Analysing Parallel Complexity of Term Rewriting*

Thaïs Baudon[1], Carsten Fuhs[2], and Laure Gonnord[3,1]

[1] LIP (UMR CNRS/ENS Lyon/UCB Lyon1/INRIA), Lyon, France
[2] Birkbeck, University of London, United Kingdom
[3] LCIS (UGA/Grenoble INP/Ésisar), Valence, France

Abstract. We revisit parallel-innermost term rewriting as a model of parallel computation on inductive data structures and provide a corresponding notion of runtime complexity parametric in the size of the start term. We propose automatic techniques to derive both upper and lower bounds on parallel complexity of rewriting that enable a direct reuse of existing techniques for sequential complexity. The applicability and the precision of the method are demonstrated by the relatively light effort in extending the program analysis tool AProVE and by experiments on numerous benchmarks from the literature.

1 Introduction

Automated inference of complexity bounds for parallel computation has seen a surge of attention in recent years [11,12,31,5,30,17]. While techniques and tools for a variety of computational models have been introduced, so far there does not seem to be any paper in this area for complexity of *term rewriting* with parallel evaluation strategies. This paper addresses this gap in the literature. We consider term rewrite systems (TRSs) as *intermediate representation* for programs with *pattern-matching* operating on *algebraic data types* 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 , }    }
```

Fig. 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 [40,19], and first ideas about parallel complexity [6], we propose a new notion of Parallel Dependency Tuples that captures such a behaviour, and methods to compute both upper and lower *parallel complexity bounds*.

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

Bounds on parallel complexity can provide insights about the potentiality of parallelisation: if sequential and parallel complexity of a function (asymptotically) coincide, this information can be useful for a parallelising compiler to refrain from parallelising the evaluation of this function. Moreover, evaluation of TRSs (as a simple functional programming language) in massively parallel settings such as GPUs is currently a topic of active research [18]. In this context, a static analysis of parallel complexity can be helpful to determine whether to rewrite on a (fast, but not very parallel) CPU or on a (slower, but massively parallel) GPU.

A preliminary version of this work with an initial notion of parallel complexity was presented in an informal extended abstract [13]. We now propose a more formal version accompanied by extensions, proofs, implementation, experiments, and related work. Sect. 2 recalls term rewriting and Dependency Tuples [37] as the basis of our approach. In Sect. 3, we introduce a notion of runtime complexity for parallel-innermost rewriting, and we harness the existing Dependency Tuple framework to compute asymptotic upper bounds on this complexity. In Sect. 4, we provide a transformation to innermost term rewriting that lets any tool for (sequential) innermost runtime complexity be reused to find upper bounds for parallel-innermost runtime complexity and, for confluent parallel-innermost rewriting, also lower bounds. Sect. 5 gives experimental evidence of the practicality of our method on a large standard benchmark set. We discuss related work in Sect. 6. Our extended authors' accepted manuscript [14] additionally has full proofs of our theorems.

2 Term Rewriting and Innermost Runtime Complexity

We assume basic familiarity with term rewriting (see, e.g., [10]) and recall standard definitions to fix notation. As customary for analysis of runtime complexity of rewriting, we consider terms as *tree-shaped* objects, without sharing of subtrees.

We first define *Term Rewrite Systems* and *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, its *size* $|t|$ is defined by: (a) if $t \in \mathcal{V}$, $|t| = 1$; (b) if $t = f(t_1, \ldots, t_n)$, then $|t| = 1 + \sum_{i=1}^{n} |t_i|$. The set $\mathcal{P}os(t)$ of the *positions* of t is defined by: (a) if $t \in \mathcal{V}$, then $\mathcal{P}os(t) = \{\varepsilon\}$, and (b) if $t = f(t_1, \ldots, t_n)$, then $\mathcal{P}os(t) = \{\varepsilon\} \cup \bigcup_{1 \le i \le n} \{i\pi \mid \pi \in \mathcal{P}os(t_i)\}$. The position ε is the *root position* of term t. If $t = f(t_1, \ldots, t_n)$, $\mathrm{root}(t) = f$ is the *root symbol* of t. The *(strict) prefix order* $>$ on positions is the strict partial order given by: $\tau > \pi$ iff there exists $\pi' \neq \varepsilon$ such that $\pi\pi' = \tau$. Two positions π and τ are *parallel* iff neither $\pi > \tau$ nor $\pi = \tau$ nor $\tau > \pi$ hold. 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.

A substitution σ is a mapping from $\mathcal{V}$ to $\mathcal{T}(\Sigma, \mathcal{V})$ with finite domain $Dom(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. We write $\{x_1 \mapsto t_1; \ldots; x_n \mapsto t_n\}$ for a substitution σ with $\sigma(x_i) = t_i$ for $1 \le i \le n$ and $\sigma(x) = x$ for all other $x \in \mathcal{V}$. We extend substitutions to terms by $\sigma(f(t_1, \ldots, f_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$. We may write $t\sigma$ for $\sigma(t)$.

For a term t, $\mathcal{V}(t)$ is the set of variables in 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 $1 \leq i \leq n$. 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 $\ell\sigma$ the *redex* of the rewrite step. If no proper subterm of $\ell\sigma$ is a possible redex, $\ell\sigma$ is an *innermost redex*, and the rewrite step is an *innermost rewrite step*, denoted by $s \xrightarrow{\text{i}}_\mathcal{R} t$.

$\Sigma_d^\mathcal{R} = \{f \mid f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}\}$ and $\Sigma_c^\mathcal{R} = \Sigma \setminus \Sigma_d^\mathcal{R}$ are the *defined* and *constructor* symbols of $\mathcal{R}$. We may also just write Σ_d and Σ_c. The set of positions with defined symbols of t is $\mathcal{P}os_d(t) = \{\pi \mid \pi \in \mathcal{P}os(t), \mathrm{root}(t|_\pi) \in \Sigma_d\}$.

For a relation $\to$, $\to^+$ is its transitive closure and $\to^*$ its reflexive-transitive closure. An object o is a *normal form* wrt a relation $\to$ iff there is no o' with $o \to o'$. A relation $\to$ is *confluent* iff $s \to^* t$ and $s \to^* u$ implies that there exists an object v with $t \to^* v$ and $u \to^* v$. A relation $\to$ is *terminating* iff there is no infinite sequence $t_0 \to t_1 \to t_2 \to \cdots$.

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

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

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

$$\begin{array}{ll} & \underline{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))} \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\mathsf{plus}(\underline{\mathsf{size}(\mathsf{Nil})}, \mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))) \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \underline{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))})) \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{plus}(\underline{\mathsf{size}(\mathsf{Nil})}, \mathsf{size}(\mathsf{Nil})))))) \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \underline{\mathsf{size}(\mathsf{Nil})})))) \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\underline{\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero})}))) \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\underline{\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{Zero}))}) \\ \xrightarrow{\text{i}}_\mathcal{R} & \mathsf{S}(\mathsf{S}(\mathsf{Zero})) \end{array}$$

This rewrite sequence uses 7 steps to reach a normal form.

We wish to provide static bounds on the length of the longest rewrite sequence from terms of a specific size. Here we use innermost evaluation strategies, which closely correspond to call-by-value strategies used in many programming languages. We focus on rewrite sequences that start with *basic terms*, corresponding to function calls where a function is applied to data objects. The resulting notion of complexity for term rewriting is known as *innermost runtime complexity*.

Definition 1 (Innermost Runtime Complexity irc **[26,37]).** *The* derivation height *of a term t wrt 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* $\mathrm{dh}(t, \to) = \omega$*. Here, ω is the smallest infinite ordinal, i.e., $\omega > n$ holds for all $n \in \mathbb{N}$.*

A term $f(t_1, \ldots, t_k)$ is basic (for a TRS $\mathcal{R}$) *iff $f \in \Sigma_d^\mathcal{R}$ and $t_1, \ldots, t_k \in \mathcal{T}(\Sigma_c^\mathcal{R}, \mathcal{V})$. $\mathcal{T}_{\text{basic}}^\mathcal{R}$ is the set of basic terms for a TRS $\mathcal{R}$. For $n \in \mathbb{N}$, the* innermost

runtime complexity *function is* $\mathrm{irc}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}^{\mathcal{R}}_{\mathrm{basic}}, |t| \leq n\}$. *For all* $P \subseteq \mathbb{N} \cup \{\omega\}$, $\sup P$ *is the least upper bound of* P, *where* $\sup \emptyset = 0$.

Many automated techniques are available [26,37,27,8,36,35] to analyse $\mathrm{irc}_{\mathcal{R}}$. We build on Dependency Tuples [37], originally designed to find upper bounds for (sequential) innermost runtime complexity. A central idea is to group all function calls by a rewrite rule *together* rather than to separate them (as with DPs for proving termination [7]). We use *sharp terms* to represent these function calls.

