Derivational Complexity is an Invariant Cost Model[⋆]

Ugo Dal Lago and Simone Martini

Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
{dallago,martini}@cs.unibo.it

Abstract. We show that in the context of orthogonal term rewriting systems, derivational complexity is an invariant cost model, both in innermost and in outermost reduction. This has some interesting consequences for (asymptotic) complexity analysis, since many existing methodologies only guarantee bounded derivational complexity.

1 Introduction

Genuine applications of computational complexity to rule-based programming are few and far between. Notwithstanding the development and diffusion of high-level programming languages of this kind, most part of the complexity analysis for them is bound to be done by reference to low level implementations with an explicit notion of constant-time computation step. Indeed, the basic step of computation for rule-based programming is not a constant-time operation, being it resolution, pattern matching, term-rewriting, higher-order beta-reduction, and so on. Therefore, to bound the actual execution time, one has to look at specific implementations of these mechanisms—in many cases, to revert to imperative programming and to an analysis on Turing machines.

In this paper we focus on (orthogonal) term rewriting systems and the problem of verifying the existence of a bound on the time needed to execute a program on any input, as a function of the input's length. Or, to devise sound (and necessarily incomplete) static techniques to infer such bounds. While real execution time clearly depends on the interpreter, for asymptotic bounds the details on the underlying interpreter become less crucial. More important, those details *should be* less crucial. For a sufficiently robust complexity class $\mathscr{P}$ (say, the polynomial-, or the elementary-time computable functions) proving that a given program may be executed in time bounded by a function in $\mathscr{P}$ should be the same whichever cost model is chosen, provided such a model is *invariant* [16] — that is, the cost attributed to the computation by the model differs by at most a polynomial from the cost of performing an equivalent computation on a Turing machine. And the simpler the cost model is, the easier is proving the existence of such bounds.

⋆ The authors are partially supported by PRIN project "CONCERTO" and FIRB grant RBIN04M8S8, "Intern. Inst. for Applicable Math."

If programs are defined by rewriting, the most natural cost model for computation time is the one induced by rewriting. Time passes whenever rewriting is performed and, more importantly, the time needed to fire a redex is *assumed* to be unitary. Indeed, several recent papers propose methodologies for proving asymptotic bounds (typically, polynomial bounds) on the *derivational complexity* of term rewriting systems, where the derivational complexity of a term t is the number of rewriting steps necessary to reduce t to its normal form.

Is this cost model invariant? Or, which is the relation between the derivational complexity of a term and the time needed to rewrite it to normal form, observed on an efficient interpreter? The question is not trivial, even if a positive answer to the invariance issue seems part of the folklore. Indeed, the literature often distinguishes between results about computational complexity on the one hand and results about derivational complexity on the other. And there are examples of rewriting systems which produce exponentially big terms in a linear number of steps. For example, the one defined by the following rewrite rules (under innermost reduction): $\mathbf{f}(0) \to \mathbf{h}$, $\mathbf{f}(n+1) \to \mathbf{g}(\mathbf{f}(n))$, and $\mathbf{g}(x) \to \mathbf{p}(x,x)$.

Aim of this paper is to fill this gap, at least partially. More precisely, we prove that terms of orthogonal term rewriting systems can be reduced to their normal form in time polynomially related to their derivational complexity, both when innermost and outermost reduction is adopted. We prove this by showing that any such rewriting system can be implemented with (term) graph rewriting. In this setting, whenever a rewriting causes a duplication of a subterm (because of a non right-linear variable in a rule), the subgraph corresponding to the duplicated term is *shared* and not duplicated. This achieves two goals. First (and most important) the *size* of all the (graphs corresponding to) terms generated along the reduction of a term t remains under control—it is polynomial in the size of t and the number of reduction steps leading t to its normal form (see Section 5). Moreover, the actual cost of manipulating these graphs is bounded by a polynomial, thus giving also a polynomial relation between the number of reduction steps and the cost of producing the graph in normal form. Since we show how graph reduction simulates term rewriting *step by step* under innermost reduction, this gives us the desired invariance results in the innermost case.

In outermost reduction the situation is even better, because in presence of sharing every graph rewriting step corresponds to *at least* one term rewriting step. In graphs, in fact, shared redexes are reduced only once and, moreover, any redex appearing in an argument which will be later discarded, will not be reduced at all. Therefore, in presence of sharing outermost reduction becomes a call-by-need strategy. And hence the invariance result also holds for outermost reduction.

Again, we believe the central argument and results of this paper may be classified as folklore (see, for instance, results of similar flavor, but for imperative programs with data-sharing, in Jones' computability textbook [10]). But we were struck by the observation that in the published literature it seems that such complexity related issues are never made explicit or used.

