# On Confluence of Parallel-Innermost Term Rewriting

Thaïs Baudon, Carsten Fuhs, Laure Gonnord

# On Confluence of Parallel-Innermost Term Rewriting*

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

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

**Abstract**

We revisit parallel-innermost term rewriting as a model of parallel computation on inductive data structures. 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

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 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 $\mathcal{P}os(t)$ of its* positions *is given as: (a) if $t \in \mathcal{V}$, then $\mathcal{P}os(t) = \{\varepsilon\}$, and (b) if $t = f(t_1, \dots, t_n)$, 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 $\varepsilon$ is the* root position *of term $t$. For $\pi \in \mathcal{P}os(t)$, $t|_\pi$ 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 $f(t_1, \dots, t_n)$,* root$(t) = f$ *is the* root symbol *of $t$. A* term rewrite system (TRS) *$\mathcal{R}$ is a set of rules $\{\ell_1 \to r_1, \dots, \ell_n \to r_n\}$ 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 $\sigma$ 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 $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$.*

*$\Sigma^{\mathcal{R}}_d = \{f \mid f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}\}$ and $\Sigma^{\mathcal{R}}_c = \Sigma \setminus \Sigma^{\mathcal{R}}_d$ are the* defined *and* constructor *symbols of $\mathcal{R}$. We may also just write $\Sigma_d$ and $\Sigma_c$.*

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

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

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

*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 $s \mathrel{\Vert\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t$, 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) $s = f(s_1, \dots, s_n)$, $t = f(t_1, \dots, t_n)$, and for all $1 \le k \le n$ either $s_k \mathrel{\Vert\!\!\xrightarrow{\mathsf{i}}}_{\mathcal{R}} t_k$ 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:*

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

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

Thus, in general a confluence proof for $\to_{\mathcal{R}}$ or $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$ does not imply confluence for $\overset{\mathsf{i}}{\parallel\!\!\!\to}_{\mathcal{R}}$.

To devise a sufficient criterion for confluence of $\overset{\mathsf{i}}{\parallel\!\!\!\to}_{\mathcal{R}}$, 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 $\overset{\mathsf{i}}{\to}_{\mathcal{R}}$.

The following standard definition of a non-overlapping TRS will be very helpful for a sufficient criterion of confluence of $\overset{\mathsf{i}}{\parallel\!\!\!\to}_{\mathcal{R}}$.

**Definition 3** (Non-Overlapping)**.** *A TRS $\mathcal{R}$ is* non-overlapping *iff for any two rules $\ell \to r, u \to v \in \mathcal{R}$ where variables have been renamed apart between the rules, there is no position $\pi$ 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 $\mathcal{R}$ is non-overlapping, and both $s \to_{\mathcal{R}} t_1$ and $s \to_{\mathcal{R}} t_2$ 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 $\mathcal{R}$ is non-overlapping, then $\mathrel{+\!\!\!\Vert}\!\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is confluent.*

Here left-linearity of $\mathcal{R}$ (i.e., in all rules $\ell \to r \in \mathcal{R}$, 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 $\mathcal{R}$ from Example 1 and Example 2 is non-overlapping and, by Corollary 1, its relation $\mathrel{+\!\!\!\Vert}\!\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ 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 $\mathrel{+\!\!\!\Vert}\!\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ 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_Innermost_Rewriting` 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:

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

[2] Our 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 ($\mathsf{max}(\mathsf{Zero}, \mathsf{Zero}) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{Zero}$). 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


[1] https://www.dcs.bbk.ac.uk/~carsten/eval/parallel_confluence/.

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

[3] Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In *Proc. FPCA 1995*, pages 226–237. ACM, 1995.

[4] Community. Confluence Competition (CoCo). http://project-coco.uibk.ac.at/.

[5] Mirtha-Lina Fernández, Guillem Godoy, and Albert Rubio. Orderings for innermost termination. In *Proc. RTA '05*, volume 3467 of *LNCS*, pages 17–31. Springer, 2005.

[6] Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *J. Autom. Reason.*, 58(1):3–31, 2017.

[7] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In *Proc. TACAS '19, Part III*, volume 11429 of *LNCS*, pages 156–166. Springer, 2019.

[8] Nao Hirokawa and Georg Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR '08*, volume 5195 of *LNCS*, pages 364–379. Springer, 2008.

[9] Nao Hirokawa, Julian Nagele, and Aart Middeldorp. Cops and CoCoWeb: Infrastructure for confluence tools. In *Proc. IJCAR '18*, volume 10900 of *LNCS*, pages 346–353. Springer, 2018. See also: https://cops.uibk.ac.at/.

[10] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. ACM*, 20(1):160–187, 1973.

[11] René Thiemann, Christian Sternagel, Jürgen Giesl, and Peter Schneider-Kamp. Loops under strategies ... continued. In *Proc. IWS '10*, volume 44 of *EPTCS*, pages 51–65, 2010.

[12] Johri van Eerd, Jan Friso Groote, Pieter Hijma, Jan Martens, and Anton Wijs. Term rewriting on GPUs. In *Proc. FSEN '21*, volume 12818 of *LNCS*, pages 175–189. Springer, 2021.

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

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

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