Fundamenta Informaticae 192(2) : 121–166 (2024)
Available at IOS Press through:
`https://doi.org/10.3233/FI-242191`

On Complexity Bounds and Confluence of Parallel Term Rewriting[*]

Thaïs Baudon
LIP (UMR CNRS/ENS Lyon/UCB Lyon1/Inria)
Lyon, France

Carsten Fuhs[†]
Birkbeck, University of London
London, United Kingdom

Laure Gonnord[‡]
LCIS, University Grenoble Alpes
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. Our approach to find lower bounds requires confluence of the parallel-innermost rewrite relation, thus we also provide effective sufficient criteria for proving confluence. 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.

Keywords: Term rewriting, confluence, complexity analysis, parallelism, static analysis

[*]This work was partially funded by the French National Agency of Research in the CODAS Project (ANR-17-CE23-0004-01). For Open Access purposes, our extended authors' accepted manuscript [1] of this paper is available under Creative Commons CC BY licence.
[†]Address for correspondence: Birkbeck, University of London, London, United Kingdom.
[‡]Also work: LIP (UMR CNRS/ENS Lyon/UCB Lyon1/Inria), Lyon, France.

1. Introduction

Automated inference of complexity bounds for parallel computation has seen a surge of attention in recent years [2, 3, 4, 5, 6, 7]. 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.

```
let rec size : tree -> int = function
 | Node (_, left, right) -> 1 + size left + size right
 | Empty -> 0
```

Figure 1. Tree size computation in OCaml

In this particular example, the recursive calls `size left` and `size right` can be done in parallel. Building on previous work on parallel-innermost rewriting [8, 9] and first ideas about parallel complexity [10], 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*.

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 [11, 12]. 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.

In this paper, we provide techniques for the synthesis of both upper and lower bounds for the parallel-innermost runtime complexity of TRSs. We motivate our focus on innermost rewrite strategies by the fact that innermost rewriting is closely related to call-by-value evaluation strategies as used by many programming languages, such as OCaml, Scala, Rust, C++, ...

Our approach to finding lower bounds requires that the input TRS is *confluent*. This means essentially that computations with the TRS have deterministic results. Thus, we also provide efficiently checkable sufficient criteria for proving confluence of parallel-innermost rewriting, which are of interest both as an ingredient for our complexity analysis and in their own right. These criteria capture the confluence of TRSs corresponding to programs with deterministic small-step semantics, the motivation of this work.

This paper is an extended journal version of a conference paper published at LOPSTR 2022 [13]. We make the following additional contributions over the conference version:

- We provide additional explanations, examples, and discussion throughout the paper.
- We state a stronger new sufficient criterion for confluence of parallel-innermost rewriting (Theorem 6.18).
- We have run more extended experiments.
- We provide proofs for all our theorems.

2. Illustrating example

We illustrate our approach informally with the help of an example.

Consider the following functional program, given as a term rewrite system with the following rewrite rules.

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

Here we use the constructor symbols Zero and S to represent natural numbers (with Zero for 0 and $\mathsf{S}(x)$ for $x + 1$). Lists are represented via the constructors Nil and Cons. Then the function d computes the double of a natural number:

$$\mathsf{d}(\mathsf{S}(\mathsf{S}(\mathsf{Zero}))) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{S}(\mathsf{S}(\mathsf{d}(\mathsf{S}(\mathsf{Zero})))) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{d}(\mathsf{Zero}))))) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{Zero}))))$$

In other words, $2 \cdot 2 = 4$. The function doubles takes a number n and computes a term representing the list $[2n, 2(n-1), \ldots, 4, 2]$. To evaluate the start term $\mathsf{doubles}(\mathsf{S}(\mathsf{Zero}))$, we need four rewrite steps with a sequential evaluation strategy. In the following rewrite sequence, coloured boxes indicate terms that are reduced by a given step, called *redexes*.[1]

$$\begin{array}{rcl} \mathsf{doubles}(\mathsf{S}(\mathsf{Zero})) & \xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{Cons}(\boxed{\mathsf{d}(\mathsf{S}(\mathsf{Zero}))}, \mathsf{doubles}(\mathsf{Zero})) \\ & \xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\boxed{\mathsf{d}(\mathsf{Zero})})), \mathsf{doubles}(\mathsf{Zero})) \\ & \xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\mathsf{Zero})), \boxed{\mathsf{doubles}(\mathsf{Zero})}) \\ & \xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\mathsf{Zero})), \mathsf{Nil}) \end{array}$$

We want to get an upper bound on the number of evaluation steps with our program for the general case. We first consider existing methods for the classic case of a call-by-value strategy in a *sequential* model: evaluate only a single redex at a time, as in the examples above. We consider start terms where a defined function is called on *data terms*, i.e., terms that use only constructor symbols. Here these terms have the form $\mathsf{d}(t)$ and $\mathsf{doubles}(t)$, where t may contain only the constructor symbols Zero, S, Nil, Cons, and variables. Our upper bound will be parametric in the size n of the start term: larger start terms usually have higher runtimes.

Dependency Tuples [14] are a standard technique for finding such upper bounds for sequential evaluation. The idea is to "desugar" the program by grouping all the function calls of a rule together (hence the name "tuple") and then to analyse how big the function call tree can become.

For example, the rule

$$\mathsf{doubles}(\mathsf{S}(x)) \to \mathsf{Cons}(\mathsf{d}(\mathsf{S}(x)), \mathsf{doubles}(x))$$

has the dependency tuple

$$\mathsf{doubles}^\sharp(\mathsf{S}(x)) \to \mathsf{Com}_2(\mathsf{d}^\sharp(\mathsf{S}(x)), \mathsf{doubles}^\sharp(x))$$

[1]Boxes that keep the same colour throughout one or more rewrite steps indicate terms that are reduced multiple (consecutive) times.

that groups the function calls $\mathsf{d}^\sharp(\mathsf{S}(x))$ and $\mathsf{doubles}^\sharp(y)$ into a single tuple, using a fresh constructor symbol Com_2 for a tuple of 2 arguments Here we ignore the constructor context Cons(...) from the original rule – evaluating it does not cost anything. We use the $\sharp$ symbol to indicate which function calls must be "paid for" in the analysis.

This dependency tuple says, informally:

$$\mathit{cost}(\mathsf{doubles}(\mathsf{S}(x))) = 1 + \mathit{cost}(\mathsf{d}(\mathsf{S}(x))) + \mathit{cost}(\mathsf{doubles}(x))$$

So evaluating a call to $\mathsf{doubles}(\mathsf{S}(x)))$ with that rule "costs" us 1 step (for using the rule itself) + the cost of evaluating the call $\mathsf{d}(\mathsf{S}(x))$ + the cost of evaluating the call $\mathsf{doubles}(x)$.

Polynomial interpretations [15] are the working horse for finding upper bounds on the complexity of such complexity problems represented by Dependency Tuples. A polynomial interpretation $\mathcal{P}ol$ maps function symbols to polynomial functions over the natural numbers, and extends naturally to terms. If we can find an interpretation $\mathcal{P}ol$ such that, among other requirements,

$$\mathcal{P}ol(\mathsf{doubles}^\sharp(\mathsf{S}(x))) \geq 1 + \mathcal{P}ol(\mathsf{d}^\sharp(\mathsf{S}(x))) + \mathcal{P}ol(\mathsf{doubles}^\sharp(x))$$

then the highest degree of a polynomial in $\mathcal{P}ol$ for a symbol $f^\sharp$ is also an upper bound for the size of the call tree possible for these Dependency Tuples with the given program: the polynomial function *overapproximates* the cost function.

Such polynomial interpretations can be found automatically using modern constraint solvers [16, 17]. For our example, we would find an interpretation of degree 2. This tells us that the complexity for evaluating a function in our program using a sequential call-by-value strategy is bounded by $\mathcal{O}(n^2)$ for n as the size of the start term. The above is just an informal overview – Section 3 provides a formal introduction to this approach to analysing complexity for *sequential* evaluation.

The bound 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. So this is as good as it gets…for a sequential evaluation strategy.

Now, how about a *parallel* call-by-value evaluation strategy, where we evaluate function calls that happen at independent positions *at the same time* rather than one after the other? It turns out that with a *parallel* strategy, we can evaluate our start term $\mathsf{doubles}(\mathsf{S}(\mathsf{Zero}))$ in just *three* steps (the two coloured redexes are reduced in parallel):

$$\begin{aligned}
\mathsf{doubles}(\mathsf{S}(\mathsf{Zero})) \quad & \mathrel{+\!\!\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \quad \mathsf{Cons}(\mathsf{d}(\mathsf{S}(\mathsf{Zero})), \mathsf{doubles}(\mathsf{Zero})) \\
& \mathrel{+\!\!\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \quad \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\mathsf{d}(\mathsf{Zero}))), \mathsf{Nil}) \\
& \mathrel{+\!\!\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \quad \mathsf{Cons}(\mathsf{S}(\mathsf{S}(\mathsf{Zero})), \mathsf{Nil})
\end{aligned}$$

Can we expect such speed-ups in the number of steps also in the general case? It turns out that, for our example, the answer is *yes*. To prove this automatically, we revisit Dependency Tuples using a little trick. The reason why parallel evaluation is faster here is that the rule

$$\mathsf{doubles}(\mathsf{S}(x)) \to \mathsf{Cons}(\mathsf{d}(\mathsf{S}(x)), \mathsf{doubles}(x))$$

makes two function calls at independent positions, which then get evaluated *in parallel*. Whichever of the two calls to $\mathsf{d}(\mathsf{S}(x))$ and to $\mathsf{doubles}(x)$ finishes last is responsible for the overall cost of using this rule and evaluating its function calls in parallel. We express this by introducing two *separate* Dependency Tuples for our rewrite rule:

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

These two Dependency Tuples are enough to capture the worst case: either the call to $\mathsf{d}^\sharp(\mathsf{S}(x))$ takes longer to evaluate than the call to $\mathsf{doubles}^\sharp(x)$ (then the first one represents the worst case), or it does not (then the second one represents the worst case).

We can now reuse the same analysis machinery as before to search for a polynomial interpretation $\mathcal{P}ol$ that solves the following constraints:

$$\begin{array}{rcl}\mathcal{P}ol(\mathsf{doubles}^\sharp(\mathsf{S}(x))) & \geq & 1 + \mathcal{P}ol(\mathsf{Com}_1(\mathsf{d}^\sharp(\mathsf{S}(x))))\\ \mathcal{P}ol(\mathsf{doubles}^\sharp(\mathsf{S}(x))) & \geq & 1 + \mathcal{P}ol(\mathsf{Com}_1(\mathsf{doubles}^\sharp(x)))\end{array}$$

Here our constraint solvers find a solution already for a parametric interpretation with templates of degree 1. This tells us that the complexity for evaluating a function in our program using a *parallel* call-by-value strategy is bounded by $\mathcal{O}(n)$ for n as the size of the start term, a strictly better bound than is possible for sequential evaluation.

However, this example is very benign: here *all* function calls triggered by a rule are at independent positions. How about a function to compute the size of a tree with a rule like the following?

$$\mathsf{size}(\mathsf{Tree}(v, l, r)) \to \mathsf{S}(\mathsf{plus}(\mathsf{size}(l), \mathsf{size}(r)))$$

Here plus must wait for the calls to size to finish, and we cannot evaluate plus in parallel with the calls to size. As we shall see in Section 4, a refinement of our method can be used to also deal with such more complicated structural dependencies between function calls.

The above example shows that our analysis of parallel complexity can reuse machinery from sequential complexity analysis provided by the Dependency Tuple framework. But Dependency Tuples are just one method out of a plethora for analysis of sequential complexity of term rewrite systems. Can we not go back from Dependency Tuples to term rewriting systems whose *sequential* complexity we can then analyse in order to get upper bounds for the *parallel* complexity of our *original* program? Section 5 shows us how this is possible.

We would also like to find *lower* bounds on the complexity of parallel evaluation. It turns out that if we are dealing with a *confluent* input program (roughly speaking, a program with deterministic results), Section 5 lets us reuse methods to find lower bounds for sequential complexity of term rewriting systems to get answers for lower bounds for the *parallel* complexity of our *original* program. But to be able to apply these methods, we must somehow know that the input program is indeed confluent. This is why Section 6 introduces methods for analysing *confluence* of a term rewrite system in our parallel evaluation strategy. Then Section 7 gives experimental evidence of the practicality of our methods on large standard benchmark sets. We discuss related work and conclude in Section 8.

Limitations. Our approach to complexity analysis of parallel-innermost rewriting is transformational, by generating problem instances that can be handled by existing complexity analysis tools for (sequential) innermost rewriting [18, 19] as backends. Therefore, the precision of our analysis is limited by the precision of these backend tools, and improvements to their precision should carry over directly also to the analysis of parallel-innermost rewriting.

Moreover, while our contributions aim to be applicable to complexity analysis of programming languages with algebraic data types that use innermost/call-by-value evaluation strategies, we present them in the setting of first-order term rewriting without built-in data types, for parallel-innermost runtime complexity. Extensions to first-order term rewriting with logical constraints [20, 21], to higher-order rewriting without [22] and with [23] logical constraints, and to rewrite strategies that rewrite at even more positions simultaneously than parallel-innermost rewriting [12] would be natural next steps. Similarly, we anticipate extensions from term rewriting to programming languages with call-by-value evaluation strategies, such as OCaml or Scala.

3. Term rewriting and innermost runtime complexity

We assume basic familiarity with term rewriting (see, e.g., [24]) and recall standard definitions to fix notation, which we illustrate in Example 3.1. 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 $t \in \mathcal{V}$, then $|t| = 1$; (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 a term 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 *context* $C[]$ is a term that contains exactly one occurrence of a special symbol $\Box$. Similar to $t[s]_\pi$ (but omitting the position π because it is implied by the sole occurrence of $\Box$), we write $C[s]$ for the term obtained from replacing $\Box$ 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 $x \in \mathcal{V}$ with $x \neq x_i$. 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 $1 \le i \le n$. 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 $\ell\sigma$ 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 $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t$.

$\Sigma_d^{\mathcal{R}} = \{f \mid f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R}\}$ 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* (also: in normal form) w.r.t. 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 3.1. (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 $\Sigma_d^{\mathcal{R}} = \{\mathsf{plus}, \mathsf{size}\}$ and $\Sigma_c^{\mathcal{R}} = \{\mathsf{Zero}, \mathsf{S}, \mathsf{Nil}, \mathsf{Tree}\}$.

First, consider the term $t = \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{Zero})))$. Its size is 5. Its positions are $\mathcal{P}os(t) = \{\varepsilon, 1, 1.1, 1.2, 1.2.1\}$. In t, at position $\pi = 1$ we have the subterm $t|_1 = \mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{Zero}))$. This term matches the first rule of our TRS $\mathsf{plus}(\mathsf{Zero}, y) \to y$ with the substitution $\sigma = \{y \mapsto \mathsf{S}(\mathsf{Zero})\}$. We can therefore reduce t to $\mathsf{S}(\mathsf{S}(\mathsf{Zero}))$.

Beginning from $t' = \mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))$, we have the following innermost rewrite sequence, where the used innermost redexes are put in coloured boxes:

$$\begin{array}{rl}
& \boxed{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))} \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\boxed{\mathsf{size}(\mathsf{Nil})}, \mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})))) \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \boxed{\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))})) \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{plus}(\boxed{\mathsf{size}(\mathsf{Nil})}, \mathsf{size}(\mathsf{Nil}))))) \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \boxed{\mathsf{size}(\mathsf{Nil})})))) \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\boxed{\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero})}))) \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\boxed{\mathsf{plus}(\mathsf{Zero}, \mathsf{S}(\mathsf{Zero}))}) \\
\xrightarrow{\mathrm{i}}_{\mathcal{R}} & \mathsf{S}(\mathsf{S}(\mathsf{Zero}))
\end{array}$$

This rewrite sequence uses 7 steps to reach a normal form as the result of the computation.

Our objective is 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 3.2. (Derivation Height dh, Innermost Runtime Complexity irc [25, 14])
For all $P \subseteq \mathbb{N} \cup \{\omega\}$, $\sup\ P$ is the least upper bound of P, where $\sup \emptyset = 0$ and ω is the smallest infinite ordinal, i.e., $\omega > n$ holds for all $n \in \mathbb{N}$. 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 $f(t_1, \dots, t_k)$ is *basic (for a TRS $\mathcal{R}$)* iff $f \in \Sigma_d^{\mathcal{R}}$ and $t_1, \dots, t_k \in \mathcal{T}(\Sigma_c^{\mathcal{R}}, \mathcal{V})$. $\mathcal{T}_{\text{basic}}^{\mathcal{R}}$ is the set of basic terms for a TRS $\mathcal{R}$. For $n \in \mathbb{N}$, the *innermost runtime complexity* function is $\mathrm{irc}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}(t, \xrightarrow{\text{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}_{\text{basic}}^{\mathcal{R}}, |t| \le n\}$.

Many automated techniques have been proposed [25, 14, 26, 27, 28, 29, 30] to analyse $\mathrm{irc}_{\mathcal{R}}$ and compute bounds on it. We build on Dependency Tuples [14], originally designed to find upper bounds for (sequential) innermost runtime complexity. A central idea is to group all function calls[2] by a rewrite rule *together* rather than to separate them, in contrast to Dependency Pairs for proving termination [31]. We use *sharp terms* to represent these function calls.

