Recently, many techniques have been introduced that allow the (automated) classification of the runtime complexity of term rewrite systems (TRSs for short). In earlier work, the authors have shown that for confluent TRSs, innermost polynomial runtime complexity induces polytime computability of the functions defined.

In this paper, we generalise the above result to full rewriting. Following our previous work, we exploit graph rewriting. We give a new proof of the adequacy of graph rewriting for full rewriting that allows for a precise control of the resources copied. In sum we completely describe an implementation of rewriting on a Turing machine (TM for short). We show that the runtime complexity of the TRS and the runtime complexity of the TM is polynomially related. Our result strengthens the evidence that the complexity of a rewrite system is truthfully represented through the length of derivations. Moreover our result allows the classification of non-deterministic polytime-computation based on runtime complexity analysis of rewrite systems.

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

1. Introduction ........................................... 3
2. Preliminaries ........................................... 4
3. Term Graph Rewriting ................................. 5
4. Adequacy of Graph Rewriting for Term Rewriting ................................. 8
5. Implementing Term Rewriting Efficiently .................. 14
6. Discussion ............................................. 23
1 Introduction

Recently we see increased interest into studies of the (maximal) derivation length of term rewrite system, compare for example [8, 7, 12, 9, 11]. We are interested in techniques to automatically classify the complexity of term rewrite systems (TRS for short) and have introduced the polynomial path order \( \text{POP}^* \) and extensions of it, cf. [1, 2]. \( \text{POP}^* \) is a restriction of the multiset path order [15] and whenever compatibility of a TRS \( \mathcal{R} \) with \( \text{POP}^* \) 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, where the initial terms are restricted argument normalised terms (aka basic terms).

We have successfully implemented this technique. As a consequence we can automatically verify for a given TRS \( \mathcal{R} \) that it admits at most polynomial runtime complexity. In this paper we study the question, whether such results are restricted to runtime complexity or can be applied also for the (automated) classification of the intrinsic computational complexity of the functions computed by the given TRS \( \mathcal{R} \). For motivation consider the TRS given in the next example. It is not difficult to see that \( \mathcal{R}_{\text{sat}} \) encodes the function problem \( \text{FSAT} \) associated to the well-known satisfiability problem \( \text{SAT} \).

Example 1.1. Consider the following TRS \( \mathcal{R}_{\text{sat}} \):

\[
\begin{align*}
1: & \text{if}(\text{tt}, t, e) \rightarrow t \\
2: & \text{if}(\text{ff}, t, e) \rightarrow e \\
3: & \text{choice}(x : xs) \rightarrow x \\
4: & \text{choice}(x : xs) \rightarrow \text{choice}(xs) \\
5: & \text{guess}(\text{nil}) \rightarrow \text{nil} \\
6: & \text{guess}(c : cs) \rightarrow \text{choice}(c) : \text{guess}(cs) \\
7: & \text{in}(x, \text{nil}) \rightarrow \text{ff} \\
8: & \text{in}(x, y : ys) \rightarrow \text{if}(x = y, \text{tt}, \text{in}(x, ys)) \\
9: & \neg 1(x) \rightarrow 0(x) \\
10: & 0(x) \rightarrow 1(x) \\
11: & \varepsilon = \varepsilon \rightarrow \text{tt} \\
12: & 1(x) = 1(y) \rightarrow x = y \\
13: & 1(x) = 0(y) \rightarrow \text{ff} \\
14: & 0(x) = 1(y) \rightarrow \text{ff} \\
15: & 0(x) = 0(y) \rightarrow x = y \\
16: & \text{verify}(\text{nil}) \rightarrow \text{tt} \\
17: & \text{verify}(l : ls) \rightarrow \text{if}(\text{in}(\neg l, ls), \text{ff}, \text{verify}(ls)) \\
18: & \text{sat}'(c, a) \rightarrow \text{if}(\text{verify}(a), a, \text{unsat}) \\
19: & \text{sat}(a) \rightarrow \text{sat}'(c, \text{guess}(c)) \\
\end{align*}
\]

These rules are compatible with \( \text{POP}^* \) and as a result we conclude that the innermost runtime complexity induced is polynomially.

FSAT is complete for the class of function problems over NP (FNP for short), compare [13]. This leads to the question, whether a characterisation of the runtime complexity of \( \mathcal{R}_{\text{sat}} \) suffices to conclude that the function computed by \( \mathcal{R}_{\text{sat}} \) belongs to the class FNP. The purpose of this paper is to provide a positive answer to this question. More precisely, we establish the following results:

Our implementation forms part of the Tyrolean Complexity Tool (\( \text{TCT} \) for short). For further information, see http://cl-informatik.uibk.ac.at/software/tct/

To our best knowledge \( \text{TCT} \) is currently the only complexity tool that can provide a complexity certificate for the TRS \( \mathcal{R}_{\text{sat}} \), compare http://termcomp.uibk.ac.at

\[1\]

\[2\]
We re-consider graph rewriting and provide a new proof of the adequacy of graph rewriting for full rewriting. This overcomes obvious inefficiencies of rewriting, when it comes to the duplication of results.

We provide a precise analysis of the resources needed in implementing graph rewriting on a Turing machine (TM for short).

Combining these results we obtain an efficient implementation of rewriting on a TM. Based on this implementation our main result on the correspondence between polynomial runtime complexity and polytime computability follows.

Our result strengthens the evidence that the complexity of a rewrite system is truthfully represented through the length of derivations. Moreover our result allows the classification of nondeterministic polytime-computation based on runtime complexity analysis of rewrite systems. This extends previous work (see [3]) that shows that for confluent TRSs, innermost polynomial runtime complexity induces polytime computability of the functions defined. Moreover it extends related work by Dal Lago and Martini (see [6, 5]) that studies the complexity of orthogonal TRSs, also applying graph rewriting techniques.

The paper is structured as follows. In Section 2 we present basic notions, in Section 3 we (briefly) recall the central concepts of graph rewriting. The adequacy theorem is provided in Section 4 and in Section 5 we show how rewriting can be implemented efficiently. Finally we discuss our results in Section 6, where the above application to computational complexity is made precise.

2 Preliminaries

We assume familiarity with the basics of term rewriting, see [4, 15]. No familiarity with graph rewriting (see [15]) is assumed. Let $R$ be a binary relation on a set $S$. We write $R^+$ for the transitive and $R^*$ for the transitive and reflexive closure of $R$. An element $a \in S$ is $R$-minimal if there exists no $b \in S$ such that $a R b$.

Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature, containing at least one constant. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted as $\mathcal{T}$. The size $|t|$ of a term $t$ is defined as usual. A term rewrite system (TRS for short) $\mathcal{R}$ over $\mathcal{T}$ is a finite set of rewrite rules $l \rightarrow r$, such that $l \notin \mathcal{V}$ and $\text{Var}(l) \supseteq \text{Var}(r)$. We write $\rightarrow_{\mathcal{R}}$ for the induced rewrite relation. The set of defined function symbols is denoted as $\mathcal{D}$, while the constructor symbols are collected in $\mathcal{C}$, clearly $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$. We use $\text{NF}(\mathcal{R})$ to denote the set of normal forms of $\mathcal{R}$ and write $s \rightarrow_{\mathcal{R}}^1 t$ if $s \rightarrow_{\mathcal{R}}^* t$ and $t \in \text{NF}(\mathcal{R})$. We define the set of values $\text{Val} := \mathcal{T}(\mathcal{C}, \mathcal{V})$, and we define $\mathcal{B} := \{ f(v_1, \ldots, v_n) \mid f \in \mathcal{D} \text{ and } v_i \in \text{Val} \}$ as the set of basic terms. Let $\square$ be a fresh constant. Terms over $\mathcal{F} \cup \{ \square \}$ and $\mathcal{V}$ are called contexts. The empty context is denoted as $\square$. 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 $t_1, \ldots, t_n$. 

A TRS is called confluent if for all $s, t_1, t_2 \in T$ with $s \rightarrow^*_R t_1$ and $s \rightarrow^*_R t_2$ there exists a term $t_3$ such that $t_1 \rightarrow^*_R t_3$ and $t_2 \rightarrow^*_R t_3$. The derivation length of a terminating term $s$ with respect to $\rightarrow$ is defined as $\text{dl}(s, \rightarrow) := \max\{n \mid \exists t. s \rightarrow^n t\}$, where $\rightarrow^n$ denotes the $n$-fold application of $\rightarrow$. The \textit{runtime complexity function} $\text{rc}_R$ with respect to a TRS $R$ is defined as $\text{rc}_R(n) := \max\{\text{dl}(t, \rightarrow_R) \mid t \in B \text{ and } |t| \leq n\}$.

### 3 Term Graph Rewriting

In the sequel we introduce the central concepts of term graph rewriting or graph rewriting for short. We closely follow the presentation of [3], for further motivation of the presented notions we kindly refer the reader to [3]. Let $R$ be a TRS over a signature $\mathcal{F}$. We keep $R$ and $\mathcal{F}$ fixed for the remainder of this paper.

A graph $G = (V_G, \text{Succ}_G, L_G)$ over the set $L$ of labels is a structure such that $V_G$ is a finite set, the nodes or vertices, $\text{Succ} : V_G \rightarrow V^*_G$ is a mapping that associates a node $u$ with an (ordered) sequence of nodes, called the successors of $u$. Note that the sequence of successors of $u$ may be empty: $\text{Succ}_G(u) = \emptyset$. Finally $L_G : V_G \rightarrow L$ is a mapping that associates each node $u$ with its label $L_G(u)$. Typically the set of labels $L$ is clear from context and not explicitly mentioned. In the following, nodes are denoted by $u, v, \ldots$ possibly followed by subscripts. We drop the reference to the graph $G$ from $V_G$, $\text{Succ}_G$, and $L_G$, i.e., we write $G = (V, \text{Succ}, L)$ if no confusion can arise from this. Further, we also write $u \in G$ instead of $u \in V$.

Let $G = (V, \text{Succ}, L)$ be a graph and let $u \in G$. Consider $\text{Succ}(u) = [u_1, \ldots, u_k]$. We call $u_i$ $(1 \leq i \leq k)$ the $i$-th successor of $u$ (denoted as $u \xleftarrow{i} u_i$). If $u \xleftarrow{i} v$ for some $i$, then we simply write $u \xrightarrow{i} v$. A node $v$ is called reachable from $u$ if $u \xrightarrow{\ast} v$, where $\ast$ denotes the reflexive and transitive closure of $\rightarrow$. We write $\xrightarrow{\ast}$ for $\rightarrow \cdot \xleftarrow{\ast}$. A graph $G$ is acyclic if $u \xrightarrow{\ast} v$ implies $u \neq v$ and $G$ is rooted if there exists a unique node $u$ such that every other node in $G$ is reachable from $u$. The node $u$ is called the root $\text{rt}(G)$ of $G$. The size of $G$, i.e., the number of nodes, is denoted as $|G|$. The depth of $S$, i.e., the length of the longest path in $S$, is denoted as $\text{dp}(S)$. We write $G \upharpoonright u$ for the subgraph of $G$ reachable from $u$.

Let $G$ and $H$ be two term graphs, possibly sharing nodes. We say that $G$ and $H$ are properly sharing if $u \in G \cap H$ implies $L_G(u) = L_H(u)$ and $\text{Succ}_G(u) = \text{Succ}_H(u)$. If $G$ and $H$ are properly sharing, we write $G \cup H$ for their union.

**Definition 3.1.** A term graph (with respect to $\mathcal{F}$ and $\mathcal{V}$) is an acyclic and rooted graph $S = (V, \text{Succ}, L)$ over labels $\mathcal{F} \cup \mathcal{V}$. Let $u \in S$ and suppose $L(u) = f \in \mathcal{F}$ such that $f$ is $k$-ary. Then $\text{Succ}(u) = [u_1, \ldots, u_k]$. On the other hand, if $L(u) \in \mathcal{V}$ then $\text{Succ}(u) = \emptyset$. We demand that any variable node is shared. That is, for $u \in S$ with $L(u) \in \mathcal{V}$, if $L(u) = L(v)$ for some $v \in \mathcal{V}$ then $u = v$.

