the canonical term graph T by unfolding v into two separate nodes $\{q \cdot i\}$ and v'. Formally we define T as follows: The nodes of T are

$$N_T := (N_S \cup \{v', \{q \cdot i\}\}) \setminus \{v\} .$$

Nodes are labelled as in S, where the label of v is used for both v' and $\{q \cdot i\}$:

$$\mathsf{lab}_T(w) := \begin{cases} \mathsf{lab}_S(v) & \text{if } w = v' \text{ or } w = \{q \cdot i\}, \\ \mathsf{lab}_S(w) & \text{otherwise.} \end{cases}$$

Finally edges are defined, for appropriate j, as follows:

$$\mathsf{succ}^j_T(w) := \begin{cases} \{q \cdot i\} & \text{if } w = \{q\} \text{ and } j = i, \\ v' & \text{if } w \neq \{q\} \text{ and } \mathsf{succ}^j_S(w) = v, \\ \mathsf{succ}^j_S(v) & \text{if } w = v' \text{ or } w = \{q \cdot i\}, \\ \mathsf{succ}^j_S(w) & \text{otherwise.} \end{cases}$$

That is, edges are obtained from S by redirecting all edges $w \xrightarrow{i}_S v$ to either $\{q \cdot i\}$ or v'; otherwise edges are kept, where the successors of v are used for both $\{q \cdot i\}$ or v'. It is not difficult to verify that for all nodes $w \in T$, $\mathcal{P}\mathrm{os}_T(w) = w$, i.e., T is a canonical term graph.

Define the function $m : N_T \to N_S$ by $m(v') := v =: m(\{q \cdot i\})$ and $m(w) = w$ otherwise. Consider a node $w \in T$. The function m satisfies $\mathsf{lab}_T(w) = \mathsf{lab}_S(m(w))$. Since $w \subseteq m(w)$ by definition, and since S and T are canonical, Lemma 4.22 gives $T \succcurlyeq_m S$, which by the shape of m can be refined to $T \sqsupset_{v'}^{\{q \cdot i\}} S$. And so $S \lhd_p T$ using that $q \cdot i \leqslant p$. Contradiction.

Now consider Assertion (2). Suppose $S\restriction_p$ is not fully collapsed. We show that S is not $\blacktriangleright_p$-minimal. By the assumption that $S\restriction_p$ is not fully collapsed, there are distinct nodes $u, v \in S\restriction_p$ with $\mathsf{U}(S\restriction_u) = \mathsf{U}(S\restriction_v)$. Assume without loss of generality that u is $\rightharpoonup_S$*-minimal*, in the sense that there is no node u' with $u \rightharpoonup^+ u'$ such that u' is not fully collapsed.

Clearly $\mathsf{lab}_S(u) = \mathsf{lab}_S(v)$ follows from $\mathsf{U}(S\restriction_u) = \mathsf{U}(S\restriction_v)$. Also $\mathsf{succ}_S(u) = \mathsf{succ}_S(v)$, since otherwise $\mathsf{succ}^i_S(u) \neq \mathsf{succ}^i_S(v)$ for some i, which contradicts minimality of u. Lemma 5.4 gives a term graph T with $S \sqsupset^u_v T$. Since u, v are nodes strictly below p in S, we obtain $S \blacktriangleright_p T$, contradiction. □

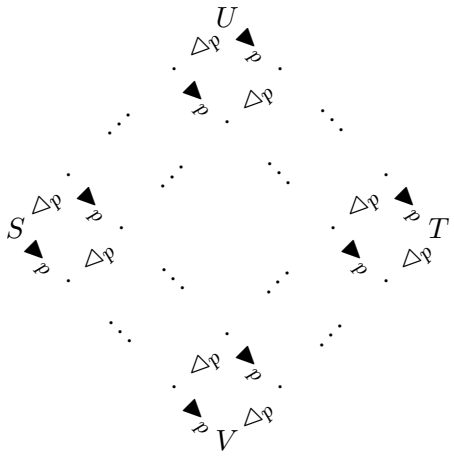
Lemma 5.11. *For all positions p, the following inclusions hold:*

(1) $\lhd_p \cdot \blacktriangleright_p = \blacktriangleright_p \cdot \lhd_p$*; and*

(2) $\lhd^m_p \cdot \blacktriangleright^n_p = \blacktriangleright^n_p \cdot \lhd^m_p$*; and*

(3) $\lhd^!_p \cdot \blacktriangleright^!_p = \blacktriangleright^!_p \cdot \lhd^!_p$.

Proof. Consider the first assertion, we show $\lhd_p \cdot \blacktriangleright_p \subseteq \blacktriangleright_p \cdot \lhd_p$. Consider the peak $T_1 \,{}^{u'}_{u}\!\sqsubset S \sqsupset^{v'}_v T_2$ as induced by $T_1 \lhd_p S \blacktriangleright_p T_2$. Since $v', v \in S$ are distinct nodes strictly below p, whereas $u', u \in S$ are distinct nodes above p, v', v, u' and u are all pairwise distinct. The proof of Lemma 5.5 thus gives $T_1 \sqsupset^{v'}_v \cdot {}^{u'}_{u}\!\sqsubset T_2$, which implies $T_1 \blacktriangleright_p \cdot \lhd_p T_2$ as desired. Dual, one obtains $\lhd_p \cdot \blacktriangleright_p \supseteq \blacktriangleright_p \cdot \lhd_p$, we conclude Assertion (1). Assertion (2) holds as $n \cdot m$ applications of (1) close the following diagram.



Finally, consider Assertion (3). We prove $\lhd_p^! \cdot \blacktriangleright_p^! \subseteq \blacktriangleright_p^! \cdot \lhd_p^!$, the inverse direction is proven dual. Suppose in the above diagram $U \lhd_p^! S \blacktriangleright_p^! V$ holds, we claim $U \blacktriangleright_p^! T \lhd_p^! V$ holds. For a proof by contradiction, suppose $U \blacktriangleright_p^! T$ or $T \lhd_p^! V$ does not hold. In the former case, there exists a term graph T' with $T \blacktriangleright_p^! T'$. Closing the so constructed peak using Assertion (1) towards V contradicts the assumption $S \blacktriangleright_p^! V$. Likewise if $T \lhd_p^! V$ does not hold the obtained peak contradicts $U \lhd_p^! S$. □

5.2. Adequacy for Full Rewriting

Theorem 5.12 (Adequacy for Full Rewriting)**.** *Let $\mathcal{G}$ be a graph rewrite system such that only variable nodes are shared in left-hand sides of $\mathcal{G}$, i.e., $L \in \Diamond^{\mathcal{V}}(\mathcal{F}, \mathcal{V})$ for each rule $L \to R \in \mathcal{G}$. Let $\mathcal{R} := \mathsf{U}(\mathcal{G})$ denote the unfolding of $\mathcal{G}$.*

(1) *The relation $\lhd\!\!\rightarrow_{\mathcal{G}}$ is adequate on $\mathcal{G}_c(\mathcal{F}, \mathcal{V})$ for $\to_{\mathcal{R}}$, whenever $\mathcal{R}$ is left-linear.*

(2) *The relation $\lhd\!\!\blacktriangleright_{\mathcal{G}}$ is adequate on $\mathcal{G}_c(\mathcal{F}, \mathcal{V})$ for $\to_{\mathcal{R}}$.*

Proof. We prove the four properties of adequacy.

(1) Surjectivity of unfolding on $\mathcal{G}_c(\mathcal{F}, \mathcal{V})$: Obviously any $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ can be represented by a tree T: $T := (N_T, \mathsf{lab}_T, \mathsf{succ}_T)$ where $N_T = \{\{p\} \mid p \in \mathcal{P}\mathrm{os}(t)\}$, and for each $\{p\} \in N_T$, $\mathsf{lab}_T(\{p\})$ is given by the symbol at position p in t, and $\mathsf{succ}_T^i(\{p\}) = \{p{\cdot}i\}$ for appropriate i. A standard induction on T gives $\mathsf{U}(T) = t$ as desired.

(2) Closure under reductions: By definition, $\mathcal{G}_c(\mathcal{F}, \mathcal{V})$ is closed under reductions.

(3) Preservation of reductions: For Assertion (1) we prove that $S \lhd\!\!\rightarrow_{\mathcal{G}} T$ implies $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{U}(T)$, whenever $\mathcal{R}$ is left-linear. For Assertion (2) we prove that $S \lhd\!\!\blacktriangleright_{\mathcal{G}} T$ implies $\mathsf{U}(S) \to_{\mathcal{R},p} \mathsf{U}(T)$.

Consider first Assertion (1), and assume $S \lhd\!\!\rightarrow_{L \to R,p} T$ for some rule $L \to R \in \mathcal{G}$. Thus $S \lhd_p^! U \longrightarrow_{L \to R,p} T$ for some term graph U, where

Lemma 5.10(1) gives that the node $u \in U$ addressed by p is not shared. By assumption on $\mathcal{G}$ only variables are shared in L, since by assumption that $\mathsf{U}(L \to R) \in \mathcal{R}$ is left-linear, it is not difficult to see that $L \in \triangle(\mathcal{F}, \mathcal{V})$. Hence U and L satisfy the preconditions of Lemma 4.50, we conclude $\mathsf{U}(U) \to_{\mathsf{U}(L\to R),p} \mathsf{U}(T)$ by Lemma 4.50(1). Since $\mathsf{U}(S) = \mathsf{U}(U)$ using Lemma 4.17 and $\lhd_p \subseteq \preccurlyeq$, and since $\mathsf{U}(L \to R) \in \mathcal{R}$ by definition, we obtain $\mathsf{U}(S) \to_{\mathcal{R}} \mathsf{U}(T)$ as desired.

Consider now Assertion (2), and assume $S \lhd_p^! U \blacktriangleright_p^! V \mathrel{\triangleleft\!\!\to}_{L\to R,p} T$ for some rule $L \to R \in \mathcal{G}$, and intermediate canonical graphs U and V. Then again Lemma 5.10(1) gives that the node $u \in U$ addressed by p is not shared. Using that $\blacktriangleright_p$ folds only nodes strictly below p, a standard induction gives that $u \in V$ is addressed by p, moreover it is not shared. Since by Lemma 5.10(2), $V|_p \in \triangledown(\mathcal{F}, \mathcal{V})$, the preconditions of Lemma 4.50 are met. We conclude $\mathsf{U}(S) = \mathsf{U}(U) = \mathsf{U}(V) \to_{\mathcal{R},p} \mathsf{U}(T)$, where the equalities follow by Lemma 4.17 as above.

(4) Simulation of reductions: Let $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ be a canonical term graph, and assume $\mathsf{U}(S) \to_{\mathsf{U}(L\to R),p} t$ for some position p, rule $\mathsf{U}(L \to R) \in \mathcal{R}$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. For Assertion (1) we prove $S \mathrel{\triangleleft\!\!\to}_{L\to R,p} T$ for some $T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ that unfolds to t, whenever $\mathcal{R}$ is left-linear. For Assertion (2) we prove $S \mathrel{\triangleleft\!\!\blacktriangleright\!\!\to}_{L\to R,p} T$ with $\mathsf{U}(T) = t$.

Consider canonical graphs U and V with $S \lhd_p^! U \blacktriangleright_p^! V$. By Lemma 5.10(2) both U and V exist, by Lemma 5.10(1) the node addressed by p is not shared in U, i.e., $u := \{p\} \in U$. Observe that $\mathsf{U}(S) = \mathsf{U}(U)$ by Lemma 4.17, and hence by assumption $\mathsf{U}(S) \to_{\mathsf{U}(L\to R),p} t$. Note that $L \in \triangle(\mathcal{F}, \mathcal{V})$ when $\mathcal{R}$ is left-linear. Lemma 4.50(2) gives a term graph T such that $U \longrightarrow_{L\to R,p} T$ and $\mathsf{U}(T) = t$. Overall we have $S \mathrel{\triangleleft\!\!\to}_{L\to R,p} T$ as required for Assertion (1).

Now consider the case when $\mathcal{R}$ is not necessarily left-linear. Observe $\mathsf{U}(S) = \mathsf{U}(U) = \mathsf{U}(V)$, hence $\mathsf{U}(V) \to_{\mathsf{U}(L\to R),p} t$ by assumption. Also, Lemma 5.10(2) gives $V|_p \in \triangledown(\mathcal{F}, \mathcal{V})$. Since $\blacktriangleright_p$ collapses nodes strictly below p only, we obtain that the node addressed by p in V is not shared, as in the term graph U. We conclude $V \longrightarrow_{L\to R} T$ for some term graph T with $\mathsf{U}(T) = t$ by Lemma 4.50(2). Overall $S \lhd_p^! U \blacktriangleright_p^! V \longrightarrow_{L\to R} T$, i.e., $S \mathrel{\triangleleft\!\!\blacktriangleright\!\!\to}_{L\to R,p} T$, as desired. □

5.3. Adequacy for Innermost Rewriting

We now refine the adequacy theorem from full rewriting to innermost rewriting.

Definition 5.13 (Innermost Graph Rewrite Relation)**.** Let $\mathcal{G}$ be a GRS. We define $S \xrightarrow{\mathsf{i}}_{\mathcal{G},L\to R,u} T$ if $S \longrightarrow_{L\to R,u} T$ and S is not $\mathcal{G}$-reducible at any node v strictly below u. We set $S \xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ if $S \xrightarrow{\mathsf{i}}_{\mathcal{G},L\to R,u} T$ holds for some graph rewrite rule $L \to R \in \mathcal{G}$ and node $u \in S$. We call $\xrightarrow{\mathsf{i}}_{\mathcal{G}}$ the *innermost graph rewrite relation* induced by $\mathcal{G}$. We also write $S \xrightarrow{\mathsf{i}}_{\mathcal{G},u} T$ or $S \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ where $p \in u \in S$ to indicate the redex node or a position p that addresses u in S respectively.

Throughout the following, we consider a graph rewrite system $\mathcal{G}$ that introduces sharing only under variable positions in right-hand sides, that is, $R \in \diamondsuit^{\mathcal{V}}(\mathcal{F}, \mathcal{V})$ for every $L \to R \in \mathcal{G}$. For such graph rewrite systems, an innermost graph rewrite step $S \xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ introduces sharing only on irreducible nodes in T. This allows us to keep the invariant that only nodes that are irreducible are possibly shared, rendering the relation $\lhd_p$ superfluous.

Definition 5.14. Let $\mathcal{G}$ denote a graph rewrite system. For canonical term graphs S and T we define

$$S \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} T \quad :\Longleftrightarrow \quad S \blacktriangleright_p^! \cdot \longrightarrow_{\mathcal{G},p} T \;,$$

for some $p \in \mathcal{P}\mathrm{os}(S)$.

Let S be a term graph. We say that S is $\mathcal{R}$-*reducible* at position p, if $\mathsf{U}(S|_p) = l\sigma$ for some rule $l \to r \in \mathcal{R}$. Otherwise S is called $\mathcal{R}$-*irreducible* at position p. The set $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ thus refers to the set of canonical term graphs where nodes addressed by $\mathcal{R}$-reducible positions are not shared. The adequacy theorem for innermost rewriting given below states that $\blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}$ is adequate for $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ on $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, where $\mathcal{R}$ denotes the unfolding of $\mathcal{G}$. Central in the proof of this adequacy theorem is to show that $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ is closed under under $\blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}$ reductions. The next lemma serves as a preparatory step, and shows that $\blacktriangleright_p$ as integrated in $\blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}$ does not harm this closure property.

Lemma 5.15. *Let $\mathcal{G}$ be a GRS, let $\mathcal{R} := \mathsf{U}(\mathcal{G})$ and suppose $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$. If*

$$S \blacktriangleright_p^! S' \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T \;,$$

then $S' \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$. Moreover, S' is $\mathcal{R}$-irreducible at every position q strictly below p.

Proof. Consider

$$S \blacktriangleright_p^n S' \xrightarrow{\mathsf{i}}_{L \to R,p} T \;,$$

where S' is minimal with respect to $\blacktriangleright_p$. By Lemma 5.10(2), $S'|_p \in \bigtriangledown_c(\mathcal{F}, \mathcal{V})$. Since there is no redex-node below p in S' by assumption, the contraposition of Lemma 4.49 gives that every proper sub-graph of $S'|_p$ is $\mathcal{R}$-irreducible.

We prove $S' \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ by induction on n. By the assumption $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ it suffices to consider the inductive step. Suppose $S \blacktriangleright_p^n S'' \blacktriangleright_p S'$, by induction hypothesis $S'' \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ holds. Then $S'' \sqsupseteq_{v_2}^{v_1} S'$ for some nodes $v_1, v_2 \in S''$ strictly below position p in S''. Consider a shared node $u \in S'$. We have to prove $\mathsf{U}(S'|_u) \in \mathrm{NF}(\mathcal{R})$. For $u \in \{v_1, v_2\}$ this is clear by the first half of the proof. For $u \notin \{v_1, v_2\}$ we have that $u \in S''$ is also shared in S''. We conclude from the induction hypothesis, using Lemma 4.17. □

Theorem 5.16 (Adequacy for Innermost Rewriting)**.** *Let $\mathcal{G}$ be a graph rewrite system such that only variable nodes are shared in left-hand and right-hand sides of $\mathcal{G}$, i.e., $L \in \diamondsuit^{\mathcal{V}}(\mathcal{F}, \mathcal{V})$ and $R \in \diamondsuit^{\mathcal{V}}(\mathcal{F}, \mathcal{V})$ for every rule $L \to R \in \mathcal{G}$. Let $\mathcal{R} := \mathsf{U}(\mathcal{G})$ denote the unfolding of $\mathcal{G}$.*

(1) *If $\mathcal{R}$ is left-linear, the relation $\xrightarrow{\mathsf{i}}_{\mathcal{G}}$ is adequate on $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$ for $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.*

(2) *The relation $\blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}$ is adequate on $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$ for $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.*

Proof. We prove the four properties of adequacy.

(1) Surjectivity of unfolding on $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$: The set $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$ contains all trees. The property follows from this as in Theorem 5.12.

(2) Closure under reductions of $\diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$: As a preparatory step, consider $S \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ for term graph $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$, but additionally assume that all sub-graphs of $S\restriction_p$ unfold to $\mathcal{R}$ normal forms. We show that $T \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$. Consider the pre-reduction step $U \hookrightarrow_{L\to R,p} V$ for $L \to R \in \mathcal{G}$ underlying $\mathcal{C}(U) = S \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T = \mathcal{C}(V)$. By Lemma 4.25 we have $U \cong S$ which gives $U \in \diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$. To conclude $T \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$, we show $V \in \diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$.

Consider first the term graph R_m obtained by applying the matching morphism $m : L \to_{\mathcal{V}} U\restriction_p$ to the right-hand side R. Let $\{v_1,\dots,v_k\} := N_L \cap N_R$ denote the nodes common to left-hand and right-hand side in $L \to R$. By assumption, these correspond to the set of variable nodes in R. We show $R_m \in \diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$, i.e., for every shared node $u \in R_m$, the sub-graph $R_m\restriction_u$ unfolds to a normal form of $\mathcal{R}$. Recall that

$$R_m = (R \oplus U)[m(v_1),\dots,m(v_k) \leftarrow v_1,\dots,v_k]\restriction_{\underline{m}(\mathrm{rt}(R))} .$$

Using $R \in \diamondsuit^{\mathcal{V}}(\mathcal{F},\mathcal{V})$, it is obvious that if $u \in R$ then u is not shared in R_m. Hence suppose $u \in U$, and by definition of R_m the node u occurs in R_m below a node $m(v_i)$ for some variable node v_i ($i \in \{1,\dots,k\}$). So in particular, the graph $R_m\restriction_u$ is a sub-graph of U below the node $m(v_i) \in U$. Since the matching morphism $m : L \to_{\mathcal{V}} U\restriction_p$ maps the variable node v_i to a node strictly below p in U, we see that $R_m\restriction_u$ occurs in U strictly below the rewrite position p. Thus this sub-graph in U corresponds to a $\mathcal{R}$ normal form by the additional assumption on S, due to Lemma 4.19(2).

Now consider $V = U[R_m]_p$. We prove that $V \in \diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$. It suffices to consider the case

$$V = (U \oplus R_m)[u \leftarrow \mathrm{rt}(R_m)]\restriction_{\mathsf{rt}(U)} ,$$

where u denotes the redex-node addressed by $p \neq \epsilon$ in U. Fix a node $v \in V$. We show that either v is not shared, or the sub-graph $V\restriction_v$ unfolds to a normal form of $\mathcal{R}$.

Suppose first $v \in R_m$. If even $v \in U$ holds, then $\mathsf{U}(R_m\restriction_v) = \mathsf{U}(U\restriction_v) \in \mathrm{NF}(\mathcal{R})$ follows as above. Otherwise $v \in R$, and v is unshared in R. Hence v is also unshared in V since the redex node u is unshared in U. The latter follows by the assumption $U \in \diamondsuit^{\mathrm{NF}(\mathcal{R})}(\mathcal{F},\mathcal{V})$, and using that $\mathsf{U}(U\restriction_u) = \mathsf{U}(S\restriction_p) \notin \mathrm{NF}(\mathcal{R})$ holds as a consequence of Lemma 4.43.

Consider now the remaining case where $v \in U$ but $v \notin R_m$. Suppose v is shared in V. This assumption gives $V|_v = U|_v$, as otherwise v is above the unshared redex node u in U. By construction of R_m and the assumption $u \notin R_m$, the node v is not reachable in V from a node that also occurs in R_m. We conclude that the node v is also a shared node in U. Hence the assumption $U \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ gives $\mathsf{U}(V|_v) = \mathsf{U}(U|_v) \in \mathrm{NF}(\mathcal{R})$ as desired. This concludes the preparatory step.

Suppose now $S \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ with $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$. We consider first the case where $\mathcal{R}$ is left-linear. Then by assumption we even have $L \in \triangle(\mathcal{F}, \mathcal{V})$, for all $L \to R \in \mathcal{G}$. As by assumption all proper sub-graphs of $S|_p$ are irreducible, the contraposition of Lemma 4.49 gives that every proper sub-graph of $S|_p$ unfolds to an $\mathcal{R}$ normal form. By the preparatory step we conclude thus $T \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ as desired.

Finally, suppose $S \mathrel{\blacktriangleright}\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ with $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$. Then we conclude $T \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ using the preparatory step in combination with Lemma 5.15.

(3) Preservation of reductions: For Assertion (1) we prove that for term graphs $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, $S \xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ implies $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{U}(T)$, whenever $\mathcal{R}$ is left-linear. For Assertion (2) we prove that if $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, then $S \mathrel{\blacktriangleright}\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ implies $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{U}(T)$.

Consider first Assertion (1), and assume $S \xrightarrow{\mathsf{i}}_{L \to R,p} T$ for $L \to R \in \mathcal{G}$ for $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$. Lemma 4.43 gives that S is $\mathcal{R}$-reducible at position p, using that $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ we conclude thus that the node addressed by p is not shared. If $\mathcal{R}$ is left-linear, $L \in \triangle(\mathcal{F}, \mathcal{V})$, and $\mathsf{U}(S) \to_{\mathcal{R}} \mathsf{U}(T)$ follows by Lemma 4.50(1). Consider a position q strictly below p. Since by assumption S is $\mathcal{G}$-irreducible at q, using $L \in \triangle(\mathcal{F}, \mathcal{V})$ the contraposition of Lemma 4.49 gives that S is $\mathcal{R}$-irreducible at q, that is $\mathsf{U}(S|_q) = \mathsf{U}(S)|_q \in \mathrm{NF}(\mathcal{R})$. We conclude $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{U}(T)$.

Consider now $S \blacktriangleright^!_p S' \xrightarrow{\mathsf{i}}_{L \to R,p} T$ for $L \to R \in \mathcal{G}$. Using Lemma 5.10(2) and Lemma 5.15 to satisfy the assumptions of Lemma 4.50(2), we again obtain $\mathsf{U}(S') \longrightarrow_{\mathcal{G}} \mathsf{U}(T)$. Since by Lemma 5.15 the term graph S' is $\mathcal{R}$-irreducible at every position strictly below p, this can be strengthened to $\mathsf{U}(S') \xrightarrow{\mathsf{i}}_{\mathcal{R},p} \mathsf{U}(T)$. Lemma 4.17 gives $\mathsf{U}(S) = \mathsf{U}(S')$. Summing up, $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{U}(T)$ holds as desired.

(4) Simulation of reductions: Assume $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R},p} t$ for $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, position p and term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. For Assertion (1) we prove $S \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ for some $T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ that unfolds to t, whenever $\mathcal{R}$ is left-linear. For Assertion (2) we prove $S \mathrel{\blacktriangleright}\!\xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ with $\mathsf{U}(T) = t$.

By the assumptions, the node addressed by position p in S is not shared. Consider first the case when $\mathcal{R}$ is left-linear. Then $S \longrightarrow_{\mathcal{G},p} T$ with $\mathsf{U}(T) = t$ holds by Lemma 4.50(2). Observe that S is not $\mathcal{G}$-reducible at any position q strictly below p. Suppose otherwise. Then Lemma 4.43 show that term graph S is $\mathcal{R}$-reducible at position q, i.e., $\mathsf{U}(S|_q) \notin \mathrm{NF}(\mathcal{R})$. Using Lemma 4.19 this contradicts $\mathsf{U}(S) \xrightarrow{\mathsf{i}}_{\mathcal{R},p} t$. Hence $S \xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ follows.

For the general case, consider $S' \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ with $S \blacktriangleright^!_p S'$. Then $S' \longrightarrow_{\mathcal{G},p} T$ with $\mathsf{U}(T) = t$ holds by Lemma 4.50(2), if $\mathsf{U}(S') = \mathsf{U}(S)$, $S'|_p \in \triangledown_c(\mathcal{F}, \mathcal{V})$, and the node u addressed by p is unshared in S'. The first two requirements are met using Lemma 4.17 and Lemma 5.10(2) respectively. Observe that since S is $\mathcal{R}$-reducible at position p and $S \in \diamondsuit_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$, the node addressed by p is unshared in S. Thus the third requirement follows by definition of $\blacktriangleright_p$. Using Lemma 4.43, we can strengthen $S' \longrightarrow_{\mathcal{G},p} T$ to $S' \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$. In total we obtain $S \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} T$ with $\mathsf{U}(T) = t$. □

Chapter 6.

An Implementation of Graph Rewriting

We now provide an implementation of graph rewrite reductions on Turing machines, and estimate the complexity of this implementation. As a first step toward this goal, in the next section we relate sizes of intermediate terms to the size of starting terms and the length of derivations. In Section 6.2 we then provide the actual implementation.

6.1. An Upper Bound on Sizes of Reducts

Opposed to term rewriting, graph rewriting induces linear size growth in the length of derivations.

Lemma 6.1. *Let $\mathcal{G}$ denote a GRS and let $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ be canonical graphs. There exists a constant $\Delta_\mathcal{G}$ depending only on $\mathcal{G}$ such that, if $S \longrightarrow_\mathcal{G} T$ then (i) $|T| \leqslant |S| + \Delta_\mathcal{G}$ and (ii) $\mathsf{dp}(T) \leqslant \mathsf{dp}(S) + \Delta_\mathcal{G}$.*

Proof. Set $\Delta_\mathcal{G} := \{|R| \mid L \to R \in \mathcal{G}\}$. Consider first a pre-reduction step $U \hookrightarrow_{L\to R,u} V$. Then every node $v \in V$ occurs either in U or in R, and consequently $|V| \leqslant |U| + \Delta_\mathcal{G}$ and $\mathsf{dp}(V) \leqslant \mathsf{dp}(U) + \Delta_\mathcal{G}$. For the latter inequality we use that $\Delta_\mathcal{G}$ also binds the depth of R.

The term graphs U and $\mathcal{C}(U)$ are isomorphic by Lemma 4.25, that is they are connected by an bijective morphism. Thus $|\mathcal{C}(U)| = |U|$ as well as $\mathsf{dp}(\mathcal{C}(U)) = \mathsf{dp}(U)$ holds. For the same reason, $|\mathcal{C}(V)| = |V|$ and $\mathsf{dp}(\mathcal{C}(V)) = \mathsf{dp}(V)$. By definition of $\longrightarrow_\mathcal{G}$ the bounds on pre-reduction steps give $|T| \leqslant |S| + \Delta_\mathcal{G}$ and $\mathsf{dp}(T) \leqslant \mathsf{dp}(S) + \Delta_\mathcal{G}$ for all canonical term graphs $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ with $S \longrightarrow_\mathcal{G} T$. □

Using our estimation on the length of restricted folding and unfolding from Lemma 5.8, we can lift Lemma 6.1 to the relations used in our adequacy theorems, and below to complete sequences.

Lemma 6.2. *Let $\mathcal{G}$ denote a GRS and let $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$. There exists a constant $\Delta_\mathcal{G}$ depending only on $\mathcal{G}$ such that:*

(1) *if $S \longrightarrow T$ for $\longrightarrow \in \{\trianglelefteq\!\!\hookrightarrow_\mathcal{G}, \trianglelefteq\!\!\blacktriangleright\!\!\hookrightarrow_\mathcal{G}\}$ then $|T| \leqslant |S| + \mathsf{dp}(S) + \Delta_\mathcal{G}$; and*

(2) *if $S \longrightarrow T$ for $\longrightarrow \in \{\xrightarrow{\mathsf{i}}_\mathcal{G}, \blacktriangleright\!\xrightarrow{\mathsf{i}}_\mathcal{G}\}$ then $|T| \leqslant |S| + \Delta_\mathcal{G}$.*

Proof. Consider first the sub-case $S \mathrel{\triangleleft\!\!\blacktriangleright\!\!\to}_{\mathcal{G}} T$ of Assertion (1), thus for some position $p \in \mathcal{P}\mathrm{os}(S)$ and canonical term graphs $U, V \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$,

$$S \mathrel{\triangleleft}^!_p U \mathrel{\blacktriangleright}^!_p V \longrightarrow_{L \to R, p} T \; .$$

We obtain

$$\begin{aligned} |T| &\leqslant |V| + \Delta_{\mathcal{G}} && \text{by Lemma 6.1} \\ &\leqslant |U| + \Delta_{\mathcal{G}} && \text{by Lemma 5.8(2)} \\ &\leqslant |S| + |p| + \Delta_{\mathcal{G}} && \text{by Lemma 5.8(1).} \end{aligned}$$

As $p \in \mathcal{P}\mathrm{os}(S)$ corresponds to an (acyclic) path in S, we obtain $|p| \leqslant \mathsf{dp}(S)$, and conclude the analysed sub-case. The sub-case $S \mathrel{\triangleleft\!\!\to}_{\mathcal{G}} T$ follows by identical reasoning. Finally, Assertion (2) follows either directly from Lemma 6.1, or using additionally Lemma 5.8(2). □

Lemma 6.3. *Let $\mathcal{G}$ denote a GRS and let $S, T \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$. There exists a constant $\Delta_{\mathcal{G}}$ depending only on $\mathcal{G}$ such that:*

(1) *if $S \longrightarrow^{\ell} T$ for $\longrightarrow \in \{\mathrel{\triangleleft\!\!\to}_{\mathcal{G}}, \mathrel{\triangleleft\!\!\blacktriangleright\!\!\to}_{\mathcal{G}}\}$ then $|T| \leqslant (\ell + 1) \cdot |S| + \ell^2 \cdot \Delta_{\mathcal{G}}$; and*

(2) *if $S \longrightarrow^{\ell} T$ for $\longrightarrow \in \{\xrightarrow{\mathsf{i}}_{\mathcal{G}}, \mathrel{\blacktriangleright\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{G}}\}$ then $|T| \leqslant |S| + \ell \cdot \Delta_{\mathcal{G}}$.*

Proof. The only interesting case is Assertion (1), Assertion (2) follows by a straight forward induction on ℓ from Lemma 6.2(2). Let $\longrightarrow \in \{\mathrel{\triangleleft\!\!\to}_{\mathcal{G}}, \mathrel{\triangleleft\!\!\blacktriangleright\!\!\to}_{\mathcal{G}}\}$, we prove the property by induction on ℓ. The base case follows trivially, so suppose the lemma holds for ℓ, we establish the lemma for $\ell + 1$. Consider a derivation $S \longrightarrow^{\ell} T \longrightarrow U$. By induction hypothesis, $|T| \leqslant (\ell+1) \cdot |S| + \ell^2 \cdot \Delta_{\mathcal{G}}$. A standard induction gives $\mathsf{dp}(T) \leqslant \mathsf{dp}(S) + \ell \cdot \Delta_{\mathcal{G}}$, using Lemma 6.1 for the inductive step. By Lemma 6.2(1) we see $|U| \leqslant |T| + \mathsf{dp}(T) + \Delta_{\mathcal{G}}$, and by definition we have $\mathsf{dp}(S) \leqslant |S|$. Summing up

$$\begin{aligned} |U| &\leqslant |T| + \mathsf{dp}(T) + \Delta_{\mathcal{G}} \\ &\leqslant \big((\ell + 1) \cdot |S| + \ell^2 \cdot \Delta_{\mathcal{G}}\big) + \big(\mathsf{dp}(S) + \ell \cdot \Delta_{\mathcal{G}}\big) + \Delta_{\mathcal{G}} \\ &\leqslant (\ell + 2) \cdot |S| + \ell^2 \cdot \Delta_{\mathcal{G}} + \ell \cdot \Delta_{\mathcal{G}} + \Delta_{\mathcal{G}} \\ &\leqslant (\ell + 2) \cdot |S| + (\ell + 1)^2 \cdot \Delta_{\mathcal{G}} \; , \end{aligned}$$

concludes the lemma. □

6.2. Implementing a Graph Rewriting Reduction

We now provide an implementation of graph rewriting on TMs. When we say *computable in time* below, we implicitly mean computable in deterministic time on a k-string TM M as given in Definition 2.10. When not mentioned otherwise, we suppose that M is deterministic. To allow for a succinct encoding, we represent a canonical term graph S as an isomorphic term graph $U \in \mathcal{G}_{\mathbb{N}}(\mathcal{F}, \mathcal{V})$ with nodes in $N_U = \{1, \ldots, |S|\}$. This is justified, as the structure of each node $u \in S$, i.e., the set of positions addressing u in S, is already implicitly contained

in U. This is witnessed by the function $\mathcal{C}$ that translates any such graph U back to its canonical form S. Term graphs U of this form are called *encoding term graphs* (of S) below.

Natural numbers $n \in \mathbb{N}$ are encoded as binary words $\ulcorner n \urcorner$. For each function symbol $f \in \mathcal{F}$, we suppose that a corresponding tape-symbols $\ulcorner f \urcorner$ is present. Further we fix an enumeration of variables, where the i^{th} variable x_i is denoted by $i \in \mathbb{N}$, that is, we set $\ulcorner x_i \urcorner := \ulcorner i \urcorner$. To represent an encoding term graph U of a canonical term graph S as word, we use an *adjacency list*. For each node $u \in \{1, \ldots, |S|\}$ we additionally store the label $\mathsf{lab}_U(u)$, and keep the order on successors. More precise, the encoding of U is given by a list containing for each $u \in U$ a triple

$$\langle \ulcorner u \urcorner, \ulcorner \mathsf{lab}_S(u) \urcorner, [\ulcorner u_1 \urcorner, \ldots, \ulcorner u_k \urcorner] \rangle \ ,$$

where $\mathsf{succ}_U(u) = [u_1, \ldots, u_k]$. Following the terminology of [75], this tuple is also called the *node-specification* of u in S. For performance reasons, we suppose that the node-specification of the root node is distinguished, and hence the root of U can be determined in linear time by scanning U once.

In the encoding term graph U of S, a single node is stored in space $\lceil \log(|S|) \rceil$. For fixed $\mathcal{F}$, the length of $\mathsf{succ}_S(u)$ is bounded from above by a constant. Thus a single node specification is stored in space $c \cdot \lceil \log(|S|) \rceil$ for $c \in \mathbb{N}$ depending only on $\mathcal{F}$. This motivates the following definition.

Definition 6.4 (Representation Size)**.** Let $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ be a canonical graph. We define $\|S\| := \lceil \log(|S|) \rceil \cdot |S|$, and call $\|S\|$ the *representation size* of S.

Before we investigate into the computational complexity of reductions, we prove some auxiliary lemmas. Modifications performed on encoding term graph below often destroy the property that the set of nodes consists of a continuous sequence of natural numbers as assumed above. The next lemma gives a quadratic algorithm that turns such modified graphs into isomorphic encoding term graphs.

Lemma 6.5. *The function that maps a graph $S \in \mathcal{G}_{\mathcal{N}}(\mathcal{F}, \mathcal{V})$, given as a list of node-specifications, to an isomorphic encoding graph S' is computable in time quadratic in the size of the input.*

Proof. We can define a TM M which traverses over S and replace every encountered node $u \in S$ by a node $m(u)$. For the so obtained term graph S', the function $m : N_S \to \{1, \ldots, |S|\}$ is a graph morphism witnessing $S \cong S'$. The morphism and the graph S' can be constructed on the fly. For that, the TM M uses a dedicated working tape that stores the morphism m as an association list. Clearly the size of m is linearly bounded in the size of the input graph. Thus lookup of $m(u)$, if defined, is a linear operation in the size of the input. If $m(u)$ is undefined, it is initialised by the next natural number not assigned so far (initially this number is 1). This initialisation can again be performed in time linear in the size of the input.

It is clear that when the complete graph S has been copied, m gives the desired morphism, and S' is an encoding graph. As each lookup in m is a linear operation, and the total number of lookups is bounded by $|S|$, overall M operates in quadratic time. □

Sometimes it is convenient to have a specific sub-graph of S marked in-place. By enriching the alphabet underlying the encoding of S, we can keep this marking stored directly in the node.

Lemma 6.6. *For each position $p \in \mathcal{P}\mathrm{os}(S)$, the sub-graph $S\restriction_p$ of $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ can be marked in quadratic time in $\|S\|$.*

Proof. Suppose the TM M holds the graph S and position p on its input tape, where S is given by its encoding graph, and the position p is given as a list of binary numbers. The term graph S is first copied to a working tape, where the marking is performed in-place. The TM M traverses the term graph then in a depth-first manner, starting from the node addressed by p. For the depth first traversal, it uses an additional working tape, which holds a *stack* of nodes not marked so far. Initially, this stack is populated by the node addressed by p only.

To determine the node addressed by p, it traverses along the path

$$\mathrm{rt}(S) \xrightarrow{i_1}_S u_1 \xrightarrow{i_2}_S \cdots \xrightarrow{i_n}_S u_n \ ,$$

induced by p, i.e., $p = i_1 \cdot i_2 \cdot \cdots i_n$. By scanning through S, the root $\mathrm{rt}(S)$ can be found in time linear in $\|S\|$. Also, given u_i $(i = 1, \ldots, n-1)$, the node u_{i+1} can be found in time linear in $\|S\|$. As a consequence, the node addressed by p can be determined in time quadratic in $\|S\|$.

Repeatedly, the TM M pops the top node u written on the stack, and pushes the successors of u on the stack, if u was not marked before. When the stack is finally empty, the resulting marked term graph is copied to the output-tape. Note that M performs at most $|S\restriction_p| \leqslant \|S\|$ iterations. Consider an iteration with $\mathsf{succ}_S(u) = [u_1, \ldots, u_k]$ for the top node u on the stack. A single iteration requires at most $k+1$ scans of the graph S. Since k is fixed by the signature, one iteration takes thus at most time $\mathcal{O}(\|S\|)$. As the size of the graph on the working tape is unmodified, copying to the output tape takes also only time $\mathcal{O}(\|S\|)$. Summing up, M takes overall $\mathcal{O}(\|S\|^2)$ steps until it halts. □

Lemma 6.7 (Complexity of Matching)**.** *Let $L, S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ be two canonical term graphs.*

(1) *In time $\mathcal{O}(|L| \cdot (\|L\| + \|S\|))$ it can be decided if there exists a matching morphism $m : L \to_{\mathcal{V}} S$;*

(2) *A matching morphism $m : L \to_{\mathcal{V}} S$, if it exists, can be computed under the same bound.*

Proof. For two mapping m_1 and m_2 from nodes N_1 and N_2 to nodes in S, the composition $m_1 \oplus m_2$ is given as follows. If m_1 and m_2 are *incompatible* in the sense that $m_1(u) \neq m_2(u)$ for some $u \in N_1 \cap N_2$, then $m_1 \oplus m_2 = \bot$. Otherwise, $m_1 \oplus m_2$ is a mapping on $N_1 \cup N_2$, given by $m_1 \oplus m_2(u) = m_1(u)$ if $u \in N_1$ and $m_1 \oplus m_2(u) = m_2(u)$ if $u \in N_2$. Finally, $\oplus$ is extended to $\bot$ so that $\bot \oplus m = \bot$ and $m \oplus \bot = \bot$.

We construct a TM M that, starting from the roots of L and S, checks the morphism conditions recursively. Moreover, the witnessing morphism m

is constructed on the fly. For this, M uses a dedicated working tape in order to store the constructed morphism m as an association list (or $\bot$ respectively). Initially m is the empty mapping. The encoding graphs L is copied to a dedicated working tapes, which in analogy to Lemma 6.6 allows the marking of visited nodes in L. The TM M simultaneously traverses the copy of L and S, starting from $\mathrm{rt}(L)$ and $\mathrm{rt}(S)$. For that it keeps a stack of pairs $(u, v) \in N_L \times N_S$ on a dedicated working tape, initialised by the pair $(\ulcorner \mathrm{rt}(L) \urcorner, \ulcorner \mathrm{rt}(S) \urcorner)$. It then iterates the following procedure.

The machine M searches the nodes $u \in L$ and $v \in S$, for $(\ulcorner u \urcorner, \ulcorner v \urcorner)$ the topmost pair on the stack. It overwrites m by $m \oplus \{\ulcorner u \urcorner \mapsto \ulcorner v \urcorner\}$. If $\mathsf{lab}_L(u) \in \mathcal{V}$ or u was already visited, it marks $u \in L$ visited. Otherwise the machine checks $\mathsf{lab}_L(u) = \mathsf{lab}_S(v)$. Should this check fail, the machine overwrites m by $\bot$. Otherwise, it pushes all pairs of successors $(\ulcorner u_i \urcorner, \ulcorner v_i \urcorner)$ (where $\mathsf{succ}^i_L(u) = u_i$ and $\mathsf{succ}^i_S(v) = v_i$) on the stack and repeats.

To see that the TM acts as intended, first suppose $\bot$ is written on the working tape. Then for pairs $(\ulcorner u \urcorner, \ulcorner v \urcorner)$ of nodes $u \in L$ and $v \in S$ on the stack, either $m' \oplus \{u \mapsto v\} = \bot$ for some intermediate map m', or $\mathsf{lab}_L(u) \neq \mathsf{lab}_S(v)$. Observe that $m' \oplus \{u \mapsto v\}$ can be conceived as the combination of a subset of necessary morphism conditions. Under our assumption, it thus follows that the morphism conditions are inconsistent, in the former case since $u \in L$ should be mapped to two distinct nodes v and $m'(u) = v'$. In the latter case the morphism breaks due to the labeling condition. It follows that the implementation is complete. Now suppose $m \neq \bot$ is written on the working tape. Then by definition of $\oplus$, m is a function from N_L to N_S, i.e., every node $u \in L$ is mapped to exactly one node $v \in S$. By construction, this function obeys the root condition, i.e., $m(\mathrm{rt}(L)) = \mathrm{rt}(S)$, and the morphism conditions for all $u \in L$. The function m thus constitutes a matching morphism $m : L \to_{\mathcal{V}} S$. It follows that the implementation is sound.

It remains to verify that M runs under the given bound. It is clear that initialisation causes no harmful overhead. Observe that due to marking of visited nodes, the number of iterations is bounded by $|L|$. At each iteration, M scans through L and S, and possibly modifies the stack and the morphism m constructed so far. The scan of L and S takes time linear in $\|L\| + \|S\|$, and also modification of the stack can be performed under this bound. As observed before, at any time the constructed morphism m is either $\bot$ or a function from (a subset of) nodes in L to S. Using a reasonable encoding of lists and tuples, the size of m on the tape is thus asymptotically bounded by $\|m\| := |L| \cdot (\lceil \log(|L|) \rceil + \lceil \log(|S|) \rceil)$. It is not difficult to see that updating m be done in time $\mathcal{O}(\|m\|) \subseteq \mathcal{O}(\|L\| + \|S\|)$. So overall an iteration is performed in time $\mathcal{O}(\|L\| + \|S\|)$, as there are at most $|L|$ iterations the execution time is thus asymptotically bounded by $|L| \cdot (\|L\| + \|S\|)$.

Lemma 6.8 (Complexity of a Rewrite Step)**.** *Let $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$, let $L \to R$ be a graph rewrite rule and let $p \in \mathcal{P}\mathrm{os}(S)$ be a position. If $S \longrightarrow_{L \to R, p} T$ then T is computable in time $\mathcal{O}(\|L\|^2 \cdot \|R\|^2 \cdot \|S\|^2)$.*

Proof. We construct a TM M that given S, p and $L \to R$, computes T with $S \longrightarrow_{L\to R,p} T$, if it exists. The machine will perform a pre-reduction step $S' \hookrightarrow_{L\to R,u} T'$, where nodes in S' are obtained by renaming nodes in S to $m(u) := u + |R|$. This ensures that $N_{S'} \cap N_R = \varnothing$. Lemma 4.36 states that S' is as good as any other term graph isomorphic to S. An adaption of the renaming algorithm underlying Lemma 6.5 shows that S' can be computed from S in time $\mathcal{O}((\|S\| + \|R\|)^2)$.

By the construction of Lemma 6.6, the TM M can first mark the sub-graph $S'|_p$ in time $\|S'\|^2$, and uses the machine of Lemma 6.7 to construct a matching morphism $m : L \to_{\mathcal{V}} S'|_p$. If this construction fails, that is the rule is not applicable, the machine stops. The execution time of this step can be bounded by $\mathcal{O}(\|L\|^2 \cdot \|S\|^2)$.

Otherwise, the machine appends R to S', and points all edges going to $v \in N_L \cap N_R$ to $m(v) \in S'$. Finally, the edges going to the redex-node $u \in S'$ are redirected to the root of R, if u was not the root of S'. Using the marking algorithm from Lemma 6.6, T' is obtained from the constructed graph by marking the corresponding reachable sub-graph, and removing unmarked nodes. Finally, the encoding graph of T is written on the output tape, employing Lemma 6.5 on the intermediate graph T'.

Observe that the encoding length of S' is bounded by $\|S\| \cdot \|R\|$, similar T' occupies at most $\mathcal{O}(\|S\| \cdot \|R\|)$ cells on the tape, at any time. Note that the construction of T' involves copying R, and performing $|R|$ redirects, where a single redirect can be performed in time linear in the encoding length of T'. By the estimation on T', marking the reachable sub-graph takes time $\mathcal{O}((\|S\|\cdot\|R\|)^2)$. Cleanup is again quadratic by Lemma 6.5. Summing up all bounds, we conclude the lemma. □

Lemma 6.9 (Complexity of Unfolding)**.** *Let $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ and let $p \in \mathcal{P}\mathrm{os}(S)$ be a position. A term graph T such that $S \lhd^!_p T$ is computable in time $\mathcal{O}(\|S\|^2)$.*

Proof. We construct a TM that given term graph S produces a term graph T such that $S \lhd^!_p T$ in time quadratic in $\|S\|$. For that, the machine traverse S along the path induced by p and introduces a fresh copy for each shared node encountered along that path. The machine has four working tapes at hand. On the first tape, the graph S is copied. Nodes in S are padded sufficiently by leading zero's, so that successors can be replaced by fresh nodes $u \leqslant 2 \cdot |S|$. The graph represented on the first tape is called the *current graph*, its size will be bound by $\mathcal{O}(\|S\|)$ at any time. On the second tape the position p, encoded as list of argument positions, is copied. The argument position referred by the tape-pointer is called *current argument position* and initially set to the first position. The third tape holds the *current node*, initially the root $\mathrm{rt}(S)$ of S. Finally, the remaining tape holds the size of the current graph in binary, the *current size*. One easily verifies that these preparatory steps can be done in time linear in $\|S\|$.

The TM now iterates the following procedure, until every argument position in p was considered. Let v be the current node, let S_i the current graph and let i be the current argument position. The machine keeps the invariant that v is not

shared in S_i. First, the node v_i with $v \xrightarrow{i}_{S_i} v_i$ in S_i is determined in time linear in $\|S\|$, the current node is replaced by v_i. Further, the pointer on the tape holding p is advanced to the next argument position. Since v is not shared, v_i is shared if and only if $v_i \in \mathsf{succ}_{S_i}(u)$ for $u \neq v$. The machine checks whether v_i is shared in the current graph. By the above observation this can be done in time linear in $\|S\|$. If v_i is not shared, the machine enters the next iteration. Otherwise, the node v_i is cloned in the following sense. First, the i-th successor v_i of v is replaced by a fresh node u. The fresh node is obtained by increasing the current node by one, this binary number is used as fresh node u. Further, the node specification $\langle \ulcorner u \urcorner, \ulcorner \mathsf{lab}_{S_i}(v_i) \urcorner, \ulcorner \mathsf{succ}_{S_i}(v_i) \urcorner \rangle$ is appended to the current graph S_i. Call the resulting graph S_{i+1}. Then $S_i \overset{u}{\underset{v_i}{\sqsubset}} S_{i+1}$ with $\mathcal{P}\mathrm{os}_{S_{i+1}}(u) = \{q\}$ and $q \leqslant p$, i.e., $S_i \lhd_p S_{i+1}$, compare the construction in Lemma 5.10(1).

When the procedure stops, the machine has computed $S = S_0 \lhd_p S_1 \lhd_p \ldots \lhd_p S_n = T$. One easily verifies that S_n is $\lhd_p$-maximal as every considered node along the path p is not shared. Each iteration takes time linear in $\|S\|$. As as at most $|p| \leqslant |S|$ iterations have to be performed, we obtain the desired bound. □

Lemma 6.10 (Complexity of Folding)**.** *Let $S \in \mathcal{G}_{\mathcal{C}}(\mathcal{F}, \mathcal{V})$ and let $p \in \mathcal{P}\mathrm{os}(S)$ be a position. A term graph T such that $S \blacktriangleright_p^! T$ is computable in time $\mathcal{O}(\|S\|^2)$.*

Proof. Define the *height* $\mathsf{ht}_U(u)$ of a node u in a term graph U inductively as follows: $\mathsf{ht}_U(u) := 0$ if $\mathsf{succ}_U(u) = []$ and $\mathsf{ht}_U(v) := 1 + \max_{v \in \mathsf{succ}_U(u)} \mathsf{ht}_U(v)$ otherwise. We drop the reference to the graph U when referring to the height of nodes in the analysis of the normalising sequence $S \blacktriangleright_p^! T$ below. This is justified as the height of nodes remain stable under $\sqsupset$-reductions.

Recall that $U \blacktriangleright_p V$ holds if there exist nodes u, v strictly below p with $U \sqsupset_v^u V$. For u, v given, the graph V is obtained by collapsing two nodes, which is a linear operation in the size of U. However, finding arbitrary nodes u and v such that $U \sqsupset_v^u V$ involves, in the worst case, a comparison of all pairs of nodes $u, v \in U$. Since up to linear many $\sqsupset$-steps in $|S|$ need to be performed, a straight forward implementation admits cubic runtime complexity. To achieve a quadratic bound in the size of the starting graph S, we construct a TM that implements a bottom up reduction-strategy. For $h \in \mathbb{N}$, define $U \sqsupset_{(h)} V$ if $U \blacktriangleright_p V$, where for the particular nodes $u, v \in U$ with $U \sqsupset_v^u V$, $\mathsf{ht}(u) = \mathsf{ht}(v) = h$ holds.

Observe also that a sequence $U_1 \sqsupset_{(h_1)} U_2 \sqsupset_{(h_2)} U_3$ can be reordered to $U_1 \sqsupset_{(h_2)} \cdot \sqsupset_{(h_1)} U_3$ whenever $h_1 \geqslant h_2$: The assumption $U_1 \sqsupset_{(h_1)} U_2 \sqsupset_{(h_2)} U_3$ gives a peak $U_2' \ {}_{(h_2)}\sqsubset U_1 \sqsupset_{(h_1)} U_2$ when $h_1 \geqslant h_2$, which can be joined by $U_2' \sqsupset_{(h_1)} U_3 \ {}_{(h_2)}\sqsubset U_2$ due to Lemma 5.5. To see that the peak exists, suppose $U_1 \sqsupset_{v_1}^{u_1} U_2 \sqsupset_{v_2}^{u_2} U_3$ holds. Let m be the morphism underlying $U_1 \sqsupset_{v_1}^{u_1} U_2$, that collapses u_1, v_1. Since m is surjective, we find nodes $u_2', v_2' \in U_1$ such that $m(u_2') = u_2$ and $m(v_2') = v_2$. By the assumption of the heights, u_2' and v_2' are *not* below u_1 or u_2. This implies, by the shape of m, that successors of $u_2, v_2 \in U_2$ coincide with successors of $u_2', v_2' \in U_1$. By Lemma 5.4 on $U_2 \sqsupset_{v_2}^{u_2} U_3$, it thus follows that $\mathsf{succ}_{U_1}(u_2') = \mathsf{succ}_{U_1}(v_2')$. Similarly, $\mathsf{lab}_{U_1}(u_2') = \mathsf{lab}_{U_1}(v_2')$, and using Lemma 5.4 from right to left constructs the peak.

The Turing Machine computes a sequence

$$S = S_0 \sqsupset^!_{(0)} S_1 \sqsupset^!_{(1)} \cdots \sqsupset^!_{(d)} S_{d+1} \,. \tag{6.1}$$

Observe that $U \blacktriangleright_p V$ if and only if $U \sqsupset_{(h)} V$ for some height $h \leqslant d$, where $d := \mathsf{dp}(U\restriction_p)$. That is, any $\blacktriangleright_p$ sequence can be written as a sequence of $\sqsupset_{(h)}$ steps of height $h \in \{0, \ldots, d\}$. Using the above observation that steps can be permuted, any normalising sequence can be turned into a sequence of steps in increasing height as above. We conclude that the above sequence is normalising, i.e., $S_{d+1} = T$. Consider a sub-sequence $S_h \sqsupset^!_{(h)} S_{h+1}$. This derivation is, without loss of generality, of the form

$$S_h = T_{h,1} \sqsupset^!_{u_1} \cdots \sqsupset^!_{u_n} T_{h,n+1} = S_{h+1} \,. \tag{6.2}$$

Here $u_1, \ldots, u_n$ are all nodes at height $h := \mathsf{ht}(u_1) = \cdots = \mathsf{ht}(u_n)$.

The TM operates in subsequent stages $h = 0, \ldots, d$. At stage h, the TM computes S_{h+1} from S_h, compare Derivation (6.1). This sub-derivation is computed in accordance to Derivation (6.2).

The TM M uses a dedicated working tape to store the *current graph* $T_{h,j}$, initially S. The size of this graph is at any time bounded by $\|S\|$, as the only operation we perform on this graph is the deletion of nodes. The *current height* is stored on a separate tape, initialised to 0. Further, the node addressed by p in S is computed by recursion on p in time quadratic in $\|S\|$. Afterwards, the quadratic marking algorithm of Lemma 6.6 is used to mark the sub-graph $S\restriction_p$ in S. Collapsing can only occur in this marked sub-graph. This finishes the initialisation phase.

The TM M uses the flags *deleted*, *temporary* and *permanent* besides the sub-term marking on nodes. The node-specifications of deleted nodes are simply ignored, which allows M to avoid cleanup after each iteration. A node is marked permanent if its height in the current graph is strictly below the current height. It follows that a node $u \in T_{h,j}$ is at the current height if and only if a successor is marked permanent. We are ready to give the procedure, suppose the machine M is at the beginning of stage h:

- Outer Loop. The machine is searching the next node u_j $(j = 1, \ldots, n)$ at height h to collapse, compare Derivation (6.2). It keeps the invariant that previously considered nodes $u_{j'} \leqslant u_j$ are marked temporary. Then the first node in the current graph $T_{h,j}$ that is neither marked temporary nor permanent, but where a successor is marked permanent, qualifies for u_j. In total, u_j can thus be determined in size linear in $T_{h,j}$, that is linear in $\|S\|$. If u_j is not found, all nodes at the height h have been considered, the machine goes into the next stage $h+1$, and marks all temporary nodes permanent. Otherwise, u_j is marked temporary, and M goes to the inner loop.

- Inner Loop. Let T be the current graph, by Lemma 5.4 $T \sqsupset_{u_j} T'$ for some term graph T' if and only if $\mathsf{lab}_T(u_j) = \mathsf{lab}_T(u'_j)$ and $\mathsf{succ}_T(u_j) = \mathsf{succ}_T(u'_j)$ for some node $u'_j \in T$. If such a node exists, it can be found by scanning T a constant number of times, thus in time linear in $\|S\|$. If no such node

is found, the loop aborts, otherwise the machine uses the recipe given in Lemma 5.4 for constructing T'. For the in-place construction of T', the TM M re-uses the node-specification of u_j in the encoding of T for the fresh node in T', and marks the node-specification u'_j deleted. Besides this, the recipe involves only a linear number of redirects in $|T|$. Clearly T' can thus be constructed in time linear in $\|S\|$.

When all stages are completed, the current graph is, as observed in the first half of the lemma, the $\blacktriangleright_p$-minimal graph S_ℓ. The current graph is then written on the output tape in two stages. During the first stage, the current graph is traversed from top to bottom, and the node specifications of non-deleted nodes is written on a separate working tape in time $\mathcal{O}(\|S\|)$. The algorithm from Lemma 6.5 translates this graph into a proper encoding graph, in time quadratic in $\|S\|$.

We now investigate on the computational complexity of the above procedure. Observe that the outer loop is iterated at most $|S\restriction_p| \leqslant |S|$ times often, over all stages. This is because at each iteration one unmarked node strictly below p is marked in the current graph. The total time spent in the outer loop, not counting the inner loop, can thus be estimated by $\mathcal{O}(|S| \cdot \|S\|) \subseteq \mathcal{O}(\|S\|^2)$. Each inner loop involves the construction of exactly one $\blacktriangleright_p$ step, using Lemma 5.8(2) we thus obtain that the inner loop body is executed at most $|S\restriction_p|$ times. The machine M thus spends at most time $\mathcal{O}(|S\restriction_p| \cdot \|S\|) \subseteq \mathcal{O}(\|S\|^2)$ in the inner loop. Overall, the main procedure, outer and inner loop traversal, thus takes time $\mathcal{O}(\|S\|^2)$. As initialisation and finalisation impose only quadratic overhead in $\|S\|$, we conclude the lemma. □

Summing up Lemmas 6.8–6.10, for a fixed rewrite system $\mathcal{G}$, term graph S and rewrite position p given, a single rewrite step $S \longrightarrow T$ can be performed in time quadratic in the representation size of S, for $\longrightarrow \in \{\mathrel{\blacktriangleleft\!\!\rightarrow}_{\mathcal{G}}, \mathrel{\blacktriangleleft\!\!\blacktriangleright\!\!\rightarrow}_{\mathcal{G}}, \xrightarrow{\mathsf{i}}_{\mathcal{G}}, \xrightarrow[\mathsf{i}]{\blacktriangleright}_{\mathcal{G}}\}$. The next lemma establishes that for an unknown rewrite position p, the complexity of computing a rewrite step is at most cubic. The given algorithm essentially exhaustively checks on all nodes in S if a rewrite step is possible. Depending on the considered rewrite relation $\longrightarrow$, the situation is slightly more complicated, as some care has to be taken in order to avoid folding or unfolding on every position $p \in \mathcal{P}\mathrm{os}(S)$, whilst retaining completeness of the algorithm.

Lemma 6.11. *For every GRS $\mathcal{G}$ and $\longrightarrow \in \{\mathrel{\blacktriangleleft\!\!\rightarrow}_{\mathcal{G}}, \mathrel{\blacktriangleleft\!\!\blacktriangleright\!\!\rightarrow}_{\mathcal{G}}, \xrightarrow{\mathsf{i}}_{\mathcal{G}}, \xrightarrow[\mathsf{i}]{\blacktriangleright}_{\mathcal{G}}\}$ there exists a Turing machine M that computes a reduction step $S \longrightarrow T$ for given S in time $\mathcal{O}(\|S\|^3)$.*

Proof. We consider the four cases for $\longrightarrow$ separately:

- Case $\longrightarrow = \mathrel{\blacktriangleleft\!\!\blacktriangleright\!\!\rightarrow}_{\mathcal{G}}$: Suppose $S \mathrel{\blacktriangleleft\!\!\blacktriangleright\!\!\rightarrow}_{L\to R,p} T$ holds. Unfolding the definition, and using Lemma 5.11(3), we have

$$S \blacktriangleright_p^! U \mathrel{\blacktriangleleft}_p^! V \longrightarrow_{L\to R,p} T \,,$$

  for some term graphs U, V. For $L \to R$ and p given, tacitly employing Lemma 5.8, we see that by Lemma 6.10, Lemma 6.8 and Lemma 6.9 the term graph T is computable in time $\mathcal{O}(\|S\|^2)$.

We provide a cubic algorithm that finds suitable position p and rule $L \to R \in \mathcal{G}$. For each position $q \in \mathcal{P}\mathrm{os}(S)$, let U_q denote the (unique) canonical term graph obtained by fully collapsing the sub-graph at position q, that is, $S \blacktriangleright^!_q U_q$ holds. We claim $S \blacktriangleright^!_q U_q \lhd^!_q V_q \longrightarrow_{L\to R,q} T$ if and only if U_q is reducible by $\longrightarrow_{L\to R,q}$. By Lemma 4.36, it suffices to prove that $U_q\restriction_q$ and $V_q\restriction_q$ are isomorphic. Let m be the morphism such that $U_q \preccurlyeq_m V_q$ holds. This morphism defines the identity on all nodes strictly below q. To see this, observe that m is just the composition of the morphism underlying the individual $\lhd_q$ steps in $U_q \lhd^!_q V_q$, compare Lemma 4.16. Hence $U_q\restriction_q$ and $V_q\restriction_q$ are trivially isomorphic.

In conclusion, a rule $L \to R \in \mathcal{G}$ and position q can be found by enumerating all term graphs U_q, and checking if they are $\longrightarrow_{L\to R,q}$ reducible. We emphasise that for $u \in S$, $U_{q_1} = U_{q_2}$ for each pair of positions $\{q_1, q_2\} \subseteq N_S(u)$. It thus suffices to enumerate U_q for a single position $q \in \mathcal{P}\mathrm{os}(u)$ for every $u \in S$.

Guided by this observation, the machine M iterates over all nodes $u \in S$, extracts a position $q \in \mathcal{P}\mathrm{os}_S(u)$ and verifies if U_q is $\longrightarrow_{L\to R,q}$ reducible for some $L \to R \in \mathcal{G}$. If so, a proper rule $L \to R$ and position q with $S \blacktriangleleft\!\blacktriangleright\!\to_{L\to R,q} T$ has been found, otherwise it continues with the next node in S.

By the above observation, this search is complete. We verify that the search operates in cubic time in $\|S\|$. Note that a position $q \in \mathcal{P}\mathrm{os}_S(u)$ can be constructed by backward traversal from u to the root of S, in time quadratic in $\|S\|$. Under this bound, also U_q can be constructed, and matched against all rules $L \to R \in \mathcal{G}$ at position q. Compare Lemma 6.10 and Lemma 6.7. Here we employ that only a constant number of rules need to be checked. There are at most $|S|$ iterations, hence the overall bound is given by $\|S\|^3$. This concludes the case $\longrightarrow = \blacktriangleleft\!\blacktriangleright\!\to$.

- Case $\longrightarrow = \blacktriangleleft\!\!\longmapsto_{\mathcal{G}}$: The construction of the machine is a simplification of the above construction, exploiting that S is $\blacktriangleleft\!\!\longmapsto_{L\to R,q}$ reducible if and only if it is $\longrightarrow_{L\to R,q}$ reducible, which implies that a redex can be determined without the construction of intermediate $\lhd_q$-maximal graphs U_q.

- Case $\longrightarrow = \xrightarrow{\mathsf{i}}_{\mathcal{G}}$: In this case $S \xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ if and only if there is a node $u \in S$ and rule $L \to R \in \mathcal{G}$ such that $S \longrightarrow_{L\to R,u} T$ holds, but $S \longrightarrow_{\mathcal{G},v} T$ does not hold for any node v strictly below u in S. Using the machine of Lemma 6.7, all redex-nodes in S can be marked in time cubic in $\|S\|$. Using this marking, it can be determined in time quadratic in $\|S\|$ if $S \xrightarrow{\mathsf{i}}_{L\to R,u} T$ holds. If this check is unsuccessful the machine aborts. Otherwise, the TM M uses the machine of Lemma 6.8 to compute the term graph T with $S \xrightarrow{\mathsf{i}}_{L\to R,p} T$ for some $p \in \mathcal{P}\mathrm{os}_S(u)$. Note that computing a position $p \in \mathcal{P}\mathrm{os}_S(u)$ incurs only quadratic overhead.

- Case $\longrightarrow = \blacktriangleright\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}$: The construction is similar to the previous one. The marking of redex-nodes is however performed with respect to $\blacktriangleright\!\longrightarrow_{\mathcal{G},q}$. By reasoning identical to the case $\longrightarrow = \blacktriangleleft\!\blacktriangleright\!\to$, it suffices to check only one

arbitrary position per node. That is, for every node $u \in S$, the TM M computes one arbitrary position $q \in \mathcal{P}\mathrm{os}(u)$, and marks the node u if the term graph U_q, with $S \blacktriangleright^!_q U_q$, is $\longrightarrow_{\mathcal{G},q}$ reducible. If u is marked, and all its successors are not marked, then M selects $p \in \mathcal{P}\mathrm{os}_S(u)$ and performs the reduction step $S \blacktriangleright\!\!\longrightarrow_{\mathcal{G},p} T$.

By reasoning identical to above, we obtain that M operates in time $\mathcal{O}(\|S\|^3)$. We show that it is correct. For this, we first prove that the following statements are equivalent.

(1) The node $u \in S$ addressed by position $p{\cdot}q \in \mathcal{P}\mathrm{os}(S)$ is marked.

(2) $U \blacktriangleright^!_{p'} U_{p'} \longrightarrow_{\mathcal{G},p'} T$ for some term graph T and $p' \in \mathcal{P}\mathrm{os}_S(u)$.

(3) $U \blacktriangleright^!_{p\cdot q} U_{p\cdot q} \longrightarrow_{\mathcal{G},p\cdot q} T$ for some term graph T.

(4) $U \blacktriangleright^!_{p} U_{p} \longrightarrow_{\mathcal{G},p\cdot q} T$ for some term graph T.

The equivalence (1) $\Leftrightarrow$ (2) follows by construction of M, and correctness of the implementation of the folding (Lemma 6.10) and matching algorithm (Lemma 6.7). In the correctness proof of the implementation of $\blacktriangleleft\!\!\blacktriangleright\!\!\rightarrow_{\mathcal{G}}$ we have seen that $U_{p'}$ is isomorphic to $U_{p\cdot q}$. Hence statement (2) is equivalent to $U \blacktriangleright^!_{p\cdot q} U_{p\cdot q} \longrightarrow_{\mathcal{G},p'} T$, as $\{p{\cdot}q, p'\} \subseteq \mathcal{P}\mathrm{os}_S(u)$. Since the positions $p{\cdot}q$ and p' address the same node in S, these positions address by Lemma 4.22 also the same node in $U_{p\cdot q}$, compare Lemma 4.22. We conclude (2) $\Leftrightarrow$ (3). Finally, consider statement (3). Since $\blacktriangleright_{p\cdot q} \subseteq \blacktriangleright_p$ by definition, and $\blacktriangleright_p$ is confluent (compare Lemma 5.5), we see that without loss of generality, $S \blacktriangleright^!_{p\cdot q} U_{p\cdot q} \blacktriangleright^!_q U_p$ holds. Using that $U_{p\cdot q}$ is $\blacktriangleright_{p\cdot q}$ minimal, we see that the morphism m underlying $U_{p\cdot q} \blacktriangleright^!_q U_p$, i.e., $U_{p\cdot q} \succcurlyeq_m U_p$, defines the identity on all nodes below position $p{\cdot}q$. That is, the sub-graphs $U_{p\cdot q}$ and U_q at position $p{\cdot}q$ are isomorphic. Hence U_p is reducible by $\longrightarrow_{\mathcal{G},p\cdot q}$ if and only if $U_{p\cdot q}$ is. This concludes (3) $\Leftrightarrow$ (4).

For soundness, suppose that M has selected a position $p \in \mathcal{P}\mathrm{os}_S(u)$ for the reduction step. Thus by construction u is marked, and all nodes v strictly below p are unmarked. Recall that $\mathcal{P}\mathrm{os}(S) = \mathcal{P}\mathrm{os}(U_p)$ (compare Lemma 4.17 and Lemma 4.22). By (1) $\Rightarrow$ (4) (with $p = \epsilon$) we get that $U \blacktriangleright^!_p U_p \longrightarrow_{\mathcal{G},p} T$ holds. The contraposition of (4) $\Rightarrow$ (1) on unmarked $v \in S$ gives that for all $p{\cdot}q \in \mathcal{P}\mathrm{os}(U_p)$, the node addressed by $p{\cdot}q$ in $\mathcal{P}\mathrm{os}(U_p)$ is not $\mathcal{G}$ reducible, so indeed $S \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G},p} T$ holds.

For completeness, suppose $S \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G},p} T_p$ holds. Thus (4) $\Rightarrow$ (1) with $q = \epsilon$ gives that u addressed by p in S is marked. Using the contraposition of (1) $\Rightarrow$ (4) we see that the nodes v strictly below $u \in S$ are not marked. Thus M will select a node $q \in \mathcal{P}\mathrm{os}(S)$, and perform a reduction step as desired. □

Employing the size approximation given in Lemma 6.3, we now lift Lemma 6.11 to sequences. This constitutes the main result of this section.

Theorem 6.12. *Let $\mathcal{G}$ be a GRS, and consider $\longrightarrow \in \{\blacktriangleleft\!\!\rightarrow_{\mathcal{G}}, \blacktriangleleft\!\!\blacktriangleright\!\!\rightarrow_{\mathcal{G}}\}$. Let $S \in \mathcal{G}_c(\mathcal{F},\mathcal{V})$ be a term graph with $\mathsf{dh}(S,\longrightarrow) = \ell$.*

(1) *A specific* $\longrightarrow$ *normal form of* s *is computable in deterministic time*

$$\mathcal{O}(\log(\ell \cdot |S|)^3 \cdot (\ell^4 \cdot |S|^3 + \ell^7)) \ .$$

(2) *Any* $\longrightarrow$ *normal form of* S *is computable in non-deterministic time*

$$\mathcal{O}(\log(\ell \cdot |S|)^2 \cdot (\ell^3 \cdot |S|^2 + \ell^5)) \ .$$

Proof. We consider first Assertion (1). Suppose $\mathsf{dh}(S, \longrightarrow) = \ell$. We use the machine M from Lemma 6.8 to compute a normalising rewrite sequence

$$S = T_0 \longrightarrow T_1 \longrightarrow \cdots \longrightarrow_{\mathcal{G}} T_l \ .$$

Using the size estimation of Lemma 6.3(1) the size of T_i $(i = 1, \ldots, l)$ is bounded by

$$|T_i| \leqslant (i+1) \cdot |S| + i^2 \cdot \Delta_{\mathcal{G}} \in \mathcal{O}(\ell \cdot |S| + \ell^2) \ ,$$

where we use that the constant $\Delta_{\mathcal{G}}$ depends only on the graph rewrite system $\mathcal{G}$. Using the rewrite algorithm from Lemma 6.11 that is cubic in the representation size of the given term, each rewrite step $T_i \longrightarrow T_{i+1}$ $(i = 0, \ldots, l-1)$ can thus be performed in time

$$\begin{aligned} \|T_i\|^3 &\in \mathcal{O}(\log(|T_i|)^3 \cdot |T_i|^3) \\ &\in \mathcal{O}(\log(\ell \cdot |S|)^3 \cdot (\ell \cdot |S| + \ell^2)^3) \\ &= \mathcal{O}(\log(\ell \cdot |S|)^3 \cdot (\ell^3 \cdot |S|^3 + \ell^6)) \ . \end{aligned}$$

As at most $l \leqslant \ell$ steps need to be performed, the theorem follows for this case.

For Assertion (2), consider an arbitrary sequence

$$S = T_0 \longrightarrow T_1 \longrightarrow \cdots \longrightarrow_{\mathcal{G}} T_l \ .$$

As indicated before in Lemma 6.11, a single rewrite step $T_i \longrightarrow T_{i+1}$ $(i = 1, \ldots, l-1)$ can be performed in quadratic time in $\|T_i\|$, if the rewrite position p is known. Nondeterminism allows the TM M to guess any rewrite position p in T_i in time $\mathcal{O}(\|T_i\|^2)$. Note that this bound includes the verification that p is indeed a valid rewrite position. Overall, any rewrite step can thus be performed in nondeterministic time

$$\mathcal{O}(\log(\ell \cdot |S|)^2 \cdot (\ell^2 \cdot |S|^2 + \ell^4)) \ ,$$

carrying out the estimation as in the deterministic case. The theorem follows.□

Theorem 6.13. *Let* $\mathcal{G}$ *be a GRS, and consider* $\longrightarrow \in \{\xrightarrow{\mathsf{i}}_{\mathcal{G}}, \blacktriangleright\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}\}$. *Let* $S \in \mathcal{G}_c(\mathcal{F}, \mathcal{V})$ *be a term graph with* $\mathsf{dh}(S, \longrightarrow) = \ell$.

(1) *A specific* $\longrightarrow$ *normal form of* S *is computable in deterministic time*

$$\mathcal{O}(\log(\ell + |S|)^3 \cdot (\ell \cdot |S|^3 + \ell^4) \ .$$

(2) *Any* $\longrightarrow$ *normal form of* S *is computable in non-deterministic time*

$$\mathcal{O}(\log(\ell + |S|)^2 \cdot (\ell \cdot |S|^2 + \ell^3) \; .$$

Proof. We follow the proof of Theorem 6.12, but replace Lemma 6.3(1) by Lemma 6.3(2) which gives

$$|T_i| \in \mathcal{O}(\ell + |S|) \; ,$$

in a normalising sequence

$$S = T_0 \longrightarrow T_1 \longrightarrow \cdots \longrightarrow_{\mathcal{G}} T_l \; .$$

Using the cubic algorithm from Lemma 6.11 that implements a single rewrite step and unfolding the definition of representation size, we see that a single rewrite step $T_i \longrightarrow T_{i+1}$ $(i = 1, \ldots, l-1)$ can be performed in time

$$\begin{aligned} \mathcal{O}(\|T_i\|^3) &\subseteq \mathcal{O}(\log(\ell + |S|)^3 \cdot (|S| + \ell)^3) \\ &= \mathcal{O}(\log(\ell + |S|)^3 \cdot (|S|^3 + \ell^3)) \; . \end{aligned}$$

The theorem follows thus from the assumption that $l \leqslant \ell$.