Definition 2 (Sharp Terms $\mathcal{T}^\sharp$). *For every* $f \in \Sigma_d$, *we introduce a fresh symbol* $f^\sharp$ *of the same arity, called a* sharp symbol. *For a term* $t = f(t_1, \ldots, t_n)$ *with* $f \in \Sigma_d$, *we define* $t^\sharp = f^\sharp(t_1, \ldots, t_n)$. *For all other terms* t, *we define* $t^\sharp = t$. $\mathcal{T}^\sharp = \{t^\sharp \mid t \in \mathcal{T}(\Sigma, \mathcal{V}), \mathrm{root}(t) \in \Sigma_d\}$ *denotes the set of* sharp terms.

To get an upper bound for sequential complexity, we "count" how often each rewrite rule is used. The idea is that when a rule $\ell \to r$ is used, the cost (i.e., number of rewrite steps for the evaluation) of the function call to the instance of ℓ is 1 + the sum of the costs of all the function calls in the resulting instance of r, counted separately in some fixed order. To group k function calls together, we use "compound symbols" Com_k of arity k, which intuitively represent the sum of the runtimes of their arguments.

Definition 3 (Dependency Tuple, DT [37]). *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* $\pi_1 \trianglerighteq \ldots \trianglerighteq \pi_n$ *for a total order* $\trianglerighteq$ *(e.g., lexicographic order) on positions. Then* $DT(\ell \to r) = \ell^\sharp \to \mathsf{Com}_n(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_n})$.[4] *For a TRS* $\mathcal{R}$, *let* $DT(\mathcal{R}) = \{DT(\ell \to r) \mid \ell \to r \in \mathcal{R}\}$.

Example 2. For $\mathcal{R}$ from Ex. 1, $DT(\mathcal{R})$ consists of the following DTs:

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

To represent the complexity of a sharp term for a set of DTs and a TRS $\mathcal{R}$, *chain trees* are used [37]. Intuitively, a chain tree for some sharp term is a dependency tree of the computations involved in evaluating this term. Each node represents a computation (the DT) on some arguments (defined by the substitution).

Definition 4 (Chain Tree, *Cplx* [37]). *Let* $\mathcal{D}$ *be a set of DTs and* $\mathcal{R}$ *be a TRS. Let* T *be a (possibly infinite) tree where each node is labelled with a DT* $q^\sharp \to \mathsf{Com}_n(w^\sharp_1, \ldots, w^\sharp_n)$ *from* $\mathcal{D}$ *and a substitution* ν, *written* $(q^\sharp \to \mathsf{Com}_n(w^\sharp_1, \ldots, w^\sharp_n) \mid \nu)$. *Let the root node be labelled with* $(s^\sharp \to \mathsf{Com}_e(r^\sharp_1, \ldots, r^\sharp_e) \mid \sigma)$. *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* $(u^\sharp \to \mathsf{Com}_m(v^\sharp_1, \ldots, v^\sharp_m) \mid \mu)$ *is the label of the node:*

[4] The order $\trianglerighteq$ must be total to ensure that the function DT is well defined wrt the order of the arguments of Com_n. The (partial!) prefix order $>$ is not sufficient here.

– $u^\sharp\mu$ *is in normal form wrt* $\mathcal{R}$*;*
– *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* $v_{i_j}^\sharp\mu \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* p_j^\sharp\delta_j$ *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* $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp) = 0$*.*

Example 3. For $\mathcal{R}$ from Ex. 1 and $\mathcal{D} = DT(\mathcal{R})$ from Ex. 2, the following is a chain tree for the term $\mathsf{size}^\sharp(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))$:

$$(\mathsf{size}^\sharp(\mathsf{Tree}(v,l,r)) \to \mathsf{Com}_3(\mathsf{size}^\sharp(l), \mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r))) \mid \{v \mapsto \mathsf{Zero}; l \mapsto \mathsf{Nil}; r \mapsto \mathsf{Nil}\})$$

$$(\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0 \mid \{\})$$

$$(\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0 \mid \{\})$$

$$(\mathsf{plus}^\sharp(\mathsf{Zero}, y) \to \mathsf{Com}_0 \mid \{y \mapsto \mathsf{Zero}\})$$

$\mathsf{size}(\mathsf{Nil}) \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* \mathsf{Zero}$

The main correctness statement in the sequential case is the following:

Theorem 1 (*Cplx* bounds Derivation Height for $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ [37]). *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 (this includes all* $t \in \mathcal{T}^\mathcal{R}_{\mathrm{basic}}$*). Then we have* $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$*. If* $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ *is confluent, then* $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = Cplx_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(t^\sharp)$*.*

For automated complexity analysis with DTs, the following notion of *DT problems* is used as a characterisation of DTs that we reduce in incremental proof steps to a trivially solved problem.

Definition 5 (DT Problem, Complexity of DT Problem [37]). *Let* $\mathcal{R}$ *be a TRS,* $\mathcal{D}$ *be a set of DTs,* $\mathcal{S} \subseteq \mathcal{D}$*. Then* $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ *is a DT problem. Its complexity function is* $\mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) = \sup\{Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \mid t \in \mathcal{T}^\mathcal{R}_{\mathrm{basic}}, |t| \leq n\}$*. The DT problem* $\langle DT(\mathcal{R}), DT(\mathcal{R}), \mathcal{R}\rangle$ *is called the* canonical DT problem *for* $\mathcal{R}$*.*

For a DT problem $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$, the set $\mathcal{D}$ contains all DTs that can be used in chain trees. $\mathcal{S}$ contains the DTs whose complexity remains to be analysed. $\mathcal{R}$ contains the rewrite rules for evaluating the arguments of DTs. Here we focus on simplifying $\mathcal{S}$ (thus $\mathcal{D}$ and $\mathcal{R}$ are fixed during the process) but techniques to simplify $\mathcal{D}$ and $\mathcal{R}$ are available as well [37,8].

Thm. 1 implies the following link between $\mathrm{irc}_\mathcal{R}$ and $\mathrm{irc}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}$:

Theorem 2 (Complexity Bounds for TRSs via Canonical DT Problems [37]). *Let* $\mathcal{R}$ *be a TRS with canonical DT problem* $\langle DT(\mathcal{R}), DT(\mathcal{R}), \mathcal{R}\rangle$*. Then we have* $\mathrm{irc}_\mathcal{R}(n) \leq \mathrm{irc}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(n)$*. If* $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ *is confluent, we have* $\mathrm{irc}_\mathcal{R}(n) = \mathrm{irc}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R}\rangle}(n)$*.*

In practice, the focus is on finding asymptotic bounds for $\mathrm{irc}_{\mathcal{R}}$. For example, Ex. 4 will show that for our TRS $\mathcal{R}$ from Ex. 1 we have $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$.

A DT problem $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ is said to be *solved* iff $\mathcal{S} = \emptyset$: we always have $\mathrm{irc}_{\langle \mathcal{D},\emptyset,\mathcal{R} \rangle}(n) = 0$. To simplify and finally solve DT problems in an incremental fashion, complexity analysis techniques called *DT processors* are used. A DT processor takes a DT problem as input and returns a (hopefully simpler) DT problem as well as an asymptotic complexity bound as an output. The largest asymptotic complexity bound returned over this incremental process is then also an upper bound for $\mathrm{irc}_{\mathcal{R}}(n)$ [37, Corollary 21].

The reduction pair processor using polynomial interpretations [37] applies a restriction of polynomial interpretations to $\mathbb{N}$ [34] to infer upper bounds on the number of times that DTs can occur in a chain tree for terms of size at most n.

Definition 6 (Polynomial Interpretation, CPI). *A* polynomial interpretation $\mathcal{P}ol$ *maps every* n*-ary function symbol to a polynomial with variables* $x_1, \ldots, x_n$ *and coefficients from* $\mathbb{N}$*.* $\mathcal{P}ol$ *extends to terms via* $\mathcal{P}ol(x) = x$ *for* $x \in \mathcal{V}$ *and* $\mathcal{P}ol(f(t_1, \ldots, t_n)) = \mathcal{P}ol(f)(\mathcal{P}ol(t_1), \ldots, \mathcal{P}ol(t_n))$*.* $\mathcal{P}ol$ *induces an order* $\succ_{\mathcal{P}ol}$ *and a quasi-order* $\succsim_{\mathcal{P}ol}$ *over terms where* $s \succ_{\mathcal{P}ol} t$ *iff* $\mathcal{P}ol(s) > \mathcal{P}ol(t)$ *and* $s \succsim_{\mathcal{P}ol} t$ *iff* $\mathcal{P}ol(s) \geq \mathcal{P}ol(t)$ *for all instantiations of variables with natural numbers.*

A complexity polynomial interpretation (CPI) $\mathcal{P}ol$ *is a polynomial interpretation where:* $\mathcal{P}ol(\mathsf{Com}_n(x_1, \ldots, x_n)) = x_1 + \cdots + x_n$*, and for all* $f \in \Sigma_c$*,* $\mathcal{P}ol(f(x_1, \ldots, x_n)) = a_1 \cdot x_1 + \cdots + a_n \cdot x_n + b$ *for some* $a_i \in \{0, 1\}$ *and* $b \in \mathbb{N}$*.*

The restriction for CPIs regarding constructor symbols enforces that the interpretation of a constructor term t (as an argument of a term for which a chain tree is constructed) can exceed its size $|t|$ only by at most a constant factor. This is crucial for soundness. Using a CPI, we can now define and state correctness of the corresponding reduction pair processor [37, Theorem 27].

