



**HAL**  
open science

# 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 Baudon<sup>1</sup>, Carsten Fuhs<sup>2</sup>, and Laure Gonnord<sup>3,1</sup>

<sup>1</sup> LIP (UMR CNRS/ENS Lyon/UCB Lyon1/INRIA), Lyon, France

<sup>2</sup> Birkbeck, University of London, United Kingdom

<sup>3</sup> 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](#).

Here, the recursive calls to `left.size()` and `right.size()` can be done in parallel. In the following, we shall consider a corresponding parallel-innermost rewrite relation. Evaluation of TRSs (as a simple functional programming language) with innermost rewrite strategies in massively parallel

```
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

settings such as GPUs is currently a topic of active research [12]. Confluence of parallel-innermost rewriting enters the picture in several ways: for TRSs, confluence determines whether the specific choice of rules makes a difference; moreover, confluence can be a prerequisite for applicability of program analysis techniques (e.g., for finding complexity bounds).

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  $t = f(t_1, \dots, t_n)$ , then  $\text{Pos}(t) = \{\varepsilon\} \cup \bigcup_{1 \leq i \leq n} \{i\pi \mid \pi \in \text{Pos}(t_i)\}$ . 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  $t[s]_\pi$  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  $f(t_1, \dots, t_n)$ ,  $\text{root}(t) = f$  is the root symbol of  $t$ . A term rewrite system (TRS)  $\mathcal{R}$  is a set of rules  $\{\ell_1 \rightarrow r_1, \dots, \ell_n \rightarrow r_n\}$  with  $\ell_i, r_i \in \mathcal{T}(\Sigma, \mathcal{V})$ ,  $\ell_i \notin \mathcal{V}$ , and  $\mathcal{V}(r_i) \subseteq \mathcal{V}(\ell_i)$  for all  $1 \leq i \leq n$ . The rewrite relation of  $\mathcal{R}$  is  $s \rightarrow_{\mathcal{R}} t$  iff there are a rule  $\ell \rightarrow r \in \mathcal{R}$ , a position  $\pi \in \text{Pos}(s)$ , and a substitution  $\sigma$  such that  $s = s[\ell\sigma]_\pi$  and  $t = s[r\sigma]_\pi$ . 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  $s \xrightarrow{i}_{\mathcal{R}} t$ .

$\Sigma_d^{\mathcal{R}} = \{f \mid f(\ell_1, \dots, \ell_n) \rightarrow r \in \mathcal{R}\}$  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  $\Sigma_d$  and  $\Sigma_c$ .

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.

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

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

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

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  $s \Downarrow_{\mathcal{R}}^i t$ , iff  $s \xrightarrow{i}_{\mathcal{R}}^+ t$ , and either (a)  $s \xrightarrow{i}_{\mathcal{R}} t$  with  $s$  an innermost redex, or (b)  $s = f(s_1, \dots, s_n)$ ,  $t = f(t_1, \dots, t_n)$ , and for all  $1 \leq k \leq n$  either  $s_k \Downarrow_{\mathcal{R}}^i t_k$  or  $s_k = t_k$  is a normal form.

74 **Example 2** (Example 1 continued). The TRS  $\mathcal{R}$  from Example 1 allows the following parallel-  
 75 innermost rewrite sequence, where innermost redexes are underlined:

$$\begin{array}{c} \text{size(Tree(Zero, Nil, Tree(Zero, Nil, Nil)))} \parallel^i_{\rightarrow \mathcal{R}} \text{S(plus(size(Nil), size(Tree(Zero, Nil, Nil))))} \\ \parallel^i_{\rightarrow \mathcal{R}} \text{S(plus(Zero, S(plus(\underline{\text{size(Nil)}}, \underline{\text{size(Nil)}))))} \parallel^i_{\rightarrow \mathcal{R}} \text{S(plus(Zero, S(plus(Zero, Zero))))} \\ \parallel^i_{\rightarrow \mathcal{R}} \text{S(plus(Zero, S(Zero)))} \parallel^i_{\rightarrow \mathcal{R}} \text{S(S(Zero))} \end{array}$$

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