2 Term Rewriting

We will consider in this paper orthogonal term rewriting systems (TRS, see [5]). Let Υ be a denumerable set of variables. A term rewriting system is a pair $\Xi = (\Sigma_\Xi, \mathcal{R}_\Xi)$ where:

- Symbols in the signature Σ_Ξ come with an *arity* and are usually called *function symbols* or simply *functions*. Moreover:
  - Terms in $\mathcal{T}(\Xi)$ are those built from function symbols and are called *closed terms*.
  - Terms in $\mathcal{V}(\Xi, \Upsilon)$ are those built from functions symbols and variables in Υ and are dubbed *terms*.
- Rules in $\mathcal{R}_\Xi$ are in the form $t \rightarrow_\Xi u$ where both t and u are in $\mathcal{V}(\Xi, \Upsilon)$. We here consider orthogonal rewriting systems only, i.e. we assume that no distinct two rules in $\mathcal{R}_\Xi$ are overlapping and that every variable appears at most once in the lhs of any rule in $\mathcal{R}_\Xi$.
- Different notions of reduction can be defined on Ξ. The (binary) *rewriting relation* $\rightarrow$ on $\mathcal{T}(\Xi)$ is defined by imposing that $t \rightarrow u$ iff there are a rule $v \rightarrow_\Xi w$ in $\mathcal{R}_\Xi$, a term context (i.e., a term with a hole) C and a substitution σ with $t = C[v\sigma]$ and $u = C[w\sigma]$. Two restrictions of this definition are innermost and outermost reduction. In the *innermost rewriting relation* $\rightarrow_{\mathsf{i}}$ on $\mathcal{T}(\Xi)$ we require that $v\sigma$ do not contain another redex. Dually, the *outermost rewriting relation* $\rightarrow_{\mathsf{o}}$ on $\mathcal{T}(\Xi)$ is defined by requiring (the specific occurrence of) $v\sigma$ not to be part of another redex in $C[v\sigma]$.

For any term t in a TRS, $|t|$ denotes the number of symbol occurrences in t, while $|t|_{\mathbf{f}}$ denotes the number of occurrences of the symbol $\mathbf{f}$ in t.

Orthogonal TRSs are confluent but not necessarily strongly confluent [5]. As a consequence, different reduction sequences may have different lengths. This does not hold when considering only outermost (or only innermost) reductions:

Proposition 1. *Given a term t, every innermost (outermost, respectively) reduction sequence leading t to its normal form has the same length.*

Proof. Immediate, since by definition there cannot be any nesting of redex while performing either innermost or outermost reduction. In innermost reduction, if a redex t is part of another redex u, then only u can be fired, by definition. As a consequence, both $\rightarrow_{\mathsf{o}}$ and $\rightarrow_{\mathsf{i}}$ are locally confluent. $\square$

As a consequence, it is meaningful to define the *outermost derivational complexity* of any t, written $\mathit{Time}_{\mathsf{o}}(t)$ as the unique n (if any) such that $t \rightarrow_{\mathsf{o}}^n u$, for u a normal form. Similarly, the *innermost derivational complexity* of t is the unique n such that $t \rightarrow_{\mathsf{i}}^n u$ (if any) and is denoted as $\mathit{Time}_{\mathsf{i}}(t)$.

3 Graph Rewriting

In this Section, we introduce term graph rewriting, following [4].

Definition 1 (Labelled Graph). *Given a signature Σ, a* labelled graph *over Σ consists of a directed acyclic graph together with an ordering on the outgoing edges of each node and a (partial) labelling of nodes with symbols from Σ such that the out-degree of each node matches the arity of the corresponding symbols (and is* 0 *if the labelling is undefined). Formally, a labelled graph is a triple $G = (V, \alpha, \delta)$ where:*

- *V is a set of* vertices*.*
- *$\alpha : V \to V^*$ is a (total)* ordering function*.*
- *$\delta : V \rightharpoonup \Sigma$ is a (partial)* labelling function *such that the length of $\alpha(v)$ is the arity of $\delta(v)$ if $\delta(v)$ is defined and is* 0 *otherwise.*

A labelled graph (V, α, δ) is closed *iff δ is a total function.*

Consider the signature $\Sigma = \{\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{p}\}$, where arities of $\mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{p}$ are 2, 1, 0 and 2, respectively. Examples of labelled graphs over the signature Σ are the following ones:

$$\begin{tikzcd}
 & \mathbf{f} \arrow[d, bend right] \arrow[d, bend left] & \\
 & \mathbf{g} \arrow[d] & \\
 & \mathbf{p} \arrow[dl] \arrow[dr] & \\
\mathbf{g} \arrow[d] & & \mathbf{h} \\
\perp & &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{f} \arrow[dr, bend right] \arrow[dr, bend left] & & \mathbf{g} \arrow[dl] \\
 & \perp &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \mathbf{f} \arrow[d] \arrow[dd, bend right] & \\
 & \mathbf{g} \arrow[d] & \\
 & \mathbf{f} \arrow[dl] \arrow[dr] & \\
\perp & & \mathbf{g} \arrow[d] \\
 & & \perp
\end{tikzcd}$$

The symbol $\perp$ denotes vertices where the underlying labelling function is undefined (and, as a consequence, no edge departs from such vertices). Their role is similar to the one of variables in terms.