Theorem 3 (Reduction Pair Processor with CPIs [37]). *Let* $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ *be a DT problem, let* $\succsim$ *and* $\succ$ *be induced by a CPI* $\mathcal{P}ol$*. Let* $k \in \mathbb{N}$ *be the maximal degree of all polynomials* $\mathcal{P}ol(f^\sharp)$ *for all* $f \in \Sigma_d$*. Let* $\mathcal{D} \cup \mathcal{R} \subseteq \succsim$*. If* $\mathcal{S} \cap \succ \neq \emptyset$*, the reduction pair processor returns the DT problem* $\langle \mathcal{D}, \mathcal{S} \setminus \succ, \mathcal{R} \rangle$ *and the complexity* $\mathcal{O}(n^k)$*. Then the reduction pair processor is sound.*

Example 4 (Ex. 2 continued). For our running example, consider the CPI $\mathcal{P}ol$ with: $\mathcal{P}ol(\mathsf{plus}^\sharp(x_1, x_2)) = \mathcal{P}ol(\mathsf{size}(x_1)) = x_1$, $\mathcal{P}ol(\mathsf{size}^\sharp(x_1)) = 2x_1 + x_1^2$, $\mathcal{P}ol(\mathsf{plus}(x_1, x_2)) = x_1 + x_2, \mathcal{P}ol(\mathsf{Tree}(x_1, x_2, x_3)) = 1 + x_2 + x_3, \mathcal{P}ol(\mathsf{S}(x_1)) = 1 + x_1, \mathcal{P}ol(\mathsf{Zero}) = \mathcal{P}ol(\mathsf{Nil}) = 1$. $\mathcal{P}ol$ orients all DTs in $\mathcal{S} = DT(\mathcal{R})$ with $\succ$ and all rules in $\mathcal{R}$ with $\succsim$. This proves $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$: since the maximal degree of the CPI for a symbol $f^\sharp$ is 2, the upper bound of $\mathcal{O}(n^2)$ follows by Thm. 3.

3 Finding Upper Bounds for Parallel Complexity

In this section we present our main contribution: an application of the DT framework from innermost runtime complexity to *parallel-innermost rewriting*.

The notion of parallel-innermost rewriting dates back at least to [40]. Informally, in a parallel-innermost rewrite step, all innermost redexes are rewritten simultaneously. This corresponds to executing all function calls in parallel using a call-by-value strategy on a machine with unbounded parallelism [15]. In the literature [39], this strategy is also known as "max-parallel-innermost rewriting".

Definition 7 (Parallel-Innermost Rewriting [19]). *A term s* rewrites innermost in parallel *to t with a TRS $\mathcal{R}$, written $s \mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}} t$, iff $s \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{+} t$, and either (a) $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ with s an innermost redex, or (b) $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, and for all $1 \leq k \leq n$ either $s_k \mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}} t_k$ or $s_k = t_k$ is a normal form.*

Example 5 (Ex. 1 continued). The TRS $\mathcal{R}$ from Ex. 1 allows the following parallel-innermost rewrite sequence, where innermost redexes are underlined:

$$\begin{aligned}
&\underline{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))} \\
\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}\ &\mathsf{S}(\mathsf{plus}(\underline{\mathsf{size}(\mathsf{Nil})}, \underline{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))})) \\
\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}\ &\mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{plus}(\underline{\mathsf{size}(\mathsf{Nil})}, \underline{\mathsf{size}(\mathsf{Nil})})))) \\
\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}\ &\mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\underline{\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero})}))) \\
\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}\ &\mathsf{S}(\underline{\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{Zero}))}) \\
\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}\ &\mathsf{S}(\mathsf{S}(\mathsf{Zero}))
\end{aligned}$$

In the second and in the third step, two innermost steps each happen in parallel (which is not possible with standard innermost rewriting: $\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}} \not\subseteq \xrightarrow{\mathsf{i}}_{\mathcal{R}}$). An innermost rewrite sequence without parallel evaluation necessarily needs two more steps to a normal form from this start term, as in Ex. 1.

Note that for all TRSs $\mathcal{R}$, $\mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}$ is terminating iff $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is terminating [19]. Ex. 5 shows that such an equivalence does *not* hold for the derivation height of a term. The question now is: given a TRS $\mathcal{R}$, how much of a speed-up might we get by a switch from innermost to parallel-innermost rewriting? To investigate, we extend the notion of innermost runtime complexity to parallel-innermost rewriting.

Definition 8 (Parallel-Innermost Runtime Complexity pirc**).** *For $n \in \mathbb{N}$, we define the* parallel-innermost runtime complexity *function as* $\mathrm{pirc}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}(t, \mathrel{\overset{\mathsf{i}}{\parallel}\!\!\to}_{\mathcal{R}}) \mid t \in \mathcal{T}^{\mathcal{R}}_{\mathrm{basic}}, |t| \leq n\}$.

In the literature on parallel computing [15,30,11], the terms *depth* or *span* are commonly used for the concept of the runtime of a function on a machine with unbounded parallelism ("wall time"), corresponding to the complexity measure of $\mathrm{pirc}_{\mathcal{R}}$. In contrast, $\mathrm{irc}_{\mathcal{R}}$ would describe the *work* of a function ("CPU time").

In the following, given a TRS $\mathcal{R}$, our goal shall be to infer (asymptotic) upper bounds for $\mathrm{pirc}_{\mathcal{R}}$ fully automatically. Of course, an upper bound for (sequential) $\mathrm{irc}_{\mathcal{R}}$ is also an upper bound for $\mathrm{pirc}_{\mathcal{R}}$. We will now introduce techniques to find upper bounds for $\mathrm{pirc}_{\mathcal{R}}$ that are strictly tighter than these trivial bounds.

To find upper bounds for runtime complexity of parallel-innermost rewriting, we can *reuse* the notion of DTs from Def. 3 for sequential innermost rewriting along with existing techniques [37] as illustrated in the following example.

Example 6. In the recursive size-rule, the two calls to $\mathsf{size}(l)$ and $\mathsf{size}(r)$ happen *in parallel* (they are *structurally independent*) and take place at *parallel positions* in the term. Thus, the cost (number of rewrite steps with $\parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}$ until a normal form is reached) for these two calls is not the *sum*, but the *maximum* of their individual costs. Regardless of which of these two calls has the higher cost, we still need to add the cost for the call to plus on the results of the two calls: plus starts evaluating only after both calls to size have finished. With σ as the used matcher for the rule and with $t\!\downarrow$ as the (here unique) normal form resulting from repeatedly rewriting a term t with $\parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}$ (the "result" of evaluating t), we have:

$$\begin{aligned}&\mathrm{dh}(\mathsf{size}(\mathsf{Tree}(v,l,r))\sigma, \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}})\\ =\ &1 + \max(\mathrm{dh}(\mathsf{size}(l)\sigma, \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}), \mathrm{dh}(\mathsf{size}(r)\sigma, \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}))\\ &\qquad + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l)\sigma\!\downarrow, \mathsf{size}(r)\sigma\!\downarrow), \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}})\end{aligned}$$

In the DT setting, we could introduce a new symbol ComPar_k that explicitly expresses that its arguments are evaluated in parallel. This symbol would then be interpreted as the maximum of its arguments in an extension of Thm. 3:

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

Although automation of the search for polynomial interpretations extended by the maximum function is readily available [23], we would still have to extend the notion of Dependency Tuples and also adapt all existing techniques in the Dependency Tuple framework to work with ComPar_k.

This is why we have chosen the following alternative approach, which is equally powerful on theoretical level and enables immediate reuse of existing techniques in the DT framework. Equivalently to the above, we can "factor in" the cost of calling plus into the maximum function:

$$\begin{aligned}&\mathrm{dh}(\mathsf{size}(\mathsf{Tree}(v,l,r))\sigma, \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}})\\ =\ &\max(1 + \mathrm{dh}(\mathsf{size}(l)\sigma, \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l)\sigma\!\downarrow, \mathsf{size}(r)\sigma\!\downarrow), \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}),\\ &\qquad 1 + \mathrm{dh}(\mathsf{size}(r)\sigma, \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l)\sigma\!\downarrow, \mathsf{size}(r)\sigma\!\downarrow), \parallel^{\mathsf{i}}\!\!\rightarrow_{\mathcal{R}}))\end{aligned}$$

Intuitively, this would correspond to evaluating $\mathsf{plus}(\ldots,\ldots)$ twice, in two parallel threads of execution, which costs the same amount of (wall) time as evaluating $\mathsf{plus}(\ldots,\ldots)$ 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".