For the nondeterministic case, we use the quadratic non-deterministic algorithm as in Theorem 6.12(2), in conjunction with the size estimation from Lemma 6.3(2). This shows that a single rewrite step $T_i \longrightarrow T_{i+1}$ $(i = 1, \ldots, l-1)$ is computable in time

$$\mathcal{O}(\|T_i\|^2) \in \mathcal{O}(\log(\ell + |S|)^2 \cdot (|S|^2 + \ell^2)) \; . \qquad \square$$

Chapter 7.

The Polynomial Invariance Theorem

We arrive at the central result of this part, the *polynomial invariance theorems*. The following intermediate theorem connects our adequacy results with the implementation of graph rewriting given in Theorem 6.12 and Theorem 6.13.

Theorem 7.1. *Let $\mathcal{R}$ be a TRS.*

(1) *There exists a deterministic Turing machine M that, given an innermost terminating term t, computes a graph representation of a normal form u of t in time $p(|t|, \mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}))$, where*

$$p(n, \ell) \in \mathcal{O}\big(\log(\ell + n)^3 \cdot (\ell \cdot n^3 + \ell^4)\big) \ .$$

(2) *There exists a non-deterministic Turing machine M that, given a terminating term t, computes a graph representation of any normal form u of t in time $p(|t|, \mathsf{dh}(t, \to_{\mathcal{R}}))$, where*

$$p(n, \ell) \in \mathcal{O}\big(\log(\ell \cdot n)^2 \cdot (\ell^3 \cdot n^2 + \ell^5)\big) \ .$$

Proof. Let $\mathcal{G}$ be a graph rewrite system that unfolds to $\mathcal{R}$, such that for every $L \to R \in \mathcal{G}$, only variable nodes are shared. It is not difficult to see that $\mathcal{G}$ exists. For the first assertion, consider a term t such that $\mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}})$ is defined. Let $T \in \Diamond_c^{\mathrm{NF}(\mathcal{R})}(\mathcal{F}, \mathcal{V})$ be the representation of t as the canonical term graph that is a tree, hence in particular $|t| = |T|$. As a consequence of the adequacy theorem for innermost rewriting, Theorem 5.16(2), we have that $\mathsf{dh}(T, \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}}) = \mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}})$ is defined, and moreover, a maximal reduction

$$T = T_0 \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} T_1 \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} \cdots \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}} T_l = U \ ,$$

yields a term graph U which unfolds to a normal form of t. By Theorem 6.13(1), the graph U is computable from T in time $p(|T|, \mathsf{dh}(T, \blacktriangleright\!\!\xrightarrow{\mathsf{i}}_{\mathcal{G}})) = p(|t|, \mathsf{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}))$ for sufficiently large t. The first assertion follows.

For the second assertion, we proceed as above, but use the adequacy theorem for full rewriting, Theorem 5.12(2), and the implementation of graph rewriting given in Theorem 6.12(2), instead. □

Theorem 7.2 (Polynomial Invariance Theorem)**.** *Let $\mathcal{R}$ be a confluent TRS with $\mathrm{rci}_{\mathcal{R}}(n) \in \mathcal{O}(g(n))$. Let $f \in \mathcal{D}$ and let $\mathcal{N}$ denote a set of non-accepting patterns.*

Then there exists a polynomial function $p : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *such that the function* $[\![f]\!]_{\mathcal{R},\mathcal{N}}$ *is computable on a deterministic Turing machine in time* $p(n, g(n))$*, where* n *denotes the sum of the sizes of the input, and*

$$p(n, \ell) \in \mathcal{O}\big(\log(\ell + n)^3 \cdot (\ell \cdot n^3 + \ell^4)\big) \ .$$

Proof. Suppose $\mathcal{R}$ is as given by the Theorem, let $f \in \mathcal{D}$ and consider a set non-accepting patterns $\mathcal{N}$. Then by Theorem 7.1(1), for any tuple of arguments $(v_1, \ldots, v_k) \in \mathcal{T}(\mathcal{C})^k$, a term graph representation U of a normal form u of $s := f(v_1, \ldots, v_k)$ is computable in time $p_r(|s|, g(|s|))$, where

$$p_r(n, \ell) \in \mathcal{O}\big(\log(\ell + n)^3 \cdot (\ell \cdot n^3 + \ell^4)\big) \ .$$

It is not difficult to see that in linear time in the representation size of U it can be decided if u is a value. Recall that the set of non-accepting patterns $\mathcal{N}$ is finite. Using soundness and completeness of graph based matching for tree representations of patterns $p \in \mathcal{N}$ (Lemma 4.43 and Lemma 4.49), by Lemma 6.7 we see that it can be decided in time quadratic in the representation size of U whether u is accepting. Since U is the result of a $\overset{\mathsf{i}}{\blacktriangleright\!\!\rightarrow}_{\mathcal{G}}$ reduction of length at most $g(|s|)$, Lemma 6.3(2) allows us to conclude that the accepting condition for u can be decided in time $p_a(|s|, g(|s|))$ for some

$$p_a(n, \ell) \in \mathcal{O}\big(\log(\ell + n)^3 \cdot (\ell \cdot n^3 + \ell^4)\big) \ .$$

In total, $[\![f]\!]_{\mathcal{R},\mathcal{N}}(v_1, \ldots, v_k)$ can thus be computed in time $p_r(|s|, g(|s|)) + p_r(|s|, g(|s|))$. Since $|s| = 1 + \sum_{i=1}^{k} |v_i|$ the theorem follows. □

Instantiating the function g by a polynomial or exponential respectively, we obtain the following corollaries.

Corollary 7.3. *Let* $\mathcal{R}$ *be a confluent TRS with* $\mathrm{rci}_{\mathcal{R}}(n) \in \mathcal{O}(n^k)$ *for some* $k \in \mathbb{N}$ *with* $k \geqslant 1$*. Let* $f \in \mathcal{D}$ *and let* $\mathcal{N}$ *denote a set of non-accepting patterns.*

Then $[\![f]\!]_{\mathcal{R},\mathcal{N}}$ *is computable in deterministic time* $p(n) \in \mathcal{O}(\log(n)^3 \cdot n^{4 \cdot k})$*. In particular, the computed function belongs to* **FP**.

Corollary 7.4. *Let* $\mathcal{R}$ *be a confluent TRS with* $\mathrm{rci}_{\mathcal{R}}(n) \in 2^{\mathcal{O}(n^k)}$ *for some* $k \in \mathbb{N}$ *with* $k \geqslant 1$*. Let* $f \in \mathcal{D}$ *and let* $\mathcal{N}$ *denote a set of non-accepting patterns.*

Then $[\![f]\!]_{\mathcal{R},\mathcal{N}}$ *is computable in deterministic time* $e(n) \in 2^{\mathcal{O}(n^k)}$*. In particular, the computed function belongs to* **FEXP**.

Theorem 7.5 (Nondeterministic Polynomial Invariance Theorem)**.** *Let* $\mathcal{R}$ *be a TRS with* $\mathrm{rc}_{\mathcal{R}}(n) \in \mathcal{O}(g(n))$*. Let* $f \in \mathcal{D}$ *and let* $\mathcal{N}$ *denote a set of non-accepting patterns.*

Then there exists a polynomial function $p : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *such that the function problem associated with* $[\![f]\!]_{\mathcal{R},\mathcal{N}}$ *is computable on a Turing machine in time* $p(n, g(n))$*, where* n *denotes the sum of the sizes of the input and*

$$p(n, \ell) \in \mathcal{O}\big(\log(\ell + n)^3 \cdot (\ell \cdot n^3 + \ell^4)\big) \ .$$

Proof. The theorem follows as in Theorem 7.2, replacing the application of Theorem 7.1(1) by Theorem 7.1(2). □

Instantiating the function g by a polynomial $p(n)$ yields by Proposition 2.19 the following corollary.

Corollary 7.6. *Let $\mathcal{R}$ be a TRS with $\mathrm{rci}_{\mathcal{R}}(n) \in \mathcal{O}(n^k)$ for some $k \in \mathbb{N}$ with $k \geqslant 1$. Let $f \in \mathcal{D}$ and let $\mathcal{N}$ denote a set of non-accepting patterns.*

Then the function problem associated with $[\![f]\!]_{\mathcal{R},\mathcal{N}}$ is computable in time $p(n) \in \mathcal{O}(\log(n)^2 \cdot n^{5 \cdot k})$. In particular, the computed function problem belongs to **FNP**.

Part II.

Order-Theoretic Characterisation of Complexity Classes

Chapter 8.

Introduction

In this part we are concerned with *order-theoretic* characterisations of the class of polytime and exponential time computable functions. Since our characterisations are *resource free*, i.e., without explicit reference to computational resources, this research is closely connected to the field of *implicit computational complexity* (*ICC* for short). Furthermore, our characterisations have also ramifications in the automated complexity analysis of rewrite systems.

Our proposed *small polynomial path order* (sPOP* for short) forms a restriction of the multiset path order. This order embodies Bellantoni and Cook's principle of *predicative recursion* [21] on compatible TRSs. Underlying the notion of predicative recursion is a separation of function parameters into *normal* and *safe* arguments. Notationally we write

$$f(t_1, \ldots, t_k; t_{k+1}, \ldots, t_{k+l})$$

to indicate this separation. *Normal* arguments are drawn to the left, and *safe* arguments to the right of the semicolon. Predicative recursion restricts primitive recursion by allowing recursion only only normal arguments, whereas only safe arguments allow substitution of recursive values. Precisely, a new function f (over binary words) is defined by *predicative recursion* (aka *safe recursion*) via the following three equations, where g, h_0 and h_1 denote previously defined functions.

$$\begin{aligned} f(\epsilon, \vec{x}; \vec{y}) &= g(\vec{x}; \vec{y}) \\ f(zi, \vec{x}; \vec{y}) &= h_i(z, \vec{x}; \vec{y}, f(z, \vec{x}; \vec{y})) \quad i = 0, 1\ . \end{aligned} \tag{\textbf{SRN}}$$

The net effect of this separation is that the stepping functions cannot in turn perform recursion on the value of $f(z, \vec{x}; \vec{y})$. Bellantoni and Cook thus impose a purely syntactic restriction on primitive recursion. This restriction is on the one hand strong enough to exclude definitions of infeasible functions. On the other hand, it is weak enough to allow the definition of all feasible functions [21].

The small polynomial path order delineates a class of rewrite systems, the class of *predicative recursive* constructor TRSs. The (innermost) runtime complexity of predicative recursive constructor TRSs is bounded by a polynomial function. This class of TRSs is thus *sound* for **FP**: any function computed by a predicative recursive constructor TRSs is computable on a Turing machine in polynomial time, in correspondence to Corollary 7.3. We can also show that this class of rewrite systems is *complete* for the **FP**: every polytime computable function is computed by a TRS from this class. In total, small polynomial path orders yield an implicit characterisation of the polytime computable functions.

In predicative recursive TRSs, computation is only permitted on safe argument positions. As a result, the runtime complexity of a predicative TRS $\mathcal{R}$ depends essentially only on the number of recursion performed in a reduction. In contrast to previous work [54, 11], this allows us to establish a precise correspondence between the runtime complexity of $\mathcal{R}$ and the *depth of recursion* of $\mathcal{R}$. Here the depth of recursion essentially amounts to the maximal number of nested recursive definitions in $\mathcal{R}$.

We extend upon this work by introducing the *exponential path order* (EPO* for short). This order is a syntactic extension of sPOP* that replaces the underlying safe recursion scheme by *safe nested recursion* [2]. As a result, we obtain a miniaturisation of the recursive path order with lexicographic status. The exponential path order order delineates the class of *predicative nested recursive TRSs*. We establish that for any predicative nested recursive constructor TRS $\mathcal{R}$, the innermost runtime complexity function is bounded by an exponential function $e(n) \in 2^{\mathcal{O}(n^k)}$ ($k \in \mathbb{N}$). Essentially relying on the polynomial invariance theorem and Arai and Eguchi's work [2], we obtain an order-theoretic characterisation of the exponential time computable functions.

In total, we present the following contributions in this part.

- We define the small polynomial path order that delineates the class of predicative recursive constructor TRSs. We establish that for any predicative recursive constructor TRS $\mathcal{R}$, the innermost runtime complexity function is bounded by a polynomial function of degree d, where d corresponds to the maximal depth of recursion in $\mathcal{R}$.

- The order sPOP* yields an order-theoretic characterisation of the class of polytime computable functions. Any function defined by a *confluent* predicative recursive constructor TRS is polytime computable. Vice versa, any polytime computable function is computed by some confluent and predicative recursive constructor TRS.

- We extend upon sPOP* by integrating *parameter substitution* in the predicative recursion scheme. This extension allows us to cover functions defined by *tail-recursion*. As by-product we obtain an alternative proof of closure under parameter substitution of the polytime computable functions.

- If a function is defined by a predicative recursive constructor TRS and the maximal depth of recursion is d, then the function is implementable on a Turing machine in time $\mathcal{O}(\log(n) \cdot n^{4 \cdot d})$, compare Theorem 7.3. We extend upon this result as follows. Suppose that the functions in the considered TRS are defined by tail-recursion only. Further, suppose that constructors are at most unary, i.e., constructors are used to encode words only. Then these functions are even computable in time $\mathcal{O}(n^d)$, on *register machines*. Vice versa, any function defined by a register machine operating in time $\mathcal{O}(n^d)$ is definable as a predicative recursive constructor TRS whose maximal depth of recursion is d, and which fulfils the above mentioned restrictions. In total, we obtain a *tight* characterisation of the functions computable on register machines in time $\mathcal{O}(n^d)$, for every $d \in \mathbb{N}$.

- We extend the definition of sPOP* by integrating *safe nested recursion* [2]. The resulting order EPO* can handle systems which admit exponential innermost runtime complexity. This order gives rise to the notion of predicative nested recursive TRS. Any function defined by a predicative nested recursive constructor TRS belongs to the class of exponential time computable function. Vice versa, any exponential time computable function is definable as a (confluent) predicative nested recursive constructor TRS. This establishes our order-theoretic characterisation of the exponential time computable functions.

Related Work. There are several accounts of predicative analysis of recursion in the (ICC) literature. We mention only those related works which are directly comparable to our work. See [18] for an overview on ICC.

Notably the clearest connection of our work is to Marion's *light multiset path order* (*LMPO* for short) [54] and the *polynomial path order* [11, 6]. Both orders form a strict extension of the here proposed order sPOP*, but lack the precision of the latter. Although LMPO characterises **FP**, the runtime complexity of compatible TRSs is not polynomially bounded in general. POP* induces a polynomial bound on the innermost runtime complexity function, but the obtained complexity certificate is usually very imprecise. In particular, due to the multiset status underlying POP*, for each $d \in \mathbb{N}$ one can form a TRS compatible with POP* that defines only a single function, but whose runtime is bounded from below by a polynomial of degree d [6].

We have also drawn motivation from [55] which provides a related fine-grained classification of the polytime computable functions based on an adaption of *tiered recursion* [52]. Here, any function defined by *strict ramified primitive recursion* is computable in time $p(n) \in \mathcal{O}(n^k)$ on register machines, where k refers to the maximal *tier* of an input argument. Notable, the tier corresponds the depth of recursion, which is also used in the present work as degree of the bounding polynomial.

In Bonfante et. al. [23] restricted classes of polynomial interpretations are studied that can be employed to obtain polynomial upper bounds on the runtime complexity of TRSs. Related to this work is also [61], where *triangular* matrix interpretations [32] are shown to induce polynomial bounds on the runtime complexity function. This work has been carried on by Neurauter et. al. [62, 56] and also Waldmann [77]. However, it is worth emphasising that these indeed powerful direct techniques basically employ semantic considerations on the rewrite systems, which are notoriously difficult to check.

Outline. This part collects the authors contributions of joint work with Eguchi and Moser. Apart from the tight characterisation of register machine computations, these results constitute already published work [8, 7]. A journal version of [8] which includes the tight characterisation has been submitted to the special issue on ICC of TCS, and is currently under review. Eguchi established that in Bellantoni and Cook's class $\mathcal{B}$, the composition scheme can be restricted. From this, our completeness result of small polynomial path orders is trivial to estab-

lish. This result is rendered in Proposition 9.35. Apart from this proposition, the results concerning small polynomial path orders constitute the authors own contributions. The exponential path order constitutes a natural combination on Eguchi and Arai's work on a function algebra $\mathcal{N}$ for the exponential time computable functions [2], the corresponding order EPO [31], and Moser and the authors work on polynomial path orders [11]. Although not contributions of the author, both the order EPO and also the function algebra $\mathcal{N}$ are restated here, since these play important roles in the soundness and completeness proof of exponential path orders.

The next chapter is concerned with small polynomial path orders. In Section 9.1 we proof that this order is *sound*, in the sense that the innermost runtime complexity of compatible constructor TRSs is bounded by a polynomial function. In Section 9.2 we then deal with the above mentioned *completeness* property. In Section 9.3 we incorporate parameter substitution into the small polynomial path order, and prove soundness for this extension. The tight correspondence to register machines is then covered in Section 9.4.

Finally, in Chapter 10 we introduce the exponential path order. Soundness of this order is proven in Section 10.1, and Section 10.2 covers completeness.

Chapter 9.

The Small Polynomial Path Order

As usual for recursive path orders, also sPOP* is parameterised by a precedence $\succsim$. For sPOP* we use the precedence in the sense that $f \succ g$ denotes precisely that the function f *is defined based on* g. Small polynomial path orders distinguish between *recursive symbols*, i.e, function symbols that are defined by recursive means, and non-recursive symbols. This separation is used in the definition of *depth of recursion* admitted by a TRS, and allows us to more tightly estimate the runtime complexity of the analysed TRS. The following definition clarifies these notions.

Definition 9.1 (Recursive Symbols, Rank, Depth of Recursion, Admissible)**.** Let $\mathcal{R}$ be a TRS with defined symbols in $\mathcal{D}$ and constructors in $\mathcal{C}$.

(1) For defined symbol $f \in \mathcal{D}$ and function symbol $g \in \mathcal{F}$ we define $f \gg_\mathcal{R} g$ if there exists a rewrite rule $f(l_1, \dots, l_n) \to r \in \mathcal{R}$ where g occurs in r. In this case we also say that f is *defined based on* g in $\mathcal{R}$.

(2) A function symbol $f \in \mathcal{D}$ is called *recursive* in $\mathcal{R}$ if $f \gg^+_\mathcal{R} f$ holds, i.e., f is defined based on itself. The set of recursive function symbols of $\mathcal{R}$ is denoted by $\mathcal{K}_\mathcal{R}$.

(3) Let $\succsim$ denote the pre-order obtained as the least extension of $\gg^*_\mathcal{R}$ where constructors are equivalent, i.e., $c \succsim d$ and $d \succsim c$ for all constructors $c, d \in \mathcal{C}$. The order $\succsim$ is called the *precedence underlying* $\mathcal{R}$.

(4) The *rank* $\mathsf{rk}_\succsim(f)$ of $f \in \mathcal{F}$ with respect to a precedence $\succsim$ is given as follows:

$$\mathsf{rk}_\succsim(f) := \max\{0\} \cup \{\mathsf{rk}_\succsim(g) \mid f \succ g\} + 1 \,.$$

(5) The *depth of recursion* $\mathsf{rd}_{\mathcal{K},\succsim}(f)$ of $f \in \mathcal{F}$ with respect to a precedence $\succsim$ and a set of recursive symbols $\mathcal{K}$ is defined as follows:

$$\mathsf{rd}_{\mathcal{K},\succsim}(f) := \begin{cases} \max\{0\} \cup \{\mathsf{rd}_{\mathcal{K},\succsim}(g) \mid f \succ g\} + 1 & \text{if } f \in \mathcal{K}, \\ \max\{0\} \cup \{\mathsf{rd}_{\mathcal{K},\succsim}(g) \mid f \succ g\} & \text{otherwise.} \end{cases}$$

Let $\succsim$ denote the precedence underlying $\mathcal{R}$. We define $\mathsf{rd}_\mathcal{R}(f) := \mathsf{rd}_{\mathcal{K}_\mathcal{R},\succsim}(f)$ for all $f \in \mathcal{F}$, and call $\mathsf{rd}_\mathcal{R}(f)$ the *depth of recursion of* f *in* $\mathcal{R}$.

(6) For recursive symbols $\mathcal{K}$ and constructors $\mathcal{C}$, we call a precedence $\succsim$ *admissible* if the underlying equivalence $\sim$ honours the separation of recursive symbols and constructors, that is, for all $f, g \in \mathcal{F}$ with $f \sim g$ we have that (i) $f \in \mathcal{K}$ implies $g \in \mathcal{K}$ and likewise (ii) $f \in \mathcal{C}$ implies that $g \in \mathcal{C}$ holds.

Observe that the precedence underlying a TRS $\mathcal{R}$ is always admissible with respect to the recursive symbols of $\mathcal{R}$ and constructors $\mathcal{C}$. We illustrate the construction of the induced precedence in the next example.

Example 9.2. Consider the following constructor TRS $\mathcal{R}_{\mathsf{arith}}$ that defines $+$ (addition), $\times$ (multiplication) in Peano arithmetic, as well as a function f computing $m \times n^2$ on inputs $m, n \in \mathbb{N}$, where naturals are given from constructors 0 and s.

$$\begin{aligned}
&\mathtt{3}\colon +(0, y) \to y && \mathtt{4}\colon +(\mathsf{s}(x), y) \to \mathsf{s}(+(x, y)) \\
&\mathtt{5}\colon \times(0, y) \to 0 && \mathtt{6}\colon \times(\mathsf{s}(x), y) \to +(y, \times(x, y)) \\
&\mathtt{7}\colon \ \mathsf{f}(x, y) \to +(x, \times(y, y))\,.
\end{aligned}$$

By rule 4, addition is defined based on the successor symbol s and itself, by rule 5 and 6 multiplication ($\times$) is defined based on the constructor 0, addition and itself. By rule 7 the defined symbol f is defined based on addition and multiplications. Hence in total, we have

$$\begin{aligned}
&\mathsf{f} \gg_{\mathcal{R}_{\mathsf{arith}}} + && \mathsf{f} \gg_{\mathcal{R}_{\mathsf{arith}}} \times && \\
&\times \gg_{\mathcal{R}_{\mathsf{arith}}} 0 && \times \gg_{\mathcal{R}_{\mathsf{arith}}} + && \times \gg_{\mathcal{R}_{\mathsf{arith}}} \times \\
&+ \gg_{\mathcal{R}_{\mathsf{arith}}} \mathsf{s} && + \gg_{\mathcal{R}_{\mathsf{arith}}} +\,. &&
\end{aligned}$$

The defined symbol f constitutes the only function symbol that is not recursive, whereas multiplication and addition are (trivially) recursive, that is, $\mathcal{K}_{\mathcal{R}_{\mathsf{arith}}} = \{+, \times\}$. Overall, the precedence induced by $\mathcal{R}_{\mathsf{arith}}$ is given by the preorder $\succsim$ satisfying

$$\mathsf{f} \ \succ \ \times \ \succ \ + \ \succ \ \mathsf{s} \ \sim \ 0\,. \qquad \lhd$$

To impose the scheme of predicative recursion on compatible TRS, sPOP* assumes an a priori defined separation of argument positions for each $f \in \mathcal{F}$ into *normal* and *safe* ones.

Definition 9.3. A *safe mapping* is a function $\mathsf{safe}\colon \mathcal{F} \to \mathcal{P}(\mathbb{N})$ that associates with every n-ary function symbol f the set of *safe argument positions* $\{i_1, \dots, i_m\} \subseteq \{1, \dots, n\}$. Argument positions not included in $\mathsf{safe}(f)$ are called *normal* and collected in $\mathsf{nrm}(f)$. For all n-ary constructors $c \in \mathcal{C}$ we require that all argument positions are safe, i.e., $\mathsf{safe}(c) = \{1, \dots, n\}$.

Throughout the following, we fix a safe mapping safe on the signature $\mathcal{F}$. To avoid notational overhead, we suppose that for each $k + l$ ary function symbol $f \in \mathcal{F}$, the first k argument positions are normal, and the remaining argument positions are safe, i.e., $\mathsf{safe}(f) = \{k + 1, \dots, k + l\}$. This allows use to write terms in *predicative!notation*

$$f(s_1, \dots, s_k; s_{k+1}, \dots, s_{k+l})\,,$$

where separation of safe from normal arguments is directly indicated in terms.

Definition 9.4 (Safe equivalence)**.** Let $\sim$ denote the equivalence underlying a precedence $\succsim$. We define *safe equivalence* $\approx_{\mathsf{s}}$ on terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ inductively as follows: $s \approx_{\mathsf{s}} t$ if one of the following alternatives hold:

(1) $s = t$; or

(2) $s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l})$, $t = g(t_1, \ldots, t_k; t_{k+1}, \ldots, t_{k+l})$ with $f \sim g$ and $s_i \approx_{\mathsf{s}} t_i$ holds for all $i = 1, \ldots, k + l$.

Definition 9.5 (Normal Sub-Term Modulo Equivalence)**.**

(1) We define the relation $\trianglelefteq\!\!\!/_{\approx}$ on terms as the relation $\trianglelefteq \cdot \approx$ (or equivalently, $\approx \cdot \trianglelefteq \cdot \approx$). If $s \trianglelefteq\!\!\!/_{\approx} t$ holds then s *is called a sub-term of* t *modulo* $\approx$.

(2) We define $\overset{\mathsf{n}}{\trianglelefteq}\!\!\!/_{\approx} \subseteq \trianglelefteq\!\!\!/_{\approx}$ so that $s \overset{\mathsf{n}}{\trianglelefteq}\!\!\!/_{\approx} t$ holds if s is a *normal sub-term modulo* $\approx$ of t, i.e., $t = f(t_1, \ldots, t_k; t_{k+1}, \ldots, t_{k+l})$ and $s \trianglelefteq\!\!\!/_{\approx} t_i$ for some *normal* argument t_i of t ($i \in \{1, \ldots, k\}$). The inverse of $\overset{\mathsf{n}}{\trianglelefteq}\!\!\!/_{\approx}$ is denoted by $\overset{\mathsf{n}}{\trianglerighteq}\!\!\!/_{\approx}$.

Definition 9.6 (Small Polynomial Path Order)**.** Let $\succsim$ denote a precedence on $\mathcal{F}$, with underlying proper order $\succ$ and equivalence $\sim$. Let $\mathcal{K} \subseteq \mathcal{D}$ denote a set of recursive function symbols, and fix a safe mapping. Then $s \succ_{\mathsf{spop}^\star} t$ for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l})$ if one of the following alternatives hold.

(1) $s_i \succsim_{\mathsf{spop}^\star} t$ for some argument s_i of s ($i \in \{1, \ldots, k + l\}$); or

(2) $f \in \mathcal{D}$, $t = g(t_1, \ldots, t_m; t_{m+1}, \ldots, t_{m+n})$ where $f \succ g$ and the following holds:

- $s \overset{\mathsf{n}}{\trianglerighteq}\!\!\!/_{\approx} t_j$ for all normal arguments $t_1, \ldots, t_m$; and
- $s \succ_{\mathsf{spop}^\star} t_j$ for all safe arguments $t_{m+1}, \ldots, t_{m+n}$; and
- t contains at most one function symbols g with $f \sim g$; or

(3) $f, g \in \mathcal{K}$, $t = g(t_1, \ldots, t_k; t_{k+1}, \ldots, t_{k+l})$ where $f \sim g$ and the following holds:

- $\langle s_1, \ldots, s_k \rangle \succ_{\mathsf{spop}^\star} \langle t_1, \ldots, t_k \rangle$; and
- $\langle s_{k+1}, \ldots, s_{k+l} \rangle \succsim_{\mathsf{spop}^\star} \langle t_{k+1}, \ldots, t_{k+l} \rangle$.

Here $s \succsim_{\mathsf{spop}^\star} t$ denotes that either $s \approx_{\mathsf{s}} t$ or $s \succ_{\mathsf{spop}^\star} t$ holds. In the last clause we use $\succ_{\mathsf{spop}^\star}$ also for the extension of $\succ_{\mathsf{spop}^\star}$ to products: $\langle s_1, \ldots, s_n \rangle \succsim_{\mathsf{spop}^\star} \langle t_1, \ldots, t_n \rangle$ means $s_i \succsim_{\mathsf{spop}^\star} t_i$ for all $i = 1, \ldots, n$, and $\langle s_1, \ldots, s_n \rangle \succ_{\mathsf{spop}^\star} \langle t_1, \ldots, t_n \rangle$ indicates that additionally $s_{i_0} \succ_{\mathsf{spop}^\star} t_{i_0}$ holds for at least one $i_0 \in \{1, \ldots, n\}$.

Definition 9.7 (Predicative Recursive TRS)**.** We call a TRS $\mathcal{R}$ *predicative recursive (of degree d)* if $\mathcal{R}$ is *compatible* with an instance of sPOP* and the maximal depth of recursion $\mathsf{rd}_{\mathcal{R}}(f)$ of $f \in \mathcal{F}$ is d. Here compatibility means that $l \succ_{\mathsf{spop}^\star} r$ holds for all rules $l \to r$, where the precedence $\succsim$ and recursive symbols $\mathcal{K}$ underlying the definition $\succ_{\mathsf{spop}^\star}$ correspond to the ones given by the rewrite system $\mathcal{R}$, compare Definition 9.1.

Theorem 9.8. *Suppose $\mathcal{R}$ is a predicative recursive constructor TRS of degree d. Then the innermost derivation height of any basic term $f(\vec{u};\vec{v})$ is bounded by a polynomial of degree $\mathsf{rd}_{\mathcal{R}}(f)$ in the sum of the depths of normal arguments $\vec{u}$. In particular, the innermost runtime function $\mathrm{rci}_{\mathcal{R}}$ is bounded by a polynomial function of degree d.*

The admittedly technical proof is postponed to the next section.

Remark. We remark that the decision problem, which asks if a TRS is compatible with an RPO, is **NP** complete [50]. On the other hand, one can checks if a given TRS is a constructor TRS, and constructs a compatible order $\succ_{\mathsf{spop}^\star}$, in deterministic polynomial time. Here, one essentially uses that the precedence is pre-determined. This removes the choice between clauses $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$ and $\succ^{\langle 3\rangle}_{\mathsf{spop}^\star}$.

Before we continue, we want to motivate the order $\succ_{\mathsf{spop}^\star}$ informally, and through various examples. We use $\succ^{\langle \mathrm{i}\rangle}_{\mathsf{spop}^\star}$ to refer to the i^{th} case in Definition 9.6, in particular we write $s \succ^{\langle \mathrm{i}\rangle}_{\mathsf{spop}^\star} t$ if $s \succ_{\mathsf{spop}^\star} t$ follows by case $\succ^{\langle \mathrm{i}\rangle}_{\mathsf{spop}^\star}$. A similar notation will be employed for the consecutively defined orders.

Consider a rule $l \to r \in \mathcal{R}$ from a predicative recursive constructor TRS $\mathcal{R}$. The case $\succ^{\langle 1\rangle}_{\mathsf{spop}^\star}$ is standard in recursive path orders, here it is the only case that allows the comparison of constructor terms.

The case $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$ imposes a restricted composition scheme on predicative recursive TRSs. The order constraints on normal argument positions ensure that in an application of the rule $l \to r$, only normal arguments of l are passed to normal arguments in r. This fulfils two purposes. First of all, it keeps the predicative separation intact. But also, its ensures that under normal argument positions in r no further function calls are introduced. This prohibits for instance the orientation of the rule

$$\mathsf{g}(x;) \to \mathsf{h}(\mathsf{r}(x;), \mathsf{s}(x;);) \,,$$

since the resulting constraint $\mathsf{g}(x;) \rhd\!\!\!/_{\approx} \mathsf{r}(x;)$ cannot be satisfied. Precisely this restriction allows us to bind the complexity of compatible TRSs in the depth of recursion only. The last restriction put onto $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$ is used to prohibit multiple recursive calls. For instance, the rule

$$\mathsf{e}(\mathsf{s}(;x);) \to \mathsf{c}(;\mathsf{e}(x;), \mathsf{e}(x;)) \,,$$

which gives rise to exponentially long reductions, cannot be oriented.

The clause $\succ^{\langle 3\rangle}_{\mathsf{spop}^\star}$ is used to handle recursive calls in right-hand sides, and is therefor only applicable to recursive symbols. Beside enforcing that recursion parameters are non-increasing, the order constraints dictate that at least one normal argument is decreasing. The recursion scheme imposed on compatible TRSs is more general than predicative recursion. Still, as in predicative recursion, the number of recursions depends only on normal arguments.

Example 9.9 (Continued from Example 9.2). Recall that the precedence underlying $\mathcal{R}_{\mathsf{arith}}$ is given by

$$\mathsf{f} \;\succ\; \times \;\succ\; + \;\succ\; \mathsf{s} \;\sim\; 0 \,,$$

and that $\mathcal{K}_{\mathcal{R}_{\mathsf{arith}}} = \{\times, +\}$. The degree of recursion of $\mathcal{R}_{\mathsf{arith}}$ is equal to 2. Consider the safe mapping given by

$$\mathsf{safe}(+) = \{2\} \quad \mathsf{safe}(\times) = \varnothing \quad \mathsf{safe}(\mathsf{f}) = \varnothing \quad \mathsf{safe}(\mathsf{s}) = \{1\} \quad \mathsf{safe}(0) = \varnothing \,.$$

We show that the constructor TRS $\mathcal{R}_{\mathsf{arith}}$ depicted in Example 9.2 is predicative recursive. The case $\succ^{\langle 1\rangle}_{\mathsf{spop}^\star}$ allows the treatment of projections as in rules 3 and 5: $+(0;y) \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} y$ holds as $y \approx y$, likewise $\times(0,y;) \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} 0$ using $0 \approx 0$.

Consider the rule

$$7\colon \mathsf{f}(x,y;) \to +(x;\times(y,y;)) \,.$$

It is oriented by $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$ only. Using $\mathsf{f} \succ \times$ and twice $\mathsf{f}(x,y;) \rhd^{\mathsf{n}}_{\approx} y$ we obtain $\mathsf{f}(x,y;) \succ^{\langle 2\rangle}_{\mathsf{spop}^\star} \times(y,y;)$. Using that also $\mathsf{f} \succ +$ and $\mathsf{f}(x,y;) \rhd^{\mathsf{n}}_{\approx} x$ hold, we obtain $\mathsf{f}(x,y;) \succ^{\langle 2\rangle}_{\mathsf{spop}^\star} +(x;\times(y,y;))$. Observe that the order constraints propagate that both arguments to f are normal.

Using additionally $\succ^{\langle 3\rangle}_{\mathsf{spop}^\star}$, we can orient the remaining rules 4 and 6 that define the recursion case of addition and multiplication respectively. We exemplify this on the rule

$$6\colon \times(\mathsf{s}(;x),y;) \to +(y;\times(x,y;)) \,.$$

One application of $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$ simplifies the orientation of rule 6 to $+(\mathsf{s}(;x),y;) \rhd^{\mathsf{n}}_{\approx} y$ and $\times(\mathsf{s}(;x),y;) \succ_{\mathsf{spop}^\star} \times(x,y;)$. The former constraint is satisfied by definition. Since $\times$ is recursive, using $\succ^{\langle 3\rangle}_{\mathsf{spop}^\star}$ the latter constraint reduces to $\langle \mathsf{s}(;x), y\rangle \succ_{\mathsf{spop}^\star} \langle x, y\rangle$, which holds as $\mathsf{s}(;x) \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} x$ and $y \approx_{\mathsf{s}} y$, and the trivial constraint $\langle\rangle \succsim_{\mathsf{spop}^\star} \langle\rangle$.

Note that any other partitioning of argument positions of multiplication invalidates the orientation of rule 6. The sub-constraint $\times(\mathsf{s}(;x),y) \succ_{\mathsf{spop}^\star} \times(x,y)$ requires that at least the first argument position of times is normal, the sub-constraint $\times(\mathsf{s}(;x),y;) \rhd^{\mathsf{n}}_{\approx} y$, which propagates that y is a recursion parameter of addition, enforces that also the second argument position of $\times$ is normal.

By Theorem 9.8 we obtain that addition admits linear, and multiplication as well as f admits quadratic innermost runtime complexity. Overall the innermost runtime complexity of $\mathcal{R}_{\mathsf{arith}}$ is quadratic. ◁

The following examples clarifies the need for data tiering.

Example 9.10 (Continued from Example 9.9). Consider the extension of $\mathcal{R}_{\mathsf{arith}}$ by the two rules

$$8\colon \mathsf{exp}(0,y) \to \mathsf{s}(;0) \qquad 9\colon \mathsf{exp}(\mathsf{s}(;x),y) \to \times(y,\mathsf{exp}(x,y);) \,,$$

that express exponentiation y^x. This definition of the exponential function cannot be formulated with Bellantoni and Cook's scheme **SRN**, since the recursive result $\mathsf{exp}(x,y)$ is substituted as recursion parameter to multiplication. In our setting, orientation of the rule 9 reduces to $\mathsf{exp}(\mathsf{s}(;x),y) \rhd^{\mathsf{n}}_{\approx} \mathsf{exp}(x,y)$, and thus the extended rewrite system is not predicative recursive. ◁

The next example is negative, in the sense that the considered TRSs admits polynomial runtime complexity, but fails to be compatible with sPOP*.

Example 9.11 (Continued from Example 9.10). Consider the TRS $\mathcal{R}_{\mathsf{arith}}$ where the rule 6 is replaced by the rule

$$\texttt{6a:}\ \ \times(\mathsf{s}(;x),y;) \to +(\times(x,y;);y)\ .$$

The resulting system admits polynomial runtime complexity. On the other hand, Theorem 9.8 is inapplicable since the system is not predicative recursive. ◁

The next two examples stress that the restriction to innermost reductions, as well as constructor TRSs, is essential for the correctness of Theorem 9.8.

Example 9.12. Consider the constructor TRS $\mathcal{R}_{\mathsf{dup}}$ given by the following rules:

$$\begin{aligned}
&\texttt{10:}\ \ \mathsf{btree}(0;) \to \mathsf{leaf} && \texttt{11:}\ \mathsf{dup}(;t) \to \mathsf{c}(t,t)\\
&\texttt{12:}\ \mathsf{btree}(\mathsf{s}(;n);) \to \mathsf{dup}(;\mathsf{btree}(n))\ ,
\end{aligned}$$

that computes a binary tree of height n on input $\mathsf{s}^n(;0)$, compare Example 3.1 on page 27 that gives a tail-recursive definition. Observe that $\mathcal{R}_{\mathsf{dup}} \subseteq \succ_{\mathsf{spop}^\star}$ using that $\mathsf{btree} \succ \mathsf{dup} \succ \mathsf{c} \sim \mathsf{leaf}$ and the safe mapping as indicated in the rules. Theorem 9.8 thus implies that the innermost runtime complexity of $\mathcal{R}_{\mathsf{dup}}$ is polynomial. On the other hand, $\mathcal{R}_{\mathsf{dup}}$ admits exponentially long outermost reductions. ◁

Example 9.13. Consider the TRS $\mathcal{R}_{\mathsf{nc}}$ given by the rules

$$\begin{aligned}
&\texttt{13:}\ \ \mathsf{f}(n;) \to \mathsf{h}(;\mathsf{gs}(n;)) && \texttt{14:}\ \ \mathsf{gs}(0;) \to 0\\
&\texttt{15:}\ \mathsf{h}(;\mathsf{g}(;n)) \to \mathsf{c}(;\mathsf{h}(;n),\mathsf{h}(;n)) && \texttt{16:}\ \mathsf{gs}(\mathsf{s}(;n);) \to \mathsf{g}(;\mathsf{gs}(n;))\\
&\texttt{17:}\ \ \mathsf{g}(;\bot) \to \mathsf{c}(;\mathsf{h}(;\bot),\mathsf{h}(;\bot))\ ,
\end{aligned}$$

where we suppose that the only constructors are $\bot$, 0 and s. Rule 17 is used to define the symbol g, and to properly set up the precedence. The rules 14 and 16 are used to translate a tower $\mathsf{s}^n(;0)$ to $\mathsf{g}^n(;0)$, using rule 13 we thus obtain a family of reductions

$$\mathsf{f}(\mathsf{s}^n(;0);) \xrightarrow{\mathsf{i}}_{\mathcal{R}_{\mathsf{nc}}} \mathsf{h}(;\mathsf{gs}(\mathsf{s}^n(;0);)) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_{\mathsf{nc}}} \mathsf{h}(;\mathsf{g}^n(;0))\ ,$$

for $n \in \mathbb{N}$. It is not difficult to see that the derivation height of the final term $\mathsf{h}(;\mathsf{g}^n(;0))$ with respect to $\xrightarrow{\mathsf{i}}_{\mathcal{R}_{\mathsf{nc}}}$ grows exponentially in n due to rule 15, and so the innermost runtime complexity of $\mathcal{R}_{\mathsf{nc}}$ is bounded by an exponential from below.

On the other hand, this system is compatible with sPOP*. Note that precedence $\gtrsim$ induced by $\mathcal{R}_{\mathsf{nc}}$ satisfies in particular

$$\mathsf{f} \ \succ\ \mathsf{gs} \ \succ\ \mathsf{g} \ \succ\ \mathsf{h} \ \succ\ \mathsf{c}\ .$$

Using this precedence, and the safe mapping as indicated in the rules, it is not difficult to see that $\mathcal{R}_{\mathsf{nc}}$ is compatible with a polynomial path order. Observe that for rule 15 we exploit that g is defined, hence one can show $\mathsf{g}(;m) \succ_{\mathsf{spop}^\star} \mathsf{c}(;\mathsf{h}(;m),\mathsf{h}(;m))$ and therefore $\mathsf{h}(;\mathsf{g}(;m)) \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} \mathsf{c}(;\mathsf{h}(;m),\mathsf{h}(;m))$ holds. Note that due to rule 17, $\mathcal{R}_{\mathsf{nc}}$ is not a constructor TRS as demanded in Theorem 9.8.◁

We emphasise that the bound provided in Theorem 9.8 is tight in the sense that for any $d \in \mathbb{N}$ we can define a predicative TRS $\mathcal{R}_d$ of degree d with innermost runtime complexity in $\Omega(n^d)$. This is clarified in the following example.

Example 9.14. We define a family of TRSs $\mathcal{R}_d$ ($d \in \mathbb{N}$) inductively as follows: $\mathcal{R}_0 := \{\mathsf{f}_0(x;) \to \mathsf{a}\}$ and $\mathcal{R}_{d+1}$ extends $\mathcal{R}_d$ by the rules

$$\mathsf{f}_{d+1}(x;) \to \mathsf{g}_{d+1}(x, x;) \qquad \mathsf{g}_{d+1}(\mathsf{s}(;x), y;) \to \mathsf{b}(;\mathsf{f}_d(y;), \mathsf{g}_{d+1}(x, y;)) \;.$$

Let $d \in \mathbb{N}$. It is easy to see that $\mathcal{R}_d$ is predicative recursive where the underlying precedence fulfils

$$\mathsf{f}_d \succ \mathsf{g}_d \succ \mathsf{f}_{d-1} \succ \mathsf{g}_{d-1} \succ \cdots \succ \mathsf{f}_0 \succ \mathsf{a} \sim \mathsf{b} \;.$$

As only g_i ($i = 1, \ldots, d$) are recursive, the recursion depth of $\mathcal{R}_d$ is d. By Theorem 9.8, the runtime complexity thus lies in $\mathcal{O}(n^d)$. But also the runtime complexity of $\mathcal{R}_d$ is in $\Omega(n^d)$: For $d = 0$ this is immediate. Otherwise, consider the term $\mathsf{f}_{d+1}(\mathsf{s}^n(;\mathsf{a});)$ ($n \in \mathbb{N}$) which reduces to $\mathsf{g}_{d+1}(\mathsf{s}^n(;\mathsf{a}), \mathsf{s}^n(;\mathsf{a});)$ in one step. As the latter iterates $\mathsf{f}_d(\mathsf{s}^n(\mathsf{a}))$ for n times, the lower bound is established by inductive reasoning. ◁

Finally we note that the order $\succ_{\mathsf{spop}^\star}$ is *blind* on constructors, in particular $\succ_{\mathsf{spop}^\star}$ collapses to the sub-term relation (modulo equivalence) on values. This is a consequence of the following lemma, which holds in particular when the quasi-precedence $\succsim$ is instantiated by the precedence underlying a predicative recursive TRS.

Lemma 9.15. *Let $s \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ be a constructor term and let $\succsim$ denote an admissible precedence. Let $\approx$ denote the equivalence on terms given by the equivalence underlying $\succsim$.*

(1) *If $s \approx t$ then $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$;*

(2) *If $s \succ_{\mathsf{spop}^\star} t$ then $s \rhd\!\!\!/_{\approx} t$, in particular $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$.*

Proof. The first assertion follows by standard induction on s, using the assumptions on $\sim$. For the second assertion, consider terms $s \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ and t with $s \succ_{\mathsf{spop}^\star} t$. Observe that since s contains only constructors, $s \succ_{\mathsf{spop}^\star} t$ follows exclusively from $\succ^{\langle 1 \rangle}_{\mathsf{spop}^\star}$, and hence $s \rhd\!\!\!/_{\approx} t$ holds. We conclude $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ from the first assertion. □

9.1. Small Polynomial Path Orders are Sound

In this section prove Theorem 9.8. By the polynomial invariance theorem we then obtain that the functions computed by a (confluent) predicative recursive constructor TRSs are polytime computable. Suppose $\mathcal{R}$ is a predicative recursive constructor TRS of degree d. Our proof makes use of a variety of ingredients. As a first step, we introduce an auxiliary order $\succ_{\mathcal{K},\ell}$, the *small polynomial path order on sequences*. Although this auxiliary order is admittedly technical, it is

easier to reason about its induced complexity. We then define the *predicative interpretation* $\mathcal{I}_{\mathcal{R}}$ which *embeds* $\mathcal{R}$ reductions into $\succ_{\mathcal{K},\ell}$, compare Figure 9.1. Consequently the derivation height of any term s is bounded by the length of $\succ_{\mathcal{K},\ell}$ descending sequences starting from $\mathcal{I}_{\mathcal{R}}(s)$, which in turn can be bounded sufficiently whenever s is basic (cf. Theorem 9.24).

$$\begin{array}{ccccccccc} s & & \xrightarrow{\mathsf{i}}_{\mathcal{R}} & s_1 & \xrightarrow{\mathsf{i}}_{\mathcal{R}} & \cdots & \xrightarrow{\mathsf{i}}_{\mathcal{R}} & s_\ell \\ \downarrow & & & \downarrow & & & & \downarrow \\ \mathcal{I}_{\mathcal{R}}(s) & \succ_{\mathcal{K},\ell} & \mathcal{I}_{\mathcal{R}}(s_1) & \succ_{\mathcal{K},\ell} & \cdots & & \succ_{\mathcal{K},\ell} & \mathcal{I}_{\mathcal{R}}(s_\ell) \end{array}$$

Figure 9.1.: Predicative Embedding of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ into $\succ_{\mathcal{K},\ell}$.

9.1.1. Small Polynomial Path Order on Sequences

The *small polynomial path order on sequences* constitutes a miniaturisation of the *path order for* **FP** as put forward in [4]. To formalise sequences of terms, we use an auxiliary variadic function symbol $\circ$. Here variadic means that the arity of $\circ$ is finite but arbitrary.

Definition 9.16 (Sequences of Terms)**.** Let $\circ \notin \mathcal{F}$ be a fresh variadic function symbol. A term $t \in \mathcal{T}(\mathcal{F} \cup \{\circ\}, \mathcal{V})$ is called a *sequence* if it is of the form $\circ(t_1, \ldots, t_k)$ for $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ $(i = 1, \ldots, k)$. The set of all *sequences* is denoted by $\mathcal{T}^\star(\mathcal{F}, \mathcal{V})$.

We always write $[t_1, \ldots, t_k]$ for $\circ(t_1, \ldots, t_k)$, and if we write $f(t_1, \ldots, t_k)$ then we implicitly assume $f \neq \circ$. We denote by $a, b, \ldots$ elements of $\mathcal{T}(\mathcal{F} \cup \{\circ\}, \mathcal{V})$, and we use $s, t, \ldots$ to denote terms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Abusing set-notation, we write $t \in [t_1, \ldots, t_k]$ if $t = t_i$ for some $i \in \{1, \ldots, k\}$. We denote by $a +\!\!+ b$ the *concatenation* of sequences. To avoid notational overhead we overload concatenation to both terms and sequences.

Definition 9.17 (Concatenation of Sequences)**.** For $a, b \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F}, \mathcal{V})$ we set

$$a +\!\!+ b := [s_1 \ \cdots \ s_k \ t_1 \ \cdots \ t_l] \,,$$

where $[s_1, \ldots, s_k] = \mathrm{lift}(a)$ and $[t_1, \ldots, t_l] = \mathrm{lift}(b)$. Here lift is defined so that $\mathrm{lift}(t) = [t]$ for $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and otherwise $\mathrm{lift}(a) = a$ for $a \in \mathcal{T}^\star(\mathcal{F}, \mathcal{V})$.

We extend the equivalence $\approx$ on terms given in Definition 2.52 to an equivalence on sequences in the obvious way.

Definition 9.18. We define $[s_1, \ldots, s_k] \approx [t_1, \ldots, t_k]$ if $s_i \approx t_i$ holds for all argument positions $i = 1, \ldots, k$.

Following the spirit of *finite approximations* of recursive path orders by Buchholz [24], the polynomial path orders on sequence is parameterised in a natural number ℓ, which is used to control the width of sequences. The second parameter $\mathcal{K}$ refers to a set of recursive symbols.

Definition 9.19. Let $\gtrsim$ denote a quasi-precedence on $\mathcal{F}$, with underlying proper order $\succ$ and equivalence $\sim$. Let $\ell \in \mathbb{N}$ with $\ell \geqslant 1$ and $\mathcal{K} \subseteq \mathcal{F}$ be a set of function symbols. Then $a \succ_{\mathcal{K},\ell} b$ holds for terms or sequences of terms $a, b \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$ if one of the following alternatives hold.

(1) $a = f(s_1,\dots,s_k)$, $b = g(t_1,\dots,t_l)$ with $f \succ g$ and the following conditions hold:
- $f(s_1,\dots,s_k) \rhd\!\!/_{\approx} t_j$ for all $j = 1,\dots,l$; and
- $l \leqslant \ell$; or

(2) $a = f(s_1,\dots,s_k)$, $b = g(t_1,\dots,t_k)$ with $f \sim g$ and the following conditions hold:
- $f, g \in \mathcal{K}$; and
- $\langle s_1,\dots,s_k\rangle \rhd\!\!/_{\approx} \langle t_1,\dots,t_k\rangle$; or

(3) $a = f(s_1,\dots,s_k)$, $b = [t_1,\dots,t_l]$ and the following conditions hold:
- $f(s_1,\dots,s_k) \succ_{\mathcal{K},\ell} t_j$ for all $j = 1,\dots,l$; and
- there is at most one occurrence of a symbol g in b with $f \sim g$; and
- $l \leqslant \ell$.

(4) $a = [s_1,\dots,s_k]$, $b = [t_1,\dots,t_l]$ and there exists terms *or* sequences b_i $(i = 1,\dots,k)$ such that:
- $[t_1,\dots,t_l] = b_1 +\!\!+ \cdots +\!\!+ b_k$; and
- $\langle s_1,\dots,s_k\rangle \succ_{\mathcal{K},\ell} \langle b_1,\dots,b_k\rangle$.

We denote by $a \gtrsim_{\mathcal{K},\ell} b$ that either $a \approx b$ or $a \succ_{\mathcal{K},\ell} b$ holds. We use $\rhd\!\!/_{\approx}$ and $\succ_{\mathcal{K},\ell}$ also for their extension to products: $\langle a_1,\dots,a_k\rangle \rhd\!\!/_{\approx} \langle b_1,\dots,b_k\rangle$ if $a_i \trianglerighteq\!\!/_{\approx} b_i$ for all $i = 1,\dots,k$, and $a_{i_0} \rhd\!\!/_{\approx} a_{i_0}$ for at least one $i_0 \in \{1,\dots,k\}$; likewise $\langle a_1,\dots,a_k\rangle \succ_{\mathcal{K},\ell} \langle b_1,\dots,b_k\rangle$ if $a_i \gtrsim_{\mathcal{K},\ell} b_i$ for all $i = 1,\dots,n$, and $a_{i_0} \succ_{\mathcal{K},\ell} a_{i_0}$ for at least one $i_0 \in \{1,\dots,k\}$.

The following lemma collects frequently used properties of $\succ_{\mathcal{K},\ell}$.

Lemma 9.20. *Let $\ell \geqslant 1$ and $\mathcal{K} \subseteq \mathcal{F}$. The order $\succ_{\mathcal{K},\ell}$ satisfies the following properties:*

(1) $\succ_{\mathcal{K},\ell} \subseteq \succ_{\mathcal{K},\ell+1}$*; and*

(2) $\approx \cdot \succ_{\mathcal{K},\ell} \subseteq \succ_{\mathcal{K},\ell}$ *and* $\succ_{\mathcal{K},\ell} \cdot \approx \subseteq \succ_{\mathcal{K},\ell}$*; and*

(3) *for all $a, b, c_1, c_2 \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$,*

$$a \succ_{\mathcal{K},\ell} b \implies c_1 +\!\!+ a +\!\!+ c_2 \succ_{\mathcal{K},\ell} c_1 +\!\!+ b +\!\!+ c_2 \,.$$

Proof. Properties (1) follows by definition, and Property (2) by a standard induction on the definition of $\succ_{\mathcal{K},\ell}$. To prove the last property, suppose $a \succ_{\mathcal{K},\ell} b$

holds. Set $[u_1, \ldots, u_k] := \mathrm{lift}(c_1)$ and $[v_1, \ldots, v_l] := \mathrm{lift}(c_2)$, and observe that by the overloading of $+\!\!+$ we have

$$c_1 = u_1 +\!\!+ \cdots +\!\!+ u_k \quad \text{and} \quad c_2 = v_1 +\!\!+ \cdots +\!\!+ v_l \ .$$

If $a = f(s_1, \ldots, s_m)$ is a term, then by assumption $f(s_1, \ldots, s_m) \succ_{\mathcal{K},\ell} b$ we have

$$\langle u_1, \ldots, u_k, f(s_1, \ldots, s_m), v_1, \ldots, v_l \rangle \succ_{\mathcal{K},\ell} \langle u_1, \ldots, u_k, b, v_1, \ldots, v_l \rangle \ ,$$

and thus

$$c_1 +\!\!+ f(s_1, \ldots, s_m) +\!\!+ c_2 \succ^{\langle 4 \rangle}_{\mathcal{K},\ell} c_1 +\!\!+ b +\!\!+ c_2 \ ,$$

holds as desired. Otherwise $a = [s_1, \ldots, s_m]$, and the assumption can be strengthened to $a \succ^{\langle 4 \rangle}_{\mathcal{K},\ell} b$. By definition $b = b_1 +\!\!+ \cdots +\!\!+ b_m$ for some terms or sequences b_j $(j = 1, \ldots, m)$ with

$$\langle s_1, \ldots, s_m \rangle \succ_{\mathcal{K},\ell} \langle b_1, \ldots, b_m \rangle \ .$$

From this we obtain

$$\langle u_1, \ldots, u_k, s_1, \ldots, s_m, v_1, \ldots, v_l \rangle \succ_{\mathcal{K},\ell} \langle u_1, \ldots, u_k, b_1, \ldots, b_m, v_1, \ldots, v_l \rangle \ .$$

Hence again the property follows by one application of $\succ^{\langle 4 \rangle}_{\mathcal{K},\ell}$. □

Lemma 9.21. *Let $\ell \geqslant 1$ and $\mathcal{K} \subseteq \mathcal{F}$, and let $\succsim$ be a quasi-precedence.*

(1) *The order $\succ_{\mathcal{K},\ell}$ is finitely branching; and*

(2) *The order $\succ_{\mathcal{K},\ell}$ is well-founded.*

Proof. We consider the first assertion. We need to prove that for all $a \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F}, \mathcal{V})$, the set $\{b \mid a \succ_{\mathcal{K},\ell} b\}$ is finite. Fix a and suppose $a \succ_{\mathcal{K},\ell} b$ holds. Consider first the case that $a = f(s_1, \ldots, s_k)$ is a term. If $b \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ then $f(s_1, \ldots, s_k) \succ^{\langle 1 \rangle}_{\mathcal{K},\ell} b$ or $f(s_1, \ldots, s_k) \succ^{\langle 2 \rangle}_{\mathcal{K},\ell} b$ holds. In both cases we see, by the order constraints on arguments, that the depth of b is bounded by the depth of $f(s_1, \ldots, s_k)$ say d. Similar, for $b = [t_1, \ldots, t_l]$ we obtain that the depth of all terms t_j $(j = 1, \ldots, l)$ is bounded by the depth of $f(s_1, \ldots, s_k)$. Since there are only finitely many terms of depth d, and for the case $b = [t_1, \ldots, t_l]$ the length l is bounded by ℓ, we conclude the case for all terms a. Finally consider the case $a = [s_1, \ldots, s_k]$. Then $[s_1, \ldots, s_k] \succ_{\mathcal{K},\ell} b$ implies that $b = b_1 +\!\!+ \cdots +\!\!+ b_k$, with $s_i \succsim_{\mathcal{K},\ell} b_i$ for all $i = 1, \ldots, k$. Using that $\approx$ preserves the depth, we conclude as above that the set $\{b_i \mid s_i \succsim_{\mathcal{K},\ell} b_i\}$ is finite. As b can only be decomposed into terms or sequences b_i $(i = 1, \ldots, k)$ with $b = b_1 +\!\!+ \cdots +\!\!+ b_k$, the claim also follows for this case.

To prove the second assertion, let $\succsim^\circ$ be the extension of $\succsim$ to $\mathcal{F} \cup \{\circ\}$ so that the variadic list symbol $\circ$ is minimal in $\succsim^\circ$. Then by induction on the definition of $\succ_{\mathcal{K},\ell}$ one can show that $\succ_{\mathcal{K},\ell} \subseteq \succ^\circ_{\mathsf{mpo}}$ holds for the multiset path order $\succ^\circ_{\mathsf{mpo}}$ induced by the quasi-precedence $\succ^\circ$. Since $\rhd/_\approx$ is included in $\succ^\circ_{\mathsf{mpo}}$, the only non-trivial case is $[s_1, \ldots, s_k] \succ^{\langle 4 \rangle}_{\mathcal{K},\ell} [t_1, \ldots, t_l]$, where $[t_1, \ldots, t_l] = b_1 +\!\!+ \cdots +\!\!+ b_k$ and

$$\langle s_1, \ldots, s_k \rangle \succ_{\mathcal{K},\ell} \langle b_1, \ldots, b_k \rangle \ .$$

Fix $i \in \{1,\dots,k\}$. If $b_i \in \mathcal{T}(\mathcal{F},\mathcal{V})$ then $s_i \gtrsim_{\mathcal{K},\ell} b_i$, otherwise $s_i \succ^{\langle 3\rangle}_{\mathcal{K},\ell} b_i$ with $b = [u_1,\dots,u_m]$ and thus $s_i \succ_{\mathcal{K},\ell} u_j$ for all $u = 1,\dots,m$. As at least one of the inequalities is strict, using the induction hypothesis it is not difficult to conclude

$$\{\!\{s_1,\dots,s_k\}\!\} \succ^{\circ}_{\mathsf{mpo}} \{\!\{t_1,\dots,t_l\}\!\} ,$$

and consequently $[s_1,\dots,s_k] \succ^{\circ}_{\mathsf{mpo}} [t_1,\dots,t_l]$ holds.

Since $\succ^{\circ}_{\mathsf{mpo}}$ is well-founded even on variadic signatures by Proposition 2.55, the second assertion follows from this. □

As a consequence of Lemma 9.21, the following function $\mathsf{G}_{\mathcal{K},\ell}$ is well-defined.

Definition 9.22. Let $\ell \geqslant 1$ and $\mathcal{K} \subseteq \mathcal{F}$. We define $\mathsf{G}_{\mathcal{K},\ell} : \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V}) \to \mathbb{N}$ by

$$\mathsf{G}_{\mathcal{K},\ell}(a) := \max\{\ l \ \mid \exists a_1,\dots,a_l.\ a \succ_{\mathcal{K},\ell} a_1 \succ_{\mathcal{K},\ell} \cdots \succ_{\mathcal{K},\ell} a_l\} .$$

Note that if $a \succ_{\mathcal{K},\ell} b$ then $\mathsf{G}_{\mathcal{K},\ell}(a) > \mathsf{G}_{\mathcal{K},\ell}(b)$ holds. Furthermore, as a consequence of Lemma 9.20(2) we obtain that if $a \approx b$ then $\mathsf{G}_{\mathcal{K},\ell}(a) = \mathsf{G}_{\mathcal{K},\ell}(b)$ holds. Hence $a \gtrsim_{\mathcal{K},\ell} b$ implies $\mathsf{G}_{\mathcal{K},\ell}(a) \geqslant \mathsf{G}_{\mathcal{K},\ell}(b)$, a fact that we use frequently below.

In the remaining of this section, we prove that $\mathsf{G}_{\mathcal{K},\ell}(a)$ is bounded by a polynomial in the depth of its argument.

Lemma 9.23. *Let $\ell \geqslant 1$ and $\mathcal{K} \subseteq \mathcal{F}$. For all sequences $[t_1,\dots,t_k] \in \mathcal{T}^\star(\mathcal{F},\mathcal{V})$ we have*

$$\mathsf{G}_{\mathcal{K},\ell}([t_1,\dots,t_k]) = \sum_{i=1}^{k} \mathsf{G}_{\mathcal{K},\ell}(t_i) .$$

Proof. Let $[t_1,\dots,t_k] \in \mathcal{T}^\star(\mathcal{F},\mathcal{V})$. As a consequence of Lemma 9.20(3),

$$\mathsf{G}_{\mathcal{K},\ell}(a + b) \geqslant \mathsf{G}_{\mathcal{K},\ell}(a) + \mathsf{G}_{\mathcal{K},\ell}(b) ,$$

holds for all $a, b \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$. Hence in particular

$$\mathsf{G}_{\mathcal{K},\ell}([t_1,\dots,t_k]) = \mathsf{G}_{\mathcal{K},\ell}(t_1 + \cdots + t_k) \geqslant \sum_{i=1}^{k} \mathsf{G}_{\mathcal{K},\ell}(t_i) .$$

To show the inverse direction, we proceed by induction on $\mathsf{G}_{\mathcal{K},\ell}([t_1,\dots,t_k])$, which is justified by Lemma 9.21. The base case $\mathsf{G}_{\mathcal{K},\ell}(a) = 0$ follows trivially. For the induction step, we show that for all $b \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$,

$$[t_1,\dots,t_k] \succ_{\mathcal{K},\ell} b \implies \mathsf{G}_{\mathcal{K},\ell}(b) < \sum_{i=1}^{k} \mathsf{G}_{\mathcal{K},\ell}(t_i) .$$

This implies $\mathsf{G}_{\mathcal{K},\ell}([t_1,\dots,t_k]) \leqslant \sum_{i=1}^{k} \mathsf{G}_{\mathcal{K},\ell}(t_i)$ as desired. Suppose $a \succ_{\mathcal{K},\ell} b$, which by definition of $\succ_{\mathcal{K},\ell}$ refines to $a \succ^{\langle 4\rangle}_{\mathcal{K},\ell} b$. Hence there exists terms or sequences b_i $(i = 1,\dots,k)$ such that $b = b_1 + \cdots + b_k$ and

$$\langle t_1,\dots,t_k\rangle \succ_{\mathcal{K},\ell} \langle b_1,\dots,b_k\rangle ,$$

holds. As a consequence, $\mathsf{G}_{\mathcal{K},\ell}(b_i) \leqslant \mathsf{G}_{\mathcal{K},\ell}(t_i)$ holds for all $i = 1, \dots, k$, where for at least one $i_0 \in \{1, \dots, k\}$ we even have $\mathsf{G}_{\mathcal{K},\ell}(b_{i_0}) < \mathsf{G}_{\mathcal{K},\ell}(s_{i_0})$. Using that

$$\mathsf{G}_{\mathcal{K},\ell}(b_i) \leqslant \mathsf{G}_{\mathcal{K},\ell}(b) < \mathsf{G}_{\mathcal{K},\ell}([t_1, \dots, t_k]) \quad \text{for all } i = 1, \dots, k,$$

induction hypothesis is applicable to b and all b_i ($i \in \{1, \dots, k\}$). Summing up we obtain

$$\mathsf{G}_{\mathcal{K},\ell}(b) = \sum_{s \in b} \mathsf{G}_{\mathcal{K},\ell}(s) = \sum_{i=1}^{k} \sum_{s \in b_i} \mathsf{G}_{\mathcal{K},\ell}(s) = \sum_{i=1}^{k} \mathsf{G}_{\mathcal{K},\ell}(b_i) < \sum_{i=1}^{k} \mathsf{G}_{\mathcal{K},\ell}(t_i) \,. \quad \square$$

Theorem 9.24. *Let $\ell \geqslant 1$ and $\mathcal{K} \subseteq \mathcal{F}$. For each $f \in \mathcal{F}$, there exists a function $p_f(n) \in \mathcal{O}(n^d)$ for $d := \mathsf{rd}_{\mathcal{K},\gtrsim}(f)$ such that*

$$\mathsf{G}_{\mathcal{K},\ell}(f(t_1, \dots, t_k)) \leqslant p_f\big(\textstyle\sum_{i=1}^{k} \mathsf{dp}(t_i)\big) \,,$$

for all terms $t_1, \dots, t_k \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

Proof. For all $d \in \mathbb{N}$, define the function $\mathsf{c}_d : \mathbb{N} \to \mathbb{N}$ as follows:

$$\mathsf{c}_d(r) := \begin{cases} 1 & \text{if } r \leqslant 1, \text{ and} \\ \mathsf{c}_d(r-1) \cdot \ell^{d+1} + 1 & \text{otherwise} \,. \end{cases}$$

We show that for all terms $s = f(s_1, \dots, s_k)$ and $b \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F}, \mathcal{V})$

$$f(s_1, \dots, s_k) \succ_{\mathcal{K},\ell} b \implies \mathsf{G}_{\mathcal{K},\ell}(b) < \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f)) \cdot \big(2 + \textstyle\sum_{i=1}^{k} \mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)} \,,$$

where $\mathsf{rd}_{\mathcal{K},\gtrsim}$ is abbreviated by rd. In the proof, we employ induction on $\mathsf{rk}(f)$ and side induction on $\sum_{i=1}^{k} \mathsf{dp}(t_i)$. Suppose $f(s_1, \dots, s_k) \succ_{\mathcal{K},\ell} b$ holds.

Consider the base case $\mathsf{rk}(f) = 1$, i.e., f is minimal with respect to $\succ$. If $f \notin \mathcal{K}$ or $\sum_{i=1}^{k} \mathsf{dp}(s_i) = 0$ then these restrictions imply that $f(s_1, \dots, s_k) \succ_{\mathcal{K},\ell} b$ can only hold when b is a sequence, in particular $b = [\,]$. Then $\mathsf{G}_{\mathcal{K},\ell}(b) = 0$ and the theorem holds. So suppose $f \in \mathcal{K}$. We consider two sub-cases: (i) $b \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and (ii) $b \in \mathcal{T}^\star(\mathcal{F}, \mathcal{V})$. For (i), by assumption $f(s_1, \dots, s_k) \succ^{\langle 2 \rangle}_{\mathcal{K},\ell} g(t_1, \dots, t_k)$ holds for $b = g(t_1, \dots, t_k)$ with $f \sim g$. Using the order constraints

$$\langle s_1, \dots, s_k \rangle \rhd\!/\!\approx \langle t_1, \dots, t_k \rangle \,,$$

we see that $\sum_{i=1}^{k} \mathsf{dp}(t_i) < \sum_{i=1}^{k} \mathsf{dp}(s_i)$ holds. Thus the side induction hypothesis is applicable on $g(t_1, \dots, t_k)$. Moreover, using that $g \in \mathcal{K}$ by the order constraints and that $\gtrsim$ is transitive, we obtain $\mathsf{rd}(g) = \mathsf{rd}(f) = 1$ and likewise $\mathsf{rk}(g) = \mathsf{rk}(f) = 1$. Summing up, we conclude

$$\begin{aligned} \mathsf{G}_{\mathcal{K},\ell}(b) &\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \big(2 + \sum_{i=1}^{k} \mathsf{dp}(t_i)\big) \\ &< \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f)) \cdot \big(2 + \sum_{i=1}^{k} \mathsf{dp}(s_i)\big) \end{aligned}$$

In the second sub-case by assumption $b = [t_1, \ldots, t_l]$ for some terms t_j ($j = 1, \ldots, l$) and thus $f(s_1, \ldots, s_k) \succ^{\langle 3 \rangle}_{\mathcal{K},\ell} [t_1, \ldots, t_l]$. The order constraints require $f(s_1, \ldots, s_k) \succ_{\mathcal{K},\ell} t_j$ for all $j = 1, \ldots, l$ with at most one occurrence of g with $f \sim g$ in some t_{j_0} ($j_0 \in \{1, \ldots, l\}$). The latter implies $l \leqslant 1$ in the considered case. For suppose otherwise, hence $f(s_1, \ldots, s_k) \succ^{\langle 1 \rangle}_{\mathcal{K},\ell} s_j$ for at least one $j \in \{1, \ldots, l\}$. This however contradicts the assumption of the base case that f is minimal in the precedence. The only non-trivial case is thus $l = 1$, which we conclude by the sub-case (i), but additionally employing Lemma 9.23. This concludes the base case $\mathsf{rk}(f) = 1$.

We now prove the inductive case $rk(f) > 1$. We perform case analysis on the last rule that concludes $f(s_1, \ldots, s_k) \succ_{\mathcal{K},\ell} b$.

- Case $f(s_1, \ldots, s_k) \succ^{\langle 1 \rangle}_{\mathcal{K},\ell} g(t_1, \ldots, t_l)$: Hence $f(s_1, \ldots, s_k) \rhd\!\!/\!\!\approx t_j$ holds for all $j = 1, \ldots, l$, and thus

$$\mathsf{dp}(t_j) \leqslant \max_{i=1}^{k} \mathsf{dp}(s_i) \leqslant \sum_{i=1}^{k} \mathsf{dp}(s_i) \qquad \text{for all } j = 1, \ldots, l\,.$$

  Using that $l \leqslant \ell$ and also $f \succ g$ under the assumption, we can use the induction hypothesis to conclude

$$\begin{aligned}
\mathsf{G}_{\mathcal{K},\ell}(g(t_1, \ldots, t_l)) &\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \big(2 + \sum_{j=1}^{l} \mathsf{dp}(t_j)\big)^{\mathsf{rd}(g)} \\
&\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \big(2 + \sum_{j=1}^{l} \sum_{i=1}^{k} \mathsf{dp}(s_i)\big)^{\mathsf{rd}(g)} \\
&\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \big(2 + \ell \cdot \sum_{i=1}^{k} \mathsf{dp}(s_i)\big)^{\mathsf{rd}(g)} \\
&\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \ell^{\mathsf{rd}(g)} \cdot \big(2 + \sum_{i=1}^{k} \mathsf{dp}(s_i)\big)^{\mathsf{rd}(g)}\,.
\end{aligned}$$

  As $f \succ g$, by definition $\mathsf{rd}(g) \leqslant \mathsf{rd}(f)$ and $\mathsf{rk}(g) < \mathsf{rk}(f)$. Hence

$$\mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \ell^{\mathsf{rd}(g)} < \mathsf{c}_{\mathsf{rd}(f)+1}(\mathsf{rk}(f) - 1) \cdot \ell^{\mathsf{rd}(f)} + 1 = \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f))$$

  and we conclude the theorem for this case.

- Case $f(s_1, \ldots, s_k) \succ^{\langle 2 \rangle}_{\mathcal{K},\ell} g(t_1, \ldots, t_k)$: Under these assumptions we have $f \sim g$, and $\langle s_1, \ldots, s_k \rangle \rhd\!\!/\!\!\approx \langle t_1, \ldots, t_k \rangle$ which implies $\sum_{i=1}^{l} \mathsf{dp}(t_j) < \sum_{i=1}^{k} \mathsf{dp}(s_i)$. Hence by induction hypothesis of the side induction we

conclude

$$\begin{aligned}\mathsf{G}_{\mathcal{K},\ell}(g(t_1,\dots,t_k)) &\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \big(2+\sum_{j=1}^{k}\mathsf{dp}(t_j)\big)^{\mathsf{rd}(g)} \\ &\leqslant \mathsf{c}_{\mathsf{rd}(g)}(\mathsf{rk}(g)) \cdot \big(1+\sum_{j=1}^{k}\mathsf{dp}(s_j)\big)^{\mathsf{rd}(g)} \\ &< \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f)) \cdot \big(2+\sum_{j=1}^{k}\mathsf{dp}(s_j)\big)^{\mathsf{rd}(f)} \,.\end{aligned}$$

In the last equation we employed $\mathsf{rk}(f) = \mathsf{rk}(g)$ and likewise $\mathsf{rd}(f) = \mathsf{rd}(g) \neq 0$, which holds as $f \sim g$ and $f, g \in \mathcal{K}$.