Definition 3.3. (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 $t = f(t_1, \dots, t_n)$ with $f \in \Sigma_d$, we define $t^\sharp = f^\sharp(t_1, \dots, t_n)$. 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. To group n function calls together, we use "compound symbols" Com_n of arity n, which intuitively represent the sum of the runtimes of their arguments.

Definition 3.4. (Dependency Tuple, DT [14])
A *dependency tuple (DT)* is a rule of the form $s^\sharp \to \mathsf{Com}_n(t_1^\sharp, \dots, t_n^\sharp)$ where $s^\sharp, t_1^\sharp, \dots, t_n^\sharp \in \mathcal{T}^\sharp$. Let $\ell \to r$ be a rule with $\mathcal{P}os_d(r) = \{\pi_1, \dots, \pi_n\}$ and $\pi_1 \gtrdot \dots \gtrdot \pi_n$ where $\gtrdot$ is the standard lexicographic order on positions. Then $DT(\ell \to r) = \ell^\sharp \to \mathsf{Com}_n(r|_{\pi_1}^\sharp, \dots, r|_{\pi_n}^\sharp)$.[3] For a TRS $\mathcal{R}$, let $DT(\mathcal{R}) = \{DT(\ell \to r) \mid \ell \to r \in \mathcal{R}\}$.

Example 3.5. For $\mathcal{R}$ from Example 3.1, $DT(\mathcal{R})$ consists of the following DTs:

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

Intuitively, the DT $\mathsf{size}^\sharp(\mathsf{Tree}(v, l, r)) \to \mathsf{Com}_3(\mathsf{size}^\sharp(l), \mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r)))$ distils the information about the function calls that we need to "count" from the right-hand side of the original

[2]Here we use the term "function call" for a subterm $f(t_1, \dots, t_n)$ with a defined symbol f at its root to capture the corresponding intuition from functional programming. In contrast to most standard functional programming languages, in term rewriting it is possible that such a function call can be evaluated in several ways (non-determinism), or not at all, so that f need not describe a (total or even partial) function from terms to terms in the mathematical sense.

[3]The original definition of Dependency Tuples [14] allows for using an *arbitrary* total order instead of the lexicographic order on positions for $\gtrdot$. The theory presented in this paper would work also with the original definition. The order $\gtrdot$ must be total to ensure that the function DT is well defined w.r.t. the order of the arguments of Com_n, so the (partial!) prefix order $>$ is not sufficient here.

rewrite rule $\mathsf{size}(\mathsf{Tree}(v,l,r)) \to \mathsf{S}(\mathsf{plus}(\mathsf{size}(l),\mathsf{size}(r)))$, and the DT represents this information in a more structured way: 1. the constructor context $\mathsf{S}(\Box)$ on the right-hand side that is not needed for counting function calls is removed; 2. now all the function calls that need to be counted are present as the *sharp terms* $\mathsf{size}^\sharp(l)$, $\mathsf{size}^\sharp(r)$, and $\mathsf{plus}^\sharp(\mathsf{size}(l),\mathsf{size}(r))$; 3. these sharp terms are direct arguments of the new compound symbol Com_3; and 4. the function calls $\mathsf{size}(l)$ and $\mathsf{size}(r)$ below $\mathsf{plus}^\sharp$ on the right-hand side are now ignored for their direct contribution to the cost (their cost is accounted for via $\mathsf{size}^\sharp(l)$ and $\mathsf{size}^\sharp(r)$), but are considered only for their normal forms from innermost evaluation that will be used for evaluating $\mathsf{plus}^\sharp$ in the recursive call.

To represent the number of rewrite steps used in the worst case to reduce a sharp term according to a set of DTs and a TRS $\mathcal{R}$, *chain trees* are used [14]. 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 function calls represented by the DT with its special syntactic structure) on some arguments (defined by the substitution).

We now use a *tree* structure to represent the computation represented by the Dependency Tuples rather than a rewrite *sequence* or a linear *chain* (as with Dependency Pairs). The reason is that we now have *several* function calls on the right-hand sides of our Dependency Tuples, and we want to trace their computations in the tree *independently*. Thus, each function call gives rise to a new subtree.

Each actual innermost rewrite sequence with $\mathcal{R}$ will have a corresponding chain tree constructed using $DT(\mathcal{R})$, with $\mathcal{R}$ used implicitly for the calls to helper functions inside the sharp terms. This will make chain trees useful for finding bounds on the maximum length of rewrite sequences with $\mathcal{R}$.

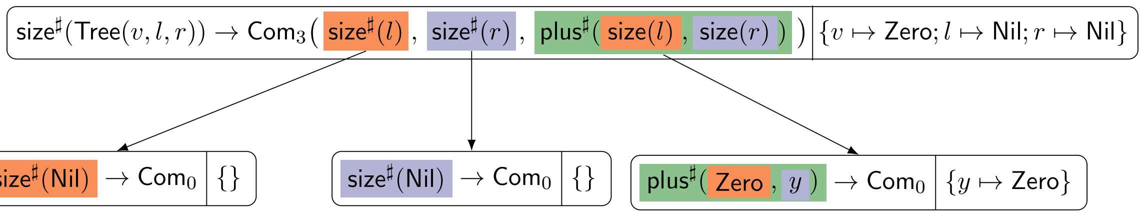
Definition 3.6. (Chain Tree [14])
Let $\mathcal{D}$ be a set of DTs and $\mathcal{R}$ be a TRS. Let T be a (possibly infinite) tree where each node is labelled with a DT $q^\sharp \to \mathsf{Com}_n(w_1^\sharp,\ldots,w_n^\sharp)$ from $\mathcal{D}$ and a substitution ν, written $(q^\sharp \to \mathsf{Com}_n(w_1^\sharp,\ldots,w_n^\sharp) \mid \nu)$. Let the root node be labelled with $(s^\sharp \to \mathsf{Com}_e(r_1^\sharp,\ldots,r_e^\sharp) \mid \sigma)$. 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 $v_{i_j}^\sharp\mu \xrightarrow{\mathrm{i}}_{\mathcal{R}}^* p_j^\sharp\delta_j$ for all $j \in \{1,\ldots,k\}$.

A chain tree represents a rewrite sequence starting from some basic term. Each node captures exactly one rewrite step in this sequence. Its label consists of the DT corresponding to the applied rewrite rule and of the substitution with which the rule matches the term. A node has at most as many children as the arity of the compound symbol on the right-hand side of its DT. Each of its children captures one of the compound symbol's arguments and represents the remaining rewrite steps for this particular subterm. In the definition, we allow innermost rewrite steps to occur from compound symbols' subterms to their corresponding child node. This does not affect the overall cost of the computation represented by the chain tree. Indeed, the cost of every redex that appears as a result of rewriting is captured by the dependency tuple. Therefore, there is no need to "count" the cost of redexes that appear in one of a compound symbol's arguments: they have already been accounted for in another of its arguments.

We illustrate this notion in Example 3.7.

Example 3.7. For $\mathcal{R}$ from Example 3.1 and $\mathcal{D} = DT(\mathcal{R})$ from Example 3.5, the following is a chain tree for the term $s^\sharp = \mathsf{size}^\sharp(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))$:


$\mathsf{size}^\sharp(\mathsf{Tree}(v,l,r)) \to \mathsf{Com}_3(\mathsf{size}^\sharp(l), \mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r)))$ | $\{v \mapsto \mathsf{Zero}; l \mapsto \mathsf{Nil}; r \mapsto \mathsf{Nil}\}$
$\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0$ | $\{\}$
$\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0$ | $\{\}$
$\mathsf{plus}^\sharp(\mathsf{Zero}, y) \to \mathsf{Com}_0$ | $\{y \mapsto \mathsf{Zero}\}$

The root node of the chain tree represents the rewrite step

$$\mathsf{size}(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil})) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{S}(\mathsf{plus}(\mathsf{size}(\mathsf{Nil}), \mathsf{size}(\mathsf{Nil})))$$

via the following DT:

$$\mathsf{size}^\sharp(\mathsf{Tree}(v,l,r)) \to \mathsf{Com}_3(\mathsf{size}^\sharp(l), \mathsf{size}^\sharp(r), \mathsf{plus}^\sharp(\mathsf{size}(l), \mathsf{size}(r)))$$

This node has three children, each corresponding to a redex that will be evaluated when rewriting $\mathsf{S}(\mathsf{plus}(\mathsf{size}(\mathsf{Nil}), \mathsf{size}(\mathsf{Nil})))$ to normal form. Its first two children represent the reduction of the subterms at positions 1.1 and 1.2: $\mathsf{size}(\mathsf{Nil}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{Zero}$, both with the corresponding DT $\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0$.

Its third child node represents the reduction of the rewritten subterm at position 1 of the right-hand side of the rewrite rule, i.e., $\mathsf{plus}(\mathsf{size}(\mathsf{Nil}), \mathsf{size}(\mathsf{Nil}))$. In this node, we capture the rewrite step $\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{Zero}$ with the DT $\mathsf{plus}^\sharp(\mathsf{Zero}, y) \to \mathsf{Com}_0$. In order to reach the term $\mathsf{plus}^\sharp(\mathsf{Zero}, \mathsf{Zero})$ from the sharp term $\mathsf{plus}^\sharp(\mathsf{size}(\mathsf{Nil}), \mathsf{size}(\mathsf{Nil}))$ that appears inside Com_3 (and to reach a normal form w.r.t. $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ in its arguments), we must first perform the rewrite steps $\mathsf{size}(\mathsf{Nil}) \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} \mathsf{Zero}$ on each of its arguments. For the purposes of reaching the third child node, these rewrite steps are "for free", as they have already been accounted for by the two first children.

Analogous to the derivation height $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}})$ for the number of rewrite steps in the longest innermost rewrite sequence from a term t, the notion of *complexity* $\mathit{Cplx}(t^\sharp)$ captures the maximum of the number of nodes of all chain trees for a sharp term $t^\sharp$. As we shall see, this complexity of a sharp term $t^\sharp$ w.r.t. chain trees provides an upper bound on the derivation height of its unsharped version t for the original TRS. One can lift Cplx to innermost runtime complexity for Dependency Tuples as a function of term size, and this notion for Dependency Tuples will provide a bound on the innermost runtime complexity of the original TRS.

Definition 3.8. (*Cplx* **[14]**)
Let $\mathcal{D}$ be a set of DTs and $\mathcal{R}$ be a TRS. 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 $\mathit{Cplx}_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp) = \sup\{|T|_{\mathcal{S}} \mid T \text{ is a } (\mathcal{D},\mathcal{R})\text{-chain tree for } s^\sharp\}$. For terms $s^\sharp$ without a $(\mathcal{D},\mathcal{R})$-chain tree, we define $\mathit{Cplx}_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s^\sharp) = 0$.

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 3.9. (DT Problem, Complexity of DT Problem [14])
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 $\mathrm{irc}_{\langle \mathcal{D},\mathcal{S},\mathcal{R} \rangle}(n) = \sup\{ \mathit{Cplx}_{\langle \mathcal{D},\mathcal{S},\mathcal{R} \rangle}(t^\sharp) \mid t \in \mathcal{T}^{\mathcal{R}}_{\text{basic}}, |t| \leq n\}$. For any TRS $\mathcal{R}$, 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 – and whose complexity we want to analyse. $\mathcal{S}$ 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 [14, 27].

Example 3.10. (Example 3.7 continued)
Our chain tree from Example 3.7 for the term $s^\sharp = \mathsf{size}^\sharp(\mathsf{Tree}(\mathsf{Zero}, \mathsf{Nil}, \mathsf{Nil}))$ has 4 nodes. Thus, we can conclude that $\mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R} \rangle}(s^\sharp) \geq 4$.

The main correctness statement in the sequential case summarises our earlier intuitions. It has a special case for *confluent* TRSs, for which Dependency Tuples capture innermost runtime complexity exactly. The reason that confluence (intuitively: results of computations are deterministic) is required is that without confluence, there can also be chain trees that do not correspond to real computations and lead to higher complexities; for an example, see [14, Example 11].

Theorem 3.11. (Cplx bounds Derivation Height for $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ [14])
Let $\mathcal{R}$ be a TRS, let $t = f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ such that all t_i are in normal form (this includes all $t \in \mathcal{T}^{\mathcal{R}}_{\text{basic}}$). Then we have $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) \leq \mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R} \rangle}(t^\sharp)$. If $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ is confluent,[4] then $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{R}}) = \mathit{Cplx}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R} \rangle}(t^\sharp)$.

Theorem 3.11 implies the following link between $\mathrm{irc}_{\mathcal{R}}$ and $\mathrm{irc}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R} \rangle}$, which also explains why $\langle DT(\mathcal{R}), DT(\mathcal{R}), \mathcal{R} \rangle$ is called the "canonical" DT problem for $\mathcal{R}$:

Theorem 3.12. (Complexity Bounds for TRSs via Canonical DT Problems [14])
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, Example 3.15 will show that for our TRS $\mathcal{R}$ from Example 3.1 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 $\mathrm{irc}_{\langle DT(\mathcal{R}),DT(\mathcal{R}),\mathcal{R} \rangle}(n)$

[4]The proofs for Theorem 3.11 and Theorem 3.12 from the literature and for our new Theorem 4.9 and Theorem 4.13 require only the property that the used rewrite relation has *unique normal forms (w.r.t. reduction)* instead of confluence. However, to streamline presentation, we follow the literature [14] and state our theorems with confluence rather than the property of unique normal forms for the rewrite relation. Note that confluence of a relation is a sufficient condition for unique normal forms, and confluence coincides with the unique normal form property if the relation is terminating [24]. Additionally, there is more readily available tool support for confluence analysis.

and hence also $\mathrm{irc}_{\mathcal{R}}(n)$ [14, Corollary 21]. In all examples that we present in Section 3 and Section 4, a single proof step with a DT processor suffices to solve the given DT problem. Thus, the complexity bound that is found by this proof step is trivially the largest bound among all proof steps and directly carries over to the original problem as an asymptotic bound. In general, *several* proof steps using potentially different DT processors may be needed to find an asymptotic complexity bound for the input TRS, and then an explicit check for the largest bound is needed.

For our examples in Section 3 and Section 4, we use the reduction pair processor using polynomial interpretations [14]. This DT processor applies a restriction of polynomial interpretations to $\mathbb{N}$ [15] 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 3.13. (Polynomial Interpretation, CPI)
A *polynomial interpretation* $\mathcal{P}ol$ maps every n-ary function symbol to a polynomial with variables $x_1, \dots, x_n$ and coefficients from $\mathbb{N}$. $\mathcal{P}ol$ extends to terms via $\mathcal{P}ol(x) = x$ for $x \in \mathcal{V}$ and $\mathcal{P}ol(f(t_1, \dots, t_n)) = \mathcal{P}ol(f)(\mathcal{P}ol(t_1), \dots, \mathcal{P}ol(t_n))$. $\mathcal{P}ol$ 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:

- $\mathcal{P}ol(\mathsf{Com}_n(x_1, \dots, x_n)) = x_1 + \cdots + x_n$, and
- for all $f \in \Sigma_c$, $\mathcal{P}ol(f(x_1, \dots, x_n)) = a_1 \cdot x_1 + \cdots + a_n \cdot x_n + b$ 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 [14, Theorem 27].

Theorem 3.14. (Reduction Pair Processor with CPIs [14])
Let $P = \langle \mathcal{D}, \mathcal{S}, \mathcal{R} \rangle$ 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 $P' = \langle \mathcal{D}, \mathcal{S} \setminus \succ, \mathcal{R} \rangle$ and the complexity $\mathcal{O}(n^k)$. Then the reduction pair processor is sound, i.e., the maximum of the asymptotic upper bound it computes and the complexity of P' is indeed an asymptotic upper bound on the complexity of its input P; see also [14, Definition 17].

Example 3.15. (Example 3.5 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 $\succsim$. Thus, with this CPI, the reduction pair processor returns the solved DT problem $\langle DT(\mathcal{R}), \emptyset, \mathcal{R} \rangle$ and 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 Theorem 3.14. Polynomial interpretations such

as those used in the examples in this paper can be found automatically using parametric interpretation templates for each function symbol and SAT- or SMT-solvers [16, 17] to find the parameter values.