## 79 4 Confluence of Parallel-Innermost Rewriting

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

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

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

89 **Example 3** (Confluence of  $\overset{i}{\rightarrow}_{\mathcal{R}}$  does not imply Confluence of  $\parallel^i_{\rightarrow \mathcal{R}}$ ). To see that we cannot  
 90 prove confluence of  $\parallel^i_{\rightarrow \mathcal{R}}$  just by using a standard off-the-shelf tool for confluence analysis of  
 91 innermost or full rewriting [4], consider the TRS  $\mathcal{R} = \{a \rightarrow f(b, b), a \rightarrow f(b, c), b \rightarrow c, c \rightarrow b\}$ .  
 92 For this TRS, both  $\overset{i}{\rightarrow}_{\mathcal{R}}$  and  $\rightarrow_{\mathcal{R}}$  are confluent. However,  $\parallel^i_{\rightarrow \mathcal{R}}$  is not confluent: we can rewrite  
 93 both  $a \parallel^i_{\rightarrow \mathcal{R}} f(b, b)$  and  $a \parallel^i_{\rightarrow \mathcal{R}} f(b, c)$ , yet there is no term  $v$  such that  $f(b, b) \parallel^i_{\rightarrow \mathcal{R}}^* v$  and  
 94  $f(b, c) \parallel^i_{\rightarrow \mathcal{R}}^* v$ . The reason is that the only possible rewrite sequences with  $\parallel^i_{\rightarrow \mathcal{R}}$  from these terms  
 95 are  $f(b, b) \parallel^i_{\rightarrow \mathcal{R}} f(c, c) \parallel^i_{\rightarrow \mathcal{R}} f(b, b) \parallel^i_{\rightarrow \mathcal{R}} \dots$  and  $f(b, c) \parallel^i_{\rightarrow \mathcal{R}} f(c, b) \parallel^i_{\rightarrow \mathcal{R}} f(b, c) \parallel^i_{\rightarrow \mathcal{R}} \dots$ ,  
 96 with no terms in common.

97 Thus, in general a confluence proof for  $\rightarrow_{\mathcal{R}}$  or  $\overset{i}{\rightarrow}_{\mathcal{R}}$  does not imply confluence for  $\parallel^i_{\rightarrow \mathcal{R}}$ .

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

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

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

<sup>1</sup>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.

109 Using non-overlappingness, a sufficient criterion that a given redex has a unique result from  
 110 a rewrite step is given in the following.

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

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

115 **Corollary 1** (Confluence of Parallel-Innermost Rewriting). *If a TRS  $\mathcal{R}$  is non-overlapping,  
 116 then  $\Downarrow^i_{\mathcal{R}}$  is confluent.*

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

119 **Example 4** (Example 2 continued). *Our TRS  $\mathcal{R}$  from Example 1 and Example 2 is non-  
 120 overlapping and, by Corollary 1, its relation  $\Downarrow^i_{\mathcal{R}}$  is confluent.*

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

123 We get the following two follow-up questions:

- 124 1. How powerful is Corollary 1 for proving confluence of  $\Downarrow^i_{\mathcal{R}}$  in practice?
- 125 2. Can we really not do better than Corollary 1?

## 126 5 Experiments

127 To assess the first question, we used the implementation of the non-overlappingness check  
 128 in the automated termination and complexity analysis tool APROVE [6]. To demonstrate  
 129 the effectiveness of our implementation, we have considered the 663 TRSs from category  
 130 `Runtime.Complexity.Innermost.Rewriting` of the Termination Problem Database (TPDB),  
 131 version 11.2 [15]. The TPDB is a collection of examples used at the annual *Termination and  
 132 Complexity Competition* [7, 14]. The above category of the TPDB is the benchmark collection  
 133 used specifically to compare tools that infer complexity bounds for runtime complexity of  