Below $S, T, \ldots$ and $L, R$, possibly followed by subscripts, always denote term graphs. We write $\text{Graph}$ for the set of all term graphs with respect to $\mathcal{F}$ and $\mathcal{V}$. Abusing notation from rewriting we set $\text{Var}(S) := \{u \mid u \in S, L(u) \in \mathcal{V}\}$, the set of variable nodes in $S$. We define the term $\text{term}(S)$ represented by $S$ as follows: $\text{term}(S) := x$ if $L(\text{rt}(S)) = \mathcal{V}$.
Proof. By a straightforward inductive argument.

Let \( x \in \mathcal{V} \) and \( \text{term}(S) := f(\text{term}(S \mid u_1), \ldots, \text{term}(S \mid u_k)) \) for \( L(\text{rt}(S)) = f \in \mathcal{F} \) and \( \text{Succ}(\text{rt}(S)) = [u_1, \ldots, u_k] \).

We adapt the notion of \textit{positions} in terms to positions in graphs in the obvious way. Positions are denoted as \( p, q, \ldots \), possibly followed by subscripts. For positions \( p \) and \( q \) we write \( pq \) for their concatenation. We write \( p \subseteq q \) if \( p \) is a prefix of \( q \), i.e., \( q = pp' \) for some position \( p' \). The size \( |p| \) of position \( p \) is defined as its length. Let \( u \in S \) be a node. The set of \textit{positions} \( \mathcal{P} \text{os}_S(u) \) of \( u \) is defined as \( \mathcal{P} \text{os}_S(u) := \{ \varepsilon \} \) if \( u = \text{rt}(S) \) and \( \mathcal{P} \text{os}_S(u) := \{ i_1 \cdots i_k \mid \text{rt}(S) \xrightarrow{L_i} \cdots \xrightarrow{L_k} u \} \) otherwise. The set of all positions in \( S \) is \( \mathcal{P} \text{os}_S := \bigcup_{u \in S} \mathcal{P} \text{os}_S(u) \). Note that \( \mathcal{P} \text{os}_S \) coincides with the set of positions of \( \text{term}(S) \).

If \( p \in \mathcal{P} \text{os}_S(u) \) we say that \( u \text{ corresponds to } p \). In this case we also write \( S \mid p \) for the subgraph \( S \mid u \). This is well defined since exactly one node corresponds to a position \( p \).

One verifies \( \text{term}(S \mid p) = \text{term}(S) \}_{p} \) for all \( p \in \mathcal{P} \text{os}_S \). We say that \( u \) is (strictly) \textit{above} a position \( p \) if \( u \) corresponds to a position \( q \) with \( q \subseteq p \) \((q < p)\). Conversely, the node \( u \) is \textit{below} \( p \) if \( u \) corresponds to \( q \) with \( p \subseteq q \).

By exploiting different degrees of \textit{sharing}, a term \( t \) can often be represented by more than one term graph. Let \( S \) be a term graph and let \( u \in S \) be a node. We say that \( u \) is \textit{shared} if the set of positions \( \mathcal{P} \text{os}_S(u) \) is not singleton. Note that in this case, the node \( u \) represents more than one subterm of \( \text{term}(S) \). If \( \mathcal{P} \text{os}_S(u) \) is singleton, then \( u \) is \textit{unshared}. The node \( u \) is \textit{minimally shared} if it is a variable node or unshared otherwise (recall that variable nodes are always shared). We say \( u \) is \textit{maximally shared} if \( \text{term}(S \mid u) = \text{term}(S \mid v) \) implies \( u = v \). The term graph \( S \) is called \textit{minimally sharing} (maximally sharing) if all nodes \( u \in S \) are minimally shared (maximally shared). Let \( s \) be a term. We collect all minimally sharing term graphs representing \( s \) in the set \( \Delta(s) \). Maximally sharing term graphs representing \( s \) are collected in \( \nabla(s) \).

We now introduce a notion for replacing a subgraph \( S \mid u \) of \( S \) by a graph \( H \).

Definition 3.2. Let \( S \) be a term graph and let \( u, v \in S \) be two nodes. Then \( S[u \leftarrow v] \) denotes the redirection of node \( u \) to \( v \): set \( r(u) := v \) and \( r(w) := w \) for all \( w \in S \setminus \{ u \} \). Set \( V' := (V_S \cup \{ v \}) \setminus \{ u \} \) and for all \( w \in V' \), \( \text{Succ}'(w) := r^*(\text{Succ}_S(w)) \) where \( r^* \) is the extension of \( r \) to sequences. Finally, set \( S[u \leftarrow v] := (V', \text{Succ}', L_S) \).

Let \( H \) be a rooted graph over \( \mathcal{F} \cup \mathcal{V} \). We define \( S[H]_u := (S[u \leftarrow \text{rt}(H)] \cup H) \mid v \) where \( v = \text{rt}(H) \) if \( u = \text{rt}(S) \) and \( v = \text{rt}(S) \) otherwise. Note that \( S[H]_u \) is again a term graph if \( u \not\in H \) and \( H \) acyclic.

The following notion of \textit{term graph morphism} plays the role of substitutions.

Definition 3.3. Let \( L \) and \( T \) be two term graphs. A morphism from \( L \) to \( T \) (denoted \( m: L \to T \)) is a function \( m: V_L \to V_T \) such that \( m(\text{rt}(L)) = \text{rt}(T) \), and for all \( u \in L \) with \( L_L(u) \in \mathcal{F} \), (i) \( L_L(u) = L_T(m(u)) \) and (ii) \( m^*(\text{Succ}_L(u)) = \text{Succ}_T(m(u)) \).

The next lemma follows essentially from assertion (ii) of Definition 3.3.

Lemma 3.4. If \( m: L \to S \) then for any \( u \in S \) we have \( m: L \mid u \to S \mid m(u) \).

\textit{Proof.} By a straightforward inductive argument. \( \square \)
Let $m : L \to S$ be a morphism from $L$ to $S$. The induced substitution $\sigma_m : \text{Var}(L) \to T$ is defined as $\sigma_m(x) := \text{term}(S \upharpoonright m(u))$ for any $u \in S$ such that $L(u) = x \in \mathcal{V}$. As an easy consequence of Lemma 3.4 we obtain the following.

**Lemma 3.5.** Let $L$ and $S$ be term graphs, and suppose $m : L \to S$ for some morphism $m$. Let $\sigma_m$ be the substitution induced by $m$. Then $\text{term}(L)\sigma_m = \text{term}(S)$.

**Proof.** The lemma has also been shown in [3, Lemma 14]. For completeness we restate the proof.

We prove that for each node $u \in L$, $\text{term}(L \upharpoonright u)\sigma_m = \text{term}(S \upharpoonright m(u))$ by induction on $l := \text{term}(L \upharpoonright u)$. To conclude $\text{term}(L)\sigma_m = \text{term}(S)$, it suffices to observe that by definition of $m$: $m(\text{rt}(L)) = \text{rt}(S)$.

If $l \in \mathcal{V}$, then by definition $L \upharpoonright u$ consists of a single (variable-)node and $\text{term}(L \upharpoonright u)\sigma_m = \text{term}(S \upharpoonright m(u))$ follows by the definition of the induced substitution $\sigma_m$. If $l = f(l_1, \ldots, l_k)$, then we have $\text{Succ}_L(u) = [u_1, \ldots, u_k]$ for some $u_1, \ldots, u_k \in L$. As $m : L \to S$ holds, Lemma 3.4 yields $m : L \upharpoonright u_i \to S \upharpoonright m(u_i)$ for all $i = 1, \ldots, k$. And induction hypothesis becomes applicable so that $l_i\sigma_m = \text{term}(S \upharpoonright m(u_i))$. Thus

$$l\sigma_m = f(l_1\sigma_m, \ldots, l_k\sigma_m) = f(\text{term}(S \upharpoonright m(u_1)), \ldots, \text{term}(S \upharpoonright m(u_k))).$$

By definition of $m$, $L_S(m(u)) = L_L(u) = f$ and $\text{Succ}_S(m(u)) = m^*(\text{Succ}_L(u)) = [m(u_1), \ldots, m(u_k)]$. Hence $f(\text{term}(S \upharpoonright m(u_1)), \ldots, \text{term}(S \upharpoonright m(u_k))) = \text{term}(S \upharpoonright m(u))$.

We write $S \geq_m T$ (or $S \geq T$ for short) if $m : S \to T$ is a morphism such that for all $u \in V_S$, Property (i) and Property (ii) in Definition 3.3 are fulfilled. For this case, $S$ and $T$ represent the same term. We write $S >_m T$ (or $S > T$ for short) when the graph morphism $m$ is additionally non-injective. If both $S \geq T$ and $T \geq S$ holds then $S$ and $T$ are isomorphic, in notation $S \cong T$. Recall that $|S|$ denotes the number of nodes in $S$.

**Lemma 3.6.** For all term graph $S$ and $T$, $S \geq_m T$ implies $\text{term}(S) = \text{term}(T)$ and $|S| \geq |T|$. If further $S >_m T$ holds then $|S| > |T|$.

**Proof.** Suppose $S \geq_m T$. We first prove $\text{term}(S) = \text{term}(T)$. We prove the lemma by induction on $S$. For the base case, suppose $S$ consists of a single node $u$ such that $L_S(u) = x \in \mathcal{V}$. As the node $u$ is the root of $S$, definition of $m$ yields that $T$ consists of a single (variable)-node labeled with $x$. (Note that $m$ is a surjective morphism.) Thus the result follows trivially. For the inductive step, suppose $L_S = f \in \mathcal{F}$ and $\text{Succ}_S = [u_1, \ldots, u_k]$. From $S \geq_m T$ we see that $L_T = f$ and $\text{Succ}_T = [m(u_1), \ldots, m(u_k)]$. As a consequence of Lemma 3.4 $S \upharpoonright u_i \geq_m T \upharpoonright m(u_i)$ and thus by induction hypothesis $\text{term}(S \upharpoonright u_i) = \text{term}(T \upharpoonright m(u_i))$ as desired.

Now observe that $m$ is by definition surjective. Thus $|S| \geq |T|$ trivially follows. Further, it is not hard to see that if $m$ is additionally non-injective, i.e., $S >_m T$, then clearly $|S| > |T|$.

Let $L$ and $R$ be two properly sharing term graphs. Suppose $\text{rt}(L) \notin \text{Var}(L)$, $\text{Var}(R) \subseteq \text{Var}(L)$ and $\text{rt}(L) \notin R$. Then the graph $L \cup R$ is called a graph rewrite rule (rule for
short), denoted by \( L \rightarrow R \). The graph \( L, R \) denotes the left-hand, right-hand side of \( L \rightarrow R \) respectively. A graph rewrite system (GRS for short) \( G \) is a set of graph rewrite rules.

Let \( G \) be a GRS, let \( S \in \text{Graph} \) and let \( L \rightarrow R \) be a rule. A rule \( L' \rightarrow R' \) is called a renaming of \( L \rightarrow R \) with respect to \( S \) if \( (L' \rightarrow R') \cong (L \rightarrow R) \) and \( V_S \cap V_{L' \rightarrow R'} = \emptyset \). Let \( L' \rightarrow R' \) be a renaming of a rule \( (L \rightarrow R) \in G \) for \( S \), and let \( u \in S \) be a node. We say \( S \) rewrites to \( T \) at redex \( u \) with rule \( L \rightarrow R \), denoted as \( S \rightarrow_{G,u,L \rightarrow R} T \), if there exists a morphism \( m: L' \rightarrow S \upharpoonright u \) and \( T = S[m(R')\upharpoonright u] \). Here \( m(R') \) denotes the structure obtained by replacing in \( R' \) every node \( v \in \text{dom}(m) \) by \( m(v) \in S \), where the labels of \( m(v) \in m(R') \) are the labels of \( m(v) \in S \). We also write \( S \rightarrow_{G,u,L \rightarrow R} T \) if \( S \rightarrow_{G,u,L \rightarrow R} T \) for the position \( p \) corresponding to \( u \) in \( S \). We set \( S \rightarrow_{G} T \) if \( S \rightarrow_{G,u,L \rightarrow R} T \) for some \( u \in S \) and \( (L \rightarrow R) \in G \). The relation \( \rightarrow_{G} \) is called the graph rewrite relation induced by \( G \). Again abusing notation, we denote the set of normal-forms with respect to \( \rightarrow_{G} \) as \( \text{NF}(G) \).

