Complexity Analysis by Graph Rewriting*

Martin Avanzini and Georg Moser

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

Abstract. Recently, many techniques have been introduced that allow the (automated) classification of the runtime complexity of term rewrite systems (TRSs for short). In this paper we show that polynomial (innermost) runtime complexity of TRSs induces polytime computability of the functions defined. In this way we show a tight correspondence between the number of steps performed in a given rewrite system and the computational complexity of an implementation of rewriting. The result uses graph rewriting as a first step towards the implementation of term rewriting. In particular, we prove the adequacy of (innermost) graph rewriting for (innermost) term rewriting.

1 Introduction

We study techniques to analyse the complexity of programs automatically. Instead of studying programs of a particular programming language directly, we focus on the complexity analysis of term rewrite systems instead. The reason is that term rewriting is a very simple formalism underlying many programming languages. Recently, many techniques to automatically assess the runtime complexity of term rewrite systems (TRSs for short) have been introduced. For example in [1,3] we introduced the *polynomial path order* POP*and extensions of it. POP* is a restriction of the multiset path order [13] and whenever compatibility of a TRS $\mathcal{R}$ with POP* can be shown then the runtime complexity of $\mathcal{R}$ is polynomially bounded. Here the *runtime complexity* of a TRS measures the maximal number of rewrite steps as a function in the size of the initial term.

We have successfully implemented this technique.[1] Thus we can automatically verify for a given TRS $\mathcal{R}$ that it admits at most polynomial runtime complexity. This opens the way to automatically verify for a given (functional) program P that its runtime complexity is polynomial (in the input size). The only restrictions in the applicability of the result are that (i) the program P is transformable into a term rewrite system $\mathcal{R}$ and (ii) a feasible (i.e., polynomial) runtime complexity with respect to $\mathcal{R}$ gives rise to a feasible runtime complexity of P. In short the transformation has to be *non-termination* and *complexity preserving*.

* This research is supported by FWF (Austrian Science Fund) projects P20133.

[1] The here mentioned polynomial path orders are one complexity technique implemented in the *Tyrolean Complexity Tool*, a complexity tool to analyse the runtime complexity of TRSs. The program is open-source and freely available at `http://cl-informatik.uibk.ac.at/software/tct/`.

As an example of the technique consider the following rendering of *insert sort* as TRS $\mathcal{R}_1$ given in [11]. Observe that for a list xs build from constructors cs and nil, $\mathsf{sort}(xs, \mathsf{len}(xs))$ returns xs sorted.

$$
\begin{aligned}
\mathsf{if}(\mathsf{true}, x, y) &\to x & 0 < \mathsf{s}(x) &\to \mathsf{true}\\
\mathsf{if}(\mathsf{false}, x, y) &\to y & \mathsf{ins}(0, x, ys) &\to \mathsf{cs}(x, ys)\\
\mathsf{len}(\mathsf{nil}) &\to 0 & \mathsf{ins}(\mathsf{s}(n), x, \mathsf{cs}(y, ys)) &\to \mathsf{if}(x < y, \mathsf{cs}(x, \mathsf{cs}(y, ys)),\\
\mathsf{len}(\mathsf{cs}(x, xs)) &\to \mathsf{s}(\mathsf{len}(xs)) & & \qquad \mathsf{cs}(y, \mathsf{ins}(n, x, ys)))\\
x < 0 &\to \mathsf{false} & \mathsf{sort}(\mathsf{nil}, 0) &\to \mathsf{nil}\\
\mathsf{s}(x) < \mathsf{s}(y) &\to x < y & \mathsf{sort}(\mathsf{cs}(x, xs), \mathsf{s}(n)) &\to \mathsf{ins}(n, x, \mathsf{sort}(xs, n))
\end{aligned}
$$

It is not difficult to see that the innermost runtime complexity of the TRS $\mathcal{R}_1$ is polynomially bounded. Moreover this fact can be verified automatically by showing that $\mathcal{R}_1 \subseteq >_{\mathsf{pop}*}$ for a suitable instance $>_{\mathsf{pop}*}$ of POP^*, cf. [1].

Once we have established a bound on the (innermost) runtime complexity of a TRS, it is natural to direct our attention to the computational complexity of the functions defined by the TRS. In particular with respect to TRS $\mathcal{R}_1$ we also want to verify (automatically if possible) that insertion sort is polytime computable. Due to the restrictive definition of $>_{\mathsf{pop}*}$ this is not difficult as the signature of $\mathcal{R}_1$ is *simple*. (See [2] for a proof.) Here a simple signature [11] essentially means that the size of any constructor term depends polynomially on its depth.

Unfortunately the restriction to a simple signature is rather restrictive. Consider the following TRS $\mathcal{R}_2$. (This is Example 8 in [9].)

$$
\begin{aligned}
\mathsf{D}(\mathsf{c}) &\to 0 & \mathsf{D}(x + y) &\to \mathsf{D}(x) + \mathsf{D}(y)\\
\mathsf{D}(\mathsf{t}) &\to 1 & \mathsf{D}(x - y) &\to \mathsf{D}(x) - \mathsf{D}(y)\\
\mathsf{D}(x \times y) &\to y \times \mathsf{D}(x) + x \times \mathsf{D}(y)
\end{aligned}
$$

Employing runtime complexity techniques developed in [10], we can automatically verify that the runtime complexity of $\mathcal{R}_2$ is polynomially bounded. However the signature of $\mathcal{R}_2$ is *not* simple and thus we cannot directly conclude polytime computability of the function computed. The main obstacle here is that due to the last rule, a single step may copy arbitrary large subterms.

In this paper we show that polynomial runtime complexity of TRSs induces polytime computability of the functions defined. The only restriction is that we consider an eager evaluation strategy, i.e., we base our study on an *innermost rewriting strategy*. We show the precise correspondence between the number of steps performed in a given rewrite system and the computational complexity of an implementation of rewriting.

In order to overcome the problem of copying we use graph rewriting—here copying is replaced by sharing—as a first step towards the implementation of term rewriting. We re-prove the adequacy of *innermost* graph rewriting for *innermost* term rewriting, compare for example [12,13]. A new proof becomes necessary as we need to control the resources needed in the simulation. This

pedantry is then used to establish the tight correspondence between the complexity of a given TRS $\mathcal{R}$ and the intrinsic computational complexity of the functions computed by $\mathcal{R}$.

The rest of the paper is organised as follows. In Section 2 we present basic notions and recall (briefly) the central concepts of graph rewriting. The adequacy theorem is provided in Section 3 and in Section 4 we show how innermost graph rewriting can be encoded efficiently. Finally we conclude in Section 5, where we also relate our work to recent work. Due to space limitations, we omit some proofs in the presentation. Those are available in the technical report [4].

2 Preliminaries

We assume familiarity with term rewriting [5,13], but no familiarity with graph rewriting (see [13]) is assumed. The purpose of this section is to fix the term rewriting notions and introduce a formulation of graph rewriting that is sufficient for our purposes.

Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a finite signature. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}$. The *arity* of a function symbol f is denoted as $\mathsf{ar}(f)$. $\mathcal{V}\mathsf{ar}(t)$ denotes the set of variables occurring in a term t. The *size* of a term t, i.e., the number of symbols appearing in t, is denoted as $|t|$. We write $t|_p$ for the *subterm* of t at position p.

A *term rewrite system* (*TRS* for short) $\mathcal{R}$ (over a signature $\mathcal{F}$) is a *finite* set of rewrite rules $l \to r$, such that $l \notin \mathcal{V}$ and $\mathcal{V}\mathsf{ar}(l) \supseteq \mathcal{V}\mathsf{ar}(r)$. The root symbols of left-hand sides of rewrite rules are called *defined*, while all other function symbols are called *constructors*. Constructor symbols are collected in $\mathcal{C} \subseteq \mathcal{F}$. The smallest rewrite relation, i.e., closed under contexts and substitutions, that contains $\mathcal{R}$ is denoted by $\to_{\mathcal{R}}$. We write $\to_{\mathcal{R}}^*$ for the transitive and reflexive closure of $\to_{\mathcal{R}}$. Let s and t be terms. If exactly n steps are performed to contract s to t we write $s \to_{\mathcal{R}}^n t$. A term $s \in \mathcal{T}$ is called a *normal form* if there is no $t \in \mathcal{T}$ such that $s \to_{\mathcal{R}} t$. With $\mathsf{NF}(\mathcal{R})$ we denote the set of all normal forms of a term rewrite system $\mathcal{R}$. We write $s \to_{\mathcal{R}}^! t$ if $s \to_{\mathcal{R}}^* t$ and $t \in \mathsf{NF}(\mathcal{R})$. A term t is called *argument normalised (with respect to $\mathcal{R}$)* if every proper subterm $t|_p \in \mathsf{NF}(\mathcal{R})$.