For example, we might use a parametric interpretation $\mathcal{P}ol_p$ with $\mathcal{P}ol_p(\mathsf{plus}^\sharp(x_1, x_2)) = p_0 + p_1x_1 + p_2x_2 + p_3x_1^2 + p_4x_1x_2 + p_5x_2^2$, $\mathcal{P}ol_p(\mathsf{S}(x_1)) = p_6 + p_7x_1$, and $\mathcal{P}ol_p(\mathsf{Com}_1(x_1)) = x_1$. Here all p_i are parameters that range over $\mathbb{N}$, with p_7 additionally being restricted to $\{0, 1\}$. For the term constraint $\mathsf{plus}^\sharp(\mathsf{S}(x), y) \succ \mathsf{Com}_1(\mathsf{plus}^\sharp(x, y))$, we would get the following parametric polynomial constraint that must be satisfied for all $x, y \in \mathbb{N}$:

$$p_0+p_1(p_6+p_7x)+p_2y+p_3(p_6+p_7x)^2+p_4(p_6+p_7x)y+p_5y^2 > p_0+p_1x+p_2y+p_3x^2+p_4xy+p_5y^2$$

We simplify the expression and get:

$$p_1p_6 + p_1p_7x + p_3p_6^2 + 2p_3p_6p_7x + p_3p_7^2x^2 + p_4p_6y + p_4p_7xy > p_1x + p_3x^2 + p_4xy$$

We group parametric coefficients for each monomial in x, y together. This yields:

$$(p_1p_6 + p_3p_6^2) \;+\; (p_1p_7 + 2p_3p_6p_7 - p_1)x \;+\; (p_3p_7^2 - p_3)x^2 \;+\; p_4p_6y \;+\; (p_4p_7 - p_4)xy > 0$$

The *absolute positiveness criterion* [32] allows us to reduce this $\exists\forall$ problem to an $\exists$ problem such that a solution for the latter problem is also a solution for the former problem:

$$p_1p_6 + p_3p_6^2 > 0 \;\wedge\; p_1p_7 + 2p_3p_6p_7 - p_1 \geq 0 \;\wedge\; p_3p_7^2 - p_3 \geq 0 \;\wedge\; p_4p_6 \geq 0 \;\wedge\; p_4p_7 - p_4 \geq 0$$

This problem can now be passed to a constraint solver for non-linear integer arithmetic, e.g., based on SAT- or SMT-solving [16, 17]. In this example, the constraint solver might return a solution with $p_1 = p_6 = p_7 = 1$ and $p_i = 0$ otherwise. This solution for our constraint system allows us to refine $\mathcal{P}ol_p$ to (part of) the above CPI $\mathcal{P}ol$ by replacing the parameters p_i with the values returned from the constraint solver. The full CPI $\mathcal{P}ol$ is obtained by considering all term constraints simultaneously and passing the constraint system to an integer constraint solver; for details, we refer to [16].

4. 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 [8]. 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 [33]. In the literature [34], this strategy is also known as "max-parallel-innermost rewriting".

Definition 4.1. (Parallel-Innermost Rewriting [9])
A term s *rewrites innermost in parallel* to t with a TRS $\mathcal{R}$, written $s \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t$, 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 $s_k \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_k$ or $s_k = t_k$ is a normal form.[5]

[5]The use of "$\vert$" in the notation $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ was suggested by van Oostrom [35] to avoid confusion with the notation $\mathrel{+\!\!\!\!\!\!\rightarrow}_{\mathcal{R}}$, which is commonly used to denote rewriting 0 or more eligible parallel redexes [24]. In contrast, in this paper we require rewriting *all* eligible (here: innermost) redexes in a step with $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$, and at least one such redex must be rewritten in such a step.

As common in parallel rewriting, the redexes that are rewritten are at *parallel positions* – that is, no position is a prefix of another. The literature [34] also contains definitions of parallel rewrite strategies where an arbitrary selection of one or more (not necessarily innermost) parallel redexes may be replaced in the parallel rewrite step rather than all innermost redexes (which are always parallel). In the innermost case, such "may" parallel rewriting would include (sequential) innermost rewriting: rewrite only one innermost redex at a time. Thus, also the worst-case time complexity of "may" parallel rewriting would be identical to that of sequential rewriting. To capture the possible speed improvements enabled by parallel rewriting on a (fictitious) machine with unbounded parallelism, we have chosen a "must" parallel rewriting strategy where *all* eligible redexes (here: innermost) must be rewritten (case (b) of Definition 4.1 rewrites *all* arguments that are not in normal form).

Example 4.2. (Example 3.1 continued)
The TRS $\mathcal{R}$ from Example 3.1 allows the following parallel-innermost rewrite sequence, where innermost redexes are colourised in an analogous way to Example 3.1:

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

In the second and in the third step, two innermost steps happen in parallel (which is not possible with standard innermost rewriting: $\mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \not\subseteq \xrightarrow{\mathsf{i}}_{\mathcal{R}}$). An innermost rewrite sequence without parallel evaluation requires two more steps to reach a normal form from this start term, as shown in Example 3.1. Note that switching from $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ to $\mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ in general does not lead to such a "speed-up": as we shall see in Theorem 4.16, the derivation height of a term does *not* necessarily decrease when $\mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is used instead of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.

Note that for all TRSs $\mathcal{R}$, $\mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is terminating iff $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is terminating [9]. Example 4.2 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 4.3. (Parallel-Innermost Runtime Complexity pirc**)**
For $n \in \mathbb{N}$, we define the *parallel-innermost runtime complexity* function as the maximum of all derivations heights of parallel executions from basic terms of size at most n:

$$\mathrm{pirc}_{\mathcal{R}}(n) = \sup\{\mathrm{dh}(t, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \mid t \in \mathcal{T}^{\mathcal{R}}_{\text{basic}}, |t| \leq n\}.$$

In the literature on parallel computing [33, 6, 2], 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 Definition 3.4 for sequential innermost rewriting along with existing techniques [14] as illustrated in the following example.

Example 4.4. 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 $\mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ 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, or equivalently, size calls *happen before* plus. 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 $\mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ (the "result" of evaluating t), we have:

$$\begin{array}{rl} & \mathrm{dh}(\mathsf{size}(\mathsf{Tree}(v,l,r))\sigma, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \\ = & 1+\max(\mathrm{dh}(\mathsf{size}(l)\sigma, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}), \mathrm{dh}(\mathsf{size}(r)\sigma, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}})) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l)\sigma\downarrow, \mathsf{size}(r)\sigma\downarrow), \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \end{array}$$

In the DT setting, we could introduce a new symbol ComPar_n 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 Theorem 3.14:

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

Although automation of the search for polynomial interpretations extended by the maximum function is readily available [36], 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_n.

This is why we have chosen the following alternative approach, which is equally powerful on theoretical level and enables immediate reuse of all techniques in the existing DT framework [14]. Equivalently to the above, we can "factor in" the cost of calling plus into the maximum function:

$$\begin{array}{rl} & \mathrm{dh}(\mathsf{size}(\mathsf{Tree}(v,l,r))\sigma, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \\ = & \max(1+\mathrm{dh}(\mathsf{size}(l)\sigma, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l)\sigma\downarrow, \mathsf{size}(r)\sigma\downarrow), \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}), \\ & \quad 1+\mathrm{dh}(\mathsf{size}(r)\sigma, \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) + \mathrm{dh}(\mathsf{plus}(\mathsf{size}(l)\sigma\downarrow, \mathsf{size}(r)\sigma\downarrow), \mathrel{\|\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}})) \end{array}$$

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:

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

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 Definition 4.5. 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 [37].

Definition 4.5. (Structural Dependency, Maximal Structural Dependency Chain Set 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 $i > j$. 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 $\forall \pi \in \mathcal{P}os_d(t), \big(\pi \not> \pi_1 \wedge (\pi_1 > \pi \Rightarrow \pi \in \{\pi_2, \ldots, \pi_k\})\big)$. We write $\mathit{MSDC}(t)$ for the set of all maximal structural dependency chains for t.

In the formula specifying *maximal* structural dependency chains for $k > 0$, the first conjunct states that the first element π_1 must be the position of an innermost defined symbol, and the second conjunct states that all positions of defined symbols above π_1 must be part of the chain as well. In other words, it is not possible to add further elements anywhere in a maximal structural dependency chain and obtain a larger structural dependency chain. Note that $\mathit{MSDC}(t) \neq \emptyset$ always holds: if $\mathcal{P}os_d(t) = \emptyset$, then $\mathit{MSDC}(t) = \{\langle\rangle\}$. Note also that since we consider maximality of structural dependency chains w.r.t. subset inclusion rather than cardinality of the sets of their elements, $\mathit{MSDC}(t)$ may contain structural dependency chains with different numbers of elements, as we shall now see in Example 4.6.

Example 4.6. 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: $\mathit{MSDC}(t) = \{\langle 1.1, 1\rangle, \langle 1.2.1, 1.2, 1\rangle\}$. The chain $\langle 1.1, 1\rangle$ corresponds to the nesting of $t|_{1.1} = \mathsf{size}(\mathsf{Nil})$ 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 $t|_{1.1}$ 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 as *Parallel Dependency Tuples*, which now embed structural dependencies in addition to the algorithmic dependencies already captured in DTs.

Definition 4.7. (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)* $PDT(\ell \to r)$: $PDT(\ell \to r) = \{\ell^\sharp \to \mathsf{Com}_k(r|^\sharp_{\pi_1}, \dots, r|^\sharp_{\pi_k}) \mid \langle \pi_1, \dots, \pi_k \rangle \in \mathit{MSDC}(r)\}$. 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 4.8. For our recursive size-rule $\ell \to r$, we have $\mathcal{P}os_d(r) = \{1, 1.1, 1.2\}$ and $\mathit{MSDC}(r) = \{\langle 1.1, 1 \rangle, \langle 1.2, 1 \rangle\}$. With $r|_1 = \mathsf{plus}(\mathsf{size}(l), \mathsf{size}(r))$, $r|_{1.1} = \mathsf{size}(l)$, and $r|_{1.2} = \mathsf{size}(r)$, we get the PDTs from Example 4.4. For the rule $\mathsf{size}(\mathsf{Nil}) \to \mathsf{Zero}$, we have $\mathit{MSDC}(\mathsf{Zero}) = \{\langle\rangle\}$, so we get $PDT(\mathsf{size}(\mathsf{Nil}) \to \mathsf{Zero}) = \{\mathsf{size}^\sharp(\mathsf{Nil}) \to \mathsf{Com}_0\}$.

Our goal is now to prove that with the canonical PDT problem for $\mathcal{R}$ as a starting point, we can *reuse* the existing Dependency Tuple Framework to find bounds on *parallel*-innermost runtime complexity, even though the DT Framework was originally introduced only with sequential innermost rewriting in mind. This allows for reuse both on theory level and on implementation level. A crucial step towards this goal is our main correctness statement:

Theorem 4.9. (Cplx bounds Derivation Height for $\overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}$)
Let $\mathcal{R}$ be a TRS, let $t = f(t_1, \dots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ such that all t_i are in normal form (e.g., when $t \in \mathcal{T}^{\mathcal{R}}_{\text{basic}}$). Then we have $\mathrm{dh}(t, \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}) \leq \mathit{Cplx}_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R} \rangle}(t^\sharp)$.

If $\overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}$ is confluent, then $\mathrm{dh}(t, \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}) = \mathit{Cplx}_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R} \rangle}(t^\sharp)$.

To prove Theorem 4.9, we need some further definitions and lemmas. Intuitively, the notion of *maximal parallel argument normal form* of a term t captures the result of reducing all its arguments to such normal forms that the number of $\overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}$ steps at root position will have "worst-case cost".

Definition 4.10. (Argument Normal Form [14], Maximal Parallel Argument Normal Form)
A term t is an *argument normal form* iff $t \in \mathcal{V}$ or $t = f(t_1, \dots, t_n)$ and all t_i are in normal form. A term $t\Downarrow$ is a *maximal parallel argument normal form* of a term t iff $t\Downarrow$ is an argument normal form such that $t \overset{\mathsf{i}}{\Vert\!\!\to}^*_{\mathcal{R}, >\varepsilon} t\Downarrow$ and for all argument normal forms t' with $t \overset{\mathsf{i}}{\Vert\!\!\to}^*_{\mathcal{R}, >\varepsilon} t'$, we have $\mathrm{dh}(t', \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}) \leq \mathrm{dh}(t\Downarrow, \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}})$. Here $u \overset{\mathsf{i}}{\Vert\!\!\to}^*_{\mathcal{R}, >\varepsilon} v$ denotes a rewrite sequence with $\overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}$ where all steps are at positions $> \varepsilon$.

Example 4.11. Consider the TRS $\{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{c}, \mathsf{f}(\mathsf{b}) \to \mathsf{d}, \mathsf{f}(\mathsf{c}) \to \mathsf{f}(\mathsf{d}), \mathsf{f}(\mathsf{d}) \to \mathsf{d}\}$. The terms $\mathsf{f}(\mathsf{b})$ and $\mathsf{f}(\mathsf{c})$ are both in argument normal form, and we have $\mathsf{f}(\mathsf{a}) \overset{\mathsf{i}}{\Vert\!\!\to}^*_{\mathcal{R}, >\varepsilon} \mathsf{f}(\mathsf{b})$ and $\mathsf{f}(\mathsf{a}) \overset{\mathsf{i}}{\Vert\!\!\to}^*_{\mathcal{R}, >\varepsilon} \mathsf{f}(\mathsf{c})$. The term $\mathsf{f}(\mathsf{c})$ is a maximal parallel argument normal form of $\mathsf{f}(\mathsf{a})$ because for any other term t that satisfies both of these criteria (argument normal form and $\mathsf{f}(\mathsf{a}) \overset{\mathsf{i}}{\Vert\!\!\to}^*_{\mathcal{R}, >\varepsilon} t$), we always have $\mathrm{dh}(t, \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}) \leq \mathrm{dh}(\mathsf{f}(\mathsf{c}), \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}})$.

The following lemma is adapted to the parallel setting from [14].

Lemma 4.12. (Parallel Derivation Heights of Nested Subterms)
Let t be a term, let $\mathcal{R}$ be a TRS such that all reductions of t with $\overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}$ are finite. Then

$$\mathrm{dh}(t, \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}) \leq \max\{ \sum_{1 \leq i \leq k} \mathrm{dh}(t|_{\pi_i}\Downarrow, \overset{\mathsf{i}}{\Vert\!\!\to}_{\mathcal{R}}) \mid \langle \pi_1, \dots, \pi_k \rangle \in \mathit{MSDC}(t)\}$$

If $\overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}$ is confluent, then we additionally have:

$$\mathrm{dh}(t, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) = \max\{\sum_{1 \le i \le k} \mathrm{dh}(t|_{\pi_i}\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \mid \langle \pi_1, \dots, \pi_k \rangle \in \mathit{MSDC}(t)\}$$

Proof:
By induction on the term size $|t|$. If $|t| = 1$, the statement follows immediately since $t\Downarrow = t$. Now consider the case $|t| > 1$. Let n be the arity of the root symbol of t. In (parallel-)innermost rewriting, a rewrite step at the root of t requires that the arguments of t have been rewritten to normal forms. Since rewriting of arguments takes place in parallel (case (b) of Definition 4.1 applies), we have

$$\mathrm{dh}(t, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \le \mathrm{dh}(t\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) + \max\{\quad \mathrm{dh}(t|_j, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \quad \mid 1 \le j \le n\}$$

If $\overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}$ is confluent, then $t\Downarrow$ is uniquely determined and we have equality in the previous as well as in the next (in)equalities.

As $|t_j| < |t|$, we can apply the induction hypothesis:

$$\begin{aligned}\mathrm{dh}(t, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) &\le \mathrm{dh}(t\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \\ &\quad + \max\left\{ \max\left\{ \sum_{1 \le i \le m} \mathrm{dh}(t|_{j.\tau_i}\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \,\middle|\, \langle \tau_1, \dots, \tau_m \rangle \in \mathit{MSDC}(t|_j) \right\} \,\middle|\, 1 \le j \le n \right\}\end{aligned}$$

Equivalently:

$$\begin{aligned}\mathrm{dh}(t, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) &\le \max\left\{ \mathrm{dh}(t\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) + \sum_{1 \le i \le m} \mathrm{dh}(t|_{j.\tau_i}\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \,\middle|\, \begin{array}{c} 1 \le j \le n \\ \langle \tau_1, \dots, \tau_m \rangle \in \mathit{MSDC}(t|_j) \end{array} \right\} \\ &= \max\left\{ \sum_{1 \le i \le k} \mathrm{dh}(t|_{\pi_i}\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) \,\middle|\, \langle \pi_1, \dots, \pi_k \rangle \in \mathit{MSDC}(t) \right\}\end{aligned}$$

For the last equality, consider that the maximal structural dependency chains $\pi_1, \dots, \pi_k$ of t can have two forms. If the root of t is a defined symbol, we have $\langle j.\tau_1, \dots, j.\tau_m, \varepsilon \rangle \in \mathit{MSDC}(t)$. Otherwise $\mathrm{dh}(t\Downarrow, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) = 0$ and thus $\langle j.\tau_1, \dots, j.\tau_m \rangle \in \mathit{MSDC}(t)$. □

We now can proceed with the proof of Theorem 4.9.

Proof:
[of Theorem 4.9]

- As the first case, consider $\mathrm{dh}(t, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) = \omega$. Since t is in argument normal form, the first rewrite step from t must occur at the root. Thus, there are $\ell_1 \to r_1 \in \mathcal{R}$ and a substitution σ_1 such that $t = \ell_1\sigma_1 \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}} r_1\sigma_1$ and $\mathrm{dh}(r_1\sigma_1, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) = \omega$. Hence, there is a minimal subterm $r_1\sigma_1|_{\pi_1}$ of $r_1\sigma_1$ such that $\mathrm{dh}(r_1\sigma_1|_{\pi_1}, \overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}) = \omega$ and all proper subterms of $r_1\sigma_1|_{\pi_1}$ terminate w.r.t. $\overset{\mathsf{i}}{\nrightarrow}\!\!\!\!\!\rightarrow_{\mathcal{R}}$. As σ_1 must instantiate all variables with normal forms, we have $\pi_1 \in \mathcal{P}os_d(r_1)$, i.e.,

$r_1\sigma_1|_{\pi_1} = r_1|_{\pi_1}\sigma_1$. In the infinite $\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}$-reduction of $r_1|_{\pi_1}\sigma_1$, all arguments are again reduced to normal forms first, and we get a term t' with $\mathrm{dh}(t', \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) = \omega$. Since t' is in argument normal form, the first rewrite step from t' must occur at the root. Thus, there are $\ell_2 \to r_2 \in \mathcal{R}$ and a substitution σ_2 such that $t' = \ell_2\sigma_2 \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}} r_2\sigma_2$ and $\mathrm{dh}(r_2\sigma_2, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) = \omega$. This argument can be continued *ad infinitum*, giving rise to an infinite path of chain tree nodes

$$(\ell_1^\sharp \to \mathsf{Com}_{n_1}(\dots, r_1|_{\pi_1}^\sharp, \dots) \mid \sigma_1), \qquad (\ell_2^\sharp \to \mathsf{Com}_{n_2}(\dots, r_2|_{\pi_2}^\sharp, \dots) \mid \sigma_2), \qquad \dots$$

Thus, $\ell_1^\sharp\sigma_1 = t^\sharp$ has an infinite chain tree, and $\mathit{Cplx}_{\langle \mathit{PDT}(\mathcal{R}), \mathit{PDT}(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp) = \omega$.