 134 *innermost rewriting*. As both the TPDB and also COPS [9], the benchmark collection used  
 135 in the Confluence Competition [4], currently do not have a specific benchmark collection for  
 136 parallel-innermost rewriting, we used this benchmark collection instead.<sup>2</sup>

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

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

$$\begin{aligned} \max(\text{Zero}, x) &\rightarrow x \\ \max(x, \text{Zero}) &\rightarrow x \\ \max(\text{S}(x), \text{S}(y)) &\rightarrow \text{S}(\max(x, y)) \end{aligned}$$

---

<sup>2</sup>Our experimental data as well as all examples are available online [1].

144 Here one can arguably see immediately that the overlap between the first and the second rule, at  
145 root position, is harmless: if both rules are applicable to the same redex, the result of a rewrite  
146 step with either rule will be the same ( $\max(\text{Zero}, \text{Zero}) \xrightarrow{i}_{\mathcal{R}} \text{Zero}$ ). However, in general, more  
147 powerful criteria for confluence of parallel-innermost rewriting would be desirable.

## 148 6 Conclusion

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

## 157 References

- 158 [1] [https://www.dcs.bbk.ac.uk/~carsten/eval/parallel\\_confluence/](https://www.dcs.bbk.ac.uk/~carsten/eval/parallel_confluence/).
- 159 [2] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge Univ. Press, 1998.
- 160 [3] Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In *Proc. FPCA*  
161 *1995*, pages 226–237. ACM, 1995.
- 162 [4] Community. Confluence Competition (CoCo). <http://project-coco.uibk.ac.at/>.
- 163 [5] Mirtha-Lina Fernández, Guillem Godoy, and Albert Rubio. Orderings for innermost termination.  
164 In *Proc. RTA '05*, volume 3467 of *LNCS*, pages 17–31. Springer, 2005.
- 165 [6] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten  
166 Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie  
167 Swiderski, and René Thiemann. Analyzing program termination and complexity automatically  
168 with AProVE. *J. Autom. Reason.*, 58(1):3–31, 2017.
- 169 [7] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The  
170 termination and complexity competition. In *Proc. TACAS '19, Part III*, volume 11429 of *LNCS*,  
171 pages 156–166. Springer, 2019.
- 172 [8] Nao Hirokawa and Georg Moser. Automated complexity analysis based on the dependency pair  
173 method. In *Proc. IJCAR '08*, volume 5195 of *LNCS*, pages 364–379. Springer, 2008.
- 174 [9] Nao Hirokawa, Julian Nagele, and Aart Middeldorp. Cops and CoCoWeb: Infrastructure for  
175 confluence tools. In *Proc. IJCAR '18*, volume 10900 of *LNCS*, pages 346–353. Springer, 2018. See  
176 also: <https://cops.uibk.ac.at/>.
- 177 [10] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. ACM*, 20(1):160–187,  
178 1973.
- 179 [11] René Thiemann, Christian Sternagel, Jürgen Giesl, and Peter Schneider-Kamp. Loops under  
180 strategies ... continued. In *Proc. IWS '10*, volume 44 of *EPTCS*, pages 51–65, 2010.
- 181 [12] Johri van Eerd, Jan Friso Groote, Pieter Hijma, Jan Martens, and Anton Wijs. Term rewriting on  
182 GPUs. In *Proc. FSEN '21*, volume 12818 of *LNCS*, pages 175–189. Springer, 2021.
- 183 [13] Jean Vuillemin. Correct and optimal implementations of recursion in a simple programming  
184 language. *J. Comput. Syst. Sci.*, 9(3):332–354, 1974.
- 185 [14] Wiki. The International Termination Competition (TermComp). [http://termination-portal.org/wiki/Termination\\_Competition](http://termination-portal.org/wiki/Termination_Competition).
- 186 [15] Wiki. Termination Problems DataBase (TPDB). <http://termination-portal.org/wiki/TPDB>.
- 187