Let $\Box$ be a fresh constant. Terms over $\mathcal{F} \cup \{\Box\}$ and $\mathcal{V}$ are called *contexts*. The empty context is denoted as $\Box$. For a context C with n holes, we write $C[t_1, \ldots, t_n]$ for the term obtained by replacing the holes from left to right in C with the terms $t_1, \ldots, t_n$. The *innermost rewrite relation* $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ of a TRS $\mathcal{R}$ is defined on terms as follows: $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ if there exists a rewrite rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s = C[l\sigma]$ and $t = C[r\sigma]$ for $l\sigma$ argument normalised.

A TRS is called *confluent* if for all $s, t_1, t_2 \in \mathcal{T}$ with $s \to_{\mathcal{R}}^* t_1$ and $s \to_{\mathcal{R}}^* t_2$ there exists a term t_3 such that $t_1 \to_{\mathcal{R}}^* t_3$ and $t_2 \to_{\mathcal{R}}^* t_3$.

In the sequel we introduce the central concepts of *term graph rewriting* or *graph rewriting* for short. We concentrate on standard approaches in the context of rewriting, compare also [12,13].

A *graph* $G = (\mathsf{V}_G, \mathsf{succ}_G, \mathsf{L}_G)$ over the set $\mathcal{L}$ is a structure such that V_G is a finite set, the *nodes* or *vertexes*, $\mathsf{succ}_G \colon \mathsf{V}_G \to \mathsf{V}_G^*$ is a mapping that associates a node n with an (ordered) sequence of nodes, called the *successors* of n. Finally $\mathsf{L}_G \colon \mathsf{V}_G \to \mathcal{L}$ is a mapping that associates each node n with its *label* $\mathsf{L}_G(n)$. Often we drop the reference to the graph G from V_G, succ_G, and L_G, i.e., we write $G = (\mathsf{V}, \mathsf{succ}, \mathsf{L})$. If possible, we will omit references to the constituents of the structure G; in particular we often write $n \in G$ instead of $n \in \mathsf{V}$. Note that the sequence of successors of n may be empty: $\mathsf{succ}(n) = []$. Typically the set of labels $\mathcal{L}$ is clear from context and not explicitly mentioned.

Definition 1. *Let $G = (\mathsf{V}, \mathsf{succ}, \mathsf{L})$ be a graph and let $n \in \mathsf{V}$. Consider $\mathsf{succ}(n) = [n_1, \ldots, n_k]$. We call n_i the* i-th successor *of n (denoted as $n \xrightarrow{i} n_i$). If there exists i such that $n \xrightarrow{i} m$, then we simply write $n \rightharpoonup m$. A node m is called* reachable *from n if $n \xrightarrow{*} m$, where $\xrightarrow{*}$ denotes the reflexive and transitive closure of $\rightharpoonup$. We write $\xrightarrow{+}$ for $\rightharpoonup \cdot \xrightarrow{*}$.*

A graph G is *acyclic* if $n \xrightarrow{+} m$ implies $n \neq m$ and G is *rooted* if there exists a unique node n such that every other node in G is reachable from n. The node n is called the *root* $\mathsf{rt}(G)$ of G. The *size* of G, i.e. the number of nodes, is denoted as $|G|$. We write $G|n$ for the subgraph of G reachable from n.

Definition 2. *A* term graph *(with respect to $\mathcal{F}$ and $\mathcal{V}$) is an acyclic and rooted graph $G = (\mathsf{V}, \mathsf{succ}, \mathsf{L})$ over $\mathcal{F} \cup \mathcal{V}$. Let $n \in \mathsf{V}$ and suppose $\mathsf{L}(n) = f \in \mathcal{F}$ such that f is k-ary ($k \geqslant 0$). Then $\mathsf{succ}(n) = [n_1, \ldots, n_k]$. On the other hand if $\mathsf{L}(n) \in \mathcal{V}$, then $\mathsf{succ}(n) = []$. We demand that any variable node is* shared*, i.e., for $n \in \mathsf{V}$ with $\mathsf{L}(n) \in \mathcal{V}$, if $\mathsf{L}(n) = \mathsf{L}(m)$ for $m \in \mathsf{V}$ then $n = m$.*

We set $\mathcal{V}\mathsf{ar}(G) := \{n \mid n \in G, \mathsf{L}(n) \in \mathcal{V}\}$ to denote the set of *variable nodes* in G. Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. We keep $\mathcal{R}$ and $\mathcal{F}$ fixed for the remainder of this paper. We write $\mathcal{G}\mathsf{raph}$ for the set of all term graphs with respect to $\mathcal{F}$ and $\mathcal{V}$.

Example 3. Consider the graph $G = (\{1, 2\}, \mathsf{succ}, \mathsf{L})$ where $\mathsf{succ}(1) = [2, 2]$ and $\mathsf{L}(1) = \mathsf{f} \in \mathcal{F}$ and $\mathsf{L}(2) = x \in \mathcal{V}$. Then G is a term graph. Intuitively G represents the term $\mathsf{f}(x, x)$ such that the variable x is shared.

We define the term $t := \mathsf{term}(G)$ *represented* by G as follows:

$$t := \begin{cases} x & \text{if } \mathsf{L}(\mathsf{rt}(G)) = x \in \mathcal{V} \\ f(t_1, \ldots, t_k) & \text{if } \mathsf{L}(\mathsf{rt}(G)) = f \in \mathcal{F} \text{ and } \mathsf{succ}(\mathsf{rt}(G)) = [n_1, \ldots, n_k] \,. \end{cases}$$

Here $t_i := \mathsf{term}(G|n_i)$ for $i = 1, \ldots, k$. We write $\mathsf{Term}(G)$ for the set of subterms of $\mathsf{term}(G)$.

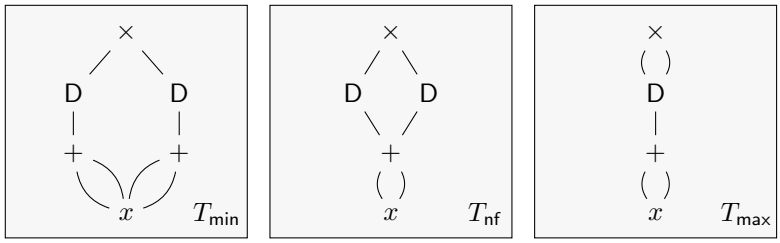
Let $G \in \mathcal{G}\mathsf{raph}$. A *position* in G is a finite sequence of positive integers. The position of $\mathsf{rt}(G)$ is the empty sequence (denoted as ε). For positions p and q we write pq for their concatenation. The set of *positions* $\mathcal{P}os_G(n)$ of $n \in G$ is defined as $\mathcal{P}os_G(n) := \{\varepsilon\}$ if $n = \mathsf{rt}(G)$ and $\mathcal{P}os_G(n) := \{i_1 \ldots i_k \mid \mathsf{rt}(G) \xrightarrow{i_1} \ldots \xrightarrow{i_k} n\}$.

Note that for any node n: $\mathcal{P}os_G(n) \neq \varnothing$. It is easy to see that for $p \in \mathcal{P}os_G(n)$, $\mathsf{term}(G\restriction n) = \mathsf{term}(G)|_p$.

We say that $n \in G$ is *shared*, if n represents more than one subterm of $\mathsf{term}(G)$. Note that n is shared if the set of positions $\mathcal{P}os_G(n)$ of n is not a singleton. If $\mathcal{P}os_G(n)$ is a singleton, then n is *unshared*.