- Now consider the case where $\mathrm{dh}(t, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \in \mathbb{N}$. We use induction on $\mathrm{dh}(t, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}})$.

  If $\mathrm{dh}(t, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) = 0$, the term t is in normal form w.r.t. $\mathcal{R}$. Thus, $t^\sharp$ is in normal form w.r.t. $\mathit{PDT}(\mathcal{R}) \cup \mathcal{R}$, and $\mathit{Cplx}_{\langle \mathit{PDT}(\mathcal{R}), \mathit{PDT}(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp) = 0$.

  If $\mathrm{dh}(t, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) > 0$, since t is in argument normal form, there are $\ell \to r \in \mathcal{R}$ and a substitution σ such that $t = \ell\sigma \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}} r\sigma = u$ and

$$\mathrm{dh}(t, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) = 1 + \mathrm{dh}(u, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \tag{1}$$

  As σ must instantiate all variables with normal forms, we have that $u|_\pi = r\sigma|_\pi$ is in normal form for all $\pi \in \mathcal{P}os_d(u) \setminus \mathcal{P}os_d(r)$. For these positions π, $u|_\pi\!\Downarrow = u|_\pi$ and $\mathrm{dh}(u|_\pi, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) = 0$. From Lemma 4.12, we get:

$$\begin{aligned}
&\mathrm{dh}(u, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}})\\
&\le \max\left\{ \sum_{1\le i\le k} \mathrm{dh}(u|_{\pi_i}\!\Downarrow, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \;\middle|\; \langle \pi_1, \dots, \pi_k\rangle \in \mathit{MSDC}(u) \right\}\\
&= \max\left\{ \sum_{1\le i\le j} \mathrm{dh}(u|_{\pi_i}\!\Downarrow, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) + \sum_{j+1\le i\le k} \mathrm{dh}(u|_{\pi_i}\!\Downarrow, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \;\middle|\; \begin{matrix} \langle \pi_1, \dots, \pi_k\rangle \in \mathit{MSDC}(u)\\ \pi_1, \dots, \pi_j \in \mathcal{P}os_d(u) \setminus \mathcal{P}os_d(r)\\ \pi_{j+1}, \dots, \pi_k \in \mathcal{P}os_d(r)\end{matrix} \right\}\\
&= \max\left\{ \sum_{1\le i\le j} \mathrm{dh}(u|_{\pi_i}, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \;\; + \sum_{j+1\le i\le k} \mathrm{dh}(u|_{\pi_i}\!\Downarrow, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \;\middle|\; \begin{matrix} \langle \pi_1, \dots, \pi_k\rangle \in \mathit{MSDC}(u)\\ \pi_1, \dots, \pi_j \in \mathcal{P}os_d(u) \setminus \mathcal{P}os_d(r)\\ \pi_{j+1}, \dots, \pi_k \in \mathcal{P}os_d(r)\end{matrix} \right\}\\
&= \max\left\{ \sum_{j+1\le i\le k} \mathrm{dh}(u|_{\pi_i}\!\Downarrow, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \;\middle|\; \begin{matrix} \langle \pi_1, \dots, \pi_k\rangle \in \mathit{MSDC}(u)\\ \pi_1, \dots, \pi_j \in \mathcal{P}os_d(u) \setminus \mathcal{P}os_d(r)\\ \pi_{j+1}, \dots, \pi_k \in \mathcal{P}os_d(r)\end{matrix} \right\}
\end{aligned} \tag{2}$$

  Note that $\mathrm{dh}(u|_\pi\!\Downarrow, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) \le \mathrm{dh}(u|_\pi, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}}) < \mathrm{dh}(t, \overset{\mathsf{i}}{+\!\!\!\!\rightarrow}_{\mathcal{R}})$ holds for all $\pi \in \mathcal{P}os_d(r)$. Thus, with the induction hypothesis, (1) and (2), we get:

$$\begin{aligned}
&\mathrm{dh}(t, \mathrel{+\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \\
&= 1 + \mathrm{dh}(u, \mathrel{+\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \\
&\leq 1 + \max \left\{ \sum_{j+1 \leq i \leq k} \mathit{Cplx}_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle}(u|_{\pi_i} \Downarrow^\sharp) \;\middle|\; \begin{array}{c} \langle \pi_1, \dots, \pi_k \rangle \in \mathit{MSDC}(u) \\ \pi_1, \dots, \pi_j \in \mathcal{P}os_d(u) \setminus \mathcal{P}os_d(r) \\ \pi_{j+1}, \dots, \pi_k \in \mathcal{P}os_d(r) \end{array} \right\}
\end{aligned} \tag{3}$$

Let $\langle \pi_1, \dots, \pi_k \rangle$ be an arbitrary maximal structural dependency chain for r. Then there exists a corresponding chain tree for $t^\sharp$ whose root node is $(\ell^\sharp \to \mathsf{Com}_k(r_1|_{\pi_1} \Downarrow^\sharp, \dots, r_k|_{\pi_k} \Downarrow^\sharp) \mid \sigma)$ and where the children of the root node are maximal chain trees for $u|_{\pi_1} \Downarrow^\sharp, \dots, u|_{\pi_k} \Downarrow^\sharp$. This follows because for all $1 \leq i \leq k$, we have $r|_{\pi_i}\sigma = u|_{\pi_i}$ and so $r|^\sharp_{\pi_i}\sigma \xrightarrow{\mathsf{i}}^*_{\mathcal{R}} u|_{\pi_i} \Downarrow^\sharp$. Together with (3), this gives $\mathrm{dh}(t, \mathrel{+\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) \leq \mathit{Cplx}_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp)$, and for confluent $\mathrel{+\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ we also get $\mathrm{dh}(t, \mathrel{+\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}) = \mathit{Cplx}_{\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle}(t^\sharp)$.

□

From Theorem 4.9, the soundness of our approach to parallel complexity analysis via the DT framework follows analogously to [14]:

Theorem 4.13. (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 $\mathrel{+\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ 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. To analyse parallel complexity of a TRS $\mathcal{R}$ instead of sequential complexity, we need to make only a single adjustment: the input for the DT framework is now the canonical *parallel* DT problem for $\mathcal{R}$ (Definition 4.7) instead of the canonical DT problem (Definition 3.9).

Specifically, with Theorem 4.13 we can use the existing reduction pair processor with CPIs (Theorem 3.14) in the DT framework to get upper bounds for $\mathrm{pirc}_{\mathcal{R}}$.

Example 4.14. (Example 4.4 continued)
For our TRS $\mathcal{R}$ computing the size function on trees, we get the set $PDT(\mathcal{R})$ with the following PDTs:

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

The interpretation $\mathcal{P}ol$ from Example 3.15 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 4.15. Note that $\mathrm{pirc}_{\mathcal{R}}(n)$ can be asymptotically lower than $\mathrm{irc}_{\mathcal{R}}(n)$, for instance for the TRS $\mathcal{R}$ of Section 2 with the following rules:

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

The upper bound $\mathrm{irc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$ is tight: from $\mathsf{doubles}(\mathsf{S}(\mathsf{S}(\ldots\mathsf{S}(\mathsf{Zero})\ldots)))$, 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:

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

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

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

Theorem 4.16. (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 $t_0 \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}} t_1 \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}} t_2 \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}} \ldots$, also $t_0 \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_1 \xrightarrow{\mathsf{i}}_{\mathcal{R}} t_2 \xrightarrow{\mathsf{i}}_{\mathcal{R}} \ldots$ holds (i.e., from basic terms, $\mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}}$ and $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ coincide);

(c) $\mathrm{pirc}_{\mathcal{R}}(n) = \mathrm{irc}_{\mathcal{R}}(n)$.

Proof:
Let $\mathcal{R}$ be a TRS such that for all rules $\ell \to r \in \mathcal{R}$, $|MSDC(r)| = 1$.

We prove part (a) by showing that for each rule $\ell \to r \in \mathcal{R}$, we have $PDT(\ell \to r) = \{DT(\ell \to r)\}$. Let $\ell \to r \in \mathcal{R}$. By construction of PDT, we get from $|MSDC(r)| = 1$ that $|PDT(\ell \to r)| = 1$. $|MSDC(r)| = 1$ implies that $\mathcal{P}os_d(r)$ is ordered by the prefix order $>$ on positions. Thus, by using the extension of $>$ to the lexicographic order on positions $\gtrdot$ used as an ingredient for the construction of $DT(\ell \to r)$, we obtain the result for part (a).

We now prove part (b). Let t_0 be a basic term for $\mathcal{R}$ with a rewrite sequence $t_0 \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}} t_1 \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}} t_2 \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\!\rightarrow}}_{\mathcal{R}} \ldots$. We show by induction over i that for all t_i, t_i contains at most one innermost redex.

For the base case, consider that the basic term t_0 contains only a single occurrence of a defined symbol, at the root. Thus, if t_0 is a redex, it is also the unique innermost redex in t_0.

For the induction step, assume that t_i has at most one innermost redex. If t_i has no redex, it is a normal form, and we are done. Otherwise, t_i has exactly one innermost redex at position τ, and in the parallel-innermost rewrite step $t_i \overset{\mathsf{i}}{\twoheadrightarrow}_{\mathcal{R}} t_{i+1}$ a rule $\ell \to r$ with matcher σ replaces $t_i|_\tau = \sigma(\ell)$ by $\sigma(r)$. The premise $|MSDC(r)| = 1$ implies that there is exactly one (empty or non-empty) maximal structural dependency chain $\langle \pi_1, \ldots, \pi_k \rangle \in MSDC(r)$.

Since the rewrite step $t_i \overset{\mathsf{i}}{\twoheadrightarrow}_{\mathcal{R}} t_{i+1}$ uses (parallel-)*innermost* rewriting, $\sigma(x)$ is in normal form for all variables x. Thus, potential redexes in term t_{i+1} can only be at positions $\tau.\pi_1, \ldots, \tau.\pi_k$. As $\langle \pi_1, \ldots, \pi_k \rangle$ is a structural dependency chain, we have $\pi_1 > \cdots > \pi_k$, which implies $\tau.\pi_1 > \cdots > \tau.\pi_k$. Thus, the term t_{i+1} has at most one innermost redex $\tau\pi_i$. This concludes part (b).

Part (c) follows directly from part (b) and the definitions of $\mathrm{pirc}_{\mathcal{R}}(n)$ and $\mathrm{irc}_{\mathcal{R}}(n)$. □

Thus, for TRSs $\mathcal{R}$ where Theorem 4.16 applies, no rewrite rule can introduce parallel redexes, and specific analysis techniques for $\mathrm{pirc}_{\mathcal{R}}$ are not needed.

5. 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 $\overset{\mathsf{i}}{\twoheadrightarrow}_{\mathcal{R}}$, 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 [25], usable replacement maps [26], the Combination Framework [27], a transformation to complexity problems for integer transition systems [28], amortised complexity analysis [30], or techniques for finding *lower* bounds [29]. 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 [18] and TCT [19] are able to find bounds on (innermost) runtime complexity of such rewrite systems.

Definition 5.1. (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 w.r.t. $\mathcal{R}_1 \cup \mathcal{R}_2$.

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

The rewrite relation $\xrightarrow{\text{i}}_{\mathcal{R}_1/\mathcal{R}_2}$ is essentially the same as $\xrightarrow{\text{i}}_{\mathcal{R}_1\cup\mathcal{R}_2}$, 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 5.2. Consider a variant of Example 3.1 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 OCaml function from Section 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 $\text{dh}(\mathsf{plus}(\mathsf{S}^n(\mathsf{Zero}), \mathsf{S}^m(\mathsf{Zero})), \xrightarrow{\text{i}}_{\mathcal{R}_1\cup\mathcal{R}_2})$ $= n+1$, we would then get $\text{dh}(\mathsf{plus}(\mathsf{S}^n(\mathsf{Zero}), \mathsf{S}^m(\mathsf{Zero})), \xrightarrow{\text{i}}_{\mathcal{R}_1/\mathcal{R}_2}) = 1$, 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 5.3. (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 Theorem 5.8, this information can be recovered.)

The following example is taken from the *Termination Problem Data Base (TPDB)* [38], a collection of examples used at the annual *Termination and Complexity Competition (termCOMP)* [39, 40] (see also Section 7):

Example 5.4. (TPDB, `HirokawaMiddeldorp_04/t002`)
Consider the following TRS $\mathcal{R}$ from category `Innermost_Runtime_Complexity` of the TPDB:

$$\begin{array}{rcl|rcl}
\mathsf{leq}(\mathsf{0}, y) & \to & \mathsf{True} & \mathsf{if}(\mathsf{True}, x, y) & \to & x \\
\mathsf{leq}(\mathsf{S}(x), \mathsf{0}) & \to & \mathsf{False} & \mathsf{if}(\mathsf{False}, x, y) & \to & y \\
\mathsf{leq}(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{leq}(x, y) & -(x, \mathsf{0}) & \to & x \\
\mathsf{mod}(\mathsf{0}, y) & \to & \mathsf{0} & -(\mathsf{S}(x), \mathsf{S}(y)) & \to & -(x, y) \\
\mathsf{mod}(\mathsf{S}(x), \mathsf{0}) & \to & \mathsf{0} & & &
\end{array}$$

$$\mathsf{mod}(\mathsf{S}(x), \mathsf{S}(y)) \to \mathsf{if}(\mathsf{leq}(y, x), \mathsf{mod}(-(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{S}(y)), \mathsf{S}(x))$$

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

$$\begin{array}{rcl|rcl}
\mathsf{leq}^\sharp(\mathsf{0}, y) & \to & \mathsf{Com}_0 & \mathsf{if}^\sharp(\mathsf{True}, x, y) & \to & \mathsf{Com}_0 \\
\mathsf{leq}^\sharp(\mathsf{S}(x), \mathsf{0}) & \to & \mathsf{Com}_0 & \mathsf{if}^\sharp(\mathsf{False}, x, y) & \to & \mathsf{Com}_0 \\
\mathsf{leq}^\sharp(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{Com}_1(\mathsf{leq}^\sharp(x, y)) & -^\sharp(x, \mathsf{0}) & \to & \mathsf{Com}_0 \\
\mathsf{mod}^\sharp(\mathsf{0}, y) & \to & \mathsf{Com}_0 & -^\sharp(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{Com}_1(-^\sharp(x, y)) \\
\mathsf{mod}^\sharp(\mathsf{S}(x), \mathsf{0}) & \to & \mathsf{Com}_0 & & &
\end{array}$$

$$\begin{array}{rcl}
\mathsf{mod}^\sharp(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{Com}_2(\mathsf{leq}^\sharp(y, x), \mathsf{if}^\sharp(\mathsf{leq}(y, x), \mathsf{mod}(-(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{S}(y)), \mathsf{S}(x))) \\
\mathsf{mod}^\sharp(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{Com}_3(-^\sharp(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{mod}^\sharp(-(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{S}(y)), \\
& & \qquad \mathsf{if}^\sharp(\mathsf{leq}(y, x), \mathsf{mod}(-(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{S}(y)), \mathsf{S}(x)))
\end{array}$$

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

Remark 5.5. One of the reviewers suggested that the use of if in Example 5.4 indicated that TRSs with innermost rewriting were inherently unable to provide a faithful representation of the evaluation strategy for conditional statements used in programming languages with call-by-value evaluation. The reason was that in the recursive mod rule, the function calls in the subterm $\mathsf{mod}(-(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{S}(y))$ are evaluated also if the call to $\mathsf{leq}(y, x)$ rewrites to False so that the "then"-branch of the conditional evaluation would never be needed. In contrast, a language like OCaml, C++, Rust, ... would evaluate the "then"-branch only after the $\mathsf{leq}(y, x)$ had evaluated to True.

There are several ways of dealing with this modelling issue in term rewriting. One way is to impose a context-sensitive rewrite strategy [41] that "freezes" the second and third argument of if. Like this, the evaluation of these arguments is delayed until after the result of evaluating the first argument is known, and only the appropriate branch is evaluated.

Another way that does not necessitate a different rewrite strategy is to reorganise the rewrite rules with a dedicated symbol cond for the *specific* conditional expression rather than using generic if rules. In Example 5.4, we could replace the recursive mod rule with the following rules (and potentially remove the if rules, which would no longer be needed):

$$\begin{array}{rcl}
\mathsf{mod}(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{cond}(\mathsf{leq}(y, x), x, y) \\
\mathsf{cond}(\mathsf{True}, x, y) & \to & \mathsf{mod}(-(\mathsf{S}(x), \mathsf{S}(y)), \mathsf{S}(y)) \\
\mathsf{cond}(\mathsf{False}, x, y) & \to & \mathsf{S}(x)
\end{array}$$

In this way, an innermost rewrite sequence would require first evaluating the instance of $\mathsf{leq}(y, x)$ to normal form, and depending on the result True or False, the corresponding cond rule will evaluate (only) the corresponding branch of the earlier if statement. We refrained from doing so in Example 5.4 to keep the link with an existing TRS from the literature that reflects different modelling choices and that our method should be able to analyse as well.

