Parallel Complexity of Term Rewriting Systems

Thaïs Baudon, Carsten Fuhs, Laure Gonnord

Parallel Complexity of Term Rewriting Systems*

Thaïs Baudon
ENS de Rennes & LIP (UMR CNRS/ENS Lyon/UCB Lyon1/INRIA), Lyon, France

Carsten Fuhs
Birkbeck, University of London, United Kingdom

Laure Gonnord
University of Lyon & LIP (UMR CNRS/ENS Lyon/UCB Lyon1/INRIA), Lyon, France

Abstract

In this workshop paper, we revisit the notion of parallel-innermost term rewriting. We provide a definition of parallel complexity and propose techniques to derive upper bounds on this complexity via the Dependency Tuple framework by Noschinski et al.

2012 ACM Subject Classification Theory of computation → Program verification, Rewrite systems; Software and its engineering → Automated static analysis, Formal software verification

Keywords and phrases Complexity analysis, Parallelism, Rewriting

1 Introduction

In this paper, we consider the problem of evaluating the potentiality of parallelisation in pattern-matching based recursive functions like the one depicted in Figure 1.

```
fn size(&self) -> int {
  match self {
    &Tree::Node { v, ref left, ref right }
    =>  left.size() + right.size() + 1,
    &Tree::Empty => 0 , }    }
```

Figure 1 Tree size computation in Rust

In this particular example, the recursive calls to `left.size()` and `right.size()` can be done in parallel. Building on previous work on parallel-innermost rewriting [6, 4], and first ideas about parallel complexity [1], we propose a new notion of Parallel Dependency Tuples that capture such a behaviour, and a method to compute *parallel complexity bounds*.

2 Parallel-innermost Term Rewriting

The following definitions are mostly standard [3].

▶ **Definition 1** (Term rewrite system, innermost rewriting). *$\mathcal{T}(\Sigma, \mathcal{V})$ denotes the set of* terms *over a finite signature Σ and the set of variables $\mathcal{V}$. For a term t, the set $\mathcal{P}os(t)$ of its* positions *is defined inductively as a set of strings of positive integers: (a) if $t \in \mathcal{V}$, then $\mathcal{P}os(t) = \{\varepsilon\}$, and (b) if $t = f(t_1, \ldots, t_n)$, then $\mathcal{P}os(t) = \{\varepsilon\} \cup \bigcup_{1 \leq i \leq n} \{i\pi \mid \pi \in \mathcal{P}os(t_i)\}$. The position ε is called the* root position *of term t. The* (strict) prefix order $<$ *on positions is the strict partial order given by: $\pi < \tau$ iff there exists $\pi' \neq \varepsilon$ such that $\pi\pi' = \tau$. For $\pi \in \mathcal{P}os(t)$, $t|_\pi$ is the subterm of t at position π, and we write $t[s]_\pi$ for the term that results from t by replacing the subterm $t|_\pi$ at position π by the term s.*

* This work was partially funded by the French National Agency of Research in the CODAS Project (ANR-17-CE23-0004-01).

For a term t, $\mathcal{V}(t)$ is the set of variables in t. If t has the form $f(t_1, \ldots, t_n)$, $\mathrm{root}(t) = f$ *is the* root *of t. A* term rewrite system (TRS) $\mathcal{R}$ *is a set of rules $\{\ell_1 \to r_1, \ldots, \ell_n \to r_n\}$ with $\ell_i, r_i \in \mathcal{T}(\Sigma, \mathcal{V})$, $\ell_i \notin \mathcal{V}$, and $\mathcal{V}(r_i) \subseteq \mathcal{V}(\ell_i)$ for all $1 \le i \le n$. The* rewrite relation *of $\mathcal{R}$ is $s \to_{\mathcal{R}} t$ iff there are a rule $\ell \to r \in \mathcal{R}$, a position $\pi \in \mathcal{P}os(s)$, and a substitution σ such that $s = s[\ell\sigma]_\pi$ and $t = s[r\sigma]_\pi$. Here, σ is called the* matcher *and the term $\ell\sigma$ is called the* redex *of the rewrite step. If $\ell\sigma$ has no proper subterm that is also a possible redex, $\ell\sigma$ is an* innermost redex, *and the rewrite step is an* innermost rewrite step *denoted by $s \xrightarrow{i}_{\mathcal{R}} t$.*