### 4 Adequacy of Graph Rewriting for Term Rewriting

In earlier work \cite{3} we have shown that graph rewriting is adequate for innermost rewriting without further restrictions on the studied TRS \( R \). In this section we generalise this result to full rewriting. The here presented adequacy theorem (see Theorem \[4.15\]) is not essentially new. Related results can be found in the extensive literature, see for example \cite{15}. In particular, in \cite{14} the adequacy theorem is stated for full rewriting and unrestricted TRSs. In this work, we take a fresh look from a complexity related point of view. We give a new proof of the adequacy of graph rewriting for full rewriting that allows for a precise control of the resources copied. This is essential for the accurate characterisation of the implementation of graph rewriting given in Section \[5\].

**Definition 4.1.** The simulating graph rewrite system \( G(R) \) of \( R \) contains for each rule \( (l \rightarrow r) \in R \) some rule \( L \rightarrow R \) such that \( L \in \triangle(l) \), \( R \in \triangle(r) \) and \( V_L \cap V_R = \text{Var}(R) \).

The next two Lemmas establish soundness in the sense that derivations with respect to \( G(R) \) correspond to \( R \)-derivations.

**Lemma 4.2.** Let \( S \) be a term graph and let \( L \rightarrow R \) be renaming of a graph rewrite rule for \( S \), i.e., \( S \cap R = \emptyset \). Suppose \( m: L \rightarrow S \) for some morphism \( m \) and let \( \sigma_m \) be the substitution induced by \( m \). Then \( \text{term}(R)\sigma_m = \text{term}(T) \) where \( T := (m(R) \cup S) \upharpoonright \text{rt}(m(R)) \).

*Proof.* We prove the more general statement that for each \( u \in R \), \( \text{term}(R \upharpoonright u)\sigma_m = \text{term}(T \upharpoonright m(u)) \), c.f. also \cite{3} Lemma 15]. First suppose \( u \in R \cap L \). Then \( R \upharpoonright u = L \upharpoonright u \) as \( L, R \) are properly shared. Employing Lemma 3.5 we have \( \text{term}(R \upharpoonright u)\sigma_m = \text{term}(L \upharpoonright u)\sigma_m = \text{term}(S \upharpoonright m(u)) \). From this the assertion follows.

Thus suppose \( u \in R \setminus L \). This subcase we prove by induction on \( r := \text{term}(R \upharpoonright u) \). The base case \( r \in V \) is trivial, as variables are shared in \( L \rightarrow R \). For the inductive step, let \( r = f(r_1, \ldots, r_k) \) with \( \text{Succ}_R(u) = \{u_1, \ldots, u_k\} \). We identify \( m \) with the extension
of $m$ to all nodes in $R$. The induction hypothesis yields $r_i \sigma_m = \text{term}(T \mid m(u_i))$ for $i = 1, \ldots, k$. By definition of $m(R)$: $m(u) = u \in m(R) \subseteq T$. Hence $\text{Succ}_T(m(u)) = \text{Succ}_m(m(R)(u)) = m^*(\text{Succ}_m(u)) = \{m(u_1), \ldots, m(u_k)\}$. Moreover $L_T(u) = L_m(R)(u) = f$ by definition. We conclude $r_i \sigma_m = f(r_1 \sigma_m, \ldots, r_k \sigma_m) = f(\text{term}(T \mid m(u_1)), \ldots, \text{term}(T \mid m(u_k))) = \text{term}(T \mid m(u))$. 

In Section 2 we introduced $\Box$ as designation of the empty context. Below we write $\Box$ for the unique (up-to isomorphism) graph representing the constant $\Box$.

**Lemma 4.3.** Let $S$ and $T$ be two be properly sharing term graphs, let $u \in S \setminus T$ and $C = \text{term}(S[\Box]_u)$. Then $\text{term}(S[T]_u) = C[\text{term}(T), \ldots, \text{term}(T)]$.

**Proof.** We proceed by induction on the size of $S$, compare also [3, Lemma 16]. In the base case $S$ consists of a single node $u$. Hence the context $C$ is empty and the lemma follows trivially.

For the induction step we can assume without loss of generality that $u \neq \text{rt}(S)$. We assume $L_S(\text{rt}(S)) = f \in F$ and $\text{Succ}_S(\text{rt}(S)) = \{v_1, \ldots, v_k\}$. For all $i \ (1 \leq i \leq k)$ such that $v_i = u$ set $C_i = \Box$ and for all $i$ such that $v_i \neq u$ but $(S[T]_u) \mid v_i = (S \mid v_i)[T]_u$ we set $C_i = \text{term}((S \mid v_i)[\Box]_u)$. In the latter sub-case induction hypothesis is applicable to conclude

$$\text{term}((S[T]_u) \mid v_i) = C_i[\text{term}(T), \ldots, \text{term}(T)].$$

Finally we set $C := f(C_1, \ldots, C_k)$ and obtain $C = \text{term}(S[\Box]_u)$. In sum we have $\text{term}(S[T]_u) = f((S[T]_u) \mid v_1, \ldots, (S[T]_u) \mid v_k) = C[\text{term}(T)]$, where $\text{term}(T)$ denotes the sequences of terms $\text{term}(T)$.

For non-left-linear TRSs $\mathcal{R}$, $\xrightarrow{\mathcal{G}(\mathcal{R})}$ does not suffice to mimic $\xrightarrow{\mathcal{R}}$. This is clarified in the following example.

**Example 4.4.** Consider the TRS $\mathcal{R}_f := \{f(x) \to \text{eq}(x, a); \text{eq}(x, x) \to \top\}$. Then $\mathcal{R}_f$ admits the derivation

$$f(a) \xrightarrow{\mathcal{R}_f} \text{eq}(a, a) \xrightarrow{\mathcal{R}_f} \top$$

but $\mathcal{G}(\mathcal{R}_f)$ cannot completely simulate the above sequence:

$$f \xrightarrow{\mathcal{G}(\mathcal{R}_f)} \text{eq} \xrightarrow{\mathcal{G}(\mathcal{R}_f)} \in \text{NF}(\mathcal{G}(\mathcal{R}_f))$$

Let $L \xrightarrow{R}$ be the rule in $\mathcal{G}(\mathcal{R}_f)$ corresponding to $\text{eq}(x, x) \xrightarrow{\top}$, and let $S$, term$(S) = \text{eq}(a, a)$, be the second graph in the above sequence. Then $L \xrightarrow{R}$ is inapplicable as we cannot simultaneously map the unique variable node in $L$ to both leaves in $S$ via a graph morphism. Note that the situation can be repaired by sharing the two arguments in $S$.

For maximally sharing graphs $S$ we can prove that redexes of $\mathcal{R}$ and (positions corresponding to) redexes of $\mathcal{G}(\mathcal{R})$ coincide. This is a consequence of the following Lemma.
Lemma 4.5. Let $l$ be a term and $s = l\sigma$ for some substitution $\sigma$. If $L \in \triangle(l)$ and $S \in \triangledown(s)$, then there exists a morphism $m: L \to S$. Further, $\sigma(x) = \sigma_m(x)$ for the induces substitution $\sigma_m$ and all variables $x \in \text{Var}(l)$.

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 $s = f(l_1\sigma, \ldots, l_k\sigma)$. Suppose $\text{Suc}_{L}(rt(L)) = [u_1, \ldots, u_k]$ and $\text{Suc}_{S}(rt(S)) = [v_1, \ldots, v_k]$. By induction hypothesis there exist morphisms $m_i: L \upharpoonright u_i \to S \upharpoonright v_i$ ($1 \leq i \leq k$) of the required form. Define $m: V_L \to V_S$ as follows. Set $m(rt(L)) = rt(S)$ and for $w \neq rt(L)$ define $m(w) = m_i(w)$ if $w \in \text{dom}(m_i)$. We claim $w \in (\text{dom}(m_i) \cap \text{dom}(m_j))$ implies $m_i(w) = m_j(w)$. For this, suppose $w \in (\text{dom}(m_i) \cap \text{dom}(m_j))$. Since $L \in \triangle(l)$, only variable nodes are shared, hence $w$ needs to be a variable node, say $L_L(w) = x \in V$. Then

\[\text{term}(S \upharpoonright m_i(w)) = \sigma_{m_i}(x) = \sigma(x) = \sigma_{m_j}(x) = \text{term}(S \upharpoonright m_j(w))\]

by induction hypothesis. As $S \in \triangledown(s)$ is maximally shared, $m_i(w) = m_j(w)$ follows. We conclude $m$ is a well-defined morphism, further $m: L \to S$.

A second problem is introduced by non-eager evaluation. Consider the following.

Example 4.6. Let $R_{\text{dup}} := \{\text{dup}(x) \rightarrow c(x, x); a \rightarrow b\}$. Then $R_{\text{dup}}$ admits the derivation

\[\text{dup}(a) \rightarrow_{R_{\text{dup}}} c(a, a) \rightarrow_{R_{\text{dup}}} c(b, a)\]

but applying the corresponding rules in $G(R_{\text{dup}})$ yields:

\[
\begin{array}{c}
\text{dup} \\
\text{a}
\end{array}
\rightarrow_{G(R_{\text{dup}})}
\begin{array}{c}
\{c\} \\
\text{a}
\end{array}
\rightarrow_{G(R_{\text{dup}})}
\begin{array}{c}
\{\} \\
\text{b}
\end{array}
\]

Application of the first rule produces a shared redex. Consequently the second step amounts to a parallel step in $R_{\text{dup}}$.

To prove adequacy of graph rewriting for term rewriting and unrestricted TRSs, we follow the standard approach [15, 14] where folding (also called collapsing) and unfolding (also referred to as copying) is directly incorporated in the graph rewrite relation. Unlike in the cited literature, we employ a very restrictive form of folding and unfolding. To this extend, we define for positions $p$ relations $\bowtie_p$ and $\bowtie_p$ on term graphs. Both relations preserve term structure. However, when $S \bowtie_p T$ holds then the subgraph $T \upharpoonright p$ admits strictly more sharing than $S \upharpoonright p$. Conversely, when $S \bowtie_p T$ holds, nodes above $p$ in $T$ admit less sharing than nodes above $p$ in $S$. Extending the graph rewrite relation $\rightarrow_{G(R)}$ by $\bowtie_p$ and $\bowtie_p$ addresses both problems highlighted in Example 4.4 and Example 4.6.

The relations $\bowtie_p$ and $\bowtie_p$ are based on single step approximations $\bowtie_m$ of $\succ_m$.

Definition 4.7. Let $\succ$ denote some total order on nodes that is irreflexive and transitive, let $\succsim$ denote the reflexive closure of $\succ$. Let $S$ be term graphs, and let $u, v \in S$ be nodes satisfying $u \succsim v$. We define $S \bowtie^u_v T$ for term graph $T$ if $S \succ_m T$ for the morphism $m$ identifying $u$ and $v$, more precisely, $m(u) = v$ and $m(w) = w$ for all $w \in S \setminus \{u\}$. We define $S \bowtie^u_v T$ if $S \bowtie^u_v T$ and $u \neq v$.  

10
We write $S \sqsupseteq_v T$ ($S \sqsupseteq_v T$) if there exists $u \in S$ such that $S \sqsupseteq^u_v T$ ($S \sqsupseteq^u_v T$) holds. Similar $S \sqsupseteq T$ ($S \sqsubseteq T$) if there exist nodes $u, v \in S$ such that $S \sqsupseteq^u_v T$ ($S \sqsupseteq^u_v T$) holds.

