On Confluence of Parallel-Innermost Term Rewriting
Thaïs Baudon, Carsten Fuhs, Laure Gonnord

To cite this version:
Thaïs Baudon, Carsten Fuhs, Laure Gonnord. On Confluence of Parallel-Innermost Term Rewriting. IWC 2022 - 11th International Workshop on Confluence, Aug 2022, Haifa, Israel. hal-03710007v2

HAL Id: hal-03710007
https://inria.hal.science/hal-03710007v2
Submitted on 4 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
On Confluence of Parallel-Innermost Term Rewriting*

Thaïs Baudon¹, Carsten Fuhs², and Laure Gonnord³,¹

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

Abstract

We revisit parallel-innermost term rewriting as a model of parallel computation on inductive data structures. We propose a simple sufficient criterion for confluence of parallel-innermost rewriting based on non-overlappingness. Our experiments on a large benchmark set indicate the practical usefulness of our criterion. We close with a challenge to the community to develop more powerful dedicated techniques for this problem.

1 Introduction

This extended abstract deals with a practical approach to proving confluence of (max-)parallel-innermost term rewriting. We consider term rewrite systems (TRSs) as intermediate representation for programs operating on inductive data structures such as trees. More specifically, TRSs can be seen as an abstraction of pattern matching on algebraic data types (ADTs), which are particularly well-suited to the implementation of operations on inductive data structures. This class of programs is gaining in importance in high-performance computing (HPC): among other examples, the scheduler of the Linux kernel uses red-black trees; and many (also systems-level) programming languages like Rust used in HPC feature ADTs. This leads to the need for compilation techniques for pattern matching on ADTs that yield a highly efficient output. One aspect of this problem pertains to the parallelisation of such programs. A small example for such a program is given in Figure 1.

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

Figure 1: Tree size computation in Rust

In Section 2, we recapitulate standard definitions and fix notations. Section 3 recapitulates the notion of parallel-innermost rewriting on which we focus in this extended abstract. In Section 4, we provide a first criterion for confluence of parallel-innermost rewriting. Section 5 provides experimental evidence of the practicality of our criterion on a large standard benchmark set. We conclude in Section 6.

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

We assume familiarity with term rewriting (see, e.g., [2]) and recall standard definitions.

Definition 1 (Term Rewrite System, Innermost Rewriting). \( \mathcal{T}(\Sigma, \mathcal{V}) \) denotes the set of terms over a finite signature \( \Sigma \) and the set of variables \( \mathcal{V} \). For a term \( t \), the set \( \text{Pos}(t) \) of its positions is given as: (a) if \( t \in \mathcal{V} \), then \( \text{Pos}(t) = \{ \varepsilon \} \), and (b) if \( t = f(t_1, \ldots, t_n) \), then \( \text{Pos}(t) = \{ \varepsilon \} \cup \bigcup_{1 \leq i \leq n} \{ \pi \mid \pi \in \text{Pos}(t_i) \} \). The position \( \varepsilon \) is the root position of term \( t \). For \( \pi \in \text{Pos}(t) \), \( t|_{\pi} \) is the subterm of \( t \) at position \( \pi \), and we write \( t|_{\pi} \) for the term that results from \( t \) by replacing the subterm \( t|_{\pi} \) at position \( \pi \) by the term \( s \).

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

\( \Sigma^R_\mathcal{R} = \{ f \mid f(\ell_1, \ldots, \ell_n) \rightarrow r \in \mathcal{R} \} \) and \( \Sigma^c_\mathcal{R} = \Sigma \setminus \Sigma^R_\mathcal{R} \) are the defined and constructor symbols of \( \mathcal{R} \). We may also just write \( \Sigma_d^\mathcal{R} \) and \( \Sigma_c^\mathcal{R} \).