If one of the vertices of a labelled graph is selected as the *root*, we obtain a term graph:

Definition 2 (Term Graphs). *A* term graph*, is a quadruple $G = (V, \alpha, \delta, r)$, where (V, α, δ) is a labelled graph and $r \in V$ is the* root *of the term graph.*

The following are graphic representations of some term graphs. The root is the only vertex drawn inside a circle.

$$\begin{tikzcd}
 & \text{\textcircled{\textbf{f}}} \arrow[d, bend right] \arrow[d, bend left] & \\
 & \mathbf{g} \arrow[d] & \\
 & \mathbf{f} \arrow[dl] \arrow[dr] & \\
\mathbf{g} \arrow[d] & & \mathbf{h} \\
\perp & &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbf{f} \arrow[dr, bend right] \arrow[dr, bend left] & & \text{\textcircled{\textbf{g}}} \arrow[dl] \\
 & \perp &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \text{\textcircled{\textbf{f}}} \arrow[d] \arrow[dd, bend right] & \\
 & \mathbf{g} \arrow[d] & \\
 & \mathbf{f} \arrow[dl] \arrow[dr] & \\
\perp & & \mathbf{g} \arrow[d] \\
 & & \perp
\end{tikzcd}$$

The notion of a homomorphism between labelled graphs is not only interesting mathematically, but will be crucial in defining rewriting:

Definition 3 (Homomorphisms). *A* homomorphism *between two labelled graphs $G = (V_G, \alpha_G, \delta_G)$ and $H = (V_H, \alpha_H, \delta_H)$ over the same signature Σ is a function φ from V_G to V_H preserving the labelled graph structure. In particular*

$$\delta_H(\varphi(v)) = \delta_G(v)$$
$$\alpha_H(\varphi(v)) = \varphi^*(\alpha_G(v))$$

for any $v \in dom(\delta_G)$, where φ^* is the obvious generalization of φ to sequences of vertices. A homomorphism *between two term graphs $G = (V_G, \alpha_G, \delta_G, r_G)$ and $H = (V_H, \alpha_H, \delta_H, r_H)$ is a homomorphism between $(V_G, \alpha_G, \delta_G)$ and $(V_H, \alpha_H, \delta_H)$ such that $\varphi(r_G) = r_H$. Two labelled graphs G and H are isomorphic iff there is a bijective homomorphism from G to H; in this case, we write $G \cong H$. Similarly for term graphs.*

In the following, we will consider term graphs modulo isomorphism, i.e., $G = H$ iff $G \cong H$. Observe that two isomorphic term graphs have the same graphical representation.

Definition 4 (Graph Rewrite Rules). *A* graph rewrite rule *over a signature Σ is a triple $\rho = (G, r, s)$ such that:*

- *G is a labelled graph;*
- *r, s are vertices of G, called the* left root *and the* right root *of ρ, respectively.*

The following are three examples of graph rewrite rules:

$$\begin{tikzcd}
& \textcircled{$\mathbf{f}$} \arrow[d, bend right] \arrow[d, bend left] & \\
& \mathbf{g} \arrow[d] & \\
& \mathbf{p} \arrow[dl] \arrow[dr] & \\
\mathbf{g} \arrow[d] & & \mathbf{h} \\
\boxed{\perp} & &
\end{tikzcd}
\qquad
\begin{tikzcd}
\textcircled{$\mathbf{f}$} \arrow[dr, bend right] \arrow[dr, bend left] & & \boxed{\mathbf{g}} \arrow[dl] \\
& \perp &
\end{tikzcd}
\qquad
\begin{tikzcd}
& \textcircled{$\mathbf{f}$} \arrow[dl] \arrow[dr] & & \boxed{\mathbf{h}} \\
\mathbf{g} \arrow[d] & & \mathbf{g} \arrow[d] & \\
\perp & & \perp &
\end{tikzcd}$$

Graphically, the left root is the (unique) node inside a circle, while the right root is the (unique) node inside a square.

Definition 5 (Subgraphs). *Given a labelled graph $G = (V_G, \alpha_G, \delta_G)$ and any vertex $v \in V_G$, the* subgraph of G rooted at v, *denoted $G \downarrow v$, is the term graph $(V_{G\downarrow v}, \alpha_{G\downarrow v}, \delta_{G\downarrow v}, r_{G\downarrow v})$ where*

- *$V_{G\downarrow v}$ is the subset of V_G whose elements are vertices which are reachable from v in G.*
- *$\alpha_{G\downarrow v}$ and $\delta_{G\downarrow v}$ are the appropriate restrictions of α_G and δ_G to $V_{G\downarrow v}$.*

- *$r_{G\downarrow v}$ is v.*

We are finally able to give the notion of a redex, that represents the occurrence of the lhs of a rewrite rule in a graph:

Definition 6 (Redexes). *Given a labelled graph G, a* redex *for G is a pair (ρ, φ), where ρ is a rewrite rule (H, r, s) and φ is a homomorphism between $H \downarrow r$ and G.*

If $((H, r, s), \varphi)$ is a redex in G, we say, with a slight abuse of notation, that $\varphi(r)$ *is* itself a redex. In most cases, this does not introduce any ambiguity.

Given a term graph G and a redex $((H, r, s), \varphi)$, the result of firing the redex is another term graph obtained by successively applying the following three steps to G:

1. The *build phase*: create an isomorphic copy of the portion of $H \downarrow s$ not contained in $H \downarrow r$ (which may contain arcs originating in $H \downarrow s$ and entering $H \downarrow r$), and add it to G, obtaining J. The underlying ordering and labelling functions are defined in the natural way.
2. The *redirection phase*: all edges in J pointing to $\varphi(r)$ are replaced by edges pointing to the copy of s. If $\varphi(r)$ is the root of G, then the root of the newly created graph will be the newly created copy of s. The graph K is obtained.
3. The *garbage collection phase*: all vertices which are not accessible from the root of K are removed. The graph I is obtained.