**Example 4.8.** Consider the term $t = (0 + 0) \times (0 + 0)$. Then $t$ is represented by the following three graphs that are related by $\sqsupseteq_2^1$ and $\sqsupseteq_1^1$ respectively.

Put otherwise, the term graph $T_2$ is obtained from $T_1$ by copying node 3, introducing the fresh node 2. The graph $T_3$ is obtained from $T_2$ by collapsing node 4 onto node 5.

Suppose $S \sqsupseteq^u_v T$. Then the morphism underlying $\sqsupseteq^u_v$ defines the identity on $V_S \setminus \{u\}$. In particular, it defines the identity on successors of $u, v \in S$. Thus the following is immediate.

**Lemma 4.9.** Let $S$ be a term graph, and let $u, v \in S$ be two distinct nodes. Then there exists a term graph $T$ such that $S \sqsupseteq^u_v T$ if and only if $L_S(u) = L_S(v)$ and $Succ_S(u) = Succ_S(v)$.

**Proof.** We prove the direction from left to right as the other is trivial. Suppose $S \sqsupseteq^u_v T$ and let $m$ be the morphism underlying $\sqsupseteq^u_v$. Observe $m(u) = m(v) = v$. And thus $L_S(u) = L_T(m(u)) = L_T(v) = L_T(m(v)) = L_S(v)$ follows. To prove the second assertion, pick nodes $u_i$ and $v_i$ such that $u \xrightarrow{i} u_i$ and $v \xrightarrow{i} v_i$. Since $S \sqsupseteq^u_v T$ we obtain $m(u_i) = m(v_i)$. Thus $u_i \neq v_i$ if and only if either $u_i = u$ or $v_i = u$. The former implies that $S$ is cyclic, the latter implies that $T$ is cyclic, contradicting that $S$ is a term graph or $m$ a morphism. We conclude $u_i = v_i$ for $i$ arbitrary, hence $Succ_S(u) = Succ_S(v)$. \qed

The restriction $u \gg v$ was put onto $\sqsupseteq^u_v$ so that $\sqsupseteq_v$ enjoys the following diamond property. Otherwise, the peak $\sqsupseteq^u_v \cdot \sqsupseteq^w_v \subseteq \sqsupseteq_w$ cannot be joined.

**Lemma 4.10.** $\sqsubseteq_u \cdot \sqsubseteq_v \subseteq \sqsubseteq_{w_1} \cdot \sqsubseteq_{w_2}$ where $w_1, w_2 \in \{u, v\}$.

**Proof.** Assume $T_1 \sqsubseteq^u_v S \sqsubseteq^w_v T_2$ for some term graphs $S, T_1$ and $T_2$. The only non-trivial case is $T_1 \sqsubseteq^u_v S \sqsubseteq^w_v T_2$ for $u' \neq u$ and $u \neq v$. We prove $T_1 \sqsubseteq_{w_1} \cdot \sqsubseteq_{w_2} T_2$ for $w_1, w_2 \in \{u, v\}$ by case analysis.

- Case $T_1 \sqsubseteq^{u'}_w S \sqsubseteq^{w'}_v T_2$ for $v' \neq u'$. We claim $T_1 \sqsubseteq^{u'}_w \cdot \sqsubseteq^{w'}_u T_2$. Let $m_1$ be the morphism underlying $\sqsubseteq^{u'}_w$ and let $m_2$ be the morphism underlying $\sqsubseteq^{w'}_u$ (c.f. Definition 4.7). We first show $L_{T_1}(v') = L_{T_1}(w)$ and Succ$_{T_1}(v') = Succ_{T_1}(w)$. Using Lemma 4.9 $S \sqsupseteq^u_v T_2$ yields $L_S(v') = L_S(w)$. Employing $v' \neq u'$ and $w \neq u'$ we see

$$L_{T_1}(v') = L_{T_1}(m_1(v')) = L_S(v') = L_S(w) = L_{T_1}(m_1(w)) = L_{T_1}(w)$$

11
where we employ \( m_1(v') = v \) and \( m_1(w) = w \). Again by Lemma 4.19 we see \( \text{Succ}_0(u') = \text{Succ}_0(w) \) and \( \text{Succ}_0(v') = \text{Succ}_0(w) \) by the assumption \( T_1 \sqsubseteq^w S \sqsubseteq^w T_2 \). We conclude \( \text{Succ}_0(v') = \text{Succ}_0(w) \) and thus

\[
\text{Succ}_0(T_1(v')) = \text{Succ}_0(T_1(m_1(v'))) = m_1^*\left(\text{Succ}_0(v')\right) = m_1^*\left(\text{Succ}_0(w)\right) = \text{Succ}_0(T_1(m_1(w))) = \text{Succ}_0(T_1(w)).
\]

By Lemma 4.19 we see \( T_1 \sqsubseteq^w S \sqsubseteq^w T_2 \) for some term graph \( U_1 \). Symmetrically, we can prove \( T_2 \sqsubseteq^w U_2 \) for some term graph \( U_2 \). Hence \( T_1 \sqsubseteq^w \cdot \sqsubseteq^w T_2 \) holds if \( U_1 = U_2 \). To prove the latter, one shows \( m_2 \cdot m_1 = m_1 \cdot m_2 \) by a straight forward case analysis.

- **CASE** \( T_1 \sqsubseteq^w S \sqsubseteq^w T_2 \) for \( u \neq v \). Without loss of generality assume \( u \succ v \). We claim \( T_1 \sqsubseteq^w \cdot \sqsubseteq^w T_2 \) and follow the pattern of the proof for the previous case. Note that \( L_{T_1}(u) = L_{T_1}(v) \) follows from \( L_S(u) = L_S(w) = L_S(v) \) as before, similar \( \text{Succ}_0(T_1(u)) = \text{Succ}_0(T_1(v)) \) follows from \( \text{Succ}_0(u) = \text{Succ}_0(w) = \text{Succ}_0(v) \) with \( w \notin \text{Succ}_0(w) \). Hence \( T_1 \sqsubseteq^w U_1 \) and symmetrically \( T_2 \sqsubseteq^w U_2 \) for some term graphs \( U_1 \) and \( U_2 \). One verifies \( m \cdot m_1 = m_1 \cdot m_2 \) for graph morphisms \( m_1 \) underlying \( \sqsubseteq^w \), \( m_2 \) underlying \( \sqsubseteq^w \), and \( m \) underlying \( \sqsubseteq^w \). We conclude \( T_1 \sqsubseteq^w \cdot \sqsubseteq^w T_2 \).

- **CASE** \( T_1 \sqsubseteq^w S \sqsubseteq^w T_2 \) for pairwise distinct \( u', v', v \). We show \( T_1 \sqsubseteq^w \cdot \sqsubseteq^w T_2 \). Let \( m \) be the morphism underlying \( \sqsubseteq^w \). Observe \( m(v) = v \) and \( m(v') = v' \) by our assumption. Hence \( L_{T_1}(v') = L_S(v') = L_S(v) = L_{T_1}(v) \) and \( L_{T_1}(v') = m^*(L_S(v')) = m^*(L_S(v)) = L_{T_1}(v) \). We obtain \( T_1 \sqsubseteq^w U_1 \) and symmetrically \( T_2 \sqsubseteq^w U_2 \). Finally, one verifies \( U_1 = U_2 \) by case analysis as above.

The above lemma implies confluence of \( \sqsubseteq \). Since \( \sqsubseteq^* = \sqsubseteq^* \), also \( \sqsubseteq \) is confluent.

**Definition 4.11.** Let \( S \) be a term graph and let \( p \) be a position in \( S \). We say that \( S \) folds strictly below \( p \) to the term graph \( T \), in notation \( S \updownarrow^p T \), if \( S \sqsubseteq^w T \) for nodes \( u, v \in S \) strictly below \( p \) in \( S \). The graph \( S \) unfolds above \( p \) to the term graph \( T \), in notation \( S \downarrow^p T \), if \( S \sqsubseteq^w T \) for some unshared node \( u \in T \) above \( p \), i.e., \( \text{Pos}_T(u) = \{ q \} \) for \( q \leq p \).

**Example 4.12.** Reconsider the term graphs \( T_1, T_2 \) and \( T_3 \) with \( T_1 \sqsubseteq^w T_2 \sqsubseteq^w T_3 \) from Example 4.8. Then \( T_1 \sqsubseteq^w T_2 \sqsubseteq^w T_3 \). Since node 3 is an unshared node above position 2 in \( T_2 \), further \( T_2 \updownarrow^2 T_3 \) since both nodes 4 and 5 are strictly below position 2 in \( T_2 \).
Note that for $S \equiv_T^n \ T$ the sets of positions $\mathcal{P} os_S$ and $\mathcal{P} os_T$ coincide, thus the $n$-fold composition $\ll^p_n$ of $\ll^p$ (and the $n$-fold composition $\triangledown^p_n$ of $\triangledown^p$) is well-defined for $p \in \mathcal{P} os_S$.

In the next two lemmas we prove that relations $\ll^p$ and $\triangledown^p$ fulfill their intended purpose.

**Lemma 4.13.** Let $S$ be a term graph and $p$ a position in $S$. If $S$ is $\ll^p$-minimal then the node corresponding to $p$ is unshared.

**Proof.** By way of contradiction, suppose $S$ is $\ll^p$-minimal but the node $w$ corresponding to $p$ is shared. We construct $T$ such that $S \ll^p T$. We pick an unshared node $v \in S$, and shared node $v_i \in S$, above $p$ such that $v \rightharpoonup v_i$. By a straight forward induction on $p$ we see that $v$ and $v_i$ exist as $w$ is shared. For this, note that at least the root of $S$ is unshared.

Define $T := (V_T, L_T, \text{Succ}_T)$ as follows: let $u$ be a fresh node such that $u \text{Succ} v_i$. set $V_T := V_S \cup \{u\}$; set $L_T(u) := L_S(v_i)$ and $\text{Succ}_T(u) := \text{Succ}_S(v_i)$; further replace the edge $v \rightharpoonup v_i$ by $v \rightharpoonup u$, that is, set $L_T(v) := \{v_1, \ldots, u, \ldots, v_i\}$ for $L_S(v) = \{v_1, \ldots, v_i, \ldots, v\}$. \(\triangledown\)

For the remaining cases, define $L_T(v) := L_S(u)$ and $\text{Succ}_T(v) := \text{Succ}_S(w)$. One easily verifies $T \equiv^n_T S$. Since by way of construction $u$ is an unshared node above $p$, $S \ll^p T$ holds.

**Lemma 4.14.** Let $S$ be a term graph, let $p$ be a position in $S$. If $S$ is $\triangledown^p$-minimal then the subgraph $S \upharpoonright p$ is maximally shared.

**Proof.** Suppose $S \upharpoonright p$ is not maximally shared. We show that $S$ is not $\triangledown^p$-minimal. Pick some node $u \in S \upharpoonright p$ such that there exists a distinct node $v \in S \upharpoonright p$ with $\text{term}(S \upharpoonright u) = \text{term}(S \upharpoonright v)$. For that we assume that $u$ is $\rightarrow$-minimal in the sense that there is no node $u'$ with $u \triangledown u'$ such that $u'$ would fulfill the above property. Clearly $L_S(u) = L_S(v)$ follows from $\text{term}(S \upharpoonright u) = \text{term}(S \upharpoonright v)$. Next, suppose $u \triangledown u_i$ and $v \triangledown v_i$ for some nodes $u_i \neq v_i$. But then $u_i$ contradicts minimality of $u$, and so we conclude $u_i = v_i$. Consequently $\text{Succ}_S(u) = \text{Succ}_S(v)$ follows as desired. Without loss of generality, suppose $u \succ v$. By Lemma 4.13 $S \equiv^n_T S$ and $S \triangledown^p T$ for some term graph $T$, since $u, v \in S \upharpoonright p$ also $S \triangledown^p T$ holds.