For a relation \( \rightarrow \), \( \rightarrow^+ \) is its transitive closure and \( \rightarrow^* \) its reflexive-transitive closure. An object \( o \) is a normal form wrt a relation \( \rightarrow \) iff there is no \( o' \) with \( o \rightarrow o' \). A relation \( \rightarrow \) is confluent iff \( s \rightarrow^+ t \) and \( s \rightarrow^+ u \) implies that there exists an object \( v \) with \( t \rightarrow^+ v \) and \( u \rightarrow^+ v \).

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

\[
\begin{align*}
\text{plus}(\text{Zero}, y) & \rightarrow y & \text{size}(\text{Nil}) & \rightarrow \text{Zero} \\
\text{plus}(S(x), y) & \rightarrow S(\text{plus}(x, y)) & \text{size}(\text{Tree}(v, l, r)) & \rightarrow S(\text{plus}(\text{size}(l), \text{size}(r)))
\end{align*}
\]

Here \( \Sigma^c_\mathcal{R} = \{ \text{plus}, \text{size} \} \) and \( \Sigma^R_\mathcal{R} = \{ \text{Zero}, \text{S}, \text{Nil}, \text{Tree} \} \). We have the following innermost rewrite sequence, where the used innermost redexes are underlined:

\[
\begin{align*}
\text{size}(\text{Tree}(\text{Zero}, \text{Nil}, \text{Tree}(\text{Zero}, \text{Nil}, \text{Nil}))) & \xrightarrow{\mathcal{R}} S(\text{plus}(\text{size}(\text{Nil}), \text{size}(\text{Tree}(\text{Zero}, \text{Nil}, \text{Nil})))) \\
\xrightarrow{\mathcal{R}} S(\text{plus}(\text{Zero}, \text{size}(\text{Tree}(\text{Zero}, \text{Nil}, \text{Nil})))) & \xrightarrow{\mathcal{R}} S(\text{plus}(\text{Zero}, S(\text{plus}(\text{size}(\text{Nil}), \text{size}(\text{Nil})))))) \\
\xrightarrow{\mathcal{R}} S(\text{plus}(\text{Zero}, S(\text{plus}(\text{Zero}, \text{size}(\text{Nil})))))) & \xrightarrow{\mathcal{R}} S(\text{plus}(\text{Zero}, S(\text{plus}(\text{Zero}, \text{Zero})))) \\
\xrightarrow{\mathcal{R}} S(\text{plus}(\text{Zero}, S(\text{Zero}))) & \xrightarrow{\mathcal{R}} S(\text{Zero})
\end{align*}
\]

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

3 Parallel-Innermost Rewriting

The notion of parallel-innermost rewriting dates back at least to the year 1974 [13]. 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 [3]. In the literature [11], this strategy is also known as “max-parallel-innermost rewriting”.

Definition 2 (Parallel-Innermost Rewriting [5]). A term \( s \) rewrites innermost in parallel to \( t \) with a TRS \( \mathcal{R} \), written \( s \xrightarrow{R} t \), iff \( s \xrightarrow{\ell} t \), and either (a) \( s \xrightarrow{\mathcal{R}} t \) 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 \( s_k \xrightarrow{R} t_k \) or \( s_k = t_k \) is a normal form.
Example 2 (Example 1 continued). The TRS $\mathcal{R}$ from Example 1 allows the following parallel-innermost rewrite sequence, where innermost redexes are underlined:

$$
\begin{align*}
\text{size(Tree(Zero, Nil, Tree(Zero, Nil, Nil))))} & \quad \xrightarrow{\mathcal{R}} \quad S(\text{size}(\text{Nil}), \text{size(Tree(Zero, Nil, Nil))))) \\
\text{S(plus(Zero, S(plus(size(Nil), size(Nil))))))} & \quad \xrightarrow{\mathcal{R}} \quad \text{S(plus(Zero, S(plus(Zero, Zero))))} \\
\text{S(plus(Zero, S(Zero)))} & \quad \xrightarrow{\mathcal{R}} \quad \text{S(S(Zero))}
\end{align*}
$$

In the second and in the third step, two innermost steps each happen in parallel. An innermost rewrite sequence without parallel evaluation necessarily needs two more steps to a normal form from this start term, as in Example 1.