$\Sigma^{\mathcal{R}}_d = \{f \mid f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}\}$ *and* $\Sigma^{\mathcal{R}}_c = \Sigma \setminus \Sigma^{\mathcal{R}}_d$ *are the* defined *and* constructor *symbols of $\mathcal{R}$. We may omit the superscript and just write Σ_d and Σ_c if $\mathcal{R}$ is not of importance or clear from the context. Finally, let $\mathcal{P}os_d(t) = \{\pi \mid \pi \in \mathcal{P}os(t), \mathrm{root}(t|_\pi) \in \Sigma_d\}$.*

The notion of parallel-innermost rewriting dates back at least to [6]. Informally, in a parallel-innermost rewrite step, all innermost redexes are rewritten simultaneously. This corresponds to executing all function calls in parallel on a machine with unbounded parallelism.

▶ **Definition 2** (Parallel-innermost rewriting [4]). *A term s* rewrites innermost in parallel *to t with a TRS $\mathcal{R}$, written $s \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} t$, iff $s \xrightarrow{i}{}^{+}_{\mathcal{R}} t$, and either (a) $s \xrightarrow{i}_{\mathcal{R}} t$ with s an innermost redex, or (b) $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, and for all $1 \le k \le n$ either $s_k \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} t_k$ or $s_k = t_k$ is a normal form.*

▶ **Example 3** (size). Consider the TRS $\mathcal{R}$ with the following rules modelling the code of Figure 1.

$$\begin{array}{rcl|rcl} \mathsf{plus}(\mathsf{Zero}, y) & \to & y & \mathsf{size}(\mathsf{Nil}) & \to & \mathsf{Zero} \\ \mathsf{plus}(\mathsf{S}(x), y) & \to & \mathsf{S}(\mathsf{plus}(x, y)) & \mathsf{size}(\mathsf{Tree}(v, l, r)) & \to & \mathsf{S}(\mathsf{plus}(\mathsf{size}(l), \mathsf{size}(r))) \end{array}$$

Here $\Sigma^{\mathcal{R}}_d = \{\mathsf{plus}, \mathsf{size}\}$ and $\Sigma^{\mathcal{R}}_c = \{\mathsf{Zero}, \mathsf{S}, \mathsf{Nil}, \mathsf{Tree}\}$. We have the following parallel innermost rewrite sequence, where innermost redexes are underlined:

$$\begin{array}{ll} & \underline{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))} \\ \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\underline{\mathsf{size}(\mathsf{Nil})}, \underline{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))})) \\ \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{plus}(\underline{\mathsf{size}(\mathsf{Nil})}, \underline{\mathsf{size}(\mathsf{Nil})})))) \\ \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\underline{\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero})}))) \\ \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} & \mathsf{S}(\underline{\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{Zero}))}) \\ \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{S}(\mathsf{Zero})) \end{array}$$

Note that in the second and in the third step, two innermost steps each are happening in parallel. A corresponding regular innermost rewrite sequence without parallel evaluation of redexes would have needed two more steps.

3 Finding Upper Bounds for Parallel Complexity

3.1 Notion of Parallel Complexity

We extend the notion of innermost runtime complexity to parallel-innermost rewriting.

▶ **Definition 4** ((Parallel) Innermost Runtime Complexity). *The* size *$|t|$ of a term t is $|x| = 1$ if $x \in \mathcal{V}$ and $|f(t_1, \ldots, t_n)| = 1 + \sum_{i=1}^{n} |t_i|$, otherwise. The* derivation height *of a term t w.r.t. a relation $\to$ is the length of the longest sequence of $\to$-steps from t: $\mathrm{dh}(t, \to) = \sup\{e \mid \exists t' \in \mathcal{T}(\Sigma, \mathcal{V}).\ t \to^e t'\}$ where $\to^e$ is the e^{th} iterate of $\to$. If t starts an infinite $\to$-sequence, we write $\mathrm{dh}(t, \to) = \omega$.*