**Theorem 4.15** (Adequacy). Let $s$ be a term and let $S$ be a term graph such that $\text{term}(S) = s$. Then

\[ s \rightarrow_{\mathcal{R}} t \text{ if and only if } S \ll^p_{\triangledown^p} \rightarrow^p_{\mathcal{G}(\mathcal{R})} T, \]

for some term graph $T$ with $\text{term}(T) = t$.

**Proof.** First, we consider the direction from right to left. Assume $S \ll^p_{\triangledown^p} U \rightarrow^p_{\mathcal{G}(\mathcal{R})} V$. Note that $\triangledown^p$ preserves $\ll^p$-minimality. We conclude $V$ is $\ll^p$-minimal as $U$ is. Let $v \in V$ be the node corresponding to $p$. By Lemma 4.13 we see $\mathcal{P} os_U(v) = \{p\}$. Now consider the step $V \rightarrow_{\mathcal{G}(\mathcal{R})} T$. There exists a renaming $L' \rightarrow R'$ of $(L \rightarrow R) \in \mathcal{G}(\mathcal{R})$ such that $m: L' \rightarrow V \upharpoonright v$ is a morphism and $T = V[m(R')]_v$. Set $l := \text{term}(L')$ and $r := \text{term}(R')$, by definition $(l \rightarrow r) \in \mathcal{R}$. By Lemma 3.5 we obtain $l \sigma_m = \text{term}(V \upharpoonright v)$ for the substitution $\sigma_m$ induced by the morphism $m$. Define the context $C := \text{term}(V[\square]_v)$.\]
As $v$ is unshared, $C$ admits exactly one occurrence of $\Box$, moreover the position of $\Box$ in $C$ is $p$. By Lemma 4.3

$$\text{term}(V) = \text{term}(V[V \upharpoonright_v]) = C[\text{term}(V \upharpoonright_v)] = C[l_\sigma_m].$$

Set $T_v := (m(R') \cup V) \upharpoonright \text{rt}(m(R'))$, and observe $T = V[m(R')]_v = V[T_v]_v$. Using Lemma 4.3 and Lemma 4.2 we see

$$\text{term}(T) = \text{term}(V[T_v]_v) = C[\text{term}(T_v)] = C[r_\sigma_m].$$

As $\text{term}(S) = \text{term}(V)$ by Lemma 3.6, $\text{term}(S) = C[l_\sigma_m] \rightarrow_{R, p} C[r_\sigma_m] = \text{term}(T)$ follows.

Finally, consider the direction from left to right. For this suppose $s = C[l_\sigma] \rightarrow_{R, p} C[r_\sigma] = t$ where the position of the hole in $C$ is $p$. Suppose $S \triangleleft_{p} U \triangleright_{p} V$ for $\text{term}(S) = s$. We prove that there exists $T$ such that $V \rightarrow_{\mathcal{G}(R), p} T$ and $\text{term}(T) = t$. Note that $V$ is $\triangleright_{p}$-minimal and, as observed above, it is also $\triangleleft_{p}$-minimal. Let $v \in V$ be the node corresponding to $p$, by Lemma 4.3 the node $v$ is unshared. Next, observe $l_\sigma = s|_p = \text{term}(S \upharpoonright_p) = \text{term}(V \upharpoonright v)$ since $\text{term}(S) = \text{term}(V)$ (c.f. Lemma 3.6). Additionally, Lemma 4.3 reveals $V \upharpoonright v \in \triangledown(l_\sigma)$. Further, by Lemma 4.3 we see

$$s = C[l_\sigma] = \text{term}(V) = \text{term}(V[V \upharpoonright_v]) = \text{term}(V[\Box_v][l_\sigma]).$$

Since the position of the hole in $C$ and $\text{term}(V[\Box_v])$ coincides, we conclude that $C = \text{term}(V[\Box_v]).$

Let $L \rightarrow R \in \mathcal{G}(\mathcal{R})$ be the rule corresponding to $(l \rightarrow r) \in \mathcal{R}$, let $(L' \rightarrow R') \equiv (L \rightarrow R)$ be a renaming for $V$. As $L' \in \triangle(l)$ and $V \upharpoonright v \in \triangledown(l_\sigma)$, by Lemma 4.3 there exists a morphism $m: L' \rightarrow V \upharpoonright v$ and hence $V \rightarrow_{\mathcal{G}(R), p} T$ for $T = V[m(R')]_v$. Note that for the induced substitution $\sigma_m$ and $x \in \text{Var}(l)$, $\sigma_m(x) = \sigma(x)$. Set $T_v := (m(R') \cup V) \upharpoonright \text{rt}(m(R'))$, hence $T = V[T_v]_v$ and moreover $r_\sigma = r_\sigma_m = \text{term}(T_v)$ follows as in the first half of the proof. Employing Lemma 4.3 we obtain

$$t = C[r_\sigma] = \text{term}(V[\Box_v])[r_\sigma] = \text{term}(V[T_v]_v) = \text{term}(T).$$

$$\Box$$

We define $S \rightarrow_{\mathcal{G}(R), p} T$ if and only if $S \triangleleft_{p} U \triangleright_{p} V \rightarrow_{\mathcal{G}(R), p} T$. Employing this notion we can rephrase the conclusion of the Adequacy Theorem as: $s \rightarrow_{R, p} t$ if and only if $S \rightarrow_{\mathcal{G}(R), p} T$ for $\text{term}(S) = s$ and $\text{term}(T) = t$.

5 Implementing Term Rewriting Efficiently

Opposed to term rewriting, graph rewriting induces linear size growth in the length of derivations. The latter holds as a single step $\rightarrow_{\mathcal{G}}$ admits constant size growth:

**Lemma 5.1.** *If $S \rightarrow_{\mathcal{G}} T$ then $|T| \leq |S| + \Delta$ for some $\Delta \in \mathbb{N}$ depending only on $\mathcal{G}$.***
Proof. Set \( \Delta := \max\{|R| | (L \rightarrow R) \in \mathcal{G}\} \) and the lemma follows by definition. \( \square \)

It is easy to see that a graph rewrite step \( S \rightarrow_{G} T \) can be performed in time polynomial in the size of the term graph \( S \). By the above lemma we obtain that \( S \) can be normalised in time polynomial in \( |S| \) and the length of derivations. In the following, we prove a result similar to Lemma 5.1 for the relation \( \rightarrow_{G} \), where (restricted) folding and unfolding is incorporated. The main obstacle is that due to unfolding, size growth of \( \rightarrow_{G} \) is not bound by a constant in general. We now investigate into the relation \( U_p \) and \( U_p \).

**Lemma 5.2.** Let \( S \) be a term graph and let \( p \) a position in \( S \).

1. If \( S \prec_{p} T \) then \( \ell \leq |p| \) and \( |T| \leq |S| + |p| \).
2. If \( S \triangleright_{p} T \) then \( \ell \leq |S| |p| \) and \( |T| \leq |S| \).

Proof. We consider the first assertion. For term graphs \( U \), let \( P_{U} = \{ w \mid \operatorname{Pos}_{U}(w) = \{ q \} \text{ and } q \leq |p| \} \) be the set of unshared nodes above \( p \). Consider \( S \prec_{p} V \). Observe that \( P_{U} \subseteq P_{v} \). Clearly, \( P_{U} \subseteq P_{V} \) but moreover \( u \in P_{V} \) whereas \( u \notin P_{U} \). Hence for \( (S \prec_{p} T) = S = S_{0} \prec_{p} \ldots \prec_{p} S_{\ell} = T \), we observe \( P_{S} P_{S_{0}} \ldots P_{S_{\ell}} = P_{T} \). Note that \( |P_{S}| \geq 1 \) since \( \text{rt}(S) \in P_{S} \). Moreover, \( |P_{T}| = |p| + 1 \) since the node corresponding to \( p \) in \( T \) is unshared (c.f. Lemma 1.13). Thus from \( P_{S_{i}} \subseteq P_{S_{i+1}} \) (\( 0 \leq i < \ell \)) we conclude \( \ell \leq |p| \). Next, we see \( |T| \leq |S| + |p| \) as \( |T| = |S| + \ell \) by definition of \( \prec_{p} \).

We now prove the second assertion. Consider term graphs \( U \) and \( V \) such that \( U \triangleright_{p} V \).

By definition \( U \supseteq_{v} V \) where nodes \( u \) and \( v \) are strictly below position \( p \) in \( U \). Hence \( U \supseteq_{v} V \) for the morphism \( m \) underlying \( \supseteq_{v} \). As a simple consequence of Lemma 3.4 we obtain \( U \supseteq_{v} V \) and thus \( |U | p | \supseteq_{v} V | p | \). From this we conclude the lemma as above, where for \( |T| \leq |S| \) we employ that if \( U \supseteq_{v} V \) then \( |V | p | = |U | p | - 1 \).

By combining the above two lemmas we derive the following:

**Lemma 5.3.** If \( S \rightarrow_{G} T \) then \( |T| \leq |S| + \text{dp}(S) + \Delta \) and \( \text{dp}(T) \leq \text{dp}(S) + \Delta \) for some \( \Delta \in \mathbb{N} \) depending only on \( \mathcal{G} \).

Proof. Consider \( S \rightarrow_{G} T \), i.e., \( S \prec_{p} U \triangleleft_{p} V \rightarrow_{G} T \) for some position \( p \) and term graphs \( U \) and \( V \). Lemma 5.2 reveals \( |U| \leq |S| + |p| \) and further \( |V| \leq |U| \) for \( \Delta := \max\{|R| | (L \rightarrow R) \in \mathcal{G}\} \). As \( |p| \leq \text{dp}(S) \) we see \( |V| \leq |S| + \text{dp}(S) \). Since \( V \rightarrow_{G} T \) implies \( |T| \leq |V| + \Delta \) (c.f. Lemma 5.1) we establish \( |T| \leq |S| + \text{dp}(S) + \Delta \). Finally, \( \text{dp}(T) \leq \text{dp}(S) + \Delta \) follows from the easy observation that both \( U \prec_{p} V \) and \( U \triangleright_{p} V \) imply \( \text{dp}(U) = \text{dp}(V) \), likewise \( V \rightarrow_{G} T \) implies \( \text{dp}(T) \leq V + \Delta \). \( \square \)

**Lemma 5.4.** If \( S \rightarrow_{G} T \) then \( |T| \leq (\ell + 1)|S| + \ell^{2} \Delta \) for \( \Delta \in \mathbb{N} \) depending only on \( \mathcal{G} \).

Proof. We prove the lemma by induction on \( \ell \). The base case follows trivially, so suppose the lemma holds for \( \ell \), we establish the lemma for \( \ell + 1 \). Consider a derivation \( S \rightarrow_{G}^{\ell} T \)
By induction hypothesis, $|T| \leq (\ell + 1)|S| + \ell^2 \Delta$. Iterative application of Lemma 5.3 reveals $\text{dp}(T) \leq \text{dp}(S) + \ell \Delta$. Thus

$$|U| \leq |T| + \text{dp}(T) + \Delta$$

$$\leq ((\ell + 1)|S| + \ell^2 \Delta) + (\text{dp}(S) + \ell \Delta) + \Delta$$

$$\leq (\ell + 2)|S| + \ell^2 \Delta + \ell \Delta + \Delta$$

$$\leq (\ell + 2)|S| + (\ell + 1)^2 \Delta.$$ 

In the sequel, we prove that an arbitrary graph rewrite step $S \rightarrow \triangleright T$ can be performed in time cubic in the size of $S$. Lemma 5.4 then allows us to lift the bound on steps to a polynomial bound on derivations in the size of $S$ and the length of derivations. We closely follow the notions of [10]. As model of computation we use $k$-tape Turing Machines (TM for short) with dedicated input- and output-tape. If not explicitly mentioned otherwise, we will use deterministic TMs. We say that a (possibly nondeterministic) TM computes a relation $R \subseteq \Sigma^* \times \Sigma^*$ if for all $(x, y) \in R$, on input $x$ there exists an accepting run such that $y$ is written on the output tape.