- Case $f(s_1,\dots,s_k) \succ^{\langle 3\rangle}_{\mathcal{K},\ell} [t_1,\dots,t_l]$: We consider two sub-cases. First suppose $f \notin \mathcal{K}$, and fix $j \in \{1,\dots,l\}$. In this case we can strengthen the order constraints to $f(s_1,\dots,s_k) \succ^{\langle 1\rangle}_{\mathcal{K},\ell} t_j$. As $f \in \mathcal{K}$, for all $g \in \mathcal{F}$ with $f \succ g$, $\mathsf{rk}(g) \leqslant \mathsf{rk}(f) - 1$ and $\mathsf{rd}(g) \leqslant \mathsf{rk}(f)$ hold. By substituting these bounds in the sub-case $f(s_1,\dots,s_k) \succ^{\langle 1\rangle}_{\mathcal{K},\ell} g(t_1,\dots,t_l)$ we see

$$\mathsf{G}_{\mathcal{K},\ell}(t_j) \leqslant \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f)-1) \cdot \ell^{\mathsf{rd}(f)} \cdot \big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)} \,.$$

As by the order constraints $l \leqslant \ell$, by Lemma 9.23 we thus obtain

$$\begin{aligned}\mathsf{G}_{\mathcal{K},\ell}([t_1,\dots,t_l]) &= \sum_{j=1}^{l}\mathsf{G}_{\mathcal{K},\ell}(t_j) \\ &\leqslant \ell \cdot \Big(\mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f)-1) \cdot \ell^{\mathsf{rd}(f)} \cdot \big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)}\Big) \\ &< \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f)) \cdot \big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)} \,.\end{aligned}$$

This concludes the case $f \notin \mathcal{K}$.

Now suppose $f \in \mathcal{K}$. The order constraints give one $j_0 \in \{1,\dots,l\}$ with $f(s_1,\dots,s_k) \succ^{\langle 1\rangle}_{\mathcal{K},\ell} t_{j_0}$ or $f(s_1,\dots,s_k) \succ^{\langle 2\rangle}_{\mathcal{K},\ell} t_{j_0}$, but $f(s_1,\dots,s_k) \succ^{\langle 1\rangle}_{\mathcal{K},\ell} t_j$ for all $j \in \{1,\dots,l\} \setminus \{j_0\}$.

Consider first an element t_j with $j \neq j_0$. Since $f \in \mathcal{K}$, whenever $f \succ g$ then $\mathsf{rk}(g) \leqslant \mathsf{rk}(f) - 1$ and $\mathsf{rd}(g) \leqslant \mathsf{rk}(f) - 1$ hold. Hence by substituting these bound in the corresponding sub-case above we get

$$\mathsf{G}_{\mathcal{K},\ell}(t_j) \;\leqslant\; \mathsf{c}_{\mathsf{rd}(f)-1}(\mathsf{rk}(f)-1) \cdot \ell^{\mathsf{rd}(f)-1} \cdot \big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)-1} \,.$$

Similar,

$$\mathsf{G}_{\mathcal{K},\ell}(t_{j_0}) \leqslant \mathsf{c}_{\mathsf{rd}(f)-1}(\mathsf{rk}(f)-1)\cdot \ell^{\mathsf{rd}(f)-1}\cdot \big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)-1} + \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f))\cdot\big(1+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)}\,.$$

Here the expression in the first line covers the case $f(s_1,\dots,s_k) \succ^{\langle 1\rangle}_{\mathcal{K},\ell} t_{j_0}$ as before. The expression in the second line accounts for the case $f(s_1,\dots,s_k) \succ^{\langle 2\rangle}_{\mathcal{K},\ell} t_{j_0}$, and is obtained exactly as in the corresponding sub-case above. Summing up, using Lemma 9.23 and $l \leqslant \ell$ we conclude

$$\begin{aligned}
&\mathsf{G}_{\mathcal{K},\ell}([t_1,\dots,t_l])\\
&\quad= \sum_{j=1}^{l}\mathsf{G}_{\mathcal{K},\ell}(t_j)\\
&\quad\leqslant \ell\cdot\Big(\mathsf{c}_{\mathsf{rd}(f)-1}(\mathsf{rk}(f)-1)\cdot\ell^{\mathsf{rd}(f)-1}\cdot\big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)-1}\Big)\\
&\qquad+\mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f))\cdot\big(1+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)}\\
&\quad< \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f))\cdot\big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)-1}\\
&\qquad+\mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f))\cdot\big(1+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)}\\
&\quad\leqslant \mathsf{c}_{\mathsf{rd}(f)}(\mathsf{rk}(f))\cdot\big(2+\sum_{i=1}^{k}\mathsf{dp}(s_i)\big)^{\mathsf{rd}(f)}\,.
\end{aligned}$$

□

9.1.2. Predicative Embedding

Let $\mathcal{R}$ denote a predicative recursive constructor TRS. We now establish the *predicative embedding* as depicted in Figure 9.1 on page 96. The *predicative interpretation* $\mathcal{I}_{\mathcal{R}}$ that we use in this embedding separates safe from normal arguments, resulting in a sequences of *normalised terms*.

Definition 9.25.

(1) For each $f \in \mathcal{F}$, let f_{n} denote a fresh function symbol. For $\mathcal{G} \subseteq \mathcal{F}$, we define the extension of $\mathcal{G}$ to *normalised symbols* by

$$\mathcal{G}_{\mathsf{n}} := \mathcal{G} \cup \big\{ f_{\mathsf{n}}/k \mid f \in \mathcal{G}, \mathsf{nrm}(f) = \{i_1,\dots,i_k\}\big\}\,.$$

(2) A term $t \in \mathcal{T}(\mathcal{F}_{\mathsf{n}},\mathcal{V})$ is called *normalised* if it is of the form $t = f_{\mathsf{n}}(t_1,\dots,t_k)$ for $t_i \in \mathcal{T}(\mathcal{F},\mathcal{V})$ $(i=1,\dots,k)$. The set of all *normalised terms* is denoted by $\mathcal{T}_{\mathsf{n}}(\mathcal{F},\mathcal{V})$.

(3) We denote by $\mathcal{T}^\star_{\mathsf{n}}(\mathcal{F},\mathcal{V})$ the set of sequences $[t_1,\dots,t_k] \in \mathcal{T}^\star(\mathcal{F}_{\mathsf{n}},\mathcal{V})$ with $t_i \in \mathcal{T}_{\mathsf{n}}(\mathcal{F},\mathcal{V})$ for all $i = 1,\dots,k$.

The interpretation $\mathcal{I}_\mathcal{R}$ maps a reducible term $f(s_1,\dots,s_k;s_{k+1},\dots,s_{k+l})$ to

$$[f_{\mathsf{n}}(s_1,\dots,s_k)] +\!\!+ a_{k+1} +\!\!+ \cdots +\!\!+ a_{k+l}\ ,$$

where f_{n} is a fresh function symbol, and the sequences a_i $(i = k+1,\dots,k+l)$ result from interpreting the safe arguments $s_{k+1},\dots,s_{k+l}$. Irreducible terms are simply deleted, i.e., interpreted as an empty sequence. This idea of deleting normal forms goes back to a note by Arai and Moser [3] and simplifies reasoning significantly.

The set of irreducible terms is denoted by N below, and kept abstract. Only in the final theorem we will instantiate N by the set of normal forms $\mathrm{NF}(\mathcal{R})$ of the considered TRS $\mathcal{R}$. This abstraction allows us to reuse the embedding in a variation of the order that we discuss in Section 14.3.

Definition 9.26 (Predicative Interpretation)**.** Let $N \subseteq \mathcal{T}(\mathcal{F},\mathcal{V})$ denote a set of terms. We define the *predicative interpretation* $\mathcal{I}_N : \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}^\star_{\mathsf{n}}(\mathcal{F},\mathcal{V})$, as follows:

$$\mathcal{I}_N(t) := \begin{cases} [\,] & \text{if } t \in N, \\ [f_{\mathsf{n}}(t_1,\dots,t_k)] +\!\!+ \mathcal{I}(t_{k+1}) +\!\!+ \cdots +\!\!+ \mathcal{I}(t_{k+l}) & \text{if } t \notin N. \end{cases}$$

For the second case we suppose $t = f(t_1,\dots,t_k;t_{k+1},\dots,t_{k+l})$. For a rewrite system $\mathcal{R}$, we set $\mathcal{I}_\mathcal{R} := \mathcal{I}_{\mathrm{NF}(\mathcal{R})}$.

To avoid notational overhead, we introduce the following restriction of the rewrite relation $\to_\mathcal{R}$.

Definition 9.27. Let N be a set of terms, and let $\mathcal{R}$ denote a TRS. We define $s \xrightarrow{N}_\mathcal{R} t$ if there exists a context C, substitution $\sigma : \mathcal{V} \to N$ and rule $l \to r$ such that $s = C[l\sigma]$, $t = C[t\sigma]$ and $l\sigma \notin N$.

Observe that when $\mathcal{R}$ is a constructor TRS, the relation $\xrightarrow{\mathrm{NF}(\mathcal{R})}_\mathcal{R}$ is identical to $\xrightarrow{\mathrm{i}}_\mathcal{R}$. In any predicative recursive constructor TRS $\mathcal{R}$, computation is only permitted on safe arguments. We capture this observation in the set $\mathcal{N}_\mathcal{R}$ of terms, defined as follows. Again $\mathrm{NF}(\mathcal{R})$ is abstracted by a set of terms N.

Definition 9.28. Let $N \in \mathcal{T}(\mathcal{F})$ be a set of ground term. We define $\mathcal{N}_N \subseteq \mathcal{T}(\mathcal{F})$ as the least set such that:

(1) $N \subseteq \mathcal{N}_N$; and

(2) if $f/k+l \in \mathcal{F}$, $s_1,\dots,s_k \in N$ and $s_{k+1},\dots,s_{k+l} \in \mathcal{N}_N$ then

$$f(t_1,\dots,t_k;t_{k+1},\dots,t_{k+l}) \in \mathcal{N}_N\ .$$

For a TRS $\mathcal{R}$, we denote by $\mathcal{N}_\mathcal{R}$ the set of terms $\mathcal{N}_{NF}$ for NF the set of ground normal forms of $\mathcal{R}$.

Call the set $N \subseteq \mathcal{T}(\mathcal{F})$ closed *under constructor contexts*, if whenever $t \in N$ and $C \in \mathcal{T}(\mathcal{C} \cup \{\Box\})$ also $C[t] \in N$ holds. The set N is *closed under sub-terms* if whenever $t \in N$, then $s \in N$ for all sub-term $s \trianglelefteq t$. The next lemma shows that these two closure properties are sufficient conditions for the set $\mathcal{N}_N$ to be closed under reductions with respect to $\xrightarrow{N}_{\mathcal{R}}$. In particular, this implies that $\mathcal{N}_{\mathcal{R}}$ is closed under $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$, since the set of ground normal forms of $\mathcal{R}$ satisfies both closure properties.

Lemma 9.29. *Let $N \subseteq \mathcal{T}(\mathcal{F})$ be a set of ground terms that is closed under constructor contexts and sub-terms. Let $l = f(l_1, \ldots, l_m; l_{m+1}, \ldots, l_{m+n}) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a constructor based term, and suppose $l \succ_{\mathsf{spop}^\star} r$ holds, where the precedence underlying $\succ_{\mathsf{spop}^\star}$ is admissible. Then*

$$s \in \mathcal{N}_N \text{ and } s \xrightarrow{N}_{\{l \to r\}} t \implies t \in \mathcal{N}_N \,.$$

Proof. Consider terms $s \in \mathcal{N}_N$ and $t \in \mathcal{T}(\mathcal{F})$ such that $s = C[l\sigma] \xrightarrow{N}_{\mathcal{R}} C[r\sigma] = t$ holds for a context C and substitution $\sigma : \mathcal{V} \to N$. We prove $t \in \mathcal{N}_N$ by induction on the context C under the assumptions of the lemma.

The proof of the base case $C = \Box$ is by induction on the definition of $\succ_{\mathsf{spop}^\star}$. Consider first the case $l \succ^{\langle 1 \rangle}_{\mathsf{spop}^\star} r$. Then $l_i \succ_{\mathsf{spop}^\star} r$ for some argument position $i \in \{1, \ldots, m+n\}$. Using that $l_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ is a constructor term we have $r \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ by Lemma 9.15, since N is closed under constructor contexts it follows that $r\sigma \in N \subseteq \mathcal{N}_N$ holds. For the remaining cases we can suppose that $r = g(r_1, \ldots, r_{m'}; r_{m'+1}, \ldots, r_{m'+n'})$. Consider first $l \succ^{\langle 2 \rangle}_{\mathsf{spop}^\star} r$. As the definition yields for each normal argument r_j of r $(j = 1, \ldots, m')$ some normal argument l_{i_j} of l $(i_j \in \{1, \ldots, m\})$ with $l_{i_j} \trianglerighteq\!\!\!/\approx r_j$, we conclude $r_j \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ and thus $r_j\sigma \in N$. For the remaining safe arguments r_j $(j = m'+1, \ldots, m'+n')$ induction hypothesis yields $r_j\sigma \in \mathcal{N}_N$, and hence by Definition we get $r\sigma \in \mathcal{N}_N$. Consider finally the case $l \succ^{\langle 3 \rangle}_{\mathsf{spop}^\star} r$. Using that l is constructor based, the order constraints together with Lemma 9.15 and assumptions on N yield $r_j\sigma \in N$ for all arguments r_j of r $(j = 1, \ldots, m'+n')$ as in the first sub-case. By definition we obtain $r_j \in \mathcal{N}_N$ for safe arguments of r, and so $r\sigma \in \mathcal{N}_N$. This concludes the base case.

Consider now the inductive step

$$s = f(s_1, \ldots, s_i, \ldots, s_k) \xrightarrow{N}_{\{l \to r\}} f(s_1, \ldots, t_i, \ldots, s_k) \,,$$

with $s_i \xrightarrow{N}_{\{l \to r\}} t_i$. Since N is closed under sub-terms, and $s_i \xrightarrow{N}_{\{l \to r\}} t_i$ requires $l\sigma \notin N$ for the substitution σ underlying this step, we see that the rewrite position i is a safe argument position of f, hence for $s \in \mathcal{N}_N$ we have $s_i \in \mathcal{N}_N$. Thus by induction hypothesis $t_i \in \mathcal{N}_N$ and so $t \in \mathcal{N}_N$, using again $i \in \mathsf{safe}(f)$.$\Box$

Suppose N is a set that is closed under constructor contexts and sub-terms. We now show that every reduction

$$t_0 \xrightarrow{N}_{\mathcal{R}} t_1 \xrightarrow{N}_{\mathcal{R}} \cdots \xrightarrow{N}_{\mathcal{R}} t_l \,,$$

for $t_0 \in \mathcal{N}_N$ can be embedded into $\succ_{\mathcal{K},\ell}$, i.e.,

$$\mathcal{I}_N(t_0) \succ_{\mathcal{K},\ell} \mathcal{I}_N(t_1) \succ_{\mathcal{K},\ell} \cdots \succ_{\mathcal{K},\ell} \mathcal{I}_N(t_l) \,,$$

holds for ℓ the maximal size of a right-hand side in $\mathcal{R}$. In particular, for the special case $N = \mathrm{NF}(\mathcal{R})$ this establishes the predicative embedding of innermost reductions from basic terms t_0. In the embedding, we use as precedence the projection of the precedence $\gtrsim$ underlying $\mathcal{R}$ to the normalised signature $\mathcal{F}_{\mathsf{n}}$, defined as follows.

Definition 9.30 (Normalised Quasi-precedence)**.** Let $\gtrsim$ be the precedence induced by $\mathcal{R}$. We define the *normalised quasi-precedence* $\sqsupseteq$ on $\mathcal{F}_{\mathsf{n}}$ such that for all $f, g \in \mathcal{F}$,

$$f_{\mathsf{n}} \sqsupseteq g_{\mathsf{n}} \quad :\Longleftrightarrow \quad f \gtrsim g \quad \text{and} \quad f \sqsupseteq g \quad :\Longleftrightarrow \quad f \gtrsim g \,.$$

Let $\sqsubseteq$ be the inverse of $\sqsupseteq$. We denote by $\sqsupset$ the strict order $\sqsupseteq \setminus \sqsubseteq$, and by $\sim$ the equivalence $\sqsupseteq \cap \sqsubseteq$. By $\approx$ we denote the extension of $\sim$ to $\mathcal{T}_{\mathsf{n}}(\mathcal{F}, \mathcal{V})$. Since the precedence $\sqsupseteq$ collapses to $\gtrsim$ if restricted to $\mathcal{F}$, no confusion can arise from this.

As a preparatory step, the next lemma considers root steps only. The full predicative embedding is then provided Lemma 9.32.

Lemma 9.31. *Let $N \subseteq \mathcal{T}(\mathcal{F})$ be a set of ground terms that is closed under constructor contexts and sub-terms, and $s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l}) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a constructor based term. Let $\succ_{\mathsf{spop}^\star}$ denote an instance of a polynomial path order as induced by an admissible precedence $\gtrsim$ and a set of recursive function symbols $\mathcal{K}$. Then for all $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and substitutions $\sigma : \mathcal{V} \to N$,*

$$s \succ_{\mathsf{spop}^\star} t \implies \mathcal{I}_N(s\sigma) \sqsupset_{\mathcal{K}_{\mathsf{n}},|t|} \mathcal{I}_N(t\sigma) \,.$$

Proof. Fix a constructor based term $s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l})$ and substitutions $\sigma : \mathcal{V} \to N$. Abbreviate $\ell := |t|$. We first show that for all terms t,

$$s \succ_{\mathsf{spop}^\star} t \implies f_{\mathsf{n}}(s_1\sigma, \ldots, s_k\sigma) \sqsupset_{\mathcal{K}_{\mathsf{n}},|t|} u \text{ for all } u \in \mathcal{I}_N(t\sigma) \,. \qquad (\dagger)$$

Suppose $s \succ_{\mathsf{spop}^\star} t$ holds, the proof is by induction on $|t|$. The non-trivial case is when $t\sigma \notin N$ as otherwise $\mathcal{I}_N(t\sigma) = [\,]$. This excludes a priori the case $s \succ^{\langle 1 \rangle}_{\mathsf{spop}^\star} t$, because then t is a constructor term by Lemma 9.15, and hence $t\sigma \in N$ assumption on N. Suppose thus $t = g(t_1, \ldots, t_m; t_{m+1}, \ldots, t_{m+n})$ for some $g \in \mathcal{F}$ and terms t_j $(j = 1, \ldots, m+n)$, where either $s \succ^{\langle 2 \rangle}_{\mathsf{spop}^\star} t$ or $s \succ^{\langle 3 \rangle}_{\mathsf{spop}^\star} t$ holds. By definition,

$$\mathcal{I}_N(t\sigma) = [g_{\mathsf{n}}(t_1\sigma, \ldots, t_m\sigma)] \mathbin{+\!\!+} \mathcal{I}_N(t_{m+1}\sigma) \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} \mathcal{I}_N(t_{m+n}\sigma) \,.$$

To prove the implication $(\dagger)$, consider first the element $u = g_{\mathsf{n}}(t_1\sigma, \ldots, t_m\sigma) \in \mathcal{I}_N(t\sigma)$. When $s \succ^{\langle 2 \rangle}_{\mathsf{spop}^\star} t$ holds we can prove

$$f_{\mathsf{n}}(s_1\sigma, \ldots, s_k\sigma) \sqsupset^{\langle 1 \rangle}_{\mathcal{K}_{\mathsf{n}},|t|} g_{\mathsf{n}}(t_1\sigma, \ldots, t_m\sigma) \,,$$

as follows.

- By the order constraint $f \succ g$ we obtain $f_{\mathsf{n}} \sqsupset g_{\mathsf{n}}$.

- Fix a normal argument position $j \in \{1,\dots,m\}$ of g. The assumption $s \succ^{\langle 2\rangle}_{\mathsf{spop}^\star} t$ gives $s \rhd^{\mathsf{n}}\!/\!\approx t_j$. Hence there exists a normal argument position $i \in \{1,\dots,k\}$ of f with $s_i \trianglerighteq\!/\!\approx t_j$, and hence $s_i\sigma \trianglerighteq\!/\!\approx t_j\sigma$ holds. In total,

$$f_{\mathsf{n}}(s_1\sigma,\dots,s_k\sigma) \rhd\!/\!\approx t_j\sigma \qquad \text{for all } j = 1,\dots,m.$$

- Trivially $m \leqslant |t|$.

When $s \succ^{\langle 3\rangle}_{\mathsf{spop}^\star} t$ holds then $m = k$ and we can prove

$$f_{\mathsf{n}}(s_1\sigma,\dots,s_k\sigma) \sqsupseteq^{\langle 2\rangle}_{\mathcal{K}_{\mathsf{n}},|t|} g_{\mathsf{n}}(t_1\sigma,\dots,t_k\sigma)\ ,$$

as follows.

- We have $f_{\mathsf{n}} \sim g_{\mathsf{n}}$ as $f \sim g$, hence also $f_{\mathsf{n}}, g_{\mathsf{n}} \in \mathcal{K}_{\mathsf{n}}$ by admissibility of the precedence.

- The assumption $s \succ^{\langle 3\rangle}_{\mathsf{spop}^\star} t$ gives $\langle s_1,\dots,s_k\rangle \succ_{\mathsf{spop}^\star} \langle t_1,\dots,t_k\rangle$. Using that s is constructor based, we can satisfy the precondition $s_i \in \mathcal{T}(\mathcal{C},\mathcal{V})$ $(i = 1,\dots,k)$ of Lemma 9.15, and see that $\langle s_1,\dots,s_k\rangle \rhd\!/\!\approx \langle t_1,\dots,t_k\rangle$ holds. Hence
$$\langle s_1\sigma,\dots,s_k\sigma\rangle \rhd\!/\!\approx \langle t_1\sigma,\dots,t_k\sigma\rangle\ ,$$
follows.

This concludes the case $u = g_{\mathsf{n}}(t_1\sigma,\dots,t_m\sigma)$.

Now consider the remaining elements $u \in \mathcal{I}_N(t\sigma)$, $u \neq g_{\mathsf{n}}(t_1\sigma,\dots,t_m\sigma)$. Fix $u \in \mathcal{I}_N(t\sigma)$. Then u occurs in the interpretation of a safe argument of $t\sigma$ by definition of the interpretation, say $u \in \mathcal{I}_N(t_j\sigma)$ for some $j \in \{m+1,\dots,m+n\}$. One verifies that $s \succ_{\mathsf{spop}^\star} t_j$ holds: in the case $s \succ^{\langle 2\rangle}_{\mathsf{spop}^\star} t$ we have $s \succ_{\mathsf{spop}^\star} t_j$ by definition; otherwise $s \succ^{\langle 2\rangle}_{\mathsf{spop}^\star} t$ holds and we even obtain $s \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} t_j$. As $|t_j| < |t|$, by induction hypothesis we have $f_{\mathsf{n}}(s_1\sigma,\dots,s_k\sigma) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},|t_j|} u$, and thus $f_{\mathsf{n}}(s_1\sigma,\dots,s_m\sigma) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},|t|} u$ using Lemma 9.20(1). Overall, we conclude the implication (†).

Fix $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ with $s \succ_{\mathsf{spop}^\star} t$. We return to the main proof, and show that $f_{\mathsf{n}}(s_1\sigma,\dots,s_m\sigma) \sqsupseteq^{\langle 3\rangle}_{\mathcal{K}_{\mathsf{n}},|t|} \mathcal{I}_N(t\sigma)$ holds, from which the lemma follows by one application of $\sqsupseteq^{\langle 4\rangle}_{\mathcal{K}_{\mathsf{n}},|t|}$.

- The implication (†) and assumption $s \succ_{\mathsf{spop}^\star} t$ gives $f_{\mathsf{n}}(l_1\sigma,\dots,l_m\sigma) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},|t|} u$ for all elements $u \in \mathcal{I}_N(t\sigma)$.

- There exists at most one occurrence of a function symbol g_{n} with $f_{\mathsf{n}} \sim g_{\mathsf{n}}$ in $\mathcal{I}_N(t\sigma)$:

  By induction on the definition of $\succ_{\mathsf{spop}^\star}$ we first show that t contains at most one occurrence of g with $f \sim g$. If $s \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} t$ then $t \in \mathcal{T}(\mathcal{C},\mathcal{V})$ by Lemma 9.15 and since s is constructor based. As defined symbols are not equivalent to constructors in the admissible precedence $\succsim$, the property follows. If $s \succ^{\langle 2\rangle}_{\mathsf{spop}^\star} t$, the claim follows by induction hypothesis and the final restriction. Finally, suppose $s \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} t$ holds. Then t is constructor

based, again using Lemma 9.15 on the order constraints of arguments. The property follows for this final case.

For a proof by contradiction, suppose now there are at least two distinct occurrences of function symbols equivalent to f_{n} in $\mathcal{I}_N(t\sigma)$, say g_{n} and h_{n}. Unfolding the recursive definition of $\mathcal{I}_N$, we obtain that there exist two distinct occurrences of g and h in $t\sigma$. Since $x\sigma \in N$ and thus $\mathcal{I}_N(x\sigma) = [\,]$, these occurrences even exist in t. Note that by definition of $\sqsupseteq$, the assumption that $f_{\mathsf{n}} \sim g_{\mathsf{n}} \sim h_{\mathsf{n}}$ gives $f \sim g \sim h$. As we have seen above, this however contradicts $s \succ_{\mathsf{spop}^\star} t$.

- The length of $\mathcal{I}_N(t\sigma)$ is bounded by $|t|$. This can be shown by a standard induction on t, using in the base case $x \in \mathcal{V}$ that $\mathcal{I}_N(x\sigma) = [\,]$ by the assumption on σ. □

Lemma 9.32. *Let $N \subseteq \mathcal{T}(\mathcal{F})$ be a set of ground terms that is closed under constructor contexts and sub-terms. Let $l = f(l_1, \dots, l_m; l_{m+1}, \dots, l_{m+n}) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a constructor based term, and suppose $l \succ_{\mathsf{spop}^\star} r$ holds, where the precedence underlying $\succ_{\mathsf{spop}^\star}$ is admissible. Then*

$$s \in \mathcal{N}_N \text{ and } s \xrightarrow{N}_{\{l \to r\}} t \implies \mathcal{I}_N(s) \sqsupset_{\mathcal{K}_{\mathsf{n}}, |r|} \mathcal{I}_N(t) \,.$$

Proof. Let $s \in \mathcal{N}_N$ and consider a rewrite step $s \xrightarrow{N}_{\{l \to r\}} t$. The base case is covered by Lemma 9.31, hence consider a rewrite step below the root. Since $s \in \mathcal{N}_N$, in Lemma 9.29 we already observed that this step is of the form

$$s = f(s_1, \dots, s_m; s_{m+1}, \dots, s_i, \dots, s_{m+n}) \xrightarrow{N}_{\{l \to r\}} f(s_1, \dots, s_m; s_{m+1}, \dots, t_i, \dots, s_{m+n}) = t \,,$$

where $s_i \xrightarrow{N}_{\{l \to r\}} t_i$. The non-trivial case is when $t \notin N$, otherwise $\mathcal{I}_N(t) = [\,]$. Using the induction hypothesis $\mathcal{I}_N(s_i) \sqsupset_{\mathcal{K}_{\mathsf{n}}, \ell} \mathcal{I}_N(t_i)$ and Lemma 9.20(3) we obtain

$$\begin{aligned} \mathcal{I}_N(s) &= f_{\mathsf{n}}(s_1, \dots, s_m) +\!\!+ \mathcal{I}_N(s_{m+1}) +\!\!+ \cdots +\!\!+ \mathcal{I}_N(s_i) +\!\!+ \cdots +\!\!+ \mathcal{I}_N(s_{m+n}) \\ &\succ_{\mathcal{K}, \ell} f_{\mathsf{n}}(s_1, \dots, s_m) +\!\!+ \mathcal{I}_N(s_{m+1}) +\!\!+ \cdots +\!\!+ \mathcal{I}_N(t_i) +\!\!+ \cdots +\!\!+ \mathcal{I}_N(s_{m+n}) \\ &= \mathcal{I}_N(t) \,, \end{aligned}$$

as desired. □

9.1.3. Putting Things Together

We finally arrive at the correctness proof of sPOP*.

Proof of Theorem 9.8. Let $\mathcal{R}$ denote a predicative recursive constructor TRS over the signature $\mathcal{F}$. We have to prove that for every $f \in \mathcal{F}$ there exists a polynomial p_f of degree $\mathsf{rd}_{\mathcal{R}}(f)$ such that the innermost derivation height of $f(\vec{u}; \vec{v})$ for values $\vec{u}, \vec{v}$ is bounded by a $p_f(n)$, where n refers to the sum of the depths of normal arguments $\vec{u}$.

Define $\ell := \max\{|r| \mid l \to r \in \mathcal{R}\}$. Consider a derivation

$$f(\vec{u}; \vec{v}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}} \cdots \xrightarrow{\mathrm{i}}_{\mathcal{R}} t_m \,,$$

which can be written as

$$f(\vec{u};\vec{v}) \xrightarrow{\mathrm{NF}(\mathcal{R})}_{\mathcal{R}} t_1 \xrightarrow{\mathrm{NF}(\mathcal{R})}_{\mathcal{R}} \cdots \xrightarrow{\mathrm{NF}(\mathcal{R})}_{\mathcal{R}} t_m \ .$$

Observe $f(\vec{u};\vec{v}) \in \mathcal{N}_{\mathcal{R}}$, hence by Lemma 9.29 it follows that $t_i \in \mathcal{N}_{\mathcal{R}}$ for all $i = 1, \ldots, m$. As a consequence of Lemma 9.32, using Lemma 9.20(1), we obtain

$$\mathcal{I}_{\mathcal{R}}(f(\vec{u};\vec{v})) = [f_{\mathsf{n}}(\vec{u})] \sqsupset_{\mathcal{K}_{\mathsf{n}},\ell} \mathcal{I}_{\mathcal{R}}(t_1) \cdots \sqsupset_{\mathcal{K}_{\mathsf{n}},\ell} \mathcal{I}_{\mathcal{R}}(t_m) \ .$$

So in particular the length m is bounded by the length of $\sqsupset_{\mathcal{K}_{\mathsf{n}},\ell}$ descending sequences starting from $[f_{\mathsf{n}}(\vec{u})]$, i.e., $m \leqslant \mathsf{G}_{\mathcal{K},\ell}([f_{\mathsf{n}}(\vec{u})]) = \mathsf{G}_{\mathcal{K},\ell}(f_{\mathsf{n}}(\vec{u}))$. Here the equality is given by Lemma 9.23. A standard induction proves that $\mathsf{rd}_{\mathcal{R}}(f) = \mathsf{rd}_{\mathcal{K},\gtrsim}(f) = \mathsf{rd}_{\mathcal{K}_{\mathsf{n}},\sqsupset}(f)$, and thus for p_f we can choose the polynomial as provided in Theorem 9.24. □

As a consequence of Theorem 9.8 and the polynomial invariance theorem, or more precise its Corollary 7.3, we obtain the following result.

Corollary 9.33 (Soundness)**.** *Let $\mathcal{R}$ be a confluent (or orthogonal) and predicative recursive constructor TRS. Then every function defined by $\mathcal{R}$ is computable in polynomial time, on a deterministic Turing machine.*

Note that this result relies on sharing as discussed in the first part of this thesis, the corollary does not hold in general using an explicit encoding of terms.

9.2. Small Polynomial Path Orders are Complete

As a final step to our *order-theoretic characterisation* of **FP** we prove that any function $f \in \mathbf{FP}$ is expressible as a confluent, even orthogonal, predicative recursive constructor TRS $\mathcal{R}_f$. Towards this goal, we employ a recursion theoretic characterisation $\mathcal{B}_{\mathcal{W}}$ of **FP**.

Definition 9.34. The class $\mathcal{B}_{\mathcal{W}}$ is inductively defined as follows. For $k, l \in \mathbb{N}$, we denote by $\mathcal{B}^{k,l}_{\mathcal{W}}$ the sub-class with k normal and l safe arguments. Let $\vec{x} := x_1, \ldots, x_k$, $\vec{y} := y_1, \ldots, y_l$ denote pairwise distinct variables, and let k, l range over $\mathbb{N}$. The class $\mathcal{B}_{\mathcal{W}}$ is defined as the least class of functions over binary words $\mathbb{W}(\mathbb{B})$ that:

(1) $\mathcal{B}_{\mathcal{W}}$ contains the *initial functions* S_0, S_1, P, $I^{k,l}_j$ for $k, l \in \mathbb{N}$ and all $j \in \{1, \ldots, k+l\}$, C and O. These functions are defined by the following equations.

$$\begin{aligned}
S_i(;x) &= xi && \text{(for } i = 0, 1\text{)}\\
P(;\epsilon) &= \epsilon\\
P(;xi) &= x && \text{(for } i = 0, 1\text{)}\\
I^{k,l}_j(\vec{x};\vec{y}) &= x_j && \text{(for all } j = 1, \ldots, k\text{)}\\
I^{k,l}_j(\vec{x};\vec{y}) &= y_{j-k} && \text{(for all } j = k+1, \ldots, l+k\text{)}\\
C(;\epsilon, y, z_1, z_2) &= y\\
C(;xi, y, z_1, z_2) &= z_i && \text{(for i = 0, 1)}\\
O(\vec{x};\vec{y}) &= \epsilon && .
\end{aligned}$$

(2) $\mathcal{B}_{\mathcal{W}}$ is closed under *weak safe composition* (**WSC**), that is, $f \in \mathcal{B}_{\mathcal{W}}$ where

$$f(\vec{x};\vec{y}) = g(x_{i_1},\ldots,x_{i_m};h_1(\vec{x};\vec{y}),\ldots,h_n(\vec{x};\vec{y}))\ ,$$

for previously defined functions $g \in \mathcal{B}_{\mathcal{W}}^{m,n}$ and $h_1,\ldots,h_n \in \mathcal{B}_{\mathcal{W}}^{k,l}$;

(3) $\mathcal{B}_{\mathcal{W}}$ is closed under *safe recursion on notation* (**SRN**), that is, $f \in \mathcal{B}_{\mathcal{W}}$ where

$$\begin{aligned} f(\varepsilon,\vec{x};\vec{y}) &= g(\vec{x};\vec{y}) \\ f(zi,\vec{x};\vec{y}) &= h_i(z,\vec{x};\vec{y},f(z,\vec{x};\vec{y})) \qquad (\text{for } i=0,1) \end{aligned}$$

for previously defined functions $g \in \mathcal{B}_{\mathcal{W}}^{k,l}$ and $h_0,h_1 \in \mathcal{B}_{\mathcal{W}}^{k+1,l+1}$.

Remark. This class $\mathcal{B}_{\mathcal{W}}$ is obtained by restricting the composition scheme underlying Bellantoni and Cook's class $\mathcal{B}$ to weak safe composition. Unlike here, Bellantoni and Cook allow the definition of a function f by *safe composition* given by the equation

$$f(\vec{x};\vec{y}) = g(h_1(\vec{x};\vec{y}),\ldots,h_m(\vec{x};\vec{y});h_{m+1}(\vec{x};\vec{y}),\ldots,h_{m+n}(\vec{x};\vec{y}))\ , \qquad (\mathbf{SC})$$

for functions g, and $h_1,\ldots,h_{m+n}$ already defined in $\mathcal{B}$.

It thus follows that our class $\mathcal{B}_{\mathcal{W}}$ is included in $\mathcal{B}$ and thus defines only polytime computable functions. Concerning the converse inclusion, the following theorem states that the class $\mathcal{B}_{\mathcal{W}}$ is large enough to capture *all* the polytime computable functions [21].

Proposition 9.35 (Eguchi [9])**.** *Every polynomial time computable function belongs to* $\mathcal{B}_{\mathcal{W}}^{k,0}$ *for some* $k \in \mathbb{N}$.

One can show this fact by following the proof of Theorem 3.7 in [37], where the unary variant of $\mathcal{B}_{\mathcal{W}}$ is defined and the inclusion corresponding to Proposition 9.35 is shown. We refrain from duplicating this technical proof here.

We use this proposition for the completeness result of polynomial path orders. As an intermediate step, following the program of Cichon and Weiermann [25], we extract from the class $\mathcal{B}_{\mathcal{W}}$ a *term rewriting characterisation* of the polytime computable functions. Beckmann and Weiermann [20] already gave a term rewriting characterisation $\mathcal{R}_{\mathcal{B}}$ of Bellantoni and Cook's class $\mathcal{B}$. We present here a modification of $\mathcal{R}_{\mathcal{B}}$ that accounts for our modified composition scheme.

Definition 9.36 (Term Rewriting Characterisation $\mathcal{R}_{\mathcal{BW}}$)**.** For each $k,l \in \mathbb{N}$ the set of function symbols $\mathcal{F}_{\mathcal{BW}}^{k,l}$ with k normal and l safe argument positions is the least set of function symbols such that:

(1) $\epsilon \in \mathcal{F}_{\mathcal{BW}}^{0,0}$, $\mathsf{s}_0,\mathsf{s}_1 \in \mathcal{F}_{\mathcal{BW}}^{0,1}$ are the only constructors; and

(2) $\mathsf{S}_0,\mathsf{S}_1 \in \mathcal{F}_{\mathcal{BW}}^{0,1}$, $\mathsf{P} \in \mathcal{F}_{\mathcal{BW}}^{0,1}$, $\mathsf{C} \in \mathcal{F}_{\mathcal{BW}}^{0,4}$, $\mathsf{O}^{k,l} \in \mathcal{F}_{\mathcal{BW}}^{k,l}$ and for all $j=1,\ldots,k+l$, $\mathsf{I}_j^{k,l} \in \mathcal{F}_{\mathcal{BW}}^{k,l}$; and

(3) if $\mathsf{g} \in \mathcal{F}^{m,n}_{\mathcal{BW}}$, $\vec{\mathsf{h}} := \mathsf{h}_1, \dots, \mathsf{h}_n \in \mathcal{F}^{k,l}_{\mathcal{BW}}$ and $1 \leqslant i_1 \leqslant \cdots \leqslant i_m \leqslant k$ then $\mathsf{WSC}[\mathsf{g}, i_1, \dots, i_m, \vec{\mathsf{h}}] \in \mathcal{F}^{k,l}_{\mathcal{BW}}$; and

(4) if $\mathsf{g} \in \mathcal{F}^{k,l}_{\mathcal{BW}}$ and $\mathsf{h}_0, \mathsf{h}_1 \in \mathcal{F}^{k+1,l+1}_{\mathcal{BW}}$ then $\mathsf{SRN}[\mathsf{g}, \mathsf{h}_0, \mathsf{h}_1] \in \mathcal{F}^{k+1,l}_{\mathcal{BW}}$.

The *predicative signature* is given by $\mathcal{F}_{\mathcal{BW}} := \bigcup_{k,l \in \mathbb{N}} \mathcal{F}^{k,l}_{\mathcal{BW}}$.

Let $\vec{x} := x_1, \dots, x_k$, $\vec{y} := y_1, \dots, y_l$ denote pairwise distinct variables and k, l range over $\mathbb{N}$. The *schema of rewrite rules* $\mathcal{R}_{\mathcal{BW}}$ is defined as the least set of rules such that:

(1) $\mathcal{R}_{\mathcal{BW}}$ contains the following rules defining initial functions:

$$
\begin{aligned}
\mathsf{S}_i(;x) &\to \mathsf{s}_i(;x) && \text{(for } i = 0, 1\text{)}\\
\mathsf{P}(;\epsilon) &\to \epsilon\\
\mathsf{P}(;xi) &\to x && \text{(for } i = 0, 1\text{)}\\
\mathsf{I}^{k,l}_j(\vec{x};\vec{y}) &\to x_j && \text{(for all } j = 1, \dots, k\text{)}\\
\mathsf{I}^{k,l}_j(\vec{x};\vec{y}) &\to y_{j-k} && \text{(for all } j = k+1, \dots, l+k\text{)}\\
\mathsf{C}(;\epsilon, y, z_1, z_2) &\to y\\
\mathsf{C}(;xi, y, z_1, z_2) &\to z_i && \text{(for i = 0, 1)}\\
\mathsf{O}(\vec{x};\vec{y}) &\to \epsilon && .
\end{aligned}
$$

(2) For $\mathsf{WSC}[\mathsf{g}, i_1, \dots, i_m, \vec{\mathsf{h}}] \in \mathcal{F}^{k,l}_{\mathcal{BW}}$ the schema $\mathcal{R}_{\mathcal{BW}}$ contains the rule

$$\mathsf{WSC}[\mathsf{g}, i_1, \dots, i_m, \vec{\mathsf{h}}](\vec{x};\vec{y}) \to \mathsf{g}(x_{i_1}, \dots, x_{i_m}; \mathsf{h_1}(\vec{x};\vec{y}), \dots, \mathsf{h_n}(\vec{x};\vec{y}))\ ,$$

(3) For $\mathsf{SRN}[\mathsf{g}, \mathsf{h}_0, \mathsf{h}_1] \in \mathcal{F}^{k+1,l}_{\mathcal{BW}}$ the schema $\mathcal{R}_{\mathcal{BW}}$ contains the rules

$$
\begin{aligned}
\mathsf{SRN}[\mathsf{g}, \mathsf{h}_0, \mathsf{h}_1](\epsilon, \vec{x};\vec{y}) &\to \mathsf{g}(\vec{x};\vec{y})\\
\mathsf{SRN}[\mathsf{g}, \mathsf{h}_0, \mathsf{h}_1](zi, \vec{x};\vec{y}) &\to \mathsf{h}_i(z, \vec{x};\vec{y}, \mathsf{SRN}[\mathsf{g}, \mathsf{h}_0, \mathsf{h}_1](z, \vec{x};\vec{y})) \text{ (for } i = 0, 1\text{).}
\end{aligned}
$$

Remark. We emphasise that the schema $\mathcal{R}_{\mathcal{B}}$ is dupped *infeasible* in [20]. Indeed $\mathcal{R}_{\mathcal{B}}$ admits an exponential lower bound on the derivation height if one considers full rewriting. This holds due to the possible duplication of redexes as illustrated in Example 9.12. The same observation carries over to our schema $\mathcal{R}_{\mathcal{BW}}$. However, this should be understood as a miss-configuration of the evaluation strategy, rather than a defect of the rewrite system. Indeed, in our completeness argument below, we exploit that $\mathcal{R}_{\mathcal{BW}}$ is predicative recursive. Thus the *innermost* runtime complexity is polynomial, as expected.

Lemma 9.37. *For each function $f \in \mathcal{B}_{\mathcal{W}}$, there exists a finite restriction $\mathcal{R}_f \subsetneq \mathcal{R}_{\mathcal{BW}}$ such that $\mathcal{R}_f$ computes f.*

Proof. The rewrite schema $\mathcal{R}_{\mathcal{BW}}$ is obtained by introducing for each defining equation in $\mathcal{B}_{\mathcal{W}}$ a separate rewrite rule. Hence for $f \in \mathcal{B}_{\mathcal{W}}$, we can take as $\mathcal{R}_f \subsetneq \mathcal{R}_{\mathcal{BW}}$ the set of rewrite rules that correspond to the equations involved in the definition of f. This set of rules is finite by the inductive construction of $\mathcal{B}_{\mathcal{W}}$. A standard induction gives that $\mathcal{R}_f$ computes the function f. □

Theorem 9.38 (Completeness)**.** *For every $f \in \mathcal{B}_{\mathcal{W}}$ there exists an orthogonal, and predicative recursive constructor TRS $\mathcal{R}_f$ of degree d. Here d equals the maximal number of nested applications of **SRN** in the definition of f.*

Proof. Consider $f \in \mathcal{B}_{\mathcal{W}}$. Take the finite restriction $\mathcal{R}_f \subsetneq \mathcal{R}_{\mathcal{BW}}$ from Lemma 9.37 that computes f. Obviously $\mathcal{R}_f$ is an orthogonal constructor TRS. Let $\succ_{\mathsf{spop}^\star}$ denote the small polynomial path order, as induced by the (strict) precedence $\succsim$ underlying $\mathcal{R}_f$, and the safe mapping as indicated in the rules. By induction on the definition of f, we show (i) $\mathcal{R}_f \subseteq \succ_{\mathsf{spop}^\star}$, and (ii) that the recursion depth d of the maximal symbol in $\mathcal{R}_f$ corresponds to the maximal number of nested applications of **SRN** in the definition of f.

The base case that f is an initial function of $\mathcal{B}_{\mathcal{W}}$ is trivial, we consider the inductive step where we distinguish two cases. First suppose f is defined by weak safe composition based on g and $\vec{h}$, i.e.,

$$f(\vec{x};\vec{y}) = g(x_{i_1},\dots,x_{i_m};h_1(\vec{x};\vec{y}),\dots,h_n(\vec{x};\vec{y}))\ .$$

Abbreviate $\mathsf{f} := \mathsf{WSC}[\mathsf{g},i_1,\dots,i_m,\mathsf{h}_1,\dots,\mathsf{h}_n]$ and consider the rewrite rule

$$\mathsf{f}(\vec{x};\vec{y}) \to \mathsf{g}(x_{i_1},\dots,x_{i_m};\mathsf{h}_1(\vec{x};\vec{y}),\dots,\mathsf{h}_n(\vec{x};\vec{y})) \in \mathcal{R}_f\ .$$

As $\mathsf{f} \succ \mathsf{g}$, and $\mathsf{f} \succ \mathsf{h}_j$ for all $j = 1,\dots,n$, with $n+1$ applications of $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$ and using $\mathsf{f}(\vec{x};\vec{y}) \succ^{\langle 1\rangle}_{\mathsf{spop}^\star} y_i$ ($y_i \in \vec{y}$), the above rule is oriented by $\succ_{\mathsf{spop}^\star}$. By induction hypothesis, the remaining rewrite rules in $\mathcal{R}_f$ that define g and $\vec{\mathsf{h}}$ are compatible with $\succ_{\mathsf{spop}^\star}$, hence $\mathcal{R}_f \subseteq \succ_{\mathsf{spop}^\star}$ holds for this case. Note that since f is not recursive, $\mathsf{rd}_{\mathcal{R}_f}(\mathsf{f})$ equals the maximal recursion depth of $\mathsf{g},\mathsf{h}_1,\dots,\mathsf{h}_n$, and thus (ii) follows directly from induction hypothesis.

Finally consider the case when f is defined by safe recursion on notation with g, h_0, and h_1, i.e.,

$$\begin{aligned} f(\varepsilon,\vec{x};\vec{y}) &= g(\vec{x};\vec{y}) \\ f(zi,\vec{x};\vec{y}) &= h_i(z,\vec{x};\vec{y},f(z,\vec{x};\vec{y})) \qquad (\text{for } i = 0,1). \end{aligned}$$

Abbreviate $\mathsf{f} := \mathsf{SRN}[\mathsf{g},\mathsf{h}_0,\mathsf{h}_1]$, and consider the rewrite rules

$$\begin{aligned} \mathsf{f}(\epsilon,\vec{x};\vec{y}) &\to \mathsf{g}(\vec{x};\vec{y}) \\ \mathsf{f}(\mathsf{s}_i(;z),\vec{x};\vec{y}) &\to \mathsf{h}_i(z,\vec{x};\vec{y},\mathsf{f}(z,\vec{x};\vec{y})) \qquad (\text{for } i = 0,1), \end{aligned}$$

defining f. Using $\mathsf{f} \succ \mathsf{g}$ the first rule can be oriented by $\succ_{\mathsf{spop}^\star}$. For the remaining rules, observe $\langle \mathsf{s}_i(;z),\vec{x}\rangle \succ_{\mathsf{spop}^\star} \langle z,\vec{x}\rangle$ and trivially $\langle\vec{y}\rangle \succsim_{\mathsf{spop}^\star} \langle\vec{y}\rangle$ hold. Hence

$$\mathsf{f}(\mathsf{s}_i(;z),\vec{x};\vec{y}) \succ^{\langle 3\rangle}_{\mathsf{spop}^\star} \mathsf{f}(z,\vec{x};\vec{y})\ ,$$

and orientation of the remaining two recursion rules follows by one application of $\succ^{\langle 2\rangle}_{\mathsf{spop}^\star}$. Hence $\mathcal{R}_f$ is predicative recursive, using the induction hypothesis on g, h_0 and h_1. For (ii), observe that

$$\mathsf{rd}_{\mathcal{R}_f}(\mathsf{f}) = \max\{\mathsf{rd}_{\mathcal{R}_f}(\mathsf{h}) \mid \mathsf{f} \succ \mathsf{h}\} + 1 = \max\{\mathsf{rd}_{\mathcal{R}_f}(\mathsf{g}),\mathsf{rd}_{\mathcal{R}_f}(\mathsf{h}_0),\mathsf{rd}_{\mathcal{R}_f}(\mathsf{h}_1)\} + 1\ ,$$

where the first equality follows by definition of the depth of recursion, and the second equality follows by construction of $\mathcal{R}_f$. Using induction hypothesis on g, h_0 and h_1 we conclude (ii). □

Corollary 9.39. *The following class of functions are equivalent:*

(1) *The class of functions computed by confluent (or orthogonal), and predicative recursive constructor TRS.*

(2) *The class of functions computable in polynomial time on a deterministic Turing machine.*

Proof. The correspondence holds by Corollary 9.33, Proposition 9.35 and Theorem 9.38. □

9.3. Parameter Substitution

Bellantoni already observed that the class $\mathcal{B}$ defined in [21] is closed under safe recursion on notation with *parameter substitution*. Here a function f is defined from functions g, h_0, h_1 and $\vec{p}$ by

$$\begin{array}{ll} f(\epsilon, \vec{x}; \vec{y}) & = g(\vec{x}; \vec{y}) \\ f(zi, \vec{x}; \vec{y}) & = h_i(z, \vec{x}; \vec{y}, f(z, \vec{x}; \vec{p}(z, \vec{x}; \vec{y}))) \qquad (i = 0, 1)\ . \end{array} \tag{$\mathbf{SRN_{PS}}$}$$

We introduce *small polynomial path order with parameter substitution* ($\mathrm{sPOP}^\star_{\mathrm{PS}}$ for short), that extends clause $\succ^{\langle 3\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$ to account for the schema ($\mathbf{SRN_{PS}}$).

Definition 9.40. Let $\gtrsim$ denote a precedence on $\mathcal{F}$, with underlying proper order $\succ$ and equivalence $\sim$. Let $\mathcal{K} \subseteq \mathcal{D}$ denote a set of recursive function symbols, and fix a safe mapping. Then $s \succ_{\mathsf{spop}^*_{\mathsf{ps}}} t$ for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l})$ if one of the following alternatives hold.

(1) $s_i \gtrsim_{\mathsf{spop}^\star} t$ for some argument s_i of s $(i \in \{1, \ldots, k+l\})$; or

(2) $f \in \mathcal{D}$, $t = g(t_1, \ldots, t_m; t_{m+1}, \ldots, t_{m+n})$ where $f \succ g$ and the following holds:

- $s \rhd\!\!\!\!/_{\approx} t_j$ for all normal arguments $t_1, \ldots, t_m$; and
- $s \succ_{\mathsf{spop}^*_{\mathsf{ps}}} t_j$ for all safe arguments $t_{m+1}, \ldots, t_{m+n}$; and
- t contains at most one function symbols g with $f \sim g$; or

(3) $f, g \in \mathcal{K}$, $t = g(t_1, \ldots, t_k; t_{k+1}, \ldots, t_{k+l})$ where $f \sim g$ and the following holds:

- $\langle s_1, \ldots, s_k\rangle \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \langle t_1, \ldots, t_k\rangle$; and
- $s \succ_{\mathsf{spop}^*_{\mathsf{ps}}} t_j$ for all safe arguments t_j;
- the safe arguments t_j $(j = k+1, \ldots, k+l)$ contain no symbols g with $f \sim g$.

Here $s \gtrsim_{\mathsf{spop}^\star} t$ denotes that either $s \approx_{\mathsf{s}} t$ or $s \succ_{\mathsf{spop}^*_{\mathsf{ps}}} t$ holds. In the last clause we use $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ also for the extension of $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ to products.

We adapt the notion of predicative recursive TRS of degree d to $\mathrm{sPOP}^\star_{\mathrm{PS}}$ in the obvious way. Parameter substitution strictly extends the analytic power of $\mathrm{sPOP}^\star$. In particular, $\mathrm{sPOP}^\star$ can handle tail-recursion as in the following example.

Example 9.41. The TRS $\mathcal{R}_{\mathsf{rev}}$ consisting of the three rules

$$\begin{aligned} &18\colon\ \mathsf{rev}'([\,];ys) \to ys && 19\colon\ \mathsf{rev}'(x :: xs;ys) \to \mathsf{rev}'(xs;x :: ys)\\ &20\colon\ \mathsf{rev}(xs;) \to \mathsf{rev}'(xs;[\,])\ , \end{aligned}$$

reverses lists formed from the constructors [] and (::). Then $\mathcal{R}_{\mathsf{rev}}$ is compatible with $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$, but due to the last rule not with $\succ_{\mathsf{spop}^\star}$. ◁

As a consequence of Theorem 9.43 below, the innermost runtime of $\mathcal{R}_{\mathsf{rev}}$ is inferred to be linear. The next lemma establishes that the predicative embedding remains intact under the modified definition.

Lemma 9.42. *Let $N \subseteq \mathcal{T}(\mathcal{F})$ be a set of ground terms that is closed under constructor contexts and sub-terms. Let $l = f(l_1,\dots,l_m;l_{m+1},\dots,l_{m+n}) \in \mathcal{T}(\mathcal{F},\mathcal{V})$ be a constructor based term, and suppose $l \succ_{\mathsf{spop}^\star} r$ holds, where the precedence underlying $\succ_{\mathsf{spop}^\star}$ is admissible. Suppose $s \in \mathcal{N}_N$ and $s \xrightarrow{N}_{\{l\to r\}} t$ holds. Then*

(1) *$t \in \mathcal{N}_N$; and*

(2) *$\mathcal{I}_N(s) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},\ell} \mathcal{I}_N(t)$.*

Proof. The first property follows exactly as in Lemma 9.29. Note that here only order constraints on normal arguments are required, and these constraints coincide in $\succ_{\mathsf{spop}^\star}$ and $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$.

For the second property, consider a substitution $\sigma : \mathcal{V} \to N$. Following the pattern of the proof of Lemma 9.31, one establishes

$$l \succ_{\mathsf{spop}^*_{\mathsf{ps}}} r \implies f_{\mathsf{n}}(l_1\sigma,\dots,l_m\sigma) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},|r|} u \text{ for all } u \in \mathcal{I}_N(r\sigma)\ , \qquad (\ddagger)$$

by induction on $|r|$. Again the only interesting case is when $r\sigma \notin N$. Then $t = g(r_1,\dots,r_{m'};r_{m'+1},\dots,r_{m'+n'})$ for some $g \in \mathcal{F}$ and terms r_j $(j = 1,\dots,m'+n')$, and

$$\mathcal{I}_N(r\sigma) = [g_{\mathsf{n}}(r_1\sigma,\dots,r_{m'}\sigma)] +\!\!+ \mathcal{I}_N(r_{m'+1}\sigma) +\!\!+ \cdots +\!\!+ \mathcal{I}_N(r_{m'+n'}\sigma)\ .$$

Exactly as in Lemma 9.31 one obtains $f_{\mathsf{n}}(l_1\sigma,\dots,l_m\sigma) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},|r|} g_{\mathsf{n}}(r_1\sigma,\dots,r_{m'}\sigma)$, as only order constraints on normal arguments in r are required here. Hence consider $u \in \mathcal{I}_N(t\sigma)$, $u \neq g_{\mathsf{n}}(r_1\sigma,\dots,r_{m'}\sigma)$. In the new case $l \succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}} r$, we have $l \succ_{\mathsf{spop}^\star} r_j$ for all safe arguments r_j of r. In contrast to the proof of Lemma 9.31, we can establish $(\ddagger)$ here directly from the induction hypothesis. Using this preparatory step, we obtain $f_{\mathsf{n}}(l_1\sigma,\dots,r_m\sigma) \sqsupseteq^{\langle 3\rangle}_{\mathcal{K}_{\mathsf{n}},|t|} \mathcal{I}_N(r\sigma)$. Crucially here, we still have that $l \succ_{\mathsf{spop}^*_{\mathsf{ps}}} r$ implies that there exists at most one function symbol g with $g \sim f$ in r, due to the extra side conditions imposed on $\succ^{\langle 3\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$.

We obtain $\mathcal{I}_N(l\sigma) \sqsupseteq^{\langle 4\rangle}_{\mathcal{K}_{\mathsf{n}},|t|} \mathcal{I}_N(l\sigma)$ as desired. Closure under contexts of this embedding then follows exactly as in Lemma 9.32. This concludes the second assertion. □

Theorem 9.43. *Suppose $\mathcal{R}$ is a predicative recursive constructor TRS of degree d (with respect to Definition 9.40). Then the innermost derivation height of any basic term $f(\vec{u};\vec{v})$ is bounded by a polynomial of degree $\mathsf{rd}_{\mathcal{R}}(f)$ in the sum of the depths of normal arguments $\vec{u}$. In particular, the innermost runtime function $\mathrm{rci}_{\mathcal{R}}$ is bounded by a polynomial of degree d.*

Proof. The proof of the theorem follows by reasoning exactly as in Theorem 9.8, replacing the application of Lemma 9.29 and Lemma 9.32 by Lemma 9.42(1) and Lemma 9.42(2) respectively. □

Using Corollary 7.3 we obtain soundness of the order for the class of polytime computable functions.

Corollary 9.44 (Soundness)**.** *Let $\mathcal{R}$ be a confluent (or orthogonal) and predicative recursive constructor TRS. Then every function defined by $\mathcal{R}$ is computable in polynomial time, on a deterministic Turing machine.*

By Theorem 9.38, predicative recursive TRSs are complete for the polytime computable functions. This holds also for the extended definition of small polynomial path orders given in Definition 9.40. Using the above soundness theorem, we thus obtain following characterisation.

Corollary 9.45. *The following class of functions are equivalent:*

(1) *The class of functions computed by confluent (or orthogonal), and predicative recursive constructor TRS, with respect to Definition 9.40.*

(2) *The class of functions computable in polynomial time on a deterministic Turing machine.*

9.4. A Tight Characterisation

The adequacy theorem in conjunction Theorem 9.43 yields that the function f computed by an orthogonal predicative recursive constructor TRS of degree d is computable in time $\mathcal{O}(\log(n)\cdot n^{4\cdot d})$ on a Turing machine. This often overestimates the intrinsic complexity of f considerable.

In this section we show that for a syntactically restricted class of predicative recursive TRSs $\mathcal{R}$, if the degree of recursion is d, then the functions computed by $\mathcal{R}$ can be computed in time $\mathcal{O}(n^d)$ on register machines. Our primary motivation for using register machines here is that the built-in copying instruction is charged unit cost. Vice versa, we show that every function f computable in time $\mathcal{O}(n^d)$ on a register machine is computable by such a predicative recursive TRS $\mathcal{R}$ of degree d.

In [10] we have already shown that ML-like predicative recursive TRS $\mathcal{R}$ of degree d (with respect to $>_{\mathsf{pop}*}$) can be computed in time $p(n) \in \mathcal{O}(n^d)$ on RMs, provided that constructors are *monadic* and no parameter substitution is used. Here a constructor is *monadic* if its arity is at most one. The latter restriction is used to avoid encoding overhead, as values $\mathcal{T}(\mathcal{C})$ can be directly stored in registers as strings. In principle it is not difficult to show that predicative

recursive TRSs of degree d can also simulate computations of RMs that operate in time $p(n) \in \mathcal{O}(n^d)$. The corresponding predicative recursive TRS $\mathcal{R}_f$ can be constructed as follows. A constructor c is used to encode configurations of M as terms. It is straight forward to formulate the one-step transition relation $\to_M$ by a predicative recursive TRS $\mathcal{R}_0^M$ of degree 0, in the sense that

$$\mathsf{M}(;\mathsf{c}(i, w_1, \ldots, w_n)) \to_M \mathsf{c}(i', w_1', \ldots, w_n') \;,$$

holds if and only if

$$(i, w_1, \ldots, w_n) \to_M (i', w_1', \ldots, w_n') \;,$$

holds. See Theorem 9.53 for such a construction. The TRS $\mathcal{R}_f$ is obtained by extending $\mathcal{R}_0^M$ with d recursive definitions that iterate $p(n)$ times the one-step transition relation of M. Hence this predicative recursive TRS $\mathcal{R}_f$ of degree d computes the final configuration of M on any input $\vec{u}$, from which the result of $f(\vec{u})$ can be extracted.

Although conceptually similar, the actual construction carried out below is slightly more involved. To get rid of the auxiliary non-monadic constructor c used for building configurations, we store the components of the configuration directly as safe arguments in the iterator. In order to modify these safe arguments, we require parameter substitution. To retain our tight soundness result in this setting, we restrict the class of considered rewrite systems to *tail-recursive* TRSs. The latter restriction avoids the need of a stack, or similar construction, in the implementation on register machines.

Definition 9.46 (Tail-Recursive)**.** We say that a TRS $\mathcal{R}$ is *tail-recursive*, if for every rule $f(\vec{u};\vec{v}) \to r \in \mathcal{R}$, if g with $g \sim f$ occurs in r then g occurs at the root position in r. The TRS $\mathcal{R}$ is *predicative tail-recursive* (of degree d), if it is tail-recursive and predicative recursive (of degree d), with respect to Definition 9.40.

For instance, the TRS $\mathcal{R}_{\mathsf{rev}}$ from Example 9.41 is a predicative tail-recursive TRS. The main result of this section states.

Theorem 9.47. *For each $d \in \mathbb{N}$, the following class of functions are equivalent:*

(1) *The class of functions computed by* ML*-like predicative tail-recursive TRSs of degree d, using only monadic constructors.*

(2) *The class of functions computable on register machines operating in time $p(n) \in \mathcal{O}(n^d)$.*

Proof. This theorem is a consequence of Theorem 9.52 and Theorem 9.53, which are proven in the subsequent two sections. □

From an intensional perspective the restriction to tail-recursion is severe. We remark that if we disallow parameter substitution, but allow tuples as values, the restriction to tail-recursive TRSs can be lifted, provided we impose a typing regime on analysed TRSs that prohibits nesting of tuples. In other words,

we would have to modify our computational model to account for multi-valued functions, as for instance in [55]. An easy adaption of [10] recovers then soundness under these assumptions, whilst retaining completeness. We feel however that such a typing regime would introduce a rather ad-hoc flavor to our formulation of computation by TRSs.

By folklore, every function can be transformed into a tail-recursive function. By the observation below Theorem 9.47, the introduced overhead is at most polynomial. Still at present it seems that such a transformation is not directly applicable to generalise Theorem 9.47 to non-tail recursion with parameter substitution. The imposed overhead seems just too high.

In the following, we suppose that the set of constructors $\mathcal{C}$ is monadic. We use $\vec{u}, \vec{v}$ and $\vec{w}$ for sequences of constructor terms, and if they are not used in rules then we suppose that they are ground, i.e., values. Denote by $\Sigma_{\mathcal{C}}$ the signature that contains for each $\mathsf{c} \in \mathcal{C}$ a dedicated letter $c \in \Sigma_{\mathcal{C}}$. The value $\mathsf{c_1}(\mathsf{c_2}(\dots \mathsf{c_{l-1}}(\mathsf{c_l})\dots))$ can be *encoded* as a word $c_1, \dots, c_l \in \mathbb{W}(\Sigma_{\mathcal{C}})$. Vice versa, for an alphabet Σ, denote by $\mathcal{C}_\Sigma$ the set of constructors that contains $\epsilon \in \mathcal{C}$ and for each letter $c \in \Sigma$ a unary constructor $\mathsf{c} \in \mathcal{C}$. Then each word $c_1, \dots, c_l \in \mathbb{W}(\Sigma)$ can be *encoded* as a value $\mathsf{c_1}(\mathsf{c_2}(\dots \mathsf{c_l}(\epsilon)\dots)) \in \mathcal{T}(\mathcal{C}_\Sigma)$. Having this correspondence in mind, we henceforth confuse words and values, avoiding notational overhead.

9.4.1. Soundness

Fix an ML-like TRS that is predicative tail-recursive $\mathcal{R}$, and denote by $\gtrsim$ the precedence underlying $\mathcal{R}$. We now show that the functions $[\![f]\!]_{\mathcal{R}}$ $(f \in \mathcal{D})$ computed by $\mathcal{R}$ can be implemented on a RM M_f operating in time $\mathcal{O}(n^d)$, for d the depth of recursion $\mathsf{rd}_{\mathcal{R}}(f)$ of f. To ease presentation, we first consider the sub-case where $\mathcal{R}$ is *simple*.

Definition 9.48 (Simple)**.** A rule $f(\vec{l_n}; \vec{l_s}) \to r$ is called *simple* if r is a constructor term or $r = g(\vec{r_g}; h_1(\vec{l_n}; \vec{l_s}), \dots, h_k(\vec{l_n}; \vec{l_s}))$, where $g \in \mathcal{F}$ and all function symbols h_i $(i = 1, \dots, k)$ are defined symbols. A TRS $\mathcal{R}$ is called *simple* if all its rules are simple.

Lemma 9.49. *If $\mathcal{R}$ is simple, then for every $f \in \mathcal{D}$, the function $[\![f]\!]_{\mathcal{R}}$ is computable on a register machine operating in time $p(n)$, where $p(n)$ is some polynomial function of degree $\mathsf{rd}_{\mathcal{R}}(f)$.*

Proof. For a sequence of values $\vec{w}$, abbreviate by $|\vec{w}|$ the sum of sizes of elements from $\vec{w}$. For each defined symbol f in $\mathcal{R}$ with k normal and l safe arguments, we define a corresponding RM M_f with input registers $\vec{x}_f = x_1^f, \dots, x_{k+l}^f$ and output register z^f. On input $\vec{u} = u_1, \dots, u_k$ and $\vec{v} = v_1, \dots, v_l$ the RMs M_f run in time

$$c_f \cdot |\vec{u}|^{\mathsf{rd}_{\mathcal{R}}(f)} + k_f \ ,$$

where c_f and k_f are constants depending only on f.

To simplify the presentation, we first suppose that the precedence of $\mathcal{R}$ is strict on defined symbols, i.e., $f \sim g$ for $f, g \in \mathcal{D}$ implies $f = g$. The construction is

by induction on the rank $\mathsf{rk}_{\gtrsim}(f)$ of f, the bound is proven by induction on the rank of f and side induction on $|\vec{u}|$. Suppose the input registers $\vec{x}_f$ hold the values $\vec{u}, \vec{v}$.

First observe that M_f is able to determine, in constant time depending only on $\mathcal{R}$, the rewrite rule applicable to $f(\vec{u}; \vec{v})$. Note that this rule is unique and exists, as $\mathcal{R}$ is by assumption orthogonal and completely defined. Since there are only a constant number of rules in $\mathcal{R}$, it suffice to realise that the time required for pattern matching depends only on $\mathcal{R}$. Suppose we want to match $f(\vec{u}; \vec{v})$ against the left hand-side $f(\vec{l_n}; \vec{l_s}) \to r \in \mathcal{R}$. Due to linearity, M_f can match the arguments $\vec{u}, \vec{v}$ against $\vec{l_n}, \vec{l_s}$ sequentially. For this, the RM M_f copies each argument in $\vec{u}, \vec{v}$ to a temporary register, and matches this value against the corresponding argument $l_i \in \vec{l_n}, \vec{l_s}$ using a constant number of jump and delete instructions.

Once the applicable rewrite rule has been identified, say after $k_m \in \mathbb{N}$ steps, the RM M_f can proceed according to its right-hand side as follows. If $f(\vec{u}; \vec{v})$ rewrites in one step to a value, say w, then $w = C[x\sigma]$ for some constructor context C and substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{C})$. Then some input register $x_i \in \vec{x}_f$ holds the word $C'[x\sigma]$. In this case M_f can provide the result w in register z_f using one copy instruction $C(x_i, z_f)$, a constant number of delete instructions $D(z_f)$ to delete C', i.e., z_f holds afterwards $x\sigma$, and a constant number of appropriate instructions to append the context C. Thus z_f holds after a constant number of steps the word $C[x\sigma]$ and trivially the theorem follows. Hence suppose $f(\vec{u}; \vec{v})$ does not rewrite to a value in one step. Since $\mathcal{R}$ is simple

$$f(\vec{u}; \vec{v}) \xrightarrow{\mathsf{i}}_{\mathcal{R}} g(w_1, \ldots, w_m; h_1(\vec{u}; \vec{v}), \ldots, h_n(\vec{u}; \vec{v})) \text{ where } h_1, \ldots, h_n \in \mathcal{D} \,.$$

As $\mathcal{R}$ is predicative recursive, $f \succ h_j$ holds for all $j = 1, \ldots, n$, and also either $f \succ g$ or $f = g$ holds (recall our assumption that $\gtrsim$ is strict on defined symbols). In both cases, order constraints on normal arguments give $f(\vec{u}; \vec{v}) \trianglerighteq\!\!\!\!/_{\approx}\ w_i$ $(i = 1, \ldots, m)$, i.e., some input register holds a super-term of w_i. As in the case where the reduct was a value, the RM $\mathcal{R}_f$ can prepare the arguments w_i in dedicated registers x_i^g for all $i = 1, \ldots, m$ in constant time. By induction hypothesis, there exist RMs M_{h_j} $(j = 1, \ldots, n)$ that on input registers $\vec{x}_{h_j}$ initialised by $\vec{u}, \vec{v}$, compute the value of $[\![h_j]\!]_{\mathcal{R}}(\vec{u}; \vec{v})$ in time $c_{h_i} + |\vec{v}|^{\mathsf{rd}_{\mathcal{R}}(h_i)} + k_{h_i}$. The RM M_f can use these machines as sub-procedures in order to compute the safe arguments given to h. For suitable chosen constants c_h as well as k_h, by induction hypothesis this requires at most

$$\sum_{i=1}^{n} (c_{h_i} \cdot |\vec{u}|^{\mathsf{rd}_{\mathcal{R}}(h_i)} + k_{h_i}) + k_1 \leqslant c_h \cdot |\vec{u}|^d + k_h \,,$$

computation steps. Here, the constant k_1 accounts for setting up the stage and $d := \max\{\mathsf{rd}_{\mathcal{R}}(h_1), \ldots, \mathsf{rd}_{\mathcal{R}}(h_n)\} \leqslant \mathsf{rd}_{\mathcal{R}}(f)$ denotes the maximal recursion depth of the defined symbols h_i. The interesting case is now when $g \in \mathcal{D}$, otherwise g is a monadic constructor and M_f finishes the computation by copying the computed argument to the output register z_f, and appending the corresponding letter for g to this output. We analyse the cases $f \succ g$ and $f = g$ for $g \in \mathcal{D}$ independently.

If $f \succ g$ then the RM M_f can use a machine M_g given by induction hypothesis that computes

$$[\![f]\!]_{\mathcal{R}}(\vec{u};\vec{v}) = [\![g]\!]_{\mathcal{R}}(w_1,\dots,w_m;[\![h_1]\!]_{\mathcal{R}}(\vec{u};\vec{v}),\dots,[\![h_n]\!]_{\mathcal{R}}(\vec{u};\vec{v}))\ ,$$

in time $c_g \cdot |\vec{w}|^{\mathsf{rd}_{\mathcal{R}}(g)} + k_g$ for $\vec{w} := w_1,\dots,w_m$, using the previously computed values $[\![h_i]\!]_{\mathcal{R}}(\vec{u};\vec{v})$ $(i = 1,\dots,n)$. In the considered case the rule is orientable by $\succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$, by the imposed order constraints on normal arguments $w_1,\dots,w_m$, we see $|w_i| \leqslant \max\{|u_j| \mid u_j \in \vec{u}\}$ for all $i = 1,\dots,m$. Consequently $|w| \leqslant m \cdot |\vec{u}|$, and hence computing $[\![f]\!]_{\mathcal{R}}(\vec{u};\vec{v})$ for $[\![h_i]\!]_{\mathcal{R}}(\vec{u};\vec{v})$ $(i = 1,\dots,n)$ given takes time

$$c_g \cdot |\vec{w}|^{\mathsf{rd}_{\mathcal{R}}(g)} + k_g \leqslant c_g \cdot (m \cdot |\vec{u}|)^{\mathsf{rd}_{\mathcal{R}}(g)} + k_g\ .$$

Recall that $k_m \in \mathbb{N}$ accounts for the time required for pattern matching. Summing up everything, the procedure finishes in time

$$k_m + \big(c_g \cdot (m \cdot |\vec{u}|)^{\mathsf{rd}_{\mathcal{R}}(g)} + k_g\big) + \big(c_h \cdot |\vec{u}|^d + k_h\big)\ .$$

Observe that by definition $d \leqslant \mathsf{rd}_{\mathcal{R}}(f)$ and $\mathsf{rd}_{\mathcal{R}}(g) \leqslant \mathsf{rd}_{\mathcal{R}}(f)$. Since m is bounded by the arity of g, it is not difficult to see that we can find sufficiently high constants $c_f, k_f \in \mathbb{N}$ depending only on $\mathcal{R}$ such that $c_f \cdot |\vec{u}|^{\mathsf{rd}_{\mathcal{R}}(f)} + k_f$ dominates the above expression.

Otherwise $f = g$, hence f is recursive. In this case the rule is orientable by $\succ^{\langle 3\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$. From the order constraint $\langle\vec{u}\rangle \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \langle\vec{w}\rangle$ on normal arguments it is not difficult to derive $|\vec{w}| < |\vec{u}|$. Recall $d = \max\{\mathsf{rd}_{\mathcal{R}}(h_1),\dots,\mathsf{rd}_{\mathcal{R}}(h_n)\} < \mathsf{rd}_{\mathcal{R}}(f)$ since f is recursive. Using the side induction hypothesis we conclude that M_f finishes in time $c_f \cdot |\vec{w}|^{\mathsf{rd}_{\mathcal{R}}(f)} + k_f$. Without loss of generality, suppose $k_h + k_m + c_h \leqslant c_f$. Overall the execution time is bounded by

$$\begin{aligned}
&k_m + \big(c_h \cdot |\vec{u}|^d + k_h\big) + \big(c_f \cdot |\vec{w}|^{\mathsf{rd}_{\mathcal{R}}(f)} + k_f\big)\\
&\quad\leqslant (c_h + k_h + k_m) \cdot |\vec{u}|^d + c_f \cdot |\vec{w}|^{\mathsf{rd}_{\mathcal{R}}(f)} + k_f && \text{using } 1 \leqslant |\vec{u}|\\
&\quad\leqslant c_f \cdot (|\vec{u}|^d + |\vec{w}|^{\mathsf{rd}_{\mathcal{R}}(f)}) + k_f && \text{as } k_h + k_m + c_h \leqslant c_f\\
&\quad\leqslant c_f \cdot |\vec{u}|^{\mathsf{rd}_{\mathcal{R}}(f)} + k_f && \text{as } |\vec{u}| > |\vec{w}| \text{ and } \mathsf{rd}_{\mathcal{R}}(f) > d,
\end{aligned}$$

as desired. We conclude this final case.