From the definition of relative TRS, we are now able to prove an upper bound (Theorem 5.6) as well as a lower bound (Theorem 5.8) for parallel complexities.

Theorem 5.6. (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 $\mathit{Cplx}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \leq \mathrm{dh}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$, and

(b) $\mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) \leq \mathrm{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n)$.

Proof:
We first show part (a) of the statement. For a DT Problem $\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle$ and a term $t^\sharp \in \mathcal{T}^\sharp$, consider an arbitrary chain tree T. We will show that if $|T|_{\mathcal{S}} = n$, then also $\mathrm{dh}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \geq n$. We consider two cases.

- First, $n = \omega$. The set of all dependency tuples is finite, thus a finite term can only have a finite number of immediate successors. Therefore, T is finitely branching, so there must be an infinite path with infinitely many nodes of the form $(u_1^\sharp \to \mathsf{Com}_{n_1}(\ldots, v_1^\sharp, \ldots) \mid \sigma_1), (u_2^\sharp \to \mathsf{Com}_{n_2}(\ldots, v_2^\sharp, \ldots) \mid \sigma_2), \ldots$ such that $u_1^\sharp \to \mathsf{Com}_{n_1}(\ldots, v_1^\sharp, \ldots), u_2^\sharp \to \mathsf{Com}_{n_2}(\ldots, v_2^\sharp, \ldots), \ldots \in \mathcal{D}$. For infinitely many $i_1 < i_2 < i_3 < \ldots$, we also have $u_i^\sharp \to \mathsf{Com}_{n_i}(\ldots, v_i^\sharp, \ldots) \in \mathcal{S}$, and for all i, we have $v_i^\sharp\sigma_i \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} u_{i+1}^\sharp\sigma_{i+1}$. Then we also have a corresponding infinite rewrite sequence

$$\begin{aligned} t^\sharp = u_1^\sharp\sigma_1 \xrightarrow{\mathsf{i}}{}^*_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})} &\; C_1[u_{i_1}^\sharp\sigma_{i_1}] \xrightarrow{\mathsf{i}}_{\mathcal{S}} C_1[\mathsf{Com}_{n_{i_1}}(\ldots, v_{i_1}^\sharp, \ldots)\sigma_{i_1}] \\ \xrightarrow{\mathsf{i}}{}^*_{(\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}} &\; C_2[u_{i_2}^\sharp\sigma_{i_2}] \xrightarrow{\mathsf{i}}_{\mathcal{S}} C_2[\mathsf{Com}_{n_{i_2}}(\ldots, v_{i_2}^\sharp, \ldots)\sigma_{i_2}] \\ \xrightarrow{\mathsf{i}}{}^*_{(\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}} &\; \cdots \end{aligned}$$

for some contexts $C_1, C_2, \ldots$ (which result from rewrite steps with rules from $\mathcal{D}$).

- Now consider the case $n \in \mathbb{N}$. We use induction. For $n = 0$, the statement trivially holds. For the induction step, let $n > 0$.

  The (potentially infinite) chain tree T has m subtrees T_i' with roots $(u_i^\sharp \to \mathsf{Com}_{q_i}(v_{i,1}^\sharp, \ldots, v_{i,q_i}^\sharp) \mid \sigma_i)$ such that $u_i^\sharp \to \mathsf{Com}_{q_i}(v_{i,1}^\sharp, \ldots, v_{i,q_i}^\sharp) \in \mathcal{S}$ and the path in the chain tree from the root to $(u_i^\sharp \to \mathsf{Com}_{q_i}(v_{i,1}^\sharp, \ldots, v_{i,q_i}^\sharp) \mid \sigma_i)$ has no outgoing edges from a node with a DT in $\mathcal{S}$.

  We show two separate statements in the induction step, which together let us conclude that $\mathrm{dh}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \geq |T_1'|_{\mathcal{S}} + \cdots + |T_m'|_{\mathcal{S}} = n$:

  $$\text{For each } T_i', \text{ the term } u_i^\sharp\sigma_i \text{ has } \mathrm{dh}(u_i^\sharp\sigma_i, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \geq |T_i'|_{\mathcal{S}}. \tag{4}$$

  $$\begin{gathered} \text{There are contexts } C_1, \ldots, C_m \text{ such that} \\ t^\sharp \xrightarrow{\mathsf{i}}{}^*_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})} \mathsf{Com}_m(C_1[u_1^\sharp\sigma_1], \ldots, C_m[u_m^\sharp\sigma_m]). \end{gathered} \tag{5}$$

  On (4): Let $i \in \{1, \ldots, m\}$ be arbitrary and fixed, let $u = u_i$, let $\sigma = \sigma_i$, let $T' = T_i'$ (to ease notation). T' is a chain tree for $u\sigma$ and its root is $(u^\sharp \to \mathsf{Com}_q(v_1^\sharp, \ldots, v_q^\sharp) \mid \sigma)$. Let this node have children $N_1 = (w_1^\sharp \to \mathsf{Com}_{r_1}(\ldots) \mid \mu_1)$, ..., $N_q = (w_q^\sharp \to \mathsf{Com}_{r_q}(\ldots) \mid \mu_q)$. For the corresponding trees T_j'' with N_j at the root, we have $|T_j''|_{\mathcal{S}} < |T'|_{\mathcal{S}} \le n$ by construction, so the induction hypothesis is applicable to the terms $w_j^\sharp\mu_j$, and we get $\mathrm{dh}(w_j^\sharp\mu_j, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$ $\ge |T_j''|_{\mathcal{S}}$ for all $1 \le j \le q$. We construct a rewrite sequence with $\xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}$ using at least $1 + |T_1''|_{\mathcal{S}} + \cdots + |T_q''|_{\mathcal{S}} = |T'|_{\mathcal{S}}$ steps with a rule from $\mathcal{S}$ as follows:

$$\begin{aligned} u^\sharp\sigma &\xrightarrow{\mathsf{i}}_{\mathcal{S}} \mathsf{Com}_q(v_1^\sharp\sigma, \ldots, v_q^\sharp\sigma) \\ &\xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} \mathsf{Com}_q(w_1^\sharp\mu_1, \ldots, v_q^\sharp\sigma) \\ &\xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} \cdots \\ &\xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} \mathsf{Com}_q(w_1^\sharp\mu_1, \ldots, w_q^\sharp\mu_q) \end{aligned}$$

With this rewrite sequence, we obtain (4) using the induction hypothesis:

$$\begin{aligned} &\mathrm{dh}(u^\sharp\sigma, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \\ \ge\ &1 + \mathrm{dh}(w_1^\sharp\mu_1, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) + \cdots + \mathrm{dh}(w_q^\sharp\mu_q, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \\ \ge\ &1 + |T_1''|_{\mathcal{S}} + \cdots + |T_q''|_{\mathcal{S}} \\ =\ &|T'|_{\mathcal{S}} \end{aligned}$$

On (5): Let the root of T be $(\ell^\sharp \to \mathsf{Com}_p(r_1^\sharp, \ldots, r_p^\sharp) \mid \nu)$. With a construction similar to the one used in the case $n = \omega$, we get:

$$\begin{aligned} t^\sharp = \ell^\sharp\nu &\xrightarrow{\mathsf{i}}_{\mathcal{D}} && \mathsf{Com}_p(r_1^\sharp\nu, \ldots, r_p^\sharp\nu) \\ &\xrightarrow{\mathsf{i}}{}^*_{(\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}} && \mathsf{Com}_p(C_1[u_1^\sharp\sigma_1], \ldots, r_p^\sharp\nu) \\ &\xrightarrow{\mathsf{i}}{}^*_{(\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}} && \cdots \\ &\xrightarrow{\mathsf{i}}{}^*_{(\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}} && \mathsf{Com}_p(C_1[u_1^\sharp\sigma_1], \ldots, C_m[u_m^\sharp\sigma_m]) \end{aligned}$$

for some contexts $C_1, \ldots, C_m$ (which result from rewrite steps with rules from $\mathcal{D}$). Note that here it suffices to reduce only in those subterms with a symbol $f^\sharp$ at their root that are on a path to one of the $C_i[u_i^\sharp\sigma_i]$, and depending on the tree structure, each $r_j^\sharp\nu$ may yield 0 or more of these m terms (note that p and m are not necessarily equal).

This concludes the induction step and hence the overall proof of part (a).

Part (b) follows from part (a), as shown in the following:

$$\begin{aligned} \mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) &= \sup\{\mathit{Cplx}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \mid t \in \mathcal{T}^{\mathcal{R}}_{\mathrm{basic}}, |t| \le n\} && \text{by Definition 3.9} \\ &\le \sup\{\mathrm{dh}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \mid t \in \mathcal{T}^{\mathcal{R}}_{\mathrm{basic}}, |t| \le n\} && \text{by part (a)} \\ &\le \sup\{\mathrm{dh}(s, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \mid s \in \mathcal{T}^{\mathcal{R}\cup\mathcal{D}}_{\mathrm{basic}}, |s| \le n\} \\ &= \mathrm{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n) \end{aligned}$$

□

Example 5.7. (Example 5.4 continued)
For the relative TRS $PDT(\mathcal{R})/\mathcal{R}$ from Example 5.4, the tool APROVE uses a transformation to integer transition systems [28] followed by an application of the complexity analysis tool COFLOCO [42, 43] to find a bound $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n) \in \mathcal{O}(n)$ and to deduce the bound $\mathrm{pirc}_{\mathcal{R}}(n) \in \mathcal{O}(n)$ for the original TRS $\mathcal{R}$ from the TPDB. In contrast, using the techniques of Section 4 without the transformation to a relative TRS from Definition 5.3, APROVE finds only a bound $\mathrm{pirc}_{\mathcal{R}}(n) \in \mathcal{O}(n^2)$.

Intriguingly, we can use our transformation from Definition 5.3 not only for finding upper bounds, but also for *lower* bounds on $\mathrm{pirc}_{\mathcal{R}}$.

Theorem 5.8. (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 $t \in \mathcal{T}^{\mathcal{D}\cup\mathcal{R}}_{\text{basic}}$, $\mathit{Cplx}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^\sharp) \geq \mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$, and

(b) $\mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n) \geq \mathrm{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n)$.

Proof:
We first consider the proof for part (a).

We use the following (many-sorted first-order monomorphic) type assignment Θ with two sorts α and β, where the arities of the symbols are respected (note that here all arguments of a given symbol have the same type):

$$\begin{aligned}\Theta(f) &= \alpha \times \cdots \times \alpha \to \alpha \text{ for } f \text{ in } \Sigma^{\mathcal{R}}\\ \Theta(f^\sharp) &= \alpha \times \cdots \times \alpha \to \beta \text{ for } f^\sharp \text{ a sharp symbol}\\ \Theta(\mathsf{Com}_k) &= \beta \times \cdots \times \beta \to \beta\end{aligned}$$

With this type assignment, for all rules $\ell \to r \in \mathcal{D} \cup \mathcal{R}$, ℓ and r are well typed and have the same type: if $\ell \to r \in \mathcal{R}$, then all occurring symbols have the same result type α, which carries over to ℓ and r. And if $\ell \to r \in \mathcal{D}$, then ℓ and r have type β. To see that ℓ and r are well typed, consider that every term ℓ has the shape $f^\sharp(s_1, \ldots, s_n)$, where $f^\sharp$ has result type β and expects all arguments to have type α, while all s_i contain only subterms of type α. Similarly, r has the shape $\mathsf{Com}_k(f_1^\sharp(t_{1,1}, \ldots, t_{1,n_1}), \ldots, f_k^\sharp(t_{k,1}, \ldots, t_{k,n_k}))$. Com_k has result type β and expects all arguments to have type β. This is the case since all $f_i^\sharp$ have result type β. And all $f_i^\sharp$, which are right below the root, expect their arguments $t_{i,j}$ to have result type α. This is the case by construction.

In the following, we consider basic terms that are well typed according to Θ as start terms. For our relative TRS $\mathcal{S}/((\mathcal{D} \setminus \mathcal{S}) \cup \mathcal{R})$, we have the following two kinds of well-typed basic terms that we need to consider:

Case 1: $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma_d$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{V})$. This term and all its subterms are of type α. Thus, this term can be rewritten by rules from $\mathcal{R}$, but not by rules from $\mathcal{D}$ (and $\mathcal{S}$), which all have type β. As rewriting preserves the type of terms, t is a normal form w.r.t. the relations $\xrightarrow{\mathrm{i}}_{\Theta(\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}))}$ and $\xrightarrow{\mathrm{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}$, and $\mathrm{dh}(t, \xrightarrow{\mathrm{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) = 0$. Since $\mathit{Cplx}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(s) \geq 0$ regardless of s, the claim follows for this case.

Case 2: $t = f^\sharp(t_1, \ldots, t_n)$ with $f \in \Sigma_d$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma_c, \mathcal{V})$. If t is a normal form, there is no tree, and $\mathrm{dh}(t, \overset{\mathrm{i}}{\to}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) = 0 = \mathit{Cplx}_{\langle \mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t)$.

Otherwise, we can convert any $\overset{\mathrm{i}}{\to}_{\mathcal{S}\cup((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})} = \overset{\mathrm{i}}{\to}_{\mathcal{D}\cup\mathcal{R}}$ rewrite sequence to a $(\mathcal{D}, \mathcal{R})$-chain tree T for t such that $\mathrm{dh}(t, \overset{\mathrm{i}}{\to}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) = |T|_{\mathcal{S}}$, including any rewrite sequence that witnesses $\mathrm{dh}(t, \overset{\mathrm{i}}{\to}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$ in the following way:

As t is a basic term, the first step in the rewrite sequence rewrites at the root of the term. Since only rules from $\mathcal{D}$ are applicable to terms with $f^\sharp$ at the root, this step uses a DT $s^\sharp \to \mathsf{Com}_k(\ldots)$ from $\mathcal{D}$. With σ as the used matcher for the rewrite step, we obtain the root node $(s^\sharp \to \mathsf{Com}_k(\ldots) \mid \sigma)$.

Now assume that we have a partially constructed chain tree T' for the rewrite sequence so far, which we have represented up until the term s that resulted from a $\overset{\mathrm{i}}{\to}_{\mathcal{D}}$ step.

If there are no further $\overset{\mathrm{i}}{\to}_{\mathcal{D}}$ steps in the rewrite sequence, we have completed our chain tree $T = T'$ as the remaining $\overset{\mathrm{i}}{\to}_{\mathcal{R}}$ suffix of the rewrite sequence does not contribute to $\mathrm{dh}(t, \overset{\mathrm{i}}{\to}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})})$ (only steps using rules from $\mathcal{S} \subseteq \mathcal{D}$ are counted).

Otherwise, our remaining rewrite sequence has the shape $s \overset{\mathrm{i}}{\to}^*_{\mathcal{R}} u \overset{\mathrm{i}}{\to}_{\mathcal{D}} v \overset{\mathrm{i}}{\to}^m_{\mathcal{D}\cup\mathcal{R}} \ldots$ for some $m \in \mathbb{N} \cup \{\omega\}$. The step $u \overset{\mathrm{i}}{\to}_{\mathcal{D}} v$ takes place at position π, using the DT $p^\sharp \to \mathsf{Com}_l(q_1^\sharp, \ldots, q_l^\sharp) \in \mathcal{D}$ and the matcher μ.

We can reorder the rewrite steps $s \overset{\mathrm{i}}{\to}^*_{\mathcal{R}} u$ by advancing all $\overset{\mathrm{i}}{\to}_{\mathcal{R}}$ steps at positions $\tau > \pi$, yielding $s \overset{\mathrm{i}}{\to}^*_{\mathcal{R},>\pi} s' \overset{\mathrm{i}}{\to}^*_{\mathcal{R},\not>\pi} u$. (This reordering is possible in our innermost setting.) Here $\overset{\mathrm{i}}{\to}_{\mathcal{R},>\pi}$ denotes an innermost rewrite step using rules from $\mathcal{R}$ at a position $\tau > \pi$, and $\overset{\mathrm{i}}{\to}_{\mathcal{R},\not>\pi}$ denotes an innermost rewrite step using rules from $\mathcal{R}$ at a position $\tau' \not> \pi$. Now we change our remaining rewrite sequence to $s \overset{\mathrm{i}}{\to}^*_{\mathcal{R},>\pi} s' \overset{\mathrm{i}}{\to}_{\mathcal{D}} u' \overset{\mathrm{i}}{\to}^*_{\mathcal{R},\not>\pi} v \overset{\mathrm{i}}{\to}^m_{\mathcal{D}\cup\mathcal{R}} \ldots$. Let $\mathsf{Com}_k(q_1'^\sharp, \ldots, q_k'^\sharp)\delta = s|_\pi$. Since the $s' \overset{\mathrm{i}}{\to}_{\mathcal{D}} u'$ rewrite step has not been encoded yet, there is a j such that $q_j'^\sharp\delta \overset{\mathrm{i}}{\to}_{\mathcal{R}} p^\sharp\mu$ has not yet been used in the construction. Therefore, there exists a node $N = (p'^\sharp \to \mathsf{Com}_k(q_1'^\sharp, \ldots, q_k'^\sharp) \mid \delta)$ where for some j, the subterm $q_j'^\sharp\delta$ in the DT of N has not yet been used for this purpose in the construction before. We encode the subsequence $s \overset{\mathrm{i}}{\to}^*_{\mathcal{R},>\pi} s' \overset{\mathrm{i}}{\to}_{\mathcal{D}} u'$ by adding the node $(p^\sharp \to \mathsf{Com}_l(q_1^\sharp, \ldots, q_l^\sharp) \mid \mu)$ to T' as a child to N.