Definition 4. *A node n is* minimally shared *if it is a variable node or unshared. We say n is* maximally shared *if for all $m \in G$ such that $\mathsf{term}(G\restriction m) = \mathsf{term}(G\restriction n)$ implies $n = m$. A term graph G is* normal form sharing (with respect to $\mathcal{R}$) *if for all nodes $n \in G$, if $\mathsf{term}(G\restriction n) \in \mathsf{NF}(\mathcal{R})$ then n is maximally shared and minimally shared otherwise. Let t be a term. We write $\triangle(t)$ ($\diamondsuit(t)$) for the set of minimally (normal form) sharing term graphs that represent t.*

Example 5. Reconsider the TRS $\mathcal{R}_2$ from the introduction. Let $t = \mathsf{D}(x + x) \times \mathsf{D}(x + x)$, represented by the term graphs T_{min}, T_{nf} and T_{max} depicted as follows:



Then $T_{\mathsf{min}} \in \triangle(t)$ as only variable nodes are shared. Further $T_{\mathsf{nf}} \in \diamondsuit(t)$ since the subterms $\{x + x, x\} = \mathsf{Term}(T_{\mathsf{nf}}) \cap \mathsf{NF}(\mathcal{R})$ are represented by unique nodes whereas all other nodes are unshared. T_{max} contains only maximally shared nodes.

Let G and H be two graphs, possibly sharing nodes. We write $G \cup H$ for their union. We say that G and H are *properly sharing* if $n \in G \cap H$ implies $\mathsf{L}_G(n) = \mathsf{L}_H(n)$ and $\mathsf{succ}_G(n) = \mathsf{succ}_H(n)$. For two properly sharing graphs G and H, $G \cup H$ is again a graph (but possibly not a term graph).

Definition 6. *Let G be a term graph and H a rooted graph over $\mathcal{F} \cup \mathcal{V}$. We denote by $G[H]_n$ the graph obtained by replacing the subgraph at node n in G by H: $G[H]_n := (G[n \leftarrow \mathsf{rt}(H)] \cup H)\restriction m$ where $m := \mathsf{rt}(H)$ if $n = \mathsf{rt}(G)$ and $m := \mathsf{rt}(G)$ otherwise. Here $G[n \leftarrow m]$ denotes the* redirection *of n to m in G: Set $\mathsf{V}_{G'} := (\mathsf{V}_G \cup \{m\}) \setminus \{n\}$ and for all $p \in \mathsf{V}_{G'}$, $\mathsf{succ}_{G'}(p) := r^*(\mathsf{succ}_G(p))$. Here r replaces n by m: $r(n) = m$ and $r(p) = p$ for all $p \in \mathsf{V}_{G'}$. Further r^* denotes the standard extension of r to sequences. Finally, set $G[n \leftarrow m] := G' = (\mathsf{V}_{G'}, \mathsf{succ}_{G'}, \mathsf{L}_G)$.*

Observe that for properly sharing term graphs G and H such that $n \notin H$ and H acyclic, $G[H]_n$ is again a term graph. Let G and H be two term graphs. A *morphism* (denoted $m\colon G \to H$) is a function $m\colon \mathsf{V}_G \to \mathsf{V}_H$ such that $m(\mathsf{rt}(G)) = \mathsf{rt}(H)$, and for all n with $\mathsf{L}_G(n) \in \mathcal{F}$, $\mathsf{L}_G(n) = \mathsf{L}_H(m(n))$ and $m^*(\mathsf{succ}_G(n)) = \mathsf{succ}_H(m(n))$. The next lemma follows directly from the definition.

Lemma 7. *Suppose* $m\colon G \to H$ *and* $m'\colon G \to H$.

1) For any $n \in G$ *we have* $m\colon G|n \to H|m(n)$.
2) For any $n \in G$ *we have* $m(n) = m'(n)$.

We write $G \geqslant_m H$ (or $G \geqslant H$ for short) if there exists a surjective morphism $m\colon G \to H$ that preservers labels from $\mathcal{V}$. In this case we have for all $n \in \mathsf{V}_G$: $\mathsf{L}_G(n) = \mathsf{L}_H(m(n))$. When the graph morphism m is non-injective we write $G >_m H$ (or $G > H$ for short). If m is injective and surjective then m is an *isomorphism*. Conclusively G and H are called *isomorphic* (denoted as $G \cong H$).

The next lemma follows by a simple inductive argument.

Lemma 8. *For all term graph* G *and* H, $G \geqslant_m H$ *implies* $\mathsf{term}(G) = \mathsf{term}(H)$.

Definition 9. *Let* L, R *be two properly sharing term graphs. Suppose* $\mathcal{V}\mathsf{ar}(R) \subseteq \mathcal{V}\mathsf{ar}(L)$, $\mathsf{L}(\mathsf{rt}(L)) \in \mathcal{F}$ *and* $\mathsf{rt}(L) \notin R$. *Then the graph* $L \cup R$ *is called a* graph rewrite rule *(*rule *for short). A rule is denoted as* $L \to R$, *where* L, R *denotes the left-hand, right-hand side of* $L \to R$, *respectively. A* graph rewrite system (GRS) $\mathcal{G}$ *is a set of graph rewrite rules.*

Note that for a rule $L \to R$, the graphs L and R share at least all variable nodes. A rule $L' \to R'$ is called a *renaming* of $L \to R$ with respect to S if $L' \to R' \cong L \to R$ and $\mathsf{V}_S \cap \mathsf{V}_{L' \to R'} = \varnothing$. Let $\mathcal{G}$ be a GRS, let $S \in \mathcal{G}\mathsf{raph}$ and let $L \to R$ be a rule. A *redex* of S with $L \to R$ is a node $u \in S$ such that there exists a renaming $L' \to R'$ of $(L \to R) \in \mathcal{G}$ with respect to S such that $m\colon L' \to S|u$ is a morphism and $T = S[m(R')]_u$. Here $m(R')$ denotes the structure obtained by replacing in R' every node $v \in L' \cap R'$ by $m(v) \in S$, where the labels of $m(v) \in m(R')$ are the labels of $m(v) \in S$. From $m\colon L' \to S|v$ we obtain that T is a term graph.

Definition 10. *We say* S rewrites *to* T *if there exists a rule* $(L \to R) \in \mathcal{G}$ *such that* $n \in S$ *is a redex with this rule. This is denoted as* $S \Rightarrow_{\mathcal{G},n,L\to R} T$. *We set* $S \Rightarrow_{\mathcal{G}} T$ *if* $S \Rightarrow_{\mathcal{G},n,L\to R} T$ *for some* $n \in S$ *and* $(L \to R) \in \mathcal{G}$. *The relation* $\Rightarrow_{\mathcal{G}}$ *is called the* graph rewrite relation *induced by* $\mathcal{G}$.

We denote the set of normal forms with respect to $\Rightarrow_{\mathcal{G}}$ as $\mathsf{NF}(\mathcal{G})$. The *innermost graph rewrite relation* $\overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}}$ is the restriction of $\Rightarrow_{\mathcal{G}}$ where arguments need to be in normal form, i.e. $S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G},n,L\to R} T$ if $S \Rightarrow_{\mathcal{G},n,L\to R} T$ and for all $m \in \mathsf{succ}_S(n)$, $S|m \in \mathsf{NF}(\mathcal{G})$.

3 Adequacy of Graph Rewriting for Term Rewriting

In this section we show that graph rewriting is adequate for term rewriting if we restrict our attention to innermost (graph) rewriting. This holds without further restrictions on the studied TRS $\mathcal{R}$. (However, we assume that $\mathcal{R}$ is finite.) The here presented adequacy theorem (see Theorem 19) is not essentially new. Related results can be found in the extensive literature, see for example [13]. In

particular in [12] the adequacy theorem is even generalised to full rewriting. As this approach involves copying, it is currently not clear whether it applies in our context. Still our treatment of innermost rewriting for unrestricted TRSs is new. (See [8] for strongly related work on orthogonal constructor TRSs.) Furthermore the detailed analysis given in this section is a necessary foundation for the precise characterisation of the implementation of graph rewriting, presented in Section 4.

Definition 11. *The* simulating graph rewrite system $\mathcal{G}(\mathcal{R})$ *of* $\mathcal{R}$ *contains for each rule* $l \to r \in \mathcal{R}$ *some rule* $L \to R$ *such that* $L \in \triangle(l)$ *and* $R \in \triangle(r)$ *have only variable-nodes in common, i.e.* $\mathsf{V}_L \cap \mathsf{V}_R = \mathcal{V}\mathsf{ar}(R)$.

Below L, R, S and T denote term graphs. Suppose $m\colon L \to S$ is a morphism. Then m induces a substitution $\sigma_m\colon \mathcal{V} \to \mathsf{Term}(S)$: For any $n \in S$ such that $\mathsf{L}(n) = x \in \mathcal{V}$ we set $\sigma_m(x) := \mathsf{term}(S{\restriction}m(n))$.

