The Exact Hardness of Deciding Derivational and Runtime Complexity*

Andreas Schnabl[1] and Jakob Grue Simonsen[2]

1 **Institute of Computer Science, University of Innsbruck**
Technikerstr. 21a, A-6020 Innsbruck, Austria
andreas.schnabl@uibk.ac.at

2 **Department of Computer Science, University of Copenhagen (DIKU)**
Njalsgade 126-128, DK-2300 Copenhagen S, Denmark
simonsen@diku.dk

Abstract

For any class C of computable total functions satisfying some mild conditions, we prove that the following decision problems are complete for level Σ_2^0 of the arithmetical hierarchy: (A) Deciding whether a term rewriting system (TRS for short) has runtime complexity bounded by a function in C. (B) Deciding whether a TRS has derivational complexity bounded by a function in C.

In particular, the problems of deciding whether a TRS has polynomially (exponentially) bounded runtime complexity (respectively derivational complexity) are Σ_2^0-complete. This places deciding polynomial derivational or runtime complexity of TRSs at the same level in the arithmetical hierarchy as deciding nontermination or nonconfluence of TRSs. We proceed to show that the related problem of deciding for a single computable function f whether a TRS has runtime complexity bounded from above by f is Π_1^0-complete. We further prove that analysing the implicit complexity of TRSs is even more difficult: The problem of deciding whether a TRS accepts a language of terms accepted by some TRS with runtime complexity bounded by a function in C is Σ_3^0-complete.

All of our results are easily extended to the notion of minimal complexity (where the length of shortest reductions to normal form is considered) and remain valid under any computable reduction strategy. Finally, all results hold both for unrestricted TRSs and for the class of orthogonal TRSs.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes; F.2.2; F.4.1; F.4.2

Keywords and phrases Term rewriting, derivational complexity, arithmetical hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2011.481

1 Introduction

Term rewriting is a simple model of non-deterministic computation that underlies much of declarative and functional programming. Term rewriting systems (TRSs for short) are finite sets of oriented equations (*rewriting rules*) on first-order terms–e.g. $c(c(x,y),z) \to c(x,c(y,z))$ for the associative law from algebra. The application of such a rule somewhere in a term–e.g. $c(2,c(c(3,3),1)) \to_{\mathcal{R}} c(2,c(3,c(3,1)))$–is the elementary step of computation in term rewriting.

In the last few years, *complexity analysis* has emerged as an important research field within the term rewriting community. Several measures of complexity have been considered

* This work was partially supported by a grant of the University of Innsbruck.

Computer Science Logic 2011 (CSL'11).
Editor: Marc Bezem; pp. 481–495

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for term rewriting. The conceptually simplest one was suggested by Hofbauer and Lautemann in [18]: The *derivational complexity function* with respect to a terminating TRS $\mathcal{R}$ relates the worst-case number of rewriting steps in a computation to the size of the initial term. Other such measures include *runtime complexity* [16], which is a restriction of derivational complexity only considering initial terms corresponding to function calls, and the complexity of the function *computed* by a TRS [8], its *implicit complexity*. While all of these measures consider a kind of worst case complexity, some average case analysis has been done, as well [7]. The focus has been on measures that take the worst case number of rewriting steps in a reduction as their metric. The most common way to establish upper bounds on these has been to infer them from a termination proof of the given TRS, see [17, 22, 30, 29] for some examples of this approach. A possible modification of this approach is to restrict existing termination proof techniques in order to obtain smaller complexity bounds; instances of this idea can be found in [19, 2, 16].

The problem of deciding, for some TRS $\mathcal{R}$, whether its derivational or runtime complexity is bounded by a single function, or a family of functions, is–unsurprisingly–undecidable. In the work reported in this paper, we follow recent investigations into the exact level of undecidability (in the arithmetical hierarchy) of questions in rewriting [24, 9], according to which many of the standard properties of rewriting (termination, normalisation, confluence) are known to be Π^0_2-complete. We also follow a much older set of investigations into the exact level of undecidability of intensional properties of programs [12, 23, 20, 1]. Compared to other models of computation such as Turing Machines, term rewriting operates in a quite nonstandard way, and it is a priori not clear that the classic results can be transferred to term rewriting. For instance, for nonlinear rewriting rules (where a variable may occur more than once in either the left- or the right-hand side) such as $f(x, x, y) \to g(x, y, y)$, it is assumed that both the equality check implicit in determining whether the rule can be applied (e.g. the first two arguments of f must be identical terms), and the copying of arbitrarily large terms (e.g. the term substituted for y can be large) can be done within a single computation step. Even more pertinent, the set of allowed "starting configurations" in derivational and runtime complexity analysis is defined much more liberally than in other models of computation. For Turing machines, only (a certain subset of all) well-formed configurations are considered, and in pure functional programs, the arguments of a function are always well-formed elements of a data type, e.g. $f(s(s(0)))$ — "f applied to 2". In contrast, derivational complexity, for example, must also consider "junk" terms that do not correspond to well-formed starting configurations such as $f(s(f(f(s(0)))))$. We verify that despite these obstacles, the classical results hold for TRSs.

Our results show that the decision problems for complexity functions such as the above are either Π^0_1-complete (for a specific function as an upper bound) or Σ^0_2-complete (for existence of an upper bound in a family of functions satisfying some mild conditions). This is in line with classical results on the degrees of undecidability of intensional complexity of programs. Our results run counter to the intuition given by the traditional approach to complexity analysis of rewriting: Upper bounds on the derivational complexity of a TRS are established by extraction from a proof of termination of the TRS. However, the exact degree of undecidability for deciding whether the derivational complexity of a TRS is (for instance) polynomially bounded is the same as for deciding whether it is *non*-terminating.

All results easily carry over to a different flavour of complexity from formal language theory, confusingly also called derivational complexity, but which is starkly different from the homonymous notion in term rewriting. In formal language theory, the derivational complexity relates an integer n to the maximum length of *shortest* derivations of sentences of length at

most n [6, 27].

For implicit complexity of TRSs, where the computational complexity of the mathematical function *computed* by a given TRS $\mathcal{R}$ is considered [5, 19, 21], the pertinent decision problems are even harder: Deciding whether the implicit complexity of $\mathcal{R}$ is, for instance, polynomial or exponential is even harder than the previously mentioned problems: Σ^0_3-complete. Again, this is in line with the classical results [12, 23]. However, in practice, the additional existential quantifier (quantifying over the possibly more efficient TRS) might make it easier to establish *upper* bounds on the implicit complexity of a TRS than to establish upper bounds on its derivational or runtime complexity as there is an additional degree of freedom in constructing the proof of the upper bound.