We fix a standard encoding for term graphs $S$. We assume that for each function symbol $f \in \mathcal{F}$ a corresponding tape-symbols is present. Nodes and variables are represented by natural numbers, encoded in binary notation and possibly padded by zeros. We fix the invariant that natural numbers $\{1, \ldots, |S|\}$ are used for nodes and variables in the encoding of $S$. Thus variables (nodes) of $S$ are representable in space $O(\log(|S|))$.

Finally, term graphs $S$ are encoded as a ordered list of node specifications, i.e., triples of the form $\langle v, L(v), \text{Succ}(v) \rangle$ for all $v \in S$ (compare [15, Section 13.3]). We call the entries of a node specification node-field, label-field and successor-field respectively. We additionally assume that each node specification has a status flag (constant in size) attached. For instance, we use this field below to mark subgraphs.

For a suitable encoding of tuples and lists, a term graph $S$ is representable in size $O(\log(|S|) \ast |S|)$. For this, observe that the length of $\text{Succ}(v)$ is bound by the maximal arity of the fixed signature $\mathcal{F}$. In this spirit, we define the representation size of a term graph $S$ as $|S| := O(\log(|S|) \ast |S|)$.

Before we investigate into the computational complexity of $\rightarrow$, we prove some auxiliary lemmas.

**Lemma 5.5.** The subgraph $S \upharpoonright u$ of $S$ can be marked in quadratic time in $|S|$.

**Proof.** We use a TM that operates as follows: First, the graph $S$ is copied from the input tape to a working tape. Then the node specifications of $v \in S \upharpoonright u$ are marked in a breath-first manner. Finally the resulting graph is written on the output tape. In order to mark nodes two flags are employed, the permanent and the temporary flag. A node $v$ is marked permanent by marking its node specification permanent, and all node specifications of $\text{Succ}(v)$ temporary. Initially, $u$ is marked. Afterward, the machine iteratively marks temporary marked node permanent, until all nodes are marked permanent.
Notice that a node $v$ can be marked in time linear in $\|S\|$. For that, the flag of the node specification of $v$ is set appropriately, and $\text{Succ}(v)$ is copied on a second working tape. Then $S$ is traversed, and the node-field of each encountered node specification is compared with the current node written on the second working. If the nodes coincide, then the flag of the node specification is adapted, the pointer on the second working tape advances to the next node, and the process is repeated. Since the length of $\text{Succ}(v)$ is bounded by a constant (as $\mathcal{F}$ is fixed), marking a node requires a constant number of iterations. Since the machine marks at most $|S| \leq \|S\|$ nodes we obtain an overall quadratic time bound in $\|S\|$.

Lemma 5.6. Let $S$ be a term graph and let $p$ a position in $S$. A term graph $T$ such that $S \prec_p^1 T$ is computable in time $O(\|S\|^2)$.

Proof. Given term graph $S$, we construct a TM that produces a term graph $T$ such that $S \prec_p^1 T$ in time quadratic in $\|S\|$. For that, the machine traverse $S$ along the path induced by $p$ and introduces a fresh copy for each shared node encountered along that path. The machine has four working tapes at hand. On the first tape, the graph $S$ is copied in such a way that nodes are padded sufficiently by leading zero’s so that successors can be replaced by fresh nodes $u \leq 2 \cdot |S|$ inplace. The graph represented on the first tape is called the current graph, its size will be bound by $O\|S\|$ at any time. On the second tape the position $p$, encoded as list of argument positions, is copied. The argument position referred by the tape-pointer is called current argument position and initially set to the first position. The third tape holds the current node, initially the root $\text{rt}(S)$ of $S$. Finally, the remaining tape holds the size of the current graph in binary, the current size. One easily verifies that these preparatory steps can be done in time linear in $S$.

The TM now iterates the following procedure, until every argument position in $p$ was considered. Let $v$ be the current node, let $S_i$ the current graph and let $i$ be the current argument position. We machine keeps the invariant that $v$ is unshared in $S_i$. First, the node $v_i$ with $v \xleftarrow{i} v_i$ in $S_i$ is determined in time linear in $\|S\|$, the current node is replaced by $v_i$. Further, the pointer on the tape holding $p$ is advanced to the next argument position. Since $v$ is unshared, $v_i$ is shared if and only if $v_i \in \text{Succ}(u)$ for $u \neq v$. The machine checks whether $v_i$ is shared in the current graph, by the above observation in time linear in $\|S\|$. If $v_i$ is unshared, the machine enters the next iteration. Otherwise, the node $v_i$ is cloned in the following sense. First, the the $i$-th successor $v_i$ of $v$ is replaced by a fresh node $u$. The fresh node is obtained by increasing the current node by one, this binary number is used as fresh node $u$. Further, the node specification $\langle u, \text{L}(v_i), \text{Succ}(v_i) \rangle$ is appended to the current graph $S_i$. Call the resulting graph $S_{i+1}$. Then $S_i \sqsubseteq^u S_{i+1}$ with $\mathcal{P}_{\text{Pos}}(S_{i+1}) = \{q\}$ and $q \leq p$, i.e., $S_i \prec_p S_{i+1}$.

When the procedure stops, the machine has computed $S = S_0 \prec_p S_1 \prec_p \ldots \prec_p S_n = T$. One easily verifies that $S_n$ is $\prec_p$-minimal as every considered node along the path $p$ is unshared. Each iteration takes time linear in $\|S\|$. As at most $|p| \leq |S|$ iterations have to be performed, we obtain the desired bound. □
Lemma 5.7. Let \( S \) be a term graph and \( p \) a position in \( S \). The term graph \( T \) such that \( S \uparrow_p T \) is computable in time \( O(||S||^2) \).

Proof. Define the height \( ht_U(u) \) of a node \( u \) in a term graph \( U \) inductively as usual:

\[
ht_U(u) := 0 \text{ if } \text{Succ}(u) = [] \text{ and } ht_U(v) := 1 + \max_{v \in \text{Succ}(u)} ht_U(v) \text{ otherwise.}
\]

We drop the reference to the graph \( U \) when referring to the height of nodes in the analysis of the normalising sequence \( S \uparrow_p T \) below. This is justified as the height of nodes remain stable under \( \sqcap \)-reductions.

Recall the definition of \( \uparrow_p \): \( U \uparrow_p V \) if there exist nodes \( u, v \) strictly below \( p \) with \( U \sqsupseteq V \). Clearly, for \( u, v \) given, the graph \( V \) is constructable from \( U \) in time linear in \(| U |\). However, finding arbitrary nodes \( u \) and \( v \) such that \( U \sqsupseteq V \) takes time quadratic in \(| U |\) worst case. Since up to linear many \( \sqcap \)-steps in \(| S |\) need to be performed, a straightforward implementation admits cubic runtime complexity. To achieve a quadratic bound in the size of the starting graph \( S \), we construct a TM that implements a bottom up reduction-strategy. More precise, the machine implements the maximal sequence

\[
S = S_1 \sqsupseteq_{u_1} S_2 \sqsupseteq_{u_2} \cdots \sqsupseteq_{u_{\ell-1}} S_{\ell}
\]

satisfying, for all \( 1 \leq i < \ell - 1 \), (i) either \( ht(u_i) = ht(u_{i+1}) \) and \( u < v \) or \( ht(u_i) < ht(u_{i+1}) \), and (ii) for \( S_i \sqsupseteq_{u_i} \cdots \sqsupseteq_{u_k} S_{i+1}, u_i \) and \( v_{i,j} \) \((1 \leq j \leq k)\) are strictly below \( p \).

By definition \( S \uparrow_p S_{\ell} \), it remains to show that the sequence \((u)\) is normalising, i.e., \( S_{\ell} \) is \( \uparrow_p \)-minimal. Set \( d := \text{dp}(S \mid p) \) and define, for \( 0 \leq h \leq d \),

\[
\sqcap_h := \bigcup_{u,v \in S^p:ht(v)=h} \sqcap_v.
\]

Observe that each \( \sqcap_{u_i} \)-step in the sequence \((u)\) corresponds to a step \( \sqcap_{(h)} \) for some \( 0 \leq h \leq d \). Moreover, it is not difficult to see that

\[
S = S_{i_0} \sqsupseteq_{(0)} S_{i_1} \sqsupseteq_{(1)} \cdots \sqsupseteq_{(d)} S_{i_{d+1}} = S_{\ell}
\]

for \( \{S_{i_0}, \ldots, S_{i_{d+1}}\} \subseteq \{S_1, \ldots, S_{\ell-1}\} \).

Next observe \( S_i \sqsupseteq_{(h_1)} S_{i+1} \sqsupseteq_{(h_2)} S_{i+2} \) and \( h_1 > h_2 \) implies \( S_i \sqsupseteq_{(h_2)} \cdot \sqsupseteq_{(h_1)} S_{i+2} \): suppose \( S_i \sqsupseteq_{u_i} S_{i+1} \sqsupseteq_{v_i} S_{i+2} \) where \( ht(u) > ht(v) \) and \( u', u, v, v' \in S \mid p \), we show \( S_i \sqsupseteq_{u'} \cdot \sqsupseteq_{v'} S_{i+2} \). Inspecting the proof of Lemma 4.10 we see \( \sqcap_u \cdot \sqcap_v \subseteq \sqcap_v \cdot \sqcap_u \) for the particular case that \( u', u, v \) and \( v' \) pairwise distinct. The latter holds as \( ht(u') = ht(u) \neq ht(v) = ht(v') \). Hence it remains to show \( S_i \sqsupseteq_{u'} S_{i+1} \) for some term graph \( S_{i+1}' \), or equivalently \( L_{S_i}(v) = L_{S_i}(v') \) and \( \text{Succ}_{S_i}(v) = \text{Succ}_{S_i}(v') \) by Lemma 4.9. The former equality is trivial, for the latter observe \( ht(u') = ht(u) > ht(v) = ht(v') \) and thus neither \( u' \notin \text{Succ}_{S_i}(v') \) nor \( u' \notin \text{Succ}_{S_i}(v) \). We see \( \text{Succ}_{S_i}(v) = \text{Succ}_{S_{i+1}}(v) = \text{Succ}_{S_{i+1}}(v') = \text{Succ}_{S_i}(v') \).

Now suppose that \( S_{\ell} \) is not \( \uparrow_p \)-minimal, i.e., \( S_{\ell} \sqsupseteq_{(h)} U \) for some \( 0 \leq h \leq d \) and term graph \( U \). But then we can permute steps in the reduction \((u)\) such that \( S_{h+1} \sqsupseteq_{(h)} V \) for some term graph \( V \). This contradicts \( \sqsubseteq_{(h)} \)-minimality of \( S_{h+1} \). We conclude that \( S_{\ell} \) is \( \uparrow_p \)-minimal. Thus sequence \((u)\) is \( \uparrow_p \)-normalising.
We now construct a TM operating in time $O(\|S\|^2)$ that, on input $S$ and $p$, computes the sequence (a). We use a dedicated working tape to store the current graph $S_i$. Initially, the term graph $S$ is copied on this working tape. Further, the node $w$ corresponding to $p$ in $S$ is computed by recursion on $p$ in time $\|S\|^2$. Afterward, the quadratic marking algorithm of Lemma 5.5 is used to mark the subgraph $S \mid w$ in $S$.