To lift the assumption on the precedence, suppose $\{f_1,\dots,f_\ell\}$ is the set of all function symbols equivalent to $f \in \mathcal{D}$, i.e., $f_1,\dots,f_\ell$ are defined by mutual recursion. Since this class is finite, one can store i (for $i = 1,\dots,\ell$) in a dedicated register of M_f, say r. Although more tedious, it is not difficult to see that the above construction can then be altered, so that M_f computes $f_{\langle r\rangle}(\vec{u};\vec{v})$ on input $\vec{u},\vec{v}$. □

We now remove the restriction that $\mathcal{R}$ is simple. For this we define the binary relation $\rightsquigarrow$ on constructor TRSs that transforms $\mathcal{R}$ into a simple TRS, retaining the assumptions on $\mathcal{R}$.

Definition 9.50. Let $h_1, \dots, h_k$ be fresh symbols not appearing in $\mathcal{F}$. Then

$$\begin{aligned}\mathcal{R} \uplus \{f(\vec{u}_f; \vec{v}_f) \to g(\vec{u}_g; t_1, \dots, t_k)\} \\ \rightsquigarrow \mathcal{R} \cup \{f(\vec{u}_f; \vec{v}_f) \to g(\vec{u}_g; h_1(\vec{u}_f; \vec{v}_f), \dots, h_k(\vec{u}_f; \vec{v}_f))\} \\ \cup \{h_i(\vec{u}_f; \vec{v}_f) \to t_i \mid i = 1, \dots, k\} \,,\end{aligned}$$

provided the *transformed rule* $f(\vec{u}_f; \vec{v}_f) \to g(\vec{u}_g; t_1, \dots, t_k)$ is not already simple.

The relation $\rightsquigarrow$ enjoys the following properties.

Lemma 9.51.

(1) *The relation $\rightsquigarrow$ is well-founded.*

(2) *Let $\mathcal{R}$ be an orthogonal, predicative tail-recursive constructor TRS of degree d that uses only monadic constructors. If $\mathcal{R} \rightsquigarrow \mathcal{S}$ then $\mathcal{S}$ enjoys the these mentioned properties too.*

(3) *If $\mathcal{R} \rightsquigarrow \mathcal{S}$ then $\xrightarrow{\mathsf{i}}_{\mathcal{R}} \subseteq \xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{S}}$.*

Proof. Let $\|\mathcal{R}\|$ denote the sum of the sizes of right-hand sides in *non-simple* rules in $\mathcal{R}$. It is not difficult to see that an infinite chain $\mathcal{R}_1 \rightsquigarrow \mathcal{R}_2 \rightsquigarrow \cdots$ would translate into an infinite descend $\|\mathcal{R}_1\| > \|\mathcal{R}_2\| > \cdots$, and thus $\rightsquigarrow$ is well-founded.

Consider the second property. Suppose $\mathcal{R} \rightsquigarrow \mathcal{S}$ with $\mathcal{R}$ satisfying all mentioned properties. Then trivially $\mathcal{S}$ is an orthogonal and tail-recursive constructor TRS. To see that $\mathcal{S}$ is predicative recursive with recursion depth d, let

$$f(\vec{u}_f; \vec{v}_f) \to g(\vec{u}_g; t_1, \dots, t_k) \,,$$

denote the rule which is replaced by

$$\begin{aligned} &f(\vec{u}_f; \vec{v}_f) \to g(\vec{u}_g; h_1(\vec{u}_f; \vec{v}_f), \dots, h_k(\vec{u}_f; \vec{v}_f)) \\ &h_i(\vec{u}_f; \vec{v}_f) \to t_i \qquad\qquad (i = 1, \dots, k). \end{aligned}$$

Let $\gtrsim$ denote the precedence underlying $\mathcal{R}$, and $\sqsupseteq$ the precedence underlying $\mathcal{S}$. Note that whereas f is defined based on h for all symbols h occurring in t_i in $\mathcal{R}$, i.e., $f \gg_{\mathcal{R}} g$, in $\mathcal{S}$ the symbol f is indirectly defined based on h via the fresh symbol h_i, i.e., $f \gg_{\mathcal{S}} h_i \gg_{\mathcal{S}} h$ holds for all h occurring in t_i $(i = 1, \dots, k)$. As otherwise $\gg_{\mathcal{R}}$ and $\gg_{\mathcal{S}}$ coincide, by definition $\sqsupseteq$ is the least extension of $\gtrsim$ such that $f \sqsupset h_i \sqsupset h$ holds for all h occurring in t_i $(i = 1, \dots, k)$, where $\sqsupset$ denotes the strict order induced by $\sqsupseteq$. As the freshly introduced symbols h_i are not recursive, from this we obtain that the recursion depth of every symbol $h \in \mathcal{F}$ is preserved, i.e. $\mathsf{rd}_{\mathcal{R}}(h) = \mathsf{rd}_{\mathcal{S}}(h)$.

It remains to verify that $\mathcal{S}$ is oriented by the order $\sqsupset_{\mathsf{spop}^*_{\mathsf{ps}}}$. Since $\gtrsim \subseteq \sqsupseteq$, and as a consequence $\succ_{\mathsf{spop}^*_{\mathsf{ps}}} \subseteq \sqsupseteq_{\mathsf{spop}^*_{\mathsf{ps}}}$, it suffices to show that the orientation $f(\vec{u}_f; \vec{v}_f) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} g(\vec{u}_g; t_1, \dots, t_k)$ of the replaced rule implies

$$f(\vec{u}_f; \vec{v}_f) \sqsupset_{\mathsf{spop}^*_{\mathsf{ps}}} g(\vec{u}_g; h_1(\vec{u}_f; \vec{v}_f), \dots, h_k(\vec{u}_f; \vec{v}_f)) \tag{9.1}$$

$$h_i(\vec{u}_f; \vec{v}_f) \sqsupset_{\mathsf{spop}^*_{\mathsf{ps}}} t_i \qquad (i = 1, \dots, k) \,. \tag{9.2}$$

We perform case analysis on the assumption. Suppose first $f(\vec{u}_f;\vec{v}_f) \succ^{\langle 1\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}} g(\vec{u}_g;t_1,\dots,t_k)$ holds. Note that since $\mathcal{R}$ is a constructor TRS, g is a constructor in the considered case. In particular g admits only safe argument positions, and thus $f(\vec{u}_f;\vec{v}_f) \sqsupseteq^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}} g(\vec{u}_g;h_1(\vec{u}_f;\vec{v}_f),\dots,h_k(\vec{u}_f;\vec{v}_f))$ holds using $f(\vec{u}_f;\vec{v}_f) \sqsupseteq^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}} h_i(\vec{u}_f;\vec{v}_f)$ for all $i = 1,\dots,k$. This concludes (9.1). The assumptions give $f(\vec{u}_f;\vec{v}_f) \rhd/\approx t_i$ for all $i = 1,\dots,k$ by Lemma 9.15(2), thus trivially $h_i(\vec{u}_f;\vec{v}_f) \sqsupseteq^{\langle 1\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}} t_i$ holds and we conclude (9.2).

Finally suppose that $f(\vec{u}_f;\vec{v}_f) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} g(\vec{u}_g;t_1,\dots,t_k)$ follows by $\succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$ or $\succ^{\langle 3\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$. Using the order constraint $f(\vec{u}_f;\vec{v}_f) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} h_i(\vec{u}_f;\vec{v}_f)$ for all $i = 1,\dots,k$, we see that (9.1) follows by $\sqsupseteq^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$ or $\sqsupseteq^{\langle 3\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$ respectively. For equation (9.2), observe that since $\mathcal{R}$ is tail-recursive, the assumption gives $f(\vec{u}_f;\vec{v}_f) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} t_i$ $(i = 1,\dots,k)$ using only applications of $\succ^{\langle 1\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$ and $\succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$. Repeating these proofs, but employing $h_i \sqsupseteq g$ instead of $f \succ g$, yields a proof of (9.2). This finishes the proof of Assertion (2).

For the final property, suppose $\mathcal{R} \rightsquigarrow \mathcal{S}$ and consider a rewrite step

$$C[f(\vec{u}_f\sigma;\vec{v}_f\sigma)] \xrightarrow{\mathrm{i}}_{\mathcal{R}} C[g(\vec{u}_g\sigma;t_1\sigma,\dots,t_k\sigma)] \,,$$

using the transformed rule $f(\vec{u}_f;\vec{v}_f) \to g(\vec{u}_g;t_1,\dots,t_k) \in \mathcal{R}$. Then

$$\begin{aligned} C[f(\vec{u}_f\sigma;\vec{v}_f\sigma)] &\xrightarrow{\mathrm{i}}_{\mathcal{S}} C[g(\vec{u}_g\sigma;h_1(\vec{u}_f\sigma;\vec{v}_f\sigma),\dots,h_k(\vec{u}_f\sigma;\vec{v}_f\sigma))] \\ &\xrightarrow{\mathrm{i}}{}^{k}_{\mathcal{S}} C[g(\vec{u}_g\sigma;t_1\sigma,\dots,t_k\sigma)] \,, \end{aligned}$$

simulates the considered step. Here we employ that $\vec{u}_f\sigma, \vec{v}_f\sigma \in \mathcal{T}(\mathcal{F},\mathcal{V})$ are normal forms of $\mathcal{R}$, since the symbols $h_1,\dots,h_k$ are supposed to be fresh, the arguments $\vec{u}_f\sigma$ and $\vec{v}_f\sigma$ are also normal forms of $\mathcal{S}$. This proves $\xrightarrow{\mathrm{i}}_{\mathcal{R}} \subseteq \xrightarrow{\mathrm{i}}{}^{+}_{\mathcal{S}}$. □

Theorem 9.52. *Let $\mathcal{R}$ denote an* ML*-like predicative tail-recursive TRSs of degree d, using only monadic constructors. For every $f \in \mathcal{D}$, the function $[\![f]\!]_{\mathcal{R}}$ is computable on a register machine operating in time $p(n)$, where $p(n)$ is some polynomial function of degree $\mathsf{rd}_{\mathcal{R}}(f)$.*

Proof. Let f be a defined function symbol from $\mathcal{R}$. Let $\mathcal{S}$ be a $\rightsquigarrow$-normal form of our analysed TRS $\mathcal{R}$. Note that by Lemma 9.51(1), $\mathcal{S}$ is well-defined and moreover it is simple by construction. By Lemma 9.51(2) and assumption on $\mathcal{R}$, the simple TRS $\mathcal{S}$ is an orthogonal, predicative tail-recursive constructor TRS of degree d that uses only monadic constructors. In particular, $\mathcal{S}$ computes the function $[\![f]\!]_{\mathcal{R}}$ by Lemma 9.51(3). Hence all requirements of Lemma 9.49 are met, except that in $\mathcal{S}$ the auxiliary function symbols h introduced by $\rightsquigarrow$ are not completely defined. Inspecting the proof of Lemma 9.49, we see that the only place where this is required is in the matching algorithm, so that for the intermediate computation of $[\![h]\!]_{\mathcal{R}}(\vec{u};\vec{v})$ exactly one applicable rewrite rule can be found. This property is retained in the implementation of $[\![f]\!]_{\mathcal{R}}$ through $\mathcal{S}$, and thus we can use the machine defined in Lemma 9.49 on $\mathcal{S}$ in order to compute $[\![f]\!]_{\mathcal{R}}$ under the given bound. □

9.4.2. Completeness

We now prove the inverse direction: given a RM M which computes a function f in time $p(n) \in \mathcal{O}(n^d)$, we construct an ML-like predicative tail-recursive TRS of degree d that computes f. For a sequence of functions symbols $\vec{f} = f_1, \ldots, f_{k+l}$ with exactly k normal and l safe arguments denote by $\vec{f}^{\langle n \rangle}(\vec{s}; \vec{t})$ the n-fold parallel composition of $\vec{f}$ on terms $\vec{s}, \vec{t}$, given by $\vec{f}^{\langle 0 \rangle}(\vec{s}; \vec{t}) := \vec{s}; \vec{t}$ and

$$\vec{f}^{\langle n+1 \rangle}(\vec{s}; \vec{t}) := \\ f_1(\vec{f}^{\langle n \rangle}(\vec{s}; \vec{t})), \ldots, f_k(\vec{f}^{\langle n \rangle}(\vec{s}; \vec{t})); f_{k+1}(\vec{f}^{\langle n \rangle}(\vec{s}; \vec{t})), \ldots, f_{k+l}(\vec{f}^{\langle n \rangle}(\vec{s}; \vec{t})) \; .$$

By a standard induction one verifies that $\vec{f}^{\langle n+m \rangle}(\vec{s}; \vec{t}) = \vec{f}^{\langle n \rangle}(\vec{f}^{\langle m \rangle}(\vec{s}; \vec{t}))$ holds for all $n, m \in \mathbb{N}$. We use

$$\vec{f}_i^{\langle n+1 \rangle}(\vec{s}; \vec{t}) := f_i(\vec{f}_i^{\langle n \rangle}(\vec{s}; \vec{t})) \; ,$$

to refer to the i^{th} element ($i = 1, \ldots, k+l$) in the sequence $\vec{f}^{\langle n+1 \rangle}(\vec{s}; \vec{t})$. Further, we denote by $\vec{f}(\vec{s}; \vec{t})$ the special case

$$\vec{f}^{\langle 1 \rangle}(\vec{s}; \vec{t}) = f_1(\vec{s}; \vec{t}), \ldots, f_k(\vec{s}; \vec{t}); f_{k+1}(\vec{s}; \vec{t}), \ldots, f_{k+l}(\vec{s}; \vec{t}) \; .$$

The next theorem shows our completeness result.

Theorem 9.53. *Consider a function f which is computable in time $p(n)$ for $p(n) \in \mathcal{O}(n^d)$ and $d \geqslant 1$ on a register machine. Then f is computed by an* ML*-like tail-recursive TRS $\mathcal{R}$ of degree d. In particular $\mathcal{R}$ uses only monadic constructors.*

Proof. Let $M = (R, \Sigma, P)$ be a RM with registers $R = \{r_1, \ldots, r_m\}$ and instructions $P = I_1, \ldots, I_l$ that computes the function f with k input registers in time $p(n) \leqslant c \cdot |n|^d + e$, for $c, e \in \mathbb{N}$.

For each $c \in \Sigma$, let c be a unary constructor symbol, and let ϵ denote a constant that represents the empty word ϵ, which allows us to represent words $\mathbb{W}(\Sigma)$ as values over these constructors. As before, we confuse values with words to avoid notational overhead. For $j = 1, \ldots, l+1$, let j denote a distinct constant. These constants are used to denote the labels of the program P, including the dedicated halting label $l+1$. The proof follows the outline given in the introduction of this section, where we use a TRS $\mathcal{R}_0^M$ to simulate the one-step transition relation of M by a TRS $\mathcal{R}_0^M$. Unlike the sketched definition of $\mathcal{R}_0^M$, we introduce a family of defined symbols $\vec{\mathsf{M}} := \mathsf{M}_0, \ldots, \mathsf{M}_m$. Here the intention is that whenever

$$(j, v_1, \ldots, v_m) \to_M (j', v_1', \ldots, v_m') \; ,$$

holds, then the term $\mathsf{M}_i(\mathsf{j}, v_1, \ldots, v_m)$ reduces to the $(i+1)^{\text{th}}$ component of the configuration $(j', v_1', \ldots, v_m')$.

The symbols $\vec{\mathsf{M}}$ are defined in the TRS $\mathcal{R}_0^M$ which contains for $i = 1, \ldots, m$ and each $j = 1, \ldots, l$ the following rules, depending on the j^{th} instruction I_j. Let $r_1, \ldots, r_m$ denote pairwise distinct variables.

(1) If $I_j = A^{(a)}(r_{i'})$, then $\mathcal{R}_0^M$ contains the rule

$$\begin{aligned} \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) &\to \mathsf{a}(r_i) && \text{if } i = i', \\ \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) &\to r_i && \text{if } i \neq i'. \end{aligned}$$

(2) If $I_j = D(r_{i'})$, then $\mathcal{R}_0^M$ contains the rule(s)

$$\begin{aligned} \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, \epsilon, \ldots, r_m) &\to \epsilon && \text{if } i = i', \\ \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, \mathsf{a}(r_i), \ldots, r_m) &\to r_i && \text{if } i = i' \text{ for all } a \in \Sigma, \\ \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) &\to r_i && \text{if } i \neq i'. \end{aligned}$$

(3) If $I_j = J^{(a)}(r_{i'})[j']$, then $\mathcal{R}_0^M$ contains the rule

$$\mathsf{M}_i(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) \to r_i \quad .$$

(4) If $I_j = C(r_{i_1}, r_{i_2})$, then $\mathcal{R}_0^M$ contains the rule

$$\begin{aligned} \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) &\to r_{i_1} && \text{if } i = i_2, \\ \mathsf{M}_i(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) &\to r_i && \text{if } i \neq i_2. \end{aligned}$$

Further, for each $j = 1, \ldots, l$, the TRS $\mathcal{R}_0^M$ contains the following rules that compute the next instruction label:

(1) If $I_j = J^{(a)}(r_i)[j']$, then $\mathcal{R}_0^M$ contains the rules

$$\begin{aligned} \mathsf{M}_0(;\mathsf{j}, r_1, \ldots, \epsilon, \ldots, r_m) &\to \mathsf{j{+}1} \\ \mathsf{M}_0(;\mathsf{j}, r_1, \ldots, \mathsf{a}(r_i), \ldots, r_m) &\to \mathsf{j}' \\ \mathsf{M}_0(;\mathsf{j}, r_1, \ldots, \mathsf{b}(r_i), \ldots, r_m) &\to \mathsf{j{+}1} && \text{for all } b \in \Sigma,\ b \neq a. \end{aligned}$$

(2) If I_j is any other instruction, then $\mathcal{R}_0^M$ contains the rule

$$\mathsf{M}_0(;\mathsf{j}, r_1, \ldots, r_i, \ldots, r_m) \to \mathsf{j{+}1} \quad .$$

Finally, the TRS $\mathcal{R}_0^M$ contains the rules

$$\begin{aligned} \mathsf{M}_0(;\mathsf{l{+}1}, r_1, \ldots, \mathsf{a}(r_i), \ldots, r_m) &\to \mathsf{l{+}1} \\ \mathsf{M}_i(;\mathsf{l{+}1}, r_1, \ldots, \mathsf{a}(r_i), \ldots, r_m) &\to \mathsf{l{+}1} && \text{for } i = 1, \ldots, m. \end{aligned}$$

This completes the definition of $\mathcal{R}_0^M$. Note that $\mathcal{R}_0^M$ is ML-like by definition. By case analysis one verifies that $\mathcal{R}_0^M$ accounts for the one-step transition relation $\to_M$ as indicated above. Using this, a standard induction gives that whenever

$$(1, w_1, \ldots, w_k, \vec{\epsilon}) \to_M^{\ell} (j, v_1, \ldots, v_m) \,,$$

then

$$\vec{\mathsf{M}}_m^{\langle \ell \rangle}(;\mathsf{1}, w_1, \ldots, w_k, \vec{\epsilon}) \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}_0^M} v_m \,, \tag{9.3}$$

holds. As M runs in time $c\cdot|\vec{w}|^d+e$, the latter reduction computes $f(w_1,\dots,w_k)$, i.e., $f(w_1,\dots,w_k)=v_m$, provided $\ell \geqslant c\cdot|\vec{w}|^d+e$.

The TRS $\mathcal{R}_f$ is an extension of $\mathcal{R}_0^M$, that defines a k-ary function f such that

$$\mathsf{f}(w_1,\dots,w_k;)\xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}_0^M}\vec{\mathsf{M}}_m^{\langle\ell\rangle}(;\mathsf{1},w_1,\dots,w_k,\vec{\epsilon}) \quad \text{for some } \ell\geqslant c\cdot|\vec{w}|^d+e, \tag{9.4}$$

The definition of f relies on auxiliary function symbols f_i^r with $2\cdot k$ normal and $m+1$ safe arguments, for $r=1,\dots,d$ and $i=0,\dots,m$, where r reflects the depth of recursion of f_i^r. The first $2\cdot k$ argument hold the recursion parameters and a copy of them, and as for M_i the safe arguments are used to hold the contents of a configuration. Let $\vec{x}:=x_1,\dots,x_k$, $\vec{y}:=y_1,\dots,y_k$ and $\vec{z}:=z_0,\dots,z_m$ denote pairwise distinct variables. We define $\mathcal{R}_f$ as the extension of $\mathcal{R}_0^M$ by the following rules, where $r=1,\dots,d$, $j=1,\dots,k$ and $a\in\Sigma$.

$$\begin{aligned}
\mathsf{f}(\vec{x};)&\to\vec{\mathsf{M}}_m^{\langle e\rangle}(;\mathsf{f}_0^d(\vec{x},\vec{x};\mathsf{1},\vec{x},\vec{\epsilon}),\dots,\mathsf{f}_m^d(\vec{x},\vec{x};\mathsf{1},\vec{x},\vec{\epsilon}))\\
\mathsf{f}_i^0(\vec{x},\vec{y};\vec{z})&\to z_i\\
\mathsf{f}_i^r(\vec{\epsilon},\vec{y};\vec{z})&\to z_i\\
\mathsf{f}_i^r(\vec{\epsilon},\mathsf{a}(x_j),\dots,x_k,\vec{y};\vec{z})&\to\\
&\quad\mathsf{f}_i^r(\vec{\epsilon},x_j,\dots,x_k,\vec{y};\vec{\mathsf{M}}^{\langle c\rangle}(;\mathsf{f}_0^{r-1}(\vec{y},\vec{y};\vec{z}),\dots,\mathsf{f}_m^{r-1}(\vec{y},\vec{y};\vec{z})))\,.
\end{aligned}$$

To verify (9.4), we show that for some $\ell\geqslant c\cdot|\vec{u}|\cdot|\vec{v}|^{r-1}$,

$$\mathsf{f}_i^r(\vec{u},\vec{v};\vec{t})\xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}}\vec{\mathsf{M}}_i^{\langle\ell\rangle}(;\vec{t})\,,$$

holds for all $r=1,\dots,d$, $i=0,\dots,m$, terms $\vec{t}$ and arbitrary words $\vec{u},\vec{v}$. We prove the claim by induction on r and side induction on the sum of the sizes of words $\vec{u}$. Consider the base case $r=1$, we prove $\mathsf{f}_i^1(\vec{u},\vec{v};\vec{t})\to\vec{\mathsf{M}}_i^{\langle c\cdot|\vec{u}|\rangle}(;\vec{t})$ for all $i=0,\dots,m$. If $|\vec{u}|=0$ then $\vec{u}=\vec{\epsilon}$ and so we have $\mathsf{f}_i^1(\vec{\epsilon},\vec{v};\vec{t})\xrightarrow{\mathrm{i}}_{\mathcal{R}}t_i=\vec{\mathsf{M}}_i^{\langle 0\rangle}(;\vec{t})$ as desired. In the inductive step of the side induction, $|\vec{u}|>0$ and hence $\vec{u}=\vec{\epsilon},\mathsf{a}(u_j),\dots,u_k$ for some $j\in\{1,\dots,k\}$ and $a\in\Sigma$. Let $\vec{u}':=\vec{\epsilon},u_j,\dots,u_k$. Then using the side induction hypothesis we have

$$\begin{aligned}
\mathsf{f}_i^1(\vec{u},\vec{v};\vec{t})&\xrightarrow{\mathrm{i}}_{\mathcal{R}_f}\mathsf{f}_i^1(\vec{u}',\vec{v};\vec{\mathsf{M}}^{\langle c\rangle}(;\vec{\mathsf{f}}^0(\vec{v},\vec{v};\vec{t})))\\
&\to^{m+1}_{\mathcal{R}_f}\mathsf{f}_i^1(\vec{u}',\vec{v};\vec{\mathsf{M}}^{\langle c\rangle}(;\vec{t}))\\
&\xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}_f}\vec{\mathsf{M}}_i^{\langle c\cdot|\vec{u}'|\rangle}(;\vec{\mathsf{M}}^{\langle c\rangle}(;\vec{t}))\\
&=\vec{\mathsf{M}}_i^{\langle c\cdot(|\vec{u}'|+1)\rangle}(;\vec{t})\,.
\end{aligned}$$

As $|\vec{u}'|+1=|\vec{u}|$ we conclude the base case.

Consider now the inductive step $r>1$. The case $|\vec{u}|=0$ follows as above, so suppose again $\vec{u}=\vec{\epsilon},\mathsf{a}(u_j),\dots,u_k$ for some $j\in\{1,\dots,k\}$, and let $\vec{u}':=\vec{\epsilon},u_j,\dots,u_k$. Then using induction hypothesis and side induction hypothesis we

conclude

$$\begin{aligned}
&\mathsf{f}_i^{r+1}(\vec{u},\vec{v};\vec{t})\\
&\quad\xrightarrow{\mathsf{i}}_{\mathcal{R}_f} \mathsf{f}_i^{r+1}(\vec{u}',\vec{v};\vec{\mathsf{M}}^{\langle c\rangle}(;\vec{\mathsf{f}}^{r}(\vec{v},\vec{v};\vec{t})))\\
&\quad\xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}_f} \mathsf{f}_i^{r+1}(\vec{u}',\vec{v};\vec{\mathsf{M}}^{\langle c\rangle}(;\vec{\mathsf{M}}^{\langle \ell_1\rangle}(;\vec{t}))) \qquad \text{where } \ell_1 \geqslant c\cdot|\vec{v}|\cdot|\vec{v}|^{r-1} = c\cdot|\vec{v}|^{r}\\
&\quad\xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}_f} \vec{\mathsf{M}}_i^{\langle \ell_2\rangle}(;\vec{\mathsf{M}}^{\langle c\rangle}(;\vec{\mathsf{M}}^{\langle \ell_1\rangle}(;\vec{t}))) \qquad \text{where } \ell_2 \geqslant c\cdot|\vec{u}'|\cdot|\vec{v}|^{r}\\
&\quad= \vec{\mathsf{M}}_i^{\langle \ell_2+c+\ell_1\rangle}(;\vec{t})\;.
\end{aligned}$$

Summing up we have

$$\ell_2 + c + \ell_1 \geqslant c\cdot(|\vec{u}'|\cdot|\vec{v}|^{r} + |\vec{v}|^{r}) = c\cdot(|\vec{u}'|+1)\cdot|\vec{v}|^{r} = c\cdot|\vec{u}|\cdot|\vec{v}|^{r}\;,$$

as desired. Overall we obtain

$$\begin{aligned}
\mathsf{f}(\vec{w};) &\xrightarrow{\mathsf{i}}_{\mathcal{R}_f} \vec{\mathsf{M}}_m^{\langle e\rangle}(;\mathsf{f}_0^d(\vec{w},\vec{w};1,\vec{w},\vec{\epsilon}),\dots,\mathsf{f}_m^d(\vec{w},\vec{w};1,\vec{w},\vec{\epsilon}))\\
&\xrightarrow{\mathsf{i}}{}^{+}_{\mathcal{R}_f} \vec{\mathsf{M}}_m^{\langle e\rangle}(;\vec{\mathsf{M}}_1^{\langle \ell\rangle}(;1,w_1,\dots,w_k,\vec{\epsilon}),\dots,\vec{\mathsf{M}}_m^{\langle \ell\rangle}(;1,w_1,\dots,w_k,\vec{\epsilon}))\\
&= \vec{\mathsf{M}}_m^{\langle \ell+e\rangle}(;1,w_1,\dots,w_k,\vec{\epsilon})
\end{aligned}$$

where $\ell + e \geqslant c\cdot|\vec{w}|\cdot|\vec{w}|^{d-1} + e = c\cdot|\vec{w}|^{d} + e$, which proves derivation (9.4). By definition $\mathcal{R}_f$ is an ML-like, tail-recursive TRS, in particular $[\![\mathsf{f}]\!]_{\mathcal{R}}$ is a function. Combining this with the observation on derivations (9.3) we conclude that this TRS implements the function f, more precise, $[\![\mathsf{f}]\!]_{\mathcal{R}}(w_1,\dots,w_k) = f(w_1,\dots,w_k)$ holds.

To conclude the theorem, we show that $\mathcal{R}_f$ is predicative recursive of degree d. Observe that the precedence $\gtrsim$ of $\mathcal{R}_f$ on defined symbols is given by

$$\mathsf{f} \succ \mathsf{f}_0^d,\dots,\mathsf{f}_m^d \succ \cdots \succ \mathsf{f}_0^0,\dots,\mathsf{f}_m^0 \succ \mathsf{M}_0,\dots,\mathsf{M}_m\;,$$

where only the symbols f_i^r are recursive in $\mathcal{R}_f$. In particular $\mathsf{rd}_{\mathcal{R}_f}(\mathsf{f}_i^r) = r$ for all $r = 0,\dots,d$ and $i = 0,\dots,m$ and so the recursion depth of $\mathcal{R}_f$ is d. Using the precedence, it is also not difficult to see that $\mathcal{R}_f$ is predicative recursive. The only non-trivial cases are to show that

$$\begin{aligned}
\mathsf{f}(\vec{x};) &\succ_{\mathsf{spop}^*_{\mathsf{ps}}} \vec{\mathsf{M}}_m^{\langle e\rangle}(;\mathsf{f}_0^d(\vec{x},\vec{x};1,\vec{x},\vec{\epsilon}),\dots,\mathsf{f}_m^d(\vec{x},\vec{x};1,\vec{x},\vec{\epsilon}))\;,\ \text{and}\\
\mathsf{f}_i^r(\vec{\epsilon},\mathsf{a}(x_j),\dots,x_k,\vec{y};\vec{z}) &\succ_{\mathsf{spop}^*_{\mathsf{ps}}}\\
&\quad \mathsf{f}_i^r(\vec{\epsilon},x_j,\dots,x_k,\vec{y};\vec{\mathsf{M}}^{\langle c\rangle}(;\mathsf{f}_0^{r-1}(\vec{y},\vec{y};\vec{z}),\dots,\mathsf{f}_m^{r-1}(\vec{y},\vec{y};\vec{z})))\;,
\end{aligned}$$

hold for all $r = 1,\dots,d$, $i = 0,\dots,m$ and $a \in \Sigma$. For the orientation of the first rule, it is straight forward to verify that

$$\mathsf{f}(\vec{x};) \succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}} \mathsf{f}_i^d(\vec{x},\vec{x};1,\vec{x},\vec{\epsilon}) \qquad \text{for all } i = 0,\dots,m,$$

holds. Using that also $\mathsf{f} \succ \mathsf{M}_i$ holds ($i = 0,\dots,m$), iterated application of $\succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$ orients the considered rule. Formally, this can be verified by a standard induction on e. In a similar spirit one can show that

$$\begin{aligned}
\mathsf{f}_i^l(\vec{\epsilon},\mathsf{a}(x_j),\dots,x_k,\vec{y};\vec{z}) &\succ_{\mathsf{spop}^*_{\mathsf{ps}}} \vec{\mathsf{M}}_i^{\langle c\rangle}(;\mathsf{f}_0^{l-1}(\vec{y},\vec{y};\vec{z}),\dots,\mathsf{f}_m^{l-1}(\vec{y},\vec{y};\vec{z}))\\
&\qquad\qquad \text{for all } i = 0,\dots,m \text{ and } l = 1,\dots,d,
\end{aligned}$$

holds. Using that also $\langle \vec{\epsilon}, \mathsf{a}(x_j), \ldots, x_k, \vec{y} \rangle \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \langle \vec{\epsilon}, x_j, \ldots, x_k, \vec{y} \rangle$ holds, orientation of the final rules follows by one application of $\succ^{\langle 3 \rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$. $\square$

Chapter 10.

The Exponential Path Order

In this chapter we provide an order theoretic characterisation of the class **FEXP**. This work is based on Arai and Eguchi [2] class $\mathcal{N}$, which gives a recursion-theoretic account of **FEXP**. Conceptually, the class $\mathcal{N}$ extends the class $\mathcal{B}_{\mathcal{W}}$ given in Definition 9.34 by *safe nested recursion (on notation)* (**SNRN** for short). This recursion scheme extends safe recursion on notation (**SRN**). On the one hand, nested recursive calls are allowed. On the other hand lexicographic descents on normal arguments are permitted. As an example, the following equations define a function f by safe nested recursion on notation.

$$
\begin{aligned}
f(\epsilon, \epsilon; z) &= g(; z) \\
f(\epsilon, yj; z) &= r_{\epsilon,j}(y; z, f(\epsilon, y; z)) && (j = 0, 1) \\
f(xi, \epsilon; z) &= r_{i,\epsilon}(x; z, f(x, x; s_{i,\epsilon}(x; z, f(x, xi; z)))) && (i = 0, 1) \\
f(xi, yj; z) &= r_{i,j}(x, y; z, f(xi, y; s_{i,j}(x, y; z, f(x, yj; z)))) && (i = 0, 1 \text{ and } j = 0, 1)\,.
\end{aligned}
$$

The following definition introduces the *exponential path order* $\succ_{\mathsf{epo}^\star}$ (EPO* for short). It is an extension of small polynomial path orders $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$, with two essential differences. In contrast to clause $\succ^{\langle 2\rangle}_{\mathsf{spop}^*_{\mathsf{ps}}}$, the linearity condition in clause $\succ^{\langle 2\rangle}_{\mathsf{epo}^\star}$ has been dropped allow multiple recursive calls. Further, in clause $\succ^{\langle 3\rangle}_{\mathsf{epo}^\star}$ we lift the product comparison on normal arguments to a lexicographic comparison.

Definition 10.1 (Exponential Path Order)**.** Let $\gtrsim$ denote a quasi-precedence on $\mathcal{F}$, with underlying proper order $\succ$ and equivalence $\sim$. Fix a safe mapping. Then $s \succ_{\mathsf{epo}^\star} t$ for terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $s = f(s_1, \ldots, s_k; s_{k+1}, \ldots, s_{k+l})$ if one of the following alternatives hold.

(1) $s_i \gtrsim_{\mathsf{epo}^\star} t$ for some argument s_i of s ($i \in \{1, \ldots, k + l\}$); or

(2) $f \in \mathcal{D}$, $t = g(t_1, \ldots, t_m; t_{m+1}, \ldots, t_{m+n})$ where $f \succ g$ and the following holds:

- $s \rhd\!\!\!/_{\approx} t_j$ for all normal arguments $t_1, \ldots, t_m$; and
- $s \succ_{\mathsf{epo}^\star} t_j$ for all safe arguments $t_{m+1}, \ldots, t_{m+n}$; or

(3) $f \in \mathcal{D}$, $t = g(t_1, \ldots, t_m; t_{m+1}, \ldots, t_{m+n})$ where $f \sim g$ and for some $j \in \{1, \ldots, \min(l, m)\}$ the following holds:

- $s_i \approx t_i$ for all $i = 1, \ldots, j - 1$; and
- $s_j \rhd\!\!\!/_{\approx} t_j$; and

- $s \trianglerighteq^{n}_{\not\approx} t_i$ for all $i = j+1, \dots, m$; and
- $s \succ_{\mathsf{epo}^\star} t_i$ for all $j = m+1, \dots, m+n$.

Here $s \gtrsim_{\mathsf{epo}^\star} t$ denotes that either $s \approx t$ or $s \succ_{\mathsf{epo}^\star} t$ holds.

Definition 10.2. We call the TRS $\mathcal{R}$ *predicative nested recursive* if $\mathcal{R}$ is compatible with an instance of EPO*.

Theorem 10.3. *Suppose $\mathcal{R}$ is a predicative recursive constructor TRS of degree d. Then the innermost derivation height of any basic term $f(\vec{u}; \vec{v})$ is bounded by an exponential $e(n) \in 2^{\mathcal{O}(n^k)}$, where $k \in \mathbb{N}$ depends only on the TRS $\mathcal{R}$ and n denotes the sum of the depths of normal arguments $\vec{u}$. In particular, the innermost runtime function $\mathrm{rci}_{\mathcal{R}}$ is bounded by an exponential in $2^{\mathcal{O}(n^k)}$.*

Example 10.4. The following TRS $\mathcal{R}_{\mathsf{fib}}$ is an extension of Example 2.28 from Steinbach and Kühlers collection of TRSs [74], and encodes the computation of the n^{th} Fibonacci number, using a nested recursive auxiliary function dfib.

$$\begin{array}{ll} 21\colon\ \mathsf{fib}(x;) \to \mathsf{dfib}(x;0) & 22\colon\ \mathsf{dfib}(0;y) \to y \\ 23\colon\ \mathsf{dfib}(\mathsf{s}(;0);y) \to \mathsf{s}(;;y) & 24\colon\ \mathsf{dfib}(\mathsf{s}(;\mathsf{s}(;x));y) \to \mathsf{dfib}(\mathsf{s}(x);\mathsf{dfib}(x;y))\ . \end{array}$$

This TRS is compatible with $\succ_{\mathsf{epo}^\star}$ using the induced precedence and safe mapping as indicated by the rules. Using $\mathsf{fib} \succ \mathsf{dfib}$ as well as $\mathsf{fib} \succ \mathsf{s}$ we can orient rules 21–23 using $\succ^{\langle 1\rangle}_{\mathsf{epo}^\star}$ and $\succ^{\langle 2\rangle}_{\mathsf{epo}^\star}$ only. For rule 24, observe $\mathsf{s}(;\mathsf{s}(;x)) \trianglerighteq_{\not\approx} \mathsf{s}(;x)$ holds, and thus $\mathsf{dfib}(\mathsf{s}(;\mathsf{s}(;x));y) \succ_{\mathsf{spop}^\star} \mathsf{dfib}(\mathsf{s}(x);\mathsf{dfib}(x;y))$ holds by two applications of $\succ^{\langle 3\rangle}_{\mathsf{epo}^\star}$ followed by one application of $\succ^{\langle 1\rangle}_{\mathsf{epo}^\star}$.

On the other hand, the rewrite system $\mathcal{R}_{\mathsf{fib}}$ is neither compatible with $\succ_{\mathsf{spop}^\star}$ nor $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$, due to the nested call in rule 24. ◁

Observe that $\succ_{\mathsf{epo}^\star}$ is sufficiently large to handle the function f defined by safe nested recursion above Definition 10.1, when formulated as rewrite system. We prove this formally later in Section 10.2. The following example stresses that the restriction to innermost reductions is still essential.

Example 10.5. Consider the TRS $\mathcal{R}_{\mathsf{ndup}}$ which consists of the following rules.

$$\begin{array}{ll} 25\colon\ \mathsf{f}(0;y) \to y & 26\colon\ \mathsf{dup}(;t) \to \mathsf{c}(t,t) \\ 27\colon\ \mathsf{f}(\mathsf{s}(;x);y) \to \mathsf{f}(x;\mathsf{dup}(;\mathsf{f}(x;y)))\ . & \end{array}$$

Then $\mathcal{R}_{\mathsf{ndup}}$ is a predicative nested recursive constructor TRS.

Employing that $\mathsf{dh}(\mathsf{dup}(;t), \to_{\mathcal{R}_{\mathsf{ndup}}}) = 1{+}2{\cdot}\mathsf{dh}(t, \to_{\mathcal{R}_{\mathsf{ndup}}})$ by rule 26, a standard induction verifies that $\mathsf{dh}(\mathsf{f}(\mathsf{s}^n(;0),t;), \to_{\mathcal{R}_{\mathsf{ndup}}}) \geqslant 2^{2^{n-1}} \cdot (1 + \mathsf{dh}(t, \to_{\mathcal{R}_{\mathsf{ndup}}}))$ holds for all $n \geqslant 1$ and terms t, thus the innermost runtime complexity function is not bounded by an exponential from $2^{\mathcal{O}(n^k)}$ for some $k \in \mathbb{N}$. ◁

10.1. Exponential Path Orders are Sound

We now prove Theorem 10.3. In correspondence to the soundness proof of sPOP*, we introduce an auxiliary order $\succ_\ell$ on sequences of terms, that allows the predicative embedding of the innermost rewrite relation. The maximal length of $\succ_\ell$ sequences can then be used to bind the length of $\mathcal{R}$ reductions sufficiently.

10.1.1. Exponential Polynomial Path Order on Sequences

The following definition introduces the *exponential path order* $\succ_\ell$ *on sequences*. This order constitutes a slight variation of the path order EPO introduced by Eguchi [31]. Its definition corresponds to Definition 9.19, where the linearity requirement in clause $\succ^{\langle 3\rangle}_{\mathcal{K},\ell}$ has been dropped, and clause $\succ^{\langle 1\rangle}_{\mathcal{K},\ell}$ is modified in order to account for the lexicographic status underlying $\succ_{\mathsf{epo}^\star}$.

Definition 10.6. Let $\gtrsim$ denote a quasi-precedence on $\mathcal{F}$, with underlying proper order $\succ$ and equivalence $\sim$. Let $\ell \in \mathbb{N}$ with $\ell \geqslant 1$. Then $a \succ_\ell b$ holds for terms or sequences of terms $a, b \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$ if one of the following alternatives hold.

(1) $a = f(s_1,\dots,s_k)$, $b = g(t_1,\dots,t_l)$ with $f \succ g$ and the following conditions hold:
- $f(s_1,\dots,s_k) \rhd\!\!\!/_{\approx} t_j$ for all $j = 1,\dots,l$; and
- $l \leqslant \ell$; or

(2) $a = f(s_1,\dots,s_k)$, $b = g(t_1,\dots,t_l)$ with $f \sim g$ and there exists some $j \in \{1,\dots,\min(k,l)\}$ such that the following conditions hold:
- $s_i \approx t_i$ for all $i = 1,\dots,j-1$; and
- $s_j \rhd\!\!\!/_{\approx} t_j$; and
- $a \rhd\!\!\!\!\!\!\!\!\!/_{\approx} t_i$ for all $i = j+1,\dots,m$; and
- $l \leqslant \ell$; or

(3) $a = f(s_1,\dots,s_k)$, $b = [t_1,\dots,t_l]$ and the following conditions hold:
- $f(s_1,\dots,s_k) \succ_\ell t_j$ for all $j = 1,\dots,l$; and
- $l \leqslant \ell$; or

(4) $a = [s_1,\dots,s_k]$, $b = [t_1,\dots,t_l]$ and there exists terms *or* sequences b_i $(i = 1,\dots,k)$ such that:
- $[t_1,\dots,t_l] = b_1 +\!\!+ \cdots +\!\!+ b_k$; and
- $\langle s_1,\dots,s_k\rangle \succ_{\mathcal{K},\ell} \langle b_1,\dots,b_k\rangle$.

We denote by $a \gtrsim_\ell b$ that either $a \approx b$ or $a \succ_\ell b$ holds. Here $\succ_\ell$ is also used for its extension to products: $\langle a_1,\dots,a_k\rangle \succ_\ell \langle b_1,\dots,b_k\rangle$ if $a_i \gtrsim_\ell b_i$ for all $i = 1,\dots,n$, and $a_{i_0} \succ_\ell a_{i_0}$ for at least one $i_0 \in \{1,\dots,k\}$.

Lemma 10.7. *Let $\ell \geqslant 1$ and let $\gtrsim$ be a quasi-precedence.*

(1) *The order $\succ_\ell$ is finitely branching; and*

(2) *The order $\succ_\ell$ is well-founded.*

Proof. The first assertion can be shown reasoning identical to Lemma 9.21(1). To prove the second assertion, let $\gtrsim^\circ$ be the extension of $\gtrsim$ to $\mathcal{F} \cup \{\circ\}$ so that the variadic list symbol $\circ$ is minimal in $\gtrsim^\circ$. Then as in Lemma 9.21(2), one can show

that $\succ_\ell \subseteq \succ^\circ_{\mathsf{rpo},\tau}$ for the recursive path order induced by the quasi-precedence $\succ^\circ$, and the status function τ that assigns to all $f \in \mathcal{F}$ a lexicographic, and to the sequence constructor $\circ$ a multiset status. This RPO is well-founded by Proposition 2.55. □

In correspondence to the functions $\mathsf{G}_{\mathcal{K},\ell}$ from Definition 9.22, we define functions H_ℓ that assigns to each term or sequence the length of its maximal $\succ_\ell$ descending sequence. By the previous lemma, this function is well-defined.

Definition 10.8. Let $\ell \geqslant 1$. We define $\mathsf{H}_\ell : \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V}) \to \mathbb{N}$ by

$$\mathsf{H}_\ell(a) := \max\{\ l \ \mid \exists a_1,\dots,a_l.\ a \succ_\ell a_1 \succ_\ell \cdots \succ_\ell a_l\} \ .$$

The following two lemmas are direct adaptions of Lemma 9.20 and Lemma 9.23 respectively, and can be proven identical to the corresponding lemmas from Section 9.1.1.

Lemma 10.9. *Let $\ell \geqslant 1$. The order $\succ_\ell$ satisfies the following properties:*

(1) $\succ_\ell \subseteq \succ_{\ell+1}$*; and*

(2) $\approx \cdot \succ_\ell \subseteq \succ_\ell$ *and* $\succ_\ell \cdot \approx \subseteq \succ_\ell$*; and*

(3) *for all $a, b, c_1, c_2 \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$,*

$$a \succ_\ell b \implies c_1 + a + c_2 \succ_\ell c_1 + b + c_2 \ .$$

Lemma 10.10. *Let $\ell \geqslant 1$. For all sequences $[t_1,\dots,t_k] \in \mathcal{T}^\star(\mathcal{F},\mathcal{V})$ we have*

$$\mathsf{H}_\ell([t_1,\dots,t_k]) = \sum_{i=1}^{k} \mathsf{H}_\ell(t_i) \ .$$

The central theorem of this section, established by Eguchi, binds the length of $\succ_\ell$ descending sequences appropriately.

Theorem 10.11 (Eguchi [7])**.** *Let $\ell \geqslant 1$. For each $f \in \mathcal{F}$, there exists a function $e_f(n) \in 2^{\mathcal{O}(n^d)}$ for some $d \in \mathbb{N}$ such that*

$$\mathsf{H}_\ell(f(t_1,\dots,t_k)) \leqslant e_f\big(\textstyle\sum_{i=1}^{k} \mathsf{dp}(t_i)\big) \ ,$$

for all terms $t_1,\dots,t_k \in \mathcal{T}(\mathcal{F},\mathcal{V})$.

Proof. To show the theorem, we show the stronger claim that for all $s = f(s_1,\dots,s_k)$ and $b \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\star(\mathcal{F},\mathcal{V})$ with $f(s_1,\dots,s_k) \succ_\ell b$,

$$\mathsf{H}_\ell(b) < (\ell+1)^{M^\ell \cdot \mathsf{rk}_\gtrsim(f) + \sum_{i=1}^{k} M^{\ell \dot{-} i}\, \mathsf{dp}(t_i)} \ ,$$

holds, where $M := \max\{\mathsf{dp}(s_i) \mid i = 1,\dots,k\} + 1$ and $n \dot{-} m := \max\{0, n-m\}$. The proof follows by induction on the lexicographic combination of $\mathsf{rk}_\gtrsim(f)$ and $\sum_{i=1}^{k} M^{\ell \dot{-} i}\, \mathsf{dp}(t_i)$. In the base case, $\mathsf{rk}_\gtrsim(f) = 1$ and $\sum_{i=1}^{k} M^{\ell \dot{-} i}\, \mathsf{dp}(t_i) = 0$. The latter implies that the arguments $s_1,\dots,s_k$ are variables or constants. A standard

induction refines $f(s_1,\dots,s_k) \succ_\ell b$ to $f(s_1,\dots,s_k) \succ_\ell^{\langle 3\rangle} [\,]$, using that in the considered case neither $\succ_\ell^{\langle 1\rangle}$ nor $\succ_\ell^{\langle 2\rangle}$ are applicable. Hence $\mathsf{H}_\ell(f(s_1,\dots,s_k)) = 1$ since $[\,]$ is minimal, the base case follows.

Consider the inductive step. We perform case analysis on the last rule that concludes $f(s_1,\dots,s_k) \succ_\ell b$.

- Case $f(s_1,\dots,s_k) \succ_\ell^{\langle 1\rangle} g(t_1,\dots,t_l)$: Define

$$N := \max\{\mathsf{dp}(t_j) \mid j = 1,\dots,l\} + 1 \,.$$

  Then $N \leqslant M$ follows from the order constraints $f(s_1,\dots,s_k) \rhd\!\!\!/_{\approx} t_j$ for $j = 1,\dots,l$. As therefore also $\sum_{j=1}^{l} N^{\ell \dot{-} j} \cdot \mathsf{dp}(t_j) < M^\ell$ we see

$$N^\ell \cdot \mathsf{rk}_\gtrsim(g) + \sum_{j=1}^{\ell} N^{\ell \dot{-} j} \cdot \mathsf{dp}(t_j) < M^\ell \cdot \mathsf{rk}_\gtrsim(g) + M^\ell \leqslant M^\ell \cdot \mathsf{rk}_\gtrsim(f) \,,$$

  where in the last inequality we use $\mathsf{rk}_\gtrsim(g) < \mathsf{rk}_\gtrsim(f)$ as given from the order constraint $f \succ g$. Hence we can even conclude

$$\mathsf{H}_\ell(g(t_1,\dots,t_l)) \leqslant (\ell+1)^{N^\ell \cdot \mathsf{rk}_\gtrsim(g) + \sum_{j=1}^{\ell} N^{\ell \dot{-} j} \cdot \mathsf{dp}(t_j)} < (\ell+1)^{M^\ell \cdot \mathsf{rk}_\gtrsim(f)} \,,$$

  by induction hypothesis, using again $f \succ g$.

- Case $f(s_1,\dots,s_k) \succ_\ell^{\langle 2\rangle} g(t_1,\dots,t_l)$: Define again

$$N := \max\{\mathsf{dp}(t_j) \mid j = 1,\dots,l\} + 1 \,.$$

  Observe that the order constraints on arguments give for all $j = 1,\dots,l$ some $i \in \{1,\dots,k\}$ such that $t_j \trianglelefteq\!\!\!/_{\approx} s_i$, hence $\mathsf{dp}(t_j) \leqslant \mathsf{dp}(s_i)$, and thus overall (i) $N \leqslant M$. Since $\mathsf{rk}_\gtrsim(f) = \mathsf{rk}_\gtrsim(g)$ in the considered case, we obtain (ii) $N^\ell \cdot \mathsf{rk}_\gtrsim(g) \leqslant M^\ell \cdot \mathsf{rk}_\gtrsim(f)$. Further, the order constraints on arguments give some $j \in \{1,\dots,\min(k,l)\}$ such that (iii) $\mathsf{dp}(t_i) = \mathsf{dp}(s_i)$ for all $i = 1,\dots,j-1$, and (iv) $\mathsf{dp}(t_j) < \mathsf{dp}(s_j)$ hold. Summing up (i)–(iv) we obtain

$$N^\ell \cdot \mathsf{rk}_\gtrsim(g) + \sum_{i=1}^{\ell} N^{\ell \dot{-} i} \cdot \mathsf{dp}(t_i) < M^\ell \cdot \mathsf{rk}_\gtrsim(f) + \sum_{i=1}^{\ell} M^{\ell \dot{-} i} \cdot \mathsf{dp}(s_i) \,.$$

  By induction hypothesis we thus get

$$\begin{aligned} \mathsf{H}_\ell(g(t_1,\dots,t_l)) &\leqslant (\ell+1)^{N^\ell \cdot \mathsf{rk}_\gtrsim(g) + \sum_{i=1}^{\ell} N^{\ell \dot{-} i} \cdot \mathsf{dp}(t_i)} \\ &< (\ell+1)^{M^\ell \cdot \mathsf{rk}_\gtrsim(f) + \sum_{i=1}^{\ell} M^{\ell \dot{-} i} \cdot \mathsf{dp}(s_i)} \,. \end{aligned}$$

- Case $f(s_1,\dots,s_k) \succ_\ell^{\langle 3\rangle} [t_1,\dots,t_l]$: In this case $f(s_1,\dots,s_k) \succ_\ell t_j$ for all $j = 1,\dots,l$, where these inequalities follow either by $\succ_\ell^{\langle 1\rangle}$ or $\succ_\ell^{\langle 2\rangle}$. As we have shown in the corresponding sub-cases, even

$$\mathsf{H}_\ell(t_j) \leqslant (\ell+1)^{M^\ell \cdot \mathsf{rk}_\gtrsim(f) + \sum_{i=1}^{\ell} M^{\ell \dot{-} i} \cdot \mathsf{dp}(s_i) - 1} \qquad \text{for all } j = 1,\dots,l,$$

holds. As in this case $l \leqslant \ell$, using this and Lemma 10.10 we obtain

$$\begin{aligned}\mathsf{H}_\ell([t_1,\dots,t_l]) &\leqslant \sum_{j=1}^{l} \mathsf{H}_\ell(t_j) \\ &\leqslant \ell \cdot (\ell+1)^{M^\ell \cdot \mathsf{rk}_\gtrsim(f) + \sum_{i=1}^{\ell} M^{\ell \dot{-} i} \cdot \mathsf{dp}(s_i) - 1} \\ &< (\ell+1)^{M^\ell \cdot \mathsf{rk}_\gtrsim(f) + \sum_{i=1}^{\ell} M^{\ell \dot{-} i} \cdot \mathsf{dp}(s_i)} \,.\end{aligned}$$

This completes the proof of the theorem. □

10.1.2. Predicative Embedding

Let $\mathcal{R}$ denote a predicative nested recursive constructor TRS. We now establish the predicative embedding of rewrite steps into the order $\succ_{\mathsf{epo}^\star}$, using the same interpretation $\mathcal{I}_\mathcal{R}$ that we have already employed in the soundness proof of sPOP*.

Recall that $\mathcal{N}_\mathcal{R}$ denotes the set of terms such that normal argument are normal forms of $\mathcal{R}$, compare Definition 9.28. The next lemma verifies that also for predicative nested recursive this set is closed under rewriting.

Lemma 10.12. *Let $\mathcal{R}$ denote a predicative nested recursive constructor TRS. If $s \in \mathcal{N}_\mathcal{R}$ and $s \xrightarrow{\mathsf{i}}_\mathcal{R} t$ then $t \in \mathcal{N}_\mathcal{R}$.*

Proof. Observe that, $f(s_1,\dots,s_k;s_{k+1},\dots,s_{k+l}) \succ_{\mathsf{epo}^\star} t$ implies that either t is a constructor term, or $t = g(t_1,\dots,t_m;t_{m+1},\dots,t_{m+n})$ where all normal arguments $t_1,\dots,t_m$ are constructor terms. This can be verified by case analysis on $\succ_{\mathsf{epo}^\star}$, where in the case $\succ^{\langle 1\rangle}_{\mathsf{epo}^\star}$ we employ that $s_i \succ_{\mathsf{epo}^\star} t$ and $s_i \in \mathcal{T}(\mathcal{C},\mathcal{V})$ implies that also $t \in \mathcal{T}(\mathcal{C},\mathcal{V})$. In particular for a substitution σ such that $f(s_1,\dots,s_k;s_{k+1},\dots,s_{k+l})\sigma$ is argument normalised, we have that $t_i\sigma \in \mathrm{NF}(\mathcal{R})$ for all normal arguments t_i of t. The lemma then follows by inductive reasoning as in the corresponding Lemma 9.29. □

Let $\gtrsim$ the precedence induced by $\mathcal{R}$. In the following, we write again $\sqsupseteq$ for the quasi-precedence on the normalised signature $\mathcal{F}_\mathsf{n}$ given in Definition 9.30. We denote by $\sqsupset$ the strict order $\sqsupseteq \setminus \sqsubseteq$, by $\sim$ the equivalence $\sqsupseteq \cap \sqsubseteq$ and by $\approx$ we denote the extension of $\sim$ to normalised terms $\mathcal{T}_\mathsf{n}(\mathcal{F},\mathcal{V})$.

The following auxiliary lemma provides the embedding of root-steps.

Lemma 10.13. *Let $\mathcal{R}$ be a TRS. Let $s = f(s_1,\dots,s_k;s_{k+1},\dots,s_{k+l}) \in \mathcal{T}(\mathcal{F},\mathcal{V})$ be a constructor based term and let $\sigma : \mathcal{V} \to \mathrm{NF}(\mathcal{R})$ be a normalising substitution. Then for all $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$,*

$$s \succ_{\mathsf{epo}^\star} t \implies \mathcal{I}_\mathcal{R}(s\sigma) \sqsupset_\ell \mathcal{I}_\mathcal{R}(t\sigma) \,,$$

where $\ell := |t|$.

Proof. Fix a constructor based term $s = f(s_1,\dots,s_k;s_{k+1},\dots,s_{k+l})$ let σ and let a normalising substitution with respect to the TRS $\mathcal{R}$. The proof follows the proof of corresponding Lemma 9.31 concerning sPOP*. We consider the

non-trivial case $t\sigma \notin \mathrm{NF}(\mathcal{R})$. This excludes the case $s \succ^{\langle 1\rangle}_{\mathsf{epo}^\star} t$, as then one can show that $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ holds, which contradicts $t\sigma \notin \mathrm{NF}(\mathcal{R})$. Hence either $s \succ^{\langle 2\rangle}_{\mathsf{epo}^\star} t$ or $s \succ^{\langle 3\rangle}_{\mathsf{epo}^\star} t$ holds. Thus $t = g(t_1, \dots, t_m; t_{m+1}, \dots, t_{m+n})$ for some $g \in \mathcal{F}$ and terms t_i $(i = 1, \dots, m+n)$, with

$$\mathcal{I}_\mathcal{R}(t\sigma) = [g_\mathsf{n}(t_1\sigma, \dots, t_m\sigma)] +\!\!+ \mathcal{I}_\mathcal{R}(t_{m+1}\sigma) +\!\!+ \cdots +\!\!+ \mathcal{I}_\mathcal{R}(t_{m+n}\sigma) \,.$$

We first show

$$s \succ_{\mathsf{spop}^\star} t \implies f_\mathsf{n}(s_1\sigma, \dots, s_k\sigma) \sqsupseteq_{\mathcal{K}_\mathsf{n}, |t|} u \text{ for all } u \in \mathcal{I}_\mathcal{R}(t\sigma) \,. \qquad (\dagger)$$

For the proof of the implication (†), suppose $s \succ_{\mathsf{spop}^\star} t$ holds. Consider first the element $u = g_\mathsf{n}(t_1\sigma, \dots, t_m\sigma) \in \mathcal{I}_\mathcal{R}(t\sigma)$. If $s \succ^{\langle 2\rangle}_{\mathsf{epo}^\star} t$ holds, we obtain

$$f_\mathsf{n}(s_1\sigma, \dots, s_k\sigma) \sqsupseteq^{\langle 1\rangle}_\ell g_\mathsf{n}(t_1\sigma, \dots, t_m\sigma) \,,$$

by reasoning identical to the corresponding case in Lemma 9.31. Hence suppose $s \succ^{\langle 3\rangle}_{\mathsf{epo}^\star} t$ holds. In this case $f \sim g$, and thus $f_\mathsf{n} \sim g_\mathsf{n}$, holds. Moreover, there exists $j \in \{1, \dots, \min(k, m)\}$ such that (i) $s_i \approx t_i$, hence $s_i\sigma \approx t_i\sigma$ for all $i = 1, \dots, j-1$, (ii) $s_j \rhd\!\!/_\approx t_j$ hence $s_j\sigma \rhd\!\!/_\approx t_j\sigma$, and (iii) $s \trianglerighteq\!\!\!\!/^{\mathsf{n}}_\approx t_i$, hence $s\sigma \trianglerighteq\!\!\!\!/^{\mathsf{n}}_\approx t_i\sigma$, holds for all $i = j+1, \dots, m$. Since $\trianglerighteq\!\!\!\!/^{\mathsf{n}}_\approx$ restricts the sub-term relation to normal arguments, (iii) translates to $f_\mathsf{n}(s_1\sigma, \dots, s_k\sigma) \rhd\!\!/_\approx t_i\sigma$ for all $i = j+1, \dots, m$. As trivially $m \leqslant |t|$, this establishes

$$f_\mathsf{n}(s_1\sigma, \dots, s_k\sigma) \sqsupseteq^{\langle 2\rangle}_\ell g_\mathsf{n}(t_1\sigma, \dots, t_m\sigma) \,.$$

This concludes the case $u = g_\mathsf{n}(t_1\sigma, \dots, t_m\sigma) \in \mathcal{I}_\mathcal{R}(t\sigma)$. For $u \in \mathcal{I}_\mathcal{R}(t\sigma)$ not of this shape, $u \in \mathcal{I}_\mathcal{R}(t_j\sigma)$ for some safe argument position of $j \in \{m+1, \dots, m+n\}$ of g. In this case we have $s \succ_{\mathsf{epo}^\star} t_j$ and the claim thus follows by induction hypothesis. We conclude (†).

Using that as observed in Lemma 9.31 the length of $\mathcal{I}_\mathcal{R}(t\sigma)$ is bounded by $|t|$, from the implication (†) we obtain $f_\mathsf{n}(s_1\sigma, \dots, s_k\sigma) \sqsupseteq^{\langle 3\rangle}_\ell \mathcal{I}_\mathcal{R}(t\sigma)$ under the assumption $s \succ_{\mathsf{epo}^\star} t$, and thus $\mathcal{I}_\mathcal{R}(s\sigma) \sqsupseteq^{\langle 4\rangle}_\ell \mathcal{I}_\mathcal{R}(t\sigma)$ holds as desired. □

Lemma 10.14. *Let $\mathcal{R}$ denote a predicative recursive constructor TRS and let $\ell := \max\{|r| \mid l \to r \in \mathcal{R}\}$. If $s \in \mathcal{N}_\mathcal{R}$ and $s \xrightarrow{\mathsf{i}}_\mathcal{R} t$ then $\mathcal{I}_\mathcal{R}(s) \sqsupseteq_{\mathcal{K}_\mathsf{n}, \ell} \mathcal{I}_\mathcal{R}(t)$.*

Proof. The proof follows in correspondence to Lemma 9.32, by induction on the rewrite position. The base case is handled by Lemma 10.13. The inductive step follows by reasoning identical to Lemma 9.32, replacing the application of Lemma 9.20(3) by Lemma 10.9(3). □

10.1.3. Putting Things Together

Proof of Theorem 10.3. Let $\mathcal{R}$ denote a predicative recursive TRS over the signature $\mathcal{F}$. Define $\ell := \max\{|r| \mid l \to r \in \mathcal{R}\}$. We prove that for every $f \in \mathcal{F}$, the innermost derivation height of $f(\vec{u}; \vec{v})$ for values $\vec{u}, \vec{v}$ is bounded by $e_f(n)$, where $n := \sum_{u_i \in \vec{u}} \mathsf{dp}(u_i)$ and $e_f(n) \in 2^{\mathcal{O}(n^d)}$ as given by Theorem 10.11.

Consider a derivation

$$f(\vec{u};\vec{v}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathrm{i}}_{\mathcal{R}} \cdots \xrightarrow{\mathrm{i}}_{\mathcal{R}} t_l \ .$$

Then Lemma 10.14 together with Lemma 10.12 translate this sequence to

$$[f_{\mathsf{n}}(\vec{u})] = \mathcal{I}_{\mathcal{R}}(f(\vec{u};\vec{v})) \sqsupset_\ell \mathcal{I}_{\mathcal{R}}(t_1) \sqsupset_\ell \cdots \sqsupset_\ell \mathcal{I}_{\mathcal{R}}(t_l) \ ,$$

hence

$$l \leqslant \mathsf{G}_{\mathcal{K},\ell}([f_{\mathsf{n}}(\vec{u})]) = \mathsf{G}_{\mathcal{K},\ell}(f_{\mathsf{n}}(\vec{u})) \leqslant e_f(n) \ ,$$

where the equality follows by Lemma 10.10, and the inequality by Theorem 10.11. □

Corollary 10.15 (Soundness)**.** *Let $\mathcal{R}$ be a confluent (or orthogonal) and predicative nested recursive constructor TRS. Then every function defined by $\mathcal{R}$ is computable in time $2^{\mathcal{O}(n^d)}$ for some $d \in \mathbb{N}$ on a deterministic Turing machine.*

10.2. Exponential Path Orders are Complete

We now show that EPO* is complete for **FEXP**. The pattern of the proof follows the completeness proof of small polynomial path orders. To this end we employ the term rewriting characterisation $\mathcal{R}_{\mathcal{N}}$ presented in [31] of Arai and Eguchi's class $\mathcal{N}$. We first pin down the notion of lexicographic descending arguments, employed in the safe nested recursion scheme, the schema $\mathcal{R}_{\mathcal{N}}$ is then introduced below in Definition 10.17.

Again we make use of constructors ϵ, s_0 and s_1 to encode binary words. Denote by Γ the alphabet $\{\epsilon, 0, 1\}$. Define the *type* $\tau(u) \in \Gamma$ of values $u \in \mathcal{T}(\{\epsilon, \mathsf{s}_0, \mathsf{s}_1\})$ such that $\tau(\epsilon) := \epsilon$, and $\tau(\mathsf{s}_i(u)) := i$ for $i = 0, 1$. We also use τ for its homomorphic extension to sequences of terms. Inversely, for type $w = i_1, \ldots, i_k \in \Gamma^k$ and terms $\vec{u} = u_1, \ldots, u_k$ we write $\mathsf{s}_w(\vec{u})$ for the sequence $\mathsf{s}_{i_1}(;u_1), \ldots, \mathsf{s}_{i_k}(;u_k)$, where $\mathsf{s}_\epsilon(u_i)$ denotes ϵ. In correspondence to the exemplified definition of the function f defined by safe nested recursion on notation on page 127, in the schema $\mathcal{R}_{\mathcal{N}}$ recursive parameters are determined by the status $\tau(u_1), \ldots, \tau(u_k)$ of the recursion arguments $u_1, \ldots, u_k$ only. This is formalised in the definition of $>^k_{\mathsf{lex}}$*-function.*

Definition 10.16 (Lexicographic Descending Argument,$>^{\boldsymbol{k}}_{\mathsf{lex}}$-function)**.** Consider the constructors $\mathcal{C} = \{\epsilon, \mathsf{s}_0, \mathsf{s}_1\}$.

(1) For constructor terms $u_1, \ldots, u_k \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ and $v_1, \ldots, v_k \in \mathcal{T}(\mathcal{C}, \mathcal{V})$, we define

$$u_1, \ldots, u_k >^k_{\mathsf{lex}} v_1, \ldots, v_k \ ,$$

if there exists $j \in \{1, \ldots, k\}$ such that for all $i = 1, \ldots, k$ the following conditions hold:

(i) $u_i = v_i$ if $i < j$;

(ii) $u_j = \mathsf{s}_0(v_j)$ or $u_j = \mathsf{s}_1(v_j)$; and

(iii) $u_{i'} = v_i$ if $i > j$ for some $i' \in \{1, \dots, k\}$.

(2) Consider a function

$$p : \{1, \dots, k\} \times \Gamma^k \to \{1, \dots, k\} \times \{\mathsf{id}, \mathsf{prec}\} \ .$$

Here prec and id denote the predecessor and identity function on values $\mathcal{T}(\{\epsilon, \mathsf{s}_0, \mathsf{s}_1\})$ respectively, where prec is defined such that $\mathsf{prec}(\epsilon) := \epsilon$ and otherwise $\mathsf{prec}(\mathsf{s}_i(; u)) := u$. Based on $(j, f) \in \{1, \dots, k\} \times \{\mathsf{id}, \mathsf{prec}\}$ define the function $\mathrm{J}^k_{(j,f)} : \mathcal{T}(\mathcal{C}, \mathcal{V})^k \to \mathcal{T}(\mathcal{C}, \mathcal{V})$ such that $\mathrm{J}^k_{(j,f)}(u_1, \dots, u_k) := f(u_j)$.

The function p is called a $>^k_{\mathsf{lex}}$-*function* if for all $\vec{u} := u_1, \dots, u_k \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ we have

$$\vec{u} \neq \vec{\varepsilon} \implies \vec{u} >^k_{\mathsf{lex}} \mathrm{J}^k_{p(1,\tau(\vec{u}))}(\vec{u}), \dots, \mathrm{J}^k_{p(k,\tau(\vec{u}))}(\vec{u}) \ .$$

Definition 10.17 (Term Rewriting Characterisation $\boldsymbol{\mathcal{R}_{\mathcal{N}}}$ [31])**.** For each $k, l \in \mathbb{N}$ the set of function symbols $\mathcal{F}^{k,l}_{\mathcal{N}}$ with k normal and l safe argument positions is the least set of function symbols such that

(1) $\epsilon \in \mathcal{F}^{0,0}_{\mathcal{N}}$, $\mathsf{s}_0, \mathsf{s}_1 \in \mathcal{F}^{0,1}_{\mathcal{N}}$ are the only constructors; and

(2) $\mathsf{S}_0, \mathsf{S}_1 \in \mathcal{F}^{0,1}_{\mathcal{N}}$, $\mathsf{P} \in \mathcal{F}^{0,1}_{\mathcal{N}}$, $\mathsf{C} \in \mathcal{F}^{0,4}_{\mathcal{N}}$, $\mathsf{O}^{k,l} \in \mathcal{F}^{k,l}_{\mathcal{N}}$ and for all $j = 1, \dots, k + l$, $\mathsf{I}^{k,l}_j \in \mathcal{F}^{k,l}_{\mathcal{N}}$; and

(3) if $\mathsf{h} \in \mathcal{F}^{m,n}_{\mathcal{N}}$, $\vec{\mathsf{h}} = \mathsf{h}_1, \dots, \mathsf{h}_n \in \mathcal{F}^{k,l}_{\mathcal{N}}$ and $1 \leqslant i_1 \leqslant \cdots \leqslant i_m \leqslant k$ then $\mathsf{WSC}[\mathsf{g}, i_1, \dots, i_m, \vec{\mathsf{h}}] \in \mathcal{F}^{k,l}_{\mathcal{N}}$; and

(4) For all $>^{k'}_{\mathsf{lex}}$-functions p_1, p_2, p_3, $\mathsf{g} \in \mathcal{F}^{k,l}_{\mathcal{N}}$ and $\mathsf{r}_w, \vec{\mathsf{s}}_w, \vec{\mathsf{t}}_w \in \mathcal{F}^{k'+k,l+1}_{\mathcal{N}}$ also

$$\mathsf{SNRN}_{p_1,p_2,p_3}\big[\mathsf{g}, \mathsf{r}_w, \vec{\mathsf{s}}_w, \vec{\mathsf{t}}_w \ (w \in \Gamma^{k'}_{\neq\vec{\epsilon}})\big] \in \mathcal{F}^{k'+k,l}_{\mathcal{N}} \ .$$

Here and below $\Gamma^{k'}_{\neq\vec{\epsilon}}$ is used to abbreviate $\Gamma^{k'} \setminus \{\epsilon, \dots, \epsilon\}$.

We set $\mathcal{F}_{\mathcal{N}} := \bigcup_{k,l\in\mathbb{N}} \mathcal{F}^{k,l}_{\mathcal{N}}$.

Let $\vec{x} := x_1, \dots, x_k$, $\vec{y} := y_1, \dots, y_l$ and $\vec{z} := z_1, \dots, z_{k'}$ denote pairwise distinct variables. The *schema of rewrite rules* $\mathcal{R}_{\mathcal{N}}$ is defined as the least set of rules such that:

(1) $\mathcal{R}_{\mathcal{N}}$ contains the following rules defining initial functions:

$$\begin{aligned}
\mathsf{S}_i(; x) &\to \mathsf{s}_i(; x) && \text{(for } i = 0, 1\text{)}\\
\mathsf{P}(; \epsilon) &\to \epsilon\\
\mathsf{P}(; xi) &\to x && \text{(for } i = 0, 1\text{)}\\
\mathsf{I}^{k,l}_j(\vec{x}; \vec{y}) &\to x_j && \text{(for all } j = 1, \dots, k\text{)}\\
\mathsf{I}^{k,l}_j(\vec{x}; \vec{y}) &\to y_{j-k} && \text{(for all } j = k+1, \dots, l+k\text{)}\\
\mathsf{C}(; \epsilon, y, z_1, z_2) &\to y\\
\mathsf{C}(; xi, y, z_1, z_2) &\to z_i && \text{(for i = 0, 1)}\\
\mathsf{O}(\vec{x}; \vec{y}) &\to \epsilon && .
\end{aligned}$$

(2) For $\mathsf{f} := \mathsf{WSC}[\mathsf{g}, i_1, \ldots, i_m, \vec{\mathsf{h}}] \in \mathcal{F}^{k,l}_{\mathcal{N}}$ the schema $\mathcal{R}_{\mathcal{N}}$ contains the rule

$$\mathsf{f}(\vec{x}; \vec{y}) \to \mathsf{g}(x_{i_1}, \ldots, x_{i_m}; \mathsf{h}_1(\vec{x}; \vec{y}), \ldots, \mathsf{h}_\mathsf{n}(\vec{x}; \vec{y})) \;,$$

(3) For $\mathsf{f} := \mathsf{SNRN}_{p_1,p_2,p_3}\big[\mathsf{g}, \mathsf{r}_w, \vec{\mathsf{s}}_w, \vec{\mathsf{t}}_w \;(w \in \Gamma^{k'}_{\neq\vec{\epsilon}})\big] \in \mathcal{F}^{k'+k,l}_{\mathcal{N}}$ the schema $\mathcal{R}_{\mathcal{N}}$ contains the rules

$$\begin{aligned}
\mathsf{f}(\vec{\epsilon}, \vec{x}; \vec{y}) &\to \mathsf{g}(\vec{x}; \vec{y}) \\
\mathsf{f}(\mathsf{s}_w(;\vec{z}), \vec{x}; \vec{y}) &\to \mathsf{r}_w(\vec{v}_{w,1}, \vec{x}; \vec{y}, \mathsf{f}(\vec{v}_{w,1}, \vec{x}; \vec{\mathsf{s}}_w(\vec{v}_{w,2}, \vec{x}; \vec{y}, a))) \qquad (w \in \Gamma^{k'}_{\neq\vec{\epsilon}}) \\
&\qquad \{a \mapsto \mathsf{f}(\vec{v}_{w,2}, \vec{x}; \vec{\mathsf{t}}_w(\vec{v}_{w,3}, \vec{x}; \vec{y}, \mathsf{f}(\vec{v}_{w,3}, \vec{x}; \vec{y})))\} \;,
\end{aligned}$$

where a is a fresh variable and $\vec{v}_{w,i} := \mathrm{J}_{p_i(1,w)}(\mathsf{s}_w(;\vec{z})), \ldots, \mathrm{J}_{p_i(k',w)}(\mathsf{s}_w(;\vec{z}))$ for $i = 1, 2, 3$.