We obtain the chain tree T'', which we extend further by encoding the rewrite sequence $u' \overset{\mathrm{i}}{\to}^*_{\mathcal{R},\not>\pi} v \overset{\mathrm{i}}{\to}^m_{\mathcal{D}\cup\mathcal{R}} \ldots$ following the same procedure.

Since our construction adds a node with a DT from $\mathcal{S}$ in the first component of the label whenever the rewrite sequence uses a rule from $\mathcal{S}$, we have $\mathrm{dh}(t, \overset{\mathrm{i}}{\to}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) = |T|_{\mathcal{S}}$ as desired. This concludes the proof for part (a).

We now prove part (b).

Innermost runtime complexity is known to be a persistent property w.r.t. type introduction [44]. For our relative TRS $\mathcal{S}/((\mathcal{D} \setminus \mathcal{S}) \cup \mathcal{R})$, this means that we may introduce an arbitrary (many-sorted first-order monomorphic) type assignment Θ for all symbols in the considered signature such that the rules in $\mathcal{R} \cup \mathcal{D}$ are well typed. We obtain a typed relative TRS $\Theta(\mathcal{S}/((\mathcal{D} \setminus \mathcal{S}) \cup \mathcal{R}))$, and $\mathrm{irc}_{\Theta(\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}))}(n) = \mathrm{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n)$ holds. Thus, only basic terms that are well typed according to Θ need to be considered as start terms for $\mathrm{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}$. We write $\Theta(\mathcal{T}_{\text{basic}}^{\mathcal{D}\cup\mathcal{R}})$ for the set of well-typed basic terms for $\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})$.

We use the type assignment Θ from part (a) to restrict the set of basic terms as start terms. With this type assignment, we obtain:

$$\begin{aligned}
&\mathrm{irc}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}(n) && \\
=\;& \mathrm{irc}_{\Theta(\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R}))}(n) && \text{by [44]}\\
=\;& \sup\{\mathrm{dh}(t, \xrightarrow{\mathsf{i}}_{\mathcal{S}/((\mathcal{D}\setminus\mathcal{S})\cup\mathcal{R})}) \mid t \in \Theta(\mathcal{T}_{\text{basic}}^{\mathcal{R}\cup\mathcal{D}}), |t| \le n\} && \text{by Definition 5.1}\\
\le\;& \sup\{\mathit{Cplx}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^{\sharp}) \mid t \in \Theta(\mathcal{T}_{\text{basic}}^{\mathcal{R}\cup\mathcal{D}}), |t| \le n\} && \text{by part (a)}\\
\le\;& \sup\{\mathit{Cplx}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(t^{\sharp}) \mid t \in \mathcal{T}_{\text{basic}}^{\mathcal{R}\cup\mathcal{D}}, |t| \le n\} && \text{drop types} \Rightarrow \text{more start terms}\\
\le\;& \mathrm{irc}_{\langle\mathcal{D},\mathcal{S},\mathcal{R}\rangle}(n)
\end{aligned}$$

□

Theorem 5.6 and Theorem 5.8 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.

Example 5.9. (Example 5.7 and Example 6.13 continued)
We continue Example 5.7. Theorem 5.8 implies that a lower bound for $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n)$ of the relative TRS $PDT(\mathcal{R})/\mathcal{R}$ from Example 5.4 carries over to $\langle PDT(\mathcal{R}), PDT(\mathcal{R}), \mathcal{R}\rangle$ and, presuming that $\overset{\mathsf{i}}{\mathrel{+\!\!\!\rightarrow}}_{\mathcal{R}}$ is confluent, also to $\mathrm{pirc}_{\mathcal{R}}(n)$ of the original TRS $\mathcal{R}$ from the TPDB. AProVE uses rewrite lemmas [29] to find the lower bound $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}(n) \in \Omega(n)$. Together with Example 5.7, we have automatically inferred that this complexity bound is *tight* if we can also prove confluence of $\overset{\mathsf{i}}{\mathrel{+\!\!\!\rightarrow}}_{\mathcal{R}}$: $\mathrm{pirc}_{\mathcal{R}}(n) \in \Theta(n)$. We shall see the missing confluence proof for $\overset{\mathsf{i}}{\mathrel{+\!\!\!\rightarrow}}_{\mathcal{R}}$ in Section 6.

Note that Theorem 4.13 requires confluence of $\overset{\mathsf{i}}{\mathrel{+\!\!\!\rightarrow}}_{\mathcal{R}}$ to derive lower bounds for $\mathrm{pirc}_{\mathcal{R}}$ from lower complexity bounds of the canonical parallel DT problem. So to use Theorem 5.8 to search for *lower* complexity bounds with existing techniques [29], we need a criterion for confluence of parallel-innermost rewriting. Section 6 shall be dedicated to proposing two sufficient syntactic criteria for confluence that can be checked automatically.

6. Confluence of parallel-innermost rewriting

Recall that confluence of a relation $\to$ means that whenever we have $t_1 \to^* t_2$ and $t_1 \to^* t_3$, there is also some t_4 with $t_2 \to^* t_4$ and $t_3 \to^* t_4$. In other words, if $\to$ is confluent, any non-determinism between steps with $\to$ that *temporarily* leads to different outcomes can always be undone to reach a common object. Methods for analysis of confluence have been a topic of interest for many years (see, e.g., [45, 46, 47] for early work), motivated both by applications in theorem proving and as a topic of study in its own right. In recent years, the development of automated tools for confluence analysis of (sequential) term rewriting has flourished. This is witnessed by the Confluence Competition [48], which has been running annually since 2012 to compare state-of-the-art tools for automated confluence analysis. However, we are not aware of any tool at the Confluence Competition that currently supports the analysis of parallel-innermost rewriting.

As an alternative, it would be tempting to use an existing tool for confluence analysis of standard rewriting, possibly restricted to innermost rewriting, as a decidable sufficient criterion for confluence of parallel-innermost rewriting. However, the following example shows that this approach is in general not sound.

Example 6.1. (Confluence of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ or $\to_{\mathcal{R}}$ does not Imply Confluence of $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$)
To see that we cannot prove confluence of $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ just by using a standard off-the-shelf tool for confluence analysis of innermost or full rewriting [48], 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 $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ and $\to_{\mathcal{R}}$ are confluent. However, $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is not confluent: we can rewrite both $\mathsf{a} \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{b})$ and $\mathsf{a} \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{c})$, yet there is no term v such that $\mathsf{f}(\mathsf{b},\mathsf{b}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}^{*} v$ and $\mathsf{f}(\mathsf{b},\mathsf{c}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}^{*} v$. The reason is that the only possible rewrite sequences with $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ from these terms are $\mathsf{f}(\mathsf{b},\mathsf{b}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{c},\mathsf{c}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{b}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \dots$ and $\mathsf{f}(\mathsf{b},\mathsf{c}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{c},\mathsf{b}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{f}(\mathsf{b},\mathsf{c}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \dots$, with no terms in common.

Thus, in general a confluence proof for $\to_{\mathcal{R}}$ or $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ does not imply confluence for $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$. Yet it seems that other criteria on $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ or $\to_{\mathcal{R}}$ may be sufficient: intuitively, the reason for non-confluence for $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ in Example 6.1 is the non-termination of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.

Proposition 6.2. Let $\mathcal{R}$ be a TRS whose innermost rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is terminating. Then $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is confluent iff $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is confluent.

The next proposition is motivated by applying techniques for proving confluence of $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$, as developed in this paper, to proving confluence of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.

Proposition 6.3. Let $\mathcal{R}$ be a (not necessarily terminating) TRS. If $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is confluent, then $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is confluent.

We conjectured Proposition 6.2 first in our informal extended abstract [49, Conjecture 1] and Proposition 6.3 in the preliminary conference version of this paper [13, Conjecture 1]. We are grateful to van Oostrom for providing proofs for both statements [35] and closing our conjectures.

6.1. Confluence of $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ for non-overlapping rules

We recall the standard notions of *uniformly confluent* and *deterministic* relations, which are special cases of confluent relations. In the following, we will use these notions to identify sufficient criteria for confluence of parallel-innermost term rewriting.

Definition 6.4. A relation $\to$ is *uniformly confluent* iff $s \to t$ and $s \to u$ imply that $t = u$ or that there exists an object v with $t \to v$ and $u \to v$, and $\to$ is *deterministic* iff for every s there is at most one t with $s \to t$.

Let us work towards a first sufficient criterion for confluence of parallel-innermost rewriting. 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 the same term u. For $\mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$, 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 $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$.

Towards a sufficient criterion for confluence of parallel-innermost rewriting, we introduce the following standard definitions used in confluence analysis:

Definition 6.5. (Unifier, Most General Unifier, see also [24])
Two terms s and t *unify* iff there exists a substitution σ (called a *unifier* of s and t) such that $s\sigma = t\sigma$. A unifier σ of s and t is called a *most general unifier* of s and t iff for all unifiers δ of s and t there exists some substitution δ' such that $(s\sigma)\delta' = s\delta = t\delta = (t\sigma)\delta'$.

Critical pairs capture the local non-determinism that arises if a given redex may be rewritten by different rewrite rules or at different positions in the same redex. They are defined with the help of most general unifiers to determine which instances of left-hand sides may lead to local non-determinism.

Definition 6.6. (Critical Pair, Critical Peak, Critical Overlay [45], see also [24])
For a given TRS $\mathcal{R}$, let $\ell \to r, u \to v \in \mathcal{R}$ be rules whose variables have been renamed apart and let π be a position in u such that $\ell|_\pi \notin \mathcal{V}$. If $\pi = \varepsilon$, we require that $\ell \to r$ and $u \to v$ are not variants of the same rule, i.e., that we cannot obtain $\ell \to r$ by renaming variables in $u \to v$. If ℓ and $u|_\pi$ unify with most general unifier σ, then we call $u\sigma[r\sigma]_\pi \rtimes v\sigma$ a *critical pair*, resulting from the *critical peak* (i.e., local non-determinism) between the steps $u\sigma \to_{\mathcal{R}} v\sigma$ and $u\sigma = u\sigma[\ell\sigma]_\pi \to_{\mathcal{R}} u\sigma[r\sigma]_\pi$. If $\pi = \varepsilon$, we call the critical pair $u\sigma[r\sigma]_\pi \rtimes v\sigma = r\sigma \rtimes v\sigma$ a *critical overlay*, and we may write $r\sigma \bowtie v\sigma$.

A critical peak is the concrete local non-determinism for rewriting a term in two different ways (with overlapping redexes, or using different rewrite rules). It results in a critical pair that describes this non-determinism in an abstract way. Finite TRSs have only finitely many critical pairs, which is very useful for analysis of confluence.

Example 6.7. Consider the (highly artificial, but illustrative) TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{b}, \mathsf{f}(x) \to \mathsf{c}, \mathsf{a} \to \mathsf{d}\}$. The rewrite relation $\to_{\mathcal{R}}$ of this TRS is not confluent: for example, we can rewrite $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{b}$ using the first rule and $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{c}$ using the second rule, and neither b nor c can be rewritten any further.

We have the following critical pairs/overlays for $\mathcal{R}$:

$\mathsf{b} \bowtie \mathsf{c}$	from the first and second rule
$\mathsf{c} \bowtie \mathsf{b}$	from the first and second rule
$\mathsf{f}(\mathsf{d}) \rtimes \mathsf{b}$	from the first and third rule

The left-hand sides of the first and the second rule, $\mathsf{f}(\mathsf{a})$ and $\mathsf{f}(x)$, unify at the root position ε with the most general unifier $\sigma = \{x \mapsto \mathsf{a}\}$. Thus, these two rules have a critical pair, and since the unification was at root position, this critical pair is also a critical overlay. If we instantiate both rules using σ, we get $\mathsf{f}(\mathsf{a}) \to \mathsf{b}$ and $\mathsf{f}(\mathsf{a}) \to \mathsf{c}$. The right-hand sides of these instantiated rules are b and c, and they are the components of the critical overlays $\mathsf{b} \bowtie \mathsf{c}$ and $\mathsf{c} \bowtie \mathsf{b}$.

Note that every critical *overlay* comes together with its mirrored version: if two different rules $\ell_1 \to r_1$ and $\ell_2 \to r_2$ unify at the root position (and can thus both be used for rewriting a redex $\ell_1\sigma = \ell_2\sigma$ at the root), there is a symmetry and thus a choice which of the terms $r_1\sigma$ and $r_2\sigma$ to write on the left and which one on the right of the critical overlay. This is why the first and second rule together produce *two* critical overlays.

Now let us consider our first and our third rule. The left-hand side $\mathsf{f}(\mathsf{a})$ of the first rule has at its position 1 the non-variable subterm a that unifies with the left-hand side of the third rule, a, using the identity substitution as the most general unifier. Thus, we consider the instantiated left-hand side of the first rule, $\mathsf{f}(\mathsf{a})$, as the redex that may be rewritten either at position 1 with the third rule, to $\mathsf{f}(\mathsf{d})$, or at the root with the first rule, to b. This leads to the critical pair $\mathsf{f}(\mathsf{d}) \rtimes \mathsf{b}$.

Note that critical pairs that are not critical overlays do not have the symmetry mentioned earlier: the "inside" rewrite step at position $\pi > \varepsilon$ is always written on the left.

Definition 6.8. (Non-Overlapping)
A TRS $\mathcal{R}$ is *non-overlapping* iff $\mathcal{R}$ has no critical pairs.

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

Lemma 6.9. ([24], 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 $t_1 = t_2$.

In other words, for non-overlapping TRSs, rewriting a specific redex has a deterministic result. In a parallel-innermost rewrite step $s \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}}_{\mathcal{R}} t$, we rewrite all innermost redexes in s at the same time, so the choice of redexes to use is also deterministic. Together, this means that in a rewrite step $s \mathrel{\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}}_{\mathcal{R}} t$, the term t is uniquely determined for s, so the relation $\mathrel{\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}}_{\mathcal{R}}$ is deterministic as well.

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

Corollary 6.10. (Confluence of Parallel-Innermost Rewriting)
If a TRS $\mathcal{R}$ is non-overlapping, then $\mathrel{\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}}_{\mathcal{R}}$ is deterministic and hence confluent.

Remark 6.11. The reasoning behind Corollary 6.10 can be generalised to *arbitrary* parallel rewrite strategies where the redexes that are rewritten are fixed, such as (max-)parallel-*outermost* rewriting [34].

Remark 6.12. Note that in contrast to similar confluence criteria for full rewriting $\to_{\mathcal{R}}$ [46, 47], here $\mathcal{R}$ is *not* required to be left-linear (i.e., $\mathcal{R}$ may have rewrite rules where the left-hand side has more than one occurrence of the same variable).

Corollary 6.10 is similar to a result by Gramlich for (uniform) confluence of (sequential) innermost rewriting $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$ for non-overlapping TRSs that are not necessarily left-linear [50, Lemma 3.2.1, Corollary 3.2.2]. For parallel-innermost rewriting, we have the stronger property that $\mathrel{\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}}_{\mathcal{R}}$ is even deterministic: for innermost rewriting with $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$, there is still the non-deterministic choice between different innermost redexes, whereas the used redexes for a step with $\mathrel{\overset{\mathsf{i}}{+\!\!\!\!\rightarrow}}_{\mathcal{R}}$ are uniquely determined.

Example 6.13. The TRSs $\mathcal{R}$ from Example 3.1, Example 4.15, and Example 5.4 are all non-overlapping, and by Corollary 6.10 their parallel-innermost rewrite relations $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ are confluent. Thus, also the tight complexity bound $\mathrm{pirc}_{\mathcal{R}}(n) \in \Theta(n)$ in Example 5.9 is confirmed.

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 programs with deterministic small-step semantics, our motivation for this work!

Example 6.14. Corollary 6.10 already fails for such natural examples as a TRS with the following rules to compute the maximum function on natural numbers:

$$\begin{array}{rcl}
\mathsf{max}(\mathsf{Zero}, x) & \to & x \\
\mathsf{max}(x, \mathsf{Zero}) & \to & x \\
\mathsf{max}(\mathsf{S}(x), \mathsf{S}(y)) & \to & \mathsf{S}(\mathsf{max}(x, y))
\end{array}$$

Here we can arguably see immediately that the overlap between the first and the second rule, at root position, is harmless: if both rules are applicable to the same redex, the result of a rewrite step with either rule will be the same ($\mathsf{max}(\mathsf{Zero}, \mathsf{Zero}) \mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} \mathsf{Zero}$). Indeed, the resulting critical pair has the form $\mathsf{max}(\mathsf{Zero}, \mathsf{Zero}) \rtimes \mathsf{max}(\mathsf{Zero}, \mathsf{Zero})$, with both components of the critical pair the same.

6.2. Confluence of $\mathrel{\vert\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ with trivial innermost critical overlays

Critical pairs like $\mathsf{max}(\mathsf{Zero}, \mathsf{Zero}) \rtimes \mathsf{max}(\mathsf{Zero}, \mathsf{Zero})$ where both components are identical are also called *trivial*. If a critical pair is trivial, it means that the non-determinism in the choice of rules or positions in the redex does not lead to a non-determinism in the result of the two rewrite steps described abstractly by the critical pair. For the purposes of confluence, such critical pairs are always harmless. For example, for analysing confluence of Example 6.14, the critical pair $\mathsf{max}(\mathsf{Zero}, \mathsf{Zero}) \rtimes \mathsf{max}(\mathsf{Zero}, \mathsf{Zero})$ can be ignored.