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

2 Preliminaries

We presuppose basic familiarity with computability theory [25, 10] and term rewriting [4, 28], but recall central definitions and notions of rewriting and computability below. Let $\mathcal{V}$ be a countably infinite set of variables, and $\mathcal{F}$ a finite signature of function symbols with fixed arities containing at least one symbol of arity 0. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted as $\mathcal{T}(\mathcal{F},\mathcal{V})$. The set of ground terms over $\mathcal{F}$ is denoted as $\mathcal{T}(\mathcal{F})$. The *root symbol* (denoted as $\mathsf{rt}(t)$) of a term t is either t itself, if $t \in \mathcal{V}$, or the symbol f, if $t = f(t_1,\ldots,t_n)$. The *size* (denoted as $|t|$) of a term t is the total number of occurrences of variables and function symbols in t. A *substitution* is a mapping $\sigma\colon \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V})$, where the number of variables x such that $\sigma(x) \neq x$ is finite. We usually write $t\sigma$ instead of $\sigma(t)$ to denote the application of σ to a term t. We introduce a fresh constant $\bullet$ (the *hole*) and define a *context* C as a term (over $\mathcal{F} \cup \{\bullet\}$ and $\mathcal{V}$) containing exactly one occurrence of $\bullet$. For a term t and a context C, $C[t]$ denotes the replacement of $\bullet$ by t in C.

Let $\mathcal{R}$ be a *finite* TRS over $\mathcal{F}$ and $\mathcal{V}$, i.e. a finite set of *rewriting rules* $l \to r$ with $l, r \in \mathcal{T}(\mathcal{F},\mathcal{V})$ such that l is not a variable, and each variable in r also occurs in l. We only consider finite TRSs in this paper. A TRS $\mathcal{R}$ induces a *rewrite relation* $\to_\mathcal{R}$ as follows: $s \to_\mathcal{R} t$ if there exist a rule $l \to r$ in $\mathcal{R}$, a context C, and a substitution σ such that $s = C[l\sigma]$ and $t = C[r\sigma]$. A term s is a *normal form* of $\to_\mathcal{R}$ if there exists no term t such that $s \to_\mathcal{R} t$. A rule $l \to r$ is *left-linear* if l does not contain multiple occurrences of the same variable. A TRS is *orthogonal* if it contains no critical pairs, and all of its rules are left-linear.

The n-fold composition of $\to_\mathcal{R}$ is denoted as $\to^n_\mathcal{R}$, and we write $\to^*_\mathcal{R}$ for the reflexive and transitive closure of $\to_\mathcal{R}$. We write $s \to^!_\mathcal{R} t$ if $s \to^*_\mathcal{R} t$ and t is a normal form of $\to_\mathcal{R}$. We write $s \to^{n,!}_\mathcal{R} t$ if $s \to^n_\mathcal{R} t$ and t is a normal form. A TRS $\mathcal{R}$ is *confluent* if for all terms s,t,u such that $u \to^*_\mathcal{R} s$ and $u \to^*_\mathcal{R} t$, there exists a term v such that $s \to^*_\mathcal{R} v$ and $t \to^*_\mathcal{R} v$. It is well-known that orthogonality of a TRS implies its confluence. The *derivation tree* of a term s wrt. $\mathcal{R}$ is the following directed graph: the nodes are all terms t such that $s \to^*_\mathcal{R} t$, and there exists an edge from t to t' iff $t \to_\mathcal{R} t'$. We say that a derivation tree is *non-circular* if no path starting from the root in the tree contains the same term more than once. Observe that if s is $\mathcal{R}$-terminating, then the derivation tree of s wrt. $\mathcal{R}$ is finite and non-circular, and s is its single source node. The *derivation height* of a term s with respect to a finitely branching, terminating binary relation $\to$ is given by $\mathsf{dh}(s,\to) = \max\{n : \exists t.s \to^n t\}$, and the *derivational complexity* function of a TRS $\mathcal{R}$ is defined as $\mathsf{dc}_\mathcal{R}(n) = \max\{\mathsf{dh}(s,\to_\mathcal{R}) : |s| \leq n\}$.

A function symbol f is a *defined symbol* of $\mathcal{R}$ if there exists a rewrite rule $l \to r$ such that $\mathsf{rt}(l) = f$; otherwise, it is a *constructor* of $\mathcal{R}$. We denote the set of defined symbols of $\mathcal{R}$ as $\mathcal{D}$, while the constructors of $\mathcal{R}$ are collected in $\mathcal{C}$. A TRS $\mathcal{R}$ is a *constructor TRS* if every rule in $\mathcal{R}$ has the shape $f(l_1, \ldots, l_n) \to r$ for $f \in \mathcal{D}$ and $l_1, \ldots, l_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. The set of basic terms is $\mathcal{B} = \{f(v_1, \ldots, v_n) : f \in \mathcal{D} \wedge v_1, \ldots, v_n \in \mathcal{T}(\mathcal{C})\}$. The *runtime complexity* function of a TRS $\mathcal{R}$ is $\mathsf{rc}_{\mathcal{R}}(n) = \max\{\mathsf{dh}(s, \to_{\mathcal{R}}) : |s| \leq n \wedge s \in \mathcal{B}\}$.

We briefly recapitulate the arithmetical hierarchy. Let $n \in \mathbb{N}$. A set $A \subseteq \mathbb{N}$ is in Σ^0_n (respectively in Π^0_n) if there is an $(n+1)$-ary decidable predicate[1] $P(x_1, \ldots, x_n, x_{n+1})$ such that A is exactly the subset of $\mathbb{N}$ for which the unary predicate $\exists x_1. \forall x_2. \cdots Q x_n. P(x_1, \ldots, x_n, x_{n+1})$ (respectively $\forall x_1. \exists x_2. \cdots Q x_n. P(x_1, \ldots, x_n, x_{n+1})$) obtains, where Q is either $\exists$ or $\forall$ depending on whether n is odd or even (respectively even or odd). We say that a set $A \subseteq \mathbb{N}$ is Σ^0_n*-hard* (respectively Π^0_n-hard) if for every set B in Σ^0_n (respectively, for every set B in Π^0_n), there exists a computable function f such that $x \in B$ iff $f(x) \in A$ for all $x \in \mathbb{N}$. A set $A \subseteq \mathbb{N}$ is Σ^0_n*-complete* (Π^0_n-complete) if it is both contained in Σ^0_n (in Π^0_n) and Σ^0_n-hard (Π^0_n-hard).