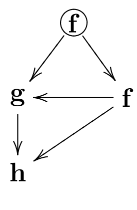
We will write $G \overset{(H,r,s)}{\longrightarrow} I$ (or simply $G \to I$, if this does not cause ambiguity) in this case.

Similarly to what we did for term rewriting, we can define two restrictions on $\to$ as follows. Let $((H, r, s), \varphi)$ be a redex in G. Then it is said to be

- An *innermost* redex iff for every redex $((J, p, q), \psi)$ in G, there is no proper path from $\varphi(r)$ to $\psi(p)$.
- An *outermost* redex iff for every redex $((J, p, q), \psi)$ in G, there is no proper path from $\psi(p)$ to $\varphi(r)$.

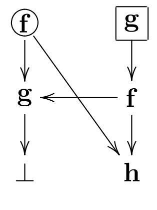
If the redex $((H, r, s), \varphi)$ is innermost we also write $G \overset{(H,r,s)}{\longrightarrow}_{\mathsf{i}} I$ or $G \to_{\mathsf{i}} I$. Similarly, for an outermost redex $((H, r, s), \varphi)$ we write $G \overset{(H,r,s)}{\longrightarrow}_{\mathsf{o}} I$ or simply $G \to_{\mathsf{o}} I$.

As an example, consider the term graph G and the rewrite rule $\rho = (H, r, s)$:



$$\begin{tikzcd}
 & \textcircled{\mathbf{f}} \arrow[dl] \arrow[dr] & \\
\mathbf{g} \arrow[d] & & \mathbf{f} \arrow[ll] \arrow[dll] \\
\mathbf{h} & &
\end{tikzcd}$$

G



$$\begin{tikzcd}
\textcircled{\mathbf{f}} \arrow[d] \arrow[ddr] & \boxed{\mathbf{g}} \arrow[d] \\
\mathbf{g} \arrow[d] & \mathbf{f} \arrow[l] \arrow[d] \\
\perp & \mathbf{h}
\end{tikzcd}$$

ρ

There is a homomorphism φ from $H \downarrow r$ to G. In particular, φ maps r to the rightmost vertex in G. Applying the build phase and the redirection phase we get J and K as follows:

$$\begin{tikzcd}
& \textcircled{\mathbf{f}} \arrow[dl] \arrow[dr] & & \mathbf{g} \arrow[d] \\
\mathbf{g} \arrow[d] & & \mathbf{f} \arrow[ll] \arrow[dll] & \mathbf{f} \arrow[lll, bend right] \arrow[dlll, bend left] \\
\mathbf{h} & & &
\end{tikzcd}
\qquad
\begin{tikzcd}
& \textcircled{\mathbf{f}} \arrow[dl] \arrow[rr] & & \mathbf{g} \arrow[d] \\
\mathbf{g} \arrow[d] & & \mathbf{f} \arrow[ll] \arrow[dll] & \mathbf{f} \arrow[lll, bend right] \arrow[dlll, bend left] \\
\mathbf{h} & & &
\end{tikzcd}$$

J K

Finally, applying the garbage collection phase, we get the result of firing the redex (ρ, φ):

$$\begin{tikzcd}
& \textcircled{\mathbf{f}} \arrow[d] \arrow[dddl, bend right] & \\
& \mathbf{g} \arrow[d] & \\
& \mathbf{f} \arrow[dl] \arrow[dr] & \\
\mathbf{g} \arrow[rr] & & \mathbf{h}
\end{tikzcd}$$

I

Given two graph rewrite rules $\rho = (H, r, s)$ and $\sigma = (J, p, q)$, ρ and σ are said to be *overlapping* iff there is a term graph G and two homomorphism φ and ψ such that (ρ, φ) and (σ, ψ) are both redexes in G with $z = \varphi(v) = \psi(w)$, where v is labelled and reachable from r, while w is labelled and reachable from s.

Definition 7. *An orthogonal graph rewriting system (GRS) over a signature Σ consists of a set of non-overlapping graph rewrite rules $\mathcal{G}$ on Σ.*

We now want to give some confluence results for GRSs. Let us first focus on outermost rewriting. Intuitively, outermost rewriting is the most efficient way of performing reduction in presence of sharing, since computation is performed only if its result does not risk to be erased. First, we need the following auxiliary lemma:

Lemma 1. *Suppose $G \rightarrow_{\circ} H$ and $G \rightarrow J$, where $H \neq J$. Then either $J \rightarrow_{\circ} H$ or there is K such that $H \rightarrow K$ and $J \rightarrow_{\circ} K$.*