For innermost rewriting, critical pairs resulting from an overlap between a redex with another redex as a subterm can be ignored as well: the outer redex is not enabled for an innermost rewrite step. For example, in Example 6.7, the critical pair $\mathsf{f}(\mathsf{d}) \rtimes \mathsf{f}(\mathsf{b})$ can be ignored. The reason is that for *innermost* rewriting, here everything is deterministic: only the rewrite step $\mathsf{f}(\mathsf{a}) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{f}(\mathsf{d})$ is innermost, but the rewrite step $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{b}$ is not. Thus, for confluence of innermost rewriting it suffices to consider critical *overlays*, which describe rewriting a redex at the *same* position of a term using two different rewrite rules, and we can ignore all other critical pairs.

Moreover, for innermost rewriting, we can even ignore those critical overlays that can result only from non-innermost rewrite steps, such as the two critical overlays $\mathsf{b} \bowtie \mathsf{c}$ and $\mathsf{c} \bowtie \mathsf{b}$ in Example 6.7: the term $\mathsf{f}(\mathsf{a})$ that causes these critical overlays in a critical peak cannot be rewritten *innermost* at the root because the subterm a must be rewritten first.

This considerations give rise to the following definitions (see, e.g., [50]):

Definition 6.15. (Trivial critical pair)
Critical pairs of the form $t \rtimes t$ are called *trivial*.

Definition 6.16. (Innermost critical overlay)
A critical overlay $s \bowtie t$ resulting from a critical peak $\ell\sigma = u\sigma \to_{\mathcal{R}} s$ and $\ell\sigma = u\sigma \to_{\mathcal{R}} t$ is called *innermost* for $\mathcal{R}$ iff we also have $\ell\sigma = u\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} s$ and $\ell\sigma = u\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$.

Gramlich combines the above observations on critical pairs in his PhD thesis [50] to the following sufficient criterion for (uniform) confluence of innermost rewriting:

Theorem 6.17. ([50], Theorem 3.5.6)
Let $\mathcal{R}$ be a TRS such that all innermost critical overlays of $\mathcal{R}$ are trivial. Then $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is uniformly confluent and hence confluent.

We can apply a similar reasoning to Theorem 6.17 to get a stronger criterion for $\mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ being deterministic.

Theorem 6.18. (Parallel-innermost confluence from only trivial innermost critical overlays)
Let $\mathcal{R}$ be a TRS such that all innermost critical overlays of $\mathcal{R}$ are trivial. Then $\mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is deterministic and hence confluent.

Theorem 6.18 subsumes Corollary 6.10: if there are no critical pairs at all, then there are also no non-trivial innermost critical overlays.

Proof:
We prove that the relation $\mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is deterministic if all innermost critical overlays of $\mathcal{R}$ are trivial. To this end, we will show that if $t_0 \mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_1$ and $t_0 \mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_2$, we have $t_1 = t_2$.

Assume $t_0 \mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_1$ and $t_0 \mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_2$ for some terms t_0, t_1, t_2. Since the rewrite step is parallel-innermost, all used redexes are fixed: all the innermost redexes. Let s_0 be an arbitrary innermost redex in t_0. If $s_0 \xrightarrow{\mathsf{i}}_{\mathcal{R}} s_1$ and $s_0 \xrightarrow{\mathsf{i}}_{\mathcal{R}} s_2$ implies $s_1 = s_2$, the statement $t_1 = t_2$ would directly follow.

Thus, assume $s_0 \xrightarrow{\mathsf{i}}_{\mathcal{R}} s_1$ and $s_0 \xrightarrow{\mathsf{i}}_{\mathcal{R}} s_2$. As these rewrite steps are innermost, rewriting must take place at the root of s_0. Assume that two different rules $\ell \to r, u \to v \in \mathcal{R}$ with variables renamed apart are used for these rewrite steps (otherwise the claim follows directly). Let δ, θ be substitutions such that $s_0 = u\delta \xrightarrow{\mathsf{i}}_{\mathcal{R}} v\delta = s_1$ and $s_0 = \ell\theta \xrightarrow{\mathsf{i}}_{\mathcal{R}} r\theta = s_2$.

There are corresponding critical peaks $u\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} v\sigma$ and $\ell\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} r\sigma$ with σ a most general unifier of u and ℓ. Since the rewrite steps are innermost, the critical peak gives rise to innermost critical overlays $v\sigma \bowtie r\sigma$ and $r\sigma \bowtie v\sigma$.

As σ is a most general unifier of u and ℓ, we have $s_0 = u\delta = u\sigma\delta'$ and $s_0 = \ell\theta = \ell\sigma\theta'$ with $\delta'(x) = \theta'(x)$ for all variables $x \in \mathcal{V}(u\delta) \cup \mathcal{V}(\ell\theta)$. Thus, $s_1 = v\delta = v\sigma\delta'$ and $s_2 = r\theta = r\sigma\theta' = r\sigma\delta'$. As the critical overlays are trivial by precondition of our theorem, we have $v\sigma = r\sigma$ and hence also $s_1 = v\sigma\delta' = r\sigma\delta' = s_2$. This concludes our proof. □

Example 6.19. Since the only innermost critical overlay $\mathsf{max}(\mathsf{Zero}, \mathsf{Zero}) \bowtie \mathsf{max}(\mathsf{Zero}, \mathsf{Zero})$ for the TRS $\mathcal{R}$ from Example 6.14 is trivial, $\mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is confluent.

Example 6.20. Since none of the critical overlays in the TRS $\mathcal{R}$ from Example 6.7 is innermost, $\mathrel{+\!\!\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}}$ is confluent.

Remark 6.21. From Theorem 6.18 one may be tempted to claim that $\overset{\mathsf{i}}{\Vert\!\!\rightarrow}_{\mathcal{R}}$ is deterministic iff $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is uniformly confluent. The "$\Rightarrow$" direction clearly holds. But for the "$\Leftarrow$" direction consider the following counterexample: $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{c}, \mathsf{b} \to \mathsf{d}, \mathsf{c} \to \mathsf{d}\}$. The relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is uniformly confluent, but $\overset{\mathsf{i}}{\Vert\!\!\rightarrow}_{\mathcal{R}}$ with $\mathsf{a} \overset{\mathsf{i}}{\Vert\!\!\rightarrow}_{\mathcal{R}} \mathsf{b}$ and $\mathsf{a} \overset{\mathsf{i}}{\Vert\!\!\rightarrow}_{\mathcal{R}} \mathsf{c}$ is not deterministic.

With Corollary 6.10 and Theorem 6.18 we have proposed two sufficient criteria for proving confluence of parallel-innermost rewriting for given TRSs that can be automated using syntactic checks only, without any search problems. These criteria specifically capture TRSs corresponding to deterministic programs.

7. Implementation and experiments

Implementation. We have implemented the contributions of this paper in the automated termination and complexity analysis tool AProVE [18]. We added or modified about 730 lines of Java code, including

- the framework of parallel-innermost rewriting;
- the generation of parallel DTs (Theorem 4.13);
- a processor to convert them to TRSs with the same complexity (Theorem 5.6, Theorem 5.8);
- the confluence tests of Corollary 6.10 and of Theorem 6.18.

As far as we are aware, this is the first implementation of a fully automated inference of complexity bounds for parallel-innermost rewriting. A preliminary implementation of our techniques in AProVE participated successfully in the new demonstration category "Runtime Complexity: TRS Parallel Innermost" at termCOMP 2022 and 2023.

This implementation is now part of the AProVE release versions and can be downloaded or used via a web interface [18]. The input format is an extension of the human-readable text format that was used to represent TRSs in early versions of the TPDB. For example, a file `size.trs` for Example 3.1 would have the content shown in Figure 2.

```
(GOAL COMPLEXITY)
(STARTTERM CONSTRUCTORBASED)
(STRATEGY PARALLELINNERMOST)
(VAR v l r x y)
(RULES
  size(Tree(v, l, r)) -> S(plus(size(l), size(r)))
  size(Nil) -> Zero
  plus(Zero, y) -> y
  plus(S(x), y) -> S(plus(x, y))
)
```

Figure 2. Input file for Example 3.1.

In this format, we can designate a rewrite rule as a *relative* rule, as in Example 5.2, by writing "`->=`" instead of "`->`".

Experiments. 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 [38].[6] 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 Theorem 4.16 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 [51] 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 [52].

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 [53, 19] from termCOMP 2021[7] 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 Theorem 4.13 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 Definition 5.3 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 either via Corollary 6.10 or via Theorem 6.18, as indicated where relevant). We also extracted each of the TRSs $PDT(\mathcal{R})/\mathcal{R}$ and used the files as inputs for the analysis of $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}$ by 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 $\mathrm{pirc}_{\mathcal{R}}$ obtained by analysing $\mathrm{irc}_{PDT(\mathcal{R})/\mathcal{R}}$ (for lower bounds, only where $\mathrel{+\!\!\!\!\xrightarrow{\mathrm{i}}}_{\mathcal{R}}$ had been proved confluent).

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

Tool	$\mathcal{O}(1)$	$\leq \mathcal{O}(n)$	$\leq \mathcal{O}(n^2)$	$\leq \mathcal{O}(n^3)$	$\leq \mathcal{O}(n^{\geq 4})$	avg. time (s)
TcT $\mathrm{irc}_{\mathcal{R}}$	4	32	51	62	67	202.9
AProVE $\mathrm{irc}_{\mathcal{R}}$	**5**	50	111	123	127	193.8
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 4	**5**	**70**	**125**	139	141	222.0
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 4 & 5	**5**	**70**	**125**	**140**	**142**	211.5
TcT $\mathrm{pirc}_{\mathcal{R}}$ Section 5	4	46	66	79	80	**189.7**
AProVE $\mathrm{pirc}_{\mathcal{R}}$ Section 5	**5**	64	99	108	108	219.8

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

[6]Version 11.3 of the TPDB was released in July 2022, but does not contain changes over version 11.2 for the category `Runtime_Complexity_Innermost_Rewriting`.

[7]For analysis of $\mathrm{irc}_{\mathcal{R}}$, both tools participated in termCOMP 2022 with their versions from termCOMP 2021.

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 70 TRSs rather than for 50 TRSs (using upper bounds on $\mathrm{irc}_{\mathcal{R}}$ as an over-approximation), an improvement by 40%. 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 Example 3.15 and Example 4.14 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 $\mathrm{pirc}_{\mathcal{R}}$.

Improvements 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. This TRS resembles a classical "syntactically vectorisable loop". In such cases, vectorisation indeed accelerates the program by this order of magnitude. Our analysis captures this kind of acceleration, however we should keep in mind that apart from vectorisation, even perfect "thread-based" parallelisation on N cores does not achieve N times acceleration, due to the cost of context switching. Our complexity result should be taken as an estimation of "parallelism potential".

We observe that adding the techniques from Section 5 to the techniques from Section 4 leads to only few examples for which better upper bounds can be found (one of them is Example 5.7).

Table 2. Lower 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 4 & 5, confluence by Corollary 6.10	165	116	**22**	**5**	**1**
APROVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 4 & 5, confluence by Theorem 6.18	**190**	133	**22**	**5**	**1**
TCT $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Corollary 6.10	165	112	0	0	0
TCT $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Theorem 6.18	**190**	131	0	0	0
APROVE $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Corollary 6.10	165	140	21	**5**	**1**
APROVE $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Theorem 6.18	**190**	**159**	21	**5**	**1**

Table 2 shows our results for lower bounds on $\mathrm{pirc}_{\mathcal{R}}$. Here we evaluated only configurations including Definition 5.3 to make inference techniques for lower bounds of $\mathrm{irc}_{\mathcal{R}}$ applicable to $\mathrm{pirc}_{\mathcal{R}}$.

Table 3. Tight 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 4 & 5, confluence by Corollary 6.10	**5**	27	0	**3**	35
APROVE $\mathrm{pirc}_{\mathcal{R}}$ Sections 4 & 5, confluence by Theorem 6.18	**5**	33	0	**3**	41
TCT $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Corollary 6.10	4	19	0	0	23
TCT $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Theorem 6.18	4	22	0	0	26
APROVE $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Corollary 6.10	**5**	33	0	**3**	41
APROVE $\mathrm{pirc}_{\mathcal{R}}$ Section 5, confluence by Theorem 6.18	**5**	**41**	0	**3**	**49**

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. As a precondition for applying our approach to lower bounds inference for $\mathrm{pirc}_{\mathcal{R}}$ for a TRS $\mathcal{R}$, we also need to find a proof for confluence of $\overset{\mathrm{i}}{+\!\!\!\!\!\!\rightarrow}_{\mathcal{R}}$. The confluence criterion of Corollary 6.10 is applicable to 165 TRSs in our benchmark set, about 56.1% of the benchmark set. Our new contribution Theorem 6.18 proves confluence of $\overset{\mathrm{i}}{+\!\!\!\!\!\!\rightarrow}_{\mathcal{R}}$ for a superset of 190 TRSs, about 64.6% of our benchmark set. This indicates that the search for more powerful criteria for proving confluence of parallel-innermost term rewriting is worthwhile.

Regarding lower bounds, we observe that non-trivial lower bounds can be inferred for 140 out of the 165 examples proved confluent via Corollary 6.10, and for 159 out of the 190 examples proved confluent via Theorem 6.18. This shows that our transformation from Section 5 has practical value since it produces relative TRSs that are generally amenable to analysis by existing program analysis tools. It also shows that the more powerful confluence analysis by Theorem 6.18 improves also the inference of lower complexity bounds. Finally, Table 3 shows that for overall 49 TRSs, the bounds that were found are asymptotically *precise*.[8]

However, the nature of the benchmark set also plays a significant role for assessing the applicability of criteria for proving confluence. Therefore, we also considered COPS [54], the benchmark collection

[8]Unfortunately, the implementation of our confluence check used in the experiments for the conference version [13] had a bug that caused it to consider some TRSs confluent where Corollary 6.10 was not applicable (specifically, overlaps due to certain non-trivial critical overlays were not detected as such). Overall, 21 TRSs were considered confluent even though Corollary 6.10 could not be applied, and 18 of the lower bounds claimed in the experimental evaluation of [13] were affected. This bug has now been fixed.

used in the *Confluence Competition* [48]. As a benchmark collection for our second experiment to assess our confluence analysis, we downloaded the 577 unsorted unconditional TRSs of COPS.[9] While the TRSs in this subset of COPS are usually analysed for confluence of full rewriting, we analysed whether the TRSs are confluent for parallel-innermost rewriting (which currently does not have a dedicated category in COPS). Our implementation determined that 60 of the 577 TRSs (about 10.4%) are non-overlapping, which implies parallel-innermost confluence by Corollary 6.10. In contrast, Theorem 6.18 finds 274 confluence proofs (about 47.5%), a significantly better result.

Still, the success rate for this benchmark set is significantly lower than for the examples from the TPDB. This is not surprising: COPS collects TRSs that provide a challenge to confluence analysis tools, whereas the analysed subset of the TPDB contains TRSs which are interesting specifically for runtime complexity analysis and often correspond to programs with deterministic results.

Runtime of the analysis. Table 1 shows the (mean) average time used by the respective tool configurations to analyse their inputs for (parallel-)innermost runtime complexity (the search for upper and lower bounds was run concurrently). With the used timeout of 300 seconds, all configurations needed between 180 and 240 seconds on average per example. It may perhaps be surprising that even the fastest configuration used over 180 seconds per example. The likely reason is that determining the asymptotic runtime complexity of a rewrite system is an optimisation problem: as long as the highest lower bound and the lowest upper bound found so far do not coincide, there is the possibility that applying further techniques in the analysis may lead to tighter bounds. Thus, complexity analysis tools like APROVE and TCT are usually configured to exhaust most of the available time by a search for upper and lower bounds, finishing their search only when (a) the current upper and lower bound coincide and can be reported as the best possible result, (b) all available techniques for the given configuration have been tried, or (c) the timeout is almost reached and the current result can be reported as the best result obtained within the given timeout. Indeed, in our experiments with a timeout of 300 seconds, both APROVE and TCT have runtimes between 290 and 300 seconds on many benchmarks.

Regarding the confluence-only analysis, on most TRSs in our collection both used criteria usually return a result within a few milliseconds. We believe that this very quick result is due to the fact that our criteria are purely syntactic and do not involve any search problems. For Corollary 6.10, the highest runtime on our benchmark suite was observed for `Frederiksen_Glenstrup/int` (183 rewrite rules) with 76 ms on a computer with an Intel Core i7-10750H CPU @ 2.60GHz. For Theorem 6.18, the highest runtime was observed for `Transformed_CSR_04/LISTUTILITIES_complete_noand_GM` (407 rewrite rules) with 135 ms on the same computer. Our implementation is not particularly optimised, so we anticipate that the criteria can be made to scale to larger examples as well.

8. 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 Section 5, and we discuss the apparent absence of work on confluence of parallel-innermost rewriting in Section 6. We now focus on automated techniques for complexity analysis of parallel/concurrent computation.

[9]The download took place on 29 May 2023.

Our notion of parallel complexity follows a large tradition of static *cost analysis*, notably for concurrent programming. The two notable works [55, 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 [56], and/or recurrence equations. Recent approaches for the Pi Calculus [2, 3] 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 [7].