For a function f and a set $\mathcal{G}$ of functions $\mathbb{N} \to \mathbb{N}$, we say that f *is globally bounded by a function in* $\mathcal{G}$ ($f \leq \mathcal{G}$) if there exists a function $g \in \mathcal{G}$ such that for all $n \in \mathbb{N}$, we have $f(n) \leq g(n)$. The set $\mathcal{G}$ is *closed under polynomial slowdown* if, for any $g \in \mathcal{G}$ and any polynomial P over $\mathbb{N}$, we have $f \leq \mathcal{G}$ for $f(x) = P(g(x))$. In particular, the set of all polynomials over $\mathbb{N}$ is closed under polynomial slowdown. We define the set of functions $\Xi(\mathcal{G})$ as $\bigcup_{g \in \mathcal{G}, a \in \mathbb{N}, b \in \mathbb{N}} g(a \cdot n + b)$.

Observe that if $\mathcal{G}$ is the set of polynomials with non-negative integer coefficients, then $\mathcal{G} = \Xi(\mathcal{G})$, $\Xi(\mathcal{G})$ is closed under polynomial slowdown, and $f \leq \Xi(\mathcal{G})$ iff f is bounded by a polynomial. On the other hand, if $\mathcal{G} = \bigcup_{a \in \mathbb{N}} g_a$ with $g_a(x) = a^x$, then $\Xi(\mathcal{G})$ is again closed under polynomial slowdown, and $f \leq \Xi(\mathcal{G})$ iff f is bounded by an exponential function; in order to see that $\Xi(\mathcal{G})$ is closed under polynomial slowdown, let $g_c \in \mathcal{G}$, and let P be a polynomial of degree d with coefficients at most a; then $P(g_c(n)) \leq g_{c+2}(d \cdot n + (d+1) \cdot a)$.

3 Turing Machines as Rewriting Systems

We assume that the reader is familiar with the standard definitions of Turing Machines [25]. The following is the specific formalisation of Turing Machines used throughout this paper.

▶ **Definition 1.** A *(deterministic single-tape) Turing Machine* is a triple (Q, Σ, δ), where

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

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

▶ **Definition 2.** A *configuration* of a single-tape Turing Machine is a triple (q, w, i), where

- $q \in Q$ denotes the current state.

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

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

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

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

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

Given a (single-tape or dual-tape) Turing Machine M, the notions "M moves from configuration α to configuration β in one step", "M runs for n steps on input word x", "M halts on configuration α", "M halts on input word x", "M accepts/rejects input word x", and "M accepts (exactly) language L" are defined in the usual way. We leave it to the reader to fill out the formal details.

Finally, given a (single-tape or dual-tape) Turing Machine M, we define the functions $\textsc{Time}_M$ and $\textsc{LifeTime}_M : \mathbb{N} \to \mathbb{N}$ as follows; we chose the name $\textsc{LifeTime}_M$ to reflect the close relationship to the Turing Machine *mortality problem*, which asks whether a given Turing Machine halts on all configurations:

$$\textsc{Time}_M(n) = \max\{m : M \text{ runs } m \text{ steps on input word } x \wedge |x| \leq n\}$$
$$\textsc{LifeTime}_M(n) = \max\{m : M \text{ runs } m \text{ steps on configuration } \alpha \wedge |\alpha| \leq n\}$$

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

We now encode dual-tape Turing Machines M as TRSs $\Delta(M)$, essentially using the encoding of [28, Chapter 5], lifted to dual-tape machines. We build up the signature $\mathcal{F}$ of $\Delta(M)$ from a set of defined symbols $\mathcal{D}$ and constructor symbols $\mathcal{C}$ as follows. For each symbol $b \in \Sigma$, the set $\mathcal{C}$ contains b as a unary function symbol, as well as a nullary symbol $\triangleright$. For each state $q \in Q$, $\mathcal{D}$ contains q as a function symbol of arity 5. Additionally, $\mathcal{D}$ contains the unary symbols ok and runM. Words over Σ are translated to terms by $\phi(\epsilon) = \triangleright$, and $\phi(bw) = b(\phi(w))$, where $b \in \Sigma$ and $w \in \Sigma^*$. A configuration (q, w, i, v, j) such that $w = w_1 \dots w_n$ and $v = v_1 \dots v_m$ is encoded as the term $q(\mathsf{ok}(\triangleright), \phi(w_{i-1} \dots w_1), \phi(w_i \dots w_n), \phi(v_{j-1} \dots v_1), \phi(v_j \dots v_m))$. This way, it is easy to simulate Turing Machine computation steps by rewriting steps. For ease of notation, we often identify w and $\phi(w)$ for $w \in \Sigma^*$ during the rest of this paper. The purpose of demanding the subterm $\mathsf{ok}(\triangleright)$ to be present in the encoding of a configuration is to ensure that configurations (in particular, unreachable configurations) can not be encoded by basic terms. Therefore, in contrast to the construction in [28, Chapter 5], $\Delta(M)$ is not a constructor TRS.

▶ **Definition 3.** Let $M = (Q, \Sigma, \delta)$ be a dual-tape Turing Machine. Then the orthogonal TRS $\Delta(M)$ is defined by the rules shown in Figure 1.

Here, the rules for the transition function of the given Turing Machine are the natural lifting of the rules given in [28, Chapter 5] to our encoding of configurations. The first of the

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

Figure 1 The TRS $\Delta(M)$ defined by a dual-tape Turing Machine M

four additional rules is responsible for rewriting a basic term of the shape $\mathsf{runM}(w)$ into the encoding of a start configuration. The other three additional rules ensure that q_a, q_r, and ok are defined symbols without violating the orthogonality of the TRS. This also implies that each term in $\mathcal{T}(\mathcal{F})$ is a word over Σ.