Lemma 12. *Let* L *and* S *be term graphs. Suppose* $m\colon L \to S$ *for some morphism* m. *Then for each node* $n \in L$, $\mathsf{term}(L{\restriction}n)\sigma_m = \mathsf{term}(S{\restriction}m(n))$. *In particular,* $\mathsf{term}(L)\sigma_m = \mathsf{term}(S)$.

Proof. We prove the lemma by induction on $l := \mathsf{term}(L{\restriction}n)$. We write σ instead of σ_m. If $l \in \mathcal{V}$, then by definition $L{\restriction}n$ consists of a single (variable-)node and $l\sigma = \mathsf{term}(S{\restriction}m(n))$ follows by the definition of the induced substitution σ. If $l = f(l_1, \ldots, l_k)$, then we have $\mathsf{succ}_L(n) = [n_1, \ldots, n_k]$ for some $n_1, \ldots, n_k \in L$. As $m\colon L \to S$ holds, Lemma 7 yields $m\colon L{\restriction}n_i \to S{\restriction}m(n_i)$ for all $i = 1, \ldots, k$. And induction hypothesis becomes applicable so that $l_i\sigma = \mathsf{term}(S{\restriction}m(n_i))$. Thus

$$l\sigma = f(l_1\sigma, \ldots, l_k\sigma) = f(\mathsf{term}(S{\restriction}m(n_1)), \ldots, \mathsf{term}(S{\restriction}m(n_k)))\ .$$

By definition of m, $\mathsf{L}_S(m(n)) = \mathsf{L}_L(n) = f$ and $\mathsf{succ}_S(m(n)) = m^*(\mathsf{succ}_L(n)) = [m(n_1), \ldots, m(n_k)]$. Hence $f(\mathsf{term}(S \restriction m(n_1)), \ldots, \mathsf{term}(S \restriction m(n_k))) = \mathsf{term}(S \restriction m(n))$.

Finally, in order to conclude $\mathsf{term}(L)\sigma = \mathsf{term}(S)$, it suffices to observe that by definition of m: $m(\mathsf{rt}(L)) = \mathsf{rt}(S)$ holds. □

Lemma 13. *Let* $L \to R$ *be a graph rewrite rule and let* S *be a term graph such that* $S \cap R = \varnothing$ *and* $m\colon L \to S$ *for some morphism* m. *Let* m' *denote the standard extension of* m *to all nodes in* R, *i.e.,* $m'(n) := m(n)$ *if* n *is in the domain of* m *and* $m'(n) := n$ *otherwise. Set* $T := (m'(R) \cup S){\restriction}\mathsf{rt}(m'(R))$. *Then for each* $n \in R$, $\mathsf{term}(R{\restriction}n)\sigma_m = \mathsf{term}(T{\restriction}m'(n))$. *In particular,* $\mathsf{term}(R)\sigma_m = \mathsf{term}(T)$.

Proof. We write σ instead of σ_m. Suppose $n \in R \cap L$. Then $R{\restriction}n = L{\restriction}n$ as L, R are properly shared. By definition of the morphism m: $m(n) \in S$. Thus by definition of T we have $\mathsf{term}(T{\restriction}m'(n)) = \mathsf{term}(S{\restriction}m(n))$. Moreover, employing Lemma 12, we have $\mathsf{term}(R{\restriction}n)\sigma = \mathsf{term}(L{\restriction}n)\sigma = \mathsf{term}(S{\restriction}m(n))$. From this the assertion follows.

Thus suppose $n \in R \setminus L$. This subcase we prove by induction on $r = \mathsf{term}(R \restriction n)$. The base case $r \in \mathcal{V}$ follows as variables are shared in $L \to R$. For the inductive step, let $r = f(r_1, \ldots, r_k)$ with $\mathsf{succ}_R(n) = [n_1, \ldots, n_k]$. The

induction hypothesis yields $r_i\sigma = \mathsf{term}(T\restriction m'(n_i))$ for $i = 1, \ldots, k$. By definition of m': $m'(n) = n \in m'(R) \subseteq T$. Hence $\mathsf{succ}_T(m'(n)) = \mathsf{succ}_{m'(R)}(n) = m'^*(\mathsf{succ}_R(n)) = [m'(n_1), \ldots, m'(n_k)]$. Moreover $\mathsf{L}_T(n) = \mathsf{L}_{m'(R)}(n) = f$ by definition. We conclude $r\sigma = f(r_1\sigma, \ldots, r_k\sigma) = f(\mathsf{term}(T\restriction m'(n_1)), \ldots, \mathsf{term}(T\restriction m'(n_k))) = \mathsf{term}(T\restriction m'(n))$. This concludes the inductive argument.

Finally, in order to conclude $\mathsf{term}(R)\sigma_m = \mathsf{term}(T)$ observe that $m(\mathsf{rt}(T)) = \mathsf{rt}(R)$ holds. □

Below we also write □ for the unique (up-to isomorphism) graph representing the constant □.

Lemma 14. *Let S and T be two properly sharing term graphs, let $n \in S \setminus T$. Then $\mathsf{term}(S[T]_n) = C[\mathsf{term}(T), \ldots, \mathsf{term}(T)]$ where $C = \mathsf{term}(S[\Box]_n)$ and the number of occurrences of the term $\mathsf{term}(T)$ equals $|\mathcal{P}os_S(n)|$.*

Proof. We proceed by induction on the size of S. In the base case S consists of a single node n. Hence the context C is empty and the lemma follows trivially. For the induction step we can assume without loss of generality that $n \neq \mathsf{rt}(S)$.

We assume $\mathsf{L}_S(\mathsf{rt}(S)) = f \in \mathcal{F}$ and $\mathsf{succ}_S(\mathsf{rt}(S)) = [m_1, \ldots, m_k]$. For all i $(1 \leqslant i \leqslant k)$ such that $m_i = n$ set $C_i = \Box$ and for all i such that $m_i \neq n$ but $(S[T]_n)\restriction m_i = (S\restriction m_i)[T]_n$ we set $C_i = \mathsf{term}((S\restriction m_i)[\Box]_n)$. In the latter sub-case induction hypothesis is applicable to conclude

$$\mathsf{term}\big(\big(S[T]_n\big)\restriction m_i\big) = C_i[\mathsf{term}(T), \ldots, \mathsf{term}(T)] \; .$$

Finally we set $C := f(C_1, \ldots, C_k)$ and obtain $C = \mathsf{term}(S[\Box]_n)$. In sum we have $\mathsf{term}(S[T]_n) = f\big(\big(S[T]_n\big)\restriction m_1, \ldots, \big(S[T]_n\big)\restriction m_k\big) = C[\overline{\mathsf{term}(T)}]$, where $\overline{\mathsf{term}(T)}$ denotes the sequences of terms $\mathsf{term}(T)$ of the required length $|\mathcal{P}os_S(n)|$. □

Lemma 15. *Let $\mathcal{R}$ be a TRS, l be a term and let $\sigma : \mathcal{V} \to \mathsf{NF}(\mathcal{R})$ be a substitution such that $s = l\sigma$ is argument normalised with respect to $\mathcal{R}$. If $L \in \triangle(l)$ and $S \in \diamondsuit(s)$, then there exists a morphism $m\colon L \to S$ such that for all variable nodes $n \in \mathcal{V}\mathsf{ar}(L)$ with $\mathsf{L}_L(n) = x$, $x\sigma = \mathsf{term}(S\restriction m(n))$.*

Proof. We prove the lemma by induction on l. It suffices to consider the induction step. Let $l = f(l_1, \ldots, l_k)$ and $s = f(l_1\sigma, \ldots, l_k\sigma)$. Let $\mathsf{succ}_L(\mathsf{rt}(L)) = [p_1, \ldots, p_k]$, let $\mathsf{succ}_S(\mathsf{rt}(S)) = [q_1, \ldots, q_k]$, and let $i \in \{1, \ldots, k\}$. By induction hypothesis there exist morphisms $m_i\colon L\restriction p_i \to S\restriction q_i$ of the required form. Either $m_i(n) = m_j(n)$ or $n \notin (\mathsf{dom}(m_i) \cap \mathsf{dom}(m_j))$.

Otherwise suppose $n \in (\mathsf{dom}(m_i) \cap \mathsf{dom}(m_j))$. We show $m_i(n) = m_j(n)$. Since $L \in \triangle(l)$, only variable nodes are shared, hence n needs to be a variable node. Suppose $\mathsf{L}_L(n) = x$. Then $\mathsf{term}(S\restriction m_i(n)) = x\sigma = \mathsf{term}(S\restriction m_j(n))$. As $S \in \diamondsuit(s)$ and $x\sigma \in \mathsf{NF}(\mathcal{R})$, $m_i(n) = m_j(n)$ has to hold.