Proposition 10.18 ([31])**.** *For each $f \in \mathbf{FEXP}$, there exists a finite restriction $\mathcal{R}_f \subsetneq \mathcal{R}_{\mathcal{N}}$ such that $\mathcal{R}_f$ computes f.*

Proof. The schema $\mathcal{R}_{\mathcal{N}}$ is obtained by introducing a rewrite rule for each defining equations in the class $\mathcal{N}$, which defines exactly the functions **FEXP**, compare [2]. Hence for $f \in \mathbf{FEXP}$, we can take as $\mathcal{R}_f \subsetneq \mathcal{R}_{\mathcal{BW}}$ the set of rewrite rules that correspond to the equations involved in the definition of f in **FEXP**. □

We arrive at the completeness result.

Theorem 10.19 (Completeness)**.** *For every $f \in \mathbf{FEXP}$ there exists a finite, orthogonal, and predicative nested recursive constructor TRS $\mathcal{R}_f$.*

Proof. Consider $f \in \mathbf{FEXP}$, and let $\mathcal{R}_f \subsetneq \mathcal{R}_{\mathcal{BW}}$ denote the TRS given by Proposition 10.18. It remains to verify that $\mathcal{R}_f$ is predicative nested recursive. For this, let $\succ_{\mathsf{epo}^\star}$ denote the exponential path order as induced by the (strict) precedence $\gtrsim$ underlying $\mathcal{R}_f$, and the safe mapping as indicated in the rules. We show $\mathcal{R}_f \subseteq \succ_{\mathsf{epo}^\star}$ by induction on the definition of f. As $\succ_{\mathsf{spop}^\star} \subseteq \succ_{\mathsf{epo}^\star}$, reusing the proof of Theorem 9.38 it suffices to consider the inductive step for the new case where f is defined by safe nested recursion on notation. For this case, consider

$$\mathsf{f} := \mathsf{SNRN}_{p_1,p_2,p_3}\big[\mathsf{g}, \mathsf{r}_w, \vec{\mathsf{s}}_w, \vec{\mathsf{t}}_w \;(w \in \Gamma^{k'}_{\neq\vec{\epsilon}})\big] \;,$$

with defining rewrite rules

$$\begin{aligned}
\mathsf{f}(\vec{\epsilon}, \vec{x}; \vec{y}) &\to \mathsf{g}(\vec{x}; \vec{y}) \\
\mathsf{f}(\mathsf{s}_w(;\vec{z}), \vec{x}; \vec{y}) &\to \mathsf{r}_w(\vec{v}_{w,1}, \vec{x}; \vec{y}, \mathsf{f}(\vec{v}_{w,1}, \vec{x}; \vec{\mathsf{s}}_w(\vec{v}_{w,2}, \vec{x}; \vec{y}, a))) \qquad (w \in \Gamma^{k'}_{\neq\vec{\epsilon}}) \\
&\qquad \{a \mapsto \mathsf{f}(\vec{v}_{w,2}, \vec{x}; \vec{\mathsf{t}}_w(\vec{v}_{w,3}, \vec{x}; \vec{y}, \mathsf{f}(\vec{v}_{w,3}, \vec{x}; \vec{y})))\} \;,
\end{aligned}$$

where $\vec{v}_{w,i} := \mathrm{J}_{p_i(1,w)}(\mathsf{s}_w(;\vec{z})), \ldots, \mathrm{J}_{p_i(k',w)}(\mathsf{s}_w(;\vec{z}))$ for $i = 1, 2, 3$. By induction hypothesis, $\mathcal{R}_f$ is predicative nested recursive if these rewrite rules are also oriented by $\succ_{\mathsf{epo}^\star}$. Orientation of the first rule follows by $\succ^{\langle 2\rangle}_{\mathsf{epo}^\star}$, to show that remaining rules are oriented, fix $w = w_1, \ldots, w_k \in \Gamma^{k'}_{\neq\vec{\epsilon}}$. Consider the recursive parameters $v_1, \ldots, v_k := \vec{v}_{w,3}$. As p_3 is a $>^{k'}_{\mathsf{lex}}$-function, it follows that

$$\mathsf{s}_w(;z) = \mathsf{s}_{w_1}(;z_1), \ldots, \mathsf{s}_{w_k}(;z_k) >^{k'}_{\mathsf{lex}} v_1, \ldots, v_k \;,$$

holds, thus there exists $j \in \{1, \ldots, k\}$ satisfying the following properties:

(1) $\mathsf{s}_{w_i}(;z_i) = v_i$, thus $\mathsf{s}_{w_i}(;z_i) \approx v_i$, if $i < j$;

(2) $\mathsf{s}_{w_j}(;z_j) = \mathsf{s}_{w_j}(v_j)$ for $w_j \in \{0,1\}$, thus $\mathsf{s}_{w_j}(;z_j) \rhd\!\!\!/_{\approx} v_j$;

(3) $\mathsf{s}_{w_{i'}}(;z_{i'}) = v_i$ and thus $\mathsf{f}(\mathsf{s}_w(;\vec{z}),\vec{x};\vec{y}) \rhd\!\!\!\!\!/_{\approx} v_i$, if $i > j$ for some $i' \in \{1,\dots,k\}$.

Using that also $\mathsf{f}(\mathsf{s}_w(;\vec{z}),\vec{x};\vec{y}) \rhd\!\!\!/_{\approx} x$ and $\mathsf{f}(\mathsf{s}_w(;\vec{z}),\vec{x};\vec{y}) \succ^{\langle 1\rangle}_{\mathsf{epo}^\star} y$ holds for all $x \in \vec{x}$ and $y \in \vec{y}$ respectively, we obtain

$$\mathsf{f}(\mathsf{s}_w(;\vec{z}),\vec{x};\vec{y}) \succ^{\langle 3\rangle}_{\mathsf{epo}^\star} \mathsf{f}(\vec{v}_w,\vec{x};\vec{y}) \;.$$

Consider now $\mathsf{t}_i \in \vec{\mathsf{t}}_w$, where by definition $\mathsf{f} \succ \mathsf{t}_i$. As by the properties (1)–(3) also $\mathsf{f}(\mathsf{s}_w(;\vec{z}),\vec{x};\vec{y}) \rhd\!\!\!/_{\approx} v_i$ $(i = 1,\dots,k)$ holds we conclude

$$\mathsf{f}(\mathsf{s}_w(;\vec{z}),\vec{x};\vec{y}) \succ^{\langle 3\rangle}_{\mathsf{epo}^\star} \mathsf{t}_i(\vec{v}_{w,3},\vec{x};\vec{y},\mathsf{f}(\vec{v}_{w,3},\vec{x};\vec{y})) \;.$$

Repeating these steps, using that p_1 and p_2 are $>^{k'}_{\mathsf{lex}}$-functions the rule can finally be oriented. As $w \in \Gamma^{k'}_{\neq\vec{\epsilon}}$ was arbitrary, this completes the proof. □

Corollary 10.20. *The following class of functions are equivalent:*

(1) *The class of functions computed by confluent (or orthogonal), and predicative nested recursive constructor TRS.*

(2) *The class of functions computable in time $e(n) \in 2^{\mathcal{O}(n^k)}$ $(k \in \mathbb{N})$ on a deterministic Turing machine.*

Proof. The correspondence holds by Theorem 10.15 and Theorem 10.19. □

Part III.

Automated Runtime Complexity Analysis

Chapter 11.

Introduction

Hofbauer and Lautemann [42] motivated the study of the derivational complexity mainly as a measure to quantify the strength of a termination technique. In the previous part we have taken a complementary view, viz, the introduction of termination technique suited for the complexity analysis of rewrite systems, with an emphasis to infer *feasible* bounds. This idea is not novel. The seminal paper by Bonfante et al. [23] gives an early account on *taming* a termination technique so that the induced complexity is polynomial. Since then, a wealth of techniques have been introduced specifically to establish polynomial complexity bounds.

Techniques range from *direct methods*, called also *base techniques* below, to *transformation techniques*. Polynomial path orders are instances of direct methods, once applied they establish directly a bound on the complexity of the analysed TRS. The *dependency pair method* [5] is a prominent instance of a transformation technique, and is used nowadays in the majority of the competitive termination provers. In particular its systematic study and the formulation of the *dependency pair framework* [76] significantly improved the ability to automatically prove termination in rewriting. In essence, this method models the call structure in a TRS as a set of syntactically restricted rewrite rules, the *dependency pairs*. Strong normalisation of these dependency pairs, relative to the original rewrite rules, certifies termination of the input system. It is well established that the DP method is unsuitable for complexity analysis, in the sense that the complexity of the obtained dependency pair system does not reflect the complexity of the analysed TRS [70, 60]. Hirokawa and Moser recover the situation with the introduction of *weak dependency pairs* [38, 40]. Weak dependency pairs can effectively be applied for polynomial runtime complexity analysis, both for full and innermost rewriting. Noschinski et al. [63] developed a variation of weak dependency pairs, called *dependency tuples*, that is in particular effective if one is interested in the innermost runtime complexity analysis.

These adaptions allow us to utilise a wealth of termination techniques that work on dependency pairs. For instance, *(safe) reduction pairs* [38, 40], *various rule transformations* [63], or *usable rules* [38] become available for the automated runtime complexity analysis of TRSs. Some very effective methods have been introduced specifically for complexity analysis in the context of dependency pairs. For instance, *path analysis* [39] decomposes the analysed rewrite relation into simpler ones, by treating paths through the *dependency graph* independently. *Knowledge propagation* [63] is another complexity technique relying on dependency graph analysis. This method allows one to propagate bounds for specific rules along the dependency graph. Besides these, various minor simplifications

are implemented in tools, mostly relying on *dependency graph analysis*.

Motivated not only by these theoretical advances, but also by the *annual international termination competition*[1] which features four dedicated complexity categories since 2008, we have designed a fully automated complexity analyser for term rewrite systems, the *Tyrolean Complexity Tool* (*TCT* for short). TCT is *open-source*, released under the *GNU Lesser General Public License* (*LGPL*) Version 3, and available from

`http://cl-informatik.uibk.ac.at/software/tct/` .

Central in TCT are the notions of *complexity problem* (*problem* for short) and *complexity processor* (*processor* for short). Here, a problem essentially consists of a (finite) representation of a rewrite relation together with a set of starting terms. A processor represents a complexity technique in our framework. It dictates how to transform a problem into sub-problems (if any), and how to relate the complexity of the obtained sub-problems to the complexity of the input problem. A *complexity proof* in our framework is then nothing more than the object obtained by repeated application of processors starting from a *canonical complexity problem*.

Given a rewrite system, our tool TCT is capable of searching for such a complexity proof without further assistance from the user. Noteworthy, TCT can analyse both runtime and derivational complexity of rewrite systems. It also features dedicated support for innermost rewriting. Our tool TCT makes use of a majority of the techniques introduced for polynomial complexity analysis, formulated as complexity processors in our framework. Besides this fully automatic mode, TCT can also operate in a semi-automatic mode, through the provided *interactive interface*.

Related Tools. Our tool is in development since 2008. It started out as an extension to the powerful termination prover T$_{\text{T}}$T$_2$ [49].[2] The current version 2.0 of TCT constitutes a complete re-implementation separate from T$_{\text{T}}$T$_2$. Nevertheless, version 2.0 inherits many of the design choices from the termination tool T$_{\text{T}}$T$_2$. In particular, the underlying notion of complexity problem can be conceived as an extension to the notion of termination problem found in T$_{\text{T}}$T$_2$. Noteworthy, also our notion of *proof search strategy* is rooted in the one of T$_{\text{T}}$T$_2$.

TCT is not the only tool that can investigate complexity properties of rewrite systems.

AProVE: The closed source termination prover AProVE[3] features powerful support for analysing the innermost runtime complexity of TRSs. In particular, dependency tuples were established by the AProVE team. In its current state, the tool AProVE lacks support both for full rewriting and derivational complexity.

[1] `http://www.termination-portal.org/wiki/Termination_Competition/`.
[2] Available from `http://cl-informatik.uibk.ac.at/software/ttt2/`.
[3] `http://aprove.informatik.rwth-aachen.de/`.

CaT: The open source complexity tool CaT[4] constitutes a tiny wrapper around the termination prover TTT2. In this implementation, termination techniques are suitably restricted so that feasible bounds on terminating TRSs can be inferred. The tool CaT features support for derivational and runtime complexity analysis. It does however not feature dedicated complexity techniques, like the aforementioned adaptions of dependency pairs. The theoretical basis [79] of CaT influenced this work significantly.

Matchbox/Poly: The tool Matchbox/Poly[5] is open source tool, and can verify polynomially bounded derivational complexity in a completely automatic way. Notably, Matchbox/Poly demonstrated first that the advanced automata techniques [77] for inferring polynomial bounds from matrix interpretation termination proofs can be implemented efficiently. This criterion forms in TCT maybe the most powerful base-technique currently.

RaML Hoffmann et al. [44, 45] provide an automatic multivariate amortised cost analysis of *Resource Aware ML* programs. This analysis exploits typing, and extends earlier results on amortised cost analysis. Notably it is parametric in the investigated resource, besides the number of execution steps it allows for instance the analysis of heap space usage.

Outline. This part collects the authors contributions of joint work Moser on TCT [15] and its underlying theoretical framework [14]. Besides a more elaborate treatment of the topic, we also integrate *argument filterings* into small polynomial path orders, which has not been covered before. The tool TCT was developed by Andreas Schnabl, Georg Moser and the author. This work covers the authors contributions on this tool. As an exception, we also briefly mention the implementations of the polynomial and matrix interpretation in Section 14.1, which are due to Schnabl. The first implementation of *dependency pairs* and *dependency graphs* (compare Section 14.4) can also be attributed to Schnabl.

In the next chapter we start with a very brief overview on the implementation of TCT, the main part of this chapter is concerned with the user interface of our tool. In Chapter 13 we then introduce the theoretical framework underlying TCT. Focusing on (innermost) runtime complexity analysis, in Chapter 14 we then establish a unified account on a majority of the techniques implemented in TCT, by formalising these as processors in our framework. On the one hand this account covers previously established techniques, notably complexity pairs as studied in [79] and safe reduction pairs [38] respective (Section 14.1), the relative combination technique for such orders as initially proposed in [79] (Section 14.2), weak dependency pairs from [38] and dependency tuples from [63] (Section 14.4). On the other hand, we also study novel techniques, viz, aforementioned combination of argument filterings with small polynomial path orders (Section 14.3 and Section 14.7), various simplification techniques (Section 14.5) and *dependency graph decomposition* (Section 14.5).

We conclude this part in Chapter 15 with our experimental evaluation of TCT.

[4] Available from `http://cl-informatik.uibk.ac.at/software/cat/`.
[5] Available from `http://dfa.imn.htwk-leipzig.de/matchbox/poly/`.

Chapter 12.

The Tyrolean Complexity Tool

Our tool is implemented in the strongly typed, lazy functional programming language Haskell[1] and compiles on the *Glasgow Haskell Compiler*[2] on GNU/Linux.

In its current form, TCT features 23 techniques for runtime and/or derivational complexity analysis. The sources consist of about 13,000 lines of code, and additionally 4,000 lines of documentation. Out of the 73 modules, 43 modules are dedicated to the implementation of the various techniques (roughly 56 % of the code), the remaining modules provide the core of TCT and utilities.

We have developed the following Haskell libraries which are used by TCT. These are separately available from the TCT homepage.[3]

- qlogic provides facilities for dealing with propositional logic, and consists of approximately 3,100 lines of code. Notably it defines an interface to SAT-solvers, including routines to efficiently translate Boolean formulas to conjunctive normal form. Also, it features support for theories over natural numbers and integers, implemented by *bit-blasting*.

- termlib provides term rewriting functionality, and consists of about 2,100 lines of code.

- parfold is a small library that provides folding capabilities over lists of concurrently evaluated monad actions, a simple but convenient abstraction to concurrent programming.

In the following, we provide a walk-through on the various user interfaces of TCT.

12.1. Web Interface

Our *web interface*, accessible from the TCT homepage, provides a convenient way to use TCT without the necessity to install the software. The interface is aimed for simplicity, compare Figure 12.1. For the curious user that wants to play around with TCT, we also provide a wealth of examples. The web interface is configured so that by default an upper bound on the *runtime complexity* of the given rewrite system is estimated. This behaviour can be modified under

[1] An open-source product of more than twenty years of cutting-edge research, available `http://haskell.org/`.

[2] The compiler is open-source and available from `http://www.haskell.org/ghc/`.

[3] `http://cl-informatik.uibk.ac.at/projects/`.

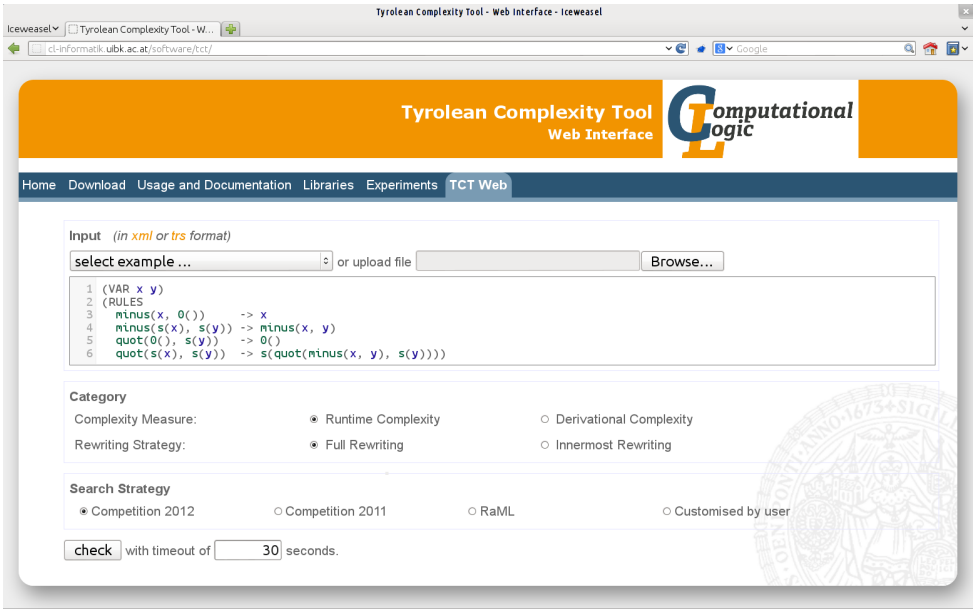


Figure 12.1.: Web Interface of TCT.

category, where the user can pick from the four different complexity measures TCT currently supports. On success, an estimated upper bound on the complexity function is presented to the user, together with a proof script that explains in considerable detail how the certificate was obtained.

To find a proof in a reasonable amount of time, the different techniques implemented in TCT need to be combined wisely. This combination depends on the one hand on the input problem. On the other hand, the combination depends also on the available hardware. In TCT, *proof search* is not hard-wired, instead it is guided by a *(proof) search strategy*. The interface allows to specify such a search strategy from a set of pre-defined proof search strategies. Besides the search strategies employed in recent competitions, the web-interface currently offers the search strategy *RaML*. This search strategy is specifically designed for functional programs given as rewrite systems. Further, a customisable search strategy allows the explicit inclusion/exclusion of methods.

12.2. Command-Line Interface

The full power of TCT is available through its command-line interface. For installation instructions we refer the reader to the homepage. Here we want to briefly outline usage and customisation, comprehensive documentation can be found online at

`http://cl-informatik.uibk.ac.at/software/tct/docs.html`.

TCT is run from the command line prompt by typing

```
$ tct [options] [-s <strategy>] <file>.
```

Here `[options]` specify an optional list of command-line options, `<strategy>` specifies optionally a proof search strategy, and `<file>` the *input file*. The input file must adhere either the old *TPDB format*[4] or the new *XTC Format*[5]. A complete list of options can be obtained by typing `tct --help`. Notable, the command line switch `-a` allows to overwrite strategy and start terms, and the switch `-t` allows to set a timeout.

12.2.1. Proof Search Strategy Format

The proof search strategy supplied with the `-s` switch is given as an *S-expression* of the form

`(<name> [:<argname> <arg>]* [<arg>]*)`,

where outermost parentheses can be dropped. Here `<name>` refers to the name of a proof technique, the list `[:<argname> <arg>]*` can be used to specify *named optional arguments*, and the list `[<arg>]*` gives a possibly empty sequence of *positional arguments*. As an example,

`best (matrix :degree 2) (timeout 3 (bounds :enrichment match))`,

provides a valid proof search strategy in TCT. Here `best` is used to run one or more search strategies in parallel, in this case the search strategies `matrix :degree 2` and `timeout 3 (bounds :enrichment match)`. The input is solved by the search strategy which produces the tightest upper bound. In total, the search strategy defined above advises TCT to check for compatibility with a *matrix interpretation* [32] that induces a quadratic upper bound, and for three seconds for *match-boundedness* [34] of the input problem. Every processor implemented in TCT, like the processors `bound` and `matrix` used above, qualifies as a search strategy here. Beside these, various *combinators* are available. In its current form, TCT provides the following combinators.

- `best <strategy> ··· <strategy>`: This combinator runs the given strategies in parallel. It proves the given problem with whichever strategy supplies the least upper bound.

- `fastest <strategy> ··· <strategy>`: This combinator runs the given strategies in parallel, and provides the proof of whichever strategy succeeds first.

- `sequentially <strategy> ··· <strategy>`: This combinator runs the given strategies sequentially, until the first strategy succeeds in solving the problem.

- `timeout <secs> <strategy>`: This search strategy behaves like the given strategy, but aborts the computation in case a proof could not be found after the given number of seconds.

[4] `http://www.lri.fr/~marche/tpdb/format.html`.
[5] `http://www.termination-portal.org/wiki/XTC_Format_Specification`.

- `ite <strategy> <strategy> <strategy>`: This combinator implements conditional branching in the expected way. We have implemented a variety of processors that check for basic properties that can be used as guard with this combinator. For instance, the search strategy `ite orthogonal` $\mathtt{s}_t$ $\mathtt{s}_e$ proceeds according to the strategy $\mathtt{s}_t$ if the problem is orthogonal, otherwise it proceeds as determined by $\mathtt{s}_e$.

- `success`: This strategy trivially succeeds.

- `fail`: This strategy always fails.

12.2.2. Configuration

Besides basic options given on the command-line, TCT can be configured by modifying the *configuration file*, which resides in `~/.tct/tct.hs` by default. This Haskell source-file defines the actual binary that is run each time TCT is called. Thus the full expressiveness of Haskell is available. As a downside, it requires also a working Haskell environment. The minimal configuration shown in Figure 12.2 is generated on the first run of TCT.

```
import Tct (tct)
import Tct.Configuration
import Tct.Interactive
import Tct.Instances
import qualified Termlib.Repl as TR

main :: IO ()
main = tct config

config :: Config
config = defaultConfig
```

Figure 12.2.: Initial Configuration of TCT.

This initial configuration consists of a set of convenient imports and the IO action `main` together with a *configuration record* `config`. The configuration record passed in `main` allows one to overwrite various flags of TCT.

Most importantly, through the field `strategies`, the configuration record allows the modification of the list of proof search strategies that can be employed. In Figure 12.3 we depict a modified configuration that defines two new search strategies, called `matrices` and `withDP`. Strategies are added by overwriting the field `strategies` with a list of declarations of the form

```
<code> ::: strategy "<name>" [<parameters-declaration>] .
```

Here `<code>` refers to a definition that evaluates to a search strategy. The string `"`*`<name>`*`"` together with the optional parameters-declaration specify how this code is accessible from the command-line. For instance, the first declaration in Figure 12.3 defines a new search strategy named *matrices*, which is available by supplying the option `-s "matrices [:start <nat>] <nat>"` to the TCT executable. Here the parameters to `matrices` are declared by

```
import Tct (tct)
import Tct.Instances
✂ ..........................................................................................
main :: IO ()
main = tct config

config :: Config
config = defaultConfig { strategies = strategies }
  where
    strategies =
      [ matrices ::: strategy "matrices" ( optional naturalArg "start" (Nat 1)
                                           :+: naturalArg )
      , withDP   ::: strategy "withDP" ]

matrices (Nat s :+: Nat n) =
  fastest [ matrix `withDimension` d `withBits` bitsForDimension d
          | d <- [s..s+n] ]
    where
      bitsForDimension d
        | d < 3 = 2
        | otherwise = 1

withDP =
  (timeout 5 dps <> dts)
  >>> try (exhaustively decomposeIndependentSG)
  >>> try cleanTail
  >>> try usableRules
  where
    dps = dependencyPairs >>> try usableRules >>> wgOnUsable
    dts = dependencyTuples
    wgOnUsable = weightgap `withDimension` 1 `wgOn` WgOnTrs
```

Figure 12.3.: Configuration defining search strategies `matrices` and `withDP`.

```
optional naturalArg "start" (Nat 1) :+: naturalArg ,
```

where the infix operator `:+:` is used to specify sequences of parameters. As indicated by the constructor `naturalArg`, the search strategy `matrices` expect two *natural numbers* as arguments. In contrast to the second parameter, the first is optional and defaults to the natural number `1`.

In Figure 12.3, these parameters are provided to the code of `matrices`. Using parameters s and n as supplied on the command-line, the code evaluates to a proof search strategy that searches for n compatible matrix interpretations of increasing dimension starting with dimension s, in parallel. Both `matrix` and `fastest`, along with other processors, combinators and *modifiers* like `withDimension` and `withBits`, are exported by the module `Tct.Instances`.

The second proof search strategy declared in Figure 12.3 defines a *transformation* called `withDP`. Transformations are search strategies that *generate* from the given input problem a possibly empty set of *sub-problems*, in a complexity-preserving manner. For every transformation t and search strategy[6] s, one can use the search strategy t `>>|` s which first applies transformation t and then solves the resulting sub-problems in accordance to s. Search strategy declarations

[6] Prior to Version 1.7, TcT only featured methods that yield a closed complexity proof, and these methods were simply called *processor*. To avoid confusion, in the present work we use the more adequate term search strategy for this notion of processor. What is referred to as transformation here, reflects the notion of processor as outlined in the introduction.

perform such a lifting of transformation implicitly, the declaration of `withDP` for instance results in a search strategy available as `withDP <strategy>`. Besides the combinator `>>|` and its variation `>>||`, where the given search strategy s is applied in parallel on all sub-problems, the module `Tct.Instances` provides a wealth of *transformation combinators*. We briefly discuss basic combinators, a full list of transformation combinators is available in the online documentation.

- t_1 `<>` t_2: This combinator, employed also in `withDP`, first applies transformation t_1, only if this is unsuccessful it applies transformation t_2 on the input problem instead.

- t_1 `<||>` t_2: This is a variation of the above combinator which applies transformations t_1 and t_2 in parallel, resulting in the sub-problems of whichever transformation succeeds first. The combinator `<||>` thus implements a form of non-deterministic choice.

- t_1 `>>>` t_2: The combinator `>>>` defines composition of transformations, in the sense that the transformation t_1 `>>>` t_2 first applies transformation t_1 and then transformation t_2 on all resulting sub-problems.

- `successive [`t_1`, ..., `t_n`]`: This implements a list-version of `>>>`.

- `try` t: Any transformation aborts if it is inapplicable. The combinator `try` overrides this behaviour, in the sense that `try` t behaves exactly like t should t succeed, otherwise it behaves as an identity.

- `force` t: The combinator `force` is dual to `try` and requires the given transformation t to abort when t is inapplicable.

- `exhaustively` t: The combinator `exhaustively`, defined by

  ```
  exhaustively t = t >>> try (exhaustively t),
  ```

  applies t in an iterated fashion.

- `withProblem` f: This combinator passes the currently analysed problem to the function f which evaluates to a transformation.

- `when` b t: This combinator only applies transformation t if the given Boolean is `True`.

In total, the search strategy `withDP` defined in Figure 12.3 applies weak dependency pairs as realised in the definition of `dps`, or dependency tuples as realised by `dts` should the former fail. This transformation is followed by a sequence of syntactic simplifications, if applicable. We remark the thoughtful use of `try`. The transformation `dps` fails if the *weight gap principle* cannot be established on all TRS rules, i.e., rules that are not dependency pairs. The latter is implemented by the transformation `wgOnUsable`, and constitutes an implementation of [40, Theorem 6.5]. We finally point out that an extended version of the transformation `withDP` is available in TCT as `toDP`.

12.3. Interactive Interface

TCT features also an *interactive interface*, TCT-i for short. In this section we guide the reader through a small interactive session that outlines the main features, elaborate documentation of this mode is again provided online. This *semi-automatic* mode is in particular useful when investigating into tight(er) complexity bounds, and to crack hard-to-solve problems.

The interactive interface constitutes essentially of a tiny wrapper around ghci, the interpreter bundled with the *Glasgow Haskell compiler*. Users familiar with ghci will note that all features available in ghci are also available in TCT-i. The interactive interface is started from the command-line by supplying the option `-i` to the TCT executable.

TCT-interactive 12.1

```
$ tct -i
  GHCi, version 7.4.1: http://www.haskell.org/ghc/  :? for help
✂.....................................................................................
  This is version 2.0 of the Tyrolean Complexity Tool.

  (c) Martin Avanzini <martin.avanzini@uibk.ac.at>,
      Georg Moser <georg.moser@uibk.ac.at>, and
      Andreas Schnabl <andreas.schnabl@uibk.ac.at>.

  This software is licensed under the GNU Lesser General Public
  License, see <http://www.gnu.org/licenses/>.

  Don't know how to start? Type 'help'.
TCT>
```

The interactive interface maintains a *proof state*, which consists conceptually of a list of *open problems* together with proof information. The command `load` *"<file>"* is used to populate the proof state by the TRS given as argument.

TCT-interactive 12.2 (Continued from Session 12.1)

```
TCT> load "examples/div.trs"

  Current Proof State ------------------------------------------------

  Selected Open Problems:
  -----------------------
    Strict Trs:
      { -(x, 0()) -> x
      , -(s(x), s(y)) -> -(x, y)
      , %(0(), s(y)) -> 0()
      , %(s(x), s(y)) -> s(%(-(x, y), s(y))) }
    StartTerms: basic terms
    Strategy: none

  ----------------------------------------------------------------------
TCT>
```

The current state can be inspected at any time by typing the command `state`. We note that the rewrite strategy and set of start terms are defined in accordance to the input file. The commands `set[DC|RC|IDC|IRC]` provide short-hands to these accordingly. Alternatively, one can use `load[DC|RC|IDC|IRC]` *"<file>"*, which override the set rewrite strategy and set of starting terms accordingly.

The primary means to modify the proof state is the use of the command `apply`. This command takes a single argument, a proof search strategy, which is applied by default on all open problems. The implementations of the various methods and combinators, as exported by `Tct.Instances`, qualify as arguments to `apply`. Since TCT-i loads the configuration file of TCT, all search strategies declared in the configuration file are available as top-level bindings, and can thus be used in conjunction with `apply`. We can use the previously developed search strategy `withDP` to simplify the loaded problem as follows.

TCT-interactive 12.3 (Continued from Session 12.2)

```
TCT> apply withDP

    Problems simplified. Use 'state' to see the current proof state.

TCT>
```

The output of `apply` is intentionally kept short.[7] By typing `state` one can observe that our initially loaded complexity problem has been replaced by the problem obtained by our transformation `withDP`. To see the proof generated so far, one can use the command `proof`. Note that as long as the list of open problems is not empty, this proof is marked as open.

TCT-interactive 12.4 (Continued from Session 12.3)

```
TCT> proof
  1) dp [OPEN]:
  -------------
      We consider the following problem:
        Strict Trs:
          { -(x, 0()) -> x
          , -(s(x), s(y)) -> -(x, y)
          , %(0(), s(y)) -> 0()
          , %(s(x), s(y)) -> s(%(-(x, y), s(y))) }
        StartTerms: basic terms
        Strategy: none

      We add the following weak dependency pairs:
      Strict DPs:

        { -^#(x, 0()) -> c_1()
        , -^#(s(x), s(y)) -> c_2(-^#(x, y))
✂.....................................................................................
      1.1) Open Problem [OPEN]:
      -------------------------

        We consider the following problem:
          Strict DPs:
            { -^#(x, 0()) -> c_1(x)
            , -^#(s(x), s(y)) -> c_2(-^#(x, y))
            , %^#(s(x), s(y)) -> c_4(%^#(-(x, y), s(y))) }
          Weak Trs:
            { -(x, 0()) -> x
            , -(s(x), s(y)) -> -(x, y) }
          StartTerms: basic terms
          Strategy: none
TCT>
```

[7] To override this behaviour and see actions performed, one can use the command `setShowProofs`, or alternatively set the field `interactiveShowProofs` to `True` in the configuration record of TCT.

Once the list of open problem is empty, a complexity bound on the input problem has been successfully established. We can do so on our running example, using the matrix processor that we have already used before.

TCT-interactive 12.5 (Continued from Session 12.4)

```
TCT> apply matrix

    Hurray, the problem was solved with certificate YES(O(1),O(n^2)).
    Use 'proof' to show the complete proof.

TCT>
```

Application of the processor results in a closed proof. This is indicated by the string `YES(O(1),O(n^2))`, which follows the convention of the complexity competition. Here `YES` indicates that the proof was in principle successful. The left component `O(1)` gives an asymptotic lower-bound. The right component `O(n^2)` states the computed asymptotic upper bound, which is quadratic in our example. The produced proof thus verifies that our initial problem, loaded in Session 12.2, has at most quadratic runtime complexity. We remark that the runtime complexity of the input TRS is even linear. Inspecting the proof we see that the imprecision in the certificate was introduced in the last proof step. Fortunately TCT-i provides a command `undo` that can be used to revert the effect of `apply`. In fact, it reverts any modification on the proof state, except of course the effect of `undo` itself. We refine the proof by restricting the *induced degree* of the constructed interpretation.

TCT-interactive 12.6 (Continued from Session 12.5)

```
TCT> undo
    Current Proof State ------------------------------------------------

    Selected Open Problems:
    -----------------------
      Strict DPs:
        { -^#(x, O()) -> c_1(x)
        , -^#(s(x), s(y)) -> c_2(-^#(x, y))
        , %^#(s(x), s(y)) -> c_4(%^#(-(x, y), s(y))) }
      Weak Trs:
        { -(x, O()) -> x
        , -(s(x), s(y)) -> -(x, y) }
      StartTerms: basic terms
      Strategy: none

    ------------------------------------------------------------------------

TCT> apply $ matrix `withDegree` Just 1

    Hurray, the problem was solved with certificate YES(O(1),O(n^1)).
    Use 'proof' to show the complete proof.

TCT>
```

Here the function `withDegree` is used to modify the default parameters as defined in `matrix`, compare Section 14.1.[8] We finally end up with a closed proof that

[8]The application operator `$` has low, right-associative binding precedence. The backticks syntax allows to write a binary function in infix notation.

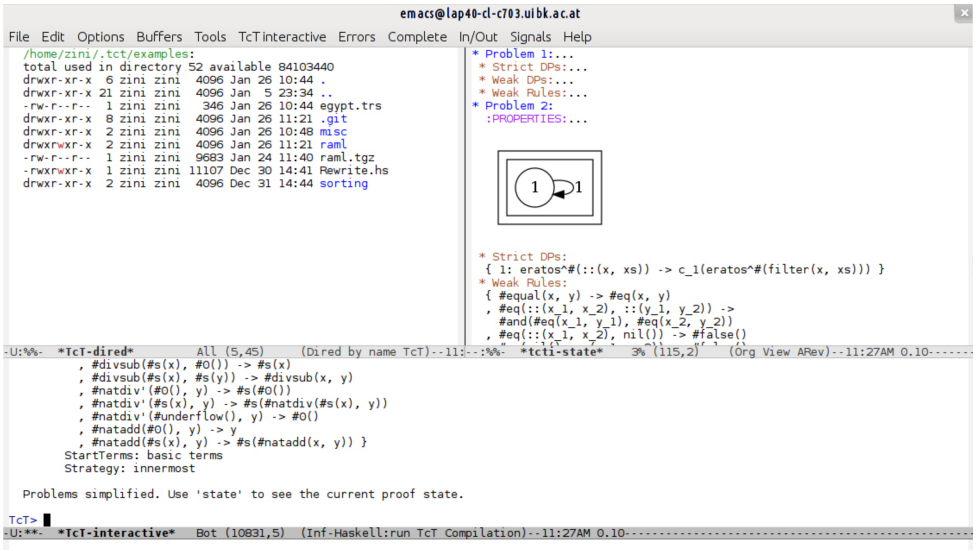


Figure 12.4.: TCT Major Mode for GNU Emacs.

verifies that our loaded TRS has linear runtime complexity. Using the command `writeProof` *"<file>"* one can write the constructed proof to the given file.

This completes the short tutorial. We remark that for the GNU Emacs[9] enthusiast, we have also crafted a small major-mode for TCT-i. This mode is available in the source distribution of TCT. The mode can be started by typing `M-x tct` into GNU Emacs. In addition to the features explained above, the major-mode provides a refurbished view on the proof state, compare Figure 12.4 which shows an example session. The graph depicted by TCT here is visualised using the dot tool of the *Graphviz* toolkit.[10]

[9] GNU Emacs is open-source and available from `http://www.gnu.org/s/emacs/`.
[10] The toolkit is open-source and available from `http://www.graphviz.org/`.

Chapter 13.

The Combination Framework Underlying TCT

In this chapter we introduce the *complexity framework* underlying TCT. The notions introduced here are greatly influenced by the work of Thiemann [76] on the dependency pair framework for termination analysis.

At the heart of our framework lies the notion of *complexity processor*, or simply *processor*. Recall that in its general form, a complexity processor dictates how to transform the analysed input *problem* into (hopefully) simpler sub-problems. It also relates the complexity of the obtained sub-problems to the complexity of the input problem. In our framework, such a processor is modeled as a set of inference rules

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

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

Conceptually, a complexity problem $\mathcal{P}$ consists of a set of *starting terms* $\mathcal{T}$ together with a relation $\to_{\mathcal{S} \cup \mathcal{W}}$ for TRSs $\mathcal{S}, \mathcal{W}$. The complexity function $\mathsf{cp}_{\mathcal{P}} : \mathbb{N} \to \mathbb{N}$ of $\mathcal{P}$ accounts for the number of applications of rules from $\mathcal{S}$ in derivations starting from terms $t \in \mathcal{T}$, measured in the size of t. To model innermost rewriting in our setting, we resort to the notion of $\mathcal{Q}$-restricted rewriting [76].

Definition 13.1 ($\mathcal{Q}$-restricted Rewrite Relation, Relative Rewriting)**.**

(1) Let $\mathcal{R}$ and $\mathcal{Q}$ be two TRSs. We define the *$\mathcal{Q}$-restricted rewrite relation* $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ such that $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ if there exists a context C, substitution σ, and rule $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$ such that $s = C[f(l_1\sigma, \ldots, l_n\sigma)]$, $t = C[r\sigma]$ and all arguments $l_i\sigma$ $(i = 1, \ldots, n)$ are $\mathcal{Q}$ normal forms.

(2) We extend, for two TRSs $\mathcal{S}$ and $\mathcal{W}$, the notion of $\mathcal{Q}$-restricted rewrite relation to a relative setting by defining

$$\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} := \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{W}} \cdot \xrightarrow{\mathcal{Q}}_{\mathcal{S}} \cdot \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{W}} \cdot$$

We call $\xrightarrow{\mathcal{Q}}_{\mathcal{R}/\mathcal{S}}$ the *$\mathcal{Q}$-restricted rewrite relation of $\mathcal{S}$ relative to $\mathcal{W}$*.

Note that for $\mathcal{Q} = \varnothing$, the relation $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ amounts to the rewrite relation of $\mathcal{R}$. For $\mathcal{Q} = \mathcal{R}$ we obtain the innermost rewrite relation of $\mathcal{R}$.

Definition 13.2 (Complexity Problem, Complexity Function)**.**

(1) A *complexity problem* $\mathcal{P}$ (*problem* for short) is a quadruple $(\mathcal{S}, \mathcal{W}, \mathcal{Q}, \mathcal{T})$, in notation $\langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$, where $\mathcal{S}, \mathcal{W}, \mathcal{Q}$ are TRSs and $\mathcal{T} \subseteq \mathcal{T}(\mathcal{F})$ a set of ground terms.

We call $\mathcal{S}$ and $\mathcal{W}$ the *strict* and *weak component* of $\mathcal{P}$ respectively. The set $\mathcal{T}$ is called the set of *starting terms* of $\mathcal{P}$.

(2) The *complexity (function)* $\mathsf{cp}_{\mathcal{P}} : \mathbb{N} \to \mathbb{N}$ of $\mathcal{P}$ is defined as the partial function

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

Consider a problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$. In the sequel $\mathcal{P}$, possibly followed by subscripts, always denotes a complexity problem. We always use $\mathcal{S}$ and $\mathcal{W}$, possibly followed by subscripts, for the strict and weak component of a complexity problem, whereas $\mathcal{R}$ refers to a set of rewrite rules that can occur in both components. We write $l \to r \in \mathcal{P}$ for $l \to r \in \mathcal{S} \cup \mathcal{W}$.

Definition 13.3 ($\mathcal{P}$-derivation)**.** Consider a problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$. The *rewrite relation of* $\mathcal{P}$ is defined as

$$\to_{\mathcal{P}} := \xrightarrow{\mathcal{Q}}_{\mathcal{S} \cup \mathcal{W}} \,.$$

A derivation $t \to_{\mathcal{P}} t_1 \to_{\mathcal{P}} \cdots$ is also called a *$\mathcal{P}$-derivation* (*starting from* t).

In this work we are mostly concerned with runtime complexity analysis, i.e., we consider basic starting terms.

Definition 13.4 (Runtime, Innermost Complexity Problem)**.** Consider a problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$.

(1) Then $\mathcal{P}$ is called a *runtime complexity problem* if $\mathcal{T} \subseteq \mathcal{T}_{\mathsf{b}}(\mathcal{D} \uplus \mathcal{C})$ holds. Otherwise it is called a *derivational complexity problem.*

For a runtime complexity problems $\mathcal{P}$ we suppose that the rewrite systems $\mathcal{S}$ and $\mathcal{W}$ do not define the constructors $\mathcal{C}$, i.e., defined symbols of $\mathcal{S} \cup \mathcal{W}$ are disjoint with $\mathcal{C}$.

(2) The problem $\mathcal{P}$ is called an *innermost complexity problem* if $\mathrm{NF}(\mathcal{Q}) \subseteq \mathrm{NF}(\mathcal{S} \cup \mathcal{W})$,

Note that for an innermost complexity problem $\mathcal{P}$ as above, the rewrite relation $\to_{\mathcal{P}}$ is included in the innermost rewrite relation of $\mathcal{R} \cup \mathcal{S}$.

Definition 13.5 (Canonical Complexity Problems)**.** Let $\mathcal{R}$ denote a TRS.

(1) $\mathcal{P}^{\mathrm{dc}}_{\mathcal{R}} := \langle \mathcal{R}/\varnothing, \varnothing, \mathcal{T}(\mathcal{F}) \rangle$ is called the *canonical derivational complexity problem* for $\mathcal{R}$; and

(2) $\mathcal{P}^{\mathrm{dci}}_{\mathcal{R}} := \langle \mathcal{R}/\varnothing, \mathcal{R}, \mathcal{T}(\mathcal{F}) \rangle$ is called the *canonical innermost derivational complexity problem* for $\mathcal{R}$; and

(3) $\mathcal{P}^{\mathrm{rc}}_{\mathcal{R}} := \langle \mathcal{R}/\varnothing, \varnothing, \mathcal{T}_{\mathsf{b}}(\mathcal{D} \uplus \mathcal{C})\rangle$ is called the *canonical runtime complexity problem*; and

(4) $\mathcal{P}^{\mathrm{rci}}_{\mathcal{R}} := \langle \mathcal{R}/\varnothing, \mathcal{R}, \mathcal{T}_{\mathsf{b}}(\mathcal{D} \uplus \mathcal{C})\rangle$ is called the *canonical innermost runtime complexity problem* for $\mathcal{R}$.

These notions are derived from the following trivial observation.

Lemma 13.6. *Let $\mathcal{R}$ denote a TRS. Then*

(1) $\mathrm{dc}_{\mathcal{R}}(n) =_{\mathsf{k}} \mathsf{cp}_{\mathcal{P}^{\mathrm{dc}}_{\mathcal{R}}}(n)$*; and*

(2) $\mathrm{dci}_{\mathcal{R}}(n) =_{\mathsf{k}} \mathsf{cp}_{\mathcal{P}^{\mathrm{dci}}_{\mathcal{R}}}(n)$*; and*

(3) $\mathrm{rc}_{\mathcal{R}}(n) =_{\mathsf{k}} \mathsf{cp}_{\mathcal{P}^{\mathrm{rc}}_{\mathcal{R}}}(n)$*; and*

(4) $\mathrm{rci}_{\mathcal{R}}(n) =_{\mathsf{k}} \mathsf{cp}_{\mathcal{P}^{\mathrm{rci}}_{\mathcal{R}}}(n)$.

Proof. We have $\to_{\mathcal{R}} = \xrightarrow{\varnothing}_{\mathcal{R}} = \xrightarrow{\varnothing}_{\mathcal{R}/\varnothing}$ and similar $\xrightarrow{\mathrm{i}}_{\mathcal{R}} = \xrightarrow{\mathcal{R}}_{\mathcal{R}/\varnothing}$ for all TRSs $\mathcal{R}$. The lemma follows by definition. □

Complexity problems are thus expressive enough to reflect derivational and runtime complexity, for full and innermost rewriting. We could have integrated support for additional strategies, for instance by allowing the notions of μ-*replacing positions* as in *context sensitive rewriting* [53] or *forbidden patterns* [36] in the definition of complexity problem. Such an extension goes beyond this thesis.

Consider a problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$. If $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ is *terminating* and *finitely branching* on $\mathcal{T}$, then the complexity function $\mathsf{cp}_{\mathcal{P}}$ is defined on all inputs, by König's Lemma. The following example shows that in the relative setting the termination property alone does not suffice that $\mathsf{cp}_{\mathcal{P}}$ is defined on all inputs.

Example 13.7. Consider the problem $\mathcal{P}_1 := \langle \mathcal{S}_1/\mathcal{W}_1, \varnothing, \mathcal{T}_1\rangle$ where

$$\mathcal{S}_1 := \{\mathsf{g}(\mathsf{s}(x)) \to \mathsf{g}(x)\} \qquad \mathcal{W}_1 := \{\mathsf{f}(x) \to \mathsf{f}(\mathsf{s}(x)),\ \mathsf{f}(x) \to \mathsf{g}(x)\}\ ,$$

and $\mathcal{T}_1 := \{\mathsf{f}(\bot)\}$. Note that for all $n \in \mathbb{N}$, maximal $\to_{\mathcal{P}_1}$ derivations are of the form

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

Hence $\mathsf{f}(\bot) \to^n_{\mathcal{S}_1/\mathcal{W}_1} \mathsf{g}(\bot)$ holds for all $n \in \mathbb{N}$. Whereas $\to_{\mathcal{S}_1/\mathcal{W}_1}$ is well-founded on $\mathcal{T}_1$, the above family of derivations shows that $\mathsf{cp}_{\mathcal{P}_1}(m) =_{\mathsf{k}} \mathsf{dh}(\mathsf{f}(\bot), \to_{\mathcal{S}_1/\mathcal{W}_1})$ is undefined for $m \geqslant 2$. ◁

The example exploits that the rewrite relation, although well-founded, is not finitely branching. The next example shows that even if $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ is not finitely branching on $\mathcal{T}$, then $\mathsf{cp}_{\mathcal{P}}$ can still be defined on all inputs.

Example 13.8 (Continued from Example 13.7). Consider the complexity problem $\mathcal{P}_2 := \langle \mathcal{S}_2/\mathcal{W}_1, \varnothing, \mathcal{T}_1\rangle$, where $\mathcal{S}_2 := \{\mathsf{g}(x) \to x\}$. The complexity function of $\mathcal{P}_2$ is constant, but $\mathsf{f}(\bot) \to_{\mathcal{S}_2/\mathcal{W}_1} \mathsf{s}^n(\bot)$ for all $n \in \mathbb{N}$, i.e, $\to_{\mathcal{S}_2/\mathcal{W}_1}$ is not finitely branching on $\mathcal{T}_1$. ◁

Definition 13.9 (Judgement, Processor, Proof)**.**

(1) A *(complexity) judgement* is a statement $\vdash \mathcal{P} \colon f$ where $\mathcal{P}$ is a complexity problem and $f : \mathbb{N} \to \mathbb{N}$. The judgement is *valid* if $\mathsf{cp}_{\mathcal{P}}$ is defined on all inputs, and $\mathsf{cp}_{\mathcal{P}} \in \mathcal{O}(f)$.

(2) A *complexity processor* Proc (*processor* for short) is an inference rule

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

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

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

We say that a complexity proof is *closed* if its set of assumptions is empty, otherwise it is *open*. We follow the usual convention and annotate side conditions as premises to inference rules. When the list of premises in a processor

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

is empty, i.e., $n = 0$, we call a processor also a *direct processor*, otherwise it is also called a *transformation*.

Soundness of a processor guarantees our formal system is correct. Completeness ensures that a deduction gives asymptotically tight bounds.

Theorem 13.10. *If there exists a closed complexity proof* $\vdash \mathcal{P} \colon f$*, then the judgement* $\vdash \mathcal{P} \colon f$ *is valid.*

Proof. The theorem follows by a standard induction on the size of proofs, exploiting that the underlying set of processors is sound. □

As we see in the sequel, this formalisation is expressive enough to cover a majority of the techniques available for the automated complexity analysis. To justify our design choices, we briefly compare our formulations to previously established notions.

Related Work. Our notion of complexity problem is a natural extension of the one established by Zankl and Korp [79] for derivational complexity analysis, which underlies the complexity tool CaT. Here a complexity problem consists of

a relative rewrite relation $\mathcal{S}/\mathcal{W}$ together with a bounding function $f : \mathbb{N} \to \mathbb{N}$. A processor in this setting is a function

$$((\mathcal{S}_1 \cup \mathcal{S}_2)/\mathcal{W}, f) \mapsto (\mathcal{S}_1/(\mathcal{S}_2 \cup \mathcal{W}), f') \ ,$$

which can shift rules from the strict component to the weak component. This processor is sound if $f(n) + \mathrm{dc}_{(\mathcal{S}_1 \cup \mathcal{S}_2)/\mathcal{W}}(n) \in \mathcal{O}(f'(n) + \mathrm{dc}_{\mathcal{S}_1/(\mathcal{S}_2 \cup \mathcal{W})}(n))$ holds. Here $\mathrm{dc}_{\mathcal{S}/\mathcal{W}}$ is defined in correspondence to the derivational complexity function of a TRS $\mathcal{R}$, by exchanging the rewrite relation $\to_{\mathcal{R}}$ by the relation $\to_{\mathcal{S}/\mathcal{W}}$. Our framework generalises these notions in various aspects. First of all, we have made the set of starting terms abstract, which allows us to cover the derivational and runtime complexity of TRSs. We also allow transformations where rules appearing in the generated sub-problem do not appear in the input problem. Instances of such transformations are for example the two notions of dependency pair transformations discussed later. We permit processors to transform the input problem into more than one sub-problem. Furthermore, we do not require linear combinations of complexity certificates. Both properties are necessary for instance in the formulation of dependency graph decomposition.

Similar to our framework is also the framework underlying AProVE [63]. In the basic setting, a complexity input consists of a triple $(\mathcal{D}, \mathcal{S}, \mathcal{R})$ for *dependency tuples* $\mathcal{S} \subseteq \mathcal{D}$ (compare Section 14.4) and rewrite rules $\mathcal{R}$. A complexity proof in this system is given as a sequence of applications of processors. Let $\mathcal{W} := \mathcal{D} \setminus \mathcal{S}$. The complexity of such a problem essentially amounts to the number of applications of dependency tuples from $\mathcal{S}$ in derivations $\xrightarrow{\mathcal{R}}_{\mathcal{S}/\mathcal{W} \cup \mathcal{R}}$ starting from a suitable set of terms $\mathcal{T}_{\mathsf{b}}^{\sharp}$. In our setting, the triple $(\mathcal{D}, \mathcal{S}, \mathcal{R})$ can thus be represented as the problem $\langle \mathcal{S}/\mathcal{W} \cup \mathcal{R}, \mathcal{R}, \mathcal{T}_{\mathsf{b}}^{\sharp} \rangle$. In this sense our notion of complexity problem is more general, since we do not restrict the strict component to dependency tuples. This generality is necessary in our setting, since dependency pairs for complexity are not available for derivational complexity analysis. Also, for the case of full rewriting the only established dependency pair transformation, viz, weak dependency pairs from [38], requires us to account some rewrite rules of the input.

Finally, we note that [63] also propose an extension of their notion of complexity problem, using an additional component $\mathcal{K}$ of *known* rules. These are rules whose complexity, in the sense of the number of applications of rules from $\mathcal{K}$ in the considered derivations, has already been assessed. This information can be reused in the *knowledge propagation* processor [63], see Section 14.5. Our framework cannot capture this extension, although we could in principle extend our notion of complexity problem sufficiently. This of course incurs some complications, as in each processor the additional component $\mathcal{K}$ needs to be treated properly.

Chapter 14.

Complexity Processors in TCT

In this section we cover those techniques that are used for (innermost) runtime complexity analysis in our tool TCT. Most of the discussed methods constitute adaptions of known techniques [79, 38, 63] to our framework. Throughout the remaining of this chapter, the following two rewrite systems will serve as running examples. The first is a small toy example.

Example 14.1. Consider the rewrite system $\mathcal{R}_{\mathsf{mult}}$ given by the following four rules.

$$\begin{array}{ll} 28\colon 0 + y \to y & 29\colon \mathsf{s}(x) + y \to \mathsf{s}(x + y) \\ 30\colon 0 \times y \to 0 & 31\colon \mathsf{s}(x) \times y \to y + (x \times y)\ . \end{array}$$

Let $\mathcal{T}_{\mathsf{mult}}$ denote basic terms with defined symbols $+, \times$ and constructors $\mathsf{s}, 0$. We denote by $\mathcal{P}_{\mathsf{mult}}$ the canonical runtime complexity problem $\langle \mathcal{R}_{\mathsf{mult}}/\varnothing, \varnothing, \mathcal{T}_{\mathsf{mult}} \rangle$, and by $\mathcal{P}_{\mathsf{mult\text{-}i}}$ the *innermost* runtime complexity $\langle \mathcal{R}_{\mathsf{mult}}/\varnothing, \mathcal{R}_{\mathsf{mult}}, \mathcal{T}_{\mathsf{mult}} \rangle$ of $\mathcal{R}_{\mathsf{mult}}$. ◁

The runtime complexity analysis of our second TRS $\mathcal{R}_{\mathsf{K}}$ is significantly more involved. This example implements Kruskal's algorithm for computing a spanning forest of minimal weight for a given graph, compare Figure 14.1.

Let N denote the nodes and E weighted edges of a graph G.

Set $F := \varnothing$ and $P := \{\{n\} \mid n \in N\}$.

For all edges $e \in E$, sorted increasingly by their weight, do:

  If source and target of e occur in $p, q \in P$ respectively, with $p \neq q$;
    Set $P := P \setminus \{p, q\} \cup \{p \cup q\}$ and $F := F \cup \{e\}$.

Return F.

Figure 14.1.: Kruskal's Algorithm.

Example 14.2. In $\mathcal{R}_{\mathsf{K}}$, a graph is represented as a value $\mathsf{graph}(N, E)$ where N and E refer to the nodes and edges respectively. We suppose nodes and weights are given as natural numbers. For simplicity, these are encoded as tally numbers $\mathsf{s}^n(0)$. Edges e are given as triples (n, w, m), where the rules

$$32\colon \mathsf{src}((n, w, m)) \to n \quad 33\colon \mathsf{wt}((n, w, m)) \to w \quad 34\colon \mathsf{trg}((n, w, m)) \to m\ ,$$

provide projections to the source node n, weight w, and target node m. The following rules contained in $\mathcal{R}_{\mathsf{K}}$ implement Kruskal's algorithm.

$$\begin{aligned}
&35: && \mathsf{forest}(\mathsf{graph}(N,E)) \to \mathsf{kruskal}(\mathsf{sort}(E),[\,],\mathsf{partitions}(N))\\
&36: && \mathsf{partitions}([\,]) \to [\,]\\
&37: && \mathsf{partitions}(n :: N) \to (n :: [\,]) :: \mathsf{partitions}(N)\\
&38: && \mathsf{kruskal}([\,],W,P) \to W\\
&39: && \mathsf{kruskal}(e :: E,W,P) \to \mathsf{kruskal?}(\mathsf{inBlock}(e,P),e,E,W,P)\\
&40: && \mathsf{kruskal?}(\mathsf{tt},e,E,W,P) \to \mathsf{kruskal}(E,W,P)\\
&41: && \mathsf{kruskal?}(\mathsf{ff},e,E,W,P) \to \mathsf{kruskal}(E,e :: W,\mathsf{join}(e,P,[\,]))\\
&42: && \mathsf{inBlock}(e,[\,]) \to \mathsf{ff}\\
&43: && \mathsf{inBlock}(e,p :: P) \to (\mathsf{src}(e) \in p \wedge \mathsf{trg}(e) \in p) \vee \mathsf{inBlock}(e,P)\\
&44: && \mathsf{join}(e,[\,],q) \to q :: [\,]\\
&45: && \mathsf{join}(e,p :: P,q) \to \mathsf{join?}(\mathsf{src}(e) \in p \vee \mathsf{trg}(e) \in p,e,p,P,q)\\
&46: && \mathsf{join?}(\mathsf{tt},e,p,P,q) \to \mathsf{join}(e,P,p +\!\!+ q)\\
&47: && \mathsf{join?}(\mathsf{ff},e,p,P,q) \to p :: \mathsf{join}(e,P,q)\ .
\end{aligned}$$

The defined symbol forest starts the computation on input graph $\mathsf{graph}(N,E)$. The rules (38)—(41) are used to iterate the loop from Figure 14.1. The rules (42) and (43) check the condition, and the remaining rules (44)–(47) execute the body of the conditional. To sort edges according to their weight, the TRS $\mathcal{R}_{\mathsf{K}}$ uses the following implementation of insertion sort.

$$\begin{aligned}
&48: && \mathsf{sort}([\,]) \to [\,]\\
&49: && \mathsf{sort}(e :: E) \to \mathsf{insert}(e,\mathsf{sort}(E))\\
&50: && \mathsf{insert}(e,[\,]) \to e :: [\,]\\
&51: && \mathsf{insert}(e,f :: E) \to \mathsf{insert?}(\mathsf{wt}(e) \leqslant \mathsf{wt}(f),e,f,E)\\
&52: && \mathsf{insert?}(\mathsf{tt},e,f,E) \to e :: (f :: E)\\
&53: && \mathsf{insert?}(\mathsf{ff},e,f,E) \to f :: \mathsf{insert}(e,E)\ .
\end{aligned}$$

For sets, a list representation is employed. Membership and union is defined as follows.

$$\begin{aligned}
&54:\ n \in [\,] \to \mathsf{ff} && 55:\ n \in (m :: p) \to n = m \vee n \in p\\
&56:\ [\,] +\!\!+ q \to q && 57:\ (n :: p) +\!\!+ q \to n :: (p +\!\!+ q)\ .
\end{aligned}$$

Finally, the following rules define standard Boolean operations, and comparisons

on natural numbers.

$$
\begin{array}{llll}
58\colon & 0 = 0 \to \mathsf{tt} & 59\colon & \mathsf{s}(x) = 0 \to \mathsf{ff} \\
60\colon & 0 = \mathsf{s}(y) \to \mathsf{ff} & 61\colon & \mathsf{s}(x) = \mathsf{s}(y) \to x = y\ , \\
62\colon & 0 \leqslant 0 \to \mathsf{tt} & 63\colon & \mathsf{s}(x) \leqslant 0 \to \mathsf{ff} \\
64\colon & 0 \leqslant \mathsf{s}(y) \to \mathsf{tt} & 65\colon & \mathsf{s}(x) \leqslant \mathsf{s}(y) \to x \leqslant y\ , \\
66\colon & \mathsf{ff} \wedge \mathsf{ff} \to \mathsf{ff} & 67\colon & \mathsf{ff} \wedge \mathsf{tt} \to \mathsf{ff} \\
68\colon & \mathsf{tt} \wedge \mathsf{ff} \to \mathsf{ff} & 69\colon & \mathsf{tt} \wedge \mathsf{tt} \to \mathsf{tt}\ , \\
70\colon & \mathsf{ff} \vee \mathsf{ff} \to \mathsf{ff} & 71\colon & \mathsf{ff} \vee \mathsf{tt} \to \mathsf{tt} \\
72\colon & \mathsf{tt} \vee \mathsf{ff} \to \mathsf{tt} & 73\colon & \mathsf{tt} \vee \mathsf{tt} \to \mathsf{tt}\ .
\end{array}
$$

We set $\mathcal{R}_{\mathsf{K}} := \{(32)\text{—}(73)\}$. Let $\mathcal{T}_{\mathsf{K}}$ denote the set of basic terms where constructors and defined symbols coincide with those of $\mathcal{R}_{\mathsf{K}}$. The problem $\langle \mathcal{R}_{\mathsf{K}}/\varnothing, \mathcal{R}_{\mathsf{K}}, \mathcal{T}_{\mathsf{K}} \rangle$ is the canonical innermost runtime complexity problem of $\mathcal{R}_{\mathsf{K}}$.$\lhd$

14.1. Suiting Reduction Orders to Complexity

Orders have been used quite early for the (automated) complexity analysis of rewrite systems. The seminal paper by Bonfante et al. [23] gives an early account on using reduction orders for complexity analysis, in the form of polynomial interpretations. In [79] pairs of orders $(\succsim, \succ)$, called *complexity pairs*, are employed to estimate the derivational complexity in a relative setting. *Safe reduction pairs* [38] constitute a variation of complexity pairs. These are useful in conjunction with *dependency pairs*, compare Section 14.4. In the following, we introduce $\mathcal{P}$*-monotone complexity pairs*, which provide a unified account of these notions.

Fix a complexity problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$. Consider a reduction

$$t = t_0 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_2 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} \cdots\ ,$$

for starting term $t \in \mathcal{T}$. Suppose we have shown termination of such sequences by means of a well founded order $\succ$ on terms: $t_i \succ t_{i+1}$ holds for all steps $t_i \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_{i+1}$. If there exists a function $f : \mathbb{N} \to \mathbb{N}$ which binds $\mathsf{dh}(t, \succ)$ in the size of $t \in \mathcal{T}$, then f gives an upper bound on the complexity function of $\mathcal{P}$. Observe that for instance recursive path orders $\succ_{\mathsf{rpo},\tau}$ are not finitely-branching. Thus $\mathsf{dh}(t, \succ_{\mathsf{rpo},\tau})$ is not necessarily well-defined. Only the restriction of $\succ_{\mathsf{rpo},\tau}$ to the rewrite relation is finitely branching, and can thus be used in the reasoning above. To allow for orders which are not finitely branching, Hirokawa and Moser [38] propose the notion of *G-collapsible order*. The following provides an adaption of this notion.

Definition 14.3 (***G***-collapsible, Induced Complexity)**.** Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ denote a complexity problem, consider a proper order $\succ$ on terms.

(1) Suppose there exists a mapping $G : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathbb{N}$ such that

$$s \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t \text{ and } s \succ t \implies G(s) > G(t) \,,$$

holds for all terms $s \in \to^*_{\mathcal{P}}(\mathcal{T})$. Then $\succ$ is called *G-collapsible* on $\mathcal{P}$.

The order $\succ$ is *collapsible* with respect to $\mathcal{P}$ if there exists a mapping G such that $\succ$ is G-collapsible on $\mathcal{P}$.

(2) Consider an order $\succ$ which is G-collapsible with respect to $\mathcal{P}$. Suppose that there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that

$$G(t) \in \mathcal{O}(f(|t|)) \text{ holds for all } t \in \mathcal{T}.$$

Then we say that $\succ$ *induces* the complexity f on $\mathcal{P}$.

Any reduction order $\succ$ compatible with $\mathcal{R}$ is collapsible with respect to the canonical complexity problems of $\mathcal{R}$. Usually however, the induced complexity is far beyond a polynomial function. For the case of RPO or polynomial interpretations, compare Proposition 2.57 and Proposition 2.62. For polynomial complexity analysis it is thus necessary to tame these techniques, as we have done with small polynomial path orders for example. For interpretation methods, this can be achieved by restricting the form of interpretation functions.

Lemma 14.4. *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ denote a complexity problem. Let $\succ$ be an order that is G-collapsible with respect to $\mathcal{P}$. Suppose*

$$s \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t \implies s \succ t \,,$$

holds for all $s \in \to^*_{\mathcal{P}}(\mathcal{T})$. Then for all terms $t \in \mathcal{T}$, $\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}})$ is defined, in particular $\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}) \leqslant G(t)$.

Proof. Consider a reduction

$$t = t_0 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_1 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_2 \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} \cdots \,,$$

for $t \in \mathcal{T}$. By the assumptions also $t = t_0 \succ t_1 \succ t_2 \succ \cdots$ holds, and since $\succ$ is G-collapsible with respect to $\mathcal{P}$ we have

$$G(t) = G(t_0) > G(t_1) > G(t_2) > \cdots \,.$$

Since $>$ is well-founded, it follows that any reduction of $t \in \mathcal{T}$ is not only finite, but its length is bounded by $G(t)$. □

Consider an order $\succ$ that induces the complexity f on $\mathcal{P}$. If this order includes the relation $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ on terms $t \in \to^*_{\mathcal{P}}(\mathcal{T})$, the above lemma shows that judgement $\vdash \mathcal{P} \colon f$ is valid. To check the inclusion, as in [79] we consider a pair of orders $(\succsim, \succ)$ on terms. Here $\succsim$ denotes a pre-order on terms, and $\succ$ an order compatible with $\succsim$: $\succsim \cdot \succ \cdot \succsim \subseteq \succ$. In [79], it is further require that both orders are monotone and stable under substitutions. In this case, the assertions $\mathcal{W} \subseteq \succsim$ and $\mathcal{S} \subseteq \succ$ imply $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} \subseteq \succ$ as desired.

Guided by the observation that monotonicity is required only on argument positions that can be rewritten in reductions of starting terms, Hirokawa and Moser [40] propose the use of *μ-monotone orders* for runtime complexity analysis. Initially introduced for termination analysis [80] of *context sensitive rewrite systems* [53], the parameter μ denotes a *replacement map*. In the realm of context sensitive rewriting, this map governs under which argument positions a rewrite step is allowed. Here the mapping μ is used to designate which arguments are *usable* in derivations, i.e., can be reduced. The following constitutes an adaption of *usable replacement maps* from rewrite systems [40] to complexity problems.

Definition 14.5 (Usable Replacement Maps)**.**

(1) A map $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ with $\mu(f) \subseteq \{1, \dots, k\}$ for every $f/k \in \mathcal{F}$ is called a *usable replacement map* on $\mathcal{F}$. The positions $\mu(f)$ are called the *usable argument positions of* f.

(2) For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, the set $\mathcal{P}\mathrm{os}_\mu(t)$ of *μ-replacing positions* in t is defined such that

$$\mathcal{P}\mathrm{os}_\mu(t) := \begin{cases} \{\epsilon\} & \text{if } t \text{ is a variable,} \\ \{\epsilon\} \cup \{i{\cdot}p \mid i \in \mu(f) \text{ and } p \in \mathcal{P}\mathrm{os}_\mu(t_i)\} & \text{if } t = f(t_1, \dots, t_k). \end{cases}$$

(3) For a binary relation $\to$ on terms we denote by $\mathcal{T}_\mu(\to)$ the set of terms t where sub-terms at non-μ-replacing positions are in normal form: $t \in \mathcal{T}_\mu(\to)$ if for all positions $p \in \mathcal{P}\mathrm{os}(t)$, if $p \notin \mathcal{P}\mathrm{os}_\mu(t)$ then $t|_p \in \mathrm{NF}(\to)$.

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

Consider a $\mathcal{P}$ derivation of a term $t \in \mathcal{T}$, and let μ denote a usable replacement map for $\mathcal{R}$ in $\mathcal{P}$. If a rule $l \to r \in \mathcal{R}$ is applied in this derivation at position p, then p is a μ-replacing position in the considered term.

Example 14.6 (Continued from Example 14.1). Consider the $\mathcal{P}_{\mathsf{mult}}$-derivation

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

where redexes are underlined. Here, and also in consecutive examples, we again use for $n \in N$ the notation $\mathbf{n}$ for the numeral $\mathsf{s}^n(0)$. Observe that if addition occurs in a context, then only under the successor symbol. This holds even for all reductions of basic terms. The map μ_+, defined by $\mu_+(\mathsf{s}) = \{1\}$ and $\mu_+(\times) = \mu_+(\mathsf{s}) = \varnothing$, thus constitutes a usable replacement map for the addition rules $\{28, 29\}$ in $\mathcal{P}_{\mathsf{mult}}$. Since for instance no argument position of addition is usable in μ_+, the second step witnesses that μ_+ does not designate a usable replacement map for the multiplication rules $\{30, 31\}$ in $\mathcal{P}_{\mathsf{mult}}$. ◁

Definition 14.7 (**$\mathcal{P}$**-monotone, Complexity Pair)**.** Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ denote a complexity problem.

(1) An order $\succ$ is called *μ-monotone* if it is monotone on μ-positions, in the sense that for all function symbols f, if $i \in \mu(f)$ and $s_i \succ t_i$ holds then

$$f(s_1, \ldots, s_i, \ldots, s_n) \succ f(s_1, \ldots, t_i, \ldots, s_n) \ ,$$

holds.

(2) A *complexity pair* $(\succsim, \succ)$ consists of a pre-order $\succsim$ and an order $\succ$ that are both closed under substitutions and satisfy $\succsim \cdot \succ \cdot \succsim \subseteq \succ$.

(3) The complexity pair $(\succsim, \succ)$ is called $\mathcal{P}$-monotone if

- $\succ$ is μ-monotone for a usable replacement map μ of $\mathcal{S}$ in $\mathcal{P}$; and
- $\succsim$ is τ-monotone for a usable replacement map τ of $\mathcal{W}$ in $\mathcal{P}$.

Definition 14.8 (Compatible)**.** We say that a complexity pair $(\succsim, \succ)$ is *compatible* with a complexity problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ if $\mathcal{W} \subseteq \succsim$ and $\mathcal{S} \subseteq \succ$ holds.

Lemma 14.9. *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ be a complexity problem, and $\mathcal{R} \subseteq \mathcal{S} \cup \mathcal{W}$ denote a set of rewrite rules in $\mathcal{P}$.*

(1) *Let μ denotes a usable replacement map for $\mathcal{R}$ in $\mathcal{P}$, and suppose $\mathcal{R}$ is compatible with a μ-monotone order $\succ$ that is stable under substitutions. Then*

$$s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t \implies s \succ t \ ,$$

holds for all $s \in \to^*_{\mathcal{P}}(\mathcal{T})$.

(2) *If $(\succsim, \succ)$ is a $\mathcal{P}$-monotone complexity pair compatible with $\mathcal{P}$, then*

$$s \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t \implies s \succ t \ ,$$

holds for all $s \in \to^*_{\mathcal{P}}(\mathcal{T})$.

Proof. Consider the first assert. Since μ is a usable replacement map for $\mathcal{R}$ in $\mathcal{P}$, it suffices to show the claim for $s \in \mathcal{T}_\mu(\xrightarrow{\mathcal{Q}}_{\mathcal{R}})$. Suppose $s \xrightarrow{\mathcal{Q}}_{\mathcal{R},p} t$, hence $p \in \mathcal{P}\mathrm{os}_\mu(t)$. We show that for every prefix q of p, $s|_q \succ t|_q$ holds. The proof is by induction on $|p| - |q|$. The base case $q = p$ is covered by compatibility and stability under substitutions. For the inductive step, consider a prefix $q{\cdot}i$ of p, where by induction hypothesis $s|_{q\cdot i} \succ t|_{q\cdot i}$. Since $p \in \mathcal{P}\mathrm{os}_\mu(s)$ it is not difficult to see that $i \in \mu(f)$. Thus

$$s|_q = f(s_1, \ldots, s|_{q\cdot i}, \ldots, s_n) \succ f(s_1, \ldots, t|_{q\cdot i}, \ldots, s_n) = t|_q \ ,$$

follows by μ-monotonicity of $\succ$. From the claim, the lemma is obtained using $q = \epsilon$.

For the second assertion, consider a $\mathcal{Q}$-restricted relative step

$$s \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{W}} \cdot \xrightarrow{\mathcal{Q}}_{\mathcal{S}} \cdot \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{W}} t \,,$$

for $s \in \to^*_{\mathcal{P}}(\mathcal{T})$. Using the assumptions on $(\succsim, \succ)$ and the inclusions $\mathcal{W} \subseteq \succsim$ and $\mathcal{S} \subseteq \succ$ to satisfy the assumptions of the first assertion, we obtain

$$s \succsim^* \cdot \succ \cdot \succsim^* t \,.$$

Hence $s \succ t$ follows by transitivity of $\succsim$ and the inclusion $\succsim \cdot \succ \cdot \succsim \subseteq \succ$. □

As immediate consequence of Lemma 14.4 and Lemma 14.9, we obtain the following processor.

Theorem 14.10 (Complexity Pair Processor)**.** *Consider a $\mathcal{P}$-monotone complexity pair $(\succsim, \succ)$ such that $\succ$ induces the complexity f on $\mathcal{P}$. The following processor is sound:*

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

Suppose no restriction is put on starting terms in $\mathcal{P}$. By definition, only the full replacement map is also a usable replacement map for rewrite rules occurring in $\mathcal{P}$. In this case, Theorem 14.10 requires that the orders $\succsim$ and $\succ$ are monotone in all argument positions, as in [79].

14.1.1. Complexity Pairs in TCT

In order to compute usable argument positions, TCT uses the approximations given in [40]. The command `uargs` allows the inspection of the usable argument positions for all rules appearing in the considered complexity problem.