▶ **Lemma 4.** *Let M be a Turing Machine. Then $\mathsf{rc}_{\Delta(M)}(n) = 0$ for all $n < 2$, and $\mathsf{rc}_{\Delta(M)}(n) = \textsc{Time}_M(n-2) + 2$ for all $n \geq 2$.*

Proof. First, observe that the only basic terms t which are not normal forms with respect to $\to_{\Delta(M)}$ have runM as their root symbol. Hence, we assume $\mathsf{rt}(t) = \mathsf{runM}$. Moreover, the single argument of runM must be a word over Σ. Then the only one-step reduct of t is the encoding of a starting configuration of M. Moreover, clearly $|t| \geq 2$. A straightforward argument (compare [28, Exercise 5.3.3]) reveals the following:

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

From these three observations, the lemma follows immediately. ◀

4 Hardness of Runtime Complexity Analysis

We now consider the hardness of establishing upper bounds on the runtime complexity of TRSs. Thanks to Lemma 4, it is possible to use existing results about the time complexity of Turing Machines [12] directly. Note that it is necessary to use dual-tape machines in order to obtain the following results exactly as formulated. If we used single-tape machines instead,

there would be an additional quadratic slowdown in the proofs for both of the results from [12] we use. Allowing multi-tape Turing Machines instead of dual-tape machines would yield an additional speedup in the order of $n \log n$ [14]; however, this additional speedup is not needed for obtaining the results below.

▶ **Lemma 5.** *Let $\mathcal{G} = \{g_1, g_2, \ldots\}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then there is a computable, strictly increasing, and total function $f\colon \mathbb{N} \to \mathbb{N}$ such that $f \not\leq \Xi(\mathcal{G})$.*

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

▶ **Proposition 6.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then the following decision problem is Σ^0_2-hard:*
Instance: *A dual-tape Turing Machine M.*
Question: *Is $\textsc{Time}_M \leq \Xi(\mathcal{G})$?*

Proof. By [12, Theorem 2], the proposition holds for the special case that $\mathcal{G}$ is the set of polynomials. Inspection of the proof of [12, Theorem 2] yields that only the following two properties of $\mathcal{G}$ are used: $\Xi(\mathcal{G})$ must contain the function $n+1$, and there must exist a computable total function f such that $f \not\leq \Xi(\mathcal{G})$. The first property follows from the assumption that $\mathcal{G}$ must consist of strictly increasing total functions, and the second property follows from Lemma 5. ◀

▶ **Theorem 7.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$. Then the following decision problem is Σ^0_2-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Is $\mathsf{rc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$?*
If the problem instances are restricted to orthogonal TRSs, Σ^0_2-completeness holds, as well.

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

We now show Σ^0_2-hardness of the problem. By Proposition 6, it is Σ^0_2-hard to decide whether $\textsc{Time}_M \leq \Xi(\mathcal{G})$, given a dual-tape Turing Machine M. From Lemma 4, it follows that $\mathsf{rc}_{\Delta(M)} \leq \Xi(\mathcal{G})$ iff $\textsc{Time}_M \leq \Xi(\mathcal{G})$. The transformation Δ is obviously computable, and $\Delta(M)$ is orthogonal. Therefore, it is Σ^0_2-hard to decide whether $\mathsf{rc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal). ◀

▶ **Proposition 8.** *Let f be a computable and total function $\mathbb{N} \to \mathbb{N}$ such that $f(n) > n$ for all $n \in \mathbb{N}$. Then the following decision problem is Π^0_1-hard:*
Instance: *A dual-tape Turing Machine M.*
Question: *Is $\textsc{Time}_M(n) \leq f(n)$ for all $n \in \mathbb{N}$?*

Proof. Straightforward generalisation of [12, Theorem 1]. ◀

▶ **Theorem 9.** *Let f be a computable and total function $\mathbb{N} \to \mathbb{N}$ such that $f(n) > n$ for all $n \in \mathbb{N}$. Then the following decision problem is Π^0_1-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Is $\mathsf{rc}_\mathcal{R}(n) \leq f(n)$ for all $n \in \mathbb{N}$?*
If the problem instances are restricted to orthogonal TRSs, Π^0_1-completeness holds, as well.

Proof. To see that the problem is contained in Π^0_1, consider the binary predicate $P(x_1, x_2)$ on $\mathbb{N}$ that obtains iff $\mathsf{rc}_\mathcal{R}(x_1) \leq f(x_1)$ where $\mathcal{R}$ is the TRS encoded by the integer x_2. As f is computable and total, and as the derivation tree of each of the finite number of terms of size at most x_1 can be computed up to depth $f(x_1)$, the predicate is obviously decidable. Hence, the answer to the question to be decided is "yes" iff the predicate $\forall x_1.P(x_1, x_2)$ obtains, and containment in Π^0_1 is shown.

We now show Π^0_1-hardness of the problem. Let $f'(n) = f(n+2) - 2$, and note that $f'(n) > n$. By Proposition 8, it is Π^0_1-hard to decide whether $\textsc{Time}_M(n) \leq f'(n)$ for all $n \in \mathbb{N}$, given a dual-tape Turing Machine M. By Lemma 4, we have $\mathsf{rc}_{\Delta(M)}(n) \leq f(n)$ for all $n \in \mathbb{N}$ iff $\textsc{Time}_M(n) \leq f'(n)$ for all $n \in \mathbb{N}$. The transformation Δ is obviously computable, and $\Delta(M)$ is orthogonal. Therefore, it is Π^0_1-hard to decide whether $\mathsf{rc}_\mathcal{R}(n) \leq f(n)$ for all $n \in \mathbb{N}$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal). ◀

5 Implicit Computational Complexity Analysis for Rewriting

In this section we establish Σ^0_3-completeness of deciding implicit complexity bounds on TRSs: Deciding whether the computation carried out by a TRS can be done within a certain time bound, possibly by another, more efficient TRS. In the literature, there exist similar results about Turing Machines [12, 23]. In order to be able to apply them, we need to establish a link between computations carried out by TRSs and Turing Machines. For one direction of this link, Lemma 4 suffices. The existence of the other direction of the link has recently been shown by Avanzini and Moser [3]. In the following, we define a simple notion of computation by a TRS, and glue the above components together.

▶ **Definition 10.** Let $\mathcal{R}$ be a TRS with signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, let f be a specific n-ary function symbol in $\mathcal{D}$ (we call f the *main function* of $\mathcal{R}$), and a another specific symbol in the signature of $\mathcal{F}$ (we call a the *accepting symbol* of $\mathcal{R}$). Then for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$ we say that $\mathcal{R}$ *accepts* $(t_1, \ldots, t_n)$ if $f(t_1, \ldots, t_n) \to^!_\mathcal{R} t$ such that $\mathsf{rt}(t) = a$. The *language accepted by* $\mathcal{R}$ is the set $\mathcal{L}(\mathcal{R}) = \{(t_1, \ldots, t_n) : t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}) \wedge \mathcal{R} \text{ accepts } (t_1, \ldots, t_n)\}$.