4 Confluence of Parallel-Innermost Rewriting

Given a TRS $\mathcal{R}$, we wish to prove (or disprove) confluence of this relation $\parallel\rightarrow_{\mathcal{R}}$. Apart from intrinsic interest in confluence as an important property of a rewrite relation, we are also motivated by applications of confluence proofs to finding lower bounds for the length of the longest derivation with $\parallel\rightarrow_{\mathcal{R}}$ from basic terms, i.e., terms $f(t_1, \ldots, t_k)$ where $f$ is a defined symbol and all $t_i$ are constructor terms. This notion of complexity of a TRS $\mathcal{R}$, which is parametric in the size of the start term, is also known as runtime complexity $[8]$.\(^1\)

To this end, might we even reuse confluence of innermost rewriting or of full rewriting (and corresponding tools) as sufficient criteria for confluence of parallel-innermost rewriting?

Alas, by the following example, in general we have to answer this question in the negative.

Example 3 (Confluence of $\rightarrow_{\mathcal{R}}$ does not imply Confluence of $\parallel\rightarrow_{\mathcal{R}}$). To see that we cannot prove confluence of $\parallel\rightarrow_{\mathcal{R}}$ just by using a standard off-the-shelf tool for confluence analysis of innermost or full rewriting $[4]$, consider the TRS $\mathcal{R} = \{a \rightarrow f(b, b), a \rightarrow f(b, c), b \rightarrow c, c \rightarrow b\}$.

For this TRS, both $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{R}}$ are confluent. However, $\parallel\rightarrow_{\mathcal{R}}$ is not confluent: we can rewrite both $a \; \parallel\rightarrow_{\mathcal{R}} f(b, b)$ and $a \; \parallel\rightarrow_{\mathcal{R}} f(b, c)$, yet there is no term $v$ such that $f(b, b) \; \parallel\rightarrow_{\mathcal{R}} v$ and $f(b, c) \; \parallel\rightarrow_{\mathcal{R}} v$. The reason is that the only possible rewrite sequences with $\parallel\rightarrow_{\mathcal{R}}$ from these terms are $f(b, b) \; \parallel\rightarrow_{\mathcal{R}} f(c, c) \; \parallel\rightarrow_{\mathcal{R}} f(b, b) \; \parallel\rightarrow_{\mathcal{R}} \ldots$ and $f(b, c) \; \parallel\rightarrow_{\mathcal{R}} f(c, b) \; \parallel\rightarrow_{\mathcal{R}} f(b, c) \; \parallel\rightarrow_{\mathcal{R}} \ldots$, with no terms in common.

Thus, in general a confluence proof for $\rightarrow_{\mathcal{R}}$ or $\rightarrow_{\mathcal{R}}$ does not imply confluence for $\parallel\rightarrow_{\mathcal{R}}$.

To devise a sufficient criterion for confluence of $\parallel\rightarrow_{\mathcal{R}}$, recall that confluence means: if a term $s$ can be rewritten to two different terms $t_1$ and $t_2$ in 0 or more steps, then it is always possible to rewrite $t_1$ and $t_2$ in 0 or more steps to one and the same term $u$. For parallel-innermost rewriting, the redexes that get rewritten are fixed: all the innermost redexes simultaneously. Thus, $s$ can reach two different terms $t_1$ and $t_2$ only if at least one of these redexes can be rewritten in two different ways using $\rightarrow_{\mathcal{R}}$.

The following standard definition of a non-overlapping TRS will be very helpful for a sufficient criterion of confluence of $\parallel\rightarrow_{\mathcal{R}}$.

Definition 3 (Non-Overlapping). A TRS $\mathcal{R}$ is non-overlapping iff for any two rules $\ell \rightarrow r, u \rightarrow v \in \mathcal{R}$ where variables have been renamed apart between the rules, there is no position $\pi$ in $\ell$ such that $\ell|_{\pi} \notin \mathcal{V}$ and the terms $\ell|_{\pi}$ and $u$ unify.