In other words, when a rule $\ell \to r$ is used, the cost of the function call to the instance of ℓ is 1 + the sum of the costs of the function calls in the resulting instance of r *that are in structural dependency with each other*. The actual cost of the function call to the instance of ℓ in a concrete rewrite sequence is the

maximum of all the possible costs caused by such *chains* of structural dependency (based on the prefix order $>$ on positions of defined function symbols in r). 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 function 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*). This induces a "happens-before" relation for the computation [33].

Definition 9 (Structural Dependency, *MSDC*). *For positions $\pi_1, \ldots, \pi_k$, we call $\langle \pi_1, \ldots, \pi_k \rangle$ a* structural dependency chain *for a term t iff $\pi_1, \ldots, \pi_k \in \mathcal{P}os_d(t)$ and $\pi_1 > \ldots > \pi_k$. Here π_i* structurally depends on *π_j in t iff $j < i$. A structural dependency chain $\langle \pi_1, \ldots, \pi_k \rangle$ for a term t is* maximal *iff $k = 0$ and $\mathcal{P}os_d(t) = \emptyset$, or $k > 0$ and $\forall \pi \in \mathcal{P}os_d(t) \,.\, \pi \not> \pi_1 \wedge (\pi_1 > \pi \Rightarrow \pi \in \{\pi_2, \ldots, \pi_k\})$. We write $MSDC(t)$ for the set of all maximal structural dependency chains for t.*

Note that $MSDC(t) \neq \emptyset$ always holds: if $\mathcal{P}os_d(t) = \emptyset$, then $MSDC(t) = \{\langle\rangle\}$.

Example 7. Let $t = \mathsf{S}(\mathsf{plus}(\mathsf{size}(\mathsf{Nil}), \mathsf{plus}(\mathsf{size}(x), \mathsf{Zero})))$. In our running example, t has the following structural dependencies: $MSDC(t) = \{\langle 11, 1\rangle, \langle 121, 12, 1\rangle\}$. The chain $\langle 11, 1\rangle$ corresponds to the nesting of $t|_{11} = \mathsf{size}(\mathsf{Nil})$ below $t|_1 = \mathsf{plus}(\mathsf{size}(\mathsf{Nil}), \mathsf{plus}(\mathsf{size}(x), \mathsf{Zero}))$, so the evaluation of $t|_1$ will have to wait at least until $t|_{11}$ has been fully evaluated.

If π structurally depends on τ in a term t, neither $t|_\tau$ nor $t|_\pi$ need to *be* a redex. Rather, $t|_\tau$ could be *instantiated* to a redex and an instance of $t|_\pi$ could become a redex after its subterms, including the instance of $t|_\tau$, have been evaluated.

We thus revisit the notion of DTs, which now embed structural dependencies in addition to the algorithmic dependencies already captured in DTs.

Definition 10 (Parallel Dependency Tuples *PDT*, Canonical Parallel DT Problem). *For a rewrite rule $\ell \to r$, we define the set of its* Parallel Dependency Tuples (PDTs) *$PDT(\ell \to r)$: $PDT(\ell \to r) = \{\ell^\sharp \to \mathsf{Com}_k(r|^\sharp_{\pi_1}, \ldots, r|^\sharp_{\pi_k}) \mid \langle \pi_1, \ldots, \pi_k \rangle \in MSDC(r)\}$. For a TRS $\mathcal{R}$, let $PDT(\mathcal{R}) = \bigcup_{\ell \to r \in \mathcal{R}} PDT(\ell \to r)$. The* canonical parallel DT problem *for $\mathcal{R}$ is $\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle$.*

Example 8. For our recursive size-rule $\ell \to r$, we have $\mathcal{P}os_d(r) = \{1, 11, 12\}$ and $MSDC(r) = \{\langle 11, 1\rangle, \langle 12, 1\rangle\}$. With $r|_1 = \mathsf{plus}(\mathsf{size}(l), \mathsf{size}(r))$, $r|_{11} = \mathsf{size}(l)$, and $r|_{12} = \mathsf{size}(r)$, we get the PDTs from Ex. 6. For the rule $\mathsf{size}(\mathsf{Nil}) \to \mathsf{Zero}$, we have $MSDC(\mathsf{Zero}) = \{\langle\rangle\}$, so we get $PDT(\mathsf{size}(\mathsf{Nil}) \to \mathsf{Zero}) = \{\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0\}$.

We can now make our main correctness statement:

Theorem 4 (*Cplx* bounds Derivation Height for $\Vdash^{\mathsf{i}}_{\to\mathcal{R}}$). *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 (e.g., when $t \in \mathcal{T}^{\mathcal{R}}_{\text{basic}}$). Then we have $\mathrm{dh}(t, \Vdash^{\mathsf{i}}_{\to\mathcal{R}}) \leq Cplx_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp)$. If $\Vdash^{\mathsf{i}}_{\to\mathcal{R}}$ is confluent, then $\mathrm{dh}(t, \Vdash^{\mathsf{i}}_{\to\mathcal{R}}) = Cplx_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp)$.*[5]

[5] The proof uses the confluence of $\mathcal{R}$ as a sufficient criterion for *unique normal forms*.

From Thm. 4, the soundness of our approach to parallel complexity analysis via the DT framework follows analogously to [37]:

Theorem 5 (Parallel Complexity Bounds for TRSs via Canonical Parallel DT Problems). *Let $\mathcal{R}$ be a TRS with canonical parallel DT problem $\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle$. Then we have* $\mathrm{pirc}_{\mathcal{R}}(n) \leq \mathrm{irc}_{\langle PDT(\mathcal{R}),PDT(\mathcal{R}),\mathcal{R}\rangle}(n)$. *If $\overset{\mathsf{i}}{\parallel\!\!\rightarrow}_{\mathcal{R}}$ is confluent, we have* $\mathrm{pirc}_{\mathcal{R}}(n) = \mathrm{irc}_{\langle PDT(\mathcal{R}),PDT(\mathcal{R}),\mathcal{R}\rangle}(n)$.

This theorem implies that we can reuse arbitrary techniques to find upper bounds for *sequential* complexity in the DT framework also to find upper bounds for *parallel* complexity, without requiring any modification to the framework.

Thus, via Thm. 3, in particular we can use polynomial interpretations in the DT framework for our PDTs to get upper bounds for $\mathrm{pirc}_{\mathcal{R}}$.

Example 9 (Ex. 6 continued). For our TRS $\mathcal{R}$ computing the size function on trees, we get the set $PDT(\mathcal{R})$ with the following PDTs:

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

The interpretation $\mathcal{P}ol$ from Ex. 4 implies $\mathrm{pirc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$. 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 10. Note that $\mathrm{pirc}_{\mathcal{R}}(n)$ can be asymptotically lower than $\mathrm{irc}_{\mathcal{R}}(n)$, for instance for the TRS $\mathcal{R}$ with the following rules:

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

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

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

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

Interestingly enough, Parallel Dependency Tuples also allow us to identify TRSs that have *no* potential for parallelisation by parallel-innermost rewriting.

Theorem 6 (Absence of Parallelism by PDTs). *Let $\mathcal{R}$ be a TRS such that for all rules $\ell \to r \in \mathcal{R}$, $|MSDC(r)| = 1$. Then: (a) $PDT(\mathcal{R}) = DT(\mathcal{R})$; (b) for all basic terms t_0 and rewrite sequences $t_0 \parallel^{\mathsf{i}}_{\mathcal{R}} t_1 \parallel^{\mathsf{i}}_{\mathcal{R}} t_2 \parallel^{\mathsf{i}}_{\mathcal{R}} \dots$, also $t_0 \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathsf{i}}_{\mathcal{R}} \dots$ holds (i.e., from basic terms, $\parallel^{\mathsf{i}}_{\mathcal{R}}$ and $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ coincide); (c) $\mathrm{pirc}_{\mathcal{R}}(n) = \mathrm{irc}_{\mathcal{R}}(n)$.*

Thus, for TRSs $\mathcal{R}$ where Thm. 6 applies, no rewrite rule can introduce parallel redexes, and specific analysis techniques for $\mathrm{pirc}_{\mathcal{R}}$ are not needed.