▶ **Definition 11.** Let $\mathcal{R}$ be a TRS with main function f of arity n, accepting symbol a, and signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, let $L \subseteq \mathcal{T}(\mathcal{C})^n$, and let $\mathcal{G}$ be a set of computable, strictly increasing, and total functions. We say that $\mathcal{R}$ *(deterministically) accepts L in time $\Xi(\mathcal{G})$* if $\mathcal{L}(\mathcal{R}) = L$, $\mathcal{R}$ is confluent, and $\mathsf{rc}_\mathcal{R} \leq \Xi(\mathcal{G})$.

As shown by the next lemma, Δ indeed relates the notions of acceptance for Turing Machines and TRSs in the natural way. It follows by the same arguments as Lemma 4.

▶ **Lemma 12.** *Let M be a dual-tape Turing Machine with tape alphabet Σ and accepting state q_a. For each word $x \in \Sigma^*$, M accepts x iff $\Delta(M)$ with main function runM and accepting symbol q_a accepts $\phi(x)$. Moreover, $\mathcal{L}(\Delta(M))$ is exactly the language accepted by M.*

▶ **Proposition 13.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions. Then the following decision problem is Σ^0_3-hard:*

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

Proof. By [23, Corollary 3], for each set C of decidable languages containing an infinite language A and all languages B such that $A \setminus B$ is finite, the following problem is Σ^0_3-hard:
Instance: A (dual-tape) Turing Machine M.
Question: Is the language accepted by M contained in C?
Fix C to be the set of all languages L decided by any (dual-tape) Turing Machine M' with $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$. As $\mathcal{G}$ contains a strictly increasing function, C contains an infinite language A and all languages B such that $A \setminus B$ is finite. For instance, Σ^*, where Σ is the tape alphabet of M, is a suitable instance of A here. Thus C satisfies the assumptions of [23, Corollary 3], and the proposition follows. ◀

Now we have all necessary ingredients to show the main theorem of this section. Proposition 13 yields the corresponding result for Turing Machines, while Lemma 12 and [3] form the bridge to term rewriting.

▶ **Theorem 14.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions such that $\Xi(\mathcal{G})$ is closed under polynomial slowdown. Then the following decision problem is Σ^0_3-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Does there exist a TRS which accepts $\mathcal{L}(\mathcal{R})$ in time $\Xi(\mathcal{G})$?*
If the problem instances are restricted to orthogonal TRSs, Σ^0_3-completeness holds, as well.

Proof. First we show that the problem is contained in Σ^0_3. Let $P(x_1, x_2, x_3, x_4)$ be the predicate on $\mathbb{N}$ that obtains exactly if the i^{th} function g_i in $\mathcal{G}$, the TRS $\mathcal{S}$ encoded by l, and the TRS $\mathcal{R}$ encoded by x_4 satisfy the following properties:

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

Observe that $P(x_1, x_2, x_3, x_4)$ is a decidable predicate: As $\mathcal{G}$ is recursively enumerable and consists of computable functions, we may compute $g_i(j \cdot x_2 + k)$; as the signature and set of rules of $\mathcal{R}$ (respectively $\mathcal{S}$) are both finite, we may compute the finite set of basic terms over $\mathcal{F}_{\mathcal{R}}$ (respectively $\mathcal{F}_{\mathcal{S}}$) of size at most x_2, and for each of these compute their derivation trees up to depth x_3 (respectively $g_i(j \cdot x_2 + k)$) and subsequently check whether the leaves of each tree consist only of normal forms, and whether all trees are non-circular. If that is the case, then the set of normal forms of the considered terms is finite, as well, and hence it is computable whether $f(t_1, \ldots, t_n) \to^!_{\mathcal{R}} u_1$ and $f(t_1, \ldots, t_n) \to^!_{\mathcal{S}} u_2$ for all relevant $t_1, \ldots, t_n, u_1, u_2$. As the answer to the question to be decided is "yes" for the TRS encoded by x_4 iff the predicate $\exists x_1.\forall x_2.\exists x_3.P(x_1, x_2, x_3, x_4)$ obtains, containment in Σ^0_3 is proved.

We now show Σ^0_3-hardness of the problem. By Proposition 13, it is Σ^0_3-hard to decide whether there exists a dual-tape Turing Machine M' accepting the same language as M such that $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$, given a dual-tape Turing Machine M. Let q_a be the accepting state of M. We set runM to be the main function, and q_a the accepting symbol of $\Delta(M)$. Note that $\Delta(M)$ is orthogonal, so the reduction described here works regardless of whether

the problem instance is restricted to be orthogonal. By Lemma 12, $L = \mathcal{L}(\Delta(M))$ is the language accepted by M. It remains to show that there exists a dual-tape Turing machine M' accepting L with $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$ iff there exists a TRS $\mathcal{R}'$ accepting L in time $\Xi(\mathcal{G})$.

In order to show the direction from left to right, suppose that there exists a dual-tape Turing machine M' accepting L with $\textsc{Time}_{M'} \leq \Xi(\mathcal{G})$. Then by employing Lemma 12 again, we also have $\mathcal{L}(\Delta(M')) = L$ if we set the main function to runM again, and the accepting symbol of $\Delta(M')$ to the accepting state of M'. Thus, $\Delta(M')$ (deterministically) accepts L in time $\Xi(\mathcal{G})$.

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

6 Hardness of Derivational Complexity Analysis

We proceed to give the completeness result for establishing upper bounds on the derivational complexity of TRSs. Unfortunately, we cannot lift the results of Section 4 directly from runtime complexity to derivational complexity. The definition of the derivational complexity of a TRS places no restrictions on the considered starting term; in particular, we have to consider encodings of unreachable configurations in the underlying Turing Machine. The crucial ingredient of the main theorem in this section is an investigation by Herman [15] of the mortality problem for Turing Machines. Herman's proof gives a concrete reduction of the mortality problem from the halting problem that involves only a polynomial overhead in time complexity. In order to use this reduction, we switch from dual-tape to single-tape Turing Machines for this section.