Define a function $m\colon \mathsf{V}_L \to \mathsf{V}_S$ as follows. Set $m(\mathsf{rt}(L)) = \mathsf{rt}(S)$ and for $p \neq \mathsf{rt}(L)$ define $m(p) = m_i(p)$ if $p \in \mathsf{dom}(m_i)$. By the above observation, m is well-defined and a morphism. □

Lemma 16. *Let t be a term and let T be a normal form sharing term graph such that $t = \mathsf{term}(T)$. Then $t \in \mathsf{NF}(\mathcal{R})$ if and only if $T \in \mathsf{NF}(\mathcal{G}(\mathcal{R}))$.*

Let T be a term graph; a node $n \in T$ is an *$\mathcal{R}$-redex* if $\mathsf{term}(T|n) \notin \mathsf{NF}(\mathcal{R})$. We call T *redex-unsharing* if none of its $\mathcal{R}$-redexes are shared.

Lemma 17. *Suppose T is redex-unsharing. Then T is normal form sharing if and only if it is not possible to find distinct nodes $p, q \in T$ such that (i) $\mathsf{term}(T|p) \in \mathsf{NF}(\mathcal{R})$, (ii) $\mathsf{L}(p) = \mathsf{L}(q)$ and (iii) $\mathsf{succ}(p) = \mathsf{succ}(q)$.*

Proof. It suffices to consider the direction from right to left as the other is trivial. Thus suppose T is not normal form shared. We show that there exist nodes p, q fulfilling the properties stated.

We pick some node $p \in T$ with $\mathsf{term}(T|p) \in \mathsf{NF}(\mathcal{R})$ and there exists a distinct node $q \in T$ with $\mathsf{term}(T|p) = \mathsf{term}(T|q)$. For that we assume that p is $\rightharpoonup$*-minimal* in the sense that there is no node p' with $p \overset{+}{\rightharpoonup} p'$ such that p' would fulfil the above properties. The node p exists as T is not normal form shared. By definition property (i) holds and property (ii) follows from $\mathsf{term}(T|p) = \mathsf{term}(T|q)$. To show property (iii) we assume $\mathsf{succ}(p) = [p_1, \ldots, p_l]$ and $\mathsf{succ}(q) = [q_1, \ldots, q_l]$ where for at least for one $i \in \{1, \ldots, l\}$, $p_i \neq q_i$. But this contradicts the minimality of p. We conclude $\mathsf{succ}(p) = \mathsf{succ}(q)$ as desired. □

The next lemma follows from the above using a simple inductive argument.

Lemma 18. *Let S be redex-unsharing. Then for all $T \in \diamondsuit(\mathsf{term}(S))$, $S \geqslant T$.*

Theorem 19 (Adequacy). *Let $\mathcal{R}$ be a TRS and let $\mathcal{G}(\mathcal{R})$ denote the simulating GRS. Suppose s is a term and $S \in \diamondsuit(s)$. Then $s \overset{\mathsf{i}}{\to}_{\mathcal{R}} t$ if and only if $S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} \cdot \geqslant T$, where $T \in \diamondsuit(t)$.*

Proof. First we consider the direction from right to left. Suppose $S \overset{\mathsf{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R}),p} T$ such that $(L \to R) \in \mathcal{G}(\mathcal{R})$, $p \in S$, $m \colon L \to S|p$ and $T = S[m(R)]_p$. Moreover, $S|n \in \mathsf{NF}(\mathcal{G})$ for all $n \in \mathsf{succ}(p)$. Define the context $C = \mathsf{term}(S[\Box]_p)$, we write σ for the induced substitution σ_m and prove

$$\mathsf{term}(S) = C[\overline{\mathsf{term}(L)\sigma}] \overset{\mathsf{i}}{\to}_{\mathcal{R}} C[\overline{\mathsf{term}(R)\sigma}] = \mathsf{term}(T) \ .$$

Observe that $S = S[S|p]_p$. By Lemma 12, $\mathsf{term}(S \restriction p) = \mathsf{term}(L)\sigma$. Due to Lemma 14 we see

$$\mathsf{term}(S) = \mathsf{term}(S[S|p]_p) = C[\overline{\mathsf{term}(S|p)}] = C[\overline{\mathsf{term}(L)\sigma}] \ .$$

By definition $T = S[m(R)]_p = S[(m(R) \cup S)|\mathsf{rt}(m(R))]_p$. Due to Lemma 13 we have $\mathsf{term}(T) = C[\overline{\mathsf{term}(R)\sigma}]$. In order to show $\mathsf{term}(S) \overset{\mathsf{i}}{\to}_{\mathcal{R}} \mathsf{term}(T)$ it suffices to verify that $\mathsf{term}(L)\sigma = \mathsf{term}(S|p)$ is argument normalised with respect to $\mathcal{R}$. Note that $l\sigma \notin \mathsf{NF}(\mathcal{R})$ and S is a normal form sharing graph. Hence the context C contains only one hole. Let u be a proper subterm of $\mathsf{term}(S \restriction p)$. As for all $n \in \mathsf{succ}_S(p)$, $S|n \in \mathsf{NF}(\mathcal{G})$, and $S|p$ is argument normalised, $u \in \mathsf{NF}(\mathcal{R})$ follows from Lemma 16. Hence $\mathsf{term}(l)\sigma$ is argument normalised.

Finally, we prove the direction from left to right. Suppose $s = C[l\sigma] \overset{\mathsf{i}}{\to}_{\mathcal{R}} C[r\sigma] = t$ with $l \to r \in \mathcal{R}$ and $l\sigma$ argument normalised. Let $p \in S$ be the

node corresponding to $l\sigma$. As $l\sigma$ is not a normal form, p is unique. Clearly $S\restriction p \in \Diamond(l\sigma)$. Let $(L \to R) \in \mathcal{G}(\mathcal{R})$. According to Lemma 15, there exists a morphism $m\colon L \to S\restriction p$ such that $x\sigma = \mathsf{term}(S\restriction m(n))$ where $\mathsf{L}_S(n) = x \in \mathcal{V}$. As $l\sigma$ is argument normalised, $S \stackrel{\mathrm{i}}{\Rightarrow}_{\mathcal{G}(\mathcal{R})} S[m(R)]_p$ follows from Lemma 16.

It remains to prove that $T' := S[m(R)]_p \geqslant T \in \Diamond(t)$. We claim $\mathsf{term}(T') = C[r\sigma] = t$. For this, let $U := (m(R) \cup S)\restriction \mathsf{rt}(m(R))$ and observe $T' = S[m(R)]_p = S[U]_p$. Due to Lemma 13, $\mathsf{term}(U) = r\sigma$. Moreover, it is easy to see that $C = \mathsf{term}(S[\Box]_p)$. Hence by Lemma 14, $\mathsf{term}(S[U]_p) = C[\mathsf{term}(U)]$ and the claim follows. Let $T \in \Diamond(t)$. We want to apply Lemma 18 to conclude $T' \geqslant T$. For that we have to prove that T' is redex-unshared. Suppose there exists $q \in T'$ such that $u := \mathsf{term}(T'\restriction q)$ is reducible with respect to $\mathcal{R}$. We show that $\mathcal{P}os_{T'}(q)$ is a singleton set.