The TM operates in stages, where in each stage the current graph $S_{ih}$ is replaced by $S_{ih+1}$ for $S_{ih} \sqcup (h) S_{ih+1}$. Consider the subsequence

$$S_{ih} = S_{j_1} \sqcup u_{j_1} \cdots \sqcup u_{j_l} S_{j_{l+1}} = S_{ih+1}$$

of sequence (a) for $j_1 := i_h$ and $j_{l+1} := i_{h+1}$. Then $h = ht(u_{j_1}) = \cdots = ht(u_{j_l})$. Call $h$ the current height. To compute the above sequence efficiently, the TM uses the flags deleted, temporary and permanent besides the subterm marking. Let $S_j \ (j_1 \leq j \leq j_{l+1})$ be the current graph. If a node $u$ is marked deleted, it is treated as if $u \not\in S_j$, that is, when traversing $S_j$ the corresponding node specification is ignored. Further, the machine keeps the invariant that when $u \in S_j \mid p$ then (the node specification of) $u$ is marked permanent if and only if $ht(u) < h$. Thus deciding whether $ht(u) = h$ for some node $u \in S_i$ amounts to checking whether $u$ is not marked permanent, but all successors $\text{Succ}(u)$ are marked permanent.

Dealing with node markings, the machine additionally record whether $u_i \in \text{Succ}(u)$ is marked permanent in the node specification of $u$. Since the length of $\text{Succ}(u)$ is bounded by a constant, this is can be done in constant space. At the beginning of each stage, the machine is in one of two states, say $p$ and $q$ (for current height $h = 0$, the initial state is $p$).

- **State p.** In this state the machine is searching the next node $u_j$ to collapse. It keeps the invariant that previously considered nodes $u_j$, for $j_1 \leq j \leq i$ are marked temporary. Reconsider the definition of the sequence (a). The node $u_j$ is the least node (with respect to $>$ underlying $\sqcup$) satisfying (i) $u_j$ is marked by the subterm marking, and (ii) $u_j$ is not marked permanent but all nodes in $\text{Succ}(u_j)$ are marked permanently, and (iii) $u_j$ is not marked temporary. Recall that node specifications are ordered in increasing order. In order to find $u_j$, the graph $S_j$ is scanned from top to bottom, solely based on the node specification properties (i) — (iii) are checked, and the first node satisfying (i) — (iii) is returned.

Suppose the node $u_j$ is found. The machine marks the node $u_j$ temporary and writes $u_j$, $L(u_j)$ and $\text{Succ}(u_j)$ on dedicated working tapes. Call $u_j$ the current node. The machine goes into state $q$ as described below. On the other hand, if $u_j$ is not found, the stage is completed as all nodes of height $h$ are temporary marked. The temporary marks, i.e., the marks of node $u_{j_1}, \ldots, u_{j_l}$ are transformed into permanent ones and the stage is completed. Notice that all nodes of height less or equal to $h$ are marked permanent this way. The invariant on permanent marks is recovered, the machine enters the next stage.

One verifies that one transition from state $p$ requires at most linearly many steps in $\|S\|$.
- State \( q \). The machine iteratively computes the sequence

\[
S_j = S_{1,j} \supseteq u_{1,j} \ldots \supseteq u_{k,j} \supseteq S_{k,j} = S_{j+1}
\]

for current node \( u_j \) as determined in state \( p \). Suppose \( S_{i,j}, 1 \leq i \leq k \) is the current graph. The machine searches for the node \( v_{i,j} \in S_{i,j} \mid p \), \( v_{i,j} \geq v_{i-1,j} \) (for \( i \geq 1 \)) such that \( S_{i,j} \supseteq u_{i,j} \supseteq S_{i,j+1} \) for current node \( u_j \in S_{i,j} \mid p \). For that, the machine scans the current graph from top to bottom, comparing label- and successor-field with the ones written on the dedicated working tapes in state \( p \). Then \( v_{i,j} \in S_{i,j} \mid p \) is checked using the subterm marking. If \( v_{i,j} \) is not found, the current graph \( S_{i,j} \) is \( \supseteq u_{i,j} \)-minimal according to Lemma 4.9. The above sequence has been computed, the machine enters state \( p \). Otherwise, the machine writes \( u_{i,j} \) on an additional working tape and applies the morphism underlying \( \supseteq u_{i,j} \) on the current graph.

When the machine exists the above procedure, the current graph is the \( \triangleright_p \)-minimal graph \( S_{\ell} \). The current graph is then written on the output tape in two stages. During the first stage, the current graph is traversed from top to bottom, and the list of non-deleted nodes \( u_1, u_2, \ldots \) is written on a separate working tape in time \( O(\|S\|) \). Let \( s \) be the isomorphism \( s(u_i) = i \). In the second stage, the current graph \( S_{\ell} \) is traversed from top to bottom a second time. For each node specification \( \langle u_i, L(u_i), \text{Succ}(u_i) \rangle \), the node specification \( \langle s(u_i), L(u_i), s \circ \text{Succ}(u_i) \rangle \) is written on the output tape. Using a counter and the list of marked nodes \( u_1, u_2, \ldots \), this is achieved in time \( O(\|S\|^2) \). The machine outputs an increasing list of node specifications, the represented graph is isomorphic to \( S_{\ell} \).

We now investigate on the computational complexity of the above procedure. All preparatory steps, that is, initialising the current graph, computing the node corresponding to \( p \) and marking the subterm \( S \mid p \), require \( O(\|S\|^2) \) many steps in total. Since every time when the machine enters state \( p \) one unmarked node is marked, we conclude that the machine enters state \( p \) at most \( \|S\| \leq \|S\| \) often. The machine enters state \( q \) either after leaving state \( p \) or when \( S_{i,j} \supseteq u_{i,j} \supseteq S_{i,j+1} \) in the reduction (2) holds. By the previous observation, and employing Lemma 5.3 on the sequence (2) we see that the constructed TM enters state \( q \) at most \( O(\|S\|) = O(\|S\|) \) often. Since each state transition requires at most \( O(\|S\|) \) many steps, we conclude that \( S_{\ell} \) is constructed in time \( O(\|S\|^2) \). Finally, writing the normalised representation of \( S_{\ell} \) on the output tape requires again at most \( O(\|S\|^2) \) many steps. Summing up, the machine operates in time \( O(\|S\|^2) \). This concludes the lemma.

**Lemma 5.8.** Let \( S \) be a term graph, let \( p \) be a position of \( S \) and let \( L \rightarrow R \) be a rewrite rule of the simulating graph rewrite system. It is decidable in time \( O(\|S\|^2 \times 2^{\|L \rightarrow R\|}) \) whether \( S \rightarrow_{p,L \rightarrow R} T \) for some term graph \( T \). Moreover, the term graph \( T \) is computable from \( S \), \( p \) and \( L \rightarrow R \) in time \( O(\|S\|^2 \times 2^{\|L \rightarrow R\|}) \).
Proof. We construct a TM that on input $S$, $p$ and $L \rightarrow R$ computes the reduct $T$ for $S \xrightarrow{p, L \rightarrow R} T$. If the latter does not hold, the machine rejects. For this we suppose that the nodes of $L \rightarrow R$ are chosen in such a way that $V_{L \rightarrow R} = \{1, \ldots, |L \rightarrow R|\}$ (we keep this invariant when constructing the final algorithm). Let $u$ be the node corresponding to $p$ in $S$. In Lemma 24] it is shown that there exists a TM operating in time $2^{O(|L|)} \ast O(|S|^2)$ that, on input $L$, $S$ and $u$, either writes on its output-tape the graph morphism $m$ such that $m: L \rightarrow S \upharpoonright u$ if it exists, or fails. The morphism $m$ is encoded as an associative list, more precisely, a list of pairs $(u, m(u))$ for $u \in L$. The size of this list is bound by $O(|L| \ast \log(|L| + |S|))$. First, this machine is used to compute $m: L \rightarrow S \upharpoonright u$, the resulting morphism is stored on a working tape. For this, the node $u$ is computed in time $||S||^2$ beforehand. If constructing the morphism fails, then rule $L \rightarrow R$ is not applicable at position $p$, i.e., $u$ is not a redex in $S$ with respect to $L \rightarrow R$. The constructed machine rejects. Otherwise, the reduct $T$ is computed as follows.

Set $L' := r(L)$ and $R' := r(R)$ for the graph morphism $r$ defined by $r(v) := v + |S|$. Then $R' \cap S = \emptyset$. We compute $T = S[m'(R')]_u$ for $m': L' \rightarrow S \upharpoonright u$ using the morphism $m: L \rightarrow S \upharpoonright u$ as computed above. Let $f(v) = m(v)$ if $L_R(v) \in V$ and $f(v) = v + |S|$ otherwise. Since $V_L \cap V_R = \text{Var} R$ we see that $T = S[f(R)]_u$.

Next, the machine constructs $S \cup f(R)$ on an additional working tape as follows. First, $S$ is copied on this tape in time linear in $||S||$. Simultaneously, $|S|$ is computed on an additional tape. Using the counter $|S|, v+|S|$ is computable in time $O(\log(|R|) + \log(|S|))$, whereas $m(v)$ for $v \in L \cap R = \text{Var} R$ is computable in time $O(|L| \ast \log(|L| + |R|))$ (traversing the associative list representing $m$). We bind the complexity of $f$ by $O(||L|| \ast ||R|| \ast ||S||)$ independent on $v$. Finally, for each node-specification $\langle v, L(v), \text{Succ}(v) \rangle$ with $L(v) \in F$ encountered in $R$, the machine appends $\langle f(v), L(v), f^*(\text{Succ}(v)) \rangle$. Employing $L \cap R = \text{Var}(R)$, one verifies that $S \cup f(R)$ is obtained this way. Overall, the runtime is $O(||L|| \ast ||R||^2 \ast ||S||)$.

Employing $||f(R)|| = O(|R| \ast \log(\max(|R|, |S|)))$, we see that $S \cup f(R)$ can be bound in size by $O(||S|| \ast ||R||)$. To obtain $T = (S \cup m(R')) \upharpoonright v = (S \cup f(R)) \upharpoonright v$ for $v$ either $\text{rt}(m(R'))$ or $\text{rt}(S)$, the quadratic marking algorithm of Lemma 5.5 is used. Finally, the marked subgraph obeying the standard encoding is written onto the output tape as in Lemma 5.7.

We sum up: it takes at most $2^{O(|L|)} \ast O(||S||^2)$ many steps to compute the morphism $m$. The graph $S \cup m(R')$ is obtained in time $O(||L|| \ast ||R|| \ast ||S||)$. Marking $T$ in $S \cup m(R')$ requires at most $O(||S|| \ast ||m(R')||^2) = O(||S||^2 \ast ||R||^2)$ many steps. Finally, the reduct $T$ is written in $||S \cup f(R)||^2 = O(||S||^2 \ast ||R||^2)$ steps onto the output-tape. Overall, the runtime is bound by $2^{O(|L|)} \ast O(||S||^2 \ast ||R||^2)$ worst case. \qed

Lemma 5.9. Let $S$ be a term graph and let $G(R)$ be the simulating graph rewrite system of $R$. If $S$ is not a normal-form of $G(R)$ then there exists a position $p$ and rule $(L \rightarrow R) \in G(R)$ such that a term graph $T$ with $S \xrightarrow{G(R), p, L \rightarrow R} T$ is computable in time $O(||S||^3)$.