Proof. Let v and w be the two redexes in G giving rise to $G \rightarrow_{\circ} H$ and $G \rightarrow J$, respectively. Similarly, let ρ and σ be the two involved rewrite rules. Clearly, there cannot be any (non-trivial) path from w to v, by definition of outermost rewriting. Now, the two rewriting steps are independent from each other (because of the non-overlapping condition). There are now two cases. Either w is erased when performing $G \rightarrow_{\circ} H$, or it is not erased. In the first case, w must be "contained" in v, and therefore, we may apply ρ to J, obtaining H. If w has not been erased, one can clearly apply ρ to J and σ to H, obtaining a fourth graph K. $\square$

The observation we just made can be easily turned into a more general result on reduction sequences of arbitrary length:

Proposition 2. *Suppose that $G \rightarrow_{\mathsf{o}}^n H$ and $G \rightarrow^m J$. Then there are K and $k, l \in \mathbb{N}$ such that $H \rightarrow^k J$, $J \rightarrow_{\mathsf{o}}^l K$ and $n + k \leq m + l$.*

Proof. An easy induction on $n + m$. □

Proposition 2 tells us that if we perform n outermost steps and m generic steps from G, we can close the diagram in such a way that the number of steps in the first branch is smaller or equal to the number of steps in the second branch.

With innermost reduction, the situation is exactly dual:

Lemma 2. *Suppose $G \rightarrow H$ and $G \rightarrow_{\mathsf{i}} J$, where $H \neq J$. Then either $J \rightarrow H$ or there is K such that $H \rightarrow_{\mathsf{i}} K$ and $J \rightarrow K$.*

Proposition 3. *Suppose that $G \rightarrow^n H$ and $G \rightarrow_{\mathsf{i}}^m J$. Then there are K and $k, l \in \mathbb{N}$ such that $H \rightarrow_{\mathsf{i}}^k J$, $J \rightarrow^l K$ and $n + k \leq m + l$.*

In presence of sharing, therefore, outermost reduction is the best one can do, while innermost reduction is the worst strategy, since we may reduce redexes in subgraphs that will be later discarded. As a by-product, we get confluence:

Theorem 1. *Suppose that $G \rightarrow_{\mathsf{o}}^n H$, $G \rightarrow^m J$ and $G \rightarrow_{\mathsf{i}}^k K$, where H, J and K are normal forms. Then $H = J = K$ and $n \leq m \leq k$.*

Proof. From $G \rightarrow_{\mathsf{o}}^n H$, $G \rightarrow^m J$ and Proposition 2, it follows that $n \leq m$ and that $H = J$. From $G \rightarrow_{\mathsf{o}}^m J$, $G \rightarrow^k K$ and Proposition 3, it follows that $m \leq k$. □

4 From Term Rewriting to Graph Rewriting

Any term t over the signature Σ can be turned into a graph G in the obvious way: G will be a tree and vertices in G will be in one-to-one correspondence with symbol occurrences in t. Conversely, any term graph G over Σ can be turned into a term t over Σ (remember: we only consider acyclic graphs here).

Similarly, any term rewrite rule $t \rightarrow u$ over the signature Σ can be translated into a graph rewrite rule (G, r, s) as follows:

- Take the graphs representing t and u. They are trees, in fact.
- From the union of these two trees, share those nodes representing the same variable in t and u. This is G.
- Take r to be the root of t in G and s to be the root of u in G.

As an example, consider the rewrite rule

$$\mathbf{f}(\mathbf{g}(x), y) \rightarrow \mathbf{g}(\mathbf{f}(y, \mathbf{f}(y, x))).$$

Its translation as a graph rewrite rule is the following:

$$\begin{tikzcd}
& \textcircled{\mathbf{f}} \arrow[dl] \arrow[dr] & & \boxed{\mathbf{g}} \arrow[d] & \\
\mathbf{g} \arrow[d] & & \perp & \mathbf{f} \arrow[l] \arrow[dr] & \\
\perp & & & & \mathbf{f} \arrow[ull, bend left] \arrow[llll, bend left]
\end{tikzcd}$$

An arbitrary term rewriting system can be turned into a graph rewriting system:

Definition 8. *Given a term rewriting system $\mathcal{R}$ over Σ, the corresponding term graph rewriting system $\mathcal{G}$ is defined as the class of graph rewrite rules corresponding to those in $\mathcal{R}$. Given a term t, $[t]_{\mathcal{G}}$ will be the corresponding graph, while the term graph G corresponds to the term $\langle G\rangle_{\mathcal{R}}$.*

Let us now consider graph rewrite rules corresponding to rewrite rules in $\mathcal{R}$. It is easy to realize that the following invariant is preserved while performing innermost rewriting in $[\mathcal{R}]_{\mathcal{G}}$: whenever any vertex v can be reached by two distinct paths starting at the root (i.e., v is *shared*), v cannot be a redex, i.e., there cannot be a redex $((G, r, s), \varphi)$ such that $\varphi(r) = v$. A term graph satisfying this invariant is said to be *redex-unshared*.