A term $f(t_1, \ldots, t_k)$ is basic (for a TRS $\mathcal{R}$) *iff $f \in \Sigma^{\mathcal{R}}_d$ and $t_1, \ldots, t_k \in \mathcal{T}(\Sigma^{\mathcal{R}}_c, \mathcal{V})$. $\mathcal{T}^{\mathcal{R}}_{\mathrm{basic}}$ is the set of basic terms for a TRS $\mathcal{R}$. For $n \in \mathbb{N}$, we define the* innermost runtime complexity *function $\mathrm{irc}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}(t, \xrightarrow{i}_{\mathcal{R}}) \mid t \in \mathcal{T}_{\mathrm{basic}}, |t| \le n\}$ and we introduce the* parallel innermost runtime complexity *function $\mathrm{irc}^{\Vert}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}(t, \mathrel{\not\Vert\!\xrightarrow{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}_{\mathrm{basic}}, |t| \le n\}$.*

In the following, given a TRS $\mathcal{R}$, our goal shall be to infer (asymptotic) upper bounds for $\mathrm{irc}^{\parallel}_{\mathcal{R}}$ fully automatically. As usual for runtime complexity, we are considering only basic terms as start terms, corresponding to a defined function called on data objects as arguments. An upper bound for (sequential) $\mathrm{irc}_{\mathcal{R}}$ is also an upper bound for $\mathrm{irc}^{\parallel}_{\mathcal{R}}$. We will introduce techniques to find upper bounds for $\mathrm{irc}^{\parallel}_{\mathcal{R}}$ that are strictly tighter than these trivial bounds.

3.2 Complexity: the sequential case

We build on the Dependency Tuple framework [5], originally introduced to determine upper bounds for (sequential) innermost runtime complexity. A central idea is to group all function calls by a rewrite rule *together* rather than to regard them separately (as for termination [2]).

▶ **Definition 5** (Sharp Terms $\mathcal{T}^\sharp$). *For every $f \in \Sigma_d$, we introduce a fresh symbol $f^\sharp$ of the same arity. For a term $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma_d$, we define $t^\sharp = f^\sharp(t_1, \ldots, t_n)$ and let $\mathcal{T}^\sharp = \{t^\sharp \mid t \in \mathcal{T}(\Sigma, \mathcal{V}), \mathrm{root}(t) \in \Sigma_d\}$.*

To compute an upper bound for sequential complexity, we "count" how often each rewrite rule is used. The idea is that the cost of the function call to the lhs of a rule is 1 + the sum of the costs of all the function calls in the rhs, counted separately. To group k function calls together, we use "compound symbols" Com_k, which intuitively represent the sum of the runtimes of their arguments. Then, we can use polynomial interpretations $\mathcal{P}ol$ with $\mathcal{P}ol(\mathsf{Com}_k(x_1, \ldots, x_k)) = x_1 + \cdots + x_k$ for all k to compute a complexity bound [5, Thm. 27].

▶ **Definition 6** (Dependency Tuple, DT [5]). *A* dependency tuple (DT) *is a rule of the form $s^\sharp \to \mathsf{Com}_n(t^\sharp_1, \ldots, t^\sharp_n)$ where $s^\sharp, t^\sharp_1, \ldots, t^\sharp_n \in \mathcal{T}^\sharp$. Let $\ell \to r$ be a rule with $\mathcal{P}os_d(r) = \{\pi_1, \ldots, \pi_n\}$ and $\pi_1 \lhd \ldots \lhd \pi_n$ for a total order $\lhd$ on positions. Then $DT(\ell \to r) = \ell^\sharp \to \mathsf{Com}_n(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_n})$. For a TRS $\mathcal{R}$, let $DT(\mathcal{R}) = \{DT(\ell \to r) \mid \ell \to r \in \mathcal{R}\}$.*