4 From Parallel DTs to Innermost Rewriting

As we have seen in the previous section, we can transform a TRS $\mathcal{R}$ with parallel-innermost rewrite relation to a DT problem whose complexity provides an upper bound of $\mathrm{pirc}_{\mathcal{R}}$ (or, for confluent $\parallel^{\mathsf{i}}_{\mathcal{R}}$, corresponds exactly to $\mathrm{pirc}_{\mathcal{R}}$). However, DTs are only one of many available techniques to find bounds for $\mathrm{irc}_{\mathcal{R}}$. Other techniques include, e.g., Weak Dependency Pairs [26], usable replacement maps [27], the Combination Framework [8], a transformation to complexity problems for integer transition systems [36], amortised complexity analysis [35], or techniques for finding *lower* bounds [22]. Thus, can we benefit also from other techniques for (sequential) innermost complexity to analyse parallel complexity?

In this section, we answer the question in the affirmative, via a generic transformation from Dependency Tuple problems back to rewrite systems whose innermost complexity can then be analysed using arbitrary existing techniques.

We use *relative rewriting*, which allows for labelling some of the rewrite rules such that their use does not contribute to the derivation height of a term. In other words, rewrite steps with these rewrite rules are "for free" from the perspective of complexity. Existing state-of-the-art tools like AProVE [24] and TcT [9] are able to find bounds on (innermost) runtime complexity of such rewrite systems.

Definition 11 (Relative Rewriting). *For two TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$, $\mathcal{R}_1/\mathcal{R}_2$ is a* relative TRS*. Its* rewrite relation *$\to_{\mathcal{R}_1/\mathcal{R}_2}$ is $\to^*_{\mathcal{R}_2} \circ \to_{\mathcal{R}_1} \circ \to^*_{\mathcal{R}_2}$, i.e., rewriting with $\mathcal{R}_2$ is allowed before and after each $\mathcal{R}_1$-step. We define the* innermost *rewrite relation by $s \xrightarrow{\mathsf{i}}_{\mathcal{R}_1/\mathcal{R}_2} t$ iff $s \to^*_{\mathcal{R}_2} s' \to_{\mathcal{R}_1} s'' \to^*_{\mathcal{R}_2} t$ for some terms s', s'' such that the proper subterms of the redexes of each step with $\to_{\mathcal{R}_2}$ or $\to_{\mathcal{R}_1}$ are in normal form wrt $\mathcal{R}_1 \cup \mathcal{R}_2$.*

The set $\mathcal{T}^{\mathcal{R}_1/\mathcal{R}_2}_{\mathrm{basic}}$ of basic terms for a relative TRS $\mathcal{R}_1/\mathcal{R}_2$ is $\mathcal{T}^{\mathcal{R}_1/\mathcal{R}_2}_{\mathrm{basic}} = \mathcal{T}^{\mathcal{R}_1 \cup \mathcal{R}_2}_{\mathrm{basic}}$. The notion of innermost runtime complexity extends to relative TRSs in the natural way: $\mathrm{irc}_{\mathcal{R}_1/\mathcal{R}_2}(n) = \sup\{\mathrm{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}_1/\mathcal{R}_2}) \mid t \in \mathcal{T}^{\mathcal{R}_1/\mathcal{R}_2}_{\mathrm{basic}}, |t| \leq n\}$

The rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}_1/\mathcal{R}_2}$ is essentially the same as $\xrightarrow{\mathsf{i}}_{\mathcal{R}_1 \cup \mathcal{R}_2}$, but only steps using rules from $\mathcal{R}_1$ count towards the complexity; steps using rules from $\mathcal{R}_2$ have no cost. This can be useful, e.g., for representing that built-in functions from programming languages modelled as recursive functions have constant cost.

Example 11. Consider a variant of Ex. 1 where $\mathsf{plus}(\mathsf{S}(x), y) \to \mathsf{S}(\mathsf{plus}(x, y))$ is moved to $\mathcal{R}_2$, but all other rules are elements of $\mathcal{R}_1$. Then $\mathcal{R}_1/\mathcal{R}_2$ would provide a modelling of the size function that is closer to the Rust function from Sect. 1. Let $\mathsf{S}^n(\mathsf{Zero})$ denote the term obtained by n-fold application of S to Zero (e.g., $\mathsf{S}^2(\mathsf{Zero}) = \mathsf{S}(\mathsf{S}(\mathsf{Zero}))$). Although $\mathrm{dh}(\mathsf{plus}(\mathsf{S}^n(\mathsf{Zero}), \mathsf{S}^m(\mathsf{Zero})), \xrightarrow{\mathrm{i}}_{\mathcal{R}_1 \cup \mathcal{R}_2}) = n+1$, we would then get $\mathrm{dh}(\mathsf{plus}(\mathsf{S}^n(\mathsf{Zero}), \mathsf{S}^m(\mathsf{Zero})), \xrightarrow{\mathrm{i}}_{\mathcal{R}_1/\mathcal{R}_2}) = 1$, corresponding to a machine model where the time of evaluating addition for integers is constant.

Note the similarity of a relative TRS and a Dependency Tuple problem: only certain rewrite steps count towards the analysed complexity. We make use of this observation for the following transformation.

Definition 12 (Relative TRS for a Dependency Tuple Problem, δ). *Let $\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ be a Dependency Tuple problem. We define the corresponding relative TRS $\delta(\langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle) = \mathcal{S}/((\mathcal{D} \setminus \mathcal{S}) \cup \mathcal{R})$.*

In other words, we omit the information that steps with our dependency tuples can happen only on top level (possibly below constructors Com_n, but above $\to_{\mathcal{R}}$ steps). (As we shall see in Thm. 8, this information can be recovered.)

The following example is taken from the *Termination Problem Data Base (TPDB)* [42], a collection of examples used at the annual *Termination and Complexity Competition (termCOMP)* [25,41] (see also Sect. 5):

Example 12 (TPDB, `HirokawaMiddeldorp_04/t002`). Consider the following TRS $\mathcal{R}$ from category `Innermost_Runtime_Complexity` of the TPDB:

$$\begin{array}{rl|rl}
\mathsf{leq}(0, y) & \to \mathsf{true} & \mathsf{if}(\mathsf{true}, x, y) & \to x \\
\mathsf{leq}(\mathsf{s}(x), 0) & \to \mathsf{false} & \mathsf{if}(\mathsf{false}, x, y) & \to y \\
\mathsf{leq}(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{leq}(x, y) & -(x, 0) & \to x \\
\mathsf{mod}(0, y) & \to 0 & -(\mathsf{s}(x), \mathsf{s}(y)) & \to -(x, y) \\
\mathsf{mod}(\mathsf{s}(x), 0) & \to 0 & & \\
\multicolumn{4}{c}{\mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{if}(\mathsf{leq}(y, x), \mathsf{mod}(-(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)), \mathsf{s}(x))}
\end{array}$$

This TRS has the following PDTs $PDT(\mathcal{R})$:

$$\begin{array}{rl|rl}
\mathsf{leq}^\sharp(0, y) & \to \mathsf{Com}_0 & \mathsf{if}^\sharp(\mathsf{true}, x, y) & \to \mathsf{Com}_0 \\
\mathsf{leq}^\sharp(\mathsf{s}(x), 0) & \to \mathsf{Com}_0 & \mathsf{if}^\sharp(\mathsf{false}, x, y) & \to \mathsf{Com}_0 \\
\mathsf{leq}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{Com}_1(\mathsf{leq}^\sharp(x, y)) & -^\sharp(x, 0) & \to \mathsf{Com}_0 \\
\mathsf{mod}^\sharp(0, y) & \to \mathsf{Com}_0 & -^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to \mathsf{Com}_1(-^\sharp(x, y)) \\
\mathsf{mod}^\sharp(\mathsf{s}(x), 0) & \to \mathsf{Com}_0 & & \\
\multicolumn{4}{c}{\mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{Com}_2(\mathsf{leq}^\sharp(y, x), \mathsf{if}^\sharp(\mathsf{leq}(y, x), \mathsf{mod}(-(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)), \mathsf{s}(x)))} \\
\multicolumn{4}{c}{\mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{Com}_3(-^\sharp(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{mod}^\sharp(-(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)), \mathsf{if}^\sharp(\mathsf{leq}(y, x), \mathsf{mod}(-(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)), \mathsf{s}(x)))}
\end{array}$$

The canonical parallel DT problem is $\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R} \rangle$. We get the relative TRS $\delta(\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R} \rangle) = PDT(\mathcal{R})/\mathcal{R}$.

Theorem 7 (Upper Complexity Bounds for $\delta(\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle)$ from $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$). *Let $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ be a DT problem. Then (a) for all $t^\sharp \in \mathcal{T}^\sharp$ with $t \in \mathcal{T}^{\mathcal{R}}_{\text{basic}}$, we have $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \leq \text{dh}(t^\sharp, \xrightarrow{\text{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$, and (b) $\text{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) \leq \text{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n)$.*

Example 13 (Ex. 12 continued). For the relative TRS $PDT(\mathcal{R})/\mathcal{R}$ from Ex. 12, the tool AProVE uses a transformation to integer transition systems [36] followed by an application of the complexity analysis tool CoFloCo [21,20] to find a bound $\text{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n) \in \mathcal{O}(n)$ and to deduce the bound $\text{pirc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$ for the original TRS $\mathcal{R}$ from the TPDB. In contrast, using the techniques of Sect. 3 without the transformation to a relative TRS from Def. 12, AProVE finds only a bound $\text{pirc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$.