TCT-interactive 14.1

```
TCT> load "examples/mult.trs"
  Current Proof State ------------------------------------------------

  Selected Open Problems:
  -----------------------
    Strict Trs:
      { +(0(), y) -> y
      , +(s(x), y) -> s(+(x, y))
      , *(0(), y) -> 0()
      , *(s(x), y) -> +(*(x, y), y) }
    StartTerms: basic terms
    Strategy: none
  ------------------------------------------------------------------------
TCT> [ua] <- uargs

  Usable Arguments with respect to Problem 1:
    Uargs(+) = {2}, Uargs(s) = {1}

TCT>
```

In its current form, TCT synthesises $\mathcal{P}$-monotone complexity pairs by constructing suitable polynomial and matrix interpretations. As even checking for

compatibility reduces to Hilbert's 10^{th} problem, the construction of a compatible interpretation is a rather difficult task. Our implementation follows the practical but incomplete approach of Contejean et.al. [26]. Here the order constraints are translated to Diophantine form. Assuming a fixed bound on coefficients, these are turned into satisfiability problems through bit-blasting. The obtained SAT-problem is then solved by relying on the SAT-solver MiniSat [30]. Cf. [70] for details on the implementation in TCT.

Polynomial and matrix interpretations are available in TCT as processors `poly` and `matrix` respectively. To control the number of bits used to encode coefficients in the bit-blasting phase, TCT provides the binary operator `withBits`. This operator takes an interpretation processor as first, and the desired number of bits as second argument.

To infer polynomial bounds from these interpretations methods, TCT restricts the shape of the interpretation functions. For the synthesis of polynomial interpretations, TCT interprets all function symbols by *strongly linear* interpretation functions $f_{\mathcal{A}}$ of the form

$$f_{\mathcal{A}}(x_1, \ldots, x_k) = \sum_{i=i_1}^{i_m} x_i + c \,,$$

where $1 \leqslant i_1 \leqslant \cdots \leqslant i_m \leqslant k$ and $c \in \mathbb{N}$. The interpretation of a term is thus related linearly to its size. The inferred complexity is linear. For runtime complexity problems, the restriction of the shape is only put onto constructor symbols. In this setting, TCT can therefor also handle problems whose complexity function is not bounded by a linear.

For matrix interpretations, TCT uses either *triangular matrices* as coefficients [61], or more sophisticated methods based on *algebraic* reasoning [62] or *automata techniques* [56]. The latter method is the default method used by TCT. To overwrite this behaviour one can use p `` `withCertBy` `` m where m is either `Triangular`, `Algebraic`, `Automaton` or `Unrestricted`. Continuing the previous session, we can advice TCT to search for a compatible $\mathcal{P}_{\mathsf{mult}}$-monotone complexity pair, given by a three-dimensional matrix interpretation, as follows.

TCT-interactive 14.2 (Continued from Session 14.1)

```
TCT> apply $ matrix `withDimension` 3 `withCertBy` Algebraic

    No Progress :(
```

On the running example, TCT fails to synthesise a proper interpretation. This is of no surprise as the linear shape of matrix interpretations fails to express bounding functions which are non-linear in more than one variable.

To control the precise degree of the polynomial bound induced by a processor p, one can use p `` `withDegree` `` d for an optional integer d. Here d is either `Nothing` or `Just` i. In the former case, TCT will not put any restrictions on the processor. In the latter case, it will suitably restrict the constructed order so that the induced complexity function is a polynomial of degree i.

TCT-interactive 14.3 (Continued from Session 14.2)

```
TCT> apply $ poly `withDegree` Nothing

    Hurray, the problem was solved with certificate YES(?,O(n^4)).
    Use 'proof' to show the complete proof.
```

TCT has constructed a suitable complexity pair, however the constructed interpretation overestimates the complexity of the considered problem considerably. Polynomial interpretations that induce a quadratic bound, suitable for our example, can be used in TCT as follows.

TCT-interactive 14.4 (Continued from Session 14.3)

```
TCT> undo
✂.....................................................................................
TCT> apply $ poly `withDegree` Just 2

  Hurray, the problem was solved with certificate YES(?,O(n^2)).
  Use 'proof' to show the complete proof.

TCT> proof
✂.....................................................................................
  The following argument positions are considered usable:
    Uargs(+) = 2, Uargs(s) = 1
  TcT has computed the following constructor-restricted polynomial
  interpretation.
            [0]() = 1

      [+](x1, x2) = 3 + 2*x1 + x2

          [s](x1) = 1 + x1

      [*](x1, x2) = 1 + 2*x1 + 2*x1*x2 + 2*x1^2

  This order satisfies the following order constraints.

         [+(0(), x)] = 5 + x
                     > x
                     = [x]

        [+(s(x), y)] = 5 + 2*x + y
                     > 4 + 2*x + y
                     = [s(+(x, y))]

         [*(0(), x)] = 5 + 2*x
                     > 1
                     = [0()]

        [*(s(x), y)] = 5 + 6*x + 2*y + 2*x*y + 2*x^2
                     > 4 + 2*y + 2*x + 2*x*y + 2*x^2
                     = [+(y, *(x, y))]
```

On our second running example, the implementation of Kruskal's algorithm given in Example 14.2, TCT does not manage to synthesise a suitable matrix or polynomial interpretation. For polynomial interpretations, TCT exhausts 4GB of RAM before it eventually aborts. This clearly indicates that for a powerful complexity analyser, more sophisticated methods are needed. To this end, we introduce in the next section a transformation technique which can be used to combine various orders in a single proof.

14.2. Relative Decomposition

A variation of the complexity pair processor, that iteratively orients disjoint subsets of $\mathcal{S}$, occurred first in [79]. The following processor constitutes a straight forward generalisation of [79, Theorem 4.4] to our setting.

Theorem 14.11 (Decompose Processor)**.** *The following processor is sound:*

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

Here $f + g$ denotes the function h defined by $h(n) := f(n) + g(n)$.

Proof. The lemma follows from the inequality

$$\mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}_1 \cup \mathcal{S}_2/\mathcal{W}}) \leqslant \mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}_1/\mathcal{S}_2 \cup \mathcal{W}}) + \mathsf{dh}(t, \xrightarrow{\mathcal{Q}}_{\mathcal{S}_2/\mathcal{S}_1 \cup \mathcal{W}})\ .$$

The decompose processor is a central ingredient for the automated complexity analysis. All participants of the recent complexity sub-division of the annual termination competition rely on variations of this processor for the combination of different proof techniques. In correspondence to the *rule removal processor* for termination analysis [76], one can combine Theorem 14.10 and Theorem 14.11. See [79] and [63] where a similar combination is proposed. This way the synthesis procedure implementing the complexity pair processor can determine a fitting partitioning of strict rules. Unlike in the rule removal processor for termination, for complexity analysis we need to keep the oriented rules in the weak component, compare [79]. Oriented rules are thus *shifted* from the strict to the weak component. This is illustrated by the following example.

Example 14.12 (Continued from Example 14.6). Consider the linear polynomial interpretation $\mathcal{A}$ over $\mathbb{N}$ such that $0_{\mathcal{A}} = 0$, $\mathsf{s}_{\mathcal{A}}(x) = x$, $x +_{\mathcal{A}} y = y$ and $x \times_{\mathcal{A}} y = 1$. Let $\mathcal{P}_{30} := \langle \{30\}/\{28, 29, 31\}, \mathcal{R}_{\mathsf{mult}}, \mathcal{T}_{\mathsf{b}}\rangle$ denote the problem that accounts for the rules `30:` $0 \times y \to 0$ in $\mathcal{P}_{\mathsf{mult}}$. The induced order $>_{\mathcal{A}}$ together with the order $\geqslant_{\mathcal{A}}$, defined by $s \geqslant_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) \geqslant [\alpha]_{\mathcal{A}}(t)$ holds for all assignments α, forms a $\mathcal{P}_{30}$-monotone complexity pair $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$. Monotonicity can be shown using the replacement maps given in Example 14.6. The order $>_{\mathcal{A}}$ induces linear complexity on $\mathcal{P}_{30}$. According to Theorem 14.11, the following tree depicts a complexity proof $\langle \{28, 29, 31\}/\{30\}, \varnothing, \mathcal{T}_{\mathsf{b}}\rangle \colon g \vdash \mathcal{P}_{\mathsf{mult}} \colon n + g$.

$$\dfrac{\dfrac{\{30\} \subseteq >_{\mathcal{A}} \quad \{28, 29, 31\} \subseteq \geqslant_{\mathcal{A}}}{\vdash \langle \{30\}/\{28, 29, 31\}, \varnothing, \mathcal{T}_{\mathsf{b}}\rangle \colon n}\ \mathsf{CP} \qquad \vdash \langle \{28, 29, 31\}/\{30\}, \varnothing, \mathcal{T}_{\mathsf{b}}\rangle \colon g}{\vdash \mathcal{P}_{\mathsf{mult}} \colon n + g}\ \mathsf{dec.}$$

The above complexity proof can now be completed iteratively, on the simpler problem $\langle \{28, 29, 31\}/\{30\}, \varnothing, \mathcal{T}_{\mathsf{b}}\rangle$. Since the complexity of $\mathcal{P}_{\mathsf{mult}}$ is quadratic, one has to use a technique beyond linear polynomial interpretations here.

Theorem 14.13 (Decompose CP Processor)**.** *Consider a $\mathcal{P}_1$-monotone complexity pair $(\succsim, \succ)$, for a complexity problem $\mathcal{P}_1 = \langle \mathcal{S}_1/\mathcal{S}_2 \cup \mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$. Suppose $\succ$ induces the complexity f on $\mathcal{P}_1$. The following processor is sound:*

$$\frac{\mathcal{S}_1 \subseteq \succ \quad \mathcal{W} \cup \mathcal{S}_2 \subseteq \succsim \qquad \vdash \langle \mathcal{S}_2/\mathcal{S}_1 \cup \mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle \colon g}{\vdash \langle \mathcal{S}_1 \cup \mathcal{S}_2/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle \colon f + g} \text{ decompose CP .}$$

Proof. Immediate consequence of Theorem 14.10 and Theorem 14.11. □

We remark that the decompose processor finds applications beyond its combination with complexity pairs, cf. Section 14.6.

14.2.1. Relative Decomposition in TCT

Relative decomposition is implemented by the transformations `decomposeBy` (Theorem 14.11) and `decomposeAnyWith` (Theorem 14.13). The former requires a *selector expression* as argument, which determines the partitioning of strict rules from the input problem. The latter takes as argument a complexity pair processor that should be used in combination with relative decomposition. Selector expressions are of type `SelectorExpression`. The module `Tct.Interactive` exports a variety of basic selector expressions. More complex expression can be constructed by means of various combinators. The following session shows the implementation of Theorem 14.13 on the problem $\mathcal{P}_{\mathsf{mult}}$. The constructed proof corresponds to the one given in Example 14.12.

TCT-interactive 14.5

```
TCT> load "examples/mult.trs"
✂ ..............................................................................
TCT> apply  $ decomposeAnyWith (poly `withDegree` Just 1)

    Problems simplified. Use 'state' to see the current proof state.

TCT> proof
  1) decompose (addition) [OPEN]:
  -------------------------------

    We consider the following problem:
      Strict Trs:
        { +(0(), x) -> x
        , +(s(x), y) -> s(+(x, y))
        , *(0(), x) -> 0()
        , *(s(x), y) -> +(y, *(x, y)) }
      StartTerms: basic terms
      Strategy: none

    We use the processor 'custom shape polynomial interpretation' to
    orient the following rules strictly.

    Trs: { *(0(), x) -> 0() }

    The induced complexity on above rules (modulo remaining rules) is
    YES(?,O(n^1)) . These rules are moved into the corresponding weak
    component(s).
```

TCT-interactive 14.6 (Continued from Session 14.5)

```
Sub-proof:
----------
  The following argument positions are considered usable:
    Uargs(+) = {2}, Uargs(s) = {1}
  TcT has computed the following constructor-restricted polynomial
  interpretation.
✂..............................................................................
We return to the main proof.

1.1) Open Problem [OPEN]:
-------------------------
 We consider the following problem:
    Strict Trs:
      { +(0(), x) -> x
      , +(s(x), y) -> s(+(x, y))
      , *(s(x), y) -> +(y, *(x, y)) }
    Weak Trs: { *(0(), x) -> 0() }
    StartTerms: basic terms
    Strategy: none
```

On Kruskal's algorithm the interpretation method fails even in combination with relative decomposition. Techniques introduced after the next section will finally allow us simplify the complexity problem $\mathcal{P}_{\mathsf{K}}$.

14.3. Small Polynomial Path Orders as Complexity Pairs

We now adapt polynomial path orders to our framework. We consider the more powerful order $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ depicted in Definition 9.40 only. Observe that the order $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ is in general not closed under contexts. As a consequence, the pair $(\succ_{\mathsf{spop}^*_{\mathsf{ps}}}, \succsim_{\mathsf{spop}^*_{\mathsf{ps}}})$ does not form a $\mathcal{P}$-monotone complexity pair. The main theorem of this section states that nevertheless the pair $(\succ_{\mathsf{spop}^*_{\mathsf{ps}}}, \succsim_{\mathsf{spop}^*_{\mathsf{ps}}})$ can be used like a complexity pair. Guided by the observation that a μ-monotone order has to consider only arguments of f included in $\mu(f)$, we integrate *argument filterings* π into the order.

Definition 14.14 (Argument Filtering)**.**

(1) An argument filtering (for a signature $\mathcal{F}$) is a mapping π that assigns to every $f/k \in \mathcal{F}$ an argument position $i \in \{1, \dots, k\}$ or a (possibly empty) list $[i_1, \dots, i_l]$ of argument positions with $1 \leqslant i_1 < \cdots < i_l \leqslant k$. In the former case $\pi(f) = i$ we say that π *collapses* on f, otherwise it is called *non-collapsing* on f. Below π always denotes an argument filtering.

(2) For each $f \in \mathcal{F}$, let f_π denote a fresh function symbol associated with f. We define

$$\mathcal{F}_\pi := \{f_\pi/l \mid f \in \mathcal{F} \text{ and } \pi(f) = [i_1, \dots, i_l]\} .$$

The sets $\mathcal{D}_\pi$ and $\mathcal{C}_\pi$ denote the defined symbols and constructors in $\mathcal{F}_\pi$, which are given by the restriction of $\mathcal{F}_\pi$ to symbols f_π associated with $f \in \mathcal{D}$ and $f \in \mathcal{C}$ respectively.

(3) We denote by π also its extension $\pi\colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F}_\pi,\mathcal{V})$ to terms, given by

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable,} \\ \pi(t_i) & \text{if } t = f(t_1,\dots,t_k) \text{ and } \pi(f) = i, \\ f(\pi(t_{i_1}),\dots,\pi(t_{i_l})) & \text{if } t = f(t_1,\dots,t_k) \text{ and } \pi(f) = [i_1,\dots,i_l]\,. \end{cases}$$

We extend π to sets of rewrite rules:

$$\pi(\mathcal{R}) := \{\pi(l) \to \pi(r) \mid l \to r \in \mathcal{R}\}\,.$$

(4) For a usable replacement map μ and argument filtering π, we say that π *agrees with* μ if for all function symbols f in the domain of μ, either (i) $\pi(f) = i$ and $\mu(f) \subseteq \{i\}$ or otherwise (ii) $\mu(f) \subseteq \pi(f)$ holds.

Definition 14.15 (Polynomial Path Orders and Argument Filterings)**.** Let π denote an argument filtering over a signature $\mathcal{F}$. For $s,t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ we define

$$s \succ^{\pi}_{\mathsf{spop}^*_{\mathsf{ps}}} t :\Leftrightarrow \pi(s) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \pi(t) \quad \text{and} \quad s \succsim^{\pi}_{\mathsf{spop}^*_{\mathsf{ps}}} t :\Leftrightarrow \pi(s) \succsim_{\mathsf{spop}^*_{\mathsf{ps}}} \pi(t)\,,$$

where $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ denotes the small polynomial path order as given by a safe mapping safe, recursive symbols $\mathcal{K}_\pi \subseteq \mathcal{D}_\pi$ and quasi precedence $\succsim$ over the signature $\mathcal{F}_\pi$. The order $\succsim_{\mathsf{spop}^*_{\mathsf{ps}}}$ denotes the extension of $\succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ by safe equivalence $\approx_{\mathsf{s}}$ on $\mathcal{F}_\pi$.

The next theorem provides our first small polynomial path order processor. We remark that the precondition that π is non-collapsing on defined symbols of $\mathcal{S}$ is essential to bind the complexity as stated in the theorem. This is illustrated in Example 14.67 in Section 14.7, where we revisit small polynomial path orders in the context of dependency pairs.

Theorem 14.16. *Consider an innermost complexity problem $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ where $\mathcal{S}$ and $\mathcal{W}$ are constructor TRSs. Let μ denote a usable replacement map for $\mathcal{S}$ in $\mathcal{P}$, and let π denote an argument filtering on the symbols in $\mathcal{P}$ that agrees with μ and that is non-collapsing on defined symbols of $\mathcal{S}$. Let $\mathcal{K}_\pi \subseteq \mathcal{D}_\pi$ denote a set of recursive function symbols, and $\succsim$ an admissible precedence on $\mathcal{F}_\pi$. The following processor is sound, for $d := \max\{0\} \cup \{\mathsf{rd}_{\succsim,\mathcal{K}_\pi}(f_\pi) \mid f_\pi \in \mathcal{F}_\pi\}$.*

$$\frac{\mathcal{S} \subseteq \succ^{\pi}_{\mathsf{spop}^*_{\mathsf{ps}}} \quad \mathcal{W} \subseteq \succsim^{\pi}_{\mathsf{spop}^*_{\mathsf{ps}}}}{\vdash \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle \colon n^d}\,.$$

Proof. By the assumptions of the processor, we have $\pi(\mathcal{S}) \subseteq \succ_{\mathsf{spop}^*_{\mathsf{ps}}}$ and $\pi(\mathcal{W}) \subseteq \succsim_{\mathsf{spop}^*_{\mathsf{ps}}}$. Consider a rewrite rule $l \to r \in \mathcal{S} \cup \mathcal{W}$ such that π is non-collapsing on the root of l. Then $\pi(l) \succsim_{\mathsf{spop}^\star} \pi(r)$ implies that variables of $\pi(r)$ are included in $\pi(l)$, and thus, $\pi(l) \to \pi(r)$ is again a rewrite rule.

Similar to Section 9.1, we use the polynomial path order on sequences to define a mapping $G\colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathbb{N}$ that measures the length of $\xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}}$ descending sequences, via predicative interpretations $\mathcal{I}_N$. Recall that the parameter N dictates which terms are deleted in the predicative interpretation. Ideally, we would like to instantiate N by the images u_π of π on irreducible terms, i.e.,

$u_\pi = \pi(u)$ for u a normal form of $\to_{\mathcal{P}}$. However the set N potentially contains also images of reducible terms, i.e, $s_\pi \in N$ with $s_\pi = \pi(s)$ but s not a normal form of $\to_{\mathcal{P}}$. As $\mathcal{I}_N(s_\pi) = [\,]$ in this case, one cannot hope for the required embedding of $\mathcal{I}_N(\pi(s)) \to_{\mathcal{P}} \mathcal{I}_N(\pi(t))$.

To resolve the issue, we consider reductions where normal forms u which are not constructor terms are replaced by a constructor $\bot$. This gives us a one-to-one correspondence between values $\mathcal{T}(\mathcal{C})$ and irreducible terms, and we can decide based on $\pi(u)$ if u should be considered reducible in the interpretation. Call a term $f(s_1, \ldots, s_n)$ a *garbage term* if $f \notin \mathcal{C}$ but $f(s_1, \ldots, s_n) \in \mathrm{NF}(\mathcal{S} \cup \mathcal{W})$. Consider the following function $(\cdot)^\bot : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ defined by

$$f(s_1, \ldots, s_n)^\bot := \begin{cases} \bot & \text{if } f(s_1, \ldots, s_n) \text{ is a garbage term,} \\ f(s_1^\bot, \ldots, s_n^\bot) & \text{otherwise.} \end{cases}$$

Here $\bot$ is an arbitrary constant from $\mathcal{C}$. Consider the restriction $\mathcal{N}_{\mathcal{T}(\mathcal{C}_\pi)}$ of ground terms over the signature $\mathcal{F}_\pi$ as given in definition Definition 9.28, where in particular normal arguments are always values from $\mathcal{T}(\mathcal{C}_\pi)$. Abbreviate $\mathcal{N}_{\mathcal{T}(\mathcal{C}_\pi)}$ by $\mathcal{N}$, $\mathcal{I}_{\mathcal{T}(\mathcal{C}_\pi)}$ by $\mathcal{I}$ and define

$$G(t) := \mathsf{G}_{\mathcal{K}_\pi, \ell}(\mathcal{I}_{\mathcal{T}(\mathcal{C}_\pi)}(t)) \,,$$

where ℓ is the maximal size of a right-hand side in $\pi(\mathcal{S} \cup \mathcal{W})$. The theorem is a straight forward consequence of the following claim.

Claim. Suppose $s \in \to_{\mathcal{P}}^*(\mathcal{T})$ with $\pi(s^\bot) \in \mathcal{N}$. Then

$$\begin{aligned} s \xrightarrow{\mathcal{Q}}_{\mathcal{S}} t &\implies G(\pi(s^\bot)) > G(\pi(t^\bot)) \text{ and } \pi(t^\bot) \in \mathcal{N} \,, \\ s \xrightarrow{\mathcal{Q}}_{\mathcal{W}} t &\implies G(\pi(s^\bot)) \geqslant G(\pi(t^\bot)) \text{ and } \pi(t^\bot) \in \mathcal{N} \,. \end{aligned}$$

Proof of Claim. Let $l \to r \in \mathcal{P}$, and consider $s \in \to_{\mathcal{P}}^*(\mathcal{T})$ with $\pi(s^\bot) \in \mathcal{N}$. Suppose $s = C[l\sigma] \xrightarrow{\mathcal{Q}}_{\mathcal{P}} C[r\sigma] = t$ with context C and substitution $\sigma : \mathcal{V} \to \mathrm{NF}(\mathcal{Q})$. We employ the following observations.

(1) $(l\sigma)^\bot = l\sigma_\bot$ where $\sigma_\bot(x) := (\sigma(x))^\bot$ for all x in the domain of σ.

The equality follows by the assumption that l is constructor based and $l\sigma$ is reducible.

(2) $\pi(l\sigma) = \pi(l)\sigma_\pi$ and $\pi(r\sigma) = \pi(r)\sigma_\pi$ where $\sigma_\pi(x) := \pi(\sigma(x))$ for all x in the domain of σ.

The property follows by a standard induction on l and r respectively.

(3) If π is non-collapsing on the root of l then

$$\pi((l\sigma)^\bot) \xrightarrow{\mathcal{T}(\mathcal{C}_\pi)}_{\{\pi(l) \to \pi(r)\}} \pi(r\sigma_\bot) \,,$$

where $\xrightarrow{\mathcal{T}(\mathcal{C}_\pi)}_{\{\pi(l) \to \pi(r)\}}$ denotes the restriction of $\to_{\{\pi(l) \to \pi(r)\}}$ as given in Definition 9.27.

Define $\sigma_{\pi,\bot}(x) := \pi(\sigma_\bot(x))$ for all x in the domain of $\sigma_\bot$. Using the previous two observations we have

$$\pi((l\sigma)^\bot) = \pi(l)\sigma_{\pi,\bot} \to_{\{\pi(l) \to \pi(r)\}} \pi(r)\sigma_{\pi,\bot} = \pi(r\sigma_\bot) \,.$$

Recall that $\sigma : \mathcal{V} \to \mathrm{NF}(\mathcal{Q})$ by assumption. Hence in particular for any x in the domain of σ, we have $\sigma(x) \subseteq \mathrm{NF}(\mathcal{S} \cup \mathcal{W})$ as $\mathcal{P}$ is an innermost complexity problem. As a consequence, $\sigma_\perp(x) \in \mathcal{T}(\mathcal{C}_\pi)$ by definition, and thus $\sigma_{\pi,\perp}(x) \in \mathcal{T}(\mathcal{C}_\pi)$. Since $\pi(l)\sigma_{\pi,\perp}$ contains a defined symbol, viz, the root f_π of $\pi(l)$, we conclude $\pi(l)\sigma_{\pi,\perp} \notin \mathcal{T}(\mathcal{C}_\pi)$. The property follows.

(4) If π collapses the root f of l then $\pi((l\sigma)^\perp) \trianglerighteq_{/\approx} \pi(r\sigma_\perp)$.
This follows by the order constraints and the assumption that $\mathcal{W}$ is a constructor TRS, compare Lemma 9.15.

(5) If $\pi((l\sigma)^\perp) \in \mathcal{T}(\mathcal{C}_\pi)$ then $\pi((r\sigma)^\perp) \in \mathcal{T}(\mathcal{C}_\pi)$; in any case $\pi((r\sigma)^\perp) \in \mathcal{N}$.
It is not difficult to see that $(r\sigma)^\perp$ can be obtained from $r\sigma_\perp$ by replacing garbage terms t in $r\sigma_\perp$ by the constructor $\perp$.

Consider first the case $\pi((l\sigma)^\perp) \in \mathcal{T}(\mathcal{C}_\pi)$. Then π collapses the root of l. In this case we already observed $\pi((l\sigma)^\perp) \trianglerighteq_{/\approx} \pi(r\sigma_\perp)$ and so $\pi(r\sigma_\perp) \in \mathcal{T}(\mathcal{C}_\pi)$. Thus $\pi((r\sigma)^\perp) \in \mathcal{T}(\mathcal{C}_\pi)$ as desired. Otherwise, suppose π does not collapse the root of l. Since $\pi(l\sigma) \in \mathcal{N}$ by assumption, the third observation together with Lemma 9.42(1) gives $\pi(r\sigma_\perp) \in \mathcal{N}$, which implies $\pi((r\sigma)^\perp) \in \mathcal{N}$.

(6) $G(\pi(r\sigma_\perp)) \geqslant G(\pi((r\sigma)^\perp))$;
Consider the non-trivial case $\pi((r\sigma)^\perp) \notin \mathcal{T}(\mathcal{C}_\pi)$, i.e., $\mathcal{I}(\pi((r\sigma)^\perp)) \neq [\,]$. Then

$$\begin{aligned}\mathcal{I}(\pi(r\sigma_\perp)) &= [g_{\mathsf{n}}(t_1,\dots,t_k)] +\!\!+ \mathcal{I}_{\mathbb{N}}(t_{k+1}) +\!\!+ \mathcal{I}(t_{k+l})\\ \mathcal{I}(\pi((r\sigma)^\perp)) &= [g_{\mathsf{n}}(t'_1,\dots,t'_k)] +\!\!+ \mathcal{I}_{\mathbb{N}}(t'_{k+1}) +\!\!+ \mathcal{I}(t'_{k+l})\ ,\end{aligned}$$

where the terms t'_i $(i = 1,\dots,k+l)$ are obtained from t_i by possibly replacing sub-terms by $\perp \in \mathcal{T}(\mathcal{C}_\pi)$. Observe that $g_{\mathsf{n}}(t_1,\dots,t_k) \sqsupseteq_{\mathcal{K}_{\mathsf{n}},\ell} g_{\mathsf{n}}(t'_1,\dots,t'_k)$ does not necessarily hold, as a garbage term deep in t_i $(i \in \{1,\dots,k\})$ might be replaced, and thus $t_i \trianglerighteq_{/\approx} t'_i$ does not necessarily hold as required. Using Theorem 9.24 together with $\mathsf{dp}(t_i) \geqslant \mathsf{dp}(t'_i)$ for all $i = 1,\dots,k$ we nevertheless have

$$\mathsf{G}_{\mathcal{K},\ell}(g_{\mathsf{n}}(t_1,\dots,t_k)) \geqslant \mathsf{G}_{\mathcal{K},\ell}(g_{\mathsf{n}}(t'_1,\dots,t'_k))\ ,$$

since by a standard induction we also have

$$\mathsf{G}_{\mathcal{K},\ell}(\mathcal{I}(t_j)) \geqslant \mathsf{G}_{\mathcal{K},\ell}(\mathcal{I}(t'_j)) \qquad \text{for all } j = k+1,\dots,k+l,$$

summing up using Lemma 9.23 we conclude $G(\pi(r\sigma_\perp)) \geqslant G(\pi((r\sigma)^\perp))$.

We return to the proof of the claim, which is by induction on C. In the base case $C = \Box$. We analyse two cases separately:

- Suppose the root of l is collapsed by π. Hence $l \to r \in \mathcal{W}$ by assumption on π. Since l is constructor based, it follows that $\pi((l\sigma)^\perp) \in \mathcal{T}(\mathcal{C}_\pi)$, thus $\pi((r\sigma)^\perp) \in \mathcal{T}(\mathcal{C}_\pi)$ holds by Observation (4). Since $\pi(u) = [\,]$ whenever $u \in \mathcal{T}(\mathcal{C}_\pi)$, we obtain $G(\pi((l\sigma)^\perp)) = 0 = \pi((r\sigma)^\perp)$ as desired.

- Suppose the root of l is not collapsed by π. In this case

$$\pi((l\sigma)^\bot) \xrightarrow{\mathcal{T}(\mathcal{C}_\pi)}_{\pi(l)\to\pi(r)} \pi(r\sigma_\bot) \ ,$$

holds by Observation (3). Consider first the case $l \succ^\pi_{\mathsf{spop}^*_{\mathsf{ps}}} r$. Using that $\pi((l\sigma)^\bot) \in \mathcal{N}$ by Lemma 9.42(2) we have $\mathcal{I}(\pi((l\sigma)^\bot)) \sqsupseteq_{\mathcal{K}_n,\ell} \mathcal{I}(\pi(r\sigma_\bot))$. This allows us to conclude $G(\pi((l\sigma)^\bot)) > G(\pi(r\sigma_\bot)) \geqslant G(\pi((r\sigma)^\bot))$ by Observation (6), and $(r\sigma)^\bot \in \mathcal{N}$ by Observations (5).

Otherwise $\pi(l) \approx_{\mathsf{s}} \pi(r)$ and $l \to r \notin \mathcal{S}$. A standard induction on the definition of $\approx_{\mathsf{s}}$ gives $\mathcal{I}(\pi(l)\sigma_{\pi,\bot}) \approx \mathcal{I}(\pi(r)\sigma_{\pi,\bot})$. Since $\pi((l\sigma)^\bot) = \pi(l)\sigma_{\pi,\bot}$ and $\pi(r\sigma_\bot) = \pi(r)\sigma_{\pi,\bot}$ putting Observations (1) and (2) together, we have $G(\pi((l\sigma)^\bot)) = G(\pi(r\sigma_\bot))$. We conclude $G(\pi((l\sigma)^\bot)) \geqslant G(\pi((r\sigma)^\bot))$ and $(r\sigma)^\bot \in \mathcal{N}$ as in the previous case.

This finishes the base case. Consider now the inductive step

$$s = f(s_1, \ldots, s_i, \ldots, s_n) \xrightarrow{\mathcal{Q}}_{\{l\to r\}} f(s_1, \ldots, t_i, \ldots, s_n) = t \ ,$$

with $s_i \xrightarrow{\mathcal{Q}}_{\{l\to r\}} t_i$. We consider the non-trivial case that t is not a garbage term, thus

$$\begin{aligned} s^\bot &= f(s_1^\bot, \ldots, s_i^\bot, \ldots, s_n^\bot) \\ t^\bot &= f(s_1^\bot, \ldots, t_i^\bot, \ldots, s_n^\bot) \ . \end{aligned}$$

Note here that s is not garbage, since it is reducible by assumption. If π deletes the rewrite position i in s, then $l \to r \notin \mathcal{S}$ by the assumption that π agrees with the usable replacement map μ of $\mathcal{S}$. Since in this case $\pi(s^\bot) = \pi(t^\bot)$, the claim follows. Hence suppose π does not delete the rewrite position i.

If $\pi(f) = i$, the assertion follows directly from induction hypothesis, hence suppose $\pi(f) = [i_1, \ldots, i_k]$ with $i \in \{i_1, \ldots, i_k\}$. Consider first the case that π does not collapse on the root of l. Using $\pi(s^\bot) \in \mathcal{N}$ it follows that the rewrite position i is a safe argument position of f_π. From this it is not difficult to conclude $\pi(t^\bot) \in \mathcal{N}$ as by induction hypothesis $\pi(t_i^\bot) \in \mathcal{N}$. Using Lemma 9.23 we conclude $G(\pi(s^\bot)) \geqslant G(\pi(t^\bot))$ from the induction hypothesis $G(\pi(t_i^\bot)) \geqslant G(\pi(s_i^\bot))$ and definition of $\mathcal{I}$. Note that if $l \to r \in \mathcal{S}$, this result can be strengthened to $G(\pi(s^\bot)) > G(\pi(s^\bot))$ as desired, using the strengthened induction hypothesis $G(\pi(s_i^\bot)) > G(\pi(t_i^\bot))$. This concludes the case where π does not collapse the root of l. Now finally suppose that π collapses the root of l. It suffices to consider the new case that i is a normal argument position. In this case $\pi(s_i^\bot) \in \mathcal{T}(\mathcal{C}_\pi)$ by the assumption $\pi(s^\bot) \in \mathcal{N}$. As a consequence of Observations (4) and (5), we know $\mathsf{dp}(\pi(s_i^\bot)) \geqslant \pi(t_i^\bot)$, i.e., depths of normal arguments are not increasing. With the help of Theorem 9.24 and Lemma 9.23 we conclude $G(\pi(s^\bot)) \geqslant G(\pi(t^\bot))$ exactly as in Observation (6). As the Observations (4) and (5) also give $\pi(t_i^\bot) \in \mathcal{T}(\mathcal{C}_\pi)$, we conclude $\pi(t^\bot) \in \mathcal{N}$. This finishes the proof of the claim. □

We return to the main proof. Consider a reduction

$$f(\vec{v}) \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_1 \cdots \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_n \ ,$$

of a basic term $f(\vec{v})$. By the claim we obtain

$$G(\pi(f(\vec{v})^{\perp})) > G(\pi(t_1^{\perp})) > \cdots > G(\pi(t_n^{\perp})) \ ,$$

using that $\pi(f(\vec{v})^{\perp}) = f_\pi(\pi(v_1), \ldots, \pi(v_k))$, Theorem 9.24 and Lemma 9.23 gives the desired bound in the sum of depths of normal arguments in the filtered term $f_\pi(\pi(v_1), \ldots, \pi(v_k))$, hence in the size of $f(\vec{v})$.

14.3.1. Small Polynomial Path Orders in TCT

Small polynomial path orders as used in Theorem 14.16 (and as later extended in Theorem 14.68 and Theorem 14.69) are implemented by the processor `spopstarPS`. To synthesise a suitable precedence, safe mapping and argument filtering, TCT encodes the order constraints into propositional logic, relying on the SAT-solver MiniSat to solve these constrains. From a satisfying assignment TCT can then construct the desired order.

14.4. Dependency Pairs for Complexity Analysis

For termination analysis, it is nowadays standard to transform a termination problem first into a *dependency pair* problem, compare [5, 76]. The *dependency pairs* $\mathsf{DP}(\mathcal{R})$ of a TRS $\mathcal{R}$ contain for each call $g(t_1, \ldots, t_n)$ in a right-hand side r of a rule $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$, where $g \in \mathcal{D}$, a *dependency pair* $f^\sharp(l_1, \ldots, l_n) \to g^\sharp(t_1, \ldots, t_n)$. The *marked symbols* $f^\sharp$ and $g^\sharp$ are *dependency pair symbols*, associated with the defined symbols f, g. Termination of $\mathcal{R}$ is equivalent to the absence of *dependency pair derivations*

$$f_0^\sharp(\vec{s}_0) \to_{\mathsf{DP}(\mathcal{R})/\mathcal{R}} f_1^\sharp(\vec{s}_1) \to_{\mathsf{DP}(\mathcal{R})/\mathcal{R}} f_2^\sharp(\vec{s}_2) \cdots \ ,$$

where one can assume that the arguments $\vec{s}_i$ ($i \in \mathbb{N}$) are terms (over the original signature) that are terminating with respect to $\mathcal{R}$. This transformation opens the door for a wealth of techniques, for an overview we refer the reader to [76].

Dependency pair derivations track single chains of successive function call, but forget contexts. To make this technique applicable for runtime complexity analysis, two variations of dependency pairs have been proposed in the literature. *Weak dependency pairs*, introduced by Hirokawa and Moser [38], group parallel function calls in a rewrite rule into a single dependency pair. *Dependency tuples*, due to Noschinski et al. [63], group parallel and nested function calls. The latter notion is only sound for innermost rewriting, whereas the former notion can be used to track arbitrary derivation. This generality comes at the expense of simplicity of the analysis carried out on the generated sub-problem. Here, one still needs to account for the rewrite steps from the input problem $\mathcal{R}$ in dependency pair derivations.

In the following, we introduce first a notion of *dependency pair complexity problem* (*DP problem* for short). We then consider the above mentioned transformations of runtime complexity problems to dependency pair complexity problems. The corresponding processors are given in Theorem 14.24 and Theorem 14.29

respectively. Neither [38] nor [63] treat relative rewriting. To cover also the case where the weak component in a complexity problem is not empty, we reprove the central theorems of [38, 63] here.

14.4.1. Dependency Pair Complexity Problems

A DP problem in our setting is a complexity problem whose strict and weak component contains also dependency pairs. As in [38, 63] we allow *compound symbols* in right hand sides of dependency pairs. The purpose of these symbols is to group function calls. As for termination, we consider derivations starting from *marked terms* only.

Definition 14.17 (Dependency Pairs, Dependency Pair Complexity Problem)**.** Let $\mathcal{F}$ be a signature with defined symbols $\mathcal{D}$.

(1) For each $f/k \in \mathcal{D}$, let $f^\sharp$ denote a fresh function symbol of arity k, the *dependency pair symbol* (of f). The least extension of $\mathcal{F}$ to all dependency pair symbols is denoted by $\mathcal{F}^\sharp$.

For a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ we define the *marking of* t as

$$t^\sharp := \begin{cases} f^\sharp(t_1, \dots, t_k) & \text{if } t = f(t_1, \dots, t_k) \text{ and } f \in \mathcal{D}, \\ t & \text{otherwise.} \end{cases}$$

For a set $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, we denote by $T^\sharp$ the set of *marked terms*

$$T^\sharp = \{t^\sharp \mid t \in T\} \ .$$

(2) We denote by $\mathcal{C}\mathsf{om} = \{\mathsf{c}_0^0, \mathsf{c}_0^1, \dots, \mathsf{c}_1^0, \mathsf{c}_1^1, \dots, \mathsf{c}_2^0, \mathsf{c}_2^1, \dots\}$ a countable infinite signature of constructor symbols, where the arity of c_k^i in $\mathcal{C}\mathsf{om}$ is k for all $i, k \in \mathbb{N}$. Symbols in $\mathcal{C}\mathsf{om}$ are called *compound symbols*.

The identity of compound symbols occurring in terms is of no importance. This justifies that we write $\textsc{com}(t_1^\sharp, \dots, t_k^\sharp)$ for terms $\mathsf{c}_k^i(t_1^\sharp, \dots, t_k^\sharp)$ $(i \in \mathbb{N})$, for $k = 1$ we denote by $\textsc{com}(t^\sharp)$ also the term $t^\sharp$.

(3) A *dependency pair* (*DP* for short) is a rewrite rule $l^\sharp \to \textsc{com}(r_1^\sharp, \dots, r_k^\sharp)$ where $l, r_1, \dots, r_k \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

(4) Let $\mathcal{S}$ and $\mathcal{W}$ be two TRSs over $\mathcal{T}(\mathcal{F}, \mathcal{V})$, and let $\mathcal{S}^\sharp$ and $\mathcal{W}^\sharp$ be two sets of dependency pairs. A *dependency pair complexity problem*, or simply *DP problem*, is a runtime complexity problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ over marked basic terms $\mathcal{T}^\sharp \subseteq \mathcal{T}_\mathsf{b}^\sharp(\mathcal{D} \uplus \mathcal{C})$.

We keep the convention that $\mathcal{R}, \mathcal{S}, \mathcal{W}, \dots$ are TRSs over $\mathcal{T}(\mathcal{F}, \mathcal{V})$, and the marked version $\mathcal{R}^\sharp, \mathcal{S}^\sharp, \mathcal{W}^\sharp, \dots$ always denote sets of dependency pairs. For each $k \in \mathbb{N}$ we write c_k for the compound symbol c_k^0.

Example 14.18 (Continued from Example 14.1). Consider the dependency pairs

$$\begin{array}{ll} 74\colon\ 0 +^\sharp y \to \mathsf{c}_0 & 75\colon\ \mathsf{s}(x) +^\sharp y \to x +^\sharp y \\ 76\colon\ 0 \times^\sharp y \to \mathsf{c}_0 & 77\colon\ \mathsf{s}(x) \times^\sharp y \to \mathsf{c}_2(x +^\sharp (x \times y), x \times^\sharp y) \ . \end{array}$$

Let $\mathcal{T}^\sharp_{\mathsf{mult}}$ be the set of marked basic terms with defined symbols $+^\sharp, \times^\sharp$ and constructors $\mathsf{s}, 0$. Then $\mathcal{P}^\sharp_{\mathsf{mult}} := \langle \{74\text{–}77\}/\mathcal{R}_{\mathsf{mult}}, \mathcal{R}_{\mathsf{mult}}, \mathcal{T}^\sharp_{\mathsf{mult}} \rangle$, where $\mathcal{R}_{\mathsf{mult}}$ are the rules for addition and multiplication depicted in Example 14.1, is a DP problem. We anticipate that the DP problem $\mathcal{P}^\sharp_{\mathsf{mult}}$ reflects the complexity of the canonical *innermost* runtime complexity problem $\mathcal{P}_{\mathsf{mult\text{-}i}}$ of $\mathcal{R}_{\mathsf{mult}}$, compare Theorem 14.29 below. ◁

Call an n-holed context C a *compound context* if it contains only compound symbols. Consider the $\mathcal{P}^\sharp_{\mathsf{mult}}$ derivation

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

Observe that any term in the above sequence can be written as $C[t_1, \ldots, t_n]$ where C is a maximal compound context, and $t_1, \ldots, t_n$ are terms without compound symbols. For instance, the last term in this sequence is given as $C[\mathbf{0} \times^\sharp \mathbf{1}, \mathbf{1} +^\sharp (\mathbf{0} \times \mathbf{1}), \mathbf{0} \times^\sharp \mathbf{1}]$ for $C := \mathsf{c}_2(\Box, \mathsf{c}_2(\Box, \Box))$. This holds even in general. Note that the terms t_i $(i = 1, \ldots, n)$ are not necessarily marked, as our notion of dependency pair problem permits collapsing rule $l^\sharp \to x$ for x a variable. We capture this observation with the set $\mathcal{T}^\sharp_{\to}$.

Definition 14.19. The set $\mathcal{T}^\sharp_{\to}$ is defined as the least set of terms such that

(1) if $t \in \mathcal{T}(\mathcal{F})$ then $t \in \mathcal{T}^\sharp_{\to}$ and $t^\sharp \in \mathcal{T}^\sharp_{\to}$; and

(2) if $t_1, \ldots, t_k \in \mathcal{T}^\sharp_{\to}$ and $\mathsf{c}_k \in \mathcal{C}\mathsf{om}$ then $\mathsf{c}_k(t_1, \ldots, t_k) \in \mathcal{T}^\sharp_{\to}$.

The simple observation can now be formalised as follows.

Lemma 14.20. *For every TRS $\mathcal{R}$ and DPs $\mathcal{R}^\sharp$, we have $\to^*_{\mathcal{R}^\sharp \cup \mathcal{R}}(\mathcal{T}^\sharp_{\to}) \subseteq \mathcal{T}^\sharp_{\to}$. In particular, $\to^*_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp) \subseteq \mathcal{T}^\sharp_{\to}$ holds for every DP problem $\mathcal{P}^\sharp$.*

Proof. Let $s \in \mathcal{T}^\sharp_{\to}$, and suppose $s \to_{\mathcal{R}^\sharp \cup \mathcal{R}} t$ holds. Then without loss of generality,

$$s = C[s_1, \ldots, s_i, \ldots, s_n] \to_{\mathcal{R}^\sharp \cup \mathcal{R}} C[s_1, \ldots, t_i, \ldots, s_n] = t \ ,$$

with $s_i \to_{\mathcal{R}^\sharp \cup \mathcal{R}} t_i$ for maximal compound context C and possibly marked terms $s_1, \ldots, s_n$ containing no compound symbols. Since compound symbols occur only in roots of right-hand sides in $\mathcal{R}^\sharp$ we see that $t_i \in \mathcal{T}^\sharp_{\to}$ and so $t \in \mathcal{T}^\sharp_{\to}$. The first half of the lemma follows from this by inductive reasoning. From this, the second half of the lemma follows, using that $\mathcal{T}^\sharp \subseteq \mathcal{T}^\sharp_{\to}$, taking $\mathcal{R}^\sharp := \mathcal{S}^\sharp \cup \mathcal{W}^\sharp$ and $\mathcal{R} := \mathcal{S} \cup \mathcal{W}$. □

14.4.2. Weak Dependency Pairs

Definition 14.21 (Weak Dependency Pairs [38])**.** Let $\mathcal{R}$ denote a TRS such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$.

(1) Consider a rule $l \to C[r_1, \ldots, r_k]$ in $\mathcal{R}$, where C is a maximal context containing only constructors. We define

$$\mathsf{WDP}(l \to r) := l^\sharp \to \textsc{com}(r_1^\sharp, \ldots, r_k^\sharp) \;,$$

and call $\mathsf{WDP}(l \to r)$ the *weak dependency pair* of $l \to r$.

(2) The *weak dependency pairs* $\mathsf{WDP}(\mathcal{R})$ of a TRS $\mathcal{R}$ are given by

$$\mathsf{WDP}(\mathcal{R}) := \{\mathsf{WDP}(l \to r) \mid l \to r \in \mathcal{R}\} \;.$$

Note that the choice of the compound symbols used in $\mathsf{WDP}(\mathcal{R})$ is arbitrary.[1]

Example 14.22 (Continued from Example 14.1). Consider the TRS $\mathcal{R}_{\mathsf{mult}}$ given in Example 14.1. Then $\mathsf{WDP}(\mathcal{R}_{\mathsf{mult}})$ consists of the following four rules:

$$\begin{array}{llll} \texttt{78:} & 0 +^\sharp y \to \mathsf{c}_0 & \texttt{79:} & \mathsf{s}(x) +^\sharp y \to x +^\sharp y \\ \texttt{80:} & 0 \times^\sharp y \to \mathsf{c}_0 & \texttt{81:} & \mathsf{s}(x) \times^\sharp y \to x +^\sharp (x \times y) \;. \end{array} \qquad \lhd$$

In [38] it is shown that for any term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$,

$$\mathsf{dh}(t, \to_{\mathcal{R}}) =_{\mathsf{k}} \mathsf{dh}(t^\sharp, \to_{\mathsf{WDP}(\mathcal{R}) \cup \mathcal{R}}) \;.$$

We extend this result to our setting, where the following lemma serves as a preparatory step.

Lemma 14.23. *Let $\mathcal{R}$ and $\mathcal{Q}$ be two TRSs, such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$. Then every derivation*

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

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

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

and vice versa.

Proof. For a term s, let $P(s) \subseteq \mathcal{P}\mathrm{os}(s)$ be the set of minimal positions p such that the symbol occurring in s at position p is either a defined symbol or a variable. Notice that all positions in $P(s)$ are parallel. Call a term $u = C[s_1, \ldots, s_n]$ *good for* s if C is a context containing only constructors and compound symbols, and there exists an injective mapping $m : P(s) \to \mathcal{P}\mathrm{os}(C)$ that associates with every $\mathcal{R}$ redex $s|_p$ a possibly marked $\mathsf{WDP}(\mathcal{R}) \cup \mathcal{R}$ redex $u|_{m(p)}$: for all $p \in P(s)$, $m(p) \in \mathcal{P}\mathrm{os}(C)$ with $C|_{m(p)} = \Box$ and $u|_{m(p)} = s|_p$ or $u|_{m(p)} = (s|_p)^\sharp$.

[1] In our implementation, we have chosen to assign for each rule a fresh compound symbol.

Consider $s \xrightarrow{\mathcal{Q}}_{l\to r,p} t$ for $l \to r \in \mathcal{R}$, and suppose $\mathsf{WDP}(l \to r) = l^\sharp \to \textsc{com}(r_1^\sharp, \ldots, r_m^\sharp)$. We show that for every term u good for $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, there exists a term v with

$$u \xrightarrow{\mathcal{Q}}_{\{\mathsf{WDP}(l\to r), l\to r\}} v \ ,$$

that is good for t. This establishes the simulation from left to right. Suppose $u = C[s_1, \ldots, s_n]$ is good for s as witnessed by the mapping $m : P(s) \to \mathcal{P}\mathrm{os}(C)$ and C of the required form. Let p' be a prefix of the rewrite position p with $p' \in P(s)$. This position exists, as the root of $s|_p$ is defined. Let $s_i = u|_{m(p')}$ be the possible marked occurrence of $s|_{p'}$ in u. We distinguish three cases.

Consider first the case $p' < p$. Then $s|_{p'} \xrightarrow{\mathcal{Q}}_{l\to r,>\varepsilon} t|_{p'}$ by assumption. The latter implies

$$u = C[s_1, \ldots, s_i, \ldots, s_n] \xrightarrow{\mathcal{Q}}_{l\to r} C[s_1, \ldots, t_i, \ldots, s_n] =: v \ ,$$

for s_i and t_i the possibly marked versions of $s|_{p'}$ and $t|_{p'}$ respectively. By assumption $p' \in P(s)$ the root of $t|_{p'}$, namely the root of $s|_{p'}$, is defined. From this it is not difficult to see that $P(s) = P(t)$ and $m : P(s) \to \mathcal{P}\mathrm{os}(C)$ witnesses that v is good for t.

Next consider that $p' = p$ and $u|_{m(p)}$ is not marked. by assumption thus $u|_{m(p)} = l\sigma$ for σ a substitution such that arguments of $l\sigma$ are $\mathcal{Q}$ normal forms. Hence

$$u = C[s_1, \ldots, l\sigma, \ldots, s_n] \xrightarrow{\mathcal{Q}}_{l\to r} C[s_1, \ldots, r\sigma, \ldots, s_n] =: v \ .$$

We claim v is good for t. Let $\{q_1, \ldots, q_k\} = P(r\sigma)$ and denote by C_r the context of $r\sigma$ with holes at positions $P(r\sigma)$. Set $C' := C[\Box, \ldots, C_r[\Box, \ldots, \Box], \ldots, \Box]$ such that $v = C'[s_1, \ldots, (r\sigma)|_{q_1}, \ldots, (r\sigma)|_{q_k}, \ldots, s_n]$. By construction, C' contains only constructors or compound symbols. Exploiting the mapping $m : P(s) \to \mathcal{P}\mathrm{os}(C)$ witnessing that u is good for s, it is not difficult to extend this to an injective function $m' : P(t) \to \mathcal{P}\mathrm{os}(C')$ witnessing that v is good for t.

Consider the final case $p' = p$ but $u|_{m(p)}$ marked. We proceed as above, but use the reduction

$$u = C[s_1, \ldots, s_i, \ldots, s_n] \xrightarrow{\mathcal{Q}}_{\mathsf{WDP}(l\to r)} C[s_1, \ldots, \textsc{com}(r_1^\sharp\sigma, \ldots, r_m^\sharp\sigma), \ldots, s_n] =: v \ ,$$

instead. For the context C_r in v, we use the maximal context of the term $\textsc{com}(r_1^\sharp\sigma, \ldots, r_m^\sharp\sigma)$ which contains only compound symbols or constructors. To see that v is good for t, one then uses that $r_1\sigma, \ldots, r_m\sigma$ contain all maximal occurrences of sub-terms of $s|_p$ which are variables or have a defined root symbol. This completes the proof of the direction from left to right.

For the direction from right to left, consider a term $u = C[s_1, \ldots, s_n]$ where C is a compound context, and s_i $(i = 1, \ldots, n)$ denote possibly marked terms without compound symbols. Call a term $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *good for* u if s is obtained from u by unmarking symbols, and replacing C with a context consisting only of constructors. By case analysis on $u \xrightarrow{\mathcal{Q}}_{\mathsf{WDP}(\mathcal{R}^\sharp)\cup\mathcal{R}} v$, it can be verified that for any such u if s is good for u, then there exists a term t with $s \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t$ that is good for v. Since the starting term $t^\sharp$ is trivially of the considered shape, the simulation follows. □

Theorem 14.24 (Weak Dependency Pair Processor)**.** *Let* $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T} \rangle$ *denote a runtime complexity problem. The following processor is sound and complete.*

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

Proof. Set $\mathcal{P}^\sharp := \langle \mathsf{WDP}(\mathcal{S}) \cup \mathcal{S}/\mathsf{WDP}(\mathcal{W}) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$. Suppose first $\mathsf{cp}_{\mathcal{P}^\sharp} \in \mathcal{O}(f(n))$. Lemma 14.23 shows that every $\to_{\mathcal{P}}$ reduction of $t \in \mathcal{T}$ is simulated by a corresponding $\to_{\mathcal{P}^\sharp}$ reduction starting from $t^\sharp \in \mathcal{T}^\sharp$. For the application of Lemma 14.23, notice that since $\mathcal{P}$ is a runtime complexity problem, neither $\mathcal{S}$ nor $\mathcal{W}$ define constructors, i.e., $\mathcal{D}_\mathcal{S} \cup \mathcal{D}_\mathcal{W} \subseteq \mathcal{D}$. Observe that every $\xrightarrow[\mathcal{Q}]{}_\mathcal{S}$ step in the considered derivation is simulated by a $\xrightarrow[\mathcal{Q}]{}_{\mathsf{WDP}(\mathcal{S})\cup\mathcal{S}}$ step. We thus obtain $\mathsf{cp}_\mathcal{P} \in \mathcal{O}(f(n))$. This proves soundness, completeness is obtained dual. □

Observe that the generated sub-problem is a DP complexity problem as given by Definition 14.17. Notice also that when the input is an innermost complexity problem, then so is the obtained DP problem.

Example 14.25 (Continued from Example 14.1 and 14.22). Reconsider the TRS $\mathcal{R}_\mathsf{mult}$ given in Example 14.1, together with $\mathsf{WDP}(\mathcal{R}_\mathsf{mult})$ depicted in Example 14.22. According to the weak dependency pair processor, the inference

$$\frac{\vdash \langle \mathsf{WDP}(\mathcal{R}_\mathsf{mult}) \cup \mathcal{R}_\mathsf{mult}/\varnothing, \varnothing, \mathcal{T}^\sharp_\mathsf{mult} \rangle \colon n^2}{\vdash \langle \mathcal{R}_\mathsf{mult}/\varnothing, \varnothing, \mathcal{T}_\mathsf{mult} \rangle \colon n^2}\ \mathsf{WDP}\ .$$

is sound and complete. ◁

The change in signature often makes the generated sub-problem easier to analyse. In particular, the generated sub-problem is amendable to many of the processors suited for dependency pair problems introduced below.

14.4.3. Dependency Tuples

Consider a DP problem of the form $\langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$. The analysis of this problem requires only an estimation of applications of DPs, which are applied in compound contexts only. This property makes the analysis considerably simpler. Some processors tailored for DP problems are even sound only in this setting, for instance *(safe) reduction pairs* (cf. Definition 14.32 below) and various syntactic simplifications proposed in Section 14.5.

In contrast to weak dependency pairs, *dependency tuples* [63] allow the translation of an *innermost* runtime complexity problem directly into a DP problem of this simpler form. This however comes at the expense of completeness. Also, a more complicated set of dependency pairs is required.

Definition 14.26 (Dependency Tuples [63])**.** Let $\mathcal{R}$ denote a TRS such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$.

(1) Consider a rule $l \to r$ in $\mathcal{R}$, and let $r_1, \dots, r_k$ denote *all* sub-terms of the right-hand side whose root symbol is in $\mathcal{D}$. We define

$$\mathsf{DT}(l \to r) := l^\sharp \to \textsc{com}(r_1^\sharp, \dots, r_k^\sharp) \;,$$

and call $\mathsf{DT}(l \to r)$ the *dependency tuple* of $l \to r$.

(2) The *dependency tuples* $\mathsf{DT}(\mathcal{R})$ of a TRS $\mathcal{R}$ are given by

$$\mathsf{DT}(\mathcal{R}) := \{\mathsf{DT}(l \to r) \mid l \to r \in \mathcal{R}\} \;.$$

Example 14.27 (Continued from Example 14.18). The four DPs (74)–(77) depicted in Example 14.18 constitute the dependency tuples of $\mathcal{R}_{\mathsf{mult}}$ from Example 14.1. ◁

The central theorem of [63] states that dependency tuples are sound for innermost runtime complexity analysis. We extend this result to a relative setting.

Lemma 14.28. *Let $\mathcal{R}$ and $\mathcal{Q}$ be two TRSs, such that the defined symbols of $\mathcal{R}$ are included in $\mathcal{D}$, and such that* $\mathrm{NF}(\mathcal{Q}) \subseteq \mathrm{NF}(\mathcal{R})$. *Then every derivation*

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

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

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

Proof. The proof follows the pattern of the proof of Lemma 14.23. Define $P(s)$ as the restriction of $\mathcal{P}\mathrm{os}(s)$ to redexes in s, more precise let $P(s)$ collect all positions p which satisfy $s|_p \xrightarrow{\mathcal{Q}}_{\mathcal{R}} u$ for some term u. Call a term $u = C[s_1, \dots, s_n]$ *good for* s if C is a context containing only constructors and compound symbols, and there exists an injective function $m : P(s) \to \mathcal{P}\mathrm{os}(C)$ such that $u|_{m(p)} = (s|_p)^\sharp$ for every position $p \in \mathcal{P}\mathrm{os}(s)$ holds. For terms t, let $\mathcal{P}\mathrm{os}_{\mathcal{D}}(t) \subseteq \mathcal{P}\mathrm{os}(t)$ denote the set of all positions p such that the root of the sub-term $r|_p$ is a defined symbol from $\mathcal{D}$.

Consider a rewrite step $s = C[l\sigma] \xrightarrow{\mathcal{Q}}_{l\to r,p} C[r\sigma] = t$ for position p, context C, substitution σ and rewrite rule $l \to r \in \mathcal{R}$. Observe that $P(t) \subseteq (P(s) \setminus \{p\}) \cup \{p{\cdot}q \mid q \in \mathcal{P}\mathrm{os}_{\mathcal{D}}(r)\}$. This holds as the substitution σ maps variables to $\mathrm{NF}(\mathcal{Q}) \subseteq \mathrm{NF}(\mathcal{R})$.

We now show that if $u = C[s_1, \dots, s_n]$ is good for s, then $u \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(\mathcal{R})/\mathcal{R}} v$ holds for some term v good for t. Set $l^\sharp \to \textsc{com}(r_1^\sharp, \dots, r_n^\sharp) := \mathsf{DT}(l \to r)$. Using that $p \in P(s)$,

$$u = C[s_1, \dots, l^\sharp\sigma, \dots, s_n] \xrightarrow{\mathcal{Q}}_{\mathsf{DT}(l\to r)} C[s_1, \dots, \textsc{com}(r_1^\sharp\sigma, \dots, r_m^\sharp\sigma), \dots, s_n] =: v' \;,$$

holds, where

$$C' = C[\Box, \dots, \textsc{com}(\Box, \dots, \Box), \dots, \Box] \text{, and}$$
$$v' = C'[s_1, \dots, r_1^\sharp\sigma, \dots, r_m^\sharp\sigma, \dots, s_n] \;.$$

We verify that $v' \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} v$ for some v good for t. Recall that by the observation on $P(t)$, every position $q \in P(t) \setminus P(s)$ can be decomposed into positions p and $q_i \in \mathcal{P}\mathrm{os}_{\mathcal{D}}(r)$ with $q = p{\cdot}q_i$ such that $r|_{q_i} = r_i$ for $i = 1, \dots, m$. Let q'_i denote the position of the occurrence $r_i^\sharp$ in the right-hand side $\mathrm{COM}(r_1^\sharp, \dots, r_m^\sharp)$, and set $m(q) := m(p){\cdot}q'_i$. Note that the resulting function is an injective function from $P(t)$ to $\mathcal{P}\mathrm{os}(C')$. By construction we have $v'|_{m(q)} = r_i^\sharp\sigma = (t|_q)^\sharp$ for all positions $q = p{\cdot}q_i \in P(t) \setminus P(s)$. For q not of this shape we have $q \in P(s) \setminus \{p\}$ by the observation on $P(t)$. Hence either $(s|_q)^\sharp = (t|_q)^\sharp$ or otherwise $s|_q \neq t|_q$ and the assumption $q \neq p$ gives $q < p$. It follows that $(t|_q)^\sharp \xrightarrow{\mathcal{Q}}{}^*_{l\to r,>\epsilon} (t|_q)^\sharp$. Rewriting in v' all terms $v'|_{m(q)} = (s|_q)^\sharp$ with $q \in P(s) \setminus \{p\}$ to $(t|_q)^\sharp$ gives the desired term v good for t. □

Theorem 14.29 (Dependency Tuple Processor)**.** *Let $\mathcal{P} = \langle \mathcal{S}/\mathcal{W}, \mathcal{Q}, \mathcal{T}\rangle$ denote an* innermost *runtime complexity problem. The following processor is sound.*

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

Proof. The theorem follows by reasoning identical to Theorem 14.24, using Lemma 14.28. □

We emphasise that for $\mathcal{W} = \varnothing$, Theorem 14.29 corresponds to [63, Theorem 10]. The following example clarifies that the restriction to innermost rewriting is necessary.

Example 14.30 (Continued from Example 9.12). Reconsider the rewrite system $\mathcal{R}_{\mathsf{dup}}$ given in Example 9.12 on page 94, which served as a counter example to soundness of small polynomial path orders for full rewriting. This TRS admits exponential long outermost reductions, obtained by successively duplicating redexes. The dependency tuples $\mathsf{DT}(\mathcal{R}_{\mathsf{dup}})$ are given by the three rules

$$\begin{aligned} &82\colon\ \mathsf{btree}^\sharp(0) \to \mathsf{c}_0 && 83\colon\ \mathsf{dup}^\sharp(;t) \to \mathsf{c}_0\\ &84\colon\ \mathsf{btree}^\sharp(\mathsf{s}(n)) \to \mathsf{c}_2(\mathsf{dup}^\sharp(\mathsf{btree}(n)), \mathsf{btree}^\sharp(n))\ . \end{aligned}$$

It is not difficult to see that in a $\mathsf{DT}(\mathcal{R}_{\mathsf{dup}}) \cup \mathcal{R}_{\mathsf{dup}}$ derivation starting from $\mathsf{btree}^\sharp(\mathbf{n})$ $(n \in \mathbb{N})$, the overall number of applications of a dependency pair is bounded linearly in n, i.e., the judgement $\vdash \langle \mathsf{DT}(\mathcal{R}_{\mathsf{dup}})/\mathcal{R}_{\mathsf{dup}}, \varnothing, \mathcal{T}_{\mathsf{b}}^\sharp\rangle \colon n$ holds. Permitting the inference

$$\frac{\vdash \langle \mathsf{DT}(\mathcal{R}_{\mathsf{dup}})/\mathcal{R}_{\mathsf{dup}}, \varnothing, \mathcal{T}_{\mathsf{b}}^\sharp\rangle \colon n}{\vdash \langle \mathcal{R}_{\mathsf{dup}}/\varnothing, \varnothing, \mathcal{T}_{\mathsf{b}}\rangle \colon n}$$

would allow us to wrongly deduce that the runtime complexity of $\mathcal{R}_{\mathsf{dup}}$ is linear.◁

Example 14.31 (Continued from Example 14.2). The following inference starts the proof of quadratic innermost runtime complexity of the TRS $\mathcal{R}_{\mathsf{K}}$ given in

Example 14.2. We transform its canonical innermost complexity problem into a DP problem using dependency tuples (Theorem 14.29).

$$\frac{\vdash \langle \mathcal{S}^\sharp_\mathsf{K}/\mathcal{R}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle \colon n^2}{\vdash \langle \mathcal{R}_\mathsf{K}/\varnothing, \mathcal{R}_\mathsf{K}, \mathcal{T}_\mathsf{K}\rangle \colon n^2}\ \mathsf{DT}\ .$$

Here $\mathcal{S}^\sharp_\mathsf{K} := \mathsf{DT}(\mathcal{R}_\mathsf{K})$ consists of the following rules.

85: $\mathsf{src}^\sharp((n,w,m)) \to \mathsf{c}_0$ 86: $\mathsf{wt}^\sharp((n,w,m)) \to \mathsf{c}_0$ 87: $\mathsf{trg}^\sharp((n,w,m)) \to \mathsf{c}_0$

88: $\mathsf{forest}^\sharp(\mathsf{graph}(N,E)) \to \mathsf{c}_3(\mathsf{kruskal}^\sharp(\mathsf{sort}(E),[\,],\mathsf{partitions}(N)), \mathsf{sort}^\sharp(E), \mathsf{partitions}^\sharp(N))$

89: $\mathsf{partitions}^\sharp([\,]) \to \mathsf{c}_0$

90: $\mathsf{partitions}^\sharp(n :: N) \to \mathsf{partitions}^\sharp(N)$

91: $\mathsf{kruskal}^\sharp([\,],W,P) \to \mathsf{c}_0$

92: $\mathsf{kruskal}^\sharp(e :: E,W,P) \to \mathsf{c}_2(\mathsf{kruskal?}^\sharp(\mathsf{inBlock}(e,P),e,E,W,P), \mathsf{inBlock}^\sharp(e,P))$

93: $\mathsf{kruskal?}^\sharp(\mathsf{tt},e,E,W,P) \to \mathsf{kruskal}^\sharp(E,W,P)$

94: $\mathsf{kruskal?}^\sharp(\mathsf{ff},e,E,W,P) \to \mathsf{c}_2(\mathsf{kruskal}^\sharp(E,e :: W,\mathsf{join}(e,P,[\,])), \mathsf{join}^\sharp(e,P,[\,]))$

95: $\mathsf{inBlock}^\sharp(e,[\,]) \to \mathsf{c}_0$

96: $\mathsf{inBlock}^\sharp(e,p :: P) \to \mathsf{c}_7((\mathsf{src}(e) \in p \wedge \mathsf{trg}(e) \in p) \vee^\sharp \mathsf{inBlock}(e,P), \mathsf{src}(e) \in p \wedge^\sharp \mathsf{trg}(e) \in p, \mathsf{src}(e) \in^\sharp p, \mathsf{trg}(e) \in^\sharp p, \mathsf{src}^\sharp(e), \mathsf{trg}(e)^\sharp, \mathsf{inBlock}^\sharp(e,P))$

97: $\mathsf{join}^\sharp(e,[\,],q) \to \mathsf{c}_0$

98: $\mathsf{join}^\sharp(e,p :: P,q) \to \mathsf{c}_6(\mathsf{join?}^\sharp(\mathsf{src}(e) \in p \vee \mathsf{trg}(e) \in p,e,p,P,q), \mathsf{src}(e) \in p \vee^\sharp \mathsf{trg}(e) \in p, \mathsf{src}(e) \in^\sharp p, \mathsf{trg}(e) \in^\sharp p, \mathsf{src}^\sharp(e), \mathsf{trg}(e)^\sharp)$

99: $\mathsf{join?}^\sharp(\mathsf{tt},e,p,P,q) \to \mathsf{c}_2(\mathsf{join}^\sharp(e,P,p +\!\!+ q), p +\!\!+^\sharp q)$

100: $\mathsf{join?}^\sharp(\mathsf{ff},e,p,P,q) \to \mathsf{join}^\sharp(e,P,q)$

101: $\mathsf{sort}^\sharp([\,]) \to \mathsf{c}_0$

102: $\mathsf{sort}^\sharp(e :: E) \to \mathsf{c}_2(\mathsf{insert}^\sharp(e,\mathsf{sort}(E)),\mathsf{sort}^\sharp(E))$

103: $\mathsf{insert}^\sharp(e,[\,]) \to \mathsf{c}_0$

104: $\mathsf{insert}^\sharp(e,f :: E) \to \mathsf{c}_4(\mathsf{insert?}^\sharp(\mathsf{wt}(e) \leqslant \mathsf{wt}(f),e,f,E), \mathsf{wt}(e) \leqslant^\sharp \mathsf{wt}(f), \mathsf{wt}^\sharp(e), \mathsf{wt}^\sharp(f))$

105: $\mathsf{insert?}^\sharp(\mathsf{tt},e,f,E) \to \mathsf{c}_0$

106: $\mathsf{insert?}^\sharp(\mathsf{ff},e,f,E) \to \mathsf{insert}^\sharp(e,E)$

107: $n \in^\sharp [\,] \to \mathsf{c}_0$

$$
\begin{array}{rll}
108: & n \in^\sharp (m :: p) \to \mathsf{c}_3(n = m \vee^\sharp n \in p, n =^\sharp m, n \in^\sharp p) & \\
109: & [\,] +\!\!+^\sharp q \to \mathsf{c}_0 & \\
110: & (n :: p) +\!\!+^\sharp q \to p +\!\!+^\sharp q & \\
111: & 0 =^\sharp 0 \to \mathsf{c}_0 & 112: \quad \mathsf{s}(x) =^\sharp 0 \to \mathsf{c}_0 \\
113: & 0 =^\sharp \mathsf{s}(y) \to \mathsf{c}_0 & 114: \quad \mathsf{s}(x) =^\sharp \mathsf{s}(y) \to x =^\sharp y \\
115: & 0 \leqslant^\sharp 0 \to \mathsf{c}_0 & 116: \quad \mathsf{s}(x) \leqslant^\sharp 0 \to \mathsf{c}_0 \\
117: & 0 \leqslant^\sharp \mathsf{s}(y) \to \mathsf{c}_0 & 118: \quad \mathsf{s}(x) \leqslant^\sharp \mathsf{s}(y) \to x \leqslant^\sharp y \\
119: & \mathsf{ff} \wedge^\sharp \mathsf{ff} \to \mathsf{c}_0 & 120: \quad \mathsf{ff} \wedge^\sharp \mathsf{tt} \to \mathsf{c}_0 \\
121: & \mathsf{tt} \wedge^\sharp \mathsf{ff} \to \mathsf{c}_0 & 122: \quad \mathsf{tt} \wedge^\sharp \mathsf{tt} \to \mathsf{c}_0 \\
123: & \mathsf{ff} \vee^\sharp \mathsf{ff} \to \mathsf{c}_0 & 124: \quad \mathsf{ff} \vee^\sharp \mathsf{tt} \to \mathsf{c}_0 \\
125: & \mathsf{tt} \vee^\sharp \mathsf{ff} \to \mathsf{c}_0 & 126: \quad \mathsf{tt} \vee^\sharp \mathsf{tt} \to \mathsf{c}_0\,. \qquad \triangleleft
\end{array}
$$

14.4.4. Reduction Pairs

Reduction pairs were introduced in the context of termination analysis [5]. *Safe reduction pairs* [38], aka COM-*monotone* reduction pairs [63], constitute a variation that accounts for compound symbols in complexity problems.

Definition 14.32 (Safe Reduction Pair [38])**.** A *reduction pair* $(\succsim, \succ)$ consists of a rewrite preorder $\succsim$ and a compatible well-founded order $\succ$ which is closed under substitutions. Here compatibility means that the inclusion $\succsim \cdot \succ \cdot \succsim \subseteq \succ$ holds.

The reduction pair $(\succsim, \succ)$ is called *safe* if the order $\succ$ is monotone in all coordinates on compound symbols, i.e., for every $\mathsf{c}_k \in \mathcal{C}\mathsf{om}$

$$\mathsf{c}_k(s_1, \ldots, s_i, \ldots, s_k) \succ \mathsf{c}_k(s_1, \ldots, t, \ldots, s_k)\,,$$

for all $i = 1, \ldots, k$ and terms $s_1, \ldots, s_k, t$, with $s_i \succ t$.

Proposition 14.33 ([38])**.** *Let $(\succsim, \succ)$ be a safe reduction pair, let $\mathcal{S}^\sharp$ be a set of weak dependency pairs and let $\mathcal{W}$ be a rewrite system. If $\mathcal{S}^\sharp \subseteq \succ$ and $\mathcal{W} \subseteq \succsim$ then $\mathsf{dh}(t, \to_{\mathcal{S}^\sharp/\mathcal{W}}) \leqslant f(t)$ where $f : \mathbb{N} \to \mathbb{N}$ is the complexity induced by $\succ$.*

The above proposition refers to the application of safe reduction pairs in the main theorem of Hirokawa and Moser [38]. Together with the following simple observation, this proposition is a straight forward consequence of our complexity pair processor (Theorem 14.10)

Lemma 14.34. *Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ be a DP problem such that the strict component contains no rewrite rules.*

(1) *Suppose μ denotes a usable replacement map for dependency pairs $\mathcal{R}^\sharp$ in $\mathcal{P}^\sharp$. Then μ_{COM} is a usable replacement map for $\mathcal{R}^\sharp$ in $\mathcal{P}^\sharp$. Here μ_{COM} denotes the restriction of μ to compound symbols in the following sense: $\mu_{\text{COM}}(\mathsf{c}_n) := \mu(\mathsf{c}_n)$ for all $\mathsf{c}_n \in \mathcal{C}\mathsf{om}$, and otherwise $\mu_{\text{COM}}(f) := \varnothing$ for $f \in \mathcal{F}^\sharp$.*

(2) *If* $(\succsim, \succ)$ *denotes a safe reduction pair, then it is also a* $\mathcal{P}$*-monotone complexity pair.*

Proof. The first assertion is a straight forward consequence of Lemma 14.20. From this one derives that reduction pairs are $\mathcal{P}$-monotone, and thus the second assertion holds. □

14.4.5. Derivation Trees

Consider a term $t \in \to_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)$, for a DP problem $\mathcal{P}^\sharp$ with starting terms $\mathcal{T}^\sharp$. Then $t = C[t_1, \ldots, t_n] \in \mathcal{T}^\sharp_\to$ for some compound context C. Any reduction of t consists of possibly interleaved, but otherwise *independent*, reductions of the terms $t_1, \ldots, t_n$. To avoid reasoning up to permutations of rewrite steps, we introduce a notion of *derivation tree* that disregards the order of parallel steps under compound contexts.