Redex-unsharedness holds for term graphs coming from terms and is preserved by innermost graph rewriting:

Lemma 3. *For every closed term t, $[t]_{\mathcal{G}}$ is redex-unshared. Moreover, if G is closed and redex-unshared and $G \rightarrow_{\mathsf{i}} I$, then I is redex-unshared.*

Proof. The fact $[t]_{\mathcal{G}}$ is redex-unshared for every t follows from the way the $[\cdot]_{\mathcal{G}}$ map is defined: it does not introduce any sharing. Now, suppose G is redex-unshared and

$$G \xrightarrow{(H,r,s)}_{\mathsf{i}} I$$

where (H, r, s) corresponds to a term rewriting rule $t \rightarrow u$. The term graph J obtained from G by the build phase is itself redex-unshared: it is obtained from G by adding some new nodes, namely an isomorphic copy of the portion of $H \downarrow s$ not contained in $H \downarrow r$. Notice that J is redex-unshared in a stronger sense: any vertex which can be reached from the newly created copy of s by two distinct paths cannot be a redex. This is a consequence of (H, r, s) being a graph rewrite rule corresponding to a term rewriting rule $t \rightarrow u$, where the only shared vertices are those where the labelling function is undefined. The redirection phase preserves itself redex-unsharedness, because only one pointer is redirected (the vertex is labelled by a function symbol) and the destination of this redirection is a vertex (the newly created copy of s) which had no edge incident to it. Clearly, the garbage collection phase preserve redex-unsharedness. □

Lemma 4. *A closed term graph G in $\mathcal{G}$ is a normal form iff $\langle G\rangle_{\mathcal{R}}$ is a normal form.*

Proof. Clearly, if a closed term graph G is in normal form, then $\langle G\rangle_{\mathcal{R}}$ is a term in normal form, because each redex in G translates to a redex in $\langle G\rangle_{\mathcal{R}}$. On the other hand, if $\langle G\rangle_{\mathcal{R}}$ is in normal form, then G must be normal itself: each redex in $\langle G\rangle_{\mathcal{R}}$ translates back to a redex in G. $\square$

Reduction on graphs correctly simulates reduction on terms:

Lemma 5. *If $G \to I$, then $\langle G\rangle_{\mathcal{R}} \to^{+} \langle I\rangle_{\mathcal{R}}$. Moreover, if $G \to_{\mathsf{i}} I$ and G is redex-unshared, then $\langle G\rangle_{\mathcal{R}} \to \langle I\rangle_{\mathcal{R}}$.*

Proof. The fact each reduction step starting in G can be mimicked by n reduction steps in $\langle G\rangle_{\mathcal{R}}$ is known from the literature. If G is redex-unshared, then $n = 1$, because no redex in a redex-unshared term graph can be shared. $\square$

As an example, consider the term rewrite rule $\mathbf{f}(\mathbf{h},\mathbf{h}) \to \mathbf{h}$ and the following term graph, which is not redex-unshared and correspond to $\mathbf{f}(\mathbf{f}(\mathbf{h},\mathbf{h}),\mathbf{f}(\mathbf{h},\mathbf{h}))$:

$$\begin{tikzcd}
& \textcircled{\mathbf{f}} \arrow[d, bend right] \arrow[d, bend left] & \\
& \mathbf{f} \arrow[dl] \arrow[dr] & \\
\mathbf{h} & & \mathbf{h}
\end{tikzcd}$$

The term graph rewrites in *one* step to the following one

$$\begin{tikzcd}
\textcircled{\mathbf{f}} \arrow[d, bend right] \arrow[d, bend left] \\
\mathbf{h}
\end{tikzcd}$$

while the term $\mathbf{f}(\mathbf{f}(\mathbf{h},\mathbf{h}),\mathbf{f}(\mathbf{h},\mathbf{h}))$ rewrites to $\mathbf{f}(\mathbf{h},\mathbf{h})$ in *two* steps.

Lemma 6. *If $t \to_{\mathsf{o}}^{n} u$, u is in normal form and $\langle G\rangle_{\mathcal{R}} = t$, then there is $m \leq n$ such that $G \to_{\mathsf{o}}^{m} I$, where $\langle I\rangle_{\mathcal{R}} = u$.*

Proof. An easy consequence of Lemma 5 and Proposition 1. $\square$

Theorem 2 (Outermost Graph-Reducibility). *For every orthogonal term rewriting system $\mathcal{R}$ over Σ and for every term t over Σ, the following two conditions are equivalent:*
1. *$t \to_{\mathsf{o}}^{n} u$, where u is in normal form;*
2. *$[t]_{\mathcal{G}} \to_{\mathsf{o}}^{m} G$, where G is in normal form and $\langle G\rangle_{\mathcal{R}} = u$.*

Moreover, $m \leq n$.

Proof. Suppose $t \to_{\mathsf{o}}^{n} u$, where u is in normal form. Then, by applying Lemma 6, we obtain a normal form G such that $[t]_{\mathcal{G}} \to_{\mathsf{o}}^{m} G$, where $m \leq n$ and $\langle G\rangle_{\mathcal{R}} = u$. Now, suppose $[t]_{\mathcal{G}} \to_{\mathsf{o}}^{m} G$ where $\langle G\rangle_{\mathcal{R}} = u$ and G is in normal form. By applying n times Lemma 5, we obtain that $\langle [t]_{\mathcal{G}}\rangle_{\mathcal{R}} \to^{n} \langle G\rangle_{\mathcal{R}} = u$ where $m \leq n$. But $\langle [t]_{\mathcal{G}}\rangle_{\mathcal{R}} = t$ and u is a normal form by Lemma 4, since G is normal. $\square$