Intriguingly, we can use our transformation from Def. 12 not only for finding upper bounds, but also for *lower* bounds on $\text{pirc}_{\mathcal{R}}$.

Theorem 8 (Lower Complexity Bounds for $\delta(\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle)$ from $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$). *Let $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ be a DT problem. Then (a) there is a type assignment s.t. for all $\ell \to r \in \mathcal{D}\cup\mathcal{R}$, ℓ and r get the same type, and for all well-typed $t \in \mathcal{T}^{\mathcal{D}\cup\mathcal{R}}_{\text{basic}}$, $Cplx_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \geq \text{dh}(t, \xrightarrow{\text{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$, and (b) $\text{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) \geq \text{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n)$.*

Thm. 7 and Thm. 8 hold regardless of whether the original DT problem was obtained from a TRS with sequential or with parallel evaluation. So while this kind of connection between DT (or DP) problems and relative rewriting may be folklore in the community, its application to convert a TRS whose *parallel* complexity is sought to a TRS with the same *sequential* complexity is new.

Note that Thm. 5 requires confluence of $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$ to derive lower bounds for $\text{pirc}_{\mathcal{R}}$ from lower complexity bounds of the canonical parallel DT problem. So to use Thm. 8 to search for *lower* complexity bounds with existing techniques [22], we need a criterion for confluence of parallel-innermost rewriting.

Example 14 (Confluence of $\xrightarrow{\text{i}}_{\mathcal{R}}$ does not Imply Confluence of $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$). To see that we cannot prove confluence of $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$ just by using a standard off-the-shelf tool for confluence analysis of innermost or full rewriting [16], consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{f}(\mathsf{b},\mathsf{b}), \mathsf{a} \to \mathsf{f}(\mathsf{b},\mathsf{c}), \mathsf{b} \to \mathsf{c}, \mathsf{c} \to \mathsf{b}\}$. For this TRS, both $\xrightarrow{\text{i}}_{\mathcal{R}}$ and $\to_{\mathcal{R}}$ are confluent. However, $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$ is not confluent: we can rewrite both $\mathsf{a} \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{b})$ and $\mathsf{a} \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{c})$, yet there is no term v such that $\mathsf{f}(\mathsf{b},\mathsf{b}) \parallel^{\text{i}}\!\!\to^*_{\mathcal{R}} v$ and $\mathsf{f}(\mathsf{b},\mathsf{c}) \parallel^{\text{i}}\!\!\to^*_{\mathcal{R}} v$. The reason is that the only possible rewrite sequences with $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$ from these terms are $\mathsf{f}(\mathsf{b},\mathsf{b}) \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \mathsf{f}(\mathsf{c},\mathsf{c}) \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{b}) \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \ldots$ and $\mathsf{f}(\mathsf{b},\mathsf{c}) \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \mathsf{f}(\mathsf{c},\mathsf{b}) \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{c}) \parallel^{\text{i}}\!\!\to_{\mathcal{R}} \ldots$, with no terms in common.

Conjecture 1. If $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$ is confluent, then $\xrightarrow{\text{i}}_{\mathcal{R}}$ is confluent.

Confluence means: if a term s can be rewritten to two different terms t_1 and t_2 in 0 or more steps, it is always possible to rewrite t_1 and t_2 in 0 or more steps to a term u. For $\parallel^{\text{i}}\!\!\to_{\mathcal{R}}$, the redexes that get rewritten are fixed: all innermost redexes simultaneously. Thus, s can rewrite to two *different* terms t_1 and t_2 only if at least one of these redexes can be rewritten in two different ways using $\xrightarrow{\text{i}}_{\mathcal{R}}$.

Towards a sufficient criterion for confluence of parallel-innermost rewriting, we introduce the following standard definition:

Definition 13 (Non-Overlapping). *A TRS $\mathcal{R}$ is* non-overlapping *iff for any two rules $\ell \to r, u \to v \in \mathcal{R}$ where variables have been renamed apart between the rules, there is no position π in ℓ such that $\ell|_\pi \notin \mathcal{V}$ and the terms $\ell|_\pi$ and u unify.*

A sufficient criterion that a given redex has a unique result from a rewrite step is given in the following.

Lemma 1 ([10], Lemma 6.3.9). *If a TRS $\mathcal{R}$ is non-overlapping, $s \to_\mathcal{R} t_1$ and $s \to_\mathcal{R} t_2$ with the redex of both rewrite steps at the same position, then $t_1 = t_2$.*

With the above reasoning, this lemma directly gives us a sufficient criterion for confluence of *parallel-innermost* rewriting.

Corollary 1 (Confluence of Parallel-Innermost Rewriting). *If a TRS $\mathcal{R}$ is non-overlapping, then $\parallel^{\mathrm{i}}\!\!\to_\mathcal{R}$ is confluent.*

So, in those cases we can actually use this sequence of transformations from a parallel-innermost TRS via a DT problem to an innermost (relative) TRS to analyse both upper and lower bounds for the original. Conveniently, these cases correspond to deterministic programs, our motivation for this work!

Example 15 (Ex. 13 continued). Cor. 1 and Thm. 8 imply that a lower bound for $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n)$ of the relative TRS $PDT(\mathcal{R})/\mathcal{R}$ from Ex. 12 carries over to $\mathrm{pirc}_\mathcal{R}(n)$ of the original TRS $\mathcal{R}$ from the TPDB. AProVE uses rewrite lemmas [22] to find the lower bound $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n) \in \Omega(n)$. Together with Ex. 13, we have automatically inferred that this complexity bound is *tight*: $\mathrm{pirc}_\mathcal{R}(n) \in \Theta(n)$.

5 Implementation and Experiments

We have implemented the contributions of this paper in the automated termination and complexity analysis tool AProVE [24]. We added or modified 620 lines of Java code, including 1. the framework of parallel-innermost rewriting; 2. the generation of parallel DTs (Thm. 5); 3. a processor to convert them to TRSs with the same complexity (Thm. 7, Thm. 8); 4. the confluence test of Cor. 1. As far as we are aware, this is the first implementation of a fully automated inference of complexity bounds for parallel-innermost rewriting. To demonstrate the effectiveness of our implementation, we have considered the 663 TRSs from category `Runtime_Complexity_Innermost_Rewriting` of the TPDB, version 11.2 [42]. This category of the TPDB is the benchmark collection used at termCOMP to compare tools that infer complexity bounds for runtime complexity of innermost rewriting, $\mathrm{irc}_\mathcal{R}$. To get meaningful results, we first applied Thm. 6 to exclude TRSs $\mathcal{R}$ where $\mathrm{pirc}_\mathcal{R}(n) = \mathrm{irc}_\mathcal{R}(n)$ trivially holds. We obtained 294 TRSs with potential for parallelism as our benchmark set. We conducted our experiments on the StarExec compute cluster [38] in the `all.q` queue. The timeout per example and tool configuration was set to 300 seconds. Our experimental data with analysis times and all examples are available online [1].