Definition 14.35 (Hypergraph)**.**

(1) A (directed) *hypergraph* over *labels* $\mathcal{L}$ is a triple $G = (N, E, \mathsf{lab})$ where N is a set of *nodes*, $E \subseteq N \times \mathcal{P}(N)$ a set of *edges*, and $\mathsf{lab} : N \cup E \to \mathcal{L}$ a *labeling function.*

For $e = \langle u, \{v_1, \ldots, v_n\}\rangle \in E$ we call the node u the *source*, and nodes $v_1, \ldots, v_n$ the *targets* of e.

(2) We denote by $\rightharpoonup_G$ the *successor relation* in the hypergraph G, i.e., $u \rightharpoonup_G v$ if there exists an edge $e = \langle u, \{v_1, \ldots, v_n\}\rangle \in E$ with $v \in \{v_1, \ldots, v_n\}$. We set $u \xrightarrow{\mathcal{K}}_G v$ for labels $\mathcal{K} \subseteq \mathcal{L}$ if additionally $\mathsf{lab}(e) \in \mathcal{K}$ holds, and abbreviate $\xrightarrow{\{l\}}_G$ by $\xrightarrow{l}_G$. If there exists a *path* $u = w_1 \rightharpoonup_G \cdots \rightharpoonup_G w_n = v$ we say that v is *reachable* from u in G.

(3) We call a hypergraph G a *hypertree* (*tree* for short) if there exists a unique node $u \in N$, the *root* of G, such that every $v \in N$ is reachable from u by a unique path.

We keep the convention that every node is the source of at most one edge.

Definition 14.36 (Derivation Tree)**.** Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ denote a dependency pair problem.

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

(i) $T \in \mathsf{DTree}_{\mathcal{P}^\sharp}(t)$ where T consists of a unique node labeled by t.

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

(2) For a $\mathcal{P}^\sharp$ derivation tree T we denote by $|T|_{\mathcal{R}^\sharp \cup \mathcal{R}}$ the edges labeled by a rule or dependency pair $l \to r \in \mathcal{R}^\sharp \cup \mathcal{R}$.

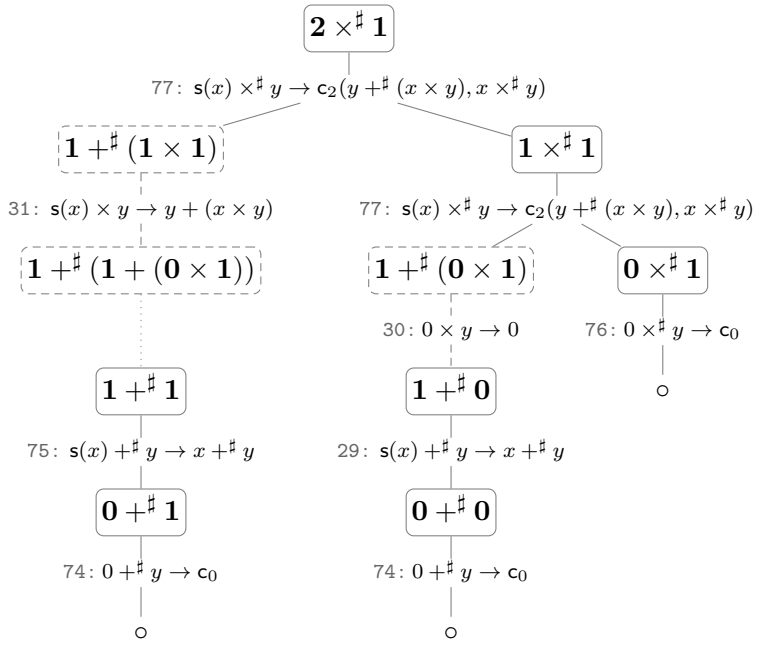


Figure 14.2.: $\mathcal{P}^\sharp_{\mathsf{mult}}$ derivation tree T_{mult} of $\mathbf{2} \times^\sharp \mathbf{1}$.

Consider a $\mathcal{P}^\sharp$ derivation tree T. Note that every edge $\langle u, \{v_1, \ldots, v_n\}\rangle$ in T *corresponds* to a rewrite steps $t \xrightarrow{\mathcal{Q}}_{l \to r} \textsc{com}(t_1, \ldots, t_n)$, with t and $t_1, \ldots, t_n$ precisely the label of source u and targets $v_1, \ldots, v_n$ respectively, and $l \to r$ the label of the considered edge. We also say that $l \to r$ was *applied* at node u in T. It is not difficult to lift this correspondence to rewrite sequence of terms $t^\sharp \in \mathcal{T}^\sharp$. This motivates our notion of the size $|T|_{\mathcal{R}^\sharp \cup \mathcal{R}}$ of T with respect to $\mathcal{R}^\sharp \cup \mathcal{R}$: $|T|_{\mathcal{R}^\sharp \cup \mathcal{R}}$ refers to the number applications of rewrite rules and dependency pairs $l \to r \in \mathcal{R}^\sharp \cup \mathcal{R}$. This correspondence leads to the characterisation of the complexity function of DP problems $\mathcal{P}^\sharp$ given in Lemma 14.38 below. The next example illustrates this.

Example 14.37 (Continued from Example 14.18). In Figure 14.2 we depict a $\mathcal{P}^\sharp_{\mathsf{mult}}$ derivation tree T_{mult} of the term $\mathbf{2} \times^\sharp \mathbf{1}$. Solid nodes indicate applications of DPs from the strict component $\mathcal{S}^\sharp_{\mathsf{mult}}$. Conversely, dashed nodes indicate applications of weak rules. The dotted lines indicate that we left out some rewrite steps with respect to $\mathcal{R}_{\mathsf{mult}}$. ◁

Lemma 14.38. *Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ be a DP problem. Then for every $t \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathcal{T}^\sharp(\mathcal{F}, \mathcal{V})$, we have*

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

In particular,

$$\mathsf{cp}_{\mathcal{P}^\sharp}(n) =_{\mathsf{k}} \max\{|T|_{\mathcal{S}^\sharp \cup \mathcal{S}} \mid T \text{ is a } \mathcal{P}^\sharp\text{-derivation tree of } t \in \mathcal{T}^\sharp \text{ with } |t| \leqslant n\} \ ,$$

holds.

Proof. We consider the first assertion. Let $t \in \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{T}^\sharp(\mathcal{F},\mathcal{V})$, and abbreviate

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

We show $\ell \geqslant_{\mathsf{k}} s$ and $s \geqslant_{\mathsf{k}} \ell$. For the first inequality, suppose s is well-defined. Hence there exists a $\mathcal{P}^\sharp$ derivation tree T of t with $|T|_{\mathcal{S}^\sharp \cup \mathcal{S}} = s$. A breath first traversal on T constructs a $\mathcal{P}^\sharp$ derivation D, such that every application of $l \to r \in \mathcal{S}^\sharp \cup \mathcal{S}$ translates to an application of $l \to r$ in D. This derivation D then witnesses $\ell \geqslant_{\mathsf{k}} s$.

Inversely, for the second equality one can construct for an arbitrary $\mathcal{P}^\sharp$ derivation D starting from $t \in \mathcal{T}^\sharp$ a $\mathcal{P}^\sharp$ derivation tree T. Every application of a rule $l \to r \in \mathcal{P}^\sharp$ in D translates to a unique edge in T. The construction is carried out by induction on the length of D. One uses that the final term in this derivation is of the form $C[u_1,\dots,u_n] \in \mathcal{T}^\sharp_\to$ for C a maximal compound context (compare Lemma 14.20). Note also that for each sub-term u_i $(i = 1,\dots,n)$, the constructed tree T contains a dedicated leaf labeled by u_i. This shows $s \geqslant_{\mathsf{k}} \ell$.□

Remark. Our notion of derivation trees is related to the notion of *chain tree* given by Noschinski et.al. [63]. Whereas in $\mathcal{P}^\sharp$ derivation trees each edge corresponds to a single rewrite step, in chain trees an edge corresponds to a dependency pair step $l^\sharp\sigma \to \textsc{com}(r_1^\sharp\sigma,\dots,r_k^\sharp\sigma)$ followed by normalisation of unmarked subterms in the reduct. This notion is of limited use in our setting, as we also want to account for the normalisation steps in Lemma 14.38.

14.4.6. Dependency Graphs for Complexity Analysis

The notion of *dependency graph*, a form of call and data flow graph, was initially proposed for the termination analysis [5]. We adapt this notion to complexity problems, compare also [39].

Definition 14.39 (Dependency Graph)**.** Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ denote a DP problem.

(1) The nodes of the *dependency graph* (*DG* for short) $\mathcal{G}$ of $\mathcal{P}^\sharp$ are the dependency pairs from $\mathcal{S}^\sharp \cup \mathcal{W}^\sharp$, and there is an arrow labeled by $i \in \mathbb{N}$ from $s^\sharp \to \textsc{com}(t_1^\sharp,\dots,t_n^\sharp)$ to $u^\sharp \to \textsc{com}(v_1^\sharp,\dots,v_m^\sharp)$ if for some substitutions $\sigma,\tau : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V})$, $t_i^\sharp\sigma \xrightarrow{\mathcal{Q}}^*_{\mathcal{S}\cup\mathcal{W}} u^\sharp\tau$ holds.

(2) A graph $\mathcal{G}$ is called an *approximated dependency graph* for $\mathcal{P}^\sharp$, if it is a sub-graph of the dependency graph of $\mathcal{P}^\sharp$ in the following sense. The nodes of $\mathcal{G}$ are the dependency pairs from $\mathcal{S}^\sharp \cup \mathcal{W}^\sharp$, and whenever there is an arrow labeled by $i \in \mathbb{N}$ from $s^\sharp \to \textsc{com}(t_1^\sharp,\dots,t_n^\sharp)$ to $u^\sharp \to \textsc{com}(v_1^\sharp,\dots,v_m^\sharp)$ in the dependency graph of $\mathcal{P}^\sharp$, then this arrow occurs also in $\mathcal{G}$.

The dependency graph is not computable in general, however it is well understood how approximated dependency graphs can be computed [76].

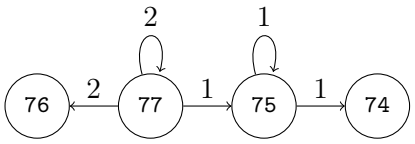


Figure 14.3.: Dependency Graph of $\mathcal{P}^\sharp_{\mathsf{mult}}$.

Example 14.40 (Continued from Example 14.18). In Figure 14.3 we depict the dependency graph of the DP problem $\mathcal{P}^\sharp_{\mathsf{mult}} = \langle \{74\text{–}77\}/\mathcal{R}_{\mathsf{mult}}, \mathcal{R}_{\mathsf{mult}}, \mathcal{T}^\sharp_{\mathsf{mult}} \rangle$, where the nodes (74)–(77) refer to the DPs given in Example 14.18. ◁

The dependency graph $\mathcal{G}$ indicates in which order dependency pairs can occur in a derivation tree of $\mathcal{P}^\sharp$. To make this intuition precise, we adapt the notion of *DP chain* known from termination analysis. Recall that for a derivation tree T, $\rightharpoonup_T$ denotes the successor relation, and $\xrightarrow{\mathcal{R}}_T$ its restriction to edges labeled by $l \to r \in \mathcal{R}$.

Definition 14.41 (Dependency Pair Chain)**.** Let T be a derivation tree, and consider a path

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

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

Example 14.42 (Continued from Example 14.40). Reconsider the $\mathcal{P}^\sharp_{\mathsf{mult}}$ derivation tree T_{mult} given in Figure 14.2. This tree gives rise to three maximal chains: 77,75,74; 77,77,75,74 and 77,77,76. ◁

Lemma 14.43. *Every chain in a $\mathcal{P}^\sharp$ derivation tree is a path in the dependency graph of the DP problem $\mathcal{P}^\sharp$.*

Proof. Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ and consider two successive elements $l_1 \to r_1 := s^\sharp \to \textsc{com}(t^\sharp_1, \ldots, t^\sharp_n)$ and $l_2 \to r_2 := u^\sharp \to \mathsf{c}_m(v^\sharp_1, \ldots, v^\sharp_m)$ in a dependency pair chain of a $\mathcal{P}^\sharp$ derivation tree T. Thus there exists nodes u_1, u_2, v_1, v_2 with

$$u_1 \xrightarrow{l_1 \to r_1}_T u_2 \xrightarrow{\mathcal{S} \cup \mathcal{W}}{}^*_T v_1 \xrightarrow{l_2 \to r_2}_T v_2 ,$$

and thus there exists substitutions σ, τ such that u_2 is labeled by $t^\sharp_i\sigma$ for some $i \in \{1, \ldots, n\}$ and v_1 by $u^\sharp\tau$. As $u_2 \xrightarrow{\mathcal{S} \cup \mathcal{W}}{}^*_T v_1$ we have $t^\sharp_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{S} \cup \mathcal{W}} u^\sharp\tau$ by definition, and thus there is an edge from $l_1 \to r_1$ to $l_2 \to r_2$ in the DG of $\mathcal{P}^\sharp$, and hence in $\mathcal{G}$. The lemma follows from this. □

14.4.7. Dependency Pairs in TCT

In TCT, the transformation `dependencyPairs` implements Theorem 14.24.

TCT-interactive 14.7 (Continued from Session 14.5)

```
TCT> apply dependencyPairs

  Problems simplified. Use 'state' to see the current proof state.

TCT> state
  Current Proof State ------------------------------------------------

  Selected Open Problems:
  -----------------------
    Strict DPs:
      { +^#(0(), x) -> c_1(x)
      , +^#(s(x), y) -> c_2(+^#(x, y))
      , *^#(s(x), y) -> c_3(+^#(y, *(x, y))) }
    Strict Trs:
      { +(0(), x) -> x
      , +(s(x), y) -> s(+(x, y))
      , *(s(x), y) -> +(y, *(x, y)) }
    Weak DPs: { *^#(0(), x) -> c_4() }
    Weak Trs: { *(0(), x) -> 0() }
    StartTerms: basic terms
    Strategy: none
  ---------------------------------------------------------------------
TCT>
```

On the canonical innermost runtime complexity problem of the TRS $\mathcal{R}_{\mathsf{K}}$ from Example 14.2, TCT produces the DP problem depicted in Example 14.31 (modulo renaming of compound symbols) as follow.

TCT-interactive 14.8

```
TCT> load "examples/kruskal.trs"
✂.....................................................................................
TCT> apply dependencyTuples

    Problems simplified. Use 'state' to see the current proof state.

TCT>
```

Note that TCT does not allow the application of dependency tuples when the given complexity problem is not an innermost runtime complexity problem. In general, TCT always takes care that preconditions of processors are taken into account.

In TCT, we currently employ the dependency graph approximation given by Thiemann [76, Section 3.1] for $\mathcal{Q}$-restricted rewriting. To inspect the approximated dependency graph, TCT provides the command `wdgs` (the acronym for *weak dependency graphs*) that both displays and returns the list of dependency graphs of all open problems. To see this information in GNU Emacs when the TCT major mode is running, issue the command `C-c s`. This opens a new frame that visualises the current state, including dependency graphs whenever a DP problem is present. Compare Figure 14.4 that shows the approximated dependency graph computed for the running example $\langle \mathcal{S}^\sharp_{\mathsf{K}}/\mathcal{R}_{\mathsf{K}}, \mathcal{R}_{\mathsf{K}}, \mathcal{T}^\sharp_{\mathsf{K}}\rangle$ loaded in Session 14.8. In the displayed dependency graph, cycles denote nodes of the dependency graph, and rectangles visualise the induced *congruence*, i.e., maximal cycles in the DG. The directed acyclic graph obtained by identifying nodes with their congruence

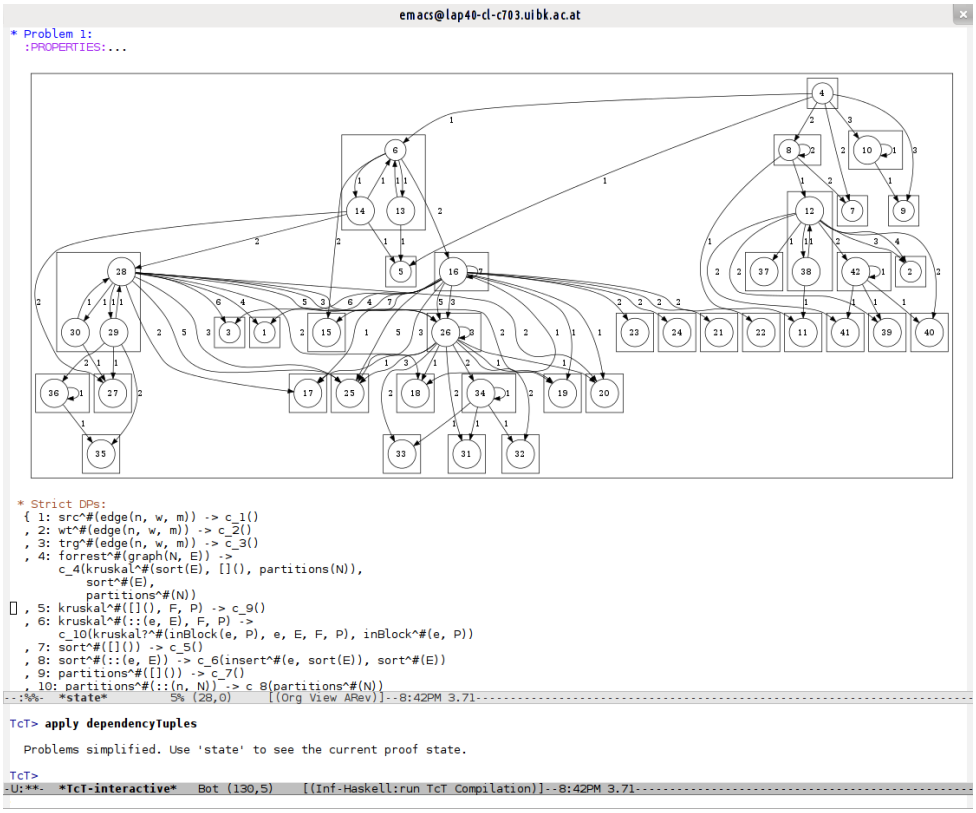


Figure 14.4.: Dependency Graphs in GNU Emacs major mode.

class is also called a *congruence dependency graph* (*CDG* for short) [39]. In correspondence to the command `wdgs`, TcT provides the command `cwdgs` which computes the congruence dependency graphs on open problems.

14.5. Syntactic Simplifications

In this section we introduce a handful of syntactic simplifications. None of the observations is very deep. Nevertheless the resulting processors are important. They allow the reduction of a complexity problem to a core set of rules, reflecting the complexity of the input problem asymptotically.

14.5.1. Usable Rules

In termination analysis it is standard to consider only those rewrite rules that can occur between applications of dependency pairs, the *usable rules*. Hirowaka and Moser [38] have shown that this technique can be safely employed for complexity analysis. Following definition captures an approximation of usable rules that looks at defined function symbols.

Definition 14.44 (Usable Rules)**.** Consider a DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$.

(1) Let $\mathcal{D}_{\mathcal{P}^\sharp}$ collect be the defined symbols in $\mathcal{P}^\sharp$, i.e.,

$$\mathcal{D}_{\mathcal{P}^\sharp} := \{f \mid f(l_1, \ldots, l_k) \to r \in \mathcal{P}^\sharp\} \,.$$

We define the binary relation $\rhd_\mathrm{d}$ on $\mathcal{D}_{\mathcal{P}^\sharp}$ such that $f \rhd_\mathrm{d} g$ holds if there exists a rule or dependency pair $f(l_1, \ldots, l_k) \to r \in \mathcal{P}^\sharp$ such that $g \in \mathcal{D}_{\mathcal{P}^\sharp}$ occurs in r. We say that f *depends on* g.

(2) The set of *usable symbols* $\mathcal{US}_{\mathcal{P}^\sharp}(t) \subseteq \mathcal{D}_{\mathcal{P}^\sharp}$ *of a term* t is defined as

$$\mathcal{US}_{\mathcal{P}^\sharp}(t) := \{g \mid f \rhd_\mathrm{d}^* g \text{ for some } f \ \in \mathcal{D}_{\mathcal{P}^\sharp} \text{ that also occurs } t.\} \,.$$

The notion of usable symbols is extended to sets of terms $\mathcal{T}$ by

$$\mathcal{US}_{\mathcal{P}^\sharp}(\mathcal{T}) := \bigcup_{t \in \mathcal{T}} \mathcal{US}_{\mathcal{P}^\sharp}(t) \,.$$

(3) The *usable rules* $\mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{R}^\sharp \cup \mathcal{R})$ in $\mathcal{P}^\sharp$ of $\mathcal{R}^\sharp \cup \mathcal{R}$ are given by

$$\mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{R}^\sharp \cup \mathcal{R}) := \{f(l_1, \ldots, l_k) \to r \in \mathcal{R}^\sharp \cup \mathcal{R} \mid f \in \mathcal{US}_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)\} \,.$$

The following auxiliary lemma shows that the usable symbols of starting terms $\mathcal{T}^\sharp$ are closed under $\mathcal{P}^\sharp$ reductions. In particular, this implies that only usable rules are ever triggered in derivations starting from $t \in \mathcal{T}^\sharp$. Observe that the lemma crucially employs that starting terms are constructor based. This drastically simplifies the proof of soundness, compared to the setting of termination analysis.

Lemma 14.45. *Let* $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ *denote a DP problem. Then*

$$\mathcal{US}_{\mathcal{P}^\sharp}(\to^*_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)) = \mathcal{US}_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp) \,.$$

Proof. The inclusion $\mathcal{US}_{\mathcal{P}^\sharp}(\to_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)) \supseteq \mathcal{US}_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)$ follows from the trivial inclusion $\to^*_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp) \supseteq \mathcal{T}^\sharp$. For the inverse inclusion, consider $s \in \to_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)$, hence there exists $t \in \mathcal{T}^\sharp$ with $s \to^\ell_{\mathcal{P}^\sharp} t$ for some $\ell \in \mathbb{N}$. We show that $\mathcal{US}_{\mathcal{P}^\sharp}(t) \subseteq \mathcal{US}_{\mathcal{P}^\sharp}(s)$ by induction on k. The base case $\ell = 0$, i.e., $t = s$ is trivial.

For the inductive step, suppose $s \to^\ell_{\mathcal{P}^\sharp} t \to_{\mathcal{P}^\sharp} u$ holds. Consider $g \in \mathcal{US}_{\mathcal{P}^\sharp}(u)$, we show $g \in \mathcal{US}_{\mathcal{P}^\sharp}(t)$. In the considered case, there exists a symbol $h \in \mathcal{D}_{\mathcal{P}^\sharp}$ in u with $h \rhd_\mathrm{d}^* g$. If h occurs also in t then by definition $g \in \mathcal{US}_{\mathcal{P}^\sharp}(t)$ holds, hence suppose h does not occur in t. As $t \to_{\mathcal{P}^\sharp} u$, there exist a rule $f(l_1, \ldots, l_k) \to r \in \mathcal{P}^\sharp$, substitution σ and context C such that $t = C[f(l_1\sigma, \ldots, l_k\sigma)]$ and $u = C[r\sigma]$. Since h does not occur in t but in u, it thus occurs in r. Hence $f \rhd_\mathrm{d} h$ and using $h \rhd_\mathrm{d}^* g$ we obtain again $g \in \mathcal{US}_{\mathcal{P}^\sharp}(t)$. It follows that $\mathcal{US}_{\mathcal{P}^\sharp}(u) \subseteq \mathcal{US}_{\mathcal{P}^\sharp}(t)$ holds, and we conclude by induction hypothesis $\mathcal{US}_{\mathcal{P}^\sharp}(t) \subseteq \mathcal{US}_{\mathcal{P}^\sharp}(s)$. □

Theorem 14.46 (Usable Rules Processor)**.** *Let* $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ *denote a DP problem. The following processor is sound and complete:*

$$\frac{\vdash \langle \mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{S}^\sharp \cup \mathcal{S}) / \mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{W}^\sharp \cup \mathcal{W}), \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon f}{\vdash \langle \mathcal{S}^\sharp \cup \mathcal{S} / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon f} \ \mathsf{Usable\ Rules} \,.$$

Proof. Consider a derivation

$$t_0 \to_{\mathcal{P}^\sharp} t_1 \to_{\mathcal{P}^\sharp} t_2 \to_{\mathcal{P}^\sharp} \cdots ,$$

for $t_0 \in \mathcal{T}^\sharp$. Then this derivation is also a derivation with respect to the generated problem $\langle \mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{S}^\sharp \cup \mathcal{S})/\mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{W}^\sharp \cup \mathcal{W}), \mathcal{Q}, \mathcal{T}^\sharp\rangle$. For this, consider a rewrite step $t_i \to_{\mathcal{P}^\sharp} t_{i+1}$ $(i \in \mathbb{N})$. Let $f(l_1, \ldots, l_k) \to r \in \mathcal{P}^\sharp$ denote the rule which triggered the rewrite step $t_i \to_{\mathcal{P}^\sharp} t_{i+1}$ $(i \in \mathbb{N})$. Then $f \in \mathcal{US}_{\mathcal{P}^\sharp}(t_i)$, by Lemma 14.45 it follows that $f \in \mathcal{US}_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)$. By definition, we have that $f(l_1, \ldots, l_k) \to r \in \mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{S}^\sharp \cup \mathcal{S})$ or $f(l_1, \ldots, l_k) \to r \in \mathcal{U}_{\mathcal{P}^\sharp}(\mathcal{W}^\sharp \cup \mathcal{W})$ respectively. This concludes soundness.

As every derivation from $t_0 \in \mathcal{T}^\sharp$ with respect to the generated problem is trivially a $\mathcal{P}^\sharp$ derivation, we see that the processor is also complete. □

Example 14.47 (Continued from Example 14.31). Consider the dependency pair problem given in Example 14.31 obtained from the dependency tuple processor. Then the defined symbols forest, kruskal and kruskal? are not usable with respect to the considered starting terms. Application of the usable rules processor gives the following inference.

$$\frac{\vdash \langle \mathcal{S}^\sharp_{\mathsf{K}}/\mathcal{U}_{\mathsf{K}}, \mathcal{R}_{\mathsf{K}}, \mathcal{T}^\sharp_{\mathsf{K}}\rangle : n^2}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{K}}/\mathcal{R}_{\mathsf{K}}, \mathcal{R}_{\mathsf{K}}, \mathcal{T}^\sharp_{\mathsf{K}}\rangle : n^2}\ \mathsf{Usable\ Rules}\ ,$$

where $\mathcal{U}_{\mathsf{K}} := \{36, 37, 42\text{–}73\}$. Hence in total, the six rules $\{35, 38\text{–}41\}$ defining forest, kruskal and kruskal? are dropped from the weak component. ◁

14.5.2. Removing of Weak Suffixes in the DG

The *leaf removal* processor introduced in [63] states that all dependency pairs that occur as *leafs* in the DG, that is, dependency pairs that constitute nodes in the DG without outgoing edges, can be dropped from the input problem. This processor is sound in the setting of [63] where all dependency pairs occur in the strict component. Without further restrictions, this processor is unsound in our setting.

Example 14.48. The following inference is not sound

$$\frac{\vdash \langle \varnothing/\{\mathsf{f}^\sharp \to \mathsf{c}_2(\mathsf{f}^\sharp, \mathsf{g}^\sharp)\}, \varnothing, \{\mathsf{f}^\sharp\}\rangle : f}{\vdash \langle \{\mathsf{g}^\sharp \to \mathsf{c}_0\}/\{\mathsf{f}^\sharp \to \mathsf{c}_2(\mathsf{f}^\sharp, \mathsf{g}^\sharp)\}, \varnothing, \{\mathsf{f}^\sharp\}\rangle : f}\ ,$$

despite the fact that $\mathsf{g}^\sharp \to \mathsf{c}_0$ is a leaf in the dependency graph of the input problem. Observe that the complexity function of the input problem is undefined for inputs greater than two, whereas the generated problem has trivially constant complexity. ◁

Another situation where removal of leafs leads to problems is when our analysis has to account for ordinary rewrite rules beside dependency pairs. Again this case is a priori excluded in [63].

Example 14.49. Consider the following inference, where $\mathsf{f}^\sharp \to \mathsf{g}^\sharp(\mathsf{h})$ denotes a leaf in the dependency graph of the input problem.

$$\frac{\vdash \langle \{\mathsf{h} \to \mathsf{h}\}/\varnothing, \varnothing, \{\mathsf{f}^\sharp\}\rangle : f}{\vdash \langle \{\mathsf{f}^\sharp \to \mathsf{g}^\sharp(\mathsf{h}),\ \mathsf{h} \to \mathsf{h}\}/\varnothing, \varnothing, \{\mathsf{f}^\sharp\}\rangle : f}\ ,$$

Then the generated problem has constant complexity, whereas the complexity function of the input problem is undefined for inputs greater than two. ◁

What we can do, is to remove dependency pairs from the weak component that do not trigger rewrite steps with respect to the strict component. Provided that the strict component of the input problem constitutes of dependency pairs only, rules amendable for removal can be determined based on the dependency graph.

Definition 14.50 (Forward Closed)**.** Let $\mathcal{G}$ be a dependency graph and let $\mathcal{R}^\sharp$ denote a set of dependency pairs. We say that $\mathcal{R}^\sharp$ is *closed under $\mathcal{G}$-successors* if for every edge from $s \to t \in \mathcal{R}^\sharp$ to $u \to v$ in $\mathcal{G}$ we have that also $u \to v \in \mathcal{R}^\sharp$.

For a complexity problem $\mathcal{P}^\sharp$, we call a set of DPs occurring in $\mathcal{P}^\sharp$ *forward closed* if this set is closed under $\mathcal{G}$-successors for the dependency graph $\mathcal{G}$ of $\mathcal{P}^\sharp$.

Theorem 14.51 (Remove Weak Suffixes Processor)**.** *Consider a DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ where $\mathcal{W}_l^\sharp$ is forward closed. The following processor is sound and complete.*

$$\frac{\vdash \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : f}{\vdash \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : f}\ \textsf{Remove Weak Suffixes}\,.$$

Proof. Let $\mathcal{P}_g^\sharp := \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$. The processor is trivially complete. To see that the processor is sound, we show that for every $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$, there exists a $\mathcal{P}_g^\sharp$ derivation tree T of $t^\sharp$ with $|T_g|_{\mathcal{S}^\sharp} = |T|_{\mathcal{S}^\sharp}$. Hence soundness follows by Lemma 14.38.

Consider a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp$. We define T_g as the derivation tree of $t^\sharp$ that is obtained by removing applications of dependency pairs from $\mathcal{W}_l^\sharp$. More precise, whenever there is an edge e labeled with a DP from $\mathcal{W}_l^\sharp$ in T, we remove e and the sub-trees rooted in the target nodes of e. Then by construction T_g is a $\mathcal{P}_g^\sharp$ derivation tree of $t^\sharp$. Suppose now $|T|_{\mathcal{S}^\sharp} \neq |T_g|_{\mathcal{S}^\sharp}$, i.e., $|T|_{\mathcal{S}^\sharp} > |T_g|_{\mathcal{S}^\sharp}$. Then T_g misses a sub-tree of T with an edge labeled by a DP $l \to r \in \mathcal{S}^\sharp$. This assumption gives rise to a DP chain where $l \to r \in \mathcal{S}^\sharp$ is preceded by a DP from $\mathcal{W}_l^\sharp$, contradicting that $\mathcal{W}_l^\sharp$ is forward closed, by Lemma 14.43. □

14.5.3. Predecessor Estimation

Noschinski et al. [63] observed that the application of a dependency pair $l \to r$ in a $\mathcal{P}^\sharp$ derivation can be estimated in terms of the application of its predecessors in the dependency graph of $\mathcal{P}^\sharp$.

Definition 14.52. Let $\mathcal{G}$ be an approximated dependency graph. We denote by $\mathsf{Pre}_{\mathcal{G}}(l \to r)$ the set of all (direct) *predecessors* of node $l \to r$ in $\mathcal{G}$. For a set of dependency pairs $\mathcal{R}^\sharp$ we set

$$\mathsf{Pre}_{\mathcal{G}}(\mathcal{R}^\sharp) := \bigcup_{l \to r \in \mathcal{R}^\sharp} \mathsf{Pre}_{\mathcal{G}}(l \to r) \ .$$

Then for an approximated dependency graph $\mathcal{G}$ of $\mathcal{P}^\sharp$, we have the following correspondence between the number of applications of dependency pairs from $\mathcal{R}^\sharp$ and $\mathsf{Pre}_{\mathcal{G}}(\mathcal{R}^\sharp)$.

Lemma 14.53. *Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ denote a DP problem, let $\mathcal{G}$ be an approximated dependency graph for $\mathcal{P}^\sharp$, and let $l \to r \in \mathcal{P}^\sharp$ denote a dependency pair. For every $\mathcal{P}^\sharp$-derivation tree T,*

$$|T|_{\mathcal{R}^\sharp \cup \mathcal{R}} \leqslant \max\{1, |T|_{(\mathcal{R}^\sharp \setminus \{l \to r\}) \cup \mathsf{Pre}_{\mathcal{G}}(l \to r) \cup \mathcal{R}} \cdot K\} \ ,$$

where K denote the maximal arity of a compound symbol in $\mathcal{P}^\sharp$.

Proof. Consider the non-trivial case $l \to r \notin \mathsf{Pre}_{\mathcal{G}}(l \to r)$ and let T denote a $\mathcal{P}^\sharp$ derivation tree with an edge labeled by $l \to r \in \mathcal{R}^\sharp$. It suffices to verify $|T|_{\{l \to r\}} \leqslant \max\{1, |T|_{\mathsf{Pre}_{\mathcal{G}}(l \to r)} \cdot K\}$. By Lemma 14.43 chains of T translate to paths in $\mathcal{G}$. Thus if $l \to r$ is applied in T, then $l \to r \notin \mathsf{Pre}_{\mathcal{G}}(l \to r)$ gives that either $l \to r$ occurs only in the beginning of chains, or is headed by a dependency pair from $\mathsf{Pre}_{\mathcal{G}}(l \to r)$. In the former case $|T|_{\{l \to r\}} = 1$. In the latter case, let $\{u_1, \ldots, u_n\}$ collect all sources of $l \to r$ edges in T. To each node $u_i \in \{u_1, \ldots, u_n\}$ we can identify a unique node $\mathsf{pre}(u_i)$ such that $\mathsf{pre}(u_i) \xrightarrow{\mathsf{Pre}_{\mathcal{G}}(l \to r)}_T \cdot \xrightarrow{\mathcal{S} \cup \mathcal{W}}_T{}^{*} u_i$. Let $\{v_1, \ldots, v_m\} = \{\mathsf{pre}(u_1), \ldots, \mathsf{pre}(u_n)\}$. Since $\xrightarrow{\mathcal{S} \cup \mathcal{W}}_T$ is non-branching, and $\mathsf{pre}(u_i)$ has at most K successors, it follows that $|T|_{\{l \to r\}} = n \leqslant K \cdot m \leqslant K \cdot |T|_{\mathsf{Pre}_{\mathcal{G}}(l \to r)}$. □

Theorem 14.54 (Predecessor Estimation Processor)**.** *Let $\mathcal{G}$ denote an approximated dependency graph of the DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}_1^\sharp \cup \mathcal{S}_2^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$. The following processor is sound:*

$$\frac{\vdash \langle \mathsf{Pre}_{\mathcal{G}}(\mathcal{S}_1^\sharp) \cup \mathcal{S}_2^\sharp \cup \mathcal{S}/\mathcal{S}_1^\sharp \cup \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon f}{\vdash \langle \mathcal{S}_1^\sharp \cup \mathcal{S}_2^\sharp \cup \mathcal{S}/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon f} \ \mathsf{Predecessor\ Estimation} \ .$$

Proof. Soundness follows trivially from Lemma 14.53, employing Lemma 14.38.□

We point out that the predecessor estimation processor is an adaption of *knowledge propagation* introduced in [63]. The notion of problem from [63] uses for this processor specifically a dedicated component $\mathcal{K}$ mentioned on page 159.

Application of the predecessor estimation processor makes only sense if $\mathsf{Pre}_{\mathcal{G}}(\mathcal{S}_1^\sharp) \neq \mathcal{S}_1^\sharp$. On the other hand, it is always safe to take for $\mathcal{S}_1^\sharp$ a maximal set of DPs such that $\mathsf{Pre}_{\mathcal{G}}(\mathcal{S}_1^\sharp) \subseteq \mathcal{S}_2$.

The combination of Theorem 14.51 and Theorem 14.54 allows us to remove DPs $\mathcal{R}^\sharp$ that occur as leafs in the DG $\mathcal{G}$ of the input problem, provided $\mathsf{Pre}_{\mathcal{G}}(\mathcal{R}^\sharp)$ constitutes of dependency pairs that occur in the strict component, as in [63]. This is clarified on our running example.

Example 14.55 (Continued from Example 14.47). Observe that the DPs

$$\mathcal{L}^\sharp := \{\mathtt{85}\text{–}\mathtt{87}, \mathtt{89}, \mathtt{91}, \mathtt{95}, \mathtt{97}, \mathtt{101}, \mathtt{103}, \mathtt{105}, \mathtt{107}, \mathtt{109}, \mathtt{111}\text{–}\mathtt{113}, \mathtt{115}\text{–}\mathtt{117}, \mathtt{119}\text{–}\mathtt{126}\}$$

depicted in Example 14.31 occur as leafs in the dependency graph $\mathcal{G}$ of the complexity problem $\langle \mathcal{S}^\sharp_\mathsf{K}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle$. Instantiating $\mathcal{S}^\sharp_1$ by $\mathcal{L}^\sharp$ and $\mathcal{S}^\sharp_2$ by

$$\mathcal{S}^\sharp_\mathsf{K} \setminus \mathcal{L}^\sharp := \{\mathtt{88}, \mathtt{90}, \mathtt{92}\text{–}\mathtt{94}, \mathtt{96}, \mathtt{98}\text{–}\mathtt{100}, \mathtt{102}, \mathtt{104}, \mathtt{106}, \mathtt{108}, \mathtt{110}, \mathtt{114}, \mathtt{118}\} \;,$$

in Theorem 14.54 we can continue the complexity proof of Example 14.47 as follows.

$$\dfrac{\dfrac{\vdash \langle (\mathcal{S}^\sharp_\mathsf{K} \setminus \mathcal{L}^\sharp)/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}{\vdash \langle (\mathcal{S}^\sharp_\mathsf{K} \setminus \mathcal{L}^\sharp)/\mathcal{L}^\sharp \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \textsf{Remove Weak Suffixes}}{\vdash \langle \mathcal{S}^\sharp_\mathsf{K}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \textsf{Predecessor Estimation} \;.$$

Observe that we employ $\mathsf{Pre}_\mathcal{G}(\mathcal{L}^\sharp) \subseteq \mathcal{S}^\sharp_\mathsf{K} \setminus \mathcal{L}^\sharp$, and that $\mathcal{L}^\sharp$ is trivially forward closed in the intermediate problem. ◁

14.5.4. Simplifying Right-hand Sides

In the proof step given in Example 14.55 we have removed all dependency pairs defining the dependency pair symbols $\mathsf{src}^\sharp$, $\mathsf{wt}^\sharp$ and $\mathsf{trg}^\sharp$ as well as $\vee^\sharp$ and $\wedge^\sharp$. Since the strict component of the considered DP problem contains only dependency pairs, it is safe to remove any calls to these symbols in right-hand sides. For instance, the dependency pair

$$\begin{aligned}\mathtt{96}\colon\quad \mathsf{inBlock}^\sharp(e, p :: P) \to \mathsf{c}_7((&\mathsf{src}(e) \in p \wedge \mathsf{trg}(e) \in p) \vee^\sharp \mathsf{inBlock}(e, P),\\ &\mathsf{src}(e) \in p \wedge^\sharp \mathsf{trg}(e) \in p, \mathsf{src}(e) \in^\sharp p,\\ &\mathsf{trg}(e) \in^\sharp p, \mathsf{src}^\sharp(e), \mathsf{trg}(e)^\sharp, \mathsf{inBlock}^\sharp(e, P)) \;,\end{aligned}$$

can be simplified to

$$\mathtt{96s}\colon\quad \mathsf{inBlock}^\sharp(e, p :: P) \to \mathsf{c}_3(\mathsf{src}(e) \in^\sharp p, \mathsf{trg}(e) \in^\sharp p, \mathsf{inBlock}^\sharp(e, P)) \;.$$

Although this simplification has no effect on the complexity of the considered problem, it still makes an automated analysis often easier. For instance, as a result of the transformation the usable rules, and also constraints for complexity pairs, can get simpler. The next processor provides a formalisation of this idea, which was first implemented in AProVE [63].

Theorem 14.56 (Simplify Right-hand Sides Processor)**.** *Let $\mathcal{G}$ denote an approximated dependency graph of the DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$. For dependency pairs $l^\sharp \to \mathrm{COM}(r^\sharp_1, \ldots, r^\sharp_k) \in \mathcal{S}^\sharp \cup \mathcal{W}^\sharp$, call an argument $r^\sharp_i$ removable if there is no outgoing edge from $l^\sharp \to \mathrm{COM}(r^\sharp_1, \ldots, r^\sharp_k)$ labeled by i. Define*

$$\mathsf{simp}_\mathcal{G}(l^\sharp \to \mathrm{COM}(r^\sharp_1, \ldots, r^\sharp_k)) := l^\sharp \to \mathrm{COM}(r^\sharp_{i_1}, \ldots, r^\sharp_{i_l}) \;,$$

where $\{r^\sharp_{i_1}, \dots, r^\sharp_{i_l}\} \subseteq \{r^\sharp_1, \dots, r^\sharp_k\}$ *collects all arguments of the right-hand side which are not removable. We denote by* $\mathsf{simp}_\mathcal{G}$ *also its homomorphic extensions to sets.*

The following processor is sound and complete.

$$\frac{\vdash \langle \mathsf{simp}_\mathcal{G}(\mathcal{S}^\sharp)/\mathsf{simp}_\mathcal{G}(\mathcal{W}^\sharp) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : f}{\vdash \langle \mathcal{S}^\sharp/\mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : f}\ \mathsf{Simplify\ RHS}\ .$$

Proof. Let $\mathcal{P}^\sharp_g$ denote the generated problem $\langle \mathsf{simp}_\mathcal{G}(\mathcal{S}^\sharp)/\mathsf{simp}_\mathcal{G}(\mathcal{W}^\sharp) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$.

Consider a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$. Call a sub-tree T_i resulting from the application of a DP $l^\sharp \to \textsc{com}(r^\sharp_1, \dots, r^\sharp_k) \in \mathcal{S}^\sharp \cup \mathcal{W}^\sharp$ in T *removable*, if its root is labeled by a term $r^\sharp_i\sigma$ for some removable argument $r^\sharp_i$ and substitution σ. By Lemma 14.43 and the assumption that $r^\sharp_i$ is removable, T_i contains no applications of a DP, i.e., $|T_i|_{\mathcal{S}^\sharp} = 0$. By deleting all removable sub-trees of T, and replacing the application of $l^\sharp \to \textsc{com}(r^\sharp_1, \dots, r^\sharp_k) \in \mathcal{S}^\sharp \cup \mathcal{W}^\sharp$ by $\mathsf{simp}_\mathcal{G}(l^\sharp \to \textsc{com}(r^\sharp_1, \dots, r^\sharp_k)) \in \mathsf{simp}_\mathcal{G}(\mathcal{S}^\sharp) \cup \mathsf{simp}_\mathcal{G}(\mathcal{W}^\sharp)$ we can thus construct a $\mathcal{P}^\sharp_g$ derivation tree T_g with $|T_g|_{\mathsf{simp}_\mathcal{G}(\mathcal{S}^\sharp)} = |T|_{\mathcal{S}^\sharp}$. This concludes soundness of the processor, by Lemma 14.38.

Inversely, completeness is shown by constructing from every $\mathcal{P}^\sharp_g$ derivation tree a $\mathcal{P}^\sharp$ derivation tree, obtained by replacing applications of $\mathsf{simp}_\mathcal{G}(l^\sharp \to \textsc{com}(r^\sharp_1, \dots, r^\sharp_k))$ by applications of $l^\sharp \to \textsc{com}(r^\sharp_1, \dots, r^\sharp_k)$. □

Example 14.57 (Continued from Example Example 14.55). By Theorem 14.56 the following inference is sound.

$$\frac{\vdash \langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}{\vdash \langle (\mathcal{S}^\sharp_\mathsf{K} \setminus \mathcal{L}^\sharp)/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \mathsf{Simplify\ RHS}\ .$$

Here $\mathcal{S}^\sharp_{\mathsf{Ks}} := \mathsf{simp}_\mathcal{G}((\mathcal{S}^\sharp_\mathsf{K} \setminus \mathcal{L}^\sharp))$ consists of the following dependency pairs.

$$\begin{aligned}
88:&\quad \mathsf{forest}^\sharp(\mathsf{graph}(N,E)) &&\to \mathsf{c}_3(\mathsf{kruskal}^\sharp(\mathsf{sort}(E),[\,],\mathsf{partitions}(N)),\\
&&&\qquad \mathsf{sort}^\sharp(E),\mathsf{partitions}^\sharp(N))\\
90:&\quad \mathsf{partitions}^\sharp(n :: N) &&\to \mathsf{partitions}^\sharp(N)\\
92:&\quad \mathsf{kruskal}^\sharp(e :: E, W, P) &&\to \mathsf{c}_2(\mathsf{kruskal?}^\sharp(\mathsf{inBlock}(e,P),e,E,W,P),\\
&&&\qquad \mathsf{inBlock}^\sharp(e,P))\\
93:&\quad \mathsf{kruskal?}^\sharp(\mathsf{tt},e,E,W,P) &&\to \mathsf{kruskal}^\sharp(E,W,P)\\
94:&\quad \mathsf{kruskal?}^\sharp(\mathsf{ff},e,E,W,P) &&\to \mathsf{c}_2(\mathsf{kruskal}^\sharp(E,e :: W,\mathsf{join}(e,P,[\,])),\\
&&&\qquad \mathsf{join}^\sharp(e,P,[\,]))\\
96s:&\quad \mathsf{inBlock}^\sharp(e,p :: P) &&\to \mathsf{c}_3(\mathsf{src}(e) \in^\sharp p, \mathsf{trg}(e) \in^\sharp p, \mathsf{inBlock}^\sharp(e,P))\\
98s:&\quad \mathsf{join}^\sharp(e,p :: P,q) &&\to \mathsf{c}_2(\mathsf{join?}^\sharp(\mathsf{src}(e) \in p \vee \mathsf{trg}(e) \in p,e,p,P,q),\\
&&&\qquad \mathsf{src}(e) \in^\sharp p)\\
99:&\quad \mathsf{join?}^\sharp(\mathsf{tt},e,p,P,q) &&\to \mathsf{c}_2(\mathsf{join}^\sharp(e,P,p +\!\!+ q), p +\!\!+^\sharp q)
\end{aligned}$$

$$
\begin{aligned}
100: \quad & \mathsf{join}?^\sharp(\mathsf{ff}, e, p, P, q) \to \mathsf{join}^\sharp(e, P, q) \\
102: \quad & \mathsf{sort}^\sharp(e :: E) \to \mathsf{c}_2(\mathsf{insert}^\sharp(e, \mathsf{sort}(E)), \mathsf{sort}^\sharp(E)) \\
104s: \quad & \mathsf{insert}^\sharp(e, f :: E) \to \mathsf{c}_2(\mathsf{insert}?^\sharp(\mathsf{wt}(e) \leqslant \mathsf{wt}(f), e, f, E), \\
& \qquad\qquad\qquad\qquad \mathsf{wt}(e) \leqslant^\sharp \mathsf{wt}(f)) \\
106: \quad & \mathsf{insert}?^\sharp(\mathsf{ff}, e, f, E) \to \mathsf{insert}^\sharp(e, E) \\
108s: \quad & n \in^\sharp (m :: p) \to \mathsf{c}_2(n =^\sharp m, n \in^\sharp p) \\
110: \quad & (n :: p) +\!\!+^\sharp q \to p +\!\!+^\sharp q \\
114: \quad & \mathsf{s}(x) =^\sharp \mathsf{s}(y) \to x =^\sharp y \\
118: \quad & \mathsf{s}(x) \leqslant^\sharp \mathsf{s}(y) \to x \leqslant^\sharp y \ .
\end{aligned}
$$

◁

14.5.5. Simplifications In TCT

The processors given in this section are implemented in TCT by the transformations `usableRules` (Theorem 14.46), `removeWeakSuffix` (Theorem 14.51), `simpPEOn` (Theorem 14.54) and `simpDPRHS` (Theorem 14.56).

In order to specify the DPs $\mathcal{S}_1^\sharp$ in Theorem 14.54, the transformation `simpPEOn` takes a selector expression as argument. To avoid that DPs from the weak component of the input problem reoccur in the strict component of the generated sub-problem, `simpPEOn` restrict the selected rules to a set so that predecessors $\mathsf{Pre}(\mathcal{S}_1^\sharp)$ occur in the strict component. The transformation `simpPE` provides a short-hand for `simpPEOn`, and selects per default a maximal subset $\mathcal{S}_1^\sharp$.

The transformation `cleanSuffix`, defined as follows, provides a short-hand for a combination of the mentioned simplifications.

```
cleanSuffix = force $
  try (exhaustively (simpPEOn allSuccsWeak))
  >>> try (removeWeakSuffix >>> try (simpDPRHS >>> try usableRules))
```

Here `allSuccsWeak` is used to select DPs from the input problem $\mathcal{P}^\sharp$ whose successors in the DG occur all in the weak component of $\mathcal{P}^\sharp$. As a consequence, `exhaustively (simpPEOn allSuccsWeak)` moves DPs that constitute maximal sub-trees in the DG to the weak component of $\mathcal{P}^\sharp$. If such DPs exist, they are removed from $\mathcal{P}^\sharp$ by the transformation `removeWeakSuffix`. If DPs have been dropped from $\mathcal{P}^\sharp$, right-hand sides are simplified and usable rules possibly recomputed. Using the transformation `cleanSuffix`, TCT is able to reproduce the proof step depicted in Example 14.57.

TCT-interactive 14.9 (Continued from Session 14.8)

```
TCT> apply cleanSuffix

    Problems simplified. Use 'state' to see the current proof state.

TCT>
```

Finally, the transformation p `` `withPEOn` `` se provides a combination of the relative decomposition processor from Theorem 14.13 and predecessor estimation.

This transformation attempts to shift rewrite rules, as specified by the selector expression *se*, to the weak component. The processor p is used to estimate the complexity of predecessors of selected rules. This is demonstrated in the following session. The loaded example motivates knowledge propagation in [63].

TCT-interactive 14.10

```
TCT> load "examples/ex25.trs"
✂ ..............................................................................
TCT> apply $ dependencyPairs >>> usableRules

    Problems simplified. Use 'state' to see the current proof state.

TCT> apply $ matrix `withDimension` 1 `withPEOn` selAllOf selStricts

    Problems simplified. Use 'state' to see the current proof state.

TCT> proof

    1) Weak Dependency Pairs [OPEN]:
    --------------------------------
✂ ..............................................................................
      1.1) simpPE [OPEN]:
      -------------------
      We consider the following problem:
          Strict DPs:
            { q^#(x, 0(), s(z)) -> c_1(q^#(x, s(z), s(z)))
            , q^#(0(), s(y), s(z)) -> c_2()
            , q^#(s(x), s(y), z) -> c_3(q^#(x, y, z)) }
          StartTerms: basic terms
          Strategy: innermost

      We use the processor 'matrix interpretation of dimension 1' to
      orient the following rules strictly.

      DPs:
        { 2: q^#(0(), s(y), s(z)) -> c_2()
        , 3: q^#(s(x), s(y), z) -> c_3(q^#(x, y, z)) }
      Sub-proof:
      ----------
✂ ..............................................................................
      We return to the main proof. Consider the set of dependency pairs

        { 1: q^#(x, 0(), s(z)) -> c_1(q^#(x, s(z), s(z)))
        , 2: q^#(0(), s(y), s(z)) -> c_2()
        , 3: q^#(s(x), s(y), z) -> c_3(q^#(x, y, z)) }

      Processor 'matrix interpretation of dimension 1' induces the
      complexity certificate YES(?,O(n^1)) on application of dependency
      pairs {2,3}. These cover all (indirect) predecessors of dependency
      pairs {1,2,3}. These dependency pairs are shifted into the weak component.

      1.1.1) Open Problem [OPEN]:
      ---------------------------

        We consider the following problem:
          Weak DPs:
            { q^#(x, 0(), s(z)) -> c_1(q^#(x, s(z), s(z)))
            , q^#(0(), s(y), s(z)) -> c_2()
            , q^#(s(x), s(y), z) -> c_3(q^#(x, y, z)) }
          StartTerms: basic terms
          Strategy: innermost
✂ ..............................................................................
```

TCT-interactive 14.11 (Continued from Session 14.10)

```
TCT> apply empty

    Hurray, the problem was solved with certificate YES(O(1),O(n^1)).
    Use 'proof' to show the complete proof.

TCT>
```

Here the selector expression `selAllOf selStricts` advices TCT to find a proof that determines the complexity of all rules appearing in the strict component of the considered problem. We emphasise that the processor `matrix` alone fails to orient the depicted DP problem.

14.6. Dependency Graph Decomposition

The simplified DP problem $\langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle$ contains roughly half of the rules from the DP problem $\langle \mathcal{S}^\sharp_\mathsf{K}/\mathcal{R}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle$. Still, TCT is not able to synthesise orders that orient all of the remaining dependency pairs, even in an iterated fashion using the relative decomposition processor. At the time of writing, also AProVE and TTT2 fail to handle this system. Motivated by the inability to synthesise suitable orders for larger examples, we introduce a novel technique, called *dependency graph decomposition* (*DG decomposition* for short). The aim of this transformation technique is to decompose the input problem into several pieces that are manageable by complexity pairs. Our work on this processor is also motivated by the fact that we are not aware of a single method that translates a complexity problem into computationally simpler sub-problems. Any proof is of the form

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

with $f \in \mathcal{O}(f_i)$ for some $i \in \{1, \ldots, n\}$. This implies that the maximal bound one can prove is essentially determined by the strength of the employed base techniques, viz, complexity pairs. In our experience however, a complexity prover is seldom able to synthesise a suitable and precise complexity pair that induces a complexity bound beyond a cubic polynomial.

Before we continue, we want to remark first that relative decomposition, as given in Theorem 14.11, can be used to decompose the input problem guided by the dependency graph.

Example 14.58 (Continued from Example 14.57). Consider the DG $\mathcal{G}_\mathsf{Ks}$ of the DP problem $\langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle$, depicted in Figure 14.5. As demarcated by the rectangles in the figure, this graph has three independent sub-graphs. Let $\mathcal{S}^\sharp_\mathsf{K} \subseteq \mathcal{S}^\sharp_\mathsf{Ks}$ and $\mathcal{S}^\sharp_\mathsf{srt} \subseteq \mathcal{S}^\sharp_\mathsf{Ks}$ denote the set of DPs as indicated in the dependency graph, and let $\mathcal{S}^\sharp_\mathsf{part} := \{\mathsf{90}\}$. Essentially as an application of the relative decomposition processor, the DPs $\mathcal{S}^\sharp_\mathsf{K}$, $\mathcal{S}^\sharp_\mathsf{srt}$, and $\mathcal{S}^\sharp_\mathsf{part}$ can be treated completely

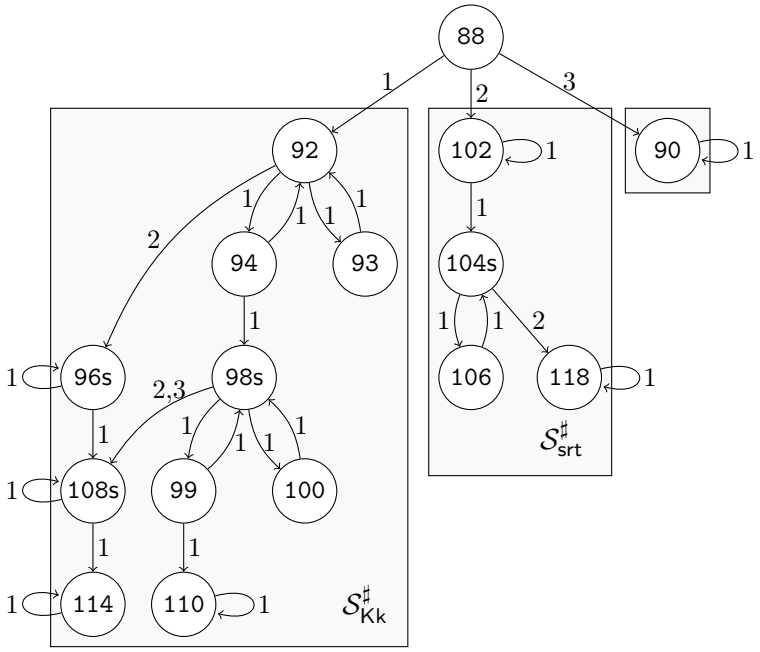


Figure 14.5.: Dependency Graph $\mathcal{G}_{\mathsf{Ks}}$ of DP problem $\langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle$.

independently. To this extend, consider the dependency pairs

$$\begin{aligned}
&\texttt{88s}\colon \mathsf{forest}^\sharp(\mathsf{graph}(N,E)) \to \mathsf{sort}^\sharp(E)\\
&\texttt{88k}\colon \mathsf{forest}^\sharp(\mathsf{graph}(N,E)) \to \mathsf{kruskal}^\sharp(\mathsf{sort}(E),[\,],\mathsf{partitions}(N))\\
&\texttt{88p}\colon \mathsf{forest}^\sharp(\mathsf{graph}(N,E)) \to \mathsf{partitions}^\sharp(N)\ ,
\end{aligned}$$

which reflect the calls from $\mathsf{forest}^\sharp$ to $\mathsf{sort}^\sharp$, $\mathsf{kruskal}^\sharp$ and $\mathsf{partitions}^\sharp$ individually. We reason below that

$$\frac{\vdash \mathcal{P}^\sharp_{\mathsf{Kk}}\colon n^2 \quad \vdash \mathcal{P}^\sharp_{\mathsf{srt}}\colon n^2 \quad \vdash \mathcal{P}^\sharp_{\mathsf{part}}\colon n^2}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle\colon n^2}$$

is a sound inference. The three sub-problems are obtained by considering only nodes from the three indicated sub-graphs, together with the calls from $\mathsf{forest}^\sharp$.

$$\begin{aligned}
\mathcal{P}^\sharp_{\mathsf{Kk}} &:= \langle \mathcal{S}^\sharp_{\mathsf{Kk}}/\{\texttt{88k}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle \qquad \mathcal{P}^\sharp_{\mathsf{srt}} := \langle \mathcal{S}^\sharp_{\mathsf{srt}}/\{\texttt{88s}\} \cup \mathcal{U}_{\mathsf{srt}}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle\\
\mathcal{P}^\sharp_{\mathsf{part}} &:= \langle \mathcal{S}^\sharp_{\mathsf{part}}/\{\texttt{88p}\}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle\ .
\end{aligned}$$

Here $\mathcal{U}_{\mathsf{srt}} := \{\texttt{34}, \texttt{48–53}, \texttt{62–65}\} \subseteq \mathcal{U}_\mathsf{K}$. To see this, consider first the following

intermediate DP problems

$$\begin{aligned}
\mathcal{P}^\sharp_{\mathsf{frst\text{-}rel}} &:= \langle \{\mathsf{88}\}/\mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{S}^\sharp_{\mathsf{srt}} \cup \mathcal{S}^\sharp_{\mathsf{part}} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle \\
\mathcal{P}^\sharp_{\mathsf{srt\text{-}rel}} &:= \langle \mathcal{S}^\sharp_{\mathsf{srt}}/\mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{S}^\sharp_{\mathsf{part}} \cup \{\mathsf{88}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle \\
\mathcal{P}^\sharp_{\mathsf{KK\text{-}rel}} &:= \langle \mathcal{S}^\sharp_{\mathsf{Kk}}/\mathcal{S}^\sharp_{\mathsf{srt}} \cup \mathcal{S}^\sharp_{\mathsf{part}} \cup \{\mathsf{88}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle \\
\mathcal{P}^\sharp_{\mathsf{part\text{-}rel}} &:= \langle \mathcal{S}^\sharp_{\mathsf{part}}/\mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{S}^\sharp_{\mathsf{srt}} \cup \{\mathsf{88}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle\ .
\end{aligned}$$

These problems are obtained from the input problem $\langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle$ by moving dependency pairs from the strict to the weak component, in accordance to the partitioning of the DG $\mathcal{G}_{\mathsf{Ks}}$. Note that dependency graphs of these problems coincide with the DG from $\mathcal{G}_{\mathsf{Ks}}$. Three applications of relative decomposition allow us to deduce

$$\dfrac{\vdash \mathcal{P}^\sharp_{\mathsf{KK\text{-}rel}} : n^2 \qquad \dfrac{\vdash \mathcal{P}^\sharp_{\mathsf{srt\text{-}rel}} : n^2 \qquad \dfrac{\vdash \mathcal{P}^\sharp_{\mathsf{part\text{-}rel}} : n^2 \qquad \vdash \mathcal{P}^\sharp_{\mathsf{frst\text{-}rel}} : n^2}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{part}} \cup \{\mathsf{88}\}/\mathcal{S}^\sharp_{\mathsf{srt}} \cup \mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{srt}} \cup \mathcal{S}^\sharp_{\mathsf{part}} \cup \{\mathsf{88}\}/\mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{Ks}}/\mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}$$

Consider the intermediate problem $\mathcal{P}^\sharp_{\mathsf{frst\text{-}rel}}$. Note that since the DP $\mathsf{88}$ has no incoming edges in the DG, i.e., $\mathsf{Pre}_{\mathcal{G}_{\mathsf{Ks}}}(\{\mathsf{88}\}) = \varnothing$, the complexity function of $\mathcal{P}^\sharp_{\mathsf{frst\text{-}rel}}$ is even constant. We can show this by one application of the predecessor estimation processor and the axiom empty.

$$\dfrac{\dfrac{}{\vdash \langle \varnothing/\mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{S}^\sharp_{\mathsf{srt}} \cup \mathcal{S}^\sharp_{\mathsf{part}} \cup \{\mathsf{88}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \mathsf{empty}}{\vdash \mathcal{P}^\sharp_{\mathsf{frst\text{-}rel}} : n^2}\ \mathsf{Predecessor\ Estimation}$$

Consider now the intermediate DP problem $\mathcal{P}^\sharp_{\mathsf{srt\text{-}rel}}$. The DPs $\mathcal{S}^\sharp_{\mathsf{Kk}}$ and $\mathcal{S}^\sharp_{\mathsf{part}}$ constitute a forward closed set of weak DPs in $\mathcal{P}^\sharp_{\mathsf{srt\text{-}rel}}$. So they can be simply dropped from $\mathcal{P}^\sharp_{\mathsf{srt\text{-}rel}}$, by Theorem 14.51. As a consequence, the right-hand side of the DP $\mathsf{88}$ can be simplified to

$$\mathtt{88s} \colon \mathsf{forest}^\sharp(\mathsf{graph}(N, E)) \to \mathsf{sort}^\sharp(E)\ ,$$

by Theorem 14.56. Finally, the set of rewrite rules can be narrowed to the usable rules $\mathcal{U}_{\mathsf{srt}}$, by Theorem 14.46. In total, this proves $\mathcal{P}^\sharp_{\mathsf{srt}} : n^2 \vdash \mathcal{P}^\sharp_{\mathsf{srt\text{-}rel}} : n^2$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \langle \mathcal{S}^\sharp_{\mathsf{srt}}/\{\mathtt{88s}\} \cup \mathcal{U}_{\mathsf{srt}}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{srt}}/\{\mathtt{88s}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \mathsf{Usable\ Rules}}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{srt}}/\{\mathsf{88}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \mathsf{Simpl.\ DP\text{-}RHS}}{\vdash \langle \mathcal{S}^\sharp_{\mathsf{srt}}/\mathcal{S}^\sharp_{\mathsf{Kk}} \cup \mathcal{S}^\sharp_{\mathsf{part}} \cup \{\mathsf{88}\} \cup \mathcal{U}_\mathsf{K}, \mathcal{R}_\mathsf{K}, \mathcal{T}^\sharp_\mathsf{K}\rangle : n^2}\ \mathsf{Remove\ Weak\ Suffixes}$$

By identical reasoning, we can prove $\mathcal{P}^\sharp_{\mathsf{Kk}} : n^2 \vdash \mathcal{P}^\sharp_{\mathsf{KK\text{-}rel}} : n^2$, and likewise $\mathcal{P}^\sharp_{\mathsf{part}} : n^2 \vdash \mathcal{P}^\sharp_{\mathsf{part\text{-}rel}} : n^2$. Putting these proofs together yields the inference outlined in the beginning of the example. ◁

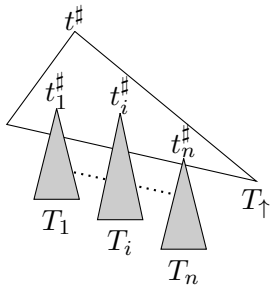


Figure 14.6.: Separation of derivation tree T in upper and lower layer.

This form of decomposition however fails to achieve our second motivation. The complexity of the input problem is reflected in the complexity of at least one of the sub-problems. In the example above, the complexity of $\mathcal{P}^\sharp_{\mathsf{Kk}}$ and $\mathcal{P}^\sharp_{\mathsf{srt}}$ is bounded by a quadratic polynomial from below.

In contrast, dependency graph decomposition seeks to analyse *recursive definitions*, as reflected by *cycles* in the DG, separately. We explain the intuition on the DP problem $\mathcal{P}^\sharp_{\mathsf{srt}}$ generated in Example 14.58, whose dependency pairs are given as follows.

$$
\begin{aligned}
&\texttt{88s}: \mathsf{forest}^\sharp(\mathsf{graph}(N,E)) \to \mathsf{sort}^\sharp(E)\\
&\texttt{102}: \quad \mathsf{sort}^\sharp(e :: E) \to \mathsf{c}_2(\mathsf{insert}^\sharp(e, \mathsf{sort}(E)), \mathsf{sort}^\sharp(E))\\
&\texttt{104s}: \quad \mathsf{insert}^\sharp(e, f :: E) \to \mathsf{c}_2(\mathsf{insert?}^\sharp(\mathsf{wt}(e) \leqslant \mathsf{wt}(f), e, f, E), \mathsf{wt}(e) \leqslant^\sharp \mathsf{wt}(f))\\
&\texttt{106}: \quad \mathsf{insert?}^\sharp(\mathsf{ff}, e, f, E) \to \mathsf{insert}^\sharp(e, E)\\
&\texttt{118}: \quad \mathsf{s}(x) \leqslant^\sharp \mathsf{s}(y) \to x \leqslant^\sharp y\ .
\end{aligned}
$$

The complexity of $\mathcal{P}^\sharp_{\mathsf{srt}}$ amounts essentially to the number of applications of a DP in a derivation from $\mathsf{sort}^\sharp(e_1 :: \cdots :: e_n :: [\,])$. Note that in each recursion step performed by $\mathsf{sort}^\sharp$ the length of the given list decreases. The number of applications of the DP `102` is thus linear in the size of the starting term. It remains to analyse the complexity of the calls to $\mathsf{insert}^\sharp$. Observe that in the i^{th} recursion step, $\mathsf{sort}^\sharp$ calls $\mathsf{insert}^\sharp$ with edge e_i and the sorted list $\mathsf{sort}(e_{i+1} :: \cdots :: e_n :: [\,])$. Using that sorting does not increase the size of its argument, it is not difficult to see that the number of applications of a dependency pair in such a call is bounded linearly in the size of (the weight of) e_i and the length of the list $e_{i+1} :: \cdots :: e_n :: [\,]$. Summing up, the complexity of $\mathcal{P}^\sharp_{\mathsf{srt}}$ is thus quadratic. Note that this bound is asymptotically tight.

Dependency graph decomposition formalises this kind of reasoning. Consider a DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp_u \cup \mathcal{S}^\sharp_l \cup \mathcal{S} / \mathcal{W}^\sharp_u \cup \mathcal{W}^\sharp_l \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$, whose strict and weak dependency pairs are partitioned such that $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ is forward closed in $\mathcal{P}^\sharp$. Forward closure of $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ in $\mathcal{P}^\sharp$ formalises that $\mathcal{S}^\sharp_u \cup \mathcal{W}^\sharp_u$ is allowed to trigger applications of DPs from $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$, but not vice versa. For instance, the set $\{\texttt{104s}, \texttt{106}, \texttt{118}\}$ forms a forward closed set of DPs in $\mathcal{P}^\sharp_{\mathsf{srt}}$.

Consider a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$. Then the forward closed set $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$ induces a separation of T into two (possibly empty) layers, demarcated by topmost applications of DPs from $\mathcal{S}^\sharp_l \cup \mathcal{W}^\sharp_l$: the *lower layer* consists of the

(maximal) subtrees $T_1, \ldots, T_n$ of T with a dependency pair $l \to r \in \mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ applied at the root. The *upper layer* consists of the derivation tree $T_\uparrow$ obtained from T by removing the sub-trees $T_1, \ldots, T_n$. Compare Figure 14.6 that illustrates this separation.

Note that by construction, $T_\uparrow$ is a $\langle \mathcal{S}_u^\sharp \cup \mathcal{S} / \mathcal{W}_u^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ derivation tree of $t^\sharp$. By forward closure of $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$, the $\mathcal{P}^\sharp$ derivations trees T_i from the lower layer are $\langle \mathcal{S}_l^\sharp \cup \mathcal{S} / \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ derivation trees of terms $t_i^\sharp$. Note that in general, $t_i^\sharp \in \mathcal{T}^\sharp$ does not hold. To extend T_i to a derivation tree of $t^\sharp \in \mathcal{T}^\sharp$, the DG decomposition processor uses the dependency pairs $\mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp)$.

Definition 14.59. For a set of DPs $\mathcal{R}^\sharp$ we define

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

Example 14.60 (Continued from Example 14.58). Consider the DP problem

$$\mathcal{P}_{\mathsf{srt}}^\sharp = \langle \mathcal{S}_{\mathsf{srt}}^\sharp / \{\mathtt{88s}\} \cup \mathcal{U}_{\mathsf{srt}}, \mathcal{R}_{\mathsf{K}}, \mathcal{T}_{\mathsf{K}}^\sharp \rangle \,,$$

together with the forward closed set of DPs $\mathcal{S}_{\mathsf{insert}}^\sharp := \{\mathtt{104s}, \mathtt{106}, \mathtt{118}\}$. Let T denote a $\mathcal{P}_{\mathsf{srt}}^\sharp$ derivation tree of $\mathsf{sort}^\sharp(e_1 :: \cdots :: e_n :: [\,])$ for edges $e_1, \ldots, e_n$. Then the nodes labeled by $\mathsf{insert}^\sharp(e_i, e'_{i+1} :: \cdots :: e'_n :: [\,])$ (for $i = 1, \ldots, n$) demarcate the upper and lower layer in T. Here $e'_{i+1} :: \cdots :: e'_n :: [\,]$ denotes the (unique) normal form of $\mathsf{sort}(e_{i+1} :: \cdots :: e_n :: [\,])$.

The set of DPs $\mathsf{sep}(\{\mathtt{88s}, \mathtt{102}\})$ consists of the following three rules.

$$\begin{aligned}
\mathtt{88s}&: \mathsf{forest}^\sharp(\mathsf{graph}(N, E)) \to \mathsf{sort}^\sharp(E) \\
\mathtt{102a}&: \mathsf{sort}^\sharp(e :: E) \to \mathsf{insert}^\sharp(e, \mathsf{sort}(E)) \\
\mathtt{102b}&: \mathsf{sort}^\sharp(e :: E) \to \mathsf{sort}^\sharp(E) \,.
\end{aligned}$$

Observe that in combination with $\mathcal{U}_{\mathsf{srt}}$, these can be used to generate the terms $\mathsf{insert}^\sharp(e_i, e'_{i+1} :: \cdots :: e'_n :: [\,])$ $(i = 1, \ldots, n)$ from the initial term $\mathsf{sort}^\sharp(e_1 :: \cdots :: e_n :: [\,])$. As a consequence, the complexity problem

$$\mathcal{P}_{\mathsf{insert}}^\sharp := \langle \mathcal{S}_{\mathsf{insert}}^\sharp / \{\mathtt{88s}, \mathtt{102a}, \mathtt{102b}\} \cup \mathcal{U}_{\mathsf{srt}}, \mathcal{R}_{\mathsf{K}}, \mathcal{T}_{\mathsf{K}}^\sharp \rangle \,,$$

accounts for applications of DPs from $\mathcal{S}_{\mathsf{srt}}^\sharp$ in a sub-tree T_i $(i \in \{1, \ldots, n\})$. Conversely, the DP problem $\mathcal{P}_{\mathsf{srt}}^\sharp = \langle \{\mathtt{102}\} / \{\mathtt{88s}\} \cup \mathcal{U}_{\mathsf{srt}}, \mathcal{R}_{\mathsf{K}}, \mathcal{T}_{\mathsf{K}}^\sharp \rangle$ accounts for the application of DPs from $\mathcal{S}_{\mathsf{srt}}^\sharp$ in the upper component $T_\uparrow$. ◁

The next two technical lemmas formalises the central proof steps carried out in the soundness proof of the dependency graph processor.