For logic programs, which – like TRSs – express an implicit parallelism, parallel complexity can be inferred using recurrence solving [4].

The tool RAML [57] 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 [6]. 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 w.r.t. 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 [11, 12]. 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 [58] in runtime complexity analysis of *Logically Constrained TRSs (LCTRSs)* [20, 21], 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* [59] 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 our motivating example (2, Example 4.15), which can have longer derivations than basic terms of the same size. Finally, towards automated parallelisation we aim to infer complexity bounds w.r.t. term *height* (terms = trees!), as suggested in [10].

For *confluence analysis*, an obvious next step towards more powerful criteria would be to adapt the classic confluence criterion by Knuth and Bendix [45]. By this criterion, a TRS $\mathcal{R}$ has a confluent rewrite relation $\to_{\mathcal{R}}$ if it is terminating and for each of its critical pairs $t_1 \rtimes\!\!\!\ltimes t_2$, there exists some term s such that $t_1 \to_{\mathcal{R}}^* s$ and $t_2 \to_{\mathcal{R}}^* s$ (i.e., t_1 and t_2 are *joinable*). Termination can in many cases be proved automatically by modern termination analysis tools [39], and for terminating TRSs, it is decidable whether two terms are joinable (by rewriting them to all possible normal forms and checking whether a common normal form has been reached). This criterion has been adapted for confluence of innermost rewriting [50, Theorem 3.5.8] via critical overlays $t_1 \bowtie t_2$. A promising next step would be to investigate if further modifications are needed for proving confluence of parallel-innermost rewriting.

Towards handling larger and more difficult inputs, it would be worth investigating to what extent confluence criteria for (parallel-)innermost rewriting can be made *compositional* [60, 61] or integrated into the *Confluence Framework* [62]. This would allow combinations of different confluence criteria to work together by focusing on different parts of a TRS for the overall confluence proof.

In a different direction, formal *certification* of the proofs found using the techniques in this paper would be highly desirable. Unfortunately, automated tools for program verification such as APROVE are not immune to logical errors in their programming. For TRSs, many proof techniques for properties such as termination, complexity bounds, and confluence have been formalised in trusted proof assistants such as COQ or ISABELLE/HOL [63, 64, 65, 66]. Based on these formalisations, proof certifiers have been created to check proof traces (for a given property). Such proof traces are usually generated by automated tools specialised in *finding* proofs, such as APROVE, whose source code has not been formally verified. The certifier then either verifies that the proof trace indeed correctly instantiates the formalised proof techniques for the given TRS, or it points out that this was not the case (ideally with a pointer to the specific step in the proof trace that could not be verified).

Acknowledgements

We thank Vincent van Oostrom and the anonymous reviewers of earlier and the current versions of this paper for helpful discussions and comments.

References

[1] Baudon T, Fuhs C, Gonnord L. On Complexity Bounds and Confluence of Parallel Term Rewriting, 2024. URL `https://arxiv.org/abs/2305.18250`.

[2] 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, volume 12648 of *Lecture Notes in Computer Science*. Springer, 2021 pp. 59–86. URL `https://doi.org/10.1007/978-3-030-72019-3_3`.

[3] 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, volume 203 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021 pp. 34:1–34:22. URL `https://doi.org/10.4230/LIPIcs.CONCUR.2021.34`.

[4] Klemen M, López-García P, Gallagher JP, Morales JF, Hermenegildo MV. 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, volume 12042 of *Lecture Notes in Computer Science*. Springer, 2019 pp. 19–35. URL `https://doi.org/10.1007/978-3-030-45260-5_2`.

[5] Albert E, Correas J, Johnsen EB, Pun KI, Román-Díez G. Parallel Cost Analysis. *ACM Trans. Comput. Logic*, 2018. **19**(4):31:1–31:37. URL `https://doi.org/10.1145/3274278`.

[6] 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, volume 9032 of *Lecture Notes in Computer Science*. Springer, 2015 pp. 132–157. URL `https://doi.org/10.1007/978-3-662-46669-8_6`.

[7] Das A, Hoffmann J, Pfenning F. Parallel complexity analysis with temporal session types. *Proc. ACM Program. Lang.*, 2018. **2**(ICFP):91:1–91:30. URL `https://doi.org/10.1145/3236786`.

[8] Vuillemin J. Correct and Optimal Implementations of Recursion in a Simple Programming Language. *J. Comput. Syst. Sci.*, 1974. **9**(3):332–354. URL `https://doi.org/10.1016/S0022-0000(74)80048-6`.

[9] 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, volume 3467 of *Lecture Notes in Computer Science*. Springer, 2005 pp. 17–31. URL `https://doi.org/10.1007/978-3-540-32033-3_3`.

[10] Alias C, Fuhs C, Gonnord L. Estimation of Parallel Complexity with Rewriting Techniques. In: Proceedings of the 15th Workshop on Termination (WST 2016). 2016 pp. 2:1–2:5. URL `https://hal.archives-ouvertes.fr/hal-01345914`.

[11] van Eerd J, Groote JF, 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, volume 12818 of *Lecture Notes in Computer Science*. Springer, 2021 pp. 175–189. URL `https://doi.org/10.1007/978-3-030-89247-0_12`.

[12] van Eerd J, Groote JF, Hijma P, Martens J, Osama M, Wijs A. Innermost many-sorted term rewriting on GPUs. *Sci. Comput. Program.*, 2023. **225**:102910. URL `https://doi.org/10.1016/j.scico.2022.102910`.

[13] Baudon T, Fuhs C, Gonnord L. Analysing Parallel Complexity of Term Rewriting. In: Villanueva A (ed.), Logic-Based Program Synthesis and Transformation - 32nd International Symposium, LOPSTR 2022, Tbilisi, Georgia, September 21-23, 2022, Proceedings, volume 13474 of *Lecture Notes in Computer Science*. Springer, 2022 pp. 3–23. URL `https://doi.org/10.1007/978-3-031-16767-6_1`.

[14] Noschinski L, Emmes F, Giesl J. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *J. Autom. Reason.*, 2013. **51**(1):27–56. URL `https://doi.org/10.1007/s10817-013-9277-6`.

[15] Lankford DS. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Texas, 1975.

[16] Fuhs C, Giesl J, Middeldorp A, Schneider-Kamp P, Thiemann R, Zankl H. SAT Solving for Termination Analysis with Polynomial Interpretations. In: Marques-Silva J, Sakallah KA (eds.), Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings, volume 4501 of *Lecture Notes in Computer Science*. Springer, 2007 pp. 340–354. URL `https://doi.org/10.1007/978-3-540-72788-0_33`.

[17] Borralleras C, Lucas S, Oliveras A, Rodríguez-Carbonell E, Rubio A. SAT Modulo Linear Arithmetic for Solving Polynomial Constraints. *J. Autom. Reason.*, 2012. **48**(1):107–131. URL `https://doi.org/10.1007/s10817-010-9196-8`.

[18] 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.*, 2017. **58**(1):3–31. Web interface and download: `https://aprove.informatik.rwth-aachen.de/`, URL `https://doi.org/10.1007/s10817-016-9388-y`.

[19] 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, volume 9636 of *Lecture Notes in Computer Science*. Springer, 2016 pp. 407–423. Web interface and download: `https://www.uibk.ac.at/en/theoretical-computer-science/research/software/tct/`, URL `https://doi.org/10.1007/978-3-662-49674-9_24`.

[20] Kop C, Nishida N. Term Rewriting with Logical Constraints. In: Fontaine P, Ringeissen C, Schmidt RA (eds.), Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings, volume 8152 of *Lecture Notes in Computer Science*. Springer, 2013 pp. 343–358. URL `https://doi.org/10.1007/978-3-642-40885-4_24`.

[21] Fuhs C, Kop C, Nishida N. Verifying Procedural Programs via Constrained Rewriting Induction. *ACM Trans. Comput. Log.*, 2017. **18**(2):14:1–14:50. URL `https://doi.org/10.1145/3060143`.

[22] Kop C. Higher Order Termination. Ph.D. thesis, VU Amsterdam, 2012.

[23] Guo L, Kop C. Higher-Order LCTRSs and Their Termination. In: Weirich S (ed.), Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II, volume 14577 of *Lecture Notes in Computer Science*. Springer, 2024 pp. 331–357. URL `https://doi.org/10.1007/978-3-031-57267-8_13`.

[24] Baader F, Nipkow T. Term rewriting and all that. Cambridge Univ. Press, 1998. ISBN 978-0-521-45520-6.

[25] 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, volume 5195 of *Lecture Notes in Computer Science*. Springer, 2008 pp. 364–379. URL `https://doi.org/10.1007/978-3-540-71070-7_32`.

[26] 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, volume 8560 of *Lecture Notes in Computer Science*. Springer, 2014 pp. 257–271. URL `https://doi.org/10.1007/978-3-319-08918-8_18`.

[27] Avanzini M, Moser G. A combination framework for complexity. *Information and Computation*, 2016. **248**:22–55. URL `https://doi.org/10.1016/j.ic.2015.12.007`.

[28] 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, volume 10483 of *Lecture Notes in Computer Science*. Springer, 2017 pp. 132–150. URL `https://doi.org/10.1007/978-3-319-66167-4_8`.

[29] Frohn F, Giesl J, Hensel J, Aschermann C, Ströder T. Lower Bounds for Runtime Complexity of Term Rewriting. *J. Autom. Reason.*, 2017. **59**(1):121–163. URL `https://doi.org/10.1007/s10817-016-9397-x`.

[30] Moser G, Schneckenreither M. Automated amortised resource analysis for term rewrite systems. *Sci. Comput. Program.*, 2020. **185**. URL `https://doi.org/10.1016/j.scico.2019.102306`.

[31] Arts T, Giesl J. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 2000. **236**:133–178.

[32] Hong H, Jakus D. Testing Positiveness of Polynomials. *J. Autom. Reason.*, 1998. **21**(1):23–38. URL `https://doi.org/10.1023/A:1005983105493`.

[33] Blelloch GE, 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. ACM, 1995 pp. 226–237. URL `https://doi.org/10.1145/224164.224210`.

[34] Thiemann R, Sternagel C, Giesl J, Schneider-Kamp P. Loops under Strategies ... Continued. In: Kirchner H, Muñoz CA (eds.), Proceedings International Workshop on Strategies in Rewriting, Proving, and Programming, IWS 2010, Edinburgh, UK, 9th July 2010, volume 44 of *EPTCS*. 2010 pp. 51–65. URL `https://doi.org/10.4204/EPTCS.44.4`.

[35] van Oostrom V. Remarks on the full parallel innermost strategy, 2023. URL `http://www.javakade.nl/research/pdf/fpi.pdf`.

[36] 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, volume 5117 of *Lecture Notes in Computer Science*. Springer, 2008 pp. 110–125. URL `https://doi.org/10.1007/978-3-540-70590-1_8`.

[37] Lamport L. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 1978. **21**(7):558–565. URL `https://doi.org/10.1145/359545.359563`.

[38] Wiki. Termination Problems DataBase (TPDB). `http://termination-portal.org/wiki/TPDB`.

[39] 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, volume 11429 of *Lecture Notes in Computer Science*. Springer, 2019 pp. 156–166. URL `https://doi.org/10.1007/978-3-030-17502-3_10`.

[40] Wiki. The International Termination Competition (TermComp). `http://termination-portal.org/wiki/Termination_Competition`.

[41] Lucas S. Context-sensitive Rewriting. *ACM Comput. Surv.*, 2021. **53**(4):78:1–78:36. URL `https://doi.org/10.1145/3397677`.

[42] 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, volume 8858 of *Lecture Notes in Computer Science*. Springer, 2014 pp. 275–295. URL `https://doi.org/10.1007/978-3-319-12736-1_15`.

[43] Flores-Montoya A. Upper and Lower Amortized Cost Bounds of Programs Expressed as Cost Relations. In: Fitzgerald JS, Heitmeyer CL, Gnesi S, Philippou A (eds.), FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings, volume 9995 of *Lecture Notes in Computer Science*. 2016 pp. 254–273. URL `https://doi.org/10.1007/978-3-319-48989-6_16`.

[44] Avanzini M, Felgenhauer B. Type introduction for runtime complexity analysis. In: WST '14. 2014 pp. 1–5. Available from `http://www.easychair.org/smart-program/VSL2014/WST-proceedings.pdf`.

[45] Knuth DE, Bendix PB. Simple Word Problems in Universal Algebras. In: Leech J (ed.), Computational Problems in Abstract Algebra. Pergamon Press, 1970 pp. 263–297.

[46] Rosen BK. Tree-Manipulating Systems and Church-Rosser Theorems. *J. ACM*, 1973. **20**(1):160–187. URL `https://doi.org/10.1145/321738.321750`.

[47] Huet GP. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. *J. ACM*, 1980. **27**(4):797–821. URL `https://doi.org/10.1145/322217.322230`.

[48] Community. The International Confluence Competition (CoCo). `http://project-coco.uibk.ac.at/`.

[49] Baudon T, Fuhs C, Gonnord L. On Confluence of Parallel-Innermost Term Rewriting. In: Winkler S, Rocha C (eds.), Proceedings of the 11th International Workshop on Confluence. 2022 pp. 31–36. URL `http://cl-informatik.uibk.ac.at/iwc/2022/proceedings.pdf`.

[50] Gramlich B. Termination and confluence: properties of structured rewrite systems. Ph.D. thesis, Kaiserslautern University of Technology, Germany, 1996. URL `https://d-nb.info/949807389`.

[51] 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, volume 8562 of *Lecture Notes in Computer Science*. Springer, 2014 pp. 367–373. `https://www.starexec.org/`, URL `https://doi.org/10.1007/978-3-319-08587-6_28`.

[52] URL `https://www.dcs.bbk.ac.uk/~carsten/eval/parallel_complexity_journal/`.

[53] TCT, version from the Termination and Complexity Competitions 2020 – 2022. URL `https://www.starexec.org/starexec/secure/details/solver.jsp?id=29575`.

[54] Hirokawa N, Nagele J, Middeldorp A. Cops and CoCoWeb: Infrastructure for Confluence Tools. In: Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, volume 10900 of *LNCS*. Springer, 2018 pp. 346–353. See also: `https://cops.uibk.ac.at/`, URL `https://doi.org/10.1007/978-3-319-94205-6_23`.

[55] Albert E, Arenas P, Genaim S, Zanardini D. Task-level analysis for a language with async/finish parallelism. In: Vitek J, Sutter BD (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. ACM, 2011 pp. 21–30. URL `https://doi.org/10.1145/1967677.1967681`.

[56] Albert E, Arenas P, Genaim S, Puebla G, Zanardini D. Cost analysis of object-oriented bytecode programs. *Theor. Comput. Sci.*, 2012. **413**(1):142–159. URL `https://doi.org/10.1016/j.tcs.2011.07.009`.

[57] Hoffmann J, Aehlig K, Hofmann M. Resource Aware ML. In: Madhusudan P, Seshia SA (eds.), Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings, volume 7358 of *Lecture Notes in Computer Science*. Springer, 2012 pp. 781–786. URL `https://doi.org/10.1007/978-3-642-31424-7_64`.

[58] 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, volume 12561 of *Lecture Notes in Computer Science*. Springer, 2020 pp. 37–55. URL `https://doi.org/10.1007/978-3-030-68446-4_2`.

[59] 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, volume 355 of *Lecture Notes in Computer Science*. Springer, 1989 pp. 167–177. URL `https://doi.org/10.1007/3-540-51081-8_107`.

[60] Shintani K, Hirokawa N. Compositional Confluence Criteria. In: Felty AP (ed.), 7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel, volume 228 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022 pp. 28:1–28:19. URL `https://doi.org/10.4230/LIPIcs.FSCD.2022.28`.

[61] Shintani K, Hirokawa N. Compositional Confluence Criteria. *Log. Methods Comput. Sci.*, 2024. **20**(1).

[62] Gutiérrez R, Vítores M, Lucas S. Confluence Framework: Proving Confluence with CONFident. In: Villanueva A (ed.), Logic-Based Program Synthesis and Transformation - 32nd International Symposium, LOPSTR 2022, Tbilisi, Georgia, September 21-23, 2022, Proceedings, volume 13474 of *Lecture Notes in Computer Science*. Springer, 2022 pp. 24–43. URL `https://doi.org/10.1007/978-3-031-16767-6_2`.

[63] Contejean E, Courtieu P, Forest J, Pons O, Urbain X. Automated Certified Proofs with CiME3. In: Schmidt-Schauß M (ed.), Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia, volume 10 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011 pp. 21–30. URL `https://doi.org/10.4230/LIPIcs.RTA.2011.21`.

[64] Blanqui F, Koprowski A. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Math. Struct. Comput. Sci.*, 2011. **21**(4):827–859. URL `https://doi.org/10.1017/S0960129511000120`.

[65] Thiemann R, Sternagel C. Certification of Termination Proofs Using CeTA. In: Berghofer S, Nipkow T, Urban C, Wenzel M (eds.), Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings, volume 5674 of *Lecture Notes in Computer Science*. Springer, 2009 pp. 452–468. URL `https://doi.org/10.1007/978-3-642-03359-9_31`.

[66] van der Weide N, Vale D, Kop C. Certifying Higher-Order Polynomial Interpretations. In: Naumowicz A, Thiemann R (eds.), 14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland, volume 268 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023 pp. 30:1–30:20. URL `https://doi.org/10.4230/LIPIcs.ITP.2023.30`.