We distinguish two cases: Either q is reachable from $\mathsf{rt}(m(R))$ in T' or it is not reachable from $\mathsf{rt}(m(R))$. If q is reachable, then we can even conclude that $q \in m(R)$. For this, suppose there exists a variable node $n \in R$ such that $\mathsf{rt}(m(R)) \stackrel{+}{\rightarrow} m(n) \stackrel{*}{\rightarrow} q$. Then $\mathsf{term}(T'\restriction m(n)) = x\sigma$ for some variable $x \in \mathcal{V}\mathsf{ar}(r)$. But $x\sigma \in \mathsf{NF}(\mathcal{R})$ as $l\sigma$ is argument normalised. This contradicts the assumption that $\mathsf{term}(T'\restriction q)$ is reducible. Hence suppose $q \in m(R)$. By Definition 9 in L, R only variables are shared and in S only normal forms are shared. Hence q can only be shared in $m(R)$ if it represents a normal form, contrary to our assumption. Hence it is unshared. Finally consider the case that q is not reachable from $\mathsf{rt}(m(R))$ in T'. This implies that $q \in S$ and thus q is unshared in T' if q is unshared in S. Suppose $u = \mathsf{term}(S\restriction q)$, i.e., the term represented by node q is unchanged by the (graph) rewrite step. Then q is unshared as by assumption u is reducible and $S \in \Diamond(s)$. Otherwise if $u \neq \mathsf{term}(S\restriction q)$, then $q \stackrel{*}{\rightarrow} p$ for the redex $p \in S$. Hence $\mathsf{term}(S\restriction q)$ is reducible and again not shared in S. □

Sometimes it is convenient to combine graph rewrite and collapse steps into one relation. Thus we define $S \stackrel{\geqslant}{\Rightarrow}_{\mathcal{G}} T$ if and only if $S \stackrel{\mathrm{i}}{\Rightarrow}_{\mathcal{G}} \cdot \geqslant T$. Employing this notion we can rephrase the conclusion of the theorem as: $s \stackrel{\mathrm{i}}{\rightarrow}_{\mathcal{R}} t$ if and only if $S \stackrel{\geqslant}{\Rightarrow}_{\mathcal{G}} T$, whenever the conditions of the theorem are fulfilled.

4 Complexity Considerations

We now prove a polynomial relationship between the number of rewrite steps admitted by $\mathcal{R}$ and the computational complexity of the functions defined. We give semantics to $\mathcal{R}$ in the most natural way. Let $\mathcal{V}\mathsf{al}$, the set of *values*, denote the set of terms generated from constructors $\mathcal{C}$ and variables $\mathcal{V}$. Further, suppose $\mathcal{R}$ is a confluent and terminating TRS. An n-ary partial function $f :\ \mathcal{V}\mathsf{al}^n \to \mathcal{V}\mathsf{al}$ is *computable by* $\mathcal{R}$ if there exists a defined function symbol $\mathsf{f} \in \mathcal{F}$ such that for all $s_1, \ldots, s_n, t \in \mathcal{V}\mathsf{al}$:

$$\mathsf{f}(s_1, \ldots, s_n) \to^{!}_{\mathcal{R}} t \iff f(s_1, \ldots, s_n) = t\ ,$$

where f is defined. Note that this is well defined as $\mathcal{R}$ is confluent and terminating. We say that $\mathcal{R}$ *computes* f, if the function $f :\ \mathcal{V}\mathsf{al}^n \to \mathcal{V}\mathsf{al}$ is defined by

the above equation. We define the *derivation length* of a term s with respect to a terminating TRS $\mathcal{R}$ and rewrite relation $\to$: $\mathrm{dl}(s, \to) = \max\{n \mid \exists t\ s \to^n t\}$. The *(innermost) runtime complexity* (with respect to $\mathcal{R}$) is defined as follows:

$$\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}(n) = \max\{\mathrm{dl}(s, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid s = f(s_1, \ldots, s_n),\ s_1, \ldots, s_n \in \mathcal{V}\mathsf{al} \text{ and } |s| \leqslant n\}\ .$$

Suppose $\mathcal{R}$ admits polynomial runtime complexity, i.e. $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}$ is bounded polynomially. To conclude polytime computability of a function f computed by $\mathcal{R}$, we implement f using graph rewriting. More precisely, to evaluate $s := \mathsf{f}(s_1, \ldots, s_n)$ for values $s_1, \ldots, s_n$, we normalise the graph S corresponding to s under $\overset{\geqslant}{\Rightarrow}_{\mathcal{G}(\mathcal{R})}$. This is admissible by the Adequacy Theorem (Theorem 19). At most polynomially many reduction steps (in $|s|$) need to be performed. Below we show that each such intermediate step can be performed in time polynomial in the size of s.

Lemma 20. *Let $\mathcal{G}$ denote a GRS, set $c := \max\{|R| \mid (L \to R) \in \mathcal{G}\}$. If $S \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} T$ then $|T| \leqslant |S| + c$.*

Hence, sizes of intermediate graphs are polynomially related to the size of S and s. (Note that $|S| \leqslant |s|$). From this we derive that polynomial innermost runtime complexity induces polytime computability.

In the following, we fix the set of nodes $\mathsf{V} := \mathbb{N}$. We represent $S \in \mathcal{G}\mathsf{raph}$ as a pair $\langle \mathsf{rt}(S), \mathit{spec}\rangle$ where *spec* is a sequence containing for each node $n \in S$ the triple $\langle n, \mathsf{L}_S(n), \mathsf{succ}_S(n)\rangle$. Following [6], we call such triples *node specifications*. We say that a term graph S is *normalised* if $\mathsf{V}_S = \{1, \ldots, |S|\}$. Representing normalised term graphs S requires space $\mathsf{O}(\log(|S|) * |S|)$ on any reasonable model of computation: Each node $n \in S$ requires space at most $\log(|S|)$. Consequently, each node specification $\langle n, \mathsf{L}_S(n), \mathsf{succ}_S(n)\rangle$ is representable in space $\mathsf{O}(\log(|S|))$. Here we employ that for a fixed signature $\mathcal{F}$, arities are bounded by some constant, moreover each label $\mathsf{L}_S(n)$ requires just constant space. As we have to store at most $|S|$ specifications (and the root node), the assertion follows.

We define $\|S\| := \mathsf{O}(\log(|S|) * |S|)$. Below we also employ the notation $\|\cdot\|$ for the space required to represent different data-types like nodes or lists. In the following, we tacitly assume that term graphs are normalised. This is justified as normalisation introduces negligible overhead as can be seen in the next lemma.

Lemma 21. *Let S be a term graph. The function that maps S to an isomorphic and normalised term graph is computable in time $\mathsf{O}(\|S\|^2)$.*

Proof. We traverse over S and replace each encountered node $n \in S$ by the node $m(n)$, where m is a graph morphism normalising S to be constructed. To obtain m, we start the overall procedure using a counter $c := 1$ and initialise m as the morphism that is undefined everywhere. At each call $m(n)$ we check whether m is defined on n. If so, we return $m(n)$, otherwise we set $m(n) := c$, $c := c + 1$ and return $m(n)$ afterwards. When the procedure stops, m will be an injective graph morphism, and $m(S)$ a normalised term graph isomorphic to S.

While traversing S in time $\mathsf{O}(\|S\|)$, we replace the root node, and for each node specification encountered we replace at most a constant number of nodes. I.e., in total $\mathsf{O}(|S|)$ calls $m(n)$ have to be performed. We can represent m in space $\|m\| = \mathsf{O}(\|S\|)$. From this we obtain that $m(n)$ is computable in time $\mathsf{O}(\|S\|)$. Overall, the procedure finished after at most $\mathsf{O}(\|S\|^2)$ steps. □

The implementation of a graph rewrite step $S \Rightarrow_{\mathcal{G}} T$ is developed stepwise in the following. The function $\mathsf{match} : (\mathcal{G}\mathsf{raph} \times \mathcal{G}\mathsf{raph} \times \mathsf{V}) \to (\mathsf{V} \to \mathsf{V}) \cup \{\bot\}$ computes morphisms between term graphs. Here $\bot$ is a designator indicating that no morphism can be found. More precisely, for $m\colon L \to S\restriction n$ we set $\mathsf{match}(L, S, n) := m$; otherwise $\mathsf{match}(L, S, n) := \bot$. The definition of match is well-formed as $m\colon L \to S$ is unique, cf. Lemma 7. Second, we use the function $\mathsf{apply} : (\mathcal{G}\mathsf{raph} \times \mathsf{V} \times \mathcal{G}\mathsf{raph} \times (\mathsf{V} \to \mathsf{V})) \to \mathcal{G}\mathsf{raph}$ defined by $\mathsf{apply}(S, n, R, m) := S[m(R)]_n$. Here we suppose $m\colon L \to S\restriction n$ for some graph rewrite rule $L \to R$. Let Rule denote the set of (normalised) graph rewrite rules over labels $\mathcal{F} \cup \mathcal{V}$. A step $S \Rightarrow_{\mathcal{G},n,L\to R} T$ is thus computed by the function $\mathsf{step} : (\mathcal{G}\mathsf{raph} \times \mathsf{V} \times \mathsf{Rule}) \to \mathcal{G}\mathsf{raph} \cup \{\bot\}$ given by

$$\mathsf{step}(S, n, L \to R) := \mathsf{apply}(S, n, R', \mathsf{match}(L', S, n)) \quad \text{where } L' \to R' = \mathsf{rename}(S, L \to R)\ .$$

Here we suppose $\mathsf{step}(S, n, L \to R) = \bot$ if $\mathsf{match}(L', S, n) = \bot$. The function $\mathsf{rename} : (\mathcal{G}\mathsf{raph} \times \mathsf{Rule}) \to \mathsf{Rule}$ is defined such that $\mathsf{rename}(S, L \to R)$ is a renaming of the rule $L \to R$ with respect to the term graph S.