Lemma 14.61. *Consider a DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S} / \mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$. Let $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ be a forward closed set of DPs in $\mathcal{P}^\sharp$, and let T be a $\mathcal{P}^\sharp$ derivation tree T of $t^\sharp \in \mathcal{T}^\sharp$. Consider the maximal sub-trees $T_1, \ldots, T_m$ of T such that $l \to r \in \mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ is applied at the root, and let $T_\uparrow$ be obtained from T by removing $T_1, \ldots, T_m$. Then*

(1) $T_\uparrow$ *is a* $\langle \mathcal{S}_u^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ *derivation tree of* $t^\sharp$*;*

(2) *for all* $i = 1, \dots, m$*, there exists a* $\langle \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp\rangle$ *derivation trees of* $t^\sharp$*, that contains* T_i *as sub-tree.*

Proof. The first assertion follows by definition. For the second, observe that on the path from the root of T to T_i, by construction only dependency pairs $l \to \mathrm{COM}(r_1, \dots, r_k) \in \mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp$ are applied. Replacing these applications by an application of $l \to r_i \in \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp)$ yields a $\langle \mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp\rangle$ derivation tree of $t^\sharp$. This tree contains T_i as sub-tree. Since $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ is forward closed, Lemma 14.43 yields that T_i contains only applications of DPs from $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ besides application of rules $l \to r \in \mathcal{S} \cup \mathcal{W}$. Hence the constructed tree is even a $\langle \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp\rangle$ derivation tree. □

Lemma 14.62. *Consider a DP problem* $\mathcal{P}^\sharp = \langle \mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$*. Let* $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ *be a forward closed set of DPs in* $\mathcal{P}^\sharp$*, and let* T *be a* $\mathcal{P}^\sharp$ *derivation tree* T *of* $t^\sharp \in \mathcal{T}^\sharp$*. Let* $T_1, \dots, T_m$ *denote the maximal sub-trees of* T *with* $l \to r \in \mathcal{R}^\sharp$ *applied at the root. There exists a constant* $\Delta \in \mathbb{N}$ *depending only on* $\mathcal{P}^\sharp$ *such that* $m \leqslant \max\{1, |T|_{\mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp)\setminus\mathcal{R}^\sharp} \cdot \Delta\}$.

Proof. The proof is a slight variation of the proof of Lemma 14.53. Let Δ be the maximal arity of a compound symbol from $\mathcal{P}^\sharp$, and observe that every node in T has at most Δ successors. Denote by $\{u_1, \dots, u_m\}$ the roots of T_i $(i = 1, \dots, m)$. The non-trivial case is $m > 1$. In this case, each path from the root of T to the nodes $u_i \in \{u_1, \dots, u_m\}$ contains at least one node with a DP applied. Let $\{v_1, \dots, v_n\}$ collect such nodes closest to $\{u_1, \dots, u_m\}$. In particular, we can thus associate to every node $u_i \in \{u_1, \dots, u_m\}$ a node $v_{i'} \in \{v_1, \dots, v_n\}$ and DP $l \to r \in \mathcal{P}^\sharp$ such that $v_{i'} \xrightarrow{\{l_i \to r_i\}}_T \cdot \xrightarrow{\mathcal{S} \cup \mathcal{W}}_T^* u_i$ holds. As $v_{i'}$ has at most Δ successors and $\xrightarrow{\mathcal{S} \cup \mathcal{W}}_T$ is non-branching, it follows that $m \leqslant \Delta \cdot n$. By Lemma 14.43, for $i = 1, \dots, m$ we see $l_i \to r_i \in \mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp)$. As T_i is maximal, $l_i \to r_i \notin \mathcal{R}^\sharp$. Hence $n \leqslant |T|_{\mathsf{Pre}_\mathcal{G}(\mathcal{R}^\sharp)\setminus\mathcal{R}^\sharp}$ and the lemma follows. □

Theorem 14.63 (Dependency Graph Decomposition Processor)**.** *Consider a DP problem* $\mathcal{P}^\sharp = \langle \mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle$ *together with an approximated dependency graph* $\mathcal{G}$ *of* $\mathcal{P}^\sharp$*.*

(1) $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ *is forward closed in* $\mathcal{P}^\sharp$*, and*

(2) $\mathsf{Pre}_\mathcal{G}(\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp) \cap \mathcal{W}_u^\sharp = \varnothing$*.*

The following processor is sound.

$$\frac{\vdash \langle \mathcal{S}_u^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : f \quad \vdash \langle \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_l^\sharp \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp) \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : g}{\vdash \langle \mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}/\mathcal{W}_u^\sharp \cup \mathcal{W}_l^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp\rangle : f * g}\,,$$

for all bounding functions f *and* g *such that* $f(n) \neq 0$ *and* $g(n) \neq 0$ *for all* $n \in \mathbb{N}$*. Here* $f * g$ *denotes the function* h *defined by* $h(n) := f(n) \cdot g(n)$*.*

Proof. Consider a $\mathcal{P}^\sharp$ derivation tree of $t^\sharp \in \mathcal{T}^\sharp$. We estimate $|T|_{\mathcal{S}_u^\sharp \cup \mathcal{S}_l^\sharp \cup \mathcal{S}}$ by a function in $\mathcal{O}(f * g)$, tacitly employing Lemma 14.38. Consider the separation of T as induced by forward closure of $\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp$ into the upper layer $T_\uparrow$, and lower layer that consists of the derivation trees T_i of $t_i^\sharp$ $(i = 1, \ldots, m)$. By Lemma 14.61(2) the trees T_i $(i = 1, \ldots, m)$ can be extended to $\langle \mathcal{S}_l^\sharp \cup \mathcal{S} / \mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp \rangle$ derivation tree T_i' of $t^\sharp$. In particular, the complexity of $\langle \mathcal{S}_l^\sharp \cup \mathcal{S} / \mathcal{W}_l^\sharp \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}_u^\sharp \cup \mathcal{W}_u^\sharp), \mathcal{Q}, \mathcal{T}^\sharp \rangle$ binds applications of $\mathcal{S}_l^\sharp \cup \mathcal{S}$ in T_i, i.e., $|T_i|_{\mathcal{S}_l^\sharp \cup \mathcal{S}} = |T_i'|_{\mathcal{S}_l^\sharp \cup \mathcal{S}}$. Hence $|T_i|_{\mathcal{S}_l^\sharp \cup \mathcal{S}} \in \mathcal{O}(g(|t^\sharp|))$ by the second precondition of the processor. Similar, Lemma 14.61(1) and the first precondition of the processor gives $|T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}} \in \mathcal{O}(f(|t^\sharp|))$. By assumption (ii) and Lemma 14.62 we see $m \leqslant \max\{1, |T|_{\mathsf{Pre}_\mathcal{G}(\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp) \setminus (\mathcal{S}_l^\sharp \cup \mathcal{W}_l^\sharp)}\} \leqslant \max\{1, |T_\uparrow|_{\mathcal{S}_u^\sharp \cup \mathcal{S}}\}$. Putting these bounds together we get

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

The next example shows that the DP problem $\mathcal{P}_{\mathsf{srt}}^\sharp$ has quadratic complexity, using Theorem 14.63.

Example 14.64 (Continued from Example 14.60). Consider the DP problem $\mathcal{P}_{\mathsf{srt}}^\sharp$ from Example 14.58, and let $\mathcal{G}$ denote the DG of $\mathcal{P}_{\mathsf{srt}}^\sharp$. We already observed that the set DPs $\mathcal{S}_{\mathsf{insert}}^\sharp = \{\mathtt{104s}, \mathtt{106}, \mathtt{118}\}$ is forward closed in $\mathcal{P}_{\mathsf{srt}}^\sharp$. As no DP in $\mathsf{Pre}_\mathcal{G}(\mathcal{S}_{\mathsf{insert}}^\sharp) = \{\mathtt{102}\}$ occurs in the weak component of $\mathcal{P}_{\mathsf{srt}}^\sharp$, also the second precondition of Theorem 14.63 is satisfied, and thus

$$\frac{\vdash \langle \{\mathtt{102}\}/\{\mathtt{88s}\} \cup \mathcal{U}_{\mathsf{srt}}\}, \mathcal{R}_\mathsf{K}, \mathcal{T}_\mathsf{K}^\sharp \rangle : n \quad \vdash \mathcal{P}_{\mathsf{insert}}^\sharp : n}{\vdash \mathcal{P}_{\mathsf{srt}}^\sharp : n^2}$$

Using simplification of right-hand sides (Theorem 14.56) and usable rules (Theorem 14.46) the judgement $\vdash \langle \{\mathtt{102}\}/\{\mathtt{88s}\} \cup \mathcal{U}_{\mathsf{srt}}\}, \mathcal{R}_\mathsf{K}, \mathcal{T}_\mathsf{K}^\sharp \rangle : n$ can be reduced to $\vdash \langle \{\mathsf{sort}^\sharp(e :: E) \to \mathsf{sort}^\sharp(E)\}/\{\mathtt{88s}\}, \mathcal{R}_\mathsf{K}, \mathcal{T}_\mathsf{K}^\sharp \rangle : n$. Employing the safe mapping with $\mathsf{safe}(\mathsf{sort}^\sharp) = \mathsf{safe}(\mathsf{forest}^\sharp) = \varnothing$ and the full argument filtering, the judgement can be shown valid using polynomial path orders (Theorem 14.16). Also, it is not difficult to define a complexity pair using polynomial interpretation that proves that the judgement $\vdash \mathcal{P}_{\mathsf{insert}}^\sharp : n$ is valid, by Theorem 14.10. We conclude $\vdash \mathcal{P}_{\mathsf{srt}}^\sharp : n^2$ by Theorem 14.63. $\lhd$

Iterated application of Theorem 14.63 extends to a separate analysis of all cycles, hence our method is closely connected to *cycle analysis* as introduced for termination in [35]. Unlike for cycle analysis, dependency graph decomposition takes the call-structure between cycles into account. As demonstrated in the next example, this is essential in the context of complexity analysis.

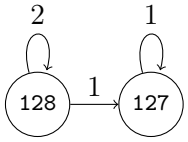
Example 14.65. Let $\mathcal{P}^\sharp_{\mathsf{exp}} := \langle \mathcal{R}^\sharp_{\mathsf{exp}}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle$ where dependency pairs $\mathcal{R}^\sharp_{\mathsf{exp}}$ are

$$\texttt{127}\colon \mathsf{d}^\sharp(\mathsf{s}(x)) \to \mathsf{d}^\sharp(x) \qquad \texttt{128}\colon \mathsf{e}^\sharp(\mathsf{s}(x)) \to \mathsf{c}_2(\mathsf{d}^\sharp(\mathsf{e}(x)), \mathsf{e}^\sharp(x)) \;,$$

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

$$\begin{aligned} &\texttt{129}\colon \mathsf{d}(0) \to 0 && \texttt{130}\colon \mathsf{d}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{d}(x)))\\ &\texttt{131}\colon \mathsf{e}(0) \to \mathsf{s}(0) && \texttt{132}\colon \mathsf{e}(\mathsf{s}(x)) \to \mathsf{d}(\mathsf{e}(x)) \;, \end{aligned}$$

that compute exponentiation on numerals. The following depicts the DG of $\mathcal{P}^\sharp_{\mathsf{exp}}$.



A decomposition of the dependency pairs in $\mathcal{P}^\sharp_{\mathsf{exp}}$ into cycles, as in termination analysis, amounts in our setting to an inference

$$\frac{\vdash \langle \{\texttt{128}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle\colon f \quad \vdash \langle \{\texttt{127}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle\colon g}{\vdash \langle \{\texttt{127},\texttt{128}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle\colon h} \;.$$

This inference is sound for termination analysis [35]. While the complexity of $\langle \{\texttt{127}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle$ and $\langle \{\texttt{128}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle$ is linear, the complexity of $\mathcal{P}^\sharp_{\mathsf{exp}}$ is exponential. Dependency graph decomposition on the other hand extends the sub-problem $\langle \{\texttt{127}\}/\mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle$ obtained from the lower cycle with the two DPs

$$\texttt{133}\colon \mathsf{e}^\sharp(\mathsf{s}(x)) \to \mathsf{d}^\sharp(\mathsf{e}(x)) \qquad \texttt{134}\colon \mathsf{e}^\sharp(\mathsf{s}(x)) \to \mathsf{e}^\sharp(x) \;.$$

The so obtained sub-problem $\langle \{\texttt{127}\}/\{\texttt{133},\texttt{134}\} \cup \mathcal{R}_{\mathsf{exp}}, \mathcal{R}_{\mathsf{exp}}, \mathcal{T}^\sharp_{\mathsf{b}}\rangle$ has exponential complexity. ◁

Finally we want to remark that DG decomposition can overestimate the complexity. For this reason it is not always beneficial to exhaustively apply this processor.

Example 14.66 (Continued from Example 14.64). The set $\{\texttt{118}\}$ constitutes a forward closed set of DPs in the DP problem $\mathcal{P}^\sharp_{\mathsf{insert}}$ depicted in Example 14.60. Consider the two DPs

$$\begin{aligned} &\texttt{104a}\colon && \mathsf{insert}^\sharp(e, f :: E) \to \mathsf{insert?}^\sharp(\mathsf{wt}(e) \leqslant \mathsf{wt}(f), e, f, E)\\ &\texttt{104b}\colon && \mathsf{insert}^\sharp(e, f :: E) \to \mathsf{wt}(e) \leqslant^\sharp \mathsf{wt}(f) \;. \end{aligned}$$

Then we have

$$\mathsf{sep}(\{\texttt{88s}, \texttt{102a}, \texttt{102b}, \texttt{104s}, \texttt{106}\}) = \{\texttt{88s}, \texttt{102a}, \texttt{102b}, \texttt{104a}, \texttt{104b}, \texttt{106}\} \;.$$

Applying the DG decomposition processor yields thus sub-problems

$$\langle \{104s, 106\}/\{88s, 102a, 102b\} \cup \mathcal{U}_{srt}, \mathcal{R}_K, \mathcal{T}_K^\sharp \rangle \ ,$$

for the upper, and

$$\langle \{118\}/\{88s, 102a, 102b, 104a, 104b, 106\} \cup \mathcal{U}_{srt}, \mathcal{R}_K, \mathcal{T}_K^\sharp \rangle \ ,$$

for the lower component. The complexity of both problems is bounded from below by a linear function. Hence employing DG decomposition, we can only proof a quadratic upper bound on the complexity of $\mathcal{P}^\sharp_{insert}$, whereas the complexity of $\mathcal{P}^\sharp_{insert}$ is in fact linear. ◁

14.6.1. Dependency Graph Decomposition in TCT

Relative decomposition guided by the DG as illustrated in Example 14.58, is implemented in TCT by the transformation `decomposeIndependentSG`, which integrates also the various simplifications defined in the previous section. The following reproduces exactly the complexity proof given in Example 14.58.

TCT-interactive 14.12 (Continued from Session 14.9)

```
TCT> apply $ exhaustively decomposeIndependentSG

    Problems simplified. Use 'state' to see the current proof state.

TCT>
```

Here the combinator `exhaustively` is necessary as `decomposeIndependentSG` uses relative decomposition exactly once.

Theorem 14.63, is implemented in TCT by the transformation `decomposeDG`. Per default, TCT chooses as DPs for the upper component a minimal set of DPs which is (i) *backward-closed*, (ii) cyclic, and (iii) contains at least one strict DP. This selection corresponds essentially to the selection of a topmost recursive definition. On the DP problem $\mathcal{P}^\sharp_{srt}$, TCT proceeds exactly as in Example 14.60. This behaviour can be modified by `decomposeDG 'selectLowerBy'` *se*. Here, the selector expression *se* indicates which dependency pairs should occur in the lower component. By extending the selection sufficiently, TCT ensures that the selected set of DPs is forward closed for the given input problem. The operators `solveUpperWith` and `solveLowerWith` can be used to specify a processor that should be used to solve the corresponding sub-problem. The following reproduces the proof given in Example 14.64.

TCT-interactive 14.13 (Continued from Session 14.12)

```
TCT> select [2]
✂------------------------------------------------------------------------------
    Selected Open Problems:
    -----------------------
      Strict DPs:
        { sort^#(::(e, E)) -> c_1(insert^#(e, sort(E)), sort^#(E))
        , insert^#(e, ::(f, E)) ->
          c_2(insert?^#(<=(wt(e), wt(f)), e, f, E), <=^#(wt(e), wt(f)))
        , insert?^#(false(), e, f, E) -> c_3(insert^#(e, E))
        , <=^#(s(n), s(m)) -> c_4(<=^#(n, m)) }
      Weak DPs: { forest^#(graph(N, E)) -> c_5(sort^#(E)) }
✂------------------------------------------------------------------------------
```

TCT-interactive 14.14 (Continued from Session 14.13)

```
      Weak Trs:
        { wt(edge(n, w, m)) -> w
        , sort([]()) -> []()
        , sort(::(e, E)) -> insert(e, sort(E))
        , insert(e, []()) -> ::(e, []())
        , insert(e, ::(f, E)) -> insert?(<=(wt(e), wt(f)), e, f, E)
        , <=(0(), 0()) -> true()
        , <=(0(), s(m)) -> true()
        , <=(s(n), 0()) -> false()
        , <=(s(n), s(m)) -> <=(n, m)
        , insert?(true(), e, f, E) -> ::(e, ::(f, E))
        , insert?(false(), e, f, E) -> ::(f, insert(e, E)) }
      StartTerms: basic terms
      Strategy: innermost
    --------------------------------------------------------------------------
TCT> apply $ decomposeDG `solveUpperWith` (cleanSuffix
                                        >>| spopstarPS `withDegree` Just 1)

    Problems simplified. Use 'state' to see the current proof state.

TCT> apply $ poly `withDegree` Just 1

    Problems simplified. Use 'state' to see the current proof state.

TCT>
```

It remains to handle the previously deselected open sub-problems generated in Session 14.12, viz, the DP problems $\mathcal{P}^\sharp_{\mathsf{Kk}}$ and $\mathcal{P}^\sharp_{\mathsf{part}}$. Using dependency graph decomposition to simplify $\mathcal{P}^\sharp_{\mathsf{Kk}}$, iterated application of polynomial interpretations together with predecessor estimation allow us to finally conclude quadratic innermost runtime complexity of the running example $\mathcal{R}_{\mathsf{K}}$.

TCT-interactive 14.15 (Continued from Session 14.14)

```
TCT> select [1..]
✂.....................................................................................
TCT> apply $ decomposeDG

    Problems simplified. Use 'state' to see the current proof state.

TCT> apply $ exhaustively $ poly `withBits` 3 `withDegree` Just 1
                                              `withPEOn` selAnyOf selStricts

    Problems simplified. Use 'state' to see the current proof state.

TCT> apply empty

    Hurray, the problem was solved with certificate YES(O(1),O(n^2)).
    Use 'proof' to show the complete proof.

TCT>
```

14.7. Small Polynomial Path Orders and Dependency Pairs

In Section 14.3 we have shown that small polynomial path orders can be used together with argument filterings in our framework, provided that the underlying argument filtering is non-collapsing on symbols defined in the strict component. The following example shows that this restriction is necessary to derive the bound given in Theorem 14.16.

Example 14.67. Let $m \in \mathbb{N}$. Consider the innermost dependency pair problem $\mathcal{P}^\sharp_{\mathsf{2m}} := \langle \mathcal{S}^\sharp_{\mathsf{exp_m}} / \mathcal{W}_{\mathsf{exp_m}}, \mathcal{W}_{\mathsf{exp_m}}, \{\mathsf{exp}^\sharp_{\mathsf{2m}}(\mathbf{n}) \mid n \in \mathbb{N}\}\rangle$, where $\mathcal{S}^\sharp_{\mathsf{exp_m}}$ constitutes of the dependency pairs

$$
\begin{aligned}
&135\colon && \mathsf{exp}^\sharp_{\mathsf{2m}}(x;) \to \mathsf{g}^\sharp_{\mathsf{m}}(x, x; \mathsf{exp}_{\mathsf{m}}(x;)) \\
&136_{\mathsf{0}}\colon && \mathsf{g}^\sharp_{\mathsf{0}}(x, y; \mathsf{s}(z)) \to \mathsf{g}^\sharp_{\mathsf{0}}(x, y; z) \\
&136_{\mathsf{i}}\colon && \mathsf{g_i}^\sharp(\mathsf{s}(x), y; z) \to \mathsf{c}_2(; \mathsf{g_{i-1}}^\sharp(y, y; z), \mathsf{g_i}^\sharp(x, y; z)) \quad (i = 1, \dots, m)
\end{aligned}
$$

and the TRS $\mathcal{W}_{\mathsf{exp_m}}$ is given by the rules

$$
\begin{aligned}
&137\colon && \mathsf{exp}_{\mathsf{m}}(x) \to \mathsf{f}_{\mathsf{m}}(x, \dots, x; 0) \\
&138\colon && \mathsf{f}_{\mathsf{0}}(; y) \to \mathsf{s}(; y) \\
&139_{\mathsf{i}}\colon && \mathsf{f_i}(0, x_2, \dots, x_i; y) \to y && (i = 1, \dots, m) \\
&140_{\mathsf{i}}\colon && \mathsf{f_i}(\mathsf{s}(; x_1), x_2, \dots, x_i; y) \to \mathsf{f_i}(x_1, x_2, \dots, x_i; \mathsf{f_{i-1}}(x_2, \dots, x_i; y)) \\
& && && (i = 1, \dots, m)\,.
\end{aligned}
$$

The complexity of $\mathcal{P}^\sharp_{\mathsf{2m}}$ is bounded from below by a polynomial of degree $2 \cdot m$. To see this, fix $n \in \mathbb{N}$ and observe that $\mathsf{f}_i(\mathbf{n_1}, \dots, \mathbf{n_i}; \mathbf{k})$ reduces to $\mathbf{n_1} \cdot \ldots \cdot \mathbf{n_i} + \mathbf{k}$, hence $\mathsf{exp}_{\mathsf{m}}(\mathbf{n})$ reduces to $\mathbf{n^m}$. It is not difficult to construct a $\mathcal{P}^\sharp_{\mathsf{2m}}$-derivation tree of $\mathsf{g_i}^\sharp(\mathbf{n}, \mathbf{n}; \mathsf{exp}_{\mathsf{m}}(\mathbf{n}))$ $(i = 1, \dots, m)$ that contains $(n-1)^i$ leafs labeled by $\mathsf{g_0}^\sharp(\mathbf{n}, \mathbf{n}; \mathbf{n^m})$. As the latter term admits $n^i - 1$ dependency pair steps, we see that for $i = m$ a maximal $\mathcal{P}^\sharp_{\mathsf{2m}}$-derivation of $\mathsf{exp}^\sharp_{\mathsf{2m}}(\mathbf{n};)$ contains $(n-1)^m \cdot (n^m - 1) \in \Omega(n^{2 \cdot m})$ nodes labeled by the DP 136_{0}.

Note that Theorem 14.16 is not applicable, due to the restrictions on the employed argument filtering. On the other hand, using an admissible precedence and safe mapping as indicated in the rules, one can show $\mathcal{S}^\sharp_{\mathsf{exp_m}} \subseteq {>^\pi_{\mathsf{spop^*_{ps}}}}$ and $\mathcal{W}_{\mathsf{exp_m}} \subseteq {\gtrsim^\pi_{\mathsf{spop^*_{ps}}}}$ with the argument filtering π given by $\pi(\mathsf{g}^\sharp_{\mathsf{0}}) := 3$, and $\pi(f) = [1, \dots, k]$ for every other symbol f/k occurring in $\mathcal{P}^\sharp_{\mathsf{2m}}$. ◁

Consider an innermost DP problems where the strict component consists only of dependency pairs, and where each compound symbol is a constant or a unary function symbol. The restriction on compound symbols implies that derivation trees degenerate to sequences, and excludes the above counter example. And indeed we can include the considered special case in Theorem 14.16. We validate this observation in the next theorem. Note that we replace here the assumed restriction on compound symbols with a slightly weaker restriction on usable argument positions. The theorem should be considered as an intermediate result.

In Theorem 14.69 below we then consider the case where neither a restriction is put on compound symbols nor on the employed argument filtering.

Theorem 14.68. *Consider an innermost DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ where $\mathcal{S}^\sharp$, $\mathcal{W}^\sharp$ and $\mathcal{W}$ are constructor TRSs. Let μ denote a usable replacement map for $\mathcal{S}^\sharp \cup \mathcal{W}^\sharp$ in $\mathcal{P}^\sharp$ such that for every compound symbols c_k in $\mathcal{P}^\sharp$, $\mu(\mathsf{c}_k) \subseteq \{i\}$ for some $i \in \{1, \dots, k\}$ holds. Let π denote an argument filtering on the symbols in $\mathcal{P}$ that agrees with μ. Let $\mathcal{K}_\pi \subseteq \mathcal{D}^\sharp_\pi$ denote a set of recursive function symbols, and $\gtrsim$ an admissible precedence. The following processor is sound, for $d := \max\{0\} \cup \{\mathsf{rd}_{\gtrsim, \mathcal{K}_\pi}(f_\pi) \mid f_\pi \in \mathcal{F}^\sharp_\pi\}$.*

$$\frac{\mathcal{S}^\sharp \subseteq {>}^\pi_{\mathsf{spop}^*_{\mathsf{ps}}} \quad \mathcal{W}^\sharp \cup \mathcal{W} \subseteq {\gtrsim}^\pi_{\mathsf{spop}^*_{\mathsf{ps}}}}{\vdash \langle \mathcal{S}^\sharp / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon n^{\max(1,d)}} \ ,$$

Proof. We revise the proof of Theorem 14.16. Consider a $\mathcal{P}^\sharp$ derivation D of $s^\sharp \in \mathcal{T}^\sharp$. By Lemma 14.20, this derivation is of the form

$$D \colon s^\sharp = C_0[\vec{t}_1] \to_{\mathcal{P}^\sharp} C_1[\vec{t}_1] \to_{\mathcal{P}^\sharp} C_2[\vec{t}_2] \to_{\mathcal{P}^\sharp} C_3[\vec{t}_3] \dots \ ,$$

where the contexts C_i ($i \in \mathbb{N}$) are maximal compound contexts. Consider a term $C_i[\vec{t}_i]$ in the above derivation, and call a term $t \in \vec{t}_i$ a *principal* term if it occurs at a usable argument position $\mathcal{P}\mathrm{os}_\mu(C_i[\vec{t}_i])$ in $C_i[\vec{t}_i]$. By the assumption that every compound symbol has at most one argument position usable for the dependency pairs $\mathcal{S}^\sharp \cup \mathcal{W}^\sharp$, every term $C_i[\vec{t}_i]$ in the above sequence has at most one principal term.

As in Theorem 14.16, abbreviate $\mathcal{N} := \mathcal{N}_{\mathcal{T}(\mathcal{C}_\pi)}$, $\mathcal{I} := \mathcal{I}_{\mathcal{T}(\mathcal{C}_\pi)}$ and define $G(t) := \mathsf{G}_{\mathcal{K},\ell}(\mathcal{I}(t))$ for ℓ the maximal size of a right-hand side in $\pi(\mathcal{S}^\sharp \cup \mathcal{W}^\sharp \cup \mathcal{W})$. Let $\mathcal{S}^\sharp_{\mathsf{nc}} := \{l \to r \in \mathcal{S}^\sharp \mid \pi \text{ is non-collapsing on the root of } l\}$ and set $\mathcal{S}^\sharp_{\mathsf{c}} := \mathcal{S}^\sharp \setminus \mathcal{S}^\sharp_{\mathsf{nc}}$. For simplicity, suppose first that the terms $C_i[\vec{t}_i]$ contain no garbage. Observe that the problem $\langle \mathcal{S}^\sharp_{\mathsf{nc}} / \mathcal{S}^\sharp_{\mathsf{c}} \cup \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ that accounts for non-collapsed dependency pairs $\mathcal{S}^\sharp_{\mathsf{nc}}$ satisfies the assumptions of Theorem 14.16. As we have observed in the proof of the theorem, $G(s^\sharp) \in \mathcal{O}(|s|^d)$ gives an upper bound on the number of applications of $\mathcal{S}^\sharp_{\mathsf{nc}}$ in D. To account also the steps with respect to $\mathcal{S}^\sharp_{\mathsf{c}}$, we exploit that ${>}_{\mathsf{spop}^*_{\mathsf{ps}}}$ collapses to $\rhd/_\approx$ on values. As a consequence, in a step $C_i[\vec{t}_i] \xrightarrow{\mathcal{Q}}_{\mathcal{S}^\sharp_{\mathsf{c}}} C_{i+1}[\vec{t}_{i+1}]$ in D the depth of the *argument filtered principal term* decreases, i.e., $\mathsf{dp}(\pi(u_i)) > \mathsf{dp}(\pi(u_{i+1}))$ for principal terms $u_i \in \vec{t}_i$ and $u_{i+1} \in \vec{t}_{i+1}$. For steps not due to $\mathcal{S}^\sharp_{\mathsf{c}}$, we exploit that either $\pi(u_i)$ is equivalent to $\pi(u_{i+1})$, or $G(\pi(u_i)) > G(\pi(u_{i+1}))$ holds by the claim of Theorem 14.16. Only in the latter case the depth of the argument filtered principal term might increase, but only by a constant Δ.

To define a global measure function H on all steps due to $\mathcal{S}^\sharp$ in $\mathcal{P}^\sharp$ derivations starting from $\mathcal{T}^\sharp$, define the binary relation $\sqsupset_\Delta$ on pairs of natural numbers such that $\langle m_1, m_2 \rangle \sqsupset_\Delta \langle n_1, n_2 \rangle$ if either $m_1 > n_1$ and $m_2 + \Delta \geqslant n_2$ or $m_1 \geqslant n_1$ and $m_2 > n_2$. Let $\sqsupseteq_\Delta$ denote the reflexive closure of $\sqsupset_\Delta$. An easy induction reveals that the maximal length of $\sqsupset_\Delta$-descending sequences starting from $\langle n_1, n_2 \rangle$ is bounded by $n_1 \cdot (\Delta + 1) + n_2$.

Let $t^\bot$ denote the term obtained by replacing garbage terms in t by $\bot \in \mathcal{C}$, as in the proof of Theorem 14.16. We define

$$H(t) := \begin{cases} \langle G(\pi(u^\bot)), \mathsf{dp}(\pi(u^\bot))\rangle & \text{if } u \text{ is the principal term in } t, \\ \langle 0, 0\rangle & \text{if } t \text{ has no principal term.} \end{cases}$$

The theorem is a consequence of the following claim.

Claim. Suppose $s \in \to^*_{\mathcal{P}^\sharp}(\mathcal{T}^\sharp)$ with $s \in \mathcal{N}$. Then

$$s \xrightarrow{\mathcal{Q}}_{\mathcal{S}^\sharp} t \implies H(s^\bot) \sqsupset_\Delta H(t^\bot) \quad \text{and} \quad s \xrightarrow{\mathcal{Q}}_{\mathcal{W}^\sharp \cup \mathcal{W}} t \implies H(s^\bot) \sqsupseteq_\Delta H(t^\bot)\,,$$

where $\Delta := \max\{\mathsf{dp}(r) \mid l \to r \in \pi(\mathcal{S}^\sharp_{\mathsf{nc}}) \cup \pi(\mathcal{W}^\sharp) \cup \pi(\mathcal{W})\}$.

Proof of Claim. Let $l \to r \in \mathcal{P}^\sharp$, and consider $s \in \to^*_{\mathcal{P}^\sharp}(\mathcal{T})$ with $s \in \mathcal{N}$. Suppose $s = C[\vec{s}] \xrightarrow{\mathcal{Q}}_{\{l\to r\}} C[\vec{t}] = t$ with compound context C and substitution $\sigma : \mathcal{V} \to \mathrm{NF}(\mathcal{Q})$.

Observe that if s lacks a principle term, so does t. In this case $l \to r \notin \mathcal{S}^\sharp$ we conclude the claim with $H(s) = \langle 0, 0\rangle = H(t)$. Also the case where only t lacks a principle term is trivial. Hence consider the only interesting case when $s_i \xrightarrow{\mathcal{Q}}_{\mathcal{R}} t_i$ holds for $s_i \in \vec{s}$ the principle term of s and t_i the principle term of t. We perform case analysis on the applied rewrite rule $l \to r \in \mathcal{P}^\sharp$.

Suppose first that $l \to r \in \mathcal{S}^\sharp$ where by assumption $\pi(l) \succ^\pi_{\mathsf{spop}^*_{\mathsf{ps}}} \pi(r)$. Then $s_i = l\sigma$ and $t_i = r\sigma$ since $l \to r$ is a dependency pair. As in the claim of Theorem 14.16, if $l \to r \in \mathcal{S}^\sharp_{\mathsf{nc}}$ then $G(\pi(s_i^\bot)) > G(\pi(t_i^\bot))$ holds, where

$$\pi(s_i^\bot) = \pi(l\sigma_\bot) \xrightarrow{\mathcal{T}(\mathcal{C}_\pi)}_{\{l\to r\}} \pi(r\sigma_\bot)\,.$$

The latter implies that

$$\mathsf{dp}(\pi(s_i^\bot)) + \Delta \geqslant \mathsf{dp}(\pi(r\sigma_\bot)) \geqslant \mathsf{dp}(\pi((r\sigma)^\bot)) = \mathsf{dp}(\pi(t_i^\bot))\,,$$

we conclude $H(s) \sqsupset_\Delta H(t)$. If on the other hand $l \to r \in \mathcal{S}^\sharp_{\mathsf{nc}}$, the claim of Theorem 14.16 gives only $G(\pi(s_i^\bot)) \geqslant G(\pi(t_i^\bot))$. Instead we have however

$$\pi(s_i^\bot) = \pi(l\sigma_\bot) \rhd\!\!/_\approx \pi(r\sigma_\bot)\,,$$

and hence $\mathsf{dp}(\pi(s_i^\bot)) > \mathsf{dp}(\pi(r\sigma_\bot)) \geqslant \mathsf{dp}(\pi(t_i^\bot))$. Again we see $H(s) \sqsupset_\Delta H(t)$.

Consider now $l \to r \in \mathcal{W}^\sharp \cup \mathcal{W}$. If π collapses the root of l, the order constraints give $\pi(l) \trianglerighteq\!\!/_\approx \pi(r)$ by the assumption that l is constructor based. It is not difficult to verify that in this case the claim holds as above, using a simple inductive argument for the case $l \to r \in \mathcal{W}$. Hence suppose π does not collapse on the root of l. For this case, we prove by induction on the rewrite position in $s_i \xrightarrow{\mathcal{Q}}_{\{l\to r\}} t_i$ that

$$\langle G(\pi(s_i^\bot)), \mathsf{dp}(\pi(s_i^\bot))\rangle \sqsupseteq_\Delta \langle G(\pi(t_i^\bot)), \mathsf{dp}(\pi(t_i^\bot))\rangle\,,$$

holds. The base case follows as in the considered case $l \to r \in \mathcal{S}^\sharp$ above if $\pi(l) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \pi(r)$ holds. For $\pi(l) \approx_{\mathsf{s}} \pi(r)$, we have $G(\pi((l\sigma)^\bot)) \geqslant G(\pi((r\sigma)^\bot))$

and $\mathsf{dp}(\pi((l\sigma)^\bot)) = \mathsf{dp}(\pi(r\sigma_\bot)) \geqslant \mathsf{dp}(\pi((r\sigma)^\bot))$. thus consider the inductive step

$$s_i = f(u_1, \ldots, u_j, \ldots, u_n) \xrightarrow{\mathcal{Q}}_{\{l \to r\}} f(u_1, \ldots, v_j, \ldots, u_n) \,,$$

with $u_j \xrightarrow{\mathcal{Q}}_{\{l \to r\}} v_j$, where without loss of generality $s_i^\bot = f(u_1^\bot, \ldots, u_j^\bot, \ldots, u_n^\bot)$ and $t_i^\bot = f(u_1^\bot, \ldots, v_j^\bot, \ldots, u_n^\bot)$ holds. If $j \notin \pi(f)$ or $\pi(f) = j$ the result is immediate, or follows directly from induction hypothesis. Otherwise $j \in \pi(f)$. Since $s \in \mathcal{N}$, also $s_i \in \mathcal{N}$ and thus the term $\pi(u_j^\bot)$ occurs as safe argument to $\pi(s_i^\bot)$. By case analysis on the induction hypothesis the claim follows. □

Using the claim, a reduction

$$f(\vec{v}) \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_1 \cdots \xrightarrow{\mathcal{Q}}_{\mathcal{S}/\mathcal{W}} t_n \,,$$

translates into a descent

$$H(f(\vec{v})^\bot) \sqsupset_\Delta H(t_1^\bot) \sqsupset_\Delta \cdots \sqsupset_\Delta H(t_n^\bot) \,.$$

For $f(\vec{v}) \in \mathcal{T}^\sharp$ we thus have $n \leqslant G(\pi(f(\vec{v})^\bot)) \cdot (\Delta + 1) + \mathsf{dp}(f(\vec{v}))$, as we have observed $G(\pi(f(\vec{v})^\bot)) \in \mathcal{O}(|f(\vec{v})|^d)$ already in Theorem 14.16 we conclude the theorem. □

Our final theorem is obtained as an application of dependency graph decomposition, using Theorem 14.16 and Theorem 14.68 to solve the two generated sub-problems. Example 14.67 indicates that the deduced bound is tight in the general case. Noteworthy, this also shows that the bound deduced by dependency graph decomposition is essentially optimal for the general case.

Theorem 14.69. *Consider an innermost DP problem $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ where $\mathcal{S}^\sharp$, $\mathcal{W}^\sharp$ and $\mathcal{W}$ are constructor TRSs. Let μ denote a usable replacement map for $\mathcal{S}^\sharp \cup \mathcal{W}^\sharp$ in $\mathcal{P}^\sharp$, and let π denote an argument filtering on the symbols in $\mathcal{P}$ that agrees with μ. Let $\mathcal{K}_\pi \subseteq \mathcal{D}_\pi^\sharp$ denote a set of recursive function symbols, and $\gtrsim$ an admissible precedence where $c_\pi \sim d_\pi$ only holds for non-compound symbols $c, d \notin \mathcal{C}\mathsf{om}$. The following processor is sound, for $d := \max\{0\} \cup \{\mathsf{rd}_{\gtrsim, \mathcal{K}_\pi}(f_\pi) \mid f_\pi \in \mathcal{F}_\pi^\sharp\}$.*

$$\frac{\mathcal{S}^\sharp \subseteq >^\pi_{\mathsf{spop}^*_{\mathsf{ps}}} \quad \mathcal{W}^\sharp \cup \mathcal{W} \subseteq \gtrsim^\pi_{\mathsf{spop}^*_{\mathsf{ps}}}}{\vdash \langle \mathcal{S}^\sharp / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \colon n^{\max(1, 2 \cdot d)}} \,.$$

Proof. Let $\mathcal{P}^\sharp = \langle \mathcal{S}^\sharp / \mathcal{W}^\sharp \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle$ be an innermost DP problem which satisfies the preconditions of the processor. To simplify matters, we assume the stronger property that $\pi(l) \approx_{\mathsf{s}} \pi(r)$ holds for every DP $l \to r \in \mathcal{W}^\sharp$. The theorem then follows also in the general case, by shifting DPs $l \to r \in \mathcal{W}^\sharp$ with $l >^\pi_{\mathsf{spop}^*_{\mathsf{ps}}} r$ to the strict component.

Consider partitions $\mathcal{W}^\sharp := \mathcal{W}^\sharp_{\mathsf{c}} \cup \mathcal{W}^\sharp_{\mathsf{nc}}$ and $\mathcal{S}^\sharp := \mathcal{S}^\sharp_{\mathsf{c}} \cup \mathcal{S}^\sharp_{\mathsf{nc}}$ such that $\mathcal{W}^\sharp_{\mathsf{c}} \cup \mathcal{S}^\sharp_{\mathsf{c}}$ collects all DPs collapsed by the argument filtering, i.e., $f^\sharp(l_1, \ldots, l_k) \to r \in \mathcal{W}^\sharp_{\mathsf{c}} \cup \mathcal{S}^\sharp_{\mathsf{c}}$ if and only if $\pi(f^\sharp) = i$ for some $i \in \{1, \ldots, k\}$. We apply the DG

decomposition processor, where the lower component is given by the collapsed rules $\mathcal{W}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}$. Define

$$\begin{aligned}\mathcal{P}^\sharp_\mathsf{nc} &:= \langle \mathcal{S}^\sharp_\mathsf{nc} / \mathcal{W}^\sharp_\mathsf{nc} \cup \mathcal{W}, \mathcal{Q}, \mathcal{T}^\sharp \rangle \text{ , and} \\ \mathcal{P}^\sharp_\mathsf{c} &:= \langle \mathcal{S}^\sharp_\mathsf{c} / \mathcal{W}^\sharp_\mathsf{c} \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}^\sharp_\mathsf{nc} \cup \mathcal{W}^\sharp_\mathsf{nc}), \mathcal{Q}, \mathcal{T}^\sharp \rangle \text{ .}\end{aligned}$$

Let $\mathcal{G}$ denote the dependency graph of $\mathcal{P}^\sharp$. Consider and edge from $s^\sharp \to \textsc{com}(t^\sharp_1, \dots, t^\sharp_k) \in \mathcal{S}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}$ to $u^\sharp \to \textsc{com}(v^\sharp_1, \dots, v^\sharp_l)$ in $\mathcal{G}$ labeled by $i \in \{1, \dots, k\}$. This means that the roots of $t^\sharp_i$ and $u^\sharp_i$ coincide. Using that π collapses the root of the constructor based term $s^\sharp$, we have $\pi(s^\sharp) \in \mathcal{T}(\mathcal{C}_\pi)$. Using that the precedence $\succsim$ underlying $\succsim_{\mathsf{spop}^*_\mathsf{ps}}$ is admissible, similar to Lemma 9.15 we see that the root of $t^\sharp_i$, and thus of $u^\sharp$, is collapsed by π. We conclude $u^\sharp \to \textsc{com}(v^\sharp_1, \dots, v^\sharp_l) \in \mathcal{S}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}$, i.e., $\mathcal{S}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}$ is forward closed in $\mathcal{P}^\sharp$.

Inversely, suppose there is an edge from $u^\sharp \to \textsc{com}(v^\sharp_1, \dots, v^\sharp_l) \in \mathcal{S}^\sharp_\mathsf{nc} \cup \mathcal{W}^\sharp_\mathsf{nc}$ to $s^\sharp \to \textsc{com}(t^\sharp_1, \dots, t^\sharp_k) \in \mathcal{S}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}$ labeled by i in the dependency graph $\mathcal{G}$. This implies that $\pi(v^\sharp_i) \in \mathcal{T}(\mathcal{C}_\pi)$, since π is non-collapsing on the root of $u^\sharp$ we can strengthen $\pi(u^\sharp) \succsim^\pi_{\mathsf{spop}^*_\mathsf{ps}} \textsc{com}(v^\sharp_1, \dots, v^\sharp_l)$ to $\pi(u^\sharp) \succ^\pi_{\mathsf{spop}^*_\mathsf{ps}} \textsc{com}(v^\sharp_1, \dots, v^\sharp_l)$ and hence even $u^\sharp \to \textsc{com}(v^\sharp_1, \dots, v^\sharp_l) \in \mathcal{S}^\sharp_\mathsf{nc}$ by the assumption $\pi(l) \approx_\mathsf{s} \pi(r)$ for $l \to r \in \mathcal{W}^\sharp_\mathsf{nc}$. We conclude $\mathsf{Pre}_\mathcal{G}(\mathcal{W}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}) \subseteq \mathcal{S}^\sharp_\mathsf{c}$. This establishes the preconditions of the DG decomposition processor (Theorem 14.63), hence the inference

$$\frac{\vdash \mathcal{P}^\sharp_\mathsf{nc} \colon n^d \quad \vdash \mathcal{P}^\sharp_\mathsf{c} \colon n^{\max(1,d)}}{\vdash \mathcal{P}^\sharp \colon n^{\max(1,2\cdot d)}} \text{ ,}$$

is sound.

By construction π is non-collapsing on defined symbols in $\mathcal{S}^\sharp_\mathsf{nc}$. Since $\mathcal{S}^\sharp_\mathsf{nc} \subseteq \mathcal{S}^\sharp \cup \mathcal{W}^\sharp$, the usable replacement μ constitutes also a usable replacement map for $\mathcal{S}^\sharp_\mathsf{nc}$ in $\mathcal{P}^\sharp_\mathsf{nc}$. Thus μ together with π satisfy the pre-conditions of Theorem 14.16, since $\mathcal{S}^\sharp_\mathsf{nc} \subseteq \succ^\pi_{\mathsf{spop}^*_\mathsf{ps}}$ and $\mathcal{W}^\sharp_\mathsf{nc} \cup \mathcal{W} \subseteq \succsim^\pi_{\mathsf{spop}^*_\mathsf{ps}}$ we conclude thus $\vdash \mathcal{P}^\sharp_\mathsf{nc} \colon n^d$.

The remaining judgement $\vdash \mathcal{P}^\sharp_\mathsf{c} \colon n^{\max(1,d)}$ is verified with the help of Theorem 14.68. Since μ is a usable replacement map for dependency pairs in $\mathcal{P}^\sharp$, by construction of $\mathcal{P}^\sharp_\mathsf{c}$ it is not difficult to verify that μ is also a usable replacement map for dependency pairs in $\mathcal{P}^\sharp_\mathsf{c}$. Consider a dependency pair $s^\sharp \to t \in \mathcal{S}^\sharp_\mathsf{c} \cup \mathcal{W}^\sharp_\mathsf{c}$ for $t = \textsc{com}(t^\sharp_1, \dots, t^\sharp_k)$. We observed already above that $\pi(s^\sharp) \trianglerighteq_{/\approx} \pi(t^\sharp)$ with $\pi(t^\sharp) \in \mathcal{T}(\mathcal{C}_\pi)$ holds. In particular, $\pi(t^\sharp)$ contains no compound symbol by the assumption on the equivalence $\sim$. Thus if t contains a compound symbol c_k, then $\pi(\mathsf{c}_k) = i$ for some $i = 1, \dots, k$. Using that π agrees with μ, we see that $\mu(\mathsf{c}_k) \subseteq \{i\}$ holds. Hence μ together with π satisfy the pre-conditions of Theorem 14.16. Since the order constraints $\mathcal{S}^\sharp_\mathsf{c} \subseteq \succ^\pi_{\mathsf{spop}^*_\mathsf{ps}}$ and $\mathcal{W}^\sharp_\mathsf{c} \cup \mathcal{W} \cup \mathsf{sep}(\mathcal{S}^\sharp_\mathsf{nc} \cup \mathcal{W}^\sharp_\mathsf{nc}) \subseteq \succsim^\pi_{\mathsf{spop}^*_\mathsf{ps}}$ are satisfied, we conclude thus $\vdash \mathcal{P}^\sharp_\mathsf{c} \colon n^{\max(1,d)}$. Note that the inclusion $\mathsf{sep}(\mathcal{S}^\sharp_\mathsf{nc} \cup \mathcal{W}^\sharp_\mathsf{nc}) \subseteq \succsim^\pi_{\mathsf{spop}^*_\mathsf{ps}}$ follows by a straight forward case analysis on the assumptions $\mathcal{S}^\sharp_\mathsf{nc} \subseteq \mathcal{S}^\sharp \subseteq \succ^\pi_{\mathsf{spop}^*_\mathsf{ps}}$ and $\mathcal{W}^\sharp_\mathsf{nc} \subseteq \mathcal{W}^\sharp \subseteq \succsim^\pi_{\mathsf{spop}^*_\mathsf{ps}}$.□

Our final example clarifies that the we can indeed not allow the precedence to identify compound symbols with non-compound symbols.

Example 14.70. Consider the predicative recursive constructor TRS $\mathcal{R}_{\mathsf{dup}'}$ consisting of the rules

$$\texttt{141:}\ \mathsf{f}(0;) \to \mathsf{leaf} \quad \texttt{142:}\ \mathsf{f}(\mathsf{s}(;x);) \to \mathsf{dup}(;\mathsf{f}(x;)) \quad \texttt{143:}\ \mathsf{dup}(;t) \to \mathsf{u}(;\mathsf{b}(;t,t)) \;.$$

This rewrite system is a variation of $\mathcal{R}_{\mathsf{dup}}$ from Example 9.12, where inner nodes of the constructed binary tree are encoded by a constructor term $\mathsf{u}(;\mathsf{b}(;\cdot,\cdot))$. Consider the innermost DP problem $\langle \mathcal{S}^\sharp_{\mathsf{dup}'}/\mathcal{R}_{\mathsf{dup}'}, \mathcal{R}_{\mathsf{dup}'}, \{\mathsf{F}^\sharp(\mathbf{n}) \mid n \in \mathbb{N}\}\rangle$ where $\mathcal{S}^\sharp_{\mathsf{dup}'}$ consisting of the two dependency pairs

$$\texttt{144:}\ \mathsf{F}^\sharp(x;) \to \mathsf{G}^\sharp(\mathsf{f}(x;)) \qquad \texttt{145:}\ \mathsf{G}^\sharp(\mathsf{u}(;\mathsf{b}(;t_1,t_2))) \to \mathsf{c}_2(;\mathsf{G}^\sharp(t_1),\mathsf{G}^\sharp(t_2)) \;.$$

Since $\mathsf{G}^\sharp$ traverses a full binary tree of height n in the reduction of a starting term $\mathsf{F}^\sharp(\mathbf{n})$, the complexity function of the considered DP problem is not bounded by a polynomial from above.

Let π denote the argument filtering that collapses only on $\mathsf{G}^\sharp$. Then we can prove

$$\mathsf{F}^\sharp_\pi(x;) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \mathsf{G}^\sharp_\pi(\mathsf{f}_\pi(x;)) \qquad \mathsf{u}_\pi(;\mathsf{b}_\pi(;t_1,t_2)) \succ_{\mathsf{spop}^*_{\mathsf{ps}}} \mathsf{c}_{2,\pi}(t_1,t_2) \;,$$

hence $\mathcal{S}^\sharp_{\mathsf{dup}'} \subseteq \succ^\pi_{\mathsf{spop}^*_{\mathsf{ps}}}$, but also $\mathcal{R}_{\mathsf{dup}'} \subseteq \succsim^\pi_{\mathsf{spop}^*_{\mathsf{ps}}}$, for an admissible precedence which satisfies

$$\mathsf{f}_\pi \succ \mathsf{dup}_\pi, 0_\pi \qquad \mathsf{dup}_\pi \succ \mathsf{u}_\pi, \mathsf{b}_\pi \qquad \mathsf{F}^\sharp_\pi \succ \mathsf{G}^\sharp_\pi, \mathsf{f}_\pi \qquad \mathsf{b}_\pi \sim \mathsf{c}_{2,\pi} \;.$$

If we allow the latter equivalence in Theorem 14.69, we would thus wrongly conclude that the complexity of the considered innermost DP problem is polynomial. ◁

Chapter 15.

Experimental Evaluation

In this chapter we present our experimental evaluation of TCT.

Data Sets. We have compiled a collection of examples specifically targeted at runtime complexity analysis. This collection, the data set RC, is mostly a compilation of examples found in the literature on the topic. It also contains some natural examples not found in the literature. For instance, it contains various sorting algorithms on lists and simple functions expressed in Peano arithmetic.[1]

To test the applicability of runtime complexity analysis of TRSs in the context of program analysis, we have employed a naive (but complexity preserving) transformation of RaML programs considered in [44] into TRSs.[2] We remark that in this transformation, we do not take typing information into account. We however use the call-by-value strategy adopted by RaML to obtain an innermost runtime complexity problem. Also, build-in operations like comparison operations on Integers and Boolean operations are assigned unitary cost. This is achieved by modelling their semantics with rewrite rules occurring in the weak component of the constructed problem. Data set RaML collects the examples obtained by this translation from the example collection available in the source repository of the RaML prototype.[3]

Setup. All experiments were conducted on a machine with a 16 core Intel® Core™ i7–3930K CPU running at 3.20GHz, and 32Gb of RAM. The employed version of TCT is version 2.0. Below we compare various techniques implemented in TCT to the complexity tool CaT (for runtime complexity analysis) and AProVE (for innermost complexity analysis). The employed binaries of these tools are the ones used in the annual termination competition of 2012.[4] Every test was run with a generous timeout of 300 seconds.

In tables below, we contrast the strength of the various proof techniques implemented in TCT. Precisely, we tested TCT in following configurations.

[1] We decided against the use of the *termination problem data base* (*TPDB* for short) as data. Unarguably, the TPDB is the most extensive selection of examples available. Still, it was primarily intended as a means to assess the strength of termination provers, and falls short of interesting examples for runtime complexity analysis.

[2] The corresponding tool is available online at `http://cl-informatik.uibk.ac.at/cbr/tools/RaML/`.

[3] Available online at `http://raml.tcs.ifi.lmu.de/`.

[4] Available through `http://termcomp.uibk.ac.at/`.

- Direct: With this configuration we check the power of direct methods. As processors, complexity pairs (Theorem 14.10) and small polynomial path orders (Theorem 14.16) are employed. Complexity pairs are synthesised as described in Section 14.1, using matrix interpretations of dimension one to four, as well as polynomial interpretations up to degree three.

- RD: With this configuration we indicate the strength of Zankl and Korp's decomposition processor in combination with complexity pairs (Theorem 14.13). To synthesise complexity pairs, we used the same configuration as for the direct approach.

- DP+RD: In this configuration TCT proceeds as above, except that as a first proof step weak dependency pairs (Theorem 14.24) for runtime, or dependency tuples (Theorem 14.29) for innermost runtime complexity analysis, are applied. In the former case TCT also tries to establish the *weightgap condition* [38]. Interpretation methods are directed towards orienting leafs in the congruence graph, so that these can be eventually dropped from the dependency problem using the DP simplifications given in Section 14.5. In total, this configuration thus reflects the body of techniques implemented in TCT that where known prior to this thesis, together with some minor simplifications.

- DGD: With this configuration we indicate the strength of our new DG decomposition processor (Theorem 14.63). Like in the previous configuration, the input problem is transformed into a DP problem. To avoid imprecision of the certificate, TCT tries to solve the DP problem as in the configuration DP+RD. When this approach gets stuck, dependency graph decomposition is applied. The procedure is repeated until the problem is eventually solved.

- DGD-g: TCT behaves in this configuration as above, except that DG decomposition is exhaustively applied directly after the dependency pair transformation.

Experiments Performed on Data Set RaML. In Tables 15.1 and 15.3 we contrast TCT in the above mentioned configurations on the data set RaML. For comparison, we also indicate experimental results obtained by AProVE on this testbed. The entries in Table 15.3 indicate the estimated upper bounds on the runtime complexity. An entry ? indicates that the tool gave up, and an entry ∞ indicates that the tool was aborted due to a timeout. Table 15.1 summarises this data, where numbers annotated to the right of each entry indicate the average execution time, in seconds.

Column Direct in Table 15.1 clearly illustrates the need for transformations. The results drawn in Column RD shows that relative decomposition gives an improvement on the direct approach, still, a majority of the example TRSs can not be handled. Column RD indicates the usefulness of the dependency pair transformations. Two additional problems can be solved in the dependency pair

Answer	Direct	RD	DP+RD	DGD	DGD-g	AProVE
$\mathcal{O}(n)$	1/0.15	1/0.38	2/14.18	3/10.21	2/0.56	2/2.89
$\mathcal{O}(n^2)$	1/0.70	3/6.07	7/7.52	13/17.45	11/4.83	6/11.01
$\mathcal{O}(n^3)$	—	3/44.67	—	3/63.08	4/6.12	1/11.95
$\mathcal{O}(n^4)$	—	—	—	1/159.03	3/40.91	—
$\mathcal{O}(n^5)$	—	—	—	1/149.30	1/78.97	—
Success	2/0.29	7/19.08	9/8.10	21/34.32	21/12.75	8/9.31
Maybe	19/9.35	12/54.85	5/86.77	—	—	—
Timeout	1	3	7	1	1	13

Table 15.1.: Overview experimental evaluation on data set RaML.

setting, and for three problems a more precise certificate could be obtained. Still, a majority of the problems remain open.

Column DGD indicates the gain in power of our tool through DG decomposition. Out of the 22 examples, only one cannot be handled with this approach. A clear drawback of this method however is that DG decomposition might overestimate the complexity, compare Example 14.66. In order to retain a precise estimation on the asymptotic worst case complexity, we are forced to apply this method only as a last means. This results in significantly higher execution times. In comparison, Column DGD-g shows a significant increase in speed, at the cost of precision of the estimated bounding function.

Finally we remark that the RaML-prototype developed by Hofmann et al. [45] beats TCT on the (untransformed) data set, in precision of the estimated bound, and in the speed of the analysis. We attribute this mainly to the naive transformation of RaML programs to TRSs. The RaML-prototype can use for instance domain information given by the semantics of RaML programs. This information is lost during transformation.

Experiments Performed on Data Set RC. Finally, in Tables 15.2 and 15.4 we compare our tool TCT to CaT (full rewriting) and AProVE (innermost rewriting) respectively, on the data set RC. For TCT we indicate results with respect to configuration without DG decomposition (configuration DP+RD) and including DG decomposition (configuration DGD).

The results depicted in Table 15.1 are similar to the ones on the data set RaML. In contrast to innermost rewriting, TCT can only partly benefit from DG decomposition. Observe that weak dependency pairs leave rules from the input problem in the strict component. In contrast, DG decomposition operates on dependency pairs only.

	Full rewriting			Innermost rewriting		
Answer	DP+RD	DGD	CaT	DP+RD	DGD	AProVE
$\mathcal{O}(1)$	1/1.01	1/0.90	—	1/1.16	1/0.09	1/1.06
$\mathcal{O}(n^1)$	8/15.77	7/9.22	6/0.42	10/2.29	8/0.91	8/2.05
$\mathcal{O}(n^2)$	—	2/4.61	—	4/9.33	11/13.02	11/2.53
$\mathcal{O}(n^3)$	—	1/44.64	—	—	3/22.59	3/6.20
$\mathcal{O}(n^4)$	—	1/52.85	—	—	2/77.99	—
$\mathcal{O}(n^5)$	—	—	—	—	2/84.33	—
Success	9/14.13	12/14.35	6/0.42	15/4.09	27/20.11	23/2.78
Maybe	—	—	—	—	—	1/168.07
Timeout	21	18	—	15	3	6

Table 15.2.: Overview experimental evaluation on data set RC.

Input	Direct	RD	DP+RD	DGD	DGD-g	AProVE
appendall	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
bfs	?	?	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	∞
bftmmult	?	?	∞	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$	∞
bitonic	?	?	∞	$\mathcal{O}(n^4)$	$\mathcal{O}(n^4)$	∞
bitvectors	∞	∞	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	∞
clevermmult	?	?	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
duplicates	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
dyade	?	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
eratosthenes	?	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	∞
flatten	?	?	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
insertionsort	?	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
listsort	?	?	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	∞
lcs	?	?	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	∞
martix	?	?	∞	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$	∞
mergesort	?	∞	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	∞
minsort	?	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
queue	?	?	∞	$\mathcal{O}(n^5)$	$\mathcal{O}(n^5)$	∞
quicksort	?	∞	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	∞
rationalpotential	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
splitandsort	?	?	?	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	∞
subtrees	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
tuples	?	?	∞	∞	∞	∞

Table 15.3.: Experimental results on data set RaML.

	Full rewriting			Innermost rewriting		
Answer	DP+RD	DGD	CaT	DP+RD	DGD	AProVE
jones1	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
jones2	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	?	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
jones4	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
jones5	∞	∞	?	∞	∞	∞
jones6	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
flatten	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
reverse	∞	$\mathcal{O}(n^2)$	?	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
shuffle	∞	$\mathcal{O}(n^3)$	?	∞	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
shuffleshuffle	∞	$\mathcal{O}(n^4)$	?	∞	$\mathcal{O}(n^4)$	∞
clique	∞	∞	?	∞	$\mathcal{O}(n^3)$	?
dcquad	∞	∞	?	∞	∞	$\mathcal{O}(n^2)$
egypt	∞	∞	?	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
eratosthenes	∞	∞	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	∞
lcs	∞	?	∞	∞	∞	∞
mmult-nat	∞	∞	?	∞	$\mathcal{O}(n^5)$	$\mathcal{O}(n^2)$
qbf	∞	∞	?	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
sat	∞	∞	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
z86	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
bits	$\mathcal{O}(n^1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
div	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	?	$\mathcal{O}(n^1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
mult	∞	∞	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
mult2	∞	∞	?	∞	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
quad	∞	∞	?	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
square	∞	∞	?	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
bubblesort-nat	∞	∞	?	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
isort	∞	∞	?	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
mergesort	∞	∞	?	∞	$\mathcal{O}(n^4)$	∞
mergesort-nat	∞	∞	?	∞	$\mathcal{O}(n^5)$	∞
quicksort-buggy	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	?	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$	$\mathcal{O}(n^1)$
quicksort	∞	∞	?	∞	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

Table 15.4.: Experimental results on data set RC.

Chapter 16.

Conclusion

This work is concerned with the automated complexity analysis of term rewrite systems. In the first part we have established that the runtime complexity of a rewrite system forms an invariant cost model for rewriting. If a TRS $\mathcal{R}$ computes a function f, this function can be implemented on a Turing machine, and the cost of this implementation is polynomially related to the runtime complexity of $\mathcal{R}$. This result is established by using graph rewriting as an intermediate language. To achieve soundness and completeness of this implementation whilst retaining effectiveness of the implementation, we presented an adequacy theorem that relies only on restricted folding and unfolding operations. For innermost reductions, we have shown that adequacy can be achieved even without unfolding. We think that these adequacy theorems are interesting on its own, as they make precise the necessary folding and unfolding operations required for the implementation of term rewriting.

In the second part we have introduced two restrictions of the recursive path order, the small polynomial path order and the exponential path order. Small polynomial path orders characterise the class of polynomial time computable functions. Further, they also provide a syntactic criteria for the automated runtime complexity analysis of rewrite systems. We have extended upon this order by introducing parameter substitution. This increases the intensionality of the order. Also, this extension allows us to establish a fine grained connection between the functions computed by predicative tail-recursive TRSs, and those computed by register machines.

Our extension of the small polynomial path order, the exponential path order, demonstrates that our approach is general enough to capture computations outside of **FP**. We conjecture that similar variations of small polynomial path orders can lead to new order-theoretic characterisations of various complexity classes. For instance, it seems that keeping the product status underlying sPOP*, but incorporating a nested recursion scheme as in EPO*, can yield an order-theoretic characterisation of the class **FE** of functions computable in time $2^{\mathcal{O}(n)}$.

In the final part we discussed our automated complexity tool TCT, and its underlying combination framework. The framework is general enough to reason about both runtime and derivational complexity, and to formulate a majority of the techniques available for proving polynomial complexity of rewrite systems. On the other hand, it is concrete enough to serve as a basis for a modular complexity analyser. Our tool TCT closely implements the discussed framework. Besides the combination framework we have introduced the notion of $\mathcal{P}$-monotone complexity

pair. This notion unifies the different orders used for complexity analysis. We have also suited small polynomial path orders to complexity problems, allowing the use of argument filterings to weaken monotonicity. Further, we have presented various simplifications employed in TCT and introduced the dependency graph decomposition processor. This processor is easy to implement, and greatly improves modularity. By combining these techniques, our complexity analyser TCT has matured to a state where we can say that it is both versatile and powerful.

Our approach to the automated complexity analysis, through rewriting techniques, can be improved in various ways. The integration of *constrained rewriting* could leverage the design of complexity preserving reductions from *real world* programs to rewrite systems. Using a suitable formalism, one can reflect domain specific information in complexity problems. The automated analysis could greatly profit from such an extension. In the quest for a complexity backend for program analysis, it might prove crucial to extend first-order rewriting to a *higher-order* setting. Last but not least, it seems also beneficial to study lower bounds. Such an analysis could inform a programmer about inefficiencies in code. Until now, this research area is hugely unexplored.

Bibliography

[1] B. .Accattoli and U. D. Lago. On the Invariance of the Unitary Cost Model for Head Reduction. In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications*, volume 15 of *Leibniz International Proceedings in Informatics*, pages 22–37, 2012.

[2] T. Arai and N. Eguchi. A New Function Algebra of EXPTIME Functions by Safe Nested Recursion. *ACM Transactions on Computational Logic*, 10(4), 2009.

[3] T. Arai and G. Moser. A note on a term rewriting characterization of PTIME. In *Proceedings of the 7th Workshop on Termination*, pages 10–13. number AIB-2004-07 of Aachener Informatik-Berichte, 2004. Extended abstract.

[4] T. Arai and G. Moser. Proofs of Termination of Rewrite Systems for Polytime Functions. In *Proceedings of the 15th Conference on the Foundations of Software Technology and Theoretical Computer*, volume 3821 of *Lecture Notes in Computer Science*, pages 529–540, 2005.

[5] T. Arts and J. Giesl. Termination of Term Rewriting using Dependency Pairs. *Theoretical Computer Science*, 236(1–2):133–178, 2000.

[6] M. Avanzini, , and G. Moser. Polynomial Path Orders. *CoRR*, cs/CC/1209.3793, 2012. Submitted to LMCS. Preprint available at `http://www.arxiv.org/abs/1209.3793/`.

[7] 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*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 123–138, 2011.

[8] M. Avanzini, N. Eguchi, and G. Moser. A New Order-theoretic Characterisation of the Polytime Computable Functions. In *Proceedings of the 10th Asian Symposium on Programming Languages and Systems*, volume 7705 of *Lecture Notes in Computer Science*, pages 280–295, 2012.

[9] M. Avanzini, N. Eguchi, and G. Moser. A New Order-theoretic Characterisation of the Polytime Computable Functions. *CoRR*, cs/CC/1201.2553, 2012. Available at `http://www.arxiv.org/abs/1201.2553`.

[10] M. Avanzini, N. Eguchi, and G. Moser. On a Correspondence between Predicative Recursion and Register Machines. In *Proceedings of*

the 12th Workshop on Termination, pages 15–19, 2012. , available at `http://cl-informatik.uibk.ac.at/users/georg/events/wst2012/`.

[11] M. Avanzini and G. Moser. Complexity Analysis by Rewriting. In *Proceedings of the 9th International Symposium on Functional and Logic Programming*, volume 4989 of *Lecture Notes in Computer Science*, pages 130–146, 2008.

[12] 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*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 33–48, 2010.

[13] M. Avanzini and G. Moser. Complexity Analysis by Graph Rewriting. In *Proceedings of the 10th International Symposium on Functional and Logic Programming*, volume 6009 of *Lecture Notes in Computer Science*, pages 257–271. Springer Verlag, 2010.

[14] M. Avanzini and G. Moser. A Combination Framework for Complexity. In *Proceedings of the 24th International Conference on Rewriting Techniques and Applications*. Leibniz International Proceedings in Informatics, 2013. To appear. Technical Report available at `http://arxiv.org/abs/1302.0973`.

[15] M. Avanzini and G. Moser. Tyrolean Complexity Tool: Features and Usage. In *Proceedings of the 24th International Conference on Rewriting Techniques and Applications*. Leibniz International Proceedings in Informatics, 2013. To appear.

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

[17] P. Bahr. Modes of Convergence for Term Graph Rewriting. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, volume 10 of *Leibniz International Proceedings in Informatics*, pages 139–154, 2011.

[18] P. Baillot, J.-Y. Marion, and S. R. D. Rocca. Guest Editorial: Special Issue on Implicit Computational Complexity. *ACM Transactions on Computational Logic*, 10(4), 2009.

[19] H. P. Barendregt, M. v. Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term Graph Rewriting. In *Volume II: Parallel Languages on PARLE: Parallel Architectures and Languages Europe*, volume 259 of *Lecture Notes in Computer Science*, pages 141–158. Springer Verlag, 1987.

[20] A. Beckmann and A. Weiermann. A Term Rewriting Characterization of the Polytime Functions and Related Complexity Classes. *Archive for Mathematical Logic*, 36:11–30, 1996.

[21] S. Bellantoni and S. Cook. A new Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity*, 2(2):97–110, 1992.

[22] P. V. E. Boas. Machine Models and Simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 1–66. The MIT Press, 1990.

[23] G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with Polynomial Interpretation Termination Proof. *Journal on Functional Programming*, 11(1):33–53, 2001.

[24] W. Buchholz. Proof-theoretic Analysis of Termination Proofs. *Annals of Pure and Applied Logic*, 75:57–65, 1995.

[25] E. A. Cichon and A. Weiermann. Term Rewriting Theory for the Primitive Recursive Functions. *Annals of Pure and Applied Logic*, 83(3):199–223, 1997.

[26] 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.

[27] U. Dal Lago and S. Martini. Derivational Complexity is an Invariant Cost Model. In *Proceedings of the 1st International Workshop on Foundational and Practical Aspects of Resource Analysis*, 2009.

[28] U. Dal Lago and S. Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In *Proc. of 36th ICALP*, volume 5556 of *Lecture Notes in Computer Science*, pages 163–174. Springer Verlag, 2009.

[29] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[30] N. Eèn and N. Sõrensson. An Extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 272–286, 2003.

[31] N. Eguchi. A Lexicographic Path Order with Slow Growing Derivation Bounds. *Mathematical Logic Quarterly*, 55(2):212–224, 2009.

[32] J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *Journal of Automated Reasoning*, 40(3):195–220, 2008.

[33] M. F. Ferreira. *Termination of Term Rewriting. Well-foundedness, Totality and Transformations.* PhD thesis, University of Utrecht, Faculty for Computer Science, 1995.

[34] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On Tree Automata that Certify Termination of Left-Linear Term Rewriting Systems. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, number 3467 in Lecture Notes in Computer Science, pages 353–367. Springer Verlag, 2005.

[35] J. Giesl, T. Arts, and E. Ohlebusch. Modular Termination Proofs for Rewriting Using Dependency Pairs. *Journal of Symbolic Computation*, 34:21–58, 2002.

[36] B. Gramlich and F. Schernhammer. Extending context-sensitivity in term rewriting. In *Proceedings 9th International Workshop on Reduction Strategies in Rewriting and Programming*, Electronic Proceedings in Theoretical Computer Science, pages 56–68, 2009.

[37] W. G. Handley and S. S. Wainer. Complexity of Primitive Recursion. In *Computational Logic, NATO ASI Series F: Computer and Systems Science*, volume 165, pages 273–300. Springer Verlag, 1999.

[38] 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*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 364–380, 2008.

[39] 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*, pages 652–666, 2008.

[40] N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. *Information and Computation*, 2012. to appear.

[41] D. Hofbauer. Termination Proofs by Multiset Path Orderings Imply Primitive Recursive Derivation Lengths. *Theoretical Computer Science*, 105(1):129–140, 1992.

[42] D. Hofbauer and C. Lautemann. Termination Proofs and the Length of Derivations. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 167–177. Springer Verlag, 1989.

[43] D. Hofbauer and J. Waldmann. Termination of String Rewriting with Matrix Interpretations. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342. Springer Verlag, 2011.

[44] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate Amortized Resource Analysis. In *Proceedings of the 38th Annual Symposium on Principles of Programming Languages*, pages 357–370. ACM, 2011.

[45] J. Hoffmann, K. Aehlig, and M. Hofmann. Resource Aware ML. In *Proceedings of the 24th International Conference on Computer Aided Verification*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer Verlag, 2012.

[46] N. D. Jones. *Computability and Complexity: From a Programming Perspective*. The MIT Press, 1997.

[47] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path orderings. Unpublished manuscript, 1980.

[48] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. On the Adequacy of Graph Rewriting for Simulating Term Rewriting. *ACM Transactions on Programming Languages and Systems*, 16:493–523, 1994.

[49] 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*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304. Springer Verlag, 2009.

[50] M. S. Krishnamoorthy and P. Narendran. On Recursive Path Ordering. *Theoretical Computer Science*, 40:323–328, 1985.

[51] D. Lankford. On Proving Term Rewriting Systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, 1979.

[52] D. Leivant. A Foundational Delineation of Computational Feasiblity. In *Proceedings of the 6th ACM/IEEE Symposium on Logic in Computer Science*, pages 2–11. IEEE Computer Society, 1991.

[53] S. Lucas. Fundamentals of Context-Sensitive Rewriting. In *Proceedings of the 22nd Seminar on Current Trends in Theory and Practice of Informatics*, Lecture Notes in Computer Science, pages 405 – 412. Springer Verlag, 1995.

[54] J.-Y. Marion. Analysing the Implicit Complexity of Programs. *Information and Computation*, 183:2–18, 2003.

[55] J.-Y. Marion. On Tiered Small Jump Operators. *Logical Methods in Computer Science*, 5(1), 2009.

[56] 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*, volume 6742 of *Lecture Notes in Computer Science*, pages 1–20. Springer Verlag, 2011.

[57] R. Milner, M. Tofte, and D. Macqueen. *The Definition of Standard ML*. The MIT Press, Cambridge, MA, USA, 1997.

[58] G. Moser. Derivational Complexity of Knuth-Bendix Orders Revisited. In *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 75–89. Springer Verlag, 2006.

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

[60] 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, volume 5595 of *Lecture Notes in Computer Science*, pages 276–290. Springer Verlag, 2009.

[61] G. Moser, A. Schnabl, and J. Waldmann. Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations. In *Proceedings of the 28th Conference on the Foundations of Software Technology and Theoretical Computer*, Leibniz International Proceedings in Informatics, pages 304–315, 2008.

[62] F. Neurauter. *Termination Analysis of Term Rewriting by Polynomial Interpretations and Matrix Interpretations.* PhD thesis, University of Innsbruck, 2012.

[63] L. Noschinski, F. Emmes, and J. Giesl. A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems. In *Proc. of 23rd CADE*, Lecture Notes in Artificial Intelligence, pages 422–438. Springer Verlag, 2011.

[64] E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer Verlag, 2001.

[65] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley Longman, second edition, 1995.

[66] D. Plump. Essentials of Term Graph Rewriting. *Electronic Notes in Theoretical Computer Science*, 51:277–289, 2001.

[67] J. A. Robinson and A. Voronkov. *Handbook of Automated Reasoning (in 2 volumes).* Elsevier and MIT Press, 2001.

[68] H. Rogers. *Theory of Recursive Functions and Effective Computability.* The MIT Press, 1987.

[69] S. L. Peyton Jones. *The Implementation of Functional Languages.* Prentice-Hall International, 1987.

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

[71] J. C. Shepherdson and H. E. Sturgis. Computability of Recursive Functions. *Journal of the Association for Computing Machinery*, 10:217–255, 1963.

[72] J. Staples. Computation on Graph-like Expressions. *Theoretical Computer Science*, 10:297–316, 1980.

[73] J. Steinbach. Extensions and Comparsoin of Simplification Orderings. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 434–448, 1989.

[74] J. Steinbach and U. Kühler. Check your Ordering - Termination Proofs and Open Problems. Technical Report SEKI-Report SR-90-25, University of Kaiserslautern, 1990.

[75] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracks in Theoretical Computer Science*. Cambridge University Press, 2003.

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

[77] J. Waldmann. Polynomially Bounded Matrix Interpretations. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 357–372, 2010.

[78] 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.

[79] H. Zankl and M. Korp. Modular Complexity Analysis via Relative Complexity. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 385–400, 2010.

[80] H. Zantema. Termination of Context-Sensitive Rewriting. In *Proceedings of the 8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, pages 172–186. Springer Verlag, 1997.

List of Notations and Abbreviations

Index