▶ **Example 7.** For our running example, we get the following DTs:

$$\begin{array}{rcl}
\mathsf{plus}^\sharp(\mathsf{Zero}, y) & \to & \mathsf{Com}_0 \\
\mathsf{plus}^\sharp(\mathsf{S}(x), y) & \to & \mathsf{Com}_1(\mathsf{plus}^\sharp(x, y)) \\
\mathsf{size}^\sharp(\mathsf{Nil}) & \to & \mathsf{Com}_0 \\
\mathsf{size}^\sharp(\mathsf{Tree}(v, l, r)) & \to & \mathsf{Com}_3(\mathsf{size}^\sharp(l), \mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r)))
\end{array}$$

The following polynomial interpretation, which orients all DTs with $\succ$ and all rules from $\mathcal{R}$ with $\succsim$, proves $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$: $\mathcal{P}ol(\mathsf{plus}^\sharp(x_1, x_2)) = \mathcal{P}ol(\mathsf{size}(x_1)) = x_1, \mathcal{P}ol(\mathsf{size}^\sharp(x_1)) = 2x_1 + x_1^2, \mathcal{P}ol(\mathsf{plus}(x_1, x_2)) = x_1 + x_2, \mathcal{P}ol(\mathsf{Tree}(x_1, x_2, x_3)) = 1 + x_2 + x_3, \mathcal{P}ol(\mathsf{S}(x_1)) = 1 + x_1, \mathcal{P}ol(\mathsf{Zero}) = \mathcal{P}ol(\mathsf{Nil}) = 1$. $\mathcal{P}ol$ is indeed strictly decreasing along rules (proving termination) and for any term t, $\mathcal{P}ol(t) \in \mathcal{O}(|t|)$.

3.3 Computing Upper Bounds for Parallel Rewriting

To find upper bounds for runtime complexity of parallel-innermost rewriting, we can *reuse* the notion of DTs from Def. 6 for sequential innermost rewriting along with existing techniques [5] and implementations. We illustrate this in the following example.

▶ **Example 8.** In the recursive size-rule, the two calls to $\mathsf{size}(l)$ and $\mathsf{size}(r)$ happen *in parallel* (this will be captured by the notion of *structural independency*). Thus, the cost for these two calls is not the *sum*, but the *maximum* of the calls. Regardless of which of these two calls

has the higher cost, we still need to add the cost for the call to plus, which starts evaluating only after both calls to size have finished. With σ as the used matcher for the rule, we have:

$$\begin{aligned} & \mathrm{dh}(\mathsf{size}(\mathsf{Tree}(v,l,r))\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}) \\ = \;& 1 + \max(\mathrm{dh}(\mathsf{size}(l)\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}), \mathrm{dh}(\mathsf{size}(r)\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}})) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l),\mathsf{size}(r))\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}) \end{aligned}$$

Equivalently, we can "factor in" the cost of calling plus into the maximum function:

$$\begin{aligned} & \mathrm{dh}(\mathsf{size}(\mathsf{Tree}(v,l,r))\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}) \\ = \;& \max(1 + \mathrm{dh}(\mathsf{size}(l)\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l),\mathsf{size}(r))\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}), \\ & \quad 1 + \mathrm{dh}(\mathsf{size}(r)\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}}) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l),\mathsf{size}(r))\sigma, \overset{\mathsf{i}}{\Vdash\!\!\rightarrow}_{\mathcal{R}})) \end{aligned}$$

Intuitively, this would correspond to evaluating $\mathsf{plus}(\mathsf{size}(l),\mathsf{size}(r))$ twice, in two parallel threads of execution, which costs the same amount of time as evaluating $\mathsf{plus}(\mathsf{size}(l),\mathsf{size}(r))$ once. We can represent this maximum of the execution times of two threads by introducing *two* DTs for our recursive size-rule:

$$\begin{aligned} \mathsf{size}^\sharp(\mathsf{Tree}(v,l,r)) &\to \mathsf{Com}_2(\mathsf{size}^\sharp(l), \mathsf{plus}^\sharp(\mathsf{size}(l),\mathsf{size}(r))) \\ \mathsf{size}^\sharp(\mathsf{Tree}(v,l,r)) &\to \mathsf{Com}_2(\mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l),\mathsf{size}(r))) \end{aligned}$$

To express the cost of a concrete rewrite sequence, we would non-deterministically choose the DT that corresponds to the "slower thread".