▶ **Proposition 15** ([13, Theorem 6]). *Let M be a dual-tape Turing Machine. Then there exists a single-tape Turing Machine M' such that M' accepts and rejects exactly the same input words as M, and $\textsc{Time}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^2, n^2\})$.*

▶ **Lemma 16.** *Let M be a single-tape Turing Machine with tape alphabet Σ. Then there exists a single-tape Turing Machine M' such that M' accepts and rejects exactly the same input words from Σ^* as M, M' halts on all configurations iff M halts on all input words, and $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^3, n^3\})$.*

Proof. By [15, Theorem 1], there exists a single-tape Turing Machine M' which accepts the same input words from Σ^* as M, and halts on all configurations iff M halts on all input words. The proof that $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^3, n^3\})$ is deferred to the extended version of this paper [26]. ◀

We now encode single-tape Turing Machines M as TRSs $\Delta_1(M)$. As in Section 3, we use the encoding of [28, Chapter 5]: a configuration (q, w, i) such that $w = w_1 \dots w_n$ is encoded as the term $q(\phi(w_{i-1} \dots w_1), \phi(w_i \dots w_n))$. However, we slightly change the rules of Δ_1 to reflect that we consider machines with only one-way infinite tapes for simplification purposes. Note that Δ_1 does not contain any mechanism to enforce any restriction on the starting term of a derivation; this is because we are considering derivational complexity (rather than runtime complexity) in this section.

▶ **Definition 17.** Let $M = (Q, \Sigma, \delta)$ be a single-tape Turing Machine. Then the orthogonal constructor TRS $\Delta_1(M)$ is defined by the rules shown in Figure 2.

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

Figure 2 The TRS $\Delta_1(M)$ defined by a single-tape Turing Machine M

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

▶ **Lemma 18.** *Let M be a single-tape Turing Machine. Then we have $\mathsf{dc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\Omega(n))$ and $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$.*

Proof. The following holds by straightforward arguments (compare [28, Exercise 5.3.3]):

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

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

It remains to show that $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$. It easily follows from the above observations that $\mathsf{rc}_{\Delta_1(M)}(n) \in \textsc{LifeTime}_M(\mathsf{O}(n))$. We use the construction of [11, Appendix B.2], which allows us to lift this upper bound to starting terms of arbitrary shape. We define two functions f and g. The function f maps ground terms over the signature $\mathcal{F}$ of $\Delta_1(M)$ to pairs containing a string over the tape alphabet, and a multiset of restricted terms over $\mathcal{F}$. The purpose of f (compare [11, Lemma B.5]) is to extract a number of restricted terms from a term. The helper function g ensures that the leftmost symbol on the tape of each configuration encoded by a restricted term is indeed a $\vdash$.

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

By [11, Lemma B.8], we get that for every term t over $\mathcal{F}$ with $f(t) = (w, \mathcal{M})$, the inequality $\mathsf{dh}(t, \to_{\Delta_1(M)}) \leq \sum_{s \in \mathcal{M}} \mathsf{dh}(s, \to_{\Delta_1(M)})$ obtains. Moreover, $|\mathcal{M}| \leq |t|$. Hence, $\mathsf{dc}_{\Delta_1(M)}(n) \leq n \cdot \mathsf{rc}_{\Delta_1(M)}(n)$. Thus, the above observations about restricted terms suffice in order to conclude $\mathsf{dc}_{\Delta_1(M)}(n) \in n \cdot \textsc{LifeTime}_M(\mathsf{O}(n))$. ◀

We are now able to transfer Proposition 6 to derivational complexity of term rewriting. Proposition 15 and Lemma 16 take care of the the unrestrictedness of the considered starting terms, and Lemma 18 performs the actual transfer from Turing Machines to TRSs.

▶ **Theorem 19.** *Let $\mathcal{G}$ be a recursively enumerable set of computable, strictly increasing, and total functions $\mathbb{N} \to \mathbb{N}$ such that $\Xi(\mathcal{G})$ is closed under polynomial slowdown. Then the following decision problem is Σ^0_2-complete:*
Instance: *A TRS $\mathcal{R}$.*
Question: *Is $\mathsf{dc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$?*
If the instances are restricted to orthogonal or constructor TRSs, Σ^0_2-completeness also holds.

Proof. The proof of containment of the problem in Σ^0_2 is identical to Theorem 7 *mutatis mutandis*, whence we only show its Σ^0_2-hardness. By Proposition 6, it is Σ^0_2-hard to decide, given a dual-tape Turing Machine M, whether $\textsc{Time}_M \leq \Xi(\mathcal{G})$. By Proposition 15 and Lemma 16, there exists a single-tape Turing machine M' such that $\textsc{LifeTime}_{M'}(n) \in \mathsf{O}(\max\{\textsc{Time}_M(n)^6, n^6\})$, and M' accepts the same language as M. As $\Xi(\mathcal{G})$ is by assumption closed under polynomial slowdown, and contains a strictly increasing function (and hence also a function dominating $i'(n) = n^6$), we have $\textsc{LifeTime}_{M'} \leq \Xi(\mathcal{G})$ iff $\textsc{Time}_M \leq \Xi(\mathcal{G})$. Moreover, by Lemma 18, we have $\mathsf{dc}_{\Delta_1(M')} \in n \cdot \textsc{LifeTime}_{M'}(\mathsf{O}(n))$. As $\Xi(\mathcal{G})$ is closed under polynomial slowdown, it follows that $\mathsf{dc}_{\Delta_1(M')} \leq \Xi(\mathcal{G})$ iff $\textsc{LifeTime}_{M'} \leq \Xi(\mathcal{G})$. The transformations used in Proposition 15 and Lemmas 16 and 18 are obviously computable, and $\Delta_1(M')$ is orthogonal. Therefore, it is Σ^0_2-hard to decide whether $\mathsf{dc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$, given a TRS $\mathcal{R}$ (independent of whether $\mathcal{R}$ is restricted to be orthogonal or a constructor TRS). Note that $\mathsf{dc}_{\Delta_1(M')} \leq \Xi(\mathcal{G})$ iff $\mathsf{rc}_{\Delta_1(M')} \leq \Xi(\mathcal{G})$, hence this is also an alternative proof of the Σ^0_2-completeness of determining whether $\mathsf{rc}_{\mathcal{R}} \leq \Xi(\mathcal{G})$, which places slightly stricter assumptions on $\mathcal{G}$, but allows $\mathcal{R}$ to be restricted to constructor TRSs. ◀

7 Hardness of Minimal Complexity

The proofs in this section and Section 8 are based on the observation that the simulation of a Turing machine M by the TRS $\Delta(M)$ has exactly one redex in each term encoding a configuration of M—that is, each restricted term. Every ilk of problem we consider concerns sets of reductions to some normal form; if there is only one possible reduction starting from every restricted term, the proofs of *hardness* of the various kinds of problems we consider remain virtually identical, regardless of whether we consider minimal or maximal reductions, and regardless of reduction strategy. This crucial observation is stated in Lemma 21 below.