The innermost case can be treated in a similar way:

Lemma 7. *If $t \rightarrow_{\mathsf{i}}^{n} u$, u is in normal form and $\langle G\rangle_{\mathcal{R}} = t$ and G is redex-unshared, then $G \rightarrow_{\mathsf{i}}^{n} I$, where $\langle I\rangle_{\mathcal{R}} = u$.*

Proof. An easy consequence of Lemma 5 and Proposition 1. □

Theorem 3 (Innermost Graph Reducibility). *For every orthogonal term rewriting system $\mathcal{R}$ over Σ and for every term t over Σ, the following two conditions are equivalent:*
1. *$t \rightarrow_{\mathsf{i}}^{n} u$, where u is in normal form;*
2. *$[t]_{\mathcal{G}} \rightarrow_{\mathsf{i}}^{n} G$, where G is in normal form and $\langle G\rangle_{\mathcal{R}} = u$.*

5 Consequences for Complexity Analysis

Theorems 2 and 3 tell us that term graph rewriting faithfully simulates term rewriting, with both outermost and innermost rewriting. In the outermost case, graph rewriting may perform better than term rewriting, because redex can be shared and one graph rewriting step may correspond to more than one term rewriting step. In innermost reduction, on the other hand, every graph step corresponds to exactly one term rewriting step.

But how much does it cost to perform reduction in a graph rewriting system $\mathcal{G}$ corresponding to a term rewriting system $\mathcal{R}$? Let us analyze more closely the combinatorics of graph rewriting, fixing our attention to outermost rewriting for the moment:

- Consider a closed term t and a term graph G such that $[t]_{\mathcal{G}} \rightarrow_{\mathsf{o}}^{*} G$.
- Every graph rewriting step makes the underlying graph bigger by at most the size of the rhs of a rewrite rule. So, if $[t]_{\mathcal{G}} \rightarrow_{\mathsf{o}}^{*} G \rightarrow_{\mathsf{o}} H$, then the difference $|H| - |G|$ cannot be too big: at most a constant k depending on $\mathcal{R}$ but independent of t. As a consequence, if $[t]_{\mathcal{G}} \rightarrow_{\mathsf{o}}^{n} G$ then $|G| \leq nk + |t|$. Here, we exploit in an essential way the possibility of sharing subterms.
- Whenever $[t]_{\mathcal{G}} \rightarrow_{\mathsf{o}}^{n} G$, computing a graph H such that $G \rightarrow H$ takes polynomial time in $|G|$, which is itself polynomially bounded by n and $|t|$.

Exactly the same reasoning can be applied to innermost reduction. Hence:

Theorem 4. *For every orthogonal term rewriting system $\mathcal{R}$, there is a polynomial $p : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for every term t the normal form of $[t]_{\mathcal{G}}$ can be computed in time at most $p(|t|, \mathit{Time}_{\mathsf{o}}(t))$ when performing outermost graph reduction and in time $p(|t|, \mathit{Time}_{\mathsf{i}}(t))$ when performing innermost graph reduction.*

We close this section by observing explicitly that the normal form of $[t]_{\mathcal{G}}$ is not a direct representation of the normal form of t. It may contain many shared subterms, that have to be "unshared" if one wants to print the normal form of t. As a limit example consider the system we already mentioned in the introduction: $f(0) = c$, $f(n+1) = g(f(n))$, and $g(x) = d(x, x)$. Here $f(n)$ will normalize in $O(n)$ steps with innermost reduction, but the normal form *as a term* is of size $O(2^n)$, being the complete binary tree of height n. We believe this has to be considered a feature of our cost model, allowing to distinguish the time (and space) needed for the computation from the one necessary for the communication of the result.

Despite the succinct representation of data via term graphs, equality of terms is efficiently computed on graph representatives. We state this as a proposition, being of interest in its own.

Proposition 4. *Given two term graphs G and H, it is decidable in time polynomial in $|G| + |H|$ whether $\langle G\rangle_{\mathcal{R}} = \langle H\rangle_{\mathcal{R}}$.*

Proof. We can give a naive procedure working in quadratic time as follows. More efficient algorithms are available, for instance Paige and Tarjan's one for bisimulation, which runs in time $O(|E| \log |V|)$, where E and V are the sets of edges and vertices, respectively, of the graphs.

The decision procedure will fill a $m \times n$ matrix, with m and n the number of nodes of G and H, respectively, using dynamic programming. Any element will contain a boolean value 1 or 0. Start by filling all the elements (v_G, v_H) where v_G is a sink of G and v_H is a sink of H (a sink is a node labeled with a function symbol of arity 0). Fill it with 1 if they are the same function symbol; with 0 otherwise. Now proceed along the inverse of the topological order (that is, go to the nodes pointing to the ones you just considered), and fill any such element (w_G, w_H) with 1, if they are the same function symbol *and* all the pairs (v_G, v_H) — with v_G i-th successor of w_G and v_H i-th successor of w_H — are marked with 1. Otherwise, fill (w_G, w_H) with 0. At the end return 1 iff (r_G,r_H) is marked with 1 where r_G and r_H are the root of G and H, respectively. □

6 Context and Related Work

Graph-reducibility of any orthogonal term rewriting system is well known [14]. However, this classical result does not mention anything about the relation between the complexity of term rewriting and the one of graph-rewriting. Quantitative analysis of the involved simulations is outside the scope of the classical results on the subject.