Alternatively, one could introduce a new symbol ComPar_k that would be interpreted as the maximum of its arguments:

$$\mathsf{size}^\sharp(\mathsf{Tree}(v,l,r)) \to \mathsf{Com}_2(\mathsf{ComPar}_2(\mathsf{size}^\sharp(l), \mathsf{size}^\sharp(r)), \mathsf{plus}^\sharp(\mathsf{size}(l),\mathsf{size}(r)))$$

However, we would then have to extend the notion of Dependency Tuples and also adapt all existing techniques in the Dependency Tuple Framework to work with ComPar_k.
(Here, unfortunately $\mathsf{plus}^\sharp$ must wait for $\mathsf{size}^\sharp$ to finish, so it cannot be evaluated in parallel, and we still need Com_2 to count it in addition to the $\mathsf{size}^\sharp$ calls.)

In other words, the cost of the function call to the lhs of a rule is 1 + the sum of the costs of all the function calls in the rhs *that are in structural dependency with each other*. The actual cost of the function call to the lhs in a concrete rewrite sequence is the *maximum* of all the possible costs of such *chains* (in the prefix order $<$ on positions) of structural dependency. Thus, *structurally independent* function calls are considered in separate DTs, whose non-determinism models the parallelism of these function calls.

The notion of *structural dependency* of functions calls is captured by Def. 9. Basically, it comes from the fact that a term cannot be evaluated before all its subterms have been reduced to normal forms (innermost rewriting/*call by value*).

▶ **Definition 9** (Structural dependency). *Let t be a term and τ_1, τ_2 be the positions of two redexes in t. Let $t_1 = t|_{\tau_1}$ and $t_2 = t|_{\tau_2}$. Then t_1* structurally depends *on t_2 iff. $\tau_1 < \tau_2$ in the prefix order $<$ (i.e., t_2 is a subterm of t_1).*

In particular, for any rule $\ell \to r \in \mathcal{R}$, any substitution σ and any two positions $\tau_1, \tau_2 \in \mathcal{P}os_d(r)$, $r|_{\tau_1}$ structurally depends on $r|_{\tau_2}$ iff. $\tau_1 < \tau_2$.

We thus revisit the notion of DTs, which now embed structural dependencies.

▶ **Definition 10** (Parallel Dependency Tuples $DT^{\|}$, PDTs). *For a rewrite rule $\ell \to r$, we define the set of its* Parallel Dependency Tuples (PDTs) *$DT^{\|}(\ell \to r)$: if $\mathcal{P}os_d(r) = \emptyset$, then $DT^{\|}(\ell \to r) = \{\ell^\sharp \to \mathsf{Com}_0\}$; otherwise, $DT^{\|}(\ell \to r) = \{\ell^\sharp \to \mathsf{Com}_k(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_k}) \mid k > 0, \pi_1 > \cdots > \pi_k$ is a maximal structural dependency chain in $\mathcal{P}os_d(r)\}$, where $\pi_1 > \cdots > \pi_k$ is a maximal structural dependency chain in $\mathcal{P}os_d(r)$ iff. $\forall \pi \in \mathcal{P}os_d(r), \pi \not> \pi_1 \wedge \pi_k \not> \pi$. For a TRS $\mathcal{R}$, let $DT^{\|}(\mathcal{R}) = \bigcup_{\ell \to r \in \mathcal{R}} DT^{\|}(\ell \to r)$.*

▶ **Example 11.** For our recursive size-rule $lhs \to rhs$, we have $\mathcal{P}os_d(rhs) = \{1, 11, 12\}$. The two maximal $>$-chains are $11 > 1$ and $12 > 1$. With $rhs|_1 = \mathsf{plus}(\mathsf{size}(l), \mathsf{size}(r))$, $rhs|_{11} = \mathsf{size}(l)$, and $rhs|_{12} = \mathsf{size}(r)$, we get the PDTs from Ex. 8.

To connect PDTs with our parallel-innermost rewrite relation $\parallel\!\!\to^{\mathsf{i}}_{\mathcal{R}}$, we need the notion of *chain tree*, which is an extension of dependency chains [2], and its complexity.