▶ **Definition 20.** We define the *minimal height* of a term s wrt. a finitely branching, terminating relation $\to$ by $\mathsf{mh}(s, \to) = \min\{n : \exists t. s \to^{n,!}_{\mathcal{R}} t\}$. The twin notions of *minimal derivational complexity* and *minimal runtime complexity* of a TRS $\mathcal{R}$ are then defined by:

$$\mathsf{mdc}_{\mathcal{R}}(n) = \max\{\mathsf{mh}(s, \to_{\mathcal{R}}) : |s| \leq n\} \qquad \mathsf{mrc}_{\mathcal{R}}(n) = \max\{\mathsf{mh}(s, \to_{\mathcal{R}}) : |s| \leq n \wedge s \in \mathcal{B}\}\ .$$

▶ **Lemma 21.** *Let M be a dual-tape Turing machine and let s be a term in the signature of $\Delta(M)$ containing exactly one redex. If $s \to_{\Delta(M)} t$, then t contains at most one redex.*

Proof. By assumption, the only redex of s is the one contracted by the step $s \to_{\Delta(M)} t$. Hence, t only contains redexes created by that step. As $\Delta(M)$ is left-linear, redexes can only be created if the right-hand side of the rule $l \to r$ employed in $s \to_{\Delta(M)} t$ overlaps with a left-hand side of some other rule. Write $s \to_{\Delta(M)} t$ as $C[l\sigma] \to_{\Delta(M)} C[r\sigma]$ for a suitable context C and substitution σ. Split on cases as follows:

(a) If l is on one of the forms $q(\cdots)$ or $\mathsf{runM}(x)$, inspection of the rules of $\Delta(M)$ yields that r is on the form $q'(\mathsf{ok}(\triangleright), \cdots)$. Clearly, r can only overlap with the left-hand side of a rule $l' \to r'$ if the overlap occurs at the root of l' and r. As $\Delta(M)$ is orthogonal, at most one such rule $l' \to r'$ can exist, and hence there is at most one redex in t.

(b) If $l = \mathsf{ok}(\mathsf{ok}(\triangleright))$, then $r = \triangleright$. Obviously, $\triangleright$ is a normal form on its own. By assumption, C contains no redex on its own, and $\Delta(M)$ is orthogonal. Therefore, $C[\triangleright]$ contains at most one redex. ◀

▶ **Theorem 22.** *Theorems 7, 9, and 14 all hold with the notion of* $\mathsf{rc}_{\mathcal{R}}$ *replaced by* $\mathsf{mrc}_{\mathcal{R}}$ mutatis mutandis.

Proof. Every basic term s in an orthogonal TRS (such as $\Delta(M)$ for a dual-tape Turing Machine M) contains at most one redex. For each TRS on the form $\Delta(M)$, it is therefore immediate by Lemma 21 that the minimum and maximum lengths of reduction to normal form from s are the same. Therefore, all arguments in the hardness proofs of Theorems 7, 9, and 14 remain sound if we replace $\mathsf{rc}_{\mathcal{R}}$ by $\mathsf{mrc}_{\mathcal{R}}$, so the hardness results follow.

For containment in the respective complexity classes, observe that in the proofs of Theorems 7, 9, and 14 (each of the three distinct variations of) the predicate P considers longest maximal paths in the derivation tree of terms; this can obviously be replaced by the shortest maximal paths, as required by $\mathsf{mrc}_{\mathcal{R}}$, without affecting computability of P. ◀

▶ **Theorem 23.** *Theorem 19 holds with the notion of* $\mathsf{dc}_{\mathcal{R}}$ *replaced by* $\mathsf{mdc}_{\mathcal{R}}$ mutatis mutandis.

Proof. Containment in Σ_2^0 follows in the same way as in the proof of Theorem 22.

We now show Σ_2^0-hardness. Observe that for any (single-tape) Turing Machine M, the TRS $\Delta_1(M)$ is orthogonal, right-linear, and nonerasing. Therefore, $\Delta_1(M)$ has the diamond property, and for any term t, we have $\mathsf{dh}(t, \to_{\Delta_1(M)}) = \mathsf{mh}(t, \to_{\Delta_1(M)})$. With this, the hardness result follows by arguments identical to those in the proof of Theorem 19. ◀

8 Hardness under Strategies

The results so far concern TRSs with unconstrained rewrite relation. In the modelling of programming languages, it is common to consider TRSs with strategies dictating the redex to be contracted in each term. Using the same ideas as in the last section, the previous results in the paper carry over to the setting of TRSs with strategies[2]. Thus, the results of the previous sections of the paper remain valid under, for example, any innermost strategy, and under deterministic strategies such as the leftmost-outermost strategy.

▶ **Definition 24.** Let $\mathcal{R}$ be a TRS. A *strategy* $\mathbb{S}$ for $\mathcal{R}$ is defined by a relation $\to_{\mathbb{S}} \subseteq \to_{\mathcal{R}}$ such that any term t is a normal form of $\to_{\mathcal{R}}$ iff it is a normal form of $\to_{\mathbb{S}}$. We call a strategy for $\mathcal{R}$ *computable* if, given a term t, the (finite) set $\{t' : t \to_{\mathbb{S}} t'\}$ is computable.

The notions of runtime and derivational complexity of TRSs with strategies are defined *mutatis mutandis*. For the next theorem, Lemma 21 is again the crucial proof ingredient.

▶ **Theorem 25.** *Let f be a computable mapping returning a computable strategy $f(\mathcal{R})$ for each TRS $\mathcal{R}$. Theorems 7, 9, and 14, 19, 22 and 23 all hold for the rewrite relation of $\mathcal{R}$ with strategy $\mathbb{S} = f(\mathcal{R})$ (where the instance in each decision problem is $\mathcal{R}$).*

[2] Here we use the notion "strategy" according to [28, Definition 9.1.1]. Note that this does not cover everything that is commonly called a "strategy" in term rewriting. For instance, the proofs of this section can not be directly carried over to *context-sensitive rewriting*.