Tool	$\mathcal{O}(1)$	$\leq \mathcal{O}(n)$	$\leq \mathcal{O}(n^2)$	$\leq \mathcal{O}(n^3)$	$\leq \mathcal{O}(n^{\geq 4})$
TcT $\mathrm{irc}_{\mathcal{R}}$	4	28	39	44	44
AProVE $\mathrm{irc}_{\mathcal{R}}$	**5**	50	110	123	127
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 3	**5**	65	**125**	**140**	**142**
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 3 & 4	**5**	**69**	**125**	139	141
TcT $\mathrm{pirc}_{\mathcal{R}}$ Section 4	3	39	52	56	57
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 4	**5**	62	96	105	105

Table 1. Upper bounds for runtime complexity of (parallel-)innermost rewriting

Tool	confluent	$\geq \Omega(n)$	$\geq \Omega(n^2)$	$\geq \Omega(n^3)$	$\geq \Omega(n^{\geq 4})$
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 3 & 4	**186**	133	**23**	**5**	**1**
TcT $\mathrm{pirc}_{\mathcal{R}}$ Section 4	**186**	59	0	0	0
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 4	**186**	**155**	22	**5**	**1**

Table 2. Lower bounds for runtime complexity of parallel-innermost rewriting

Tool	$\Theta(1)$	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^3)$	Total
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 3 & 4	**5**	32	**1**	**3**	41
TcT $\mathrm{pirc}_{\mathcal{R}}$ Section 4	3	21	0	0	24
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 4	**5**	**37**	**1**	**3**	**46**

Table 3. Tight bounds for runtime complexity of parallel-innermost rewriting

As remarked earlier, we always have $\mathrm{pirc}_{\mathcal{R}}(n) \leq \mathrm{irc}_{\mathcal{R}}(n)$, so an upper bound for $\mathrm{irc}_{\mathcal{R}}(n)$ is always a legitimate upper bound for $\mathrm{pirc}_{\mathcal{R}}(n)$. Thus, we include upper bounds for $\mathrm{irc}_{\mathcal{R}}$ found by the state-of-the-art tools AProVE and TcT [2,9]. from termCOMP 2021 as a "baseline" in our evaluation. We compare with several configurations of AProVE and TcT that use the techniques of this paper for $\mathrm{pirc}_{\mathcal{R}}$: "AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 3" also uses Thm. 5 to produce canonical parallel DT problems as input for the DT framework. "AProVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 3 & 4" additionally uses the transformation from Def. 12 to convert a TRS $\mathcal{R}$ to a relative TRS $PDT(\mathcal{R})/\mathcal{R}$ and then to analyse $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n)$ (for lower bounds only together with a confluence proof via Cor. 1). We also extracted each of the TRSs $PDT(\mathcal{R})/\mathcal{R}$ and used the files as inputs for AProVE and TcT from termCOMP 2021. "AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 4" and "TcT $\mathrm{pirc}_{\mathcal{R}}$ Section 4" provide the results for $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}$ (for lower bounds, only where $\parallel^{1}\!\!\rightarrow_{\mathcal{R}}$ had been proved confluent).

Table 1 gives an overview over our experimental results for upper bounds. For each configuration, we state the number of examples for which the corresponding asymptotic complexity bound was inferred. A column "$\leq \mathcal{O}(n^k)$" means that the corresponding tools proved a bound $\leq \mathcal{O}(n^k)$ (e.g., the configuration "AProVE $\mathrm{irc}_{\mathcal{R}}$" proved constant or linear upper bounds in 50 cases). Maximum values in a column are highlighted in bold. We observe that upper complexity bounds improve in a noticeable number of cases, e.g., linear bounds on $\mathrm{pirc}_{\mathcal{R}}$ can now be inferred for 69 TRSs rather than for 50 TRSs (using upper bounds on $\mathrm{irc}_{\mathcal{R}}$ as an over-approximation), an improvement by 38%. Note that this does *not* indicate deficiencies in the existing tools for $\mathrm{irc}_{\mathcal{R}}$, which had not been designed with analysis of $\mathrm{pirc}_{\mathcal{R}}$ in mind – rather, it shows that specialised techniques for analysing $\mathrm{pirc}_{\mathcal{R}}$ are a worthwhile subject of investigation. Note also that Ex. 4 and Ex. 9 show that even for TRSs with potential for parallelism, the actual

parallel and sequential complexity may still be asymptotically identical, which further highlights the need for dedicated analysis techniques for $\mathrm{pirc}_{\mathcal{R}}$.

The improvement from $\mathrm{irc}_{\mathcal{R}}$ to $\mathrm{pirc}_{\mathcal{R}}$ can be drastic: for example, for the TRS `TCT_12/recursion_10`, the bounds found by AProVE change from an upper bound of sequential complexity of $\mathcal{O}(n^{10})$ to a (tight) upper bound for parallel complexity of $\mathcal{O}(n)$. (This TRS models a specific recursion structure, with rules $\{\mathsf{f}_0(x) \to \mathsf{a}\} \cup \{\mathsf{f}_i(x) \to \mathsf{g}_i(x,x),\ \mathsf{g}_i(\mathsf{s}(x),y) \to \mathsf{b}(\mathsf{f}_{i-1}(y),\mathsf{g}_i(x,y)) \mid 1 \leq i \leq 10\}$, and is highly amenable to parallelisation.) We observe that adding the techniques from Sect. 4 to the techniques from Sect. 3 leads to only few examples for which better upper bounds can be found (one of them is Ex. 13).

Table 2 shows our results for lower bounds on $\mathrm{pirc}_{\mathcal{R}}$. Here we evaluated only configurations including Def. 12 to make inference techniques for lower bounds of $\mathrm{irc}_{\mathcal{R}}$ applicable to $\mathrm{pirc}_{\mathcal{R}}$. The reason is that a lower bound on $\mathrm{irc}_{\mathcal{R}}$ is not necessarily also a lower bound for $\mathrm{pirc}_{\mathcal{R}}$ (the whole *point* of performing innermost rewriting in parallel is to reduce the asymptotic complexity!), so using results by tools that compute lower bounds on $\mathrm{irc}_{\mathcal{R}}$ for comparison would not make sense. We observe that non-trivial lower bounds can be inferred for 155 out of the 186 examples proved confluent via Cor. 1. This shows that our transformation from Sect. 4 has practical value since it produces relative TRSs that are generally amenable to analysis by existing program analysis tools. Finally, Table 3 shows that for overall 46 TRSs, the bounds that were found are asymptotically *precise*.

6 Related Work, Conclusion, and Future Work

Related work. We provide pointers to work on automated analysis of (sequential) innermost runtime complexity of TRSs at the start of Sect. 4. We now focus on automated techniques for complexity analysis of parallel/concurrent computation.

Our notion of parallel complexity follows a large tradition of static *cost analysis*, notably for concurrent programming. The two notable works [4,5] address async/finish programs where tasks are explicitly launched. The authors propose several metrics such as the total number of spawned tasks (in any execution of the program) and a notion of parallel complexity that is roughly the same as ours. They provide static analyses that build on techniques for estimating costs of imperative languages with functions calls [3], and/or recurrence equations. Recent approaches for the Pi Calculus [11,12] compute the *span* (our parallel complexity) through a new typing system. Another type-based calculus for the same purpose has been proposed with session types [17].

For logic programs, which – like TRSs – express an implicit parallelism, parallel complexity can be inferred using recurrence solving [31].

The tool RAML [29] derives bounds on the worst-case evaluation cost of first-order functional programs with list and pair constructors as well as pattern matching and both sequential and parallel composition [30]. They use two typing derivations with specially annotated types, one for the *work* and one for the *depth* (parallel complexity). Our setting is more flexible wrt the shape of user-defined

data structures (we allow for tree constructors of arbitrary arity), and our analysis deals with both data structure and control in an integrated manner.

Conclusion and future work. We have defined parallel-innermost runtime complexity for TRSs and proposed an approach to its automated analysis. Our approach allows for finding both upper and lower bounds and builds on existing techniques and tools. Our experiments on the TPDB indicate that our approach is practically usable, and we are confident that it captures the potential parallelism of programs with pattern matching.

Parallel rewriting is a topic of active research, e.g., for GPU-based massively parallel rewrite engines [18]. Here our work could be useful to determine which functions to evaluate on the GPU. More generally, parallelising compilers which need to determine which function calls should be compiled into parallel code may benefit from an analysis of parallel-innermost runtime complexity such as ours.

DTs have been used [43] in runtime complexity analysis of *Logically Constrained TRSs (LCTRSs)* [32], an extension of TRSs by built-in data types from SMT theories (integers, arrays, ...). This work could be extended to parallel rewriting. Moreover, analysis of *derivational complexity* [28] of parallel-innermost term rewriting can be a promising direction. Derivational complexity considers the length of rewrite sequences from arbitrary start terms, e.g., $\mathsf{d}(\mathsf{d}(\ldots(\mathsf{d}(\mathsf{S}(\mathsf{Zero})))\ldots))$ in Ex. 10, which can have longer derivations than basic terms of the same size. Finally, towards automated parallelisation we aim to infer complexity bounds wrt term *height* (terms = trees!), as suggested in [6].

Acknowledgements. We thank the anonymous reviewers for helpful comments.