▶ **Definition 12** (Chain Tree, *Cplx* [5])**.** *Let $\mathcal{D}$ be a set of DTs and $\mathcal{R}$ be a TRS. Let T be a (possibly infinite) tree whose nodes are labelled with a DT from $\mathcal{D}$ and a substitution. Let the root node be labelled with $(s^\sharp \to \mathsf{Com}_n(\ldots) \mid \sigma)$. Then T is a $(\mathcal{D}, \mathcal{R})$-*chain tree for $s^\sharp\sigma$ *iff. the following conditions hold for any node of T, where $(u^\sharp \to \mathsf{Com}_m(v_1^\sharp, \ldots, v_m^\sharp) \mid \mu)$ is the label of the node:*

- *$u^\sharp\mu$ is in normal form w.r.t. $\mathcal{R}$;*
- *if this node has the children $(p_1^\sharp \to \mathsf{Com}_{m_1}(\ldots) \mid \delta_1), \ldots, (p_k^\sharp \to \mathsf{Com}_{m_k}(\ldots) \mid \delta_k)$, then there are pairwise different $i_1, \ldots, i_k \in \{1, \ldots, m\}$ with $v_{i_j}^\sharp\mu \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} p_j^\sharp\delta_j$ for all $j \in \{1, \ldots, k\}$.*

Let $\mathcal{S} \subseteq \mathcal{D}$ and $s^\sharp \in \mathcal{T}^\sharp$. For a chain tree T, $|T|_{\mathcal{S}} \in \mathbb{N} \cup \{\omega\}$ is the number of nodes in T labelled with a DT from $\mathcal{S}$. We define $Cplx_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle}(s^\sharp) = \sup\{|T|_{\mathcal{S}} \mid T$ is a $(\mathcal{D}, \mathcal{R})$-chain tree for $s^\sharp\}$. For terms $s^\sharp$ without a $(\mathcal{D}, \mathcal{R})$-chain tree, we define $Cplx_{\langle \mathcal{D}, \mathcal{S}, \mathcal{R}\rangle}(s^\sharp) = 0$.

We can now make our main correctness claim:

▶ **Proposition 13** (*Cplx* bounds Derivation Height for $\parallel\!\!\to^{\mathsf{i}}_{\mathcal{R}}$)**.** *Let $\mathcal{R}$ be a TRS, let $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ such that all t_i are in normal form. Then we have $\mathrm{dh}(t, \parallel\!\!\to^{\mathsf{i}}_{\mathcal{R}}) \leq Cplx_{\langle DT^{\parallel}(\mathcal{R}), DT^{\parallel}(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp)$. If $\parallel\!\!\to^{\mathsf{i}}_{\mathcal{R}}$ is confluent, then $\mathrm{dh}(t, \parallel\!\!\to^{\mathsf{i}}_{\mathcal{R}}) = Cplx_{\langle DT^{\parallel}(\mathcal{R}), DT^{\parallel}(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp)$.*

Thus, in particular we can use polynomial interpretations in the DT framework for our PDTs to get upper bounds for $\mathrm{irc}^{\parallel}_{\mathcal{R}}$.

▶ **Example 14** (Ex. 8 continued)**.** For our TRS $\mathcal{R}$ computing the size function on trees, we get the set $DT^{\parallel}(\mathcal{R})$ with the following PDTs:

$$\begin{array}{l|l}
\mathsf{plus}^\sharp(\mathsf{Zero}, y) \to \mathsf{Com}_0 & \mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0 \\
\mathsf{plus}^\sharp(\mathsf{S}(x), y) \to \mathsf{Com}_1(\mathsf{plus}^\sharp(x, y)) & \mathsf{size}^\sharp(\mathsf{Tree}(v, l, r)) \to \mathsf{Com}_2(\mathsf{size}^\sharp(l), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r))) \\
 & \mathsf{size}^\sharp(\mathsf{Tree}(v, l, r)) \to \mathsf{Com}_2(\mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r)))
\end{array}$$

The interpretation $\mathcal{P}ol$ from Ex. 7 implies $\mathrm{irc}^{\parallel}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$. This bound is tight: consider $\mathsf{size}(t)$ for a comb-shaped tree t where the first argument of Tree is always Zero and the third is always Nil. The function plus, which needs time linear in its first argument, is called linearly often on data linear in the size of the start term. Due to the structural dependencies, these calls do not happen in parallel (so call $k+1$ to plus must wait for call k).