Asymptotic complexity analysis in the context of term rewriting systems has received a lot of attention in the last ten years [**?**,**?**,3, 9]. In some cases time complexity results are a consequence of an argument taking into account both the number of reduction steps and some other parameter (e.g., the size of intermediate terms [**?**]), so to bound the actual cost of the computation with an *ad hoc* combination of these two dimensions. In other cases [3, 9], results about the derivational complexity of TRS are kept distinct from other results about actual computation time. This body of research has been the main motivation for our work.

In a recent paper [7] we proved a close correspondence between orthogonal constructor term rewriting systems and weak call-by-value λ-calculus. In particular the two systems can simulate each other with a linear overhead, taking as cost model for both systems the number of reduction steps to normal form, that is the most natural one. This should not confuse the reader who knows that "optimal λ-reduction is not elementary recursive" [2, 1], meaning that there are terms whose normalization requires on a Turing machine a time hyperexponential in the number of optimal beta-reductions (which are a sophisticated form of graph-rewritings with partial sharing). For these results to hold is essential to take *full* beta-reduction, where we are allowed to reduce a redex also inside a λ-abstraction.

Graph rewriting has been considered in this paper as a technical tool to obtain our main result. An interesting research line would be to situate graph rewriting – and its complexity theory – in the context of those other machine models with a dynamically changing configuration structure, like Knuth's "linking automata" [11], Schönage's storage modification machines [15], and especially their common moral ancestor — Kolmogorov and Uspensky's machines [12, 8]. This would be particularly interesting in the study of classes like linear time and real time. Indeed, while the class of polynomial functions is very robust and coincide on all these different models (and on Turing machines, of course), these automata seem to give a better understanding of the lower complexity classes. After some preliminary investigation, the authors are convinced that the task of relating term graph rewriting and pointer machines from a complexity point of view is not trivial. For example, garbage collection is a non-local operation that is implicitly performed as part of any term graph rewriting step, while in pointer machines only the "programmer" is responsible for such (potentially costly) operations.

Acknowledgments

We owe to Guillaume Bonfante the observation that our results, previously formulated for *constructor* orthogonal rewriting systems, hold indeed for any orthogonal system, as it is now stated in the paper.

References

1. A. Asperti, P. Coppola, and S. Martini. (Optimal) duplication is not elementary recursive. *Inf. Comput.*, 193(1):21–56, 2004.
2. A Asperti and H. G. Mairson. Parallel beta reduction is not elementary recursive. *Inf. Comput.*, 170(1):49–80, 2001.
3. Martin Avanzini and Georg Moser. Complexity analysis by rewriting. In *FLOPS*, pages 130–146, 2008.
4. H. Barendregt, M. Eekelen, J. Glauert, J. Kennaway, M. Plasmeijer, and M. Sleep. Term graph rewriting. In J. de Bakker, A. Nijman, and P. Treleaven, editors, *Volume II: Parallel Languages on PARLE: Parallel Architectures and Languages Europe*, pages 141–158. Springer-Verlag, 1986.
5. Erik Barendsen. Term graph rewriting. In Terese (M. Bezem, J.W. Klop, and R. de Vrijer), editors, *Term Rewriting Systems*, chapter 13, pages 712–743. Cambridge Univ. Press, 2003.

6. Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. Quasi-interpretations and small space bounds. In *Term Rewriting and Applications*, volume 3467 of *LNCS*, pages 150–164. Springer, 2005.
7. Ugo Dal Lago and Simone Martini. On constructor rewriting systems and the lambda calculus. In *ICALP*, volume 5556 of *LNCS*, pages 163–174. Springer, 2009.
8. Yuri Gurevich. On Kolmogorov machines and related issues. *Bulletin of the European Association for Theoretical Computer Science*, 35:71–82, 1988.
9. Nao Hirokawa and Georg Moser. Automated complexity analysis based on the dependency pair method. In *IJCAR*, pages 364–379, 2008.
10. Neil D. Jones. *Computability and Complexity from a Programming Perspective*. MIT Press, 1997.
11. Donald Knuth. *The Art of Computer Programming, Vol. 1.* Prentice Hall, 1973. Pages 462-463.
12. A.N. Kolmogorov and V. Uspensky. On the definition of algorithm. *Uspekhi Mat. Naut*, 13(4):3–28, 1958. In Russian. English translation in AMS Translations, series 2, vol. 21(1963), 217-245.
13. Jean-Yves Marion and Jean-Yves Moyen. Efficient first order functional program interpreter with time bound certifications. In *Logic for Programming and Automated Reasoning, 7th International Conference, Proceedings*, volume 1955 of *LNCS*, pages 25–42. Springer, 2000.
14. Detlef Plump. Graph-reducible term rewriting systems. In *Graph-Grammars and Their Application to Computer Science*, pages 622–636, 1990.
15. A. Schönage. Storage modification machines. *SIAM J. Comput.*, 9:490–508, 1980.
16. Peter van Emde Boas. Machine models and simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 1–66. MIT Press, 1990.