We give bounds on the computational complexity of match, rename and apply. Here we essentially translate the definition into programs and prove that those programs operate under the desired bounds, c.f. [4]. We implement $\mathsf{match}(L, S, n)$ by recursion on the term representation of L, which explains why $\mathsf{match}(L, S, n)$ operates in time exponential in $|L|$. For $\mathcal{R}$ fixed, $2^{\mathsf{O}(\|L\|)}$ is constant and thus harmless for our concerns.

Lemma 22. *Let S be a term graph and $L \to R$ a graph rewrite rule. Then*

1) $\mathsf{match}(L, S, n)$ *is computable in time* $2^{\mathsf{O}(\|L\|)} * \|S\|^2$, *and*
2) $\mathsf{rename}(S, L \to R)$ *is computable in time* $\mathsf{O}(\|S\| + \|L \to R\|)$, *and*
3) $\mathsf{apply}(S, n, R, m)$ *is computable in time* $\mathsf{O}((\|S\| + \|L \to R\|)^3)$.

Lemma 23. *Let S be a term graph, $L \to R$ a graph rewrite rule and $n \in L$ be a node. Then* $\mathsf{step}(S, n, L \to R)$ *is computable in time* $\mathsf{O}(2^{\mathsf{O}(\|L\to R\|)} * \|S\|^3)$.

Proof. Employing $\|L\| < \|L \to R\|$, the lemma follows from Lemma 22. □

Remark 24. Note that for $\mathcal{G}$ fixed and $(L \to R) \in \mathcal{G}$, the bound $\mathsf{O}(2^{\mathsf{O}(\|L\to R\|)} * \|S\|^3)$ reduces to $\mathsf{O}(\|S\|^3)$.

Let S and T be term graphs such that $S \stackrel{\geqslant}{\Rightarrow}_{\mathcal{G}} T$. To show that T is efficiently computable from S, it remains to verify that collapsing to normal form sharing graphs is feasible. We introduce the function $\mathsf{share} : \mathcal{G}\mathsf{raph} \to \mathcal{G}\mathsf{raph}$ that maps (redex-unsharing) term graphs S to their normal form sharing counterparts. To be more precise, $S \geqslant \mathsf{share}(S)$ with $\mathsf{share}(S)$ normal form sharing.

Lemma 25. *Let S be redex-unsharing and let $p \in S$ be some $\rightharpoonup$-minimal node such that there exists a distinct node $q \in S$ with (i) $\mathsf{L}(p) = \mathsf{L}(q)$ and (ii) $\mathsf{succ}(p) = \mathsf{succ}(q)$. Then $S\restriction p$ is normal form sharing.*

Proof. For a proof by contradiction, suppose $S\restriction p$ is not normal form sharing. By Lemma 17, there exist distinct nodes $p', q' \in S\restriction p$ such that labels and successor function coincide in $S\restriction p$ for p' and q'. By definition $p \overset{*}{\rightharpoonup} p'$ and $p \overset{*}{\rightharpoonup} q'$ such that in at least one case the path considered in $S\restriction p$ has non-zero length. Suppose we have $p \overset{+}{\rightharpoonup} p'$. Then the node p' is a counter example to minimality of p. □

Lemma 26. *Let S be redex-unsharing.* $\mathsf{share}(S)$ *is computable in time* $\mathsf{O}(\|S\|^4)$.

Proof. We compute $\mathsf{share}(S)$ by computing term graphs $S_1, \ldots, S_n$ with $S = S_1 > \cdots > S_n = S'$ such that for all $i = 1, \ldots, n$, the graph S_i is redex-unsharing. Lemma 18 guarantees that the above sequence is well-defined. Since $S_i > S_{i+1}$ implies $|S_i| > |S_{i+1}|$, the number of iterations is bound by $|S|$.

It suffices to show one iteration. We show that the computation of S_{i+1} from S_i, can be performed in time $\mathsf{O}(\|S\|^3)$. Including normalisation (operating in time $\mathsf{O}(\|S\|^2)$, cf. Lemma 21), we obtain that $\mathsf{share}(S)$ is computable in time $|S| * \mathsf{O}(\|S\|^3) + \mathsf{O}(\|S\|^2) \leqslant \mathsf{O}(\|S\|^4)$.

Suppose we have constructed the redex-unsharing graph S_i. To obtain S_{i+1} from S_i, we search for two distinct nodes p and q such that $\mathsf{succ}_{S_i}(p) = \mathsf{succ}_{S_i}(q)$, $\mathsf{L}_{S_i}(p) = \mathsf{L}_{S_i}(q)$ and $\mathsf{term}(S_i\restriction p) \in \mathsf{NF}(\mathcal{R})$. By Lemma 17, nodes p and q exist if and only if S_i is normal form shared. If p and q exist we obtain S_{i+1} by identifying nodes p and q in S_i. Otherwise S_i is normal form shared, we set $S_{i+1} := S_i$ and the procedure stops.

Set $R := \{p \mid \exists q.\ p \neq q \wedge \mathsf{succ}_{S_i}(p) = \mathsf{succ}_{S_i}(q) \wedge \mathsf{L}_{S_i}(p) = \mathsf{L}_{S_i}(q)\}$. Thus, reformulating the above approach, we search for $p \in R$ with $\mathsf{term}(S_i\restriction p) \in \mathsf{NF}(\mathcal{R})$. To this extend, let $R_{min} \subseteq R$ be the restriction of R to $\rightharpoonup$-minimal nodes; $R_{min} := \{p \mid p \in R \wedge \neg\exists p' \in R.\ p' \overset{+}{\rightharpoonup} p\}$. Clearly $p' \overset{+}{\rightharpoonup} p$ and $\mathsf{term}(S_i\restriction p') \in \mathsf{NF}(\mathcal{R})$ implies $\mathsf{term}(S_i\restriction p) \in \mathsf{NF}(\mathcal{R})$. It is thus sufficient to search for some $p \in R_{min}$ with $\mathsf{term}(S_i\restriction p) \in \mathsf{NF}(\mathcal{R})$, or equivalently $S_i\restriction p \in \mathsf{NF}(\mathcal{G})$. The latter observation results from Lemma 16, since by Lemma 25 the subgraph $S_i\restriction p$ is normal form sharing due to the definition of R_{min}.

Suppose $p \in R_{min}$ with $\mathsf{term}(S_i\restriction p) \in \mathsf{NF}(\mathcal{R})$. By definition of R_{min} there exists a distinct node $q \in S_i$ such that $\mathsf{term}(S_i\restriction p) = \mathsf{term}(S_i\restriction q)$. To obtain S_{i+1} from S_i, we identify p and q, i.e. we set $S_{i+1} := m(S_i)$ where the graph morphism $m\colon S_i \to S_{i+1}$ is defined by $m(q) = p$ and $m(n) = n$ for $n \in \mathsf{V}_{S_i} \setminus \{q\}$. Then S_{i+1} is redex-unsharing, moreover $S_i >_m S_{i+1}$ as desired.