Proof. The TM searches for a rule $(L \rightarrow R) \in G$ and position $p$ such that $S \xrightarrow{G(R), p, L \rightarrow R} T$ for some term graph $T$. For this, it enumerates the rules $(L \rightarrow R) \in G$ on a separate
working tape. For each rule \( L \rightarrow R \), each node \( u \in S \) and some \( p \in \text{Pos}_S \) it computes \( S_1 \) such that \( S \triangleright^1_p S_1 \) in time quadratic in \( |S|^2 \) (c.f. Lemma 5.7). Using the machine of Lemma 5.7, it decides in time \( 2^{O(|L|)} \ast O(|S_1|^2) \) whether rule \( L \rightarrow R \) applies to \( S_1 \) at position \( p \). Since \( \mathcal{R} \) is fixed, \( 2^{O(|L|)} \) is constant, thus the TM decides whether rule \( L \rightarrow R \) applies in time \( O(|S_1|^2) = O(|S|^2) \). Note that the choice of \( p \in \text{Pos}_S(u) \) is irrelevant, since \( S \triangleright^1_p S_1 \) and \( S \triangleright^1_p S_2 \) for \( p_i, p_j \in \text{Pos}_S(u) \) implies \( S_1 \cong S_2 \). Hence the node corresponding to \( p_i \) in \( S_1 \) is a redex with respect to \( L \rightarrow R \) if and only if the node corresponding to \( p_j \) is. Suppose rule \( L \rightarrow R \) applies at \( S_1 \mid p \). One verifies \( S_1 \mid p \cong S_2 \mid p \) for term graph \( S_2 \) such that \( S \triangleright^1_p S_2 \). We conclude \( S \rightarrow_{G(\mathcal{R}), p, L \rightarrow R} T \) for some position \( p \) and rule \( (L \rightarrow R) \in \mathcal{G}(\mathcal{R}) \) if and only if the above procedure succeeds. From \( u \) one can extract some position \( p \in \text{Pos}_S(u) \) in time quadratic in \( |S| \). This can be done for instance by implementing the function \( \text{pos}(u) = \varepsilon \) if \( u = \text{rt}(S) \) and \( \text{pos}(u) = p_i \) for some node \( v \in S \) with \( v \overset{\rightarrow}{\rightarrow} u \) and \( \text{pos}(v) = p \). Overall, the position \( p \in \text{Pos}_S \) and rule \( (L \rightarrow R) \in \mathcal{G} \) is found if and only if \( S \rightarrow_{G(\mathcal{R}), p, L \rightarrow R} T \) for some term graph \( T \). Since \( |S| \leq |S| \) nodes, and only a constant number of rules have to be checked, the overall runtime is \( O(|S|^3) \).

To obtain \( T \) from \( S \), \( p \), and \( L \rightarrow R \), the machine now combines the machines from Lemma 5.6, Lemma 5.7, and Lemma 5.8. These steps can even be performed in time \( O(|S|^2) \), employing that the size of intermediate graphs is bound linear in the size of \( S \) (compare Lemma 5.2) and that sizes of \( (L \rightarrow R) \in \mathcal{G}(\mathcal{R}) \) are constant. \( \square \)

**Lemma 5.10.** Let \( S \) be a term graph and let \( \ell := \text{dl}(S, \rightarrow_{G(\mathcal{R})}) \). Suppose \( \ell = \Omega(|S|) \).

1. Some normal-form of \( S \) that is computable in deterministic time \( O(\log(\ell)^3 \ast \ell^5) \).

2. Any normal-form of \( S \) is computable in nondeterministic time \( O(\log(\ell)^2 \ast \ell^5) \).

**Proof.** We prove the first assertion. Consider the normalising derivation

\[
S = T_0 \rightarrow_{G(\mathcal{R})} \ldots \rightarrow_{G(\mathcal{R})} T_i = T. \tag{1}
\]

where, for \( 0 \leq i < l \), \( T_i \) is obtained from \( T_{i+1} \) as given by Lemma 5.9. By Lemma 5.4 we see \( |T_i| \leq (\ell + 1)|S| + \ell^2 \Delta = O(\ell^2) \). Here the latter equality follows by the assumption \( \ell = \Omega(|S|) \). Recall \( \|T_i\| = O(\log(|T_i|) \ast |T_i|) \) \((0 \leq i < l)\) and hence \( \|T_i\| = O(\log(\ell)^2 \ast \ell^2) = O(\log(\ell) \ast \ell^2) \). From this, and Lemma 5.9, we obtain that \( T_{i+1} \) is computable from \( T_i \) in time \( O(|T_i|^3) = O(\log(\ell)^3 \ast \ell^5) \). Since \( l \leq \text{dl}(S, \rightarrow_{G(\mathcal{R})}) = \ell \) we conclude the first assertion.

We now consider the second assertion. Reconsider the proof Lemma 5.9. For a given rewrite-position \( p \), a step \( S \rightarrow_{G(\mathcal{R})} T \) can be performed in time \( O(|S|^2) \). A nondeterministic TM can guess some position \( p \), and verify whether the node corresponding to \( p \) is a redex in time \( O(|S|^2) \). In total, the reduce \( T \) can be obtained in nondeterministic time \( O(|S|^2) \). Hence, following the proof of the first assertion, one easily verifies the second assertion. \( \square \)
6 Discussion

We present an application of our result in the context of implicit computational complexity theory (see also [6 5]).

Definition 6.1. Let $\mathcal{N} \subseteq \text{Val}$ be a finite set of non-accepting patterns. We call a term $t$ accepting (with respect to $\mathcal{N}$) if there exists no $p \in \mathcal{N}$ such that $p\sigma = t$ for some substitution $\sigma$. We say that $R$ computes the relation $R \subseteq \text{Val} \times \text{Val}$ with respect to $\mathcal{N}$ if there exists $f \in \mathcal{D}$ such that for all $s, t \in \text{Val},$

$$R(s, t) :\iff f(s) \rightarrow^*_R t \text{ and } t \text{ is accepting}.$$  

On the other hand, we say that a relation $R$ is computed by $\mathcal{R}$ if $R$ is defined by the above equations with respect to some set $\mathcal{N}$ of non-accepting patterns.

For the case that $\mathcal{R}$ is confluent we also say that $\mathcal{R}$ computes the (partial) function induced by the relation $R$.

The reader may wonder why we restrict to binary relations, but this is only a non-essential simplification that eases the presentation. The assertion that for normal-forms $t, t'$ is accepting amounts to our notion of accepting run of a TRS $\mathcal{R}$. This aims to eliminate by-products of the computation that should not be considered as part of the relation $R$. (A typical example would be the constant $\perp$ if the TRS contains a rule of the form $t \rightarrow \perp$ and $\perp$ is interpreted as undefined.) The restriction that $\mathcal{N}$ is finite is essential for the simulation results below: If we implement the computation of $\mathcal{R}$ on a TM, then we also have to be able to effectively test whether $t$ is accepting.

To compute a relation defined by $\mathcal{R}$, we encode terms as graphs and perform graph rewriting using the simulating GRS $\mathcal{G}(\mathcal{R})$.

Theorem 6.2. Let $\mathcal{R}$ be a terminating TRS, moreover suppose $\text{rc}_\mathcal{R}(n) = O(n^k)$ for all $n \in \mathbb{N}$ and some $k \in \mathbb{N}, k \geq 1$. The relations computed by $\mathcal{R}$ are computable in nondeterministic time $O(n^{5k+2})$. Further, if $\mathcal{R}$ is confluent then the functions computed by $\mathcal{R}$ are computable in deterministic time $O(n^{7k+3})$. Here $n$ refers to the size of the input term.

Proof. We investigate into the complexity of a relation $R$ computed by $\mathcal{R}$. For that, single out the corresponding defined function symbol $f$ and fix some argument $t \in \text{Val}$. Suppose the underlying set of non-accepting patterns is $\mathcal{N}$. By definition, $R(s, t)$ if and only if $f(s) \rightarrow^*_R t$ and $t \in \text{Val}$ is accepting with respect to $\mathcal{N}$. Let $S$ be a term graph such that $\text{term}(S) = f(s)$ and recall that $|S| \leq |f(s)|$. Set $\ell := \text{dl}(S, \rightarrow^*_\mathcal{G}(\mathcal{R}))$. By the Adequacy Theorem 4.13, we conclude $S \rightarrow^*_\mathcal{G}(\mathcal{R}) T$ where $\text{term}(T) = t$, and moreover, $\ell \leq \text{rc}_\mathcal{R}(|f(s)|) = O(n^k)$. By Lemma 5.9, we see that $T$ is computable from $S$ in nondeterministic time $O(\log(\ell)^2 \cdot \ell^5) = O(\log(n^k)^2 \cdot n^{5k}) = O(n^{5k+2})$. Clearly, we can decide in time linear in $|T| = \mathcal{O}(\ell^2) = O(n^{2k})$ (c.f. Lemma 5.3) whether $\text{term}(T) \in \text{Val}$, further in time quadratic in $|T|$ whether $\text{term}(T)$ is accepting. For the latter, we use the matching algorithm of Lemma 5.8 on the fixed set of non-accepting patterns, where we employ $p\sigma = \text{term}(T)$ if and only if there exists a morphism $m : P \rightarrow T$ for
$P \in \Delta(p)$ (c.f. Lemma 4.3 and Lemma 3.5). Hence overall, the accepting condition can be checked in (even deterministic) time $O(n^{4k})$. If the accepting condition fails, the TM rejects, otherwise it accepts a term graph $T$ representing $t$. The machine does so in nondeterministic time $O(n^{5k+2})$ in total. As $s$ was chosen arbitrary, we conclude the first half of the theorem.

Finally, the second half follows by identical reasoning, where we use the deterministic TM as given by 5.10 instead of the nondeterministic one.

Let $R$ be a binary relation such that $R(x, y)$ can be decided by some nondeterministic TM in time polynomial in the size of $x$. The function problems $R_F$ associated with $R$ is: given $x$, find some $y$ such that $R(x, y)$ holds. The class FNP is the class of all functional problems defined in the above way, compare [13]. FP is the subclass resulting if we only consider function problems in FNP that can be solved in polynomial time by some deterministic TM. As by-product of Theorem 6.2 we obtain:

**Corollary 6.3.** Let $\mathcal{R}$ be a terminating TRS with polynomially bounded runtime complexity. Suppose $\mathcal{R}$ computes the relation $R$. Then $R_F \in$ FNP for the function problem $R_F$ associated with $R$. Moreover, if $\mathcal{R}$ is confluent then $R_F \in$ FP.

**Proof.** The nondeterministic TM $M$ as given by Theorem 6.2 (the deterministic TM $M$, respectively) can be used to decide membership $(s, t) \in R$. Observe that by the assumptions on $\mathcal{R}$, the runtime of $M$ is bounded polynomially in the size of $s$. Recall that $s$ is represented as some term graph $S$ with $\text{term}(S) = s$, in particular $s$ is encoded over the alphabet of $M$ in size $\|S\| = O(\log(|S|) \ast |S|)$ for $|S| \leq |s|$. Thus trivially $M$ operates in time polynomially in the size of $S$.

**References**

[1] M. Avanzini and G. Moser. Complexity Analysis by Rewriting. In Proc. of 9th FLOPS, volume 4989 of LNCS, pages 130–146. Springer Verlag, 2008.

[2] M. Avanzini and G. Moser. Dependency Pairs and Polynomial Path Orders. In Proc. of 20th RTA, volume 5595 of LNCS, pages 48–62. Springer Verlag, 2009.

[3] M. Avanzini and G. Moser. Complexity Analysis by Graph Rewriting. In Proc. of 11th FLOPS, LNCS. Springer Verlag, 2010. To appear.

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

[5] U. Dal Lago and S. Martini. Derivational Complexity is an Invariant Cost Model. In Proc. of 1st FOPARA, 2009.

[6] U. Dal Lago and S. Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In Proc. of 36th ICALP, volume 5556 of LNCS, pages 163–174. Springer Verlag, 2009.
[7] J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *JAR*, 40(3):195–220, 2008.

[8] A. Koprowski and J. Waldmann. Arctic Termination . . . Below Zero. In *Proc. of 19th RTA*, volume 5117 of *LNCS*, pages 202–216. Springer Verlag, 2008.

[9] M. Korp and A. Middeldorp. Match-bounds revisited. *IC*, 207(11):1259–1283, 2009.

[10] Dexter C. Kozen. *Theory of Computation*. Springer Verlag, first edition, 2006.

[11] G. Moser and A. Schnabl. The Derivational Complexity Induced by the Dependency Pair Method. In *Proc. of 20th RTA*, volume 5595 of *LNCS*, pages 255–260. Springer Verlag, 2009.

[12] G. Moser, A. Schnabl, and J. Waldmann. Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations. In *Proc. of 28th FST-TICS*, pages 304–315. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[13] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, second edition, 1995.

[14] D. Plump. Essentials of Term Graph Rewriting. *ENTCS*, 51:277–289, 2001.

[15] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracks in Theoretical Computer Science*. Cambridge University Press, 2003.