References

1. https://www.dcs.bbk.ac.uk/~carsten/eval/parallel_complexity/
2. https://www.starexec.org/starexec/secure/details/solver.jsp?id=29575
3. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: Cost analysis of object-oriented bytecode programs. Theor. Comput. Sci. **413**(1), 142–159 (2012), https://doi.org/10.1016/j.tcs.2011.07.009
4. Albert, E., Arenas, P., Genaim, S., Zanardini, D.: Task-level analysis for a language with async/finish parallelism. In: Vitek, J., Sutter, B.D. (eds.) Proceedings of the ACM SIGPLAN/SIGBED 2011 conference on Languages, compilers, and tools for embedded systems, LCTES 2011, Chicago, IL, USA, April 11-14, 2011. pp. 21–30. ACM (2011), https://doi.org/10.1145/1967677.1967681
5. Albert, E., Correas, J., Johnsen, E.B., Pun, V.K.I., Román-Díez, G.: Parallel cost analysis. ACM Trans. Comput. Log. **19**(4), 31:1–31:37 (2018), https://doi.org/10.1145/3274278
6. Alias, C., Fuhs, C., Gonnord, L.: Estimation of Parallel Complexity with Rewriting Techniques. In: Proceedings of the 15th Workshop on Termination (WST 2016). pp. 2:1–2:5 (2016), https://hal.archives-ouvertes.fr/hal-01345914
7. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science **236**, 133–178 (2000)
8. Avanzini, M., Moser, G.: A combination framework for complexity. Information and Computation **248**, 22–55 (2016), https://doi.org/10.1016/j.ic.2015.12.007

9. Avanzini, M., Moser, G., Schaper, M.: TcT: Tyrolean Complexity Tool. In: Chechik, M., Raskin, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9636, pp. 407–423. Springer (2016), https://doi.org/10.1007/978-3-662-49674-9_24
10. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge Univ. Press (1998)
11. Baillot, P., Ghyselen, A.: Types for complexity of parallel computation in pi-calculus. In: Yoshida, N. (ed.) Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12648, pp. 59–86. Springer (2021), https://doi.org/10.1007/978-3-030-72019-3_3
12. Baillot, P., Ghyselen, A., Kobayashi, N.: Sized Types with Usages for Parallel Complexity of Pi-Calculus Processes. In: Haddad, S., Varacca, D. (eds.) 32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference. LIPIcs, vol. 203, pp. 34:1–34:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021), https://doi.org/10.4230/LIPIcs.CONCUR.2021.34
13. Baudon, T., Fuhs, C., Gonnord, L.: Parallel complexity of term rewriting systems. In: 17th International Workshop on Termination (WST 2021). pp. 45–50 (2021), https://hal.archives-ouvertes.fr/hal-03418400/document
14. Baudon, T., Fuhs, C., Gonnord, L.: Analysing parallel complexity of term rewriting (2022). https://doi.org/10.48550/ARXIV.2208.01005, https://arxiv.org/abs/2208.01005
15. Blelloch, G.E., Greiner, J.: Parallelism in sequential functional languages. In: Williams, J. (ed.) Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995. pp. 226–237. ACM (1995), https://doi.org/10.1145/224164.224210
16. Community: The international Confluence Competition (CoCo), http://project-coco.uibk.ac.at/
17. Das, A., Hoffmann, J., Pfenning, F.: Parallel complexity analysis with temporal session types. Proc. ACM Program. Lang. **2**(ICFP), 91:1–91:30 (2018), https://doi.org/10.1145/3236786
18. van Eerd, J., Groote, J.F., Hijma, P., Martens, J., Wijs, A.: Term rewriting on GPUs. In: Hojjat, H., Massink, M. (eds.) Fundamentals of Software Engineering - 9th International Conference, FSEN 2021, Virtual Event, May 19-21, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12818, pp. 175–189. Springer (2021), https://doi.org/10.1007/978-3-030-89247-0_12
19. Fernández, M., Godoy, G., Rubio, A.: Orderings for innermost termination. In: Giesl, J. (ed.) Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3467, pp. 17–31. Springer (2005), https://doi.org/10.1007/978-3-540-32033-3_3
20. Flores-Montoya, A.: Upper and lower amortized cost bounds of programs expressed as cost relations. In: Fitzgerald, J.S., Heitmeyer, C.L., Gnesi, S., Philippou, A. (eds.) FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus,

November 9-11, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9995, pp. 254–273 (2016), https://doi.org/10.1007/978-3-319-48989-6_16
21. Flores-Montoya, A., Hähnle, R.: Resource analysis of complex programs with cost equations. In: Garrigue, J. (ed.) Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8858, pp. 275–295. Springer (2014), https://doi.org/10.1007/978-3-319-12736-1_15
22. Frohn, F., Giesl, J., Hensel, J., Aschermann, C., Ströder, T.: Lower bounds for runtime complexity of term rewriting. J. Autom. Reason. **59**(1), 121–163 (2017), https://doi.org/10.1007/s10817-016-9397-x
23. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: Maximal termination. In: Voronkov, A. (ed.) Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5117, pp. 110–125. Springer (2008), https://doi.org/10.1007/978-3-540-70590-1_8
24. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with AProVE. J. Autom. Reason. **58**(1), 3–31 (2017), https://doi.org/10.1007/s10817-016-9388-y
25. Giesl, J., Rubio, A., Sternagel, C., Waldmann, J., Yamada, A.: The termination and complexity competition. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11429, pp. 156–166. Springer (2019), https://doi.org/10.1007/978-3-030-17502-3_10
26. Hirokawa, N., Moser, G.: Automated complexity analysis based on the dependency pair method. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5195, pp. 364–379. Springer (2008), https://doi.org/10.1007/978-3-540-71070-7_32
27. Hirokawa, N., Moser, G.: Automated complexity analysis based on context-sensitive rewriting. In: Dowek, G. (ed.) Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8560, pp. 257–271. Springer (2014), https://doi.org/10.1007/978-3-319-08918-8_18
28. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Dershowitz, N. (ed.) Rewriting Techniques and Applications, 3rd International Conference, RTA-89, Chapel Hill, North Carolina, USA, April 3-5, 1989, Proceedings. Lecture Notes in Computer Science, vol. 355, pp. 167–177. Springer (1989). https://doi.org/10.1007/3-540-51081-8_107, https://doi.org/10.1007/3-540-51081-8_107
29. Hoffmann, J., Aehlig, K., Hofmann, M.: Resource aware ML. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. Lecture Notes in Computer Science, vol. 7358, pp. 781–786. Springer (2012), https://doi.org/10.1007/978-3-642-31424-7_64
30. Hoffmann, J., Shao, Z.: Automatic static cost analysis for parallel programs. In: Vitek, J. (ed.) Programming Languages and Systems - 24th European Symposium

on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9032, pp. 132–157. Springer (2015), https://doi.org/10.1007/978-3-662-46669-8_6

31. Klemen, M., López-García, P., Gallagher, J.P., Morales, J.F., Hermenegildo, M.V.: A general framework for static cost analysis of parallel logic programs. In: Gabbrielli, M. (ed.) Logic-Based Program Synthesis and Transformation - 29th International Symposium, LOPSTR 2019, Porto, Portugal, October 8-10, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12042, pp. 19–35. Springer (2019), https://doi.org/10.1007/978-3-030-45260-5_2
32. Kop, C., Nishida, N.: Term rewriting with logical constraints. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8152, pp. 343–358. Springer (2013), https://doi.org/10.1007/978-3-642-40885-4_24
33. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**(7), 558–565 (1978), https://doi.org/10.1145/359545.359563
34. Lankford, D.S.: Canonical algebraic simplification in computational logic. Tech. Rep. ATP-25, University of Texas (1975)
35. Moser, G., Schneckenreither, M.: Automated amortised resource analysis for term rewrite systems. Sci. Comput. Program. **185** (2020), https://doi.org/10.1016/j.scico.2019.102306
36. Naaf, M., Frohn, F., Brockschmidt, M., Fuhs, C., Giesl, J.: Complexity analysis for term rewriting by integer transition systems. In: Dixon, C., Finger, M. (eds.) Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10483, pp. 132–150. Springer (2017), https://doi.org/10.1007/978-3-319-66167-4_8
37. Noschinski, L., Emmes, F., Giesl, J.: Analyzing innermost runtime complexity of term rewriting by dependency pairs. J. Autom. Reason. **51**(1), 27–56 (2013), https://doi.org/10.1007/s10817-013-9277-6
38. Stump, A., Sutcliffe, G., Tinelli, C.: Starexec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8562, pp. 367–373. Springer (2014), https://doi.org/10.1007/978-3-319-08587-6_28, https://www.starexec.org/
39. Thiemann, R., Sternagel, C., Giesl, J., Schneider-Kamp, P.: Loops under strategies ... continued. In: Kirchner, H., Muñoz, C.A. (eds.) Proceedings International Workshop on Strategies in Rewriting, Proving, and Programming, IWS 2010, Edinburgh, UK, 9th July 2010. EPTCS, vol. 44, pp. 51–65 (2010). https://doi.org/10.4204/EPTCS.44.4, https://doi.org/10.4204/EPTCS.44.4
40. Vuillemin, J.: Correct and optimal implementations of recursion in a simple programming language. J. Comput. Syst. Sci. **9**(3), 332–354 (1974), https://doi.org/10.1016/S0022-0000(74)80048-6
41. Wiki: The International Termination Competition (TermComp), http://termination-portal.org/wiki/Termination_Competition
42. Wiki: Termination Problems DataBase (TPDB), http://termination-portal.org/wiki/TPDB

43. Winkler, S., Moser, G.: Runtime complexity analysis of logically constrained rewriting. In: Fernández, M. (ed.) Logic-Based Program Synthesis and Transformation - 30th International Symposium, LOPSTR 2020, Bologna, Italy, September 7-9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12561, pp. 37–55. Springer (2020), https://doi.org/10.1007/978-3-030-68446-4_2