\(^1\)The details of our approach to finding complexity bounds are outside of the scope of the present extended abstract; what matters here is that it provides an application for techniques to prove confluence of parallel-innermost rewriting. Thus, more powerful techniques for proving confluence of parallel-innermost rewriting potentially allow for larger applicability of techniques for finding lower bounds for runtime complexity of parallel-innermost rewriting.
Using non-overlappingness, a sufficient criterion that a given redex has a unique result from a rewrite step is given in the following.

**Lemma 1** ([2], Lemma 6.3.9). If a TRS $R$ is non-overlapping, and both $s \rightarrow_R t_1$ and $s \rightarrow_R t_2$ with the used redex of both rewrite steps at the same position, then $t_1 = t_2$.

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

**Corollary 1** (Confluence of Parallel-Innermost Rewriting). If a TRS $R$ is non-overlapping, then $\parallel\rightarrow_R$ is confluent.

Here left-linearity of $R$ (i.e., in all rules $\ell \rightarrow r \in R$, every variable occurs at most once in $\ell$), as in Rosen’s criterion for confluence of full rewriting [10], is not required.

**Example 4** (Example 2 continued). Our TRS $R$ from Example 1 and Example 2 is non-overlapping and, by Corollary 1, its relation $\parallel\rightarrow_R$ is confluent.

The reasoning behind Corollary 1 can be generalised to arbitrary strategies where the redexes that are rewritten are fixed, such as (max-)parallel-outermost rewriting [11].

We get the following two follow-up questions:

1. How powerful is Corollary 1 for proving confluence of $\parallel\rightarrow_R$ in practice?
2. Can we really not do better than Corollary 1?

5 Experiments

To assess the first question, we used the implementation of the non-overlappingness check in the automated termination and complexity analysis tool AProVE [6]. To demonstrate the effectiveness of our implementation, we have considered the 663 TRSs from category Runtime_Complexity_InnermostRewriting of the Termination Problem Database (TPDB), version 11.2 [15]. The TPDB is a collection of examples used at the annual Termination and Complexity Competition [7, 14]. The above category of the TPDB is the benchmark collection used specifically to compare tools that infer complexity bounds for runtime complexity of innermost rewriting. As both the TPDB and also COPS [9], the benchmark collection used in the Confluence Competition [4], currently do not have a specific benchmark collection for parallel-innermost rewriting, we used this benchmark collection instead.2

In our experiments, AProVE determined for 447 out of 663 TRSs (about 67.4%) that they are non-overlapping. By Corollary 1, this means that their parallel-innermost rewrite relations are confluent. Thus, already with the simple (and efficiently checkable) criterion of Corollary 1 we cover a large number of TRSs occurring “in the wild”.

At the same time, this reinforces the second question: Can we not do better than this? Corollary 1 already fails for such natural examples as a TRS with the following rules to compute the maximum function on natural numbers:

\[
\begin{align*}
\max(0, x) & \rightarrow x \\
\max(x, 0) & \rightarrow x \\
\max(S(x), S(y)) & \rightarrow S(\max(x, y))
\end{align*}
\]

2Our experimental data as well as all examples are available online [1].
Here one 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 ($\mathsf{max}(\text{Zero}, \text{Zero}) \xrightarrow{R} \text{Zero}$). However, in general, more powerful criteria for confluence of parallel-innermost rewriting would be desirable.

6 Conclusion

We are not aware of other work that explicitly discusses automatically checkable criteria for confluence of parallel-innermost rewriting. As such, this extended abstract tries to make a first attempt at filling this gap, by using non-overlappingness as a sufficient criterion. Our experiments indicate that non-overlappingness provides a good “baseline” for a sufficient criterion for confluence of parallel-innermost rewriting. At the same time, techniques based on checks for non-overlappingness are one of the most basic tools in a confluence prover’s toolbox. Thus, this paper also poses the challenge to the community to develop stronger techniques for proving (and disproving!) confluence of parallel-innermost rewriting.

References