Proof. Observe that if term s contains exactly one redex, then for any term t and strategy $\mathbb{S}$ for $\mathcal{R}$, we have $s \to_{\mathcal{R}} t$ iff $s \to_{\mathbb{S}} t$. For every TRS on the form $\Delta(M)$, each basic term of $\Delta(M)$ has at most one redex. By Lemma 21, it is immediate that the lengths of all reductions to normal form from s are the same. Therefore, all arguments in the hardness proofs of Theorems 7, 9, 14, and 22, remain sound under $\mathbb{S}$.

For Σ^0_2-hardness of the remaining two properties, observe that for any (single-tape) Turing Machine M, the TRS $\Delta_1(M)$ is orthogonal, right-linear, and nonerasing. Therefore, $\Delta_1(M)$ has the diamond property, and for any term t, all reductions from t to its (unique) normal form have the same length. In particular, we have $\mathsf{dh}(t, \to_{\Delta_1(M)}) = \mathsf{dh}(t, \to_{\mathbb{S}})$. Hence, the hardness proofs for Theorems 19 and 23 remain sound when restricted to $\mathbb{S}$.

To prove containment in the respective classes of the arithmetical hierarchy, observe that each containment proof in Theorems 7, 9, 14, 19, 22, and 23 is done by computing the derivation tree starting from a term s to a certain depth. The derivation tree with respect to a strategy can be obtained by pruning the full derivation tree: A branch $t \to t''$ (and thus, the entire subtree starting from t'') is cut off if $t'' \notin \{t' : t \to_{\mathbb{S}} t'\}$. As the strategy is computable, the pruning operation is clearly computable, hence also the pruned derivation trees, and we may thus replace the trees in the proofs of the above theorems by their pruned versions, concluding the proof. ◀

9 Conclusion and Suggestions for Future Work

We have proved that a number of problems related to bounding the derivational and runtime complexity of rewrite systems are complete for classes in the arithmetical hierarchy. We hope that our results may be used to prove the exact hardness other problems in applied logic—this would avoid the tedium of pure reduction from Turing machines.

A related open problem is Problem #107 of RTALooP[3], a list of open problems collected by term rewriters: what are complete characterisations of polynomial derivational complexity?

Furthermore, recent efforts have been made to devise automated methods for showing whether the derivational or runtime complexity of a given TRS is polynomial, see for instance [2, 16, 30, 29]. All of this recent work was focused on proving termination of a TRS by some restricted means, and then extracting a complexity bound from that termination proof. However, the position of this problem in the arithmetical hierarchy (which is the same as the position of *non*termination analysis) suggests that it would be promising to try to certify polynomial complexity bounds for rewrite systems in a completely novel way.

References

1 A. Asperti. The intensional content of Rice's theorem. In *Proc. 35th POPL*, pages 113–119. ACM, 2008.

2 M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proc. 9th FLOPS*, volume 4989 of *LNCS*, pages 130–146, 2008.

3 M. Avanzini and G. Moser. Closing the gap between runtime complexity and polytime computability. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 33–48, 2010.

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

5 G. Bonfante, E. A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *J. Funct. Program.*, 11(1):33–53, 2001.

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

6 R. Book. Time-bounded grammars and their languages. *J. Comput. Syst. Sci.*, 5(4):397–429, 1971.

7 C. Choppy, S. Kaplan, and M. Soria. Complexity analysis of term-rewriting systems. *Theor. Comput. Sci.*, 67(2):261–282, 1989.

8 E. A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *Proc. 11th CADE*, volume 607 of *LNCS*, pages 139–147, 1992.

9 J. Endrullis, H. Geuvers, J. G. Simonsen, and H. Zantema. Levels of undecidability in rewriting. *Inform. Comput.*, 209(2):227 – 245, 2011.

10 M. Fernandez. *Models of Computation: An Introduction to Computability Theory*. Undergraduate topics in computer science. Springer London, 2009.

11 M. C. F. Ferreira. *Termination of term rewriting*. PhD thesis, Universiteit Utrecht, 1995.

12 P. Hájek. Arithmetical hierarchy and complexity of computation. *Theor. Comput. Sci.*, 8:227–237, 1979.

13 J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *T. Am. Math. Soc.*, 117:285–306, 1965.

14 F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape Turing Machines. *J. ACM*, 13(4):533–546, 1966.

15 G. T. Herman. Strong computability and variants of the uniform halting problem. *Math. Logic Quart.*, 17:115–131, 1971.

16 N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. 4th IJCAR*, volume 5195 of *LNCS*, pages 364–379, 2008.

17 D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *Theor. Comput. Sci.*, 105(1):129–140, 1992.

18 D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. 3rd RTA*, volume 355 of *LNCS*, pages 167–177, 1989.

19 J.-Y. Marion. Analysing the implicit complexity of programs. *Inform. Comput.*, 183(1):2–18, 2003.

20 J.-Y. Marion and J.-Y. Moyen. Heap-size analysis for assembly programs, 2006. Unpublished manuscript. Available at `http://hal.archives-ouvertes.fr/docs/00/06/78/38/PDF/main.pdf`.

21 J.-Y. Marion and R. Péchoux. Sup-interpretations, a semantic method for static analysis of program resources. *ACM Trans. Comput. Log.*, 10(4), 2009.

22 G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 255–269, 2009.

23 K. W. Regan. Arithmetical degrees of index sets for complexity classes. In *Logic and Machines*, volume 171 of *LNCS*, pages 118–130, 1983.

24 G. Roşu. Equality of streams is a Π^0_2-complete problem. In *Proc. 11th ICFP*. ACM, 2006.

25 H. Rogers Jr. *Theory of Recursive Functions and Effective Computability*. The MIT Press, paperback edition, 1987.

26 A. Schnabl and J. G. Simonsen. The exact hardness of deciding derivational and runtime complexity, 2011. Extended version. Available at `http://cl-informatik.uibk.ac.at/users/aschnabl`.

27 S. Sippu. Derivational complexity of context-free grammars. *Inform. Control*, 53(1–2):52–65, 1982.

28 TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

29 J. Waldmann. Polynomially bounded matrix interpretations. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 357–372, 2010.

30 H. Zankl and M. Korp. Modular complexity analysis via relative complexity. In *Proc. 21st RTA*, volume 6 of *LIPIcs*, pages 385–400, 2010.