One easily verifies that R is computable in time $\mathsf{O}(\|S_i\|^2) \leqslant \mathsf{O}(\|S\|^2)$. The cardinality of R is bound by $|S_i| \leqslant |S|$, as S is normalised we see that $\|R\| \leqslant \mathsf{O}(\|S\|)$. Using a quadratic reachability-algorithm to check $p' \overset{+}{\rightharpoonup} p$, R_{min} is computable from R in time $\|R\| * \mathsf{O}(\|S_i\|^2) \leqslant \mathsf{O}(\|S\|^3)$. As $R_{min} \subseteq R$ we have to check $S_i\restriction p \in \mathsf{NF}(\mathcal{G})$ at most $|S_i| \leqslant |S|$ times. For this, we check $\mathsf{match}(L, S_i, p) = \bot$ for all rules $(L \to R) \in \mathcal{G}(\mathcal{R})$. The latter can be done in time $\mathsf{O}(\|S_i\|^2) \leqslant \mathsf{O}(\|S\|^2)$ (cf. Lemma 22). We conclude that $p \in R_{min}$ with $\mathsf{term}(S_i\restriction p) \in \mathsf{NF}(\mathcal{R})$ can be

found in time $\mathsf{O}(\|S\|^2) + \mathsf{O}(\|S\|^3) + |S| * \mathsf{O}(\|S\|^2) \leqslant \mathsf{O}(\|S\|^3)$. Searching the corresponding node $q \in S_i$ and applying the morphism m can be done in time $\mathsf{O}(\|S\|)$. Overall, the runtime for computing S_{i+1} from S_i is bound by $\mathsf{O}(\|S\|^3)$ as desired. □

Theorem 27. *Let $\mathcal{R}$ be a confluent and terminating TRS, moreover suppose $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}(n) = \mathsf{O}(n^k)$ for all $n \in \mathbb{N}$ and some $k \in \mathbb{N}$. The functions computed by $\mathcal{R}$ are computable in time $\mathsf{O}(n^{5*(k+1)})$.*

Proof. To compute the functions defined by $\mathcal{R}$, we encode terms as graphs and perform graph rewriting using the simulating GRS $\mathcal{G} := \mathcal{G}(\mathcal{R})$. Let f be a function computed by $\mathcal{R}$, let $\mathsf{f} \in \mathcal{F}$ be the associated function symbol. Fix values $s_1, \ldots, s_n$ such that $f(s_1, \ldots, s_n) = t$ is defined. Let $s = \mathsf{f}(s_1, \ldots, s_n)$ and $S \in \Diamond(s)$. By definition and confluence of $\mathcal{R}$, we obtain $s \xrightarrow{\mathsf{i}}{}^{!}_{\mathcal{R}} t$. We show that $f(s_1, \ldots, s_n)$ is computable in time $|s|^{5*(k+1)}$. For this, consider some derivation

$$S = T_0 \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} \ldots \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} T_l = T \tag{$\dagger$}$$

with $T \in \mathsf{NF}(\mathcal{G})$. By Theorem 19 we conclude $\mathsf{term}(T) = t$.

We first analyse a single step $T_i \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} T_{i+1}$ from the derivation ($\dagger$). In order to compute T_{i+1} from T_i, we identify the corresponding redex in T_i. Define $\mathsf{redex}(n) := T_i\restriction n \notin \mathsf{NF}(\mathcal{G}) \wedge \forall n_j \in \mathsf{succ}_{T_i}(n).\ T_i\restriction n_j \in \mathsf{NF}(\mathcal{G})$. Then the node $n \in T_i$ is an innermost redex if and only if $\mathsf{redex}(n)$ holds. In order to check $T_i\restriction q \in \mathsf{NF}(\mathcal{G})$ for node $q \in T_i$, as before we apply the function match a constant number of times. Due to Lemma 22 we see that $\mathsf{redex}(n)$ is computable in time $\mathsf{O}(\|T_i\|^2)$ overall. We find the desired redex in T_i in time $\mathsf{O}(\|T_i\|^3)$: we traverse T_i and return the first $n \in T_i$ encountered satisfying $\mathsf{redex}(n)$. Without loss of generality, we can assume n is the redex in $T_i \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} T_{i+1}$ (otherwise, we adapt the derivation ($\dagger$) appropriately). Let $T'_{i+1} = \mathsf{step}(s, n, L \to R)$ for the first applicable rule $(L \to R) \in \mathcal{G}$. By Lemma 23, since we have to check only a constant number of rules $L \to R$, T'_{i+1} is computable from T_i in time $\mathsf{O}(\|T_i\|^3)$. As observed in the proof of Theorem 19, the graph T'_{i+1} is redex-unsharing. Using Lemma 26, we finally obtain $T_{i+1} = \mathsf{share}(T'_{i+1})$ in time $\mathsf{O}(\|T'_{i+1}\|^4)$. Summing all up, employing $\|T'_{i+1}\| \leqslant \|T_i\| + c$ for some fixed $c \in \mathbb{N}$ (cf. Lemma 20), T_{i+1} is computable from T_i in time $\mathsf{O}(\|T_i\|^4)$.

We return to the derivation ($\dagger$). Note $s \in \mathcal{B}$ and further $T_i \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} T_{i+1}$ implies $\mathsf{term}(T_i) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{term}(T_{i+1})$. Thus we conclude $l = \mathsf{O}(|s|^k)$ for some $k \in \mathbb{N}$. Let $j \in \{0, \ldots, l-1\}$. Lemma 20 implies $|T_j| \leqslant |S| + j * c \leqslant \mathsf{O}(|s|^k)$ for some fixed $c \in \mathbb{N}$. Here we employ that $|S| \leqslant |s|$. Recall $\|T_j\| = \mathsf{O}(\log(|T_j|) * |T_j|)$. Thus $\|T_j\| \leqslant \mathsf{O}(\log(|s|^k) * |s|^k) \leqslant \mathsf{O}(|s|^{k+1})$. And so each step $T_j \overset{\geqslant}{\Rightarrow}_{\mathcal{G}} T_{j+1}$ can be performed in time $\mathsf{O}(\|T_j\|^4) \leqslant \mathsf{O}(|s|^{4*(k+1)})$ using the above observation. In total, we obtain that $S \in \Diamond(s)$ can be $\overset{\geqslant}{\Rightarrow}_{\mathcal{G}}$-normalised in time $\mathsf{O}(|s|^k) * \mathsf{O}(|s|^{4*(k+1)}) \leqslant \mathsf{O}(|s|^{5*(k+1)})$. We conclude the theorem. □

5 Conclusion

Recently, many techniques have been introduced that allow the (automated) classification of the runtime complexity of TRS. In this paper we show that polynomial innermost runtime complexity of TRSs induces polytime computability of the functions defined. As a side-result we present a simulation between (innermost) term rewriting and (innermost) graph rewriting.

The latter result is related to *implicit computational complexity* and in particular to a recent result by Dal Lago and Martini. In [8] Dal Lago and Martini establish that orthogonal constructor TRSs and lambda calculus with weak call-by-value reduction simulate each other with linear overhead. The proof of this [8] (compare also [7]) provides a variant of Theorem 19 for the restricted case that the TRS in question is constructor and orthogonal. By augmenting the innermost graph rewrite relation by collapse steps, our result prevail also in the general case.

References

1. M. Avanzini and G. Moser. Complexity Analysis by Rewriting. In *Proc. 9th FLOPS*, volume 4989 of *LNCS*, pages 130–146, 2008.
2. M. Avanzini and G. Moser. Complexity Analysis by Rewriting. Technical report, Computational Logic, November 2008. Available under `http://cl-informatik.uibk.ac.at/~zini/publications/FLOPS08_techreport.pdf`.
3. M. Avanzini and G. Moser. Dependency Pairs and Polynomial Path Orders. In *Proc. 20th RTA*, volume 5595, pages 48–62. Springer Verlag, 2009.
4. M. Avanzini and G. Moser. Complexity Analysis by Graph Rewriting. Technical report, Computational Logic, Dezember 2010. Available under `http://cl-informatik.uibk.ac.at/~zini/publications/FLOPS10_techreport.pdf`.
5. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
6. H. Barendregt, M. v. Eekelen, J. Glauert, R. Kennaway, M. Plasmeijer, and R. Sleep. Towards an Intermediate Language based on Graph Rewriting. In *Proc. PARLE (2)*, volume 259 of *LNCS*, pages 159–175. Springer Verlag, 1987.
7. U. Dal Lago and S. Martini. Derivational complexity is an invariant cost model. In *Proc. 1st FOPARA*, 2009.
8. U. Dal Lago and S. Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In *Proc. 36th ICALP*, volume 5556 of *LNCS*, pages 163–174. Springer Verlag, 2009.
9. N. Dershowitz. 33 Examples of Termination. In *French Spring School of Theoretical Computer Science*, volume 909 of *LNCS*, pages 16–26, 1995.
10. N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In *Proc. 4th IJCAR*, volume 5195 of *LNCS*, pages 364–380, 2008.
11. J.-Y. Marion. Analysing the Implicit Complexity of Programs. *IC*, 183:2–18, 2003.
12. D. Plump. Essentials of Term Graph Rewriting. *ENTCS*, 51, 2001.
13. TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracks in Theoretical Computer Science*. Cambridge University Press, 2003.