▶ **Example 15.** Note that $\mathrm{irc}^{\parallel}(n)$ can be asymptotically lower than $\mathrm{irc}(n)$, for instance in:

$$\begin{array}{lcl|lcl}
\mathsf{doubles}(\mathsf{Zero}) & \to & \mathsf{Nil} & \mathsf{d}(\mathsf{Zero}) & \to & \mathsf{Zero} \\
\mathsf{doubles}(\mathsf{S}(x)) & \to & \mathsf{Cons}(\mathsf{d}(\mathsf{S}(x)), \mathsf{doubles}(x)) & \mathsf{d}(\mathsf{S}(x)) & \to & \mathsf{S}(\mathsf{S}(\mathsf{d}(x)))
\end{array}$$

The upper bound $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$ is tight: from a term $\mathsf{doubles}(\mathsf{S}(\mathsf{S}(\ldots \mathsf{S}(\mathsf{Zero}) \ldots)))$, we get linearly many calls of the linear-time function d on arguments of size linear in the start term. However, the Parallel Dependency Tuples in this example are:

$$\begin{array}{lcl|lcl}
\mathsf{doubles}^\sharp(\mathsf{Zero}) & \to & \mathsf{Com}_0 & \mathsf{d}^\sharp(\mathsf{Zero}) & \to & \mathsf{Com}_0 \\
\mathsf{doubles}^\sharp(\mathsf{S}(x)) & \to & \mathsf{Com}_1(\mathsf{d}^\sharp(\mathsf{S}(x))) & \mathsf{d}^\sharp(\mathsf{S}(x)) & \to & \mathsf{Com}_1(\mathsf{d}^\sharp(x)) \\
\mathsf{doubles}^\sharp(\mathsf{S}(x)) & \to & \mathsf{Com}_1(\mathsf{doubles}^\sharp(x)) & & &
\end{array}$$

Then the following polynomial interpretation, which orients all DTs with $\succ$ and all rules from $\mathcal{R}$ with $\succsim$, proves $\mathrm{irc}^{\parallel}_{\mathcal{R}} \in \mathcal{O}(n)$: $\mathcal{P}ol(\mathsf{doubles}^\sharp(x_1)) = \mathcal{P}ol(\mathsf{d}(x_1)) = 2x_1, \mathcal{P}ol(\mathsf{d}^\sharp(x_1)) = x_1$,
$\mathcal{P}ol(\mathsf{doubles}(x_1)) = \mathcal{P}ol(\mathsf{Cons}(x_1, x_2)) = \mathcal{P}ol(\mathsf{Zero}) = \mathcal{P}ol(\mathsf{Nil}) = 1, \mathcal{P}ol(\mathsf{S}(x_1)) = 1 + x_1$.

4 Conclusion

We have come up with a notion of parallel runtime complexity and a concrete algorithm to compute upper bounds on this complexity on TRSs. Future work includes practical design of *parallel rewriting engines* that infer rewriting schedules from parallel dependency tuples, as well as the formalisation of complexity w.r.t. term *height* (considering terms as trees), which seems to be more practical for our parallelisation needs.

References

1 Christophe Alias, Carsten Fuhs, and Laure Gonnord. Estimation of Parallel Complexity with Rewriting Techniques. In *Proc. WST '16*, pages 2:1–2:5, 2016.

2 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

3 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge Univ. Press, 1998.

4 Mirtha-Lina Fernández, Guillem Godoy, and Albert Rubio. Orderings for innermost termination. In *Proc. RTA '05*, pages 17–31, 2005.

5 Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *J. Autom. Reason.*, 51(1):27–56, 2013.

6 Jean Vuillemin. Correct and optimal implementations of recursion in a simple programming language. *J. Comput. Syst. Sci.*, 9(3